Robot hand self-perception via 2D segmentation: a deep learning approach

Alexandre Almeida
alexandre.de.almeida@tecnico.ulisboa.pt

Instituto Superior Técnico, Lisboa, Portugal

December 2019

Abstract

We propose using deep learning to perform segmentation on images containing robotic hands. This serves many applications, *e.g.* helping the robot with tasks requiring accurate knowledge about the positioning of the robot's hands. Detecting its own hands, through vision, is more reliable than using kinematics because, in the latter, each joint of the arm must be properly calibrated, otherwise small offsets will propagate and create large errors when estimating the hand's pose. Maintaining all the joints properly calibrated is impractical and time consuming. To overcome the challenge of collecting and annotating large amounts of images, the datasets are generated in simulation, using domain randomization. We use a model, pre-trained on a large scale dataset, and fine-tune it on our datasets. We make experiments, with different training datasets and strategies, showing results on different types of datasets, with images generated in simulation and real indoor environments, with the real robot. From these experiments we create a final model, trained solely on synthetic images, that achieves an average IoU of 82% on synthetic validation data and $63,5\%$ on real validation data. These results were achieved with only 1000 training images and 3 hours of training time on a single GPU. We do not intend to create a robust model, but rather develop a methodology requiring low amounts of data to achieve reasonable performance, on real data, and give detailed insight on how to properly generate variability in the data and how to fine-tune a complex model to a very different task.
Keywords: Robotic Hands, Segmentation, Deep Learning, Domain Randomization, Indoor environments

1. Introduction

Vision is an important sensing capability that serves many purposes in modern robots. For a robot to be aware of the position of its own body and physical boundaries (self-perception), through vision, it must learn to process and interpret the images it collects from cameras. Contrary to methods without visual feedback (*e.g.* direct kinematics), vision based methods (*e.g.* keypoint estimation and segmentation) are not prone to calibration errors. If the position of an initial joint (*e.g.* the shoulder joint) is miscalculated, the error will propagate through all the rest of the joints (until reaching the hand), while with vision feedback this does not happen, because the position of the hand is not determined based on the position of the other joints.

For the purpose of this work, pose/keypoint estimation methods [24, 1] are not enough since they only provide the position of some keypoints (joints of the hand) and not the whole hand. While this is enough for some tasks, *e.g.* hand gestures recognition, it might not be enough for other tasks that require full knowledge about the spatial layout of the hand, *e.g.* object grasping.

CNNs (Convolutional Neural Networks) recently started to solve many problems in computer vision, due to their high efficiency in extracting features from images. Said features can be used to classify objects and make bounding box predictions for each object. One of the latest works in this area, Mask RCNN [6], achieved great accuracy at these tasks, while also retrieving a binary mask for each object. The advantages of using this type of networks is that no feature engineering is required (feature extraction is part of the deep learning process). Recent improvements in parallel programming with GPUs (e.g. CUDA [1]) make deep learning algorithms a good choice for many computer vision problems. Moreover, there are weights available, pre-trained on large datasets (e.g. COCO [14], ImageNet [21]) that can be used to initialize training, leading to improvements in generalization, when comparing to methods that initialize weights using

[1] CUDA webpage https://developer.nvidia.com/cuda-zone

normal distributions, for example.

Considering all this, our goal with this work is to propose a deep learning method to segment the hand(s) of a robot from the background. To achieve this task, we have to overcome a few major challanges, namely: (i) how to collect large amounts of training data without any pre-existent datasets, and (ii) how to fit a model, pre-trained on COCO, to our specific domain (indoor cluttered domain, with robotic hands and large amount of background objects). In order to solve challenge (i), we propose to use and explore domain randomization [23], a method that relies on training a model with exclusively simulated data and retain most of the performance, in real data. With this method, we generate the training samples, without the need to collect and annotate images manually. To overcome challenge (ii), we fine-tune the pre-trained weights and the *hyperparameters*.

2. Background

Our work builds upon recent successes in deep learning methods that predict the class of each pixel in an image [20, 12, 6, 17]. All these methods are end-to-end trainable, meaning that they require no pre or post-processing of data and feature extraction is a part of the learning process, and are built upon the FCN [22] architecture. The downside of these characteristics is that these networks require large amounts of training data and pre-existent datasets are not always available or easy to create.

2.1. Fully Convolutional Network

A FCN (Fully Convolutional Network) [22] is a state-of-the-art CNN that, to our knowledge, outperforms current methods in pixel-wise classification tasks (segmentation), without the need for any pre or post-processing (e.g. superpixels, proposals, like in [3, 5]). This method extends previous classification networks to the task of semantic segmentation by up-sampling final layers to match the input image size, using deconvolutional layers [2]. These layers do not need to be fixed to perform bilinear interpolation, instead they can learn how to up-sample optimally.

2.2. Semantic segmentation

The task of semantic segmentation consists in assigning a label for each class present in the input image. Different instances of the same class get assigned with the same label.

U-Net [12] is an architecture built to perform this type of task. It takes an image as input, down-samples it to lower dimensions through convolution and max pooling operations in order to convert the original image into feature vectors and is followed by up-sampling paths that apply transpose convolutions [2], in order to recover and map the results back to the dimensions of the original image. This is similar to the original architecture of a FCN [22], but with the important modification that in the up-sampling part, a large number of feature channels is used, which allows the network to propagate context information to higher resolution layers. This type of network was designed, having in mind biomedical applications (*e.g.* segmentation of moving cells), making it an ideal choice to segment images with low background clutter and clear boundaries between different objects (no overlapping), but it is not enough for our application due to the lack of pre-trained models in data distributions similar to our target domain.

RefineNet [20] is a multi-path network, *i.e.* it exploits features at multiple resolutions and not only the resolution of the input image, in order to achive high-resolution semantic segmentation. Also, this network uses ResNet [7] models, pre-trained on ImageNet [21], as the fundamental building block to extract features for further classification. Although this architecture seems appropriate for our application, it does not solve the more complex problem of instance segmentation.

2.3. Instance segmentation

Instance segmentation differs from semantic segmentation in the sense that labels are also instance-aware, meaning that different instances of the same class get assigned with different labels. This type of problem requires not only making predictions for each pixel but also separate each instance present in the image and classify them separately.

Recent deep learning approaches to this problem [18, 17, 6] rely on making candidate object proposals and then classify each proposal separately. In the methods proposed in [18, 17], segmentation precedes recognition, which is slower and less accurate, according to more recent studies in [6]. Instead, the Mask RCNN, proposed in [6], is divided into two stages. The first stage is called the RPN (Region Proposal Network) where it follows a method, first proposed in [4], to generate candidate object proposals by generating anchors of different sizes and ratios, centered in every pixel of the feature map and classify each anchor as background or foreground. Each anchor classified as foreground is accepted for the second stage, which consists in assigning each anchor its final class, making a bounding box regression that predicts a delta that is applied to the anchor (for the bounding box prediction) and retrieving a the binary mask of the object present in the anchor. These three tasks are done in parallel, in order to increase speed.

To our knowledge, Mask RCNN [6] is, overall, the

most successful state-of-the-art deep learning architecture, for tasks of both object recognition and instance segmentation, showing clear improvements of previous methods, both in speed and in accuracy. Also, Mask RCNN models pre-trained on COCO [14] are available online for download, making this the architecture upon which we build our work.

3. Related work

The problem of segmenting a hand, through vision, has been mainly focused on human hands, and not the robot's hands or arms. The main differences between these two problems are: (i) many human hand segmentation methods rely on skin-color information [9, 8], which perform poorly on cluttered domains with different lighting effects, and (ii) there is a wide variety of human hand datasets available and already annotated [10] whilst, to our knowledge, there is no pre-existent annotated dataset of humanoid robotic hands, in first person view.

In [11], J. Leitner *et al.* propose a method to detect the hands of two different humanoid robotic models. The authors also propose two approaches for detection: (i) detect the finger tips of the robots' hands, and (ii) detect the full hand. The second task is more desirable, since it provides more information, but it is described as a more complicated task, requiring ten times more training time, to reach the best solution, than the first approach. However promising, this work does not present numerical results due to the lack of suitable datasets, making it hard to compare with our work.

4. Methodology

In this work, we propose to extract the bounding box and a binary mask of the instances of humanoid robotic hands, present in an input image. For the neural network architecture, we use Mask RCNN [6] with the ResNet-101-FPN [7, 13] backbone, *i.e.* the feature extraction process.

First, to disambiguate some important terms, we define *hyperparameters* as any parameter related to the training process, that can be tuned manually without having to make any change in the network's architecture and are not changed during training (*e.g.* learning rate, loss functions, number of train/val epochs). Also, we define *model parameters* or *weights* as the parameters that are adjusted during training through backpropagation, to decrease the training loss (*e.g.* the kernels/filters weights). Finally, a *model* is the name given to an instance of the network trained with a given train and validation dataset and with the selected set of *hyperparameters*.

Our methodology can be divided into three essential processes. The data generation is where we describe the methods used to generate training and simulated validation images and the corresponding groundtruth data, *i.e.* the bounding boxes and masks, to feed the network during training. The learning process can be seen as the strategies that can be used to train a *model* pre-trained on another dataset, to perform a different classification task, with significant changes in the domain. We also give an overview of the architecture used for this process and explain some of the choices made in this part. The last process is the evaluation, where we evaluate the loss and accuracy of trained *models*, using appropriate metrics and real validation datasets that will be defined in Section 5. For a better overview of this methodology, we provide a pipeline, in Figure 1.

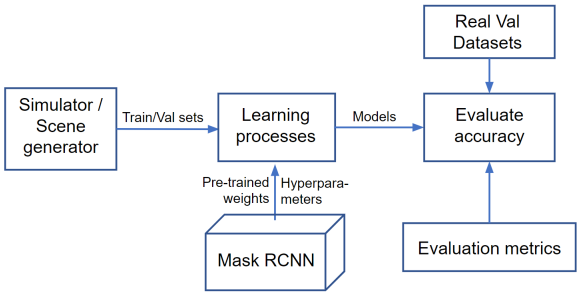

Figure 1: Overview of the methodology pipeline, using the architecture of Mask R-CNN [6].

4.1. Data generation

To our knowledge, there is no pre-existent dataset of humanoid robotic hands with annotated groundtruth masks, available for research. Considering the large amount of time required to collect and annotate enough training and validation images, we propose to use a different approach. We build our own custom dataset in simulation, using a 3D model of a humanoid robotic hand and domain randomization [23] to generate a dataset with enough variability to overcome the *reality gap*, *i.e.* to achieve reasonable performance when validating a *model*, trained only with simulated images, on real images.

In the simulation domain, we can randomize properties of the scene related to the background, foreground, the camera and lighting effects. In this work, we will focus largely on randomizing background and foreground properties because these show greater improvements for the overall accuracy, in the real domain.

For the background, we consider two components, distractor objects and background wall. For the distractor objects, we generate four types of 3D parametric objects (ellipsoids, parallelepipeds, elliptic cylinders and spherocylinders) with randomized shapes, sizes and colors (solid colors only). In respect to the background wall, we generate three types of walls, solid random colors, Perlin noise

[16] and real backgrounds.

In the foreground component, we randomize the properties of the 3D hand model. This includes the position and orientation of the arm (which contains the hand and wrist), relative to the world frame, and the orientation of the hand relative to the arm.

The most important camera properties are its position and orientation, but since we already randomize the position and orientation of all the objects in the scene, we do not randomize the camera properties. As for the lighting effects, we generate 3 lights and randomize the position and orientation of each light.

The target domain (real domain) of our work may contain high clutter, *i.e.* high amount of different colors, overlapping objects, and different lighting effects, around the objects of interests (the hands). To help overcome this complexity we also use real backgrounds, in our training and validation simulated datasets. To do this, we search for a number of images belonging to the super-category *indoor* of the COCO [14] dataset and overlap a randomized instance of the 3D hand model, since the Mask RCNN architecture is not designed to train images without, at least, one positive instance. Instead, every training image has (implicitly) negative samples that correspond to regions where the hand is not present. In figure 2, we provide with 3 examples of images with real backgrounds used for the training and validation simulated sets.

Figure 2: 3 examples of images using real backgrounds, from the COCO dataset.

To generate groundtruth data, we automatically extract a binary mask of the image (where 1 corresponds to hand pixels and 0 to background pixels), from the simulator. The bounding box coordinates can be extracted from the mask, so we will only explain how to extract the mask. In the simulation domain we use three cameras (with the same position and orientation), one to render the 3D hand model, other to render the background and the third to render the whole scene. By extracting the depth images of each camera, we know which pixels (in the image of the entire scene) belong to the hand and which belong to the background and we assign the value 1 or 0, accordingly.

4.2. Learning

During training, the goal is to minimize the error of the predictions in the train set, while also keeping a high generalization capacity, *i.e.* the capacity of retaining good results when validated with data that was not used to adjust the *model parameters* (*e.g.* the validation set). More importantly, the generalization capacity must also be extended when the domain changes substantially, *e.g.* when we validate with real images. According to J. Yosinski *et al.* [25], that can be achieved by using the knowledge of previously trained models in large scale datasets and by adopting a good training strategy, *i.e.* choosing the correct layers to freeze and those to fine-tune, during the learning process.

In theory, initial layers extract features that represent more general aspects of the dataset and, as such, the weights tend to not change much in these layers, as in deeper layers, even when the dataset and the classification task are both changed. As such, initializing the weights to an already trained (and accurate) values will almost always give good convergence.

We initialize the network with *weights* pre-trained in the COCO [14] dataset, except for the classification layers that predict the classes, bounding boxes and masks, because we change the number of output classes to 2. The training and validation datasets used are generated in simulation, following the process explained in Section 4.1. We sample, approximately, a total of 1000 images from the simulator, in which 80% are used for training and the remaining 20% are used for validation. The training loss is defined as the sum of the five losses (with equal weights) and, for each learning process, we also monitor the sum of the five losses on the validation set. When the total validation loss does not decrease for 15 epochs, we stop training, to avoid overfitting, and choose the model with the lowest validation loss. We also set a maximum of 150 training epochs, in case the loss does not stop decreasing.

In the Mask R-CNN [6] architecture, the total loss is divided into five losses. Two are to account for the loss when proposing a region of interest that contains an object, in the RPN [19] stage. The *rpn_class_loss* is for the predicted binary class, object/not object, and the *rpn_bbox_loss* is for the predicted bounding box that contains the object. The three remaining losses account for the predictions of the final class of the object (*mrcnn_class_loss*), hand or background, the final predicted offsets for the bounding box (*mrcnn_bbox_loss*) and the predicted binary mask (*mrcnn_mask_loss*).

5. Experimental setup

In this section, we will give some details on the platforms used to implement our processes, as well as defining the metrics used to evaluate the trained models and the real validation datasets in which we make these evaluations.

5.1. Implementation details

To implement the data generation process, we use Unity [2], a game development platform. The reason we chose this software is due to its high speed at rendering textures, allowing it to generate images at a higher rate than other 3D simulation platforms. To render the hand in the images, we import a 3D model of Vizzy [15], a humanoid robot designed for assistive robotics. Some of the physical restrictions of the robot are also implemented in Unity, like the ranges of angular movements in some joints (forearm pronation, wrist abduction and wrist flexion, all as specified in [15]). This is done to avoid unfeasible hand poses that are not coherent with the ranges of the real domain, which is our target domain.

As for the parts of Vizzy used in simulation, we discovered that only using the hand (without a visible arm attached) leads to poor results in segmentation in the real domain (where there is a visible arm attached). So, to improve the results, we also attached the 3D model of the arm to the hand, in Unity. Since the rest of the body is not (usually) visible in the real domain, we do not use it in simulation.

For the learning process, we use an implementation of Mask RCNN [3], that follows the architecture of the original work [6]. This implementation is open-source and it is the most widely used for Mask RCNN, from what we found so far.

We have a public github repository [4] with all the code used and created to implement our methods and experiments.

5.2. Real validation datasets

To evaluate the accuracy of the models, in the real domain, we collected and annotated images of the real robot's hands, in three different types of backgrounds: (i) low cluttered backgound, a simple solid background with different hand positions and orientations, (ii) medium cluttered background, where we place some distractor objects on a solid colored table and we change the poses of the camera, the distractor objects and the hand, and (iii) high cluttered background, where we change the pose of the camera and the hand and the lighting effects, in an indoor office-like environment with high number of different objects and patterns. For a better understanding of the environments, we provide two images of each dataset, in figure 3

[2] Unity webpage https://unity.com/

[3] mask rcnn github https://github.com/matterport/Mask_RCNN

[4] Public github repository, with our project https://github.com/alexalm4190/Mask_RCNN-Vizzy_Hand

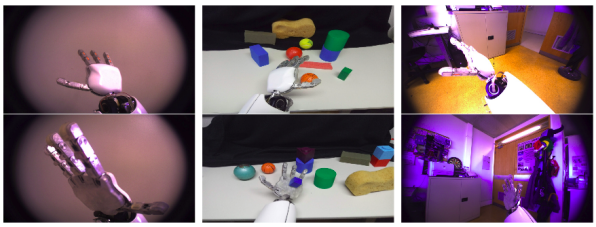

Figure 3: Low cluttered, medium cluttered and high cluttered backgrounds from left to right, respectively.

5.3. Evaluation metrics

The evaluation metric used in the original work of Mask RCNN is the mAP. This metric is widely used for multi-classification tasks, where the goal is to detect as much positive instances as possible, while maintaining a reasonable accuracy in the masks and bounding box predictions. In the context of our work, this metric cannot be used, since the priorities are switched. We typically only have 1 or 2 positive instances in each image and we want to have the highest accuracy possible when predicting the mask and the bounding box for each instance.

We use three main metrics, IoU, precision and recall. Each metric is calculated for each image and averaged across all the images, in the validation datasets. Given a predicted mask of a positive instance and its corresponding groundtruth mask, all these metrics can be calculated as follows:

$$IoU = \frac{TP}{TP + FP + FN} \tag{1}$$

$$Precision = \frac{TP}{TP + FP} \tag{2}$$

$$Recall = \frac{TP}{TP + FN}, \tag{3}$$

where TP, FP and FN are the true positives, false positives and false negatives as shown in figure 4

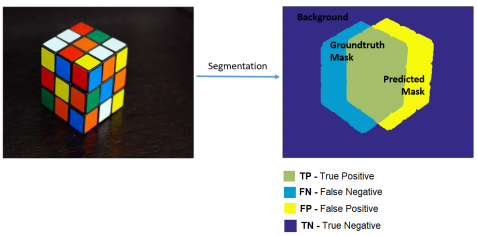

Figure 4: Visualization of the true positives, false positives and false negatives, given a predicted mask and its groundtruth mask.

6. Experiments and results

In this section, a series of experiments are conducted to produce both numerical and visual results, in order to test our hypotheses and compare different approaches. These experiments also aim

to explore, in an empirical way, the effects of different training strategies and domain randomization parameters.

6.1. Domain Randomization results

In this section we show the results of training the model with different datasets generated in Unity. We train the *model* on six different datasets, to measure the impact that some of the randomized aspects of the scene have on the accuracy of the *model* when validated with real images.

With these six datasets we evaluate the impact of the following major aspects: (i) attaching the arm to the hand in the 3D model of the robot, against only randomizing the hand position and orientation without the arm appearing in the images, (ii) Using Perlin noise to generate a more randomized and cluttered background, instead of only randomizing solid colored backgrounds, (iii) The use of distractor objects to generate partial occlusions of the hand, (iv) generation of different lighting effects, and (v) the use of real images as background. In Figure 5, we show images from each of the six datasets, generated in Unity. Note that each different feature (except for the arm attachment) is added to the others and does not replace previous features, *e.g.* in the *Perlin noise* dataset we have images with Perlin noise backgrounds and images with solid colered backgrounds. Each dataset has, approximately, 1000 images.

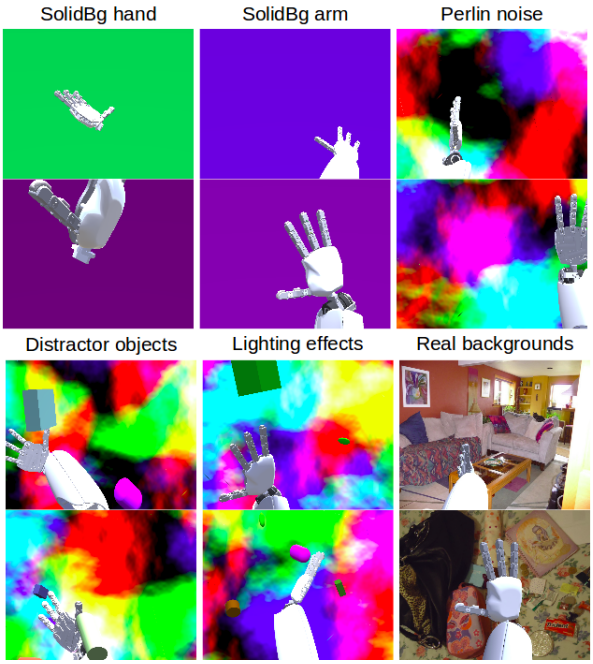

Figure 5: Datasets generated in Unity, using domain randomization.

In Figure 6 we present the results when validating the *models* on each real validation dataset, trained in each of the six datasets, generated in Unity.

The hand and arm of the robot are very similar in color and texture, when the back of the hand is turned to the camera. By training Mask RCNN with a 3D model of the hand only (dark blue dataset, in Figure 6) we can achieve a very high recall, but the precision is very low due to parts of the arm visible in the image being detected as part of the hand. To avoid these false positives, the 3D model of the arm must also be attached to the hand, in simulation, leading to a substantial improvement in *precision*, with the dataset *SolidBg arm* (orange dataset, in Figure 6), on all real validation datasets. However, this improvement comes with a small cost in *recall*, making it harder for the model to detect the entirety of the hand. In general, this change is very positive because the average *IoU* increases in all real validation datasets.

The Perlin noise effect gives the *model* a capability in separating cluttered backgrounds from the hand. In Figure 6, we can see this effect improves the average *IoU* in all real validation datasets, especially in the most cluttered ones. This capability can also be extended by two important features. First, the addition of distractor objects provides the training set with images where the hand is occluded by other objects. This also affects mostly the medium and high cluttered datasets which have images with different objects (in size, shape and color) around and in front of the hand. Secondly, by adding images with real backgrounds, in the train set, we achieve the greatest performance in *IoU* for the highest cluttered dataset.

When changing the lighting effects, we do not have significant differences in performance. We did not, however, explored many strategies in this regard. Future works could further explore this feature by introducing different types of lights with, for example, spotlight effects and place a different number of lights in different areas of the scene.

6.2. Transfer learning results

In order to conduct this experiment, we divide the network into 3 stages, (i) the backbone (ResNet-101-FPN [7, 13]) is the part with most layers, with the purpose of learning to extract the most relevant features from the input images, at different resolution levels, (ii) the proposal stage (RPN [19]) is the part that learns to propose bounding boxes containing objects of interest, and (iii) the classification stage is the part that learns to predict a class label, a bounding box refinement and a binary mask for each object predicted in the previous stage.

For training and validation, we use the dataset with real backgrounds (see Section 6.1) generated in Unity. For validation on real data, we merge our 3 real validation datasets into a single set and apply

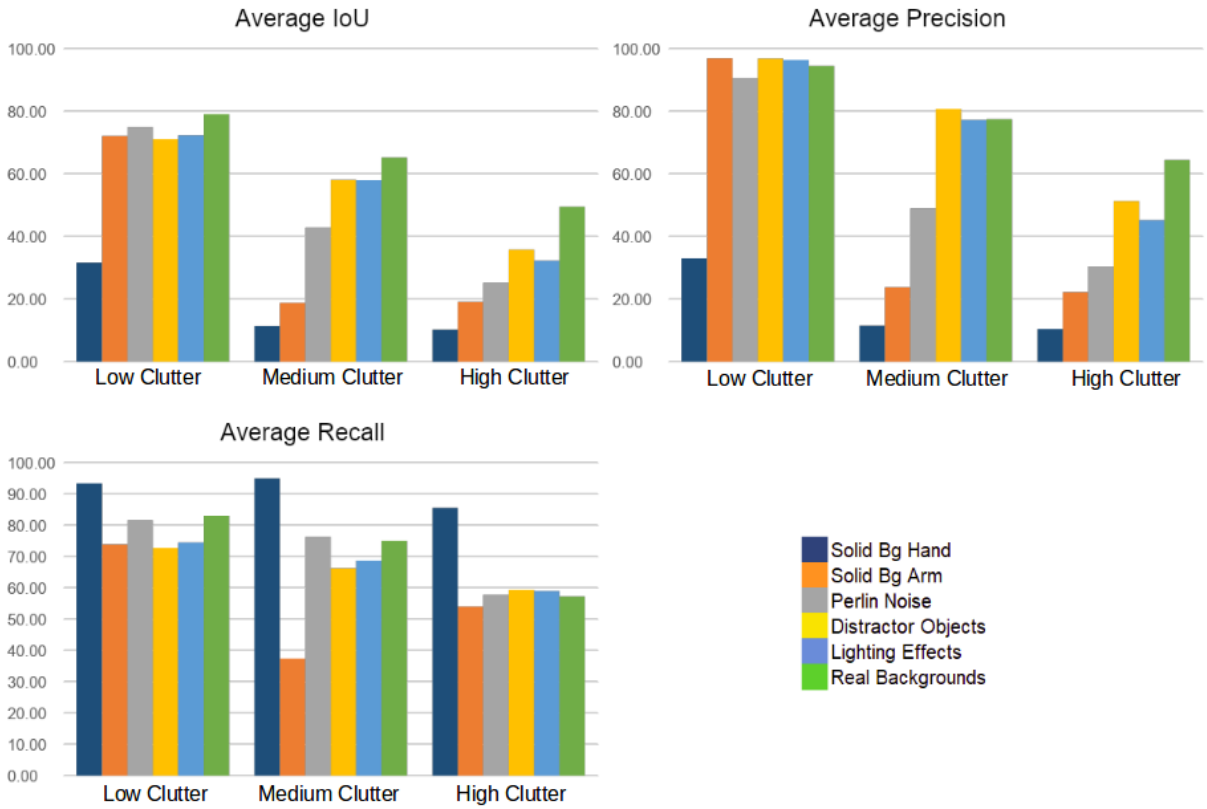

Figure 6: Results when validating models in different real validation datasets, trained with different simulated datasets.

the evaluation metrics to it.

We train 4 *models*, each initialized in a different way. The goal is to evaluate different combinations of initializations, by transferring *weights*, pretrained on COCO [14], to the given stage(s) and train the remaining stage(s) from scratch. The results of inferring the real and simulated validation datasets on these 4 *models* are shown in Table 1.

Table 1: Results of inferring 4 *models*, each trained from different initialization points, on the real and simulated validation datasets, using the evaluation metrics.

Transferred weights	Simulation			Real		
	Rec	*Pre*	*IoU*	*Rec*	*Pre*	*IoU*
Backbone.	85,8	88,4	77,5	64,3	71,8	54,4
Backbone and proposal stage.	**90,0**	93,5	**84,8**	62,5	**86,9**	58,2
Backbone and classification stage.	89,3	**93,8**	84,5	55,2	68,1	50,8
All stages.	87,2	92,8	82,0	**72,2**	79,5	**63,4**

Furthermore, we divide the backbone into 4 blocks of layers, where the last layer of each block is the feed-forward connection to a different level of the FPN [13] which, after a series of convolutional layers, results in 4 different feature map levels, to be used for region proposals and classification.

We train 5 different *models*, each with a different number of backbone stages (or blocks) frozen (during training). After this process, we apply the evaluation metrics (just the IoU, because now we are only interested in measuring the accuracy) to each *model*, when inferring the real and simulated validation sets. The results of this experiment can be seen in Figure 7.

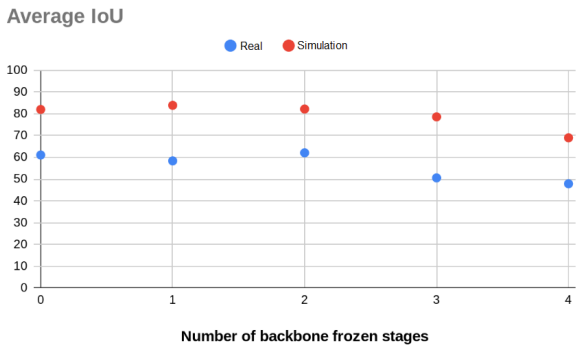

Figure 7: Results (average IoU) of inferring the real and simulated validation sets on 5 different *models*, each trained with a number of backbone stages (or blocks) frozen.

6.3. Instance segmentation results

In order to test our final model's capacity to generalize to data that was neither used in training nor in validation, we collected and annotated 21 more images, with the real robot, in an environment similar to the one in which the high clutter dataset was collected. We inferred these images on our final model, yielding an average IoU of $56,3\%$. When comparing to the results of inferring the high clutter dataset in this model (which yielded an average IoU of $55,4\%$), we can conclude that our model

has the capacity of generalizing to unseen data, that was not used in the process of adjusting the methodology. These results can be visualized in Figure 8.

Figure 8: Visualization of the predicted masks, bounding boxes and class scores, for 6 images of the real test dataset.

We show, in Figure 9, results of inferring images where both the real robot's hands are visible.

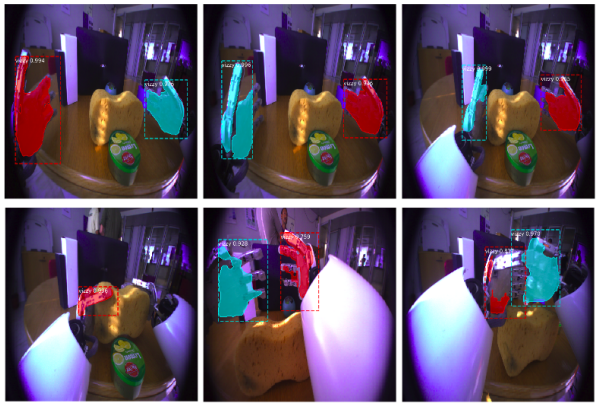

Figure 9: Visualization of instance segmentation results.

The *model* used to obtain these results was trained on a training set where each image only contains 1 positive instance of the robot's hand (the 3D model of the right hand). We show that, even when trained with exactly 1 positive instance, in each training image, the *model* is able to detect more than 1 positive instance (left and right hands of the real robot) when validating with real images, without any data augmentation. The top images of Figure 9 even show reasonable results for both hands, but when their boundaries start to get close (as we can see in the bottom images) their masks start to misclassify too many pixels (however, the bounding boxes are still correctly predicted, in most cases).

7. Conclusions

With this work, it was intended to give a real robot the ability to perceive all the points belonging to its own hand, in an image. This is done by fine-tuning a pre-trained CNN to extract a binary masks of the robot's hand, from images. To do this, we had to face two main challenges: (i) collect and annotate enough images, to train a deep CNN, and (ii) overcome the differences in the domains and tasks between the pre-trained model and our target model.

To overcome the challenges inherent to this work, we developed a process for generating training and validation images, using Unity and domain randomization concepts. Also, we applied transfer learning methods to the complex architecture of Mask RCNN, in order to find a good strategy that fine-tunes the model to the new task.

With our approach, we were able to create a simple process for fine-tuning the *weights* of a complex model, using only, approximately, 1000 images for training and validation, each image only containing 1 positive instance (*i.e.* 1 robotic hand) and simple scenes, without any need for data augmentation or other pre-processing methods. When evaluating our models on some real environments we can retain most of the performance we get from evaluating in simulated environments. However, high cluttered environments can still affect this performance, significantly.

Furthermore, since it is a very time consuming task to collect and annotate real images, we are not able to create a real dataset large enough to make strong conclusions and evaluate the performance in many types of environment, with a high variability in the all the components of the environments.

Overall, we give a detailed insight on different learning strategies and data generation methods, which can be used to train a state-of-the-art network (Mask RCNN) and achieve the objective of this work, with a good performance.

With respect to the data generation process, future work can explore more randomization parameters and more suitable configurations for each parameter.

Concerning the learning process, although the implementation of Mask RCNN used allows for a large number of *hyperparameters* to be tuned, we only tune the learning rate. If any future work has the possibility of training the network with several GPUs, at the same time, we recommend that a more detailed *hyperparameter* search is made. Finally, another important addition to this work would be to include the information of the robot's kinematics, in the region proposal stage, in order to relieve the burden of generating proposals, from the network.

References

[1] B. Doosti. Hand pose estimation: A survey. *CoRR*, abs/1903.01013, 2019.

[2] V. Dumoulin and F. Visin. A guide to convolution arithmetic for deep learning. *arXiv preprint arXiv:1603.07285*, 2016.

[3] C. Farabet, C. Couprie, L. Najman, and Y. LeCun. Learning hierarchical features for scene labeling. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 35(8):1915–1929, Aug 2013.

[4] R. Girshick. Fast r-cnn. In *2015 IEEE International Conference on Computer Vision (ICCV)*, pages 1440–1448, Dec 2015.

[5] B. Hariharan, P. Arbeláez, R. Girshick, and J. Malik. Simultaneous Detection and Segmentation. In D. Fleet, T. Pajdla, B. Schiele, and T. Tuytelaars, editors, *Computer Vision – ECCV 2014*, pages 297–312, Cham, 2014. Springer International Publishing.

[6] K. He, G. Gkioxari, P. Dollár, and R. B. Girshick. Mask r-cnn. *2017 IEEE International Conference on Computer Vision (ICCV)*, pages 2980–2988, 2017.

[7] K. He, X. Zhang, S. Ren, and J. Sun. Deep residual learning for image recognition. In *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 770–778, June 2016.

[8] M. J. Jones and J. M. Rehg. Statistical Color Models with Application to Skin Detection. *International Journal of Computer Vision*, 46(1):81–96, jan 2002.

[9] P. Kakumanu, S. Makrogiannis, and N. Bourbakis. A survey of skin-color modeling and detection methods. *Pattern Recognition*, 40:1106–1122, 03 2007.

[10] A. U. Khan and A. Borji. Analysis of hand segmentation in the wild. In *2018 IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 4710–4719, June 2018.

[11] J. Leitner, S. Harding, M. Frank, A. Förster, and J. Schmidhuber. Humanoid learns to detect its own hands. *2013 IEEE Congress on Evolutionary Computation, CEC 2013*, 06 2013.

[12] G. Lin, A. Milan, C. Shen, and I. Reid. Refinenet: Multi-path refinement networks for high-resolution semantic segmentation. In *2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 5168–5177, July 2017.

[13] T.-Y. Lin, P. Dollár, R. B. Girshick, K. He, B. Hariharan, and S. J. Belongie. Feature pyramid networks for object detection. *2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 936–944, 2017.

[14] 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. *Lecture Notes in Computer Science*, page 740–755, 2014.

[15] P. Moreno, R. Nunes, R. Figueiredo, R. Ferreira, A. Bernardino, J. Santos-Victor, R. Beira, L. Vargas, D. Aragão, and M. Aragão. Vizzy: A Humanoid on Wheels for Assistive Robotics. In L. P. Reis, A. P. Moreira, P. U. Lima, L. Montano, and V. Muñoz-Martinez, editors, *Robot 2015: Second Iberian Robotics Conference*, pages 17–28, Cham, 2016. Springer International Publishing.

[16] K. Perlin. Improving noise. *Proceedings of the 29th annual conference on Computer graphics and interactive techniques - SIGGRAPH '02*, 21:681, 07 2002.

[17] P. O. Pinheiro, R. Collobert, and P. Dollár. Learning to segment object candidates. In *Proceedings of the 28th International Conference on Neural Information Processing Systems - Volume 2*, NIPS'15, pages 1990–1998, Cambridge, MA, USA, 2015. MIT Press.

[18] P. O. Pinheiro, T.-Y. Lin, R. Collobert, and P. Dollár. Learning to Refine Object Segments. In B. Leibe, J. Matas, N. Sebe, and M. Welling, editors, *Computer Vision – ECCV 2016*, pages 75–91, Cham, 2016. Springer International Publishing.

[19] S. Ren, K. He, R. Girshick, and J. Sun. Faster r-cnn: Towards real-time object detection with region proposal networks. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 39, 2015.

[20] O. Ronneberger, P. Fischer, and T. Brox. U-Net: Convolutional Networks for Biomedical Image Segmentation. In N. Navab, J. Hornegger, W. M. Wells, and A. F. Frangi, editors, *Medical Image Computing and Computer-Assisted Intervention – MICCAI 2015*, pages 234–241, Cham, 2015. Springer International Publishing.

[21] O. Russakovsky, J. Deng, H. Su, J. Krause, S. Satheesh, S. Ma, Z. Huang, A. Karpathy,

A. Khosla, M. Bernstein, A. Berg, and L. Fei-Fei. Imagenet large scale visual recognition challenge. *International Journal of Computer Vision*, 115(3):211–252, 12 2015.

[22] E. Shelhamer, J. Long, and T. Darrell. Fully convolutional networks for semantic segmentation. *IEEE Trans. Pattern Anal. Mach. Intell.*, 39(4):640–651, Apr. 2017.

[23] J. Tobin, R. H. Fong, A. Ray, J. Schneider, W. Zaremba, and P. Abbeel. Domain randomization for transferring deep neural networks from simulation to the real world. *2017 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 23–30, 2017.

[24] P. Vicente, L. Jamone, and A. Bernardino. Robotic hand pose estimation based on stereo vision and gpu-enabled internal graphical simulation. *Journal of Intelligent & Robotic Systems*, 83(3):339–358, Sept. 2016.

[25] J. Yosinski, J. Clune, Y. Bengio, and H. Lipson. How transferable are features in deep neural networks? In Z. Ghahramani, M. Welling, C. Cortes, N. D. Lawrence, and K. Q. Weinberger, editors, *Advances in Neural Information Processing Systems 27*, pages 3320–3328. Curran Associates, Inc., 2014.