Politecnico di Torino

Faculty of Engineering

**Master in Computer Engineering**
**Embedded Systems**


# Convolutional Neural Networks-based recognition pipeline for Robot Manipulation

*Supervisor:*
Prof. **Andrea Calimera**

*Candidate:*
**Alessandro Costantini**
*Matr:232520*

December 2018

# Contents

# List of Figures

# List of Tables

# Acknowledgements

This thesis represents the final step in my studies, and it contains the overview of six months of work at the Brown University Engineering Research Center. In particular, I took part in a robotic project designed by Professor Iris Bahar of Brown University and professor Odest Chadwicke Jenkins of University of Michigan.

This project aims to develop a new computational flow to aid in the design of robot perception techniques. Within this project scope, Professor Jenkins and his group at the University of Michigan will explore robust methods for simultaneous object and pose recognition and professor Bahar and her group at Brown University will explore efficient methods for CNN-based convolution. The project has been executed as a tightly coordinated collaboration via weekly video conference meetings with the entire research group and through smaller face-to-face meetings with researchers from each institution.

I would first like to thank my thesis advisor Andrea Calimera of the Politecnico di Torino. He gave me the possibility to participate in this project supporting me in my travel in the US.

I would also like to thank Professor Iris Bahar, my supervisor at Brown University, that accepted me to take part in the robotic project and introduced me in the powerful world of machine learning.

Moreover, I would to express my respect to professor Odest Chadwicke Jenkins, person in charge for University of Michigan, for the great knowledge that he transmitted to me.

In the end, I must express my very gratitude to my colleagues Yanqi Liu (Brown University) and Zhiqiang Sui, Zhefan Ye, Shiyang Lu (University of Michigan) for the great job that we made together and for the mutual support, without which we would not be able to achieve the obtained results.

Author

Alessandro Costantini

# Chapter 1

# Introduction

## 1.1 Introduction

The future of robotics predicts that robots will interact more every day with human beings and their environments. To achieve this integration, robots need to acquire *human intelligence*, which is defined as the ability to learn and to apply knowledge and skills. There is a urgent need for algorithms to provide robots with such kind of *skills*. In particular, for robots to autonomously perform manipulation tasks in human environments, they should have a proper understanding of objects of objects in the surrounding environment, as well as their pose state and their geometry. These information are paramount to make robots able to take the right action. For common human environments, such perceptual understanding remains an open issue, super challenging when considering cluttered environments.

Object detection techniques used for robot perception have greatly improved over the last years, due to the wide involvement of *convolutional neural networks* (CNNs). However, the excellent performance achieved by CNNs depend heavily on large numbers of floating point operations, requiring considerable energy consumption. In particular, this cost grows with the number of layers and weight parameters in the CNN. For efficiency, however, this computation cost is not a viable solution for embedded platforms and on-board processing for autonomous robots, where resources available are not unlimited, and cloud computing is not always applicable.

The basic assumption behind this work is that very high accuracies are not needed for certain applications, like grasping and manipulation. Hence, the relaxed accuracy constraint might help to improve efficiency.

This analysis is performed within a two-stage framework composed by a *Faster-RCNN* object detection followed by pose estimation, via the Iterative Closest Point

(*ICP*) algorithm. In particular, it shows that a smaller network model reaches comparable accuracy to generate grasping results, as a larger CNN model. By accuracy, it has been considered not just the bounding boxes produced by the CNN network in its ability to detect objects, but more importantly, in the final pose estimation that represents the output from the second stage of our approach. This distinction is important because an CNN accuracy measurement may not reflect the value of their output as it is passed to pose estimation stage. Accuracy from the first stage is not necessarily an indication of overall accuracy after the second stage.

All the experiments has been done using a small dataset created specifically for the study in order to perform physical experiments. This work shows that a large dataset is not needed to achieve high accuracy. Moreover, runtime and energy consumption for Faster-RCNN object detection has been conducted on two different systems: a powerful GPU and a embedded platform for machine learning application.

Before the arrival of Convolutional Neural Networks, object recognition was performed using features-based algorithms, such as *SIFT* (Scale-Invariant Feature Transform). Unlike CNNs, which are power-hungry models, such algorithms does not require great computational resources, making them more suitable for resource-constrained application. A SIFT-based object detection approach in the first stage has been developed and tested to understand if it could guarantee reasonable tolerance.

Furthermore, this work shows that relying solely on the CNN to make hard decisions for scene perception tasks leaves several open issue and vulnerabilities, especially when dealing with complex scenes (e.g., with non-ideal lighting or clutter). It has been explored how a *generative sample* approach for pose estimation in the second stage can improve performance of the entire framework. These experiments vary CNN model complexity for object recognition and evaluate levels of inaccuracy that can be recovered by generative pose inference. These experiment shows promise in providing robust robot manipulation for complex scenes, that also have some similarities to scenarios used for malicious attacks.

# Chapter 2

# Background

## 2.1 Problem Statement

Robot manipulation in human environments is predicted to grow, as the number of researches done with the aim to provides robots with autonomous skills. In order to interact autonomously, robots should have knowledge about object in the surrounding human environments, such as location and pose state. However, performing manipulation in unstructured and cluttered environments is particularly challenging due to many factors.

In this study, it has been designed a physical model whit the aims to develop a new computational flow to aid in the design of robot perception techniques. The physical environment is composed by a robot in front of a table that contains a random scene of objects. The model receives as input a RGB-D observation from the robot's camera and provides a motion trajectory for robot execution of a manipulation action. The perception problem can be split in three stage: detection of relevant objects, pose estimation for these objects, and generation of the manipulation motion trajectory, as illustrated in Figure 2.1. This study concerns and presents only the first two stages. The manipulation is assumed successful if the pose estimation is correct. Formation of robot trajectories are mostly considered through the invocation of one of many possible motion planning algorithms [43]. In the first stage, input is provided as RGB image which contains a complex scene of objects, as shown is figure 2.1(a). For each relevant object, the detector will predict a 2D image-space rectangular, called bounding box, around the region of interest (i.e. where the interested object has been predicted) and a class label representing the identifier of the predicted class. An example of detection is showed in figure 2.1(b), where the *spray bottle* has been surrounded by a red bounding boxes predicted by

the object detector.

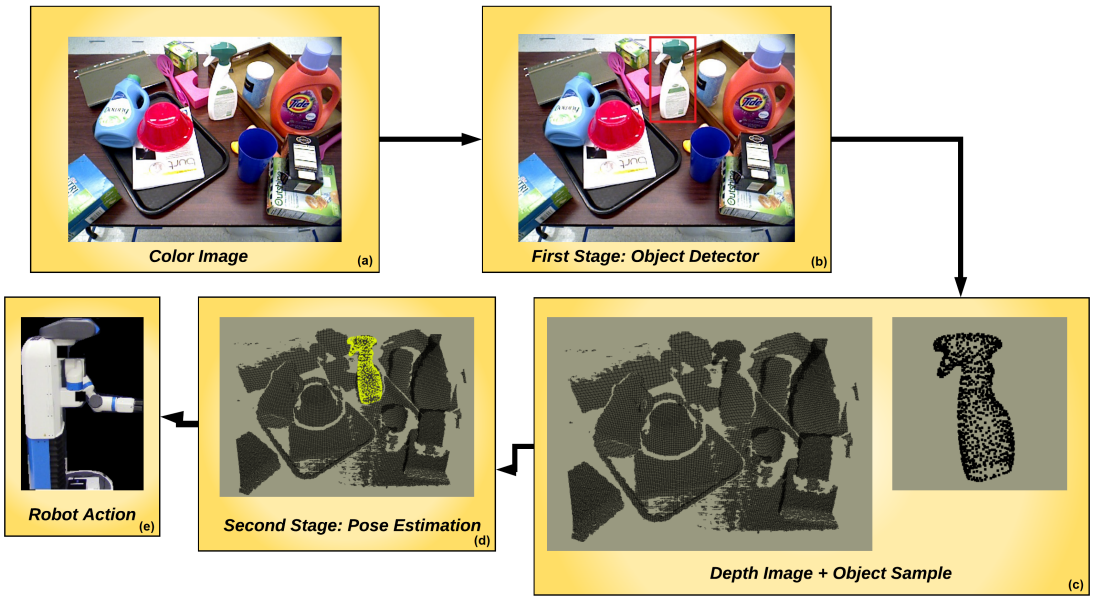



Figure 2.1: Framework Approach: detection, pose estimation, robot manipulation

Starting from first stage outputs, in the second stage, a 6DoF (six degree-of-freedom) pose will be estimated for each detected object, figure 2.1(d). It has been assumed that only one instance per object can be in the same scene. In the third stage,figure 2.1(e), manipulation actions consist of move a certain object from its estimated location to a desired pose.

Convolutional Neural Networks has reached high accuracy in computer vision tasks, such as classification and detection, founding place also in robotic applications where machine has to perform autonomously. In figure 2.2 is represented the usage of GPUs by the winner project of ImageNet Large Scale Visual Recognition Competition (ILSVRC) from 2010 to 2014. In 2012, with the appearance of Alexnet [23], GPUs has been stater to be utilized due to the high computational resource requested by the neural networks. With the appearance of deeper and improved networks, for example VGG [41] in 2014, the increasing ratio of GPU usage has been more significant respect the decreasing ratio of the error rate.

However the computational power of these network are not always available in embedded platform world. In this work has been performed an analysis of different model of Neural Networks within the framework described above, in order to comprehend how the accuracy, hence the power consumption, affects the final results

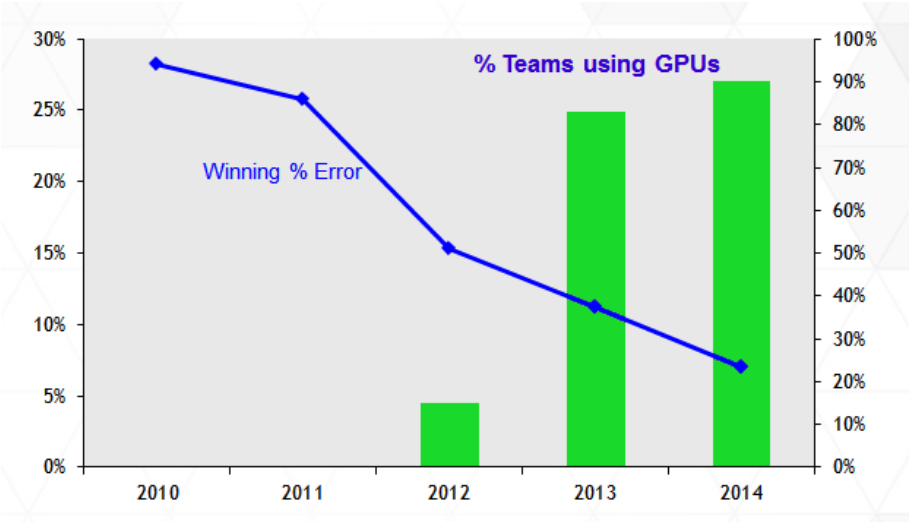



Figure 2.2: Usage of GPU by winner networks in ILSVRC over the year (source: nvidia.com)

of a system that does not rely only on CNN. Moreover, a generative-sample pose estimation stage has been developed to improve the robustness of the entire process.

## 2.2 First Stage: from Classification to Detection

For robots to autonomously perform manipulation tasks, they must recognize which objects are in the surrounding environment. Similar to human perception, machine works with images. When a computer sees an image (receive an image as input), it sees an array of pixel values, with a variable dimension depending on the resolution and size of that image. For example, with a color image (24 bit RGB JPG format) with size is 480 x 480, the computer receive an array with size equal to 480 x 480 x 3 bytes. Each of these bytes is a value from 0 to 255 which describes the pixel intensity. These numbers, meaningless to human, are the only inputs available to the computer.

In computer vision, classification consists of receive as input an image, and to outputs numbers that represent the probability of the image to belong to a certain class(like cat, dog, car, human, etc...). Starting form this number (i.e. pixel values) and with classification algorithms, the machine can recognize which objects are in the image.

Before 2012, most of researches made use of features-extraction algorithms and linear classifiers [40] [56]. But in the last years, machine learning, in particular deep

neural networks, has changed profoundly image classification approach, becoming the state-of-art for this type of applications.

### 2.2.1 Support Vector Machine

Support Vector Machine (SVM) is a supervised learning algorithm usually used for classification problems. Supervised means that it requires labeled training data. Given a training set of images, SVM represents them in a space whose dimensions are equal to the number of features of each point. SVM attempts to create a *linearly* separable hyperplane through these points in order to classify the data into two separated groups, maximizing the margin. Margin is the distance between the hyperplane and the closest points, called support vector. Test images are then mapped into the space and predicted to belong to a class based on region, which are created by the hyperplane, where they fall. When the number of features and images are huge, the separation hyperplane is not trivial to found, and the SVM can only provide an accurate solution. Exact solutions require very long time to be computed. In order to reach the most accurate solution, SVM has tunable parameters:

- **Kernel**: is a function that maps data to a higher dimension where the data is separable. Most popular are Gaussian and Polynomial.
- **Regularization**(also called **C**): determines the influence of SVM misclassification. For large values of C, the optimization will choose a smaller-margin hyperplane if that hyperplane could classify correctly all the training points. Conversely, a very small value of C will cause the optimizer to look for a larger-margin separating hyperplane, even if that hyperplane misclassifies more points.

Raw pixels data are no easy to be elaborated for machine learning tasks, or for images comparison in general. For example, taken images of the same objects changing scale, distance or light condition, could change dramatically the pixel value or even only the location (shifted and transformed in a complex way). Moreover, they can not be used by the SVM as representation of point in the space. Features, informally defined as interesting parts of images, instead, are at more holistic level than raw pixels, and they can be used by SVMs. SIFT represent a well studied technique developed to detect and describe features which can be used cleverly by a linear classifier, such as SVM.

### 2.2.2 SIFT

Scale-Invariant Feature Transform (**SIFT**) is a features detection algorithm that locate and characterize local features (called keypoints) in images and it collects them in "large local features vectors". Keypoints can be represented by a circular image with an orientation through four parameters: {x,y} coordinates of the center, scale of the region and orientation (angle). A SIFT keypoint is invariant to scaling, orientation, illumination changes, and it is not sensible in the presence of little quantity of noise [30], making it suitable for classification tasks. A SIFT descriptor, figure 2.3 [8], is a local image gradients at selected scale and rotation that describes each keypoint region.

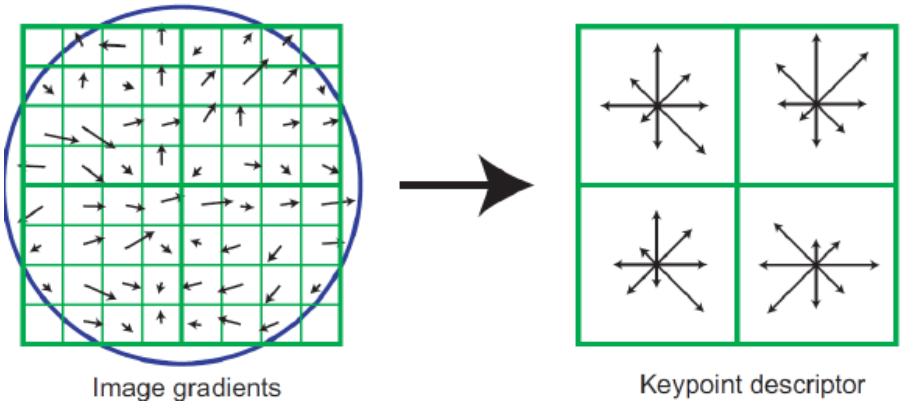



Figure 2.3: Representation of SIFT descriptor

### 2.2.3 BOVW

SIFT alone, like SVM, is not able to perform as image classifier, because it provides only a way to extract features. Bag of Visual Word (BoVW) [6], instead, is a framework that combines both approaches creating a objects classifier. Born initially for document classification, as Bag of Word, it has found place in computer vision [44] [42], where documents has been replaced with images and words with features, and the name has been updated to Bag of Visual Word. The algorithm is usually composed by three steps:

- **Features Extraction**: find interesting point in the images; SIFT, HOG (histogram oriented gradient) or SURF (Speeded up robust features) are the most common algorithm used in this step.
- **Codebook Generation** after the first step each image is represented as a

vector of features. This features has needed to be grouped by similarity. Every class in the dataset has certain characteristic features, also well recognizable by human eye. Similar features can provide an approximate estimate as to what the image is. Therefore when the machine is trained over several images, similar features are able to describe similar portions of a class, and they can be grouped together to develop a vast vocabulary base (or codebook). The grouping phase is called clustering. The most common algorithm to perform clustering is K-Means. Each of these groups collectively represents a word and all groups in totality shape a complete vocabulary generated from the training data.

- **Train & Test** The SVM are trained with the vocabulary. Then features are extracted from the test images and classified by the SVM model trained before.

### 2.2.4 Convolutional Neural Networks

Convolutional Neural Networks are, as ordinary Neural Networks, inspired by biological brain: they are connected neurons that have learnable weights and biases. Each neuron receives some inputs, performs a matrix multiplication and pass the results to one or many other neurons. Differently for regular Neural Networks, where all neurons of one layer are connected with all neurons of the next layer(*fully connected*), in CNNs neurons in one layer do not connect to all the neurons in the next layer but only to a small region of it. Moreover, CNN layers are organized in 3 dimensions: width, height and depth. CNN are typically used in computer vision tasks, such as image classification, where they represent the state-of-the-art: starting from the raw image pixels the predict class scores. Their name derives from the mathematical operation most involved in the hidden layer: the convolution. Actually, in CNNs, it is a cross-correlation rather than a convolution, but this term is used by convention. As regular Neural Network, CNN are composed by one input layer, one output layer and multiple hidden layer. The hidden layers of a CNN are typically of the following types:

**Convolution Layer** : Convolution is an operation on two functions (usually represented by matrices) of a real value [27] to produce a third function (also a matrix). In other words, it merges two sets of information. In this way the network is able to recognize interesting patterns (vertical edges, horizontal edges, round shapes, etc.). The two matrix used in the convolution are part of the input image and a small matrix called filter (or kernel). For example,

in VGG architecture filters are 3x3 matrices. Number contained in filters are called weights. Depending on these weights in the filter, the output matrix detects specific patterns present in the input image. Weights value are learned (i.e. updated) by the network during the backpropagation, that is the milestone in machine learning, and it is described in the next paragraph. CNNs are composed by various numbers of consecutive layers. Each layer is able to learn large number of kernels, depending of its dimensions. After that a layer has performed the convolution through its filters, it passes the outputs to the next layer that are able to learn their own kernels based on the received convolved image. This is what gives the CNN ability to "see" the various pattern in images and build them up into larger features. Figure 2.4[1] represents a visualization of filters in a convolution layer.

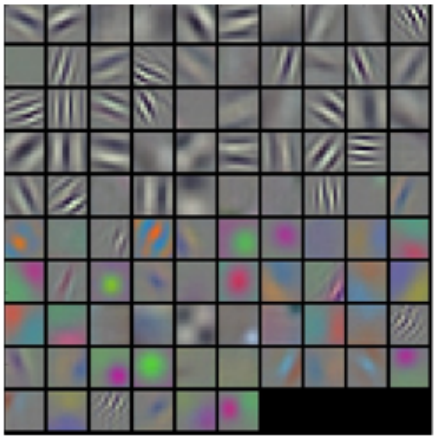


Figure 2.4: Visual representation of filters in a CNN

**ReLU** (Rectified Linear Units): The rectifier function is an activation function $f(z) = Max(0, z)$ which can be used by neurons to add non linearity to the network (Figure 2.5). ReLU usually follow a convolution layer and sets all negative values in the matrix produce by the convolution to zero and all other values are kept constant. Applying the rectifier function increase the non-linearity in the input images. A linear equation is easy to solve but it is limited in their complexity and have less power to learn complex functional mappings from data. Adding a non-linearity increase the learning power of the network. ReLU is the most common activation function because it can be computed efficiently respect other activation function (like the sigmoid and

[1]Source: CS231N course

hyperbolic tangent) without making a significant difference to generalization accuracy.

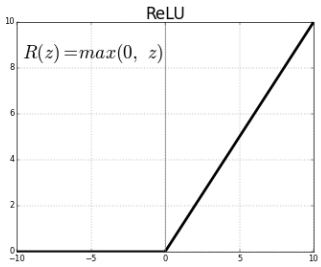



Figure 2.5: ReLU function

**Dropout** : it is a regularization method that randomly sets to zero the activations of hidden units at each training epochs. Randomly ignoring nodes prevents dependencies between nodes. This allows the network to learn more robust relationship and it prevents overfitting problem [18].

**Pooling** combine the outputs of neuron clusters at one layer into a single neuron in the next layer. The pooling layer serves to progressively reduce the spatial size of the layer, hence to reduce the number of parameters and amount of computation of the network. The most common used pooling layer is *max-pooling*, which slides a window on the input, like a normal convolution, and get the biggest value on the window as output. Other type of pooling units are average pooling or L2-norm pooling. Max-pooling, as ReLU, are the most used due to the lower complexity to be extracted.

**Fully Connected** fully connected (FC) layer takes all neurons in the previous layer (they can be part of fully connected, pooling, or convolutional layer) and connects it to every single neuron of the next layer. After feature extraction network need to classify the data into classes, this can be done using a FC neural network. In place of fully connected layers, it is also possible to use a SVM. But FC is generally added instead of SVM to make the model end-to-end trainable.

Convolution layer plus non-linear element (e.g. ReLU) plus the polling layer form the Convolutional Layer, as shown in figure 2.6

The feature extraction and learning process of SIFT is very different than CNNs. SIFT is much more simpler to design and it has less number of parameters compared to CNNs, hence SIFT needs much less processing power and memory. Moreover

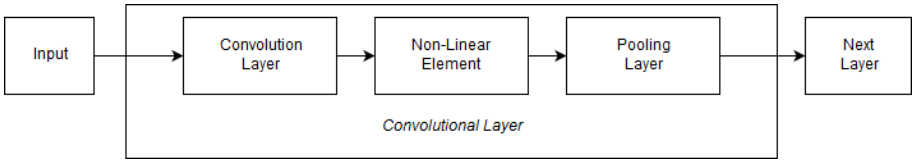



Figure 2.6: Basic Scheme of Convolutional Layer

SIFT is comparatively fast. Although, SIFT is more relevant for identification tasks while CNNs has good generalization abilities, due to the learning capacity, and they reach better for classification and categorization tasks.

### Training a CNN

CNNs are a sub-family of Deep Learning algorithm, hence they have some type of *gradient descent* in order to *learn*. In CNNs world, *gradient descent* is performed by *backpropagation*. Backpropagation can be resumed into 4 steps: the forward pass, the loss function, the backward pass, and the weights update. During the *forward pass*, the network receives the image as input and predicts the output. After the prediction, the *loss function* is calculated. It represents the difference between the estimated and the true values respect the input image. The goal is to have a prediction equal to the original label of the image. This is reached by the minimization of the loss function. The aim of *backward pass* is to find which weights are more meaningful in order to reduce the loss function and to adjust them. Once computed this derivative, *weight* are *update*. Once trained the CNN, test phase can be performed: using new images(never seen by the network), it consists of comparing the outputs with the ground truth and see if the network works.

### AlexNet

Created in 2012 by Alex Krizhevsky, Ilya Sutskever and Geoffrey Hinton [23], to compete in the ImageNet Large Scale Visual Recognition Challenge(ILSVRC), it reached a Top-5 error rate of 15.3%. Top-5 error meaning that the predicted label is one of top 5 predictions (the 5 ones with the highest probabilities).

It can be considered the pioneer CNN network and it has great impact in the world of computer vision. It is composed by 5 Convolutional Layers and 3 Fully Connected Layers and the inputs are RGB image of size 256x256 pixels. Random crops of size 227x227 (instead of 224x224[2]) were generated from inside the 256x256

[2]paper contains input images equal to 224x224, but numbers make sense only of actually 227 by 227

images to feed the first layer of AlexNet. Relu is applied after convolutional and fully connected layer while dropout is applied before the first and the second fully connected layer (Fig. 2.7).

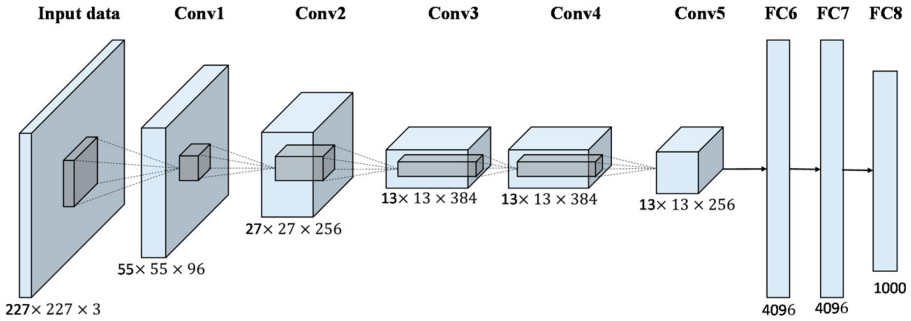



Figure 2.7: AlexNet Architecture

The network has 62.3 million parameters, and needs 1.1 billion computation units in a forward pass. Convolutional layers, which contain 65% of all the parameters, consumes ≈95% of the computation.

### VGG

Visual Geometry Group (VGG) is a convolutional neural network model, proposed by K. Simonyan and A. Zisserman, appeared in ILSVRC in 2014 [41]. VGG substitutes first two layers of Alexnet (with size 11 e 5) with concatenated filter of size 3x3, which increases the depth of the network and enables to learn more complex features. The VGG convolutional layers are followed by 3 fully connected layers as Alexnet. It achieves the top-5 accuracy of 92.3 % on ImageNet. Figure 2.8 shows different implementations of vgg architecture: VGG11, VGG13, VGG16, VGG19 are represented respectively by lecter A,B,D,E. This architectures are different in the number of layers. Deeper networks have more filter, hence they can learn and detect most pattern, reaching higher accuracy. On the other hand, smaller networks require less data in order to converge, and the training time results faster. By observing the addition of layers one by one, with VGG-16 and VGG-19 the accuracy improvement is slowing down. Adding more layer can not bring best results and it can also lead to a performance degradations. This issues will be solved with the ResNet architectures. When people are talking about VGGNet, they usually mention VGG-16 and VGG-19.

| ConvNet Configuration | | | | | |
|---|---|---|---|---|---|
| A | A-LRN | B | C | D | E |
| 11 weight layers | 11 weight layers | 13 weight layers | 16 weight layers | 16 weight layers | 19 weight layers |
| input (224 × 224 RGB image) | | | | | |
| conv3-64 | conv3-64<br>**LRN** | conv3-64<br>**conv3-64** | conv3-64<br>conv3-64 | conv3-64<br>conv3-64 | conv3-64<br>conv3-64 |
| maxpool | | | | | |
| conv3-128 | conv3-128 | conv3-128<br>**conv3-128** | conv3-128<br>conv3-128 | conv3-128<br>conv3-128 | conv3-128<br>conv3-128 |
| maxpool | | | | | |
| conv3-256<br>conv3-256 | conv3-256<br>conv3-256 | conv3-256<br>conv3-256 | conv3-256<br>conv3-256<br>**conv1-256** | conv3-256<br>conv3-256<br>**conv3-256** | conv3-256<br>conv3-256<br>conv3-256<br>**conv3-256** |
| maxpool | | | | | |
| conv3-512<br>conv3-512 | conv3-512<br>conv3-512 | conv3-512<br>conv3-512 | conv3-512<br>conv3-512<br>**conv1-512** | conv3-512<br>conv3-512<br>**conv3-512** | conv3-512<br>conv3-512<br>conv3-512<br>**conv3-512** |
| maxpool | | | | | |
| conv3-512<br>conv3-512 | conv3-512<br>conv3-512 | conv3-512<br>conv3-512 | conv3-512<br>conv3-512<br>**conv1-512** | conv3-512<br>conv3-512<br>**conv3-512** | conv3-512<br>conv3-512<br>conv3-512<br>**conv3-512** |
| maxpool | | | | | |
| FC-4096 | | | | | |
| FC-4096 | | | | | |
| FC-1000 | | | | | |
| soft-max | | | | | |

Figure 2.8: Possible configurations of VGG

### ResNet

As it is possible to see, from Alexnet to VGG, increasing the depth of the network also accuracy increases, but with a limit. During the backpropagation phase, the signal that should change the weights, becomes very small at the earlier layers, because of increased depth. Hence the learning rate of that earlier layers are almost negligible. This issue is called *vanishing gradient*.

Another problem related to the depth of a neural network is called degradation problem. With network depth increasing, accuracy gets saturated and then degrades rapidly and adding more layer leads to higher training error. The core idea of Residual networks, ResNet [15] is introducing a "*identity skip-connections*" that skips one or more layers. ResNet allows training of such deep networks by constructing the network through blocks called residual block (Figure 2.9). ResNet won 1st place in the ILSVRC 2015 classification competition with top-5 error rate of 3.57%

In this work, four different ResNet has been used: ResNet18, ResNet34, ResNet50, ResNet101, ResNet152, where the number represent the number of layers. Resnet152 had 152 layers, 10 times deeper than VGG16 networks. The residual module permit to reach such depth, never reachable by a VGG architecture due to the problem described above.

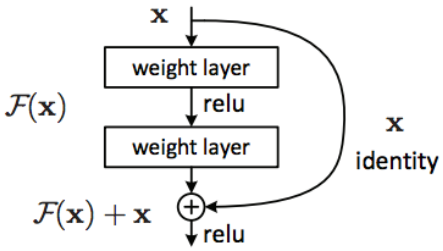



Figure 2.9: residual block: the fundamental building block of ResNet. Source https://arxiv.org/pdf/1512.03385.pdf

#### MobileNet

MobileNets [19], are a efficient convolutional neural network class for mobile and embedded vision designed by researches at Google. MobileNets split the classical convolution into a 3x3 depthwise convolution and a 1x1 pointwise convolution. A depthwise convolution performs convolution keeping the channels separate. In CNNs, where the depth is equal to three, MobileNets use one filter for each of the three dimensions in the depthwise convolution. Pointwise convolution, instead uses a 1x1 kernel, or a kernel that iterates through every single point. MobileNets are not usually as accurate as the bigger network, but it find application context where resource-accuracy trade-off has more tolerance. Table shows the computational complexity between MobileNet and VGG16, extracted from [19].

| Framework | Model | mAP | Mult&Add (Billion) | Paramenters (Million) |
|---|---|---|---|---|
| Faster R-CNN | VGG16 | **22.9%** | 64.3 | 138.5 |
| | MobileNet | **16.4%** | 25.2 | 6.1 |

Table 2.1: COCO object detection results comparison using different frameworks and network architectures.

### 2.2.5 Object Detection

In this work, the goal is to detect multiple instance of different objects in the same image, hence it is necessary to transform the classification problem into a detection problem. In very simple terms, classification means answering *what*, and detection means answering *where* and to answer to the latter question is obviously more difficult, but there are algorithm that permit to transform a classification task into a detection one:

- **Sliding Window Algorithm**
- **region proposals network**
- **YOLO solution**

Figures 2.10 and 2.11 shows an example of a detection: red bounding boxes are drawn around the interesting object in order to describe their location in the image.


Figure 2.10: Example of a input image: table with household objects (some belonging to the dataset and some not). Not all the object are completely visible.


Figure 2.11: Example of Object Detection on the input image. Red line are called Bounding Boxes

### Sliding Window

A rectangular region of fixed dimensions that "slides" across an image creating a set of sub-images, each of which contains only a region of the initially image. Each of these sub-image are used as input by the normal classifier, which determines if the region contains some "object of interest", converting the classifier into a detector. The parameters of this algorithm are the dimensions of the window and the bin size. The bin size is the step, in pixels, of the window when it slides across the x or y axes. (Figure 2.12).

In Figure 2.12 the dimensions of the image are 600*400. Using a window's size equal to 150*150 and a bin size of 50 (a black square of the grid is 50*50). Hence the total number of sub-images are 45. Halving the bin size, the number of total sub-images grows by a factor of four, obtaining 180. A small size creates more number of sub-image to be elaborated, increasing the runtime of the algorithm. On the

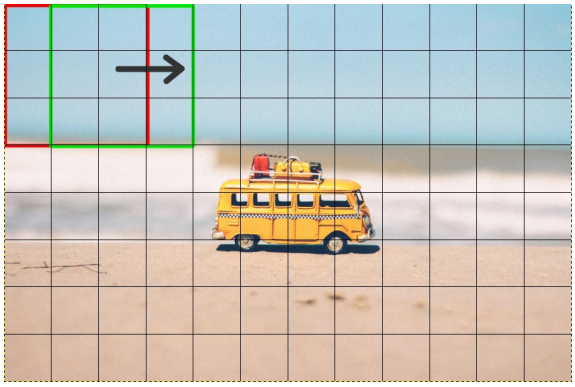


Figure 2.12: Example of a Sliding Window Approach: the image is divided in sub-image, and every image is elaborated separately

other hand, choosing a big windows size increase the noise amount in the image(i.e. background), probably making more difficult the classification task.

### Region Proposal

In order to use CNNs in detection applications instead of the merely classification, many studied have proposed different approaches showing that a CNN can lead to higher object detection performance as compared to systems based on simpler features extractor. The first region proposal can be found in RCCN [11], where these region are created using a process called Selective Search. Selective Search creates windows of different sizes, and for each size tries to group together adjacent pixels by texture or color. These region proposals are warped into a square in order to be elaborated by a convolutional neural network.The CNN works as a feature extractor. These features are inputs for a SVM that classify the region proposal. Although it has shown good results, the number of regions (around 2000 per image) makes the execution time not indifferent. A further improve comes with Faster-RCNN [38], where, to predict region proposals, an additional network, called RPN, is used instead of using selective search algorithm to find the region proposals. An Faster-RCNN is a network that simultaneously predicts object bounds and object scores at each position, merging RPN module and Fast R-CNN into a single network by sharing their convolutional features. RPN can be summarized in 3 steps:

- from the input image, a convolution network extracts a feature maps
- Then sliding window is run on these feature maps. The size of sliding window is n x n. For each sliding window, a set of n x n anchors are generated which

all have the same center.

- Finally, the n x n spatial features extracted from the feature maps are elaborated by another network which has two tasks: classification and regression. The first output is the class probability while the second outputs determines a predicted bounding box.

#### YOLO

YOLO (You Only Look Once), is a framework for object detection [37] that gets rid of region proposal, that are the milestone for RCNN family, and involve a single neural network predicts bounding boxes and class probabilities directly from full images in one evaluation. YOLO divides the input image into an SxS grid. Each grid predicts B bounding boxes and confidence scores and a set of class probability. At test time YOLO multiplies the class probabilities and the box confidence score obtaining a class confidence scores for each box. YOLO has 24 convolutional layers followed by 2 fully connected layers. The former are used to extract features from the input image, while the latter predicts output probabilities and coordinates. Differently from Faster-RCNN, that is composed by two networks, YOLO benefits of only one network to reduce the total runtime, that makes it extremely fast.

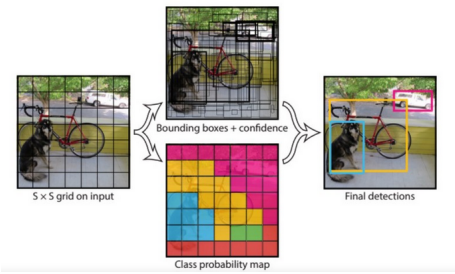



Figure 2.13: YOLO workflow

However, has shown in figure 2.14[3] YOLO accuracy fails to achieve comparable accuracy as Faster-RCNN, hence the latter has been chosen for this work.

## 2.3 Second stage: Pose Estimation

In robotic manipulation tasks, it is required to estimate the pose of objects in order to plan motion actions. However, unlike the first stage, only the 2D-image can

[3]source: https://cv-tricks.com/

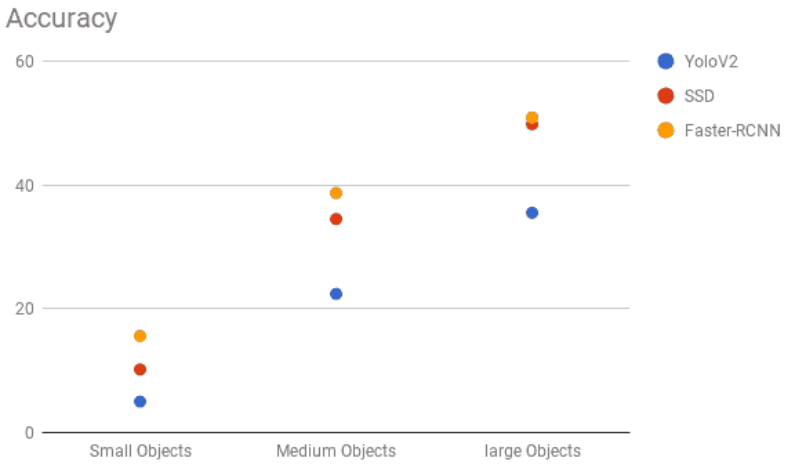



Figure 2.14: YOLO vs SSD vs Faster-RCNN for various sizes

be not enough to estimate this pose. Hence the use of RGB-D camera is required. It provides a depth map of the scene represented by acloud of points. A point cloud is a collection of points in 3D-space, which are on the external surfaces of objects. Hence, this map contains information about the distance of each point of the scene's surfaces from the camera. An example in figure 2.15, which it contains the complete point cloud of a bottle od Downy, objects presents in the dataset used for this study. It is obtained merging many depth images of the object that has been taken from different views.

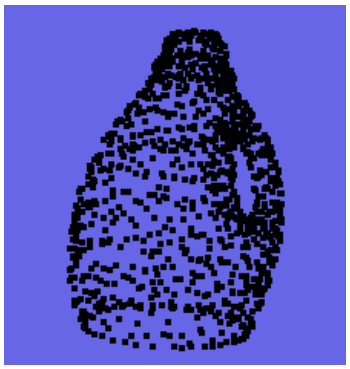


Figure 2.15: Front view of Detergent Bottle cloud point

### 2.3.1 ICP

Iterative closest point [1] is an algorithm which purpose is to find the "minimum distance" between two collections of points. ICP represent the state-of-art in pose

estimation to achieve optimal path planning for robots. The main idea of the algorithm is iteratively modify the translation and the rotation needed to minimize the distance from the input sample point cloud to the reference. This distance is defined as the sum of squared differences between the coordinates of the two point clouds. Fig.2.16 shows an ICP example applied on a line (2d function).

Figure 2.16: Example of ICP algorithm applied on a 2D group of point

Due to the fact that there is no information about the initial pose of object, ICP can take different time (i.e. number of iteration) to find the match between the two cloud. This means that ICP is sensitive to the initial pose of the object.

### 2.3.2 Particle Filter

Particle filters or Sequential Monte Carlo (SMC) is a family of algorithm used for estimating information starting from partially or noisy observation. The particle filtering can be resumed in the following steps:

1. **Generate** with a random distribution of a collection of samples (*particles*).
2. **Assign** a weight to each sample. This value represents how likely the particle is respect the observation. If the particle is very similar to the observation, the weight will be high, otherwise, a low value will be assigned to it. In this way, part of the collection appears to be more likely than the others.
3. **Resampling**: substitute the particles with low weight with other generated sample. This new sample are not generated with a random distribution like the first ones, but they will be generated based on the particles with higher value.
4. **Diffusion**: the regenerated particle will have the same weight to the old ones, making them useless. With diffusion, a random transformation is applied on the whole collection.

5. **Re-evaluate** the collections of sample, i.e update the weight after the diffusion, and repeat the resampling-diffusion until all the particle converge to be consistent with the observation.

In this work the observation are represented by the point clouds cropped by the bounding boxes and the sample are hypothesis of the pose state of the objects. Unlike the ICP, this approach is not sensitive to the initial pose of the object because many initial orientation are compared at the same time.

## 2.4 Performance Metrics

In order to evaluate detection results, a metrics is necessary for both object detection and pose estimation. For the detection side,it has been use the evaluation standard method from the Pascal VOC Challenge [7]. The first stage has as output a set of bounding boxes, each of which hold a confidence score. For each test scene a annotation file contains all the ground truth bounding box.

If the overlapping area, defined as intersection over union (IoU), between the detected bounding box with a certain class label and the ground truth of the same class is greater than 0.5, the detection is considered correct, or called true positive (TP). Otherwise, if the IoU is under 0.5 it is called false positive (FP). For each class of object, if the predicted bounding box does not overlap any ground truth If no detections satisfy the overlapping criterion with the ground truth, it becomes a false negative (FN).

Average Precision (AP) is the mean precision given different recall thresholds, which describes the shape of the PR curve. Precision is defined as $\frac{TP}{TP+FP}$, and recall is defined as $\frac{TP}{TP+FN}$. Finally, the mean average precision (mAP) of all object classes is obtained through all the AP for each object class.

For evaluating pose accuracy, the formula of mean of the pairwise point distance from two sets of point clouds from [17] has been used:

$$m = avg_{x_1 \in M}(min_{x_2 \in M}(\|(Rx1 * T) - (\bar{R}x2 - \bar{T}))\|) \tag{2.1}$$

Where M is the estimated model, R and T respectively the ground truth rotation and translation, and $\bar{R}$ and $\bar{T}$ the estimated rotation and translation, while $\mathbf{x}$ are the model point. This metric is also able to consider symmetric objects, which is useful for some object in the dataset. Such object are: *sugar box,toy, salt box, red bowl,coke can, clorox bottle, blue cup, ranch bottle.*

## 2.5 Object Segmentation

With object recognition, the neural network (or any of the other approach for object recognition) "draw" a bounding box around the desired object in a image. In most of the computer vision tasks, this type of object labeling can be acceptable enough for the scope. Although, many other tasks, require more precision about the boundaries of wanted objects, and bounding boxes are too generic. If object detection is able to locate the region that contains object, the object segmentation responds to the need to locate the boundaries in addition to that region. Image segmentation assigns a label to every pixel, and it is able to create different bounding shape instead of simply rectangular. Fig 2.17 shows an example of object segmentation applied on a vehicle recognition. All the pixel that are detected to belong to the "*car*" are colored in green, while all the other (black ones) are not labeled or label as "*background*". In the end, before the training phase, all the images have to be labeled in a right way in order to train correctly the network, and labeling is done by human. While bounding boxes are simple a couple of coordinates (two vertices of the rectangle), the segmented section is a complex and usually long list of pixel coordinates that compose the outline of the image. So label one image for segmentation require a bigger effort than to label for detection. For datasets that contain thousand images for train (like the one used for this study), the time used to create the dataset is not irrelevant. For this reason, segmentation is not taken into account in this work, but a further explanation of advantages and disadvantages within a possible future use of segmentation has discussed in the Chapter 4, where the two different pipeline used for this study are explained in detail.

Figure 2.17: Example of Image segmentation

# Chapter 3

# Related Work

After the appearance of AlexNet [23] in the ImageNet LSVRC-2010 contest, usage of neural networks in computer vision tasks has grown more and more. One of the most popular object detector framework, Regional Convolutional Neural Network (RCNN) [11], combines convolutional layers to extract the features of proposed bounding boxes and a linear classifier to classify the object. However, the execution time did not make it possible to use it in real time applications, due to the large number of region of interest (ROI) proposed by the selective search. (≈2,000). Fast RCNN [10] improves the runtime of the object detector, warping ROIs into one single layer using the RoI pooling, . With a speedup solution,a further improvement has been presented in Faster RCNN [38], by adding a extra convolutional network as a Region Proposal Network (RPN) that shares convolutional layers with the object detection network and reduces the cost of computing regional proposals. While the R-CNN family approach treats the task as a classification task, the YOLO approach [37] do it as a unified regression problem. It forgets the regional proposal pipeline,the milestone in all RCNN study, and uses a single CNN network for both classification and localizing the object using bounding boxes. However, it is not able to achieve a comparable accuracy as faster RCNN. A recent study, Mask-RCNN [55], shows an extension of Faster RCNN that with the addition of a new branch, it predicts a binary mask indicating whether or not a given pixel is part of an object, which greatly improves the detection accuracy. However, Mask-RCNN requires that training images with annotation specifically created for segmentation, requiring huge effort respect bounding box methods.

With the increasing of neural network depth, there was also a considerable increase in accuracy [48], but the energy consumption has become more significant withal. Recent works have explored the hardware complexity of the neural network

and how to design lighter networks that are more hardware friendly, while achieving similar accuracy. In particular, the work of [47] presents a summary of popular deep neural network architectures trained on ImageNet in terms of accuracy as well as computation complexity. The work of [47] also introduces various algorithms and hardware optimization for designing energy efficient deep neural network. However, there are no works targeted specifically on analysis accuracy, hardware cost or and energy consumption for CNN involved as object detector within robot manipulation applications.

Given the large number of floating point operations required in neural network, coupled with the sparsity in weights and feature maps, a lot of works have focused on network compression. Two common strategies are pruning to reduce the number of weights and quantization to reduce bits used to store weights. In the work of Han et al. [14], the authors learn connections within networks, prune low weight connections, and fine-tune sparse network in order to compress network parameters by 9× and 13× for AlexNet[23] and VGG16[41] respectively. For network quantization, precision reduction of both weights and feature maps are used. One common technique is floating point to fixed point conversion [28]. BinaryNet [5] and TernaryNet [60] further compress the network by having weights and activations represented as (-1, 1) and (-1,0,1) respectively. This not only reduces the network size but also replaces most arithmetic operations with bit-wise operations.

There are a few works that present CNN designs specifically for embedded hardware applications. For instance, SqueezeNet [22] and MobileNet [19] are two CNN architectures that are designed for embedded and mobile applications. Squeezenet uses $1 \times 1$ filters and decreases the number of input channels to reduce the network parameters while increasing the network depth. MobileNet uses depth-wise separable convolution to reduce the parameters and computations, which also makes the model more amenable to hardware parallel computing.

Long *et al.* [29] propose fully convolutional networks (FCN) for semantic segmentation by replacing fully connected layers in traditional CNN with $1 \times 1$ convolutional layers. FCNs take images of arbitrary size and provide per-pixel classification labels. However, FCNs are not able to separate neighboring objects within the same category to obtain instance-level labels; hence it is not possible to directly re-task FCN for object detection purposes. Nonetheless, most unified approaches are based on FCN to localize and classify objects using the same networks. Recently, there has been a trend to utilize FCN to perform both object localization and classification [39], [38] [37], [20]. Sermanet *et al.* propose an integrated CNN framework for

classification, localization and detection in a multiscale and sliding window fashion [39]. Morris *et al.* [31] propose a fully-convolutional Pyramid Network in operate at successive resolutions as information flows up the pyramid to the lowest resolution. In this work, the input to SAND approach's CNN stage is a pyramid of images with different scales in order to generate a heatmap for the second stage.

Moreover, while many study [23] [41] [12] make use of large database, other research has been focused on training network with smaller datasets. In the works of Lai et al. [26, 25] and Silberman et al. [33], has been utilized dataset composed composed of a much less significant number comparing to ImageNet. As explainded it these work, with this study is shown that , in order to achieve high level of accuracy, a huge database is not necessayThese representative datasets resemble of the scale of training assumed in this paper.

Reliable operation of autonomous mobile manipulators remains an open challenge for robotics, where perception remains a critical bottleneck. Within the well-known sense-plan-act paradigm, truly autonomous robot manipulators need the ability to perceive the world, reason over manipulation actions afforded by objects towards a given goal, and carry out these actions in terms of physical motion. However, performing manipulation in unstructured and cluttered environments is particularly challenging due to many factors. Particularly, to execute a task with specific grasp points demands first recognizing object and estimating its precise pose.

For object and pose estimation, PR2 interactive manipulation [3] segments non-touching objects from a flat surface by clustering of surface normals. This work uses RGBD data from cameras that provide both color and depth values at every pixel. Similarly, Collet *et al.*presented a discriminative approach, MOPED, to detect object and estimate object pose using iterative clustering-estimation (ICE) using multiple color cameras [4]. Narayanan *et al.* [32] integrate A* global search with the heuristics neural networks to perform scene estimation from RGBD, assuming known identification of objects. Papazov *et al.* [35] used a bottom-up approach of matching the 3D object geometries using RANSAC and retrieval by hashing methods.

Deep learning on RGBD has also been applied to robotic grasp detection through deep reinforcement learning, such as work by Gualtier *et al.* [13], and supervised shape completion, such as by Varley *et al.* [53]. For manipulation in cluttered environments, Ten Pas *et al.* have shown success detecting viable grasp poses in RGBD point clouds using geometric inference [51] and estimation by deep neural networks [50]. Sui *et al.* [45] built on these methods to recognize objects as well as

graspable poses, in order to perform purposeful goal-directed manipulation. This work combines the output of discriminative inference methods, such as CNNs, with probabilistic generative inference to improve robustness. While demonstrating effectiveness, the methods above use neural network architectures, such as VGG16, that is expensive in both computation and energy. These models also assume improved recognition accuracy directly implies improvement in robot manipulation, which needs greater validation experimentally.

Szegedy *et al.*[49] demonstrated that adversarial examples are misclassified by different classifiers both in the case of different architectures or different subsets of the training data[21] [2][52]. These results are confirmed also in cases where the differences between these examples were indistinguishable to the human eye. Kurakin *et al.* [24] confirmed the results in a simple physical scenario. Papernot *et al.* [36] showed a case of a black-box attack against a neural network, where adversaries have no knowledge about the model. Others works proposed a possible solution during the training phase: Panda *et al.* [34] proposed to inject random noise on the training data and Zheng *et al.* [59] presented a stability training method to avoid mis-prediction due small input distortion.

Defending against adversarial examples is not a trivial problem because it requires neural network models to produce good outputs for every possible input; however, in a physical scenario neural networks must work well only on a very small number of all the many possible inputs. In this work, in order to deal with adversarial scenarios,it has been chosen to not modify the initial training set, thereby avoiding an excessive human effort during the data collection phase.

# Chapter 4

# Two-Stage Framework For Scene Estimation

The purpose of this work is to demonstrate that for certain tasks, such as robot manipulation, the same final results can be reached with neural network which are not necessarily deep, and therefore the most accurate.

In order to validate this statement, a neural network analysis has been performed through a two-stage pipeline for robot perception called *Baseline*, discussed in section 4.4. This pipeline is composed by a Faster-RCNN object detector plus a simple pose estimation method through the Iterative Closest Point Algorithm. After the results obtained by this pipeline,hence a full scene estimation, robot can plan motion actions.

Moreover,CNNs output are usually used as input by a different system in order to take some type decisions. These decisions can be very crucial in those tasks, such as robot manipulation in cluttered human environment, that require a certain level of robustness. In the section 4.5, it is explained a further two-stage pipeline, called *SAND*, that introduce a different pose estimation method through a generative probabilistic inference that is able to recover potentially imperfect decisions produced by a CNN increasing the robustness of the entire framework. *SAND* method, beside compensate hard-decisions made by the first stage, and it also permits to use less accurate networks in the first stage, hence less computational effort, hence less energy consumption. While the *Baseline* first-stage is implemented by the Faster-RCNN, for the *SAND* a Pyramid-CNN framework has been chosen in order to create an output more exploitable by the novel pose estimation method. Figure 4.1 resumes the main sections of the two distinct pipeline used for this study. Starting from the same inputs, a single colored image and a single depth image of the same scene,

both approaches will produce a full scene estimation. The scene estimation will be further used by the robot in order to carry out actions in terms of physical motions.

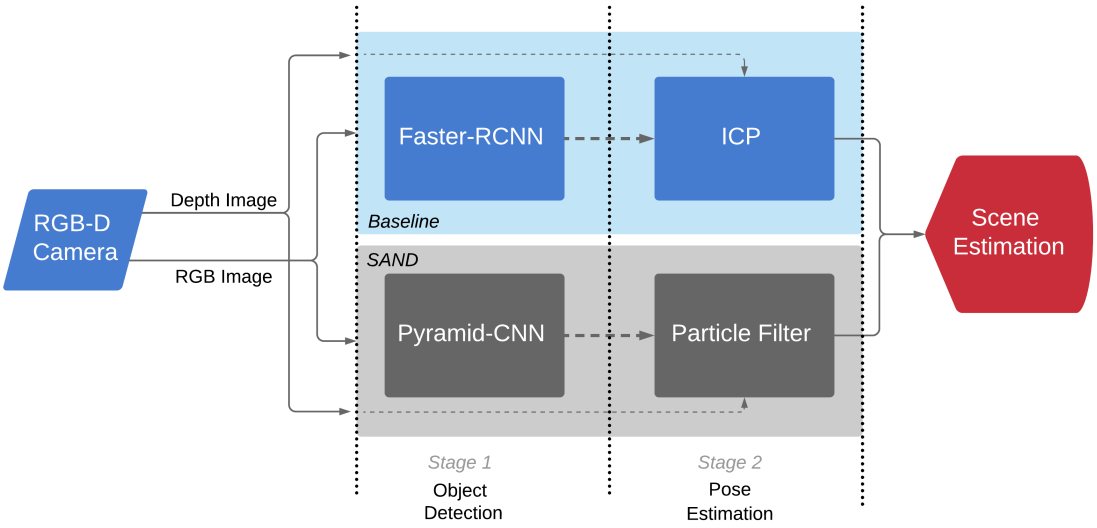



Figure 4.1: *Baseline* and *SAND* pipelines

For both approaches, different neural network architectures have been implemented in order to examine the implications of the depth, hence the energy consumption, on the final accuracy:

- AlexNet
- VGG11, VGG13, VGG16, VGG19
- ResNet18, ResNet34, ResNet50, ResNet101, ResNet152
- Mobilenet (only for *Baseline*)

In the end, a further first-stage based on a feature detector plus a linear classifier has been developed and described in section 4.8. This stage does not imply the use of neural networks, and require less computational power, but with a low accuracy.

## 4.1 Software

### 4.1.1 Pytorch

Pytorch [1] is an open source end-to-end deep learning platform, developed by Facebook's artificial-intelligence research group, that permit fast development for Machine Learning application.

The main advantages provided by this library are:

**PyTorch Tensors** : multidimensional arrays, similar to Numpy arrays, with the advantages that they can be elaborated on GPUs (thanks to CUDA supports). This permits to strongly reduce the development time. Moreover, Pytorch provides a full set of mathematical operation that may have Tensor as terms.

**Run on GPUs** : Pytorch has been developed in order to run on GPUs with CUDA support(section 4.1.2). To perform the computation on the GPU instead on CPU (set by default), it is enough to copy input data and network models on the GPU, using the function *.cuda()*. This function is available for all type of tensors and modules. Calling *.cuda()* within a module, Pytorch will take care of copying recursively all child modules into the GPU.

**Autograd module** PyTorch uses a technique called automatic differentiation to compute the gradient, the essential element in machine learning. It provide a recorder that records what operations are performed during the entire process, and then it replays it backward to compute final gradients. Therefor, it is not required to the programmer to compute the gradient descent by hand, saving a lot of effort. For CNN, backpropagation ( how to neural network calculate gradient descent) are also implemented in this module. Listing 4.1 reports the implementation code of backpropagation use in Pyramid-CNN in the *SAND* pipeline.

```
1          if not opts.is_test:
2              opts.optimizer.zero_grad()
3              loss.backward()
4              opts.optimizer.step()
```

Listing 4.1: Code for backpropagation pass for Pyramid-CNN

[1] https://pytorch.org/

The first line is used to differentiate the train phase from the test phase. In fact, during the test phase, the backpropagation must not be calculated. At the end of each epoch, gradients have to set to zero in order to avoid accumulation derived by backward process, so the *zero_grad()* function can be used. *loss.backward()* calculates the gradient, and *.step()* update the weights in the network.

**Optim module** provides optimization algorithms used for building neural networks, necessary also to specify how to perform loss function for backpropagation. Pytorch implementation code of backpropagation use in Pyramid-CNN has shown in Listing 4.2

```
1              net = alexnet(pretrained=True, ...
       num_classes=opts.num_cls, num_shape=opts.num_shape)
2          opts.optimizer = optim.SGD(
3              net.parameters(), lr=opts.lr, momentum=opts.momentum,
4              weight_decay=opts.w_decay)
5
6      if torch.cuda.is_available():
7          net.cuda()
8          net = torch.nn.DataParallel(
9              net, device_ids=range(torch.cuda.device_count()))
10
11     if opts.loss == 'ce':
12         opts.criterion = nn.CrossEntropyLoss()
13     elif opts.loss == 'bce':
14         opts.criterion = nn.BCELoss()
```

Listing 4.2: optimizer and loss function configuration for Pyramid-CNN

The first line is used to initialize the network, AlexNet in this case. After the initialization the stochastic gradient descent method has configured and the model is copied on th GPU. In the end, type of loss function has been selected. *CrossEntropyLoss* is used for multi-class classifier, while *BCEloss*(Binary Cross Entropy) is used for single class classification problem.

**nn module** The nn package contains a set of Modules, which are similar to neural network layers. Each Module receives input and computes corresponding output, holding internal learnable parameters if needed. Note that all inputs, outputs and parameters are simply Tensors. The nn package also provides a set of different loss functions, useful during neural networks training, in or-

der to understand how accurate is the network. Listing 4.3 shows AlexNet architecture build with *nn* module for Pyramid-CNN.

```
1          self.features = nn.Sequential(
2              nn.Conv2d(3, 64, kernel_size=11, stride=4, padding=2),
3              nn.ReLU(inplace=True),
4              nn.MaxPool2d(kernel_size=3, stride=2),
5              nn.Conv2d(64, 192, kernel_size=5, padding=2),
6              nn.ReLU(inplace=True),
7              nn.MaxPool2d(kernel_size=3, stride=2),
8              nn.Conv2d(192, 384, kernel_size=3, padding=1),
9              nn.ReLU(inplace=True),
10             nn.Conv2d(384, 256, kernel_size=3, padding=1),
11             nn.ReLU(inplace=True),
12             nn.Conv2d(256, 256, kernel_size=3, padding=1),
13             nn.ReLU(inplace=True),
14             nn.MaxPool2d(kernel_size=3, stride=2),
15         )
```

Listing 4.3: AlexNet Architecure Example

For example, the first layer of AlexNet is a convolution layer with input channel equals to 3, output channel equal to 64 and kernel size, stride and padding respectively 11,4,6. In Pytorch this layer is created with line 2 of Listing 4.3: *nn.Conv2d(3, 64, kernel_size=11, stride=4, padding=2).*

**Pretrained Model** Pytorch provides CNN model pretrained on ImageNet. A pretrained model is merely a list of saved internal parameters that was previously computed by training the network on a dataset. In particular, all Pytorch pretrained model has been pretrained on ImageNet. Usually, when a network is initialized for the first time, the weights (internal parameters) inside are random numbers. Instead of initializing the model with these random weights, initializing it using a pretrained weights reduces the number of steps necessary for the output to converge, therefore reducing the training time. This is true also if the two dataset (for the pretraining and for the training) contain different classes, because for classification task, features may be similar. Available pretrained models allow to save a lot of time because, for example, 14 days are needed in order to train ResNet-50 on ImageNet using an NVIDIA M40 GPU [58]. Code 4.4 shows how to load a model of AlexNet pretrained on ImageNet using Pytorch. Note that definition of AlexNet architecture (used in

line 8) is defined in different file, for instance Listing 4.3.

```
1  import torch.nn as nn
2  import torch.utils.model_zoo as model_zoo
3  model_urls = {
4      'alexnet': ...
       'https://download.pytorch.org/models/alexnet-owt-4df8aa71.pth',
5  }
6
7  def alexnet(pretrained=False, **kwargs):
8      model = AlexNet(**kwargs)
9      if pretrained:
10         pretrained_dict = model_zoo.load_url(model_urls['alexnet'])
11         model_dict = model.state_dict()
12         pretrained_dict = {k: v for k,v in pretrained_dict.items() ...
       if k in model_dict}
13         model_dict.update(pretrained_dict)
14         model.load_state_dict(model_dict)
15     return model
```

Listing 4.4: Example of how to use load pretrained weight when a netwrok is initialized

### 4.1.2 CUDA

In the early days in the history of machine learning, algorithms were executed in single processor environments, where bottlenecks can lead to substantial delays in model processing. The aim of deep learning is to speed up the convergence of model training using multiple nodes, so applying data parallelism is rather intuitive. In the recent years, GPUs had evolved into highly parallel multi-core systems allowing very efficient data parallelism. Hence Training time is highly reduced with GPUs. For instance, training on a GPU is more than ten times faster than the corresponding process on a CPU. In order to speed up deep learning computing, NVIDIA provides a parallel computing platform called CUDA®[2]. This platform permits the developer to direct the compiler to the portion of the application that maps to the GPU.

[2]https://developer.nvidia.com/cuda-zone

## 4.2 Hardware

In this work all the networks has been trained and tested on a desktop environment with a *NVIDIA Titan XP*, and Intel Xeon Processor E5. The graphic card is an high-performance GPU developed specifically for deep learning application with 38400 cuda core, that are responsible for dealing with all the data that moves through a GPU.

Moreover, all networks implemented with Faster-RCNN model are also tested on the embedded system-on-module NVIDIA Jetson TX2 after the training on the desktop environment equipped with the Titan XP. NVIDIA Jetson TX2 finds place many application field, from image stabilization to self-driving car. Running deep learning experiment on a desktop environment, such as the computer with the Titan XP, is clearly faster than the Jeston board, but the energy consumption is not comparable, due to the fact that Jetson only consume fifteen watts, while the Titan XP can reach 260 watts. In a embedded system or a mobile robot, energy consumption surely represents a very strong constraint.

The Jetson CPU is composed by a dual-core Nvidia Denver 2.0 and a quad-core ARM Cortex A57. The Jetson TX2 has three power modes:

**Max-Q** = maximum energy efficiency, about 7.5 Watts.

**Max-P** = higher frequency (and power consumption) respect Max-Q.

**Max-N** = full clock speeds, around 15 Watts.

Table 4.1 report for each power mode the configuration and the working frequency for each CPU and GPU. Obviously, the GPU in Max P (max performance) works with higher frequency, therefore CNNs will operate faster, sacrificing power consumption.

| Mode | Mode Name | Denver 2 | Freq | ARM A57 | Freq | GPU Freq |
|---|---|---|---|---|---|---|
| 0 | **Max-N** | 2 | 2.0 GHz | 4 | 2.0 GHz | **1.30 GHz** |
| 1 | **Max-Q** | 0 | | 4 | 1.2 GHz | **0.85 GHz** |
| 2 | **Max-P Core-All** | 2 | 1.4 GHz | 4 | 1.4 GHz | 1.12 GHz |
| 3 | Max-P ARM | 0 | | 4 | 2.0 GHz | **1.12 GHz** |
| 4 | Max-P Denver | 2 | 2.0 GHz | 0 | | 1.12 GHz |

Table 4.1: Hardware configuration of Nvidia Jeston TX2 operating mode

For the Jetson TX2, networks has been tested at 3 different GPU frequencies: 1.30GHz, 1.12GHz and 0.85 GHz. Table 4.2 reports the main differences between

the two systems. Titan XP has more CUDA core respect the embedded board, hence the latter will consume less power, with the trade-off of perform operation in more time. Due to this difference in the time performance, it is not useful to train all model on both system. Therefore all the networks has been trained on the Titan XP and loaded on the board afterwards in order to perform measures about runtime and energy consumption.

| | Titan XP | Jetson TX2 |
|---|---|---|
| No. Cuda Core | 3840 | 256 |
| Memory Size (GB) | 12 | 8 |
| Memory Bandwidth (GB/sec) | 547.7 | 59.7 |

Table 4.2: Comparison between NVIDIA Titan XP and Jetson TX2

## 4.3 Dataset

Unlike most of the latest researches on image classification [41] [23] [37] [38], that use well known datasets, such as the large Imagenet or COCO[3], that contains more than 1 million of image, in this work has been developed a own dataset in order to perform the real experiments. The dataset, called *progress*, is composed by 15 household object (Fig.4.2), plus the background class.


Figure 4.2: One image for each class in the *progress* dataset

About 200 images are taken for each class, except for the background that have

[3]http://cocodataset.org/

about one thousand sample, with a total of ≈4500 training images for the whole dataset, a small number compared to other datasets. For example MNIST database (Modified National Institute of Standards and Technology database) has 6000 images per class. On the other side, Imagenet have 1.2 million images for 1000 class of objects. Respect the dataset just mentioned, the *progress* dataset is obviously lighter in terms of memory required to be stored. Also the effort used by human to produce and annotate all images is much smaller. However this bring to a drawback: having a light dataset, during the training, the number of images could not allows the neural network to achieve the required convergence. This is obliviously more true for large networks. The use of pretrained model and data augmentation permits to overcome this problem. Pretraining is explained in section 4.1.1, while data augmentation consists of geometrically transform (e.g. flip,shift, scale,etc) original images and to add them in the dataset. Purpose of the project is to recognize object in a cluttered scene placed in a random manner. Hence, for each class in the dataset, sample images has been taken from different point of view and orientation. This permit to have a framework robust to the orientation of objects. In addition to the training images, which contain an instance of only one class each, there are also 60 complex image, thirty of which are used for *train & validate*, while the other thirty are used for the *test*. Figs 2.10 2.11 shows an example of test complex scene and the bounding boxes predicted with the object detector. Every complex scene contains:

- Up to one images for each class of the dataset. Not all of these object are completely visible from the camera: one purpose of this project is to create robust manipulation in cluttered human environment.
- Object that are not present in the dataset. This can create False Positive detection.

For each image (train or test, single object or complex scene) a text file has been created with the annotation on all the information useful to the training like the example 4.5, such as the label of the objects in the image, his location and orientation.

```
1 <object>
2       <name>ranch</name>
3       <pose>
4          <x>-1.59645</x>
5          <y>-2.63976</y>
6          <z>-0.010124</z>
7          <roll>0.764579</roll>
8          <pitch>0.114293</pitch>
9          <yaw>0.734804</yaw>
```

```
10          </pose>
11          <truncated>0</truncated>
12          <difficult>0</difficult>
13          <bndbox>
14              <xmin>180</xmin>
15              <ymin>103</ymin>
16              <xmax>331</xmax>
17              <ymax>198</ymax>
18          </bndbox>
19      </object>
```

Listing 4.5: Example of annotation file of one object in a complex scene

All annotations file has been created by scratch by human, requiring expensive human effort. Therefore, this aspect dictate the choice of use a small dataset.

## 4.4 Baseline

Faster-RCNN represents the stat-of-the-art object detection, therefore it is used in this work as first stage of the first pipeline approach. ICP, alike, is a well-known method involved to obtain 6D pose estimation. The union of these methods creates a two-stage pipeline applicable on the robot manipulation system. In this work, this approach is called *Baseline*. Considering the good level of performance reached by both in the literature, no further optimization has been described in this work for this approach. In fact, the purpose of this section is to study the relationship between cost of computation and accuracy of a neural network within robot perception application. CNNs with many layers have shown themselves to have best accuracy, with the drawback of more energy consumption. In autonomous mobile robotic, the energy represents a hard constraint during the development phase. Demonstrate that a good amount of energy consumption can be reach at the expense of a small loss of accuracy is useful to respect the constraint.

### 4.4.1 First Stage: Faster-RCNN

Faster R-CNN from this work has been developed from [57]. This code has been extended in order to work whit the *progress* dataset. Moreover, all network implementation has been developed to work with this framework. Such network are MobileNet, AlexNet, VGG(11,13,16,19), ResNet(18,34,50,101,152). This model is composed by the following high-level blocks:

- **Anchor Generation**: generates a fixed number of anchors of different scales and aspect ratios.

- **Proposal block**: Transform the anchors according to the bounding box regression coefficients to generate transformed anchors.
- **Anchor Target Layer**: produce a set of anchors and the corresponding class labels and target regression coefficients to train the Region Proposal Network.
- **RPN loss** compute the loss function for the RPN in order to train the RPN.
- **Proposal Target Block**: reduces the list of anchors produced by the proposal block and produce class specific bounding box regression used by the proposal block.
- **ROI Pooling**: perform the spatial pooling on the region proposed by the previous layer.
- **Classification** predicts the class probability for each region proposal.
- **Classification Loss**: compute the loss function for the classification block.

At the end of the process, the network is able to produce a set of bounding boxes for each object for each test scene. These bounding boxes are composed by the coordinates and the dimension of the 2d rectangle and the class score. High score correspond to high probability that the bounding box contains the object. These outputs are then passed to se second stage in order to perform the pose estimation. An example of a Faster-RCNN output can be see in the following box with the fomat $\{Ymin, Xmin, Ymax, Xmin, Score\}$:

```
39 286 161 390 0.878
26 215 340 402 0.084
97 290 278 373 0.016
94 129 220 359 0.015
101 85 198 196 0.014
472 584 479 639 0.007
39 0 140 37 0.002
62 573 224 639 0.002
120 227 240 307 0.002
```

**Training & Test**

After the initialization and the loading of the pretrained weights, the network is not yet able to preform the detection on the *progress* dataset, because it do not

"know" the objects contained in that dataset. Hence, a training phase is necessary. During the training, sample images are elaborated by the network and the predicted outputs are compared with the ground truth label, the loss function is derivate, and the weights updated. This procedure is done with the entire training dataset, for many times (epochs), lead to a convergence in the model, and it is able to perform in a right way. An epoch is a complete "view" of the dataset by the network. In order to train the network a python script has been developed. This script receive the following input:

- **Network Name**: chosen among all the model cited before. In the initialization phase, the correspondent pretrained weights are loaded.
- **Total epochs number**: the number of complete passes through the training dataset. Big number of epochs equivalent longer time for training. After ≈100 epochs accuracy value converges and no improvements has been shown. Thus, for this work, each networks has been trained for one hundred epochs. At the beginning of the training, in order to augment the data sample, each images has been flipped horizontally and added to the dataset, doubling its size. In each epoch, the order of elaboration of the images is created randomly, augmenting the non-linearity.
- **Validation epochs**: Every batch of epochs equal to this parameter, the network is tested with the thirty complex scenes of the validation set in order to calculate the accuracy. Every objects present in the validation scenes is also present in the training set of images. In this way the network should be able to recognize "in a easy way ", or in other words, with high value of probability, aforesaid object. In an ideal world, the accuracy of the validation phase should be 100%. At the end of validation, if the accuracy is greater than the previous value, the model is saved. Saving model every defined period of time permits also to stop the training in order to resume it in a different moment.
- **Batch size**: number of training samples (i.e. images) elaborated by the network consecutively before to update the weights with back propagation. Little batch size means updating the model frequently, costing more computationally and taking significantly longer to train models. Big batch size, instead, may result in premature convergence of the model to a less optimal set of parameters. Hence, popular batch sizes include 32,48(used in this work), 64, and 128 samples.

### 4.4.2 Second Stage: ICP

The purpose of second stage is to estimate object pose starting from a bounding box, composed by a rectangular region plus a label, provided by the first stage. Second stage extracts the point cloud delimited by the bounding box and through ICP it estimate the pose. ICP is a discriminative method that iteratively minimize the distance between two sets of points cloud. This distance is defined as the sum of Euclidean squared error between two point clouds. During each iteration, the algorithm selects two subsets of points using RANSAC [9] from their corresponding point clouds to calculate the distance. In this work, a implementation of ICP provided by Point Cloud Library (PCL)[4] has been used. After the distance between two point sets is lower than the convergence threshold, a rotation and a translation has been obtained. They represents the transformation to the sample image in the database final object pose. They will use successively to plan the motion actions of the robot.

As cited in section 2.3.1, ICP is sensitive to initial pose of the object, and in this work, the number of iterations has been fixed, in order to not fall in a infinite or very long loop. Some algorithm has been developed in order to obtain a quasi-initial pose of the object, like PCA (principal component analysis) [16]. However, due to the random nature of the disposition of the object in the scene, PCA has not lead improvements.

The bounding boxes predicted by the first-stage is used to crop the point cloud of the depth image with the RGB-D camera on the robot. The point cloud of the entire scene contains all the group of point of the interesting object, but also it contains point of other the background. Whatever it is accuracy of the bounding box, every cropped region of the depth image will have some noisy point. This noise can affect the performance of the ICP because the point cloud matching could be make using point that are not of the required object. A possible solution, that remove part of the noisy point, is the segmentation, as used in [54] where ICP is combined with a neural network-based object segmentation stage. As said in Section 2.5, this approach has not been used in this work, but surely it pose a next step in this research that can help to increase the accuracy. However, the ICP still does not have some of the advantages provided by SAND approach.

[4] http://pointclouds.org/

## 4.5 SAND

Perception is the main bottleneck to perform autonomous mobile manipulation tasks. This bottleneck also increase in cluttered environment, where only part of the target are visible and the pose estimation is not easy. In this section, a different two-stage paradigm, called *SAND* (*SA*mpling *N*etwork *D*ensity filter) [46], for object detection and 6D pose estimation is described. This approach performs the scene estimation through a end-to-end CNN detection stage plus a generative pose estimation method based on sampling-based local search. Differently for ICP used in *Baseline*, this work demonstrate that the *SAND* can recover some hard-decisions made by neural network stage, improving the robustness of the framework, also against adversarial attack.

### 4.5.1 First Stage: Pyramid-CNN

The first stage of the SAND method (i.e. the object detector) is a simple CNN network classifier mutated in a detector with a sliding window stage applied on the input image. Every single input image (or tested image), has been elaborated by the CNN at 5 different level of scale (0.75, 1.0, 1.5, 2.0, 2.5). While the Faster-RCNN produces a set of bounding boxes for each object in the scene, the output of the Pyramid-CNN is instead a heatmap. For each level of scale, each pixel that belong to a region created during the sliding window, after be elaborated by the CNN, hold a value. This value it is the probability of the pixels to fall in a certain class. The value of the pixels in the same region, due to not overlapped sliding window, is the same. For each input image, for each object and for each level of scale, the pyramid CNN produces a different heatmap, as shown by figure 4.3.

Here a file representation of part of a heatmap for one object({ *Ymin Xmin Ymax Xmax Score*}:

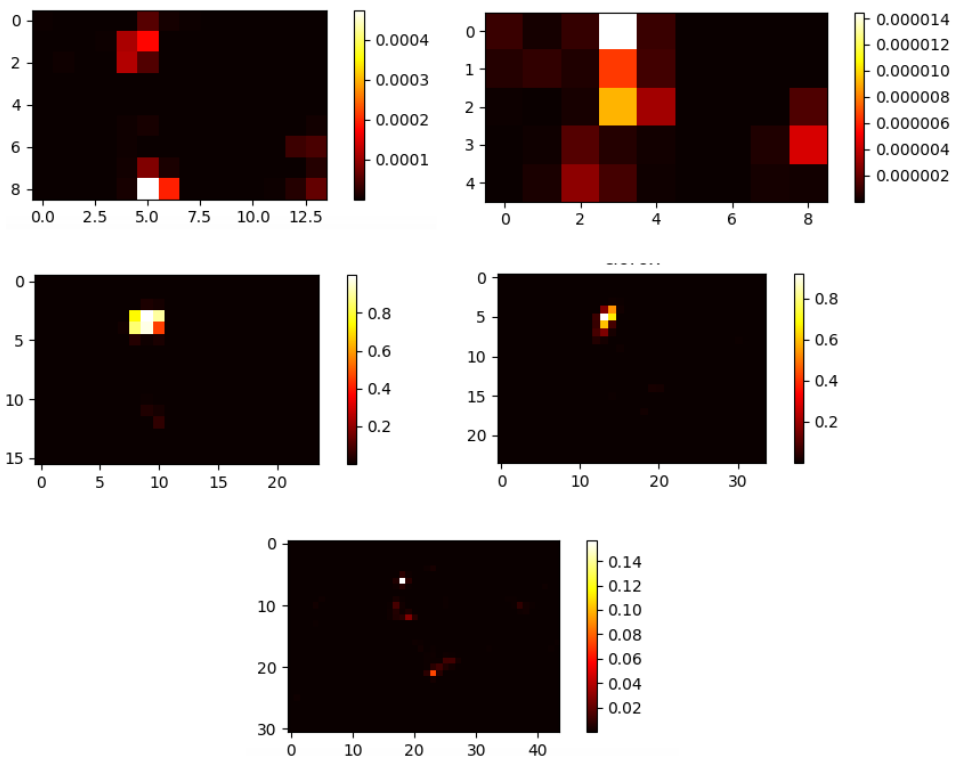


Figure 4.3: Example of an Heatmap created by Pyramid-CNN

```
...
21 21 319 319 0.000000
21 64 319 362 0.000000
21 107 319 405 0.000000
21 149 319 447 0.000000
21 192 319 490 0.000000
21 235 319 533 0.000000
21 277 319 575 0.238473
21 320 319 618 0.000001
21 363 319 640 0.000009
...
```

Figure 4.4 shows Pyramid-CNN prediction with *clorox* class score less than 0.7. A thresholding as used for visualization because the number of predictions is too big to be clear visualized in the same image.

Moreover, the Pyramid-CNN architecture, is composed by two different branches, as shown in Fig. 4.5. The first is the classifier, the ordinary branch which is found in all the CNN, that classify images. The other branch, instead, is used to discovered,

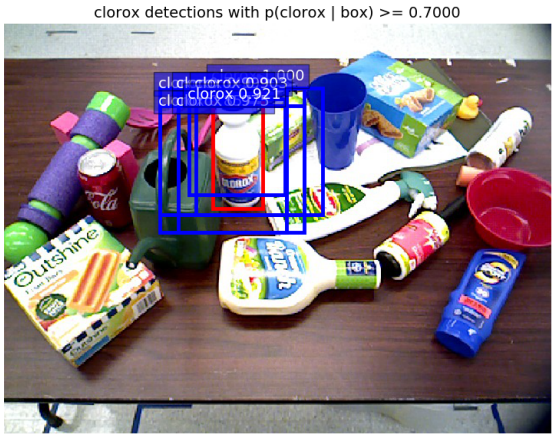



Figure 4.4: Pyramid-CNN prediction Example. Red bounding box is the ground truth

not the class of the object, but the shape of that object (i.e the orientation in the space).

Fig. 4.5 [46] shows the architecture of Pyramid-CNN that implements the VGG16 layers. Note that the image shows only the VGG16 architecture, but the additional branch has been implemented in all the networks used for the study. Python code of this architecture is defined in listing 4.6 where the two branches are *self.classifier1* and *self.shape*.

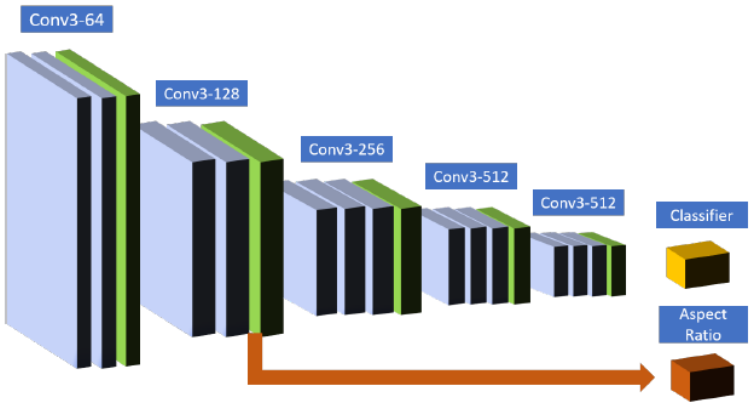



Figure 4.5: Pyramid-CNN architecture

```
1  class VGG(nn.Module):
2
3      def __init__(self, features, num_classes=16, num_shape=7):
4          super(VGG, self).__init__()
5          self.features = features
```

```
6           self.classifier1 = nn.Sequential(
7               nn.Conv2d(512, 4096, kernel_size=7),
8               nn.ReLU(inplace=True),
9               nn.Dropout(),
10              nn.Conv2d(4096, 4096, kernel_size=1),
11              nn.ReLU(inplace=True),
12              nn.Dropout(),
13              nn.Conv2d(4096, num_classes, kernel_size=1),
14          )
15          self.shape = nn.Sequential(
16              nn.Conv2d(512, 4096, kernel_size=7),
17              nn.ReLU(inplace=True),
18              nn.Dropout(),
19              nn.Conv2d(4096, 4096, kernel_size=1),
20              nn.ReLU(inplace=True),
21              nn.Dropout(),
22              nn.Conv2d(4096, num_shape, kernel_size=1),
23          )
24          self._initialize_weights()
25
26      def forward(self, x):
27          feat = self.features(x)
28
29          class_label = self.classifier1(feat)
30          shape_label = self.shape(feat)
31          return [class_label, shape_label]
32  cfg = {
33      'A': [64, 'M', 128, 'M', 256, 256, 'M', 512, 512, 'M', 512, 512, 'M'],
34      'B': [64, 64, 'M', 128, 128, 'M', 256, 256, 'M', 512, 512, 'M', 512, 512, 'M'],
35      'D': [64, 64, 'M', 128, 128, 'M', 256, 256, 256, 'M', 512, 512, 512, 'M', 512, 512, ...
        512, 'M'],
36      'E': [64, 64, 'M', 128, 128, 'M', 256, 256, 256, 256, 'M', 512, 512, 512, 512, 'M', ...
        512, 512, 512, 512, 'M'],
37  }
38  def vgg16(pretrained=False, **kwargs):
39      """VGG 16-layer model (configuration "D")"""
40      model = VGG(make_layers(cfg['D']), **kwargs)
41      if pretrained:
42          pretrained_dict = model_zoo.load_url(model_urls['vgg16'])
43          model_dict = model.state_dict()
44          pretrained_dict = {k: v for k,v in pretrained_dict.items() if k in model_dict}
45          model_dict.update(pretrained_dict)
46          model.load_state_dict(model_dict)
47      return model
```

Listing 4.6: Implementation of Pyramid-CNN with VGG16

Respect the Faster-RCNN, this first stage produce more false positive. This number is not high in Faster-RCNN due to the involvement of RPN, that removes region that are considered not interesting, performing a thresholding. The thresholding has been avoided in the *SAND* pipeline in order to not rely solely on the decision produced by first stage.

### Offline Hard Negative Mining

During the training phase of a classifier, both positive and negative examples are used. Positive example are images of the object that must be classified (e.g. the battle of *clorox*), while the negative example are images that does not contain the aforesaid object (for example background images or images with other object). Also with these two type of images, the classifier could still predict falsely the ob-

ject. With Offline Hard Negative Mining(OHNM), also called Offline Hard Example Mining, these false prediction are added to the train set and labeled as negative example (i.e. they do not contain the object). Once these images have been added, the classifier model have been retrained with the new dataset, increasing the accuracy. However, increasing of number of images in the dataset lead to longer training time.

Pyramid-CNN alone, produces many false positives, decreasing the accuracy, due to the big number of input image(created by the sliding window). In order to reduce this number, the OHNM has been applied twice during the training of each CNN model. Table 4.3 shows the mAP before and after the hard negative mining. As it is possible to see the OHNM brings growth for all network models, therefore reducing the number of false positive. To compare the two approaches(*Baseline & SAND*) in a fairly way the same method has been applied also on the Faster-RCNN detector, but the OHNM does not provide improvements, as shown in the Table 4.4. There is no accuracy gain by reason of the accuracy of the Faster-RCNN is already very high, this involves that the number of false positive is very small, such as the falsely detected object. For instance, the number of image added with OHNM to the dataset of a Pyramid-CNN network is about$\approx$ 300, while for the same network implemented through the Faster-RCNN is about $\approx$ 50. Consequently the latter number, added in the *progress* database can not bring sensible improvements. Finally, considering that the OHNM at least double up the training time (one re-training with more images), this method has not been used in the *Baseline*.

| **Network** | **mAP before OHNM** | **mAP after OHNM** |
|---|---|---|
| AlexNet | 0.301 | 0.327 |
| VGG11 | 0.219 | 0.246 |
| VGG13 | 0.223 | 0.279 |
| VGG16 | 0.216 | 0.296 |
| VGG19 | 0.212 | 0.247 |
| ResNet18 | 0.129 | 0.225 |
| ResNet34 | 0.159 | 0.188 |
| ResNet50 | 0.173 | 0.242 |
| ResNet101 | 0.157 | 0.236 |
| ResNet152 | 0.137 | 0.217 |

Table 4.3: OHNM applied on Pyramid-CNN

| **Network** | **mAP before OHNM** | **mAP after OHNM** |
|---|---|---|
| Alexnet | 0.876 | 0.848 |
| VGG11 | 0.877 | 0.892 |
| VGG13 | 0.914 | 0.928 |
| VGG16 | 0.944 | 0.930 |
| VGG19 | 0.896 | 0.907 |
| ResNet18 | 0.903 | 0.901 |
| ResNet34 | 0.913 | 0.910 |
| ResNet50 | 0.907 | 0.897 |
| ResNet101 | 0.902 | 0.921 |
| ResNet152 | 0.924 | 0.926 |

Table 4.4: OHNM applied on Faster-RCNN

### Training & Test

As referring to section 4.4.1, a similar training procedure is used for Pyramid-CNN, with the addition of OHNM. In order to apply the OHNM the loss function and the accuracy are calculated and monitored with Tensorboard. When accuracy reached value higher than 99%, the training phase was stopped and OHMN applied. Note that the accuracy is calculated on the training data. This procedure has been applied twice for each network, resulting in three training phase. Each complete training phase lasted ≈2 hours for small networks (like AlexNet) and ≈4 hours for the very deep ones (like VGG16, VGG19). Figure 4.6 show the flowchart of a complete training procedure for Pyramid-CNN.

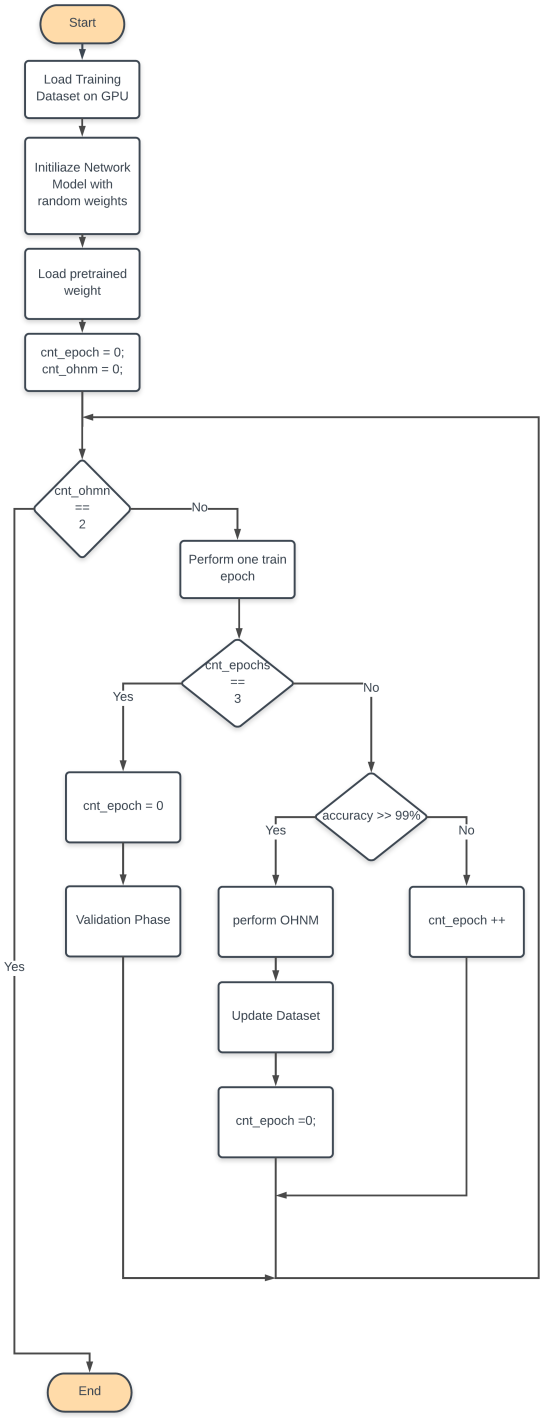



Figure 4.6: Training algorithm for Pyramid-CNN

### 4.5.2 Second Stage: Particle Filter

The output created by Pyramid-CNN, a pyramid of heatmaps, are used as proposals by the second stage, called *Particle filter*. Each pixel in the heatmap represents a bounding box with a categorical distribution of all object classes and each heatmap corresponds with a fixed shape of bounding box, defined by the shape branch of the Pyramid-CNN. The generative sampling-based search in the second stage of the SAND is inspired by sampling methods. In order to estimate the pose of one object predicted by the first stage, the Particle filter executes this procedure:

1. Started from the label of the object a a collection of samples are initialized. Each sample represent an hypotheses or the final pose and it contains a weight score. The weight is calculate combining the class score predicted from the first stage and the score of the geometric correspondence with the point cloud obtained by cropping the scene through the bounding box of the first stage (i.e. the detecte region that should contain the object)

2. Evaluation of the weights of each new sample obtained by the previous phase.

3. Resamping and Diffusion phase. After the weight of each sample has been evaluated, a resampling and diffusion process is performed over the collection of samples. Resampling consists of update the weight of the sample with last computed weight. Diffusion, instead, consists in appling a translation or a rotation or a mix of both in random way to the point cloud sample. Each sample in the collection receives a different type of diffusion.

4. The process of evaluation, resampling, diffusion repeats until convergence of the object pose.

5. In final iteration, the sample with higher score is selected and a rotation and a translation has been obtained. They represents the transformation to the sample image in the database final object pose.

Since a wrong prediction made by the first stage will lead to a not correct manipulation by the robot, these method provides robustness over the search space, permitting to recover some imperfect first stage output. Table 4.5 resumes the main difference between the two approaches: *Baseline* and *SAND*.

| | **Baseline** | **SAND** |
|---|---|---|
| *output first stage* | bounding boxes | heatmaps |
| *Thresholding* | yes | no (sand uses false positive to generate more hypothesis) |
| *OHNM* | no (mAP does not change) | yes (help to remove some false positive) |

Table 4.5: Differences between *Baseline* & *SAND*

## 4.6 Adversarial attack

Many researches [49][21] [2][52] has shown the weakness against adversarial attacks or misleading examples. Defending against adversarial examples is not a trivial problem because it requires neural network models to produce good outputs for every possible input; however, in a physical scenario neural networks must work well only on a very small number of all the many possible inputs. In this work, in order to deal with adversarial scenarios, it has been chosen to not modify the initial training set, and to exploit the robustness provided by the particle filer. It has been assumed that the attacker has not the possibility to tamper the robot or modify the image after it has been taken by the camera. He has only the possibility of modify the scene environment in three ways:

- Changing the surface aspect of an object
- Increasing the clutter in the scene introducing new object, hiding parts of some object
- Changing the light conditions

## 4.7 Robot Grasping

As obtained the pose estimation of an object, the robot can plane its motion and perform the grasping task. In this work, the development motion/grasping is not taken in account, assuming that correct pose estimation leads a correct motion and grasping. This assumption has been posit through different physical experiments. For the *Baseline* experiments has been run 10 trails of the robot grasping task for both VGG11 and VGG16. For both approaches it has been achieved 6/10 success rate. For the *SAND approach*, instead, the robot grasping has been performed during the analysis with adversarial attack. In the end, both second stages are not

able to understand if the prediction provided by the second stage is correct or wrong. In particular, if the first stage labels a region with a certain class, the second stage will provided anyhow the best pose estimation of that object in that region, even if the object is not present in that region. Future works will include more detailed solution for this open problem.

## 4.8 Feature-Based Approach

As mentioned before, CNNs depict the state-of-art object detection algorithm. However, the computational complexity required by neural network can not be provided by all the system. In particular, in mobile applications, not always there are unlimited resource (or even just a GPU) as they can be found in a desktop environment. One possible solution is to use neural network developed for this type of task, such as MobileNet. MobileNet reduce the number of layer in order to reduce the energy consumption, at the expense of the accuracy. One other possible solution is not use neural network model-based approach, as BoVW described in section 4.8. For this work, the *vl_feat*[5] library has been choose (MATLAB version). This library provides all the component required to perform the Bag Of Word model.

- **features extraction** through dense multi-scale SIFT descriptors(*vl_dsift*) . The main advantage of using vl_dsift over vl_sift is speed. DenseSIFT is more fast (about x30 speedup), moreover, the type of feature descriptors used by DenseSIFT is usually used for object categorization. VLFeat provides examples to how set hyperameters of DenseSIFT to perform as SIFT.
- **Elkan k-means** for fast visual word dictionary construction. Elkan's algorithm is a variation of k-means algorithm that uses the triangular inequality to avoid many distance calculations when assigning points to clusters, making the codebook construction faster than the main apporch.
- SVM classifiers

### 4.8.1 First Stage: SIFT

The main steps of the SIFT-based first stage follow the flows of the original BoW:

1. **Feature Extraction** by DenseSIFT on the train set of images. DenseSIFT computes descriptors for densely sampled keypoints with identical size. The

[5] http://www.vlfeat.org/

same training dataset of Pyramid-CNN has been use: 15 classes plus the background class. Respect the Pyramid-CNN, for this approach the number of background images has been increased by ten times to reduce part of wrong prediction.

2. **Dictionary Definition** Constructed using k-means (provided by VL_Feat) on the features extracted by the DenseSIFT.

3. **Train** SVMs. One SVM is created for each class of the fifteen objects of the dataset, plus one for the background.

4. **Sliding Window** every test image is divided in sub-images exploiting the Sliding Window method. Many different size of the *window* has been tested. Too small window are not able to detect objects, but using a too large one increases the risk that in the same examined region there are more objects, making the detection not easy. Big windows size lead also to increase more background, hence more noise in every sub-image. At the end, five different window's size has been chose: 90,120,150,180,200 (side measure in pixels). Bin sliding window size(step in pixel of the window respect to the x and y axes) is equal to 32 pixels. 64 pixels have proven themselves to be too big for this work, because a entire object could contained in about 64 pixels, making the classification not trivial. Bin size of 16 pixels, instead, makes runtime too long, without significant increases in accuracy. Moreover, the number of false positive increase with the number of sub-images, hence using small window size or small bin can affect badly the final accuracy.

5. **Test**: once obtained the weight from the SVM, each sub-images is testes with each SVM and a set of probability for each class is created for each sub-images.

6. **Prune False Positive**: as for the Pyramid-CNN, each region created by the sliding window must have a predicted label. Since the detector has not high accuracy, the number of False Positives has been resulted to be high, misleading the second stage. In order to reduce FP, it has been removed all regions with a certain label, which have not at least a "neighboring" region with the same label. This approach reduces both FP and TP, but with a reducing radio more higher for FP.

### 4.8.2 Second Stages with SIFT

The previous first-stages (Faster-RCNN and Pyramid-CNN) has been studied in order to produce a right output format for the corresponding second stage. In fact, Faster-RCNN bounding boxes are not usable within the *SAND approach* because the latter requires an heatmap as input. In the same time, the heatmap created by the Pyramid-CNN are not operable by ICP, that would use only the region with higher probability. The SIFT approach, instead, has been designed in order to work with both second-stages. Considering that the sliding window (in the SIFT), are applied with different size, it is possible to produce heatmaps for the particle filter. Merging all the region for every size dimension, with the same label, a final bounding box for ICP can be obtained.

# Chapter 5

# Experimental Results

## 5.1 Results Representation

In order to have a comprehensive view of the results from the first stage, a python algorithm has been developed to calculate the **mAP** of each network using the predicted bounding boxes and the ground truth annotations for the thirty test scenes, using the metric explain in section 2.4. Here, an example of the results, after a test phase:

```
Eval: downy, AP: 0.272933646016
Eval: toy, AP: 0.145952879464
Eval: blue_cup, AP: 0.282473296347
Eval: coke, AP: 0.373721340388
Eval: ranch, AP: 0.384120088527
Eval: spray_bottle, AP: 0.114080893394
Eval: sugar, AP: 0.233317730962
Eval: tide, AP: 0.362564009566
Eval: detergent, AP: 0.405121223561
Eval: clorox, AP: 0.164186729837
Eval: scotch_brite, AP: 0.0969596140052
Eval: red_bowl, AP: 0.512405611909
Eval: waterpot, AP: 0.396016637352
Eval: sunscreen, AP: 0.242525998981
Eval: salt, AP: 0.394781780116
mAP: 0.292077432028
```

For the second stage, after the test phase, a curve of accuracy is provided for each object, figure 5.1, as the average curve on the entire test set, figure 5.2. On the

Y axis is reported the accuracy, while on X axis is reported the Distance Threshold.

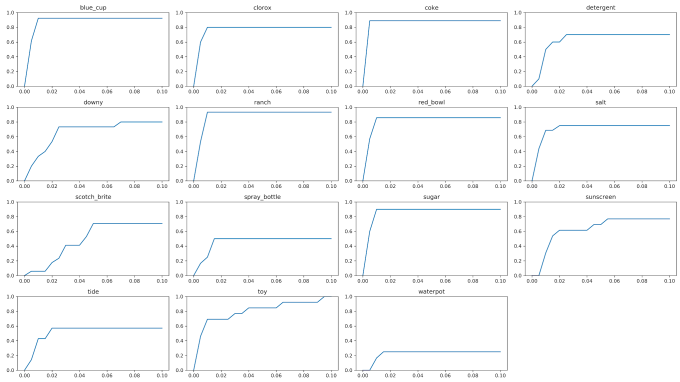



Figure 5.1: Accuracy curve for each object in the test scene. This graphs are the results of VGG16 within the *SAND* pipeline

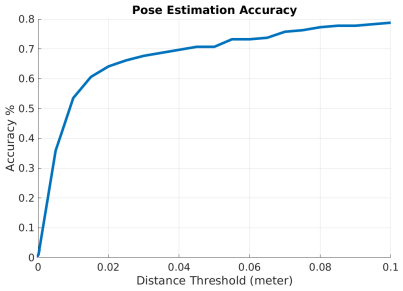



Figure 5.2: Average curve of all object in the test scenes. This graphs are the results of VGG16 within the *SAND* pipeline

## 5.2 Computational Complexity

Table 5.1 shows a comparison of computation complexity of the Faster-RCNN framework implemented with various CNN architectures. For time reason, this comparison has not been extracted for Pyramid CNN, but the consumption trend is reasonably the same. MACC (multiply and accumulate operations),activation (size of feature map) and weight parameter has been estimated from NetScope[1]. Number of MACC operations represents the index of computation complexity of the network. Number of weight parameters and the size of the intermediate feature map are used to determine if the model can be loaded on an embedded platform. For instance, the Jetson TX2 has 8GB of memory, so it is able to handle all the CNN models used in for this work. Referring to the table, adding more convolutional layers has a much bigger impact on total number of MACC operations and memory activation than the number of weight parameters. Therefore depth of the network impacts total energy consumption more than number of weights.

[1]https://dgschwend.github.io/netscope/quickstart.html

| | AlexNet | MobileNet | VGG11 | VGG13 | VGG16 | VGG19 | ResNet18 | ResNet34 | ResNet50 | ResNet101 | ResNet152 |
|---|---|---|---|---|---|---|---|---|---|---|---|
| # conv layers | 5 | 27 | 8 | 10 | 13 | 16 | 17 | 33 | 49 | 100 | 151 |
| filter size | 3,5,11 | 1,3 | 3 | 3 | 3 | 3 | 3,7 | 3,7 | 1,3,7 | 1,3,7 | 1,3,7 |
| # channels | 1-256 | 3-1024 | 3-512 | 3-512 | 3-512 | 3-512 | 3-512 | 3-512 | 3-2048 | 3-2048 | 3-2048 |
| # filters | 96-384 | 32-1024 | 64-512 | 64-512 | 64-512 | 64-512 | 64-512 | 64-512 | 64-2048 | 64-2048 | 64-2048 |
| MACCs (billions) | 107 | 170 | 799 | 1160 | 1560 | 1950 | 56 | 94 | 111 | 147 | 183 |
| activations (billions) | 0.18 | 0.27 | 1.60 | 2.52 | 2.77 | 3.02 | 0.16 | 0.24 | 0.68 | 0.91 | 1.19 |
| parameters (billions) | 0.058 | 0.005 | 0.131 | 0.132 | 0.137 | 0.142 | 0.012 | 0.022 | 0.028 | 0.047 | 0.060 |

Table 5.1: Computation complexity for Faster-RCNN with various CNN model

## 5.3 Analysis of Baseline

Table 5.2 reports the accuracy achieved by the object detector implemented with various CNN models from either first stages. More complex networks do not necessarily generate better results than smaller networks, even though more complex networks can produce less prediction error on ImageNet [41]. For example, in this work, VGG13 has higher mAP accuracy respect the deeper VGG19. Similarly, in the ResNet family, the 50-layers network has better mAP than the 101-layers one. This result is due in part to the limited size of our training dataset *progess*. Compared to ImageNet, that has also categories of objects completely different from each other, *progress* dataset is very small. However, small dataset does not necessarily indicates that a small network should always choose.

Figure 5.4 shows the accuracy vs. distance threshold results for the pose estimation produced by *baseline*. The table 5.4 shown the same number of the graph using a distance threshold of 0.1m. Direct comparisons between pose estimation accuracy values and object detection accuracy values are not meaningful since these values are derived in different ways. With this analysis, it has been demonstrated that lower first stage accuracy does not necessarily lead to lower second stage accuracy. For example, VGG11 and VGG16 have different mAP in the first stage, but they achieve the same pose estimation accuracy after ICP stage. These results demonstrate that the object detection accuracy should not be the sole predictor of accuracy in the pose estimation stage. Using smaller CNNs for detection can generate comparable result as using bigger networks.

### 5.3.1 Mobilenet

There are a few works that present CNN designs specifically for embedded hardware applications. In this work, in the *Baseline* approach one of these network has

| **Network** | **Faster-RCNN(mAP)** (baseline) | **Pyramid-CNN(mAP)** (SAND filter) |
|---|---|---|
| AlexNet | 0.860 | **0.327** |
| VGG11 | 0.885 | 0.246 |
| VGG13 | 0.915 | 0.279 |
| VGG16 | **0.926** | 0.296 |
| VGG19 | 0.896 | 0.247 |
| ResNet18 | 0.903 | 0.225 |
| ResNet34 | 0.914 | 0.188 |
| ResNet50 | 0.919 | 0.242 |
| ResNet101 | 0.897 | 0.236 |
| ResNet152 | 0.924 | 0.217 |

Table 5.2: Object detection (first stage) accuracy among different networks with progress dataset

been implemented: MobileNet [19]. In MobileNet, as shown in table **??**, the number of parameters is significantly lower than the other networks. For instance, it has ≈500 thousand of parameters, while the second network in the list that has fewer parameters is ResNet18 with ≈1200 thousand. However, even consuming 5 times less than very deep network (VGG16,VGG19),does yield low detection accuracy on our dataset(table 5.3), A possible cause could be that MobileNet has many convolutional layers so it requires more training data to reach the convergence. Moreover, in deeper network, the problem with gradient vanishing becomes more significant. For MobileNet, which has much worse detection accuracy than the other networks, it achieves pose estimation accuracy that is closer to the other networks (though it still remains the worst performing), figure 5.3.

| **Network** | **Faster-RCNN** (mAP) | **ICP Acc** (%) |
|---|---|---|
| MobileNet | 0.307 | 0.530 |

Table 5.3: *Baseline* implemented with MobileNet

## 5.4 Baseline VS SAND

Within the SAND pipeline, as reported in Table 5.2, performance of Pyramid-CNN as object detector are much lower than the Faster-RCNN, because the former

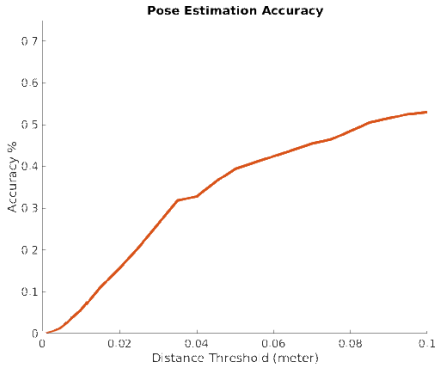



Figure 5.3: Accuracy of MobileNet (*Baseline*)

does not perform hard thresholding for the object detection stage, pruning many false positive. Hence, Pyramid-CNN generates more detection output that lead to more false positives. In order to reduce part of these number, OHNM has been used, has mentioned in section 4.5.1. However, even if number of false positive prediction still high than Faster-RCNN, the SAND approach is able to explore and search more thoroughly in order to correct false detections.

Referring to the last columns of Table 5.2 and Table 5.4, the results evince that for pyramid-CNN a higher first stage accuracy can help to reach a higher second stage accuracy, but it is not always necessary. For example, VGG19 and VGG11 have lower first stage accuracy than VGG16, but they have same or better second stage accuracy, endorsing the statement hypothesized in the previous section: lower first stage accuracy does not necessarily mean that this will lead to lower second stage accuracy. Figure 5.6 contains the comparison between the pose estimation accuracy of *Baseline* and *SAND* approaches using AlexNet, VGG16, and ResNet50 in the object detection stage. In general, *SAND* is better than *Baseline*, even if for the 0.1 meter threshold, both approach reaches comparable results. But within a tighter distance threshold, e.g. 0.02 meter, *SAND* pipeline is consistently and significantly better than the *Baseline* approach.

## 5.5 Adversarial Example

In order to show how the SAND approach can perform well with adversarial example, all the three types of attack simulations (changing the lighting condition, improving the amount of occlusion, and the aspect of an object) has been applied on the same basic image (fig 5.7(a)). Either approaches, *Baseline* and *SAND*, have been executed on the different modified images with the final purpose to complete

| **Network** | **ICP Acc(%)** (baseline) | **Particle Filtering Acc(%)** (SAND filter) |
|---|---|---|
| AlexNet | 0.662 | 0.788 |
| VGG11 | 0.707 | 0.742 |
| VGG13 | 0.687 | 0.753 |
| VGG16 | 0.702 | 0.758 |
| VGG19 | 0.707 | 0.783 |
| ResNet18 | 0.697 | 0.727 |
| ResNet34 | 0.697 | 0.646 |
| ResNet50 | 0.692 | 0.727 |
| ResNet101 | 0.697 | 0.712 |
| ResNet152 | 0.697 | 0.682 |

Table 5.4: Pose estimation (second stage) accuracy among different networks using a distance threshold of $0.1m$.

a correct manipulation on the can of *coke*.

Fig 5.7, is a resume of the whole experiment. First column shows the input scenes. In the second and in the third, are respectively presented the detections predicted by the Faster-RCNN approach and after the two-stage SAND. In last column there are finally the demonstration of the correct robot grasp performed after the right prediction of the SAND. With the basic scene (i.e. without modification), both approaches are able to identify correctly all the four objects on the table (coke, ranch, clorox and downy). Other objects on the table are not in the train set, therefore they are not recognizable. Both the occlusion and the change of aspect, that are pretty similar modifications, have the same outcome, all the objects are detected by the SAND, while the Faster-RCNN does not detect the coke. Finally, it tuned out that the scene with different lighting condition contains the modification that deceives more both approaches. While *SAND* still detect well the coke, but the ranch is not detected, the Faster-RCNN struggles to identify all the objects: only one bounding box is present, but the prediction is wrong. In fact, it predicts *waterpot* instead of *downy*. This last attack states that CNN are weak against non ideal light conditions. If the first stage does not detect the object, the grasping is not possible, hence only the grasping after SAND has shown.

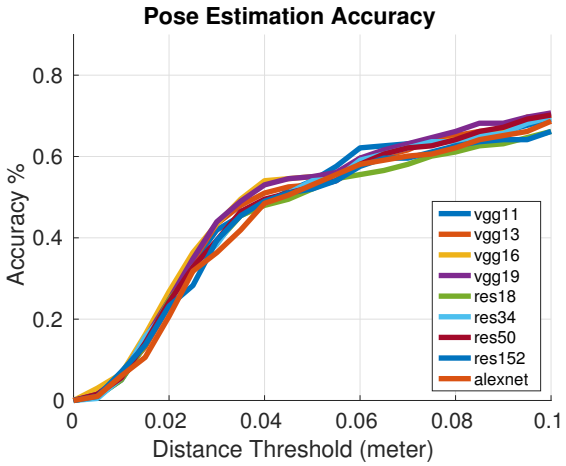



Figure 5.4: The final pose estimation accuracy of the *baseline* approach composed of Faster-RCNN and ICP. The y-axis is the accuracy percentage based on different distance threshold on x-axis.

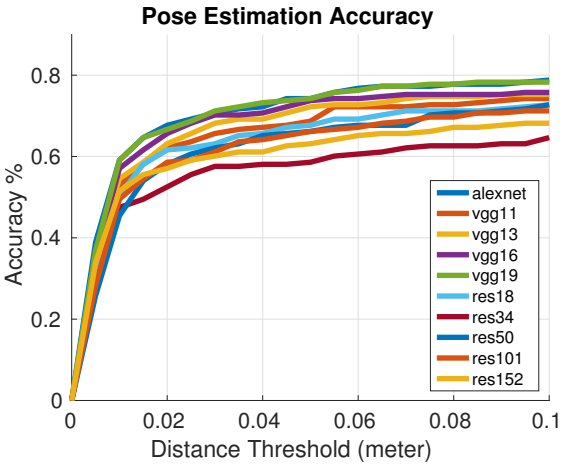



Figure 5.5: The final pose estimation accuracy of the *SAND filter* approach composed of Pyramid-CNN and particle filtering. The y-axis is the accuracy percentage based on different distance threshold on x-axis.

## 5.6 Power Analysis

In order to estimate the total energy consumption for every CNN model, the following equation has been used:

$$E \approx \sum_{n=0}^{N} p_n * \Delta t,$$

where $N$ is the total number of samples taken during the object detection stage.

NVIDIA provides a profiling tool to analyze its GPU characteristics (power, temperature, ecc...) during the utilization: **nvprof**. For more details refer to: Nvidia Website[2]. For the Titan Xp, the power $p_t$ has been read using the nvprof tool sampled at $\Delta t \approx 50$ms. For the Jetson board, instead, this tool is not available, therefore the power dissipation $p_t$ has been read through an on-board INA3221 power monitor every $\Delta t \approx 5$ms. This difference of $\Delta t$ is due to the fact that nvprof does not permit to change the sample time, and in the Jeston TX2 $\Delta t$ has been selected in order to achieve better results without affect the power.In fact, higher reading frequency lead to an increase of time necessary to the processor to perform the readings, increasing the total runtime, invalidating the measurements. Figure 5.8

[2]https://docs.nvidia.com/cuda/profiler-users-guide/index.html

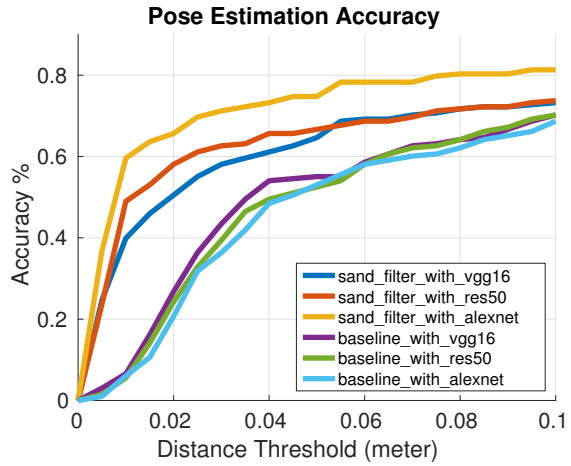



Figure 5.6: Pose accuracy of the *SAND filter* approach comparing with *baseline* approach.

shows the CNNs power consumption respectively for the Nvidia Jetson TX2 and the Nvidia Titan XP. Besides the power consumption, also the runtime has been measured on both systems. Either time and power are measured during a process of one image by the detection phase, where the network model is already loaded in the memory and ready start.

Due to lack of time, power and time measurement has been done only for Faster-RCNN, but the consumption trend can be considered reasonably the same also for Pyramid-CNN.

## 5.7 Mention on SIFT

The SIFT approach has been tested initially as classification task on the *progress* dataset. Every objects in test scene has been cropped using the ground truth information and elaborated by SIFT first stage. In the classification task is reached 95,5%. After the classification test, with the Sliding Window algorithm the SIFT approach has been test on the entire image, with five different windows size with the same bin size. In the table 5.5 are reported the mAP for each size, and the accuracy after the ICP stage is shown in figure 5.10.

The various size has been then used together in order to create the heatmap for the particle filter-based second stage, where it reaches the accuracy of 0.519, as is shown by figure 5.11.

The object detector based on BoW is not able to reach the same performance of a CNN-based detector. Also for robotic tasks, where the accuracy of first stage it does

(a) Basic Scene (b) Faster-RCNN Detection (c) Two-Stage Detection (d) Grasping

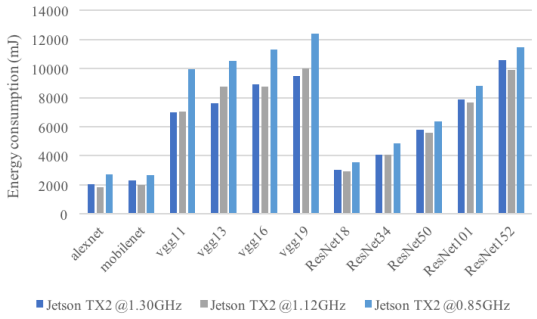
(e) Dark Scene (f) Faster-RCNN Detection (g) Two-Stage Detection (h) Grasping

(i) Cluttered Scene (j) Faster-RCNN Detection (k) Two-Stage Detection (l) Grasping

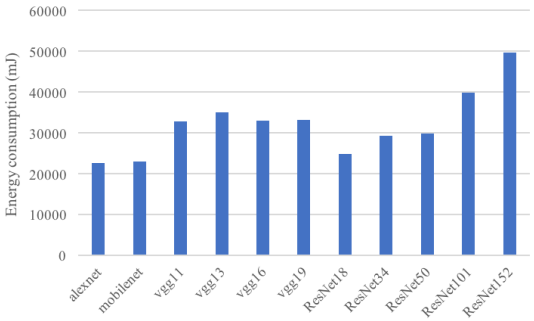
(m) Surface-Change (n) Faster-RCNN Detection (o) Two-Stage Detection (p) Grasping

Figure 5.7: Full process - from detection to complete grasp - with adversarial example. The robot task is to pick only the can of Coke.

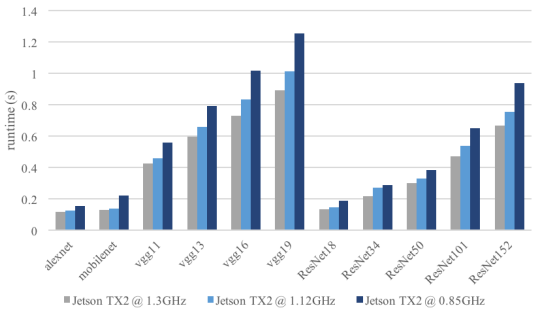
not have to be high, as demonstrated in the CNN analysis the non-learning approach is not sufficient. However, SIFT approach does not need to have large computational resources to process the images, and the training time (feature extraction plus SVM training) is much more faster then CNN approach: minutes for the first, hours for the second Moreover, the Particle filter has demonstrated again to be able to improve the predictions of the first stage, while ICP can not. This improved results are shown in figure 5.11 and figure 5.10, where the latter has low accuracy. In the end, using the same adversarial examples of 5.7, the SIFT approach (followed by Particle Filter) can detect the *can of coke* in two out of three malicious attack, as show in figure 5.9.



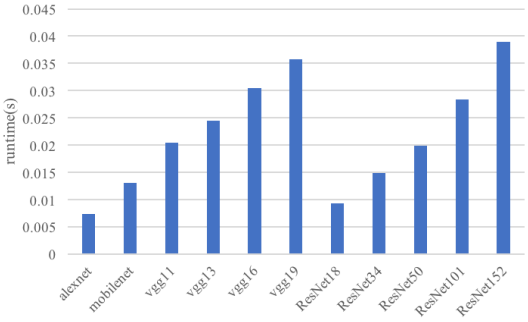

(a) Energy Jetson TX2




(b) Energy Titan XP




(c) Runtime TX2




(d) Runtime Titan XP

Figure 5.8: Energy consumption and runtime measurements

(a) Dark Scene

(b) Clutter Scene

(c) Surface Change Scene

Figure 5.9: Adversarial Attacks with SIFT + Particle filter: it detects the coke in the overlapped modification scene and in the surface change scene, missing the object in the dark scene

| window size | Fist Stage (mAP) | ICP (% accuracy) |
|---|---|---|
| 90 | 0.143 | 0.485 |
| 120 | 0.200 | 0.439 |
| 150 | 0.179 | 0.404 |
| 180 | 0.126 | 0.388 |
| 200 | 0.188 | 0.444 |

Table 5.5: Accuracy of SIFT-based first stage and ICP with different size of sliding window

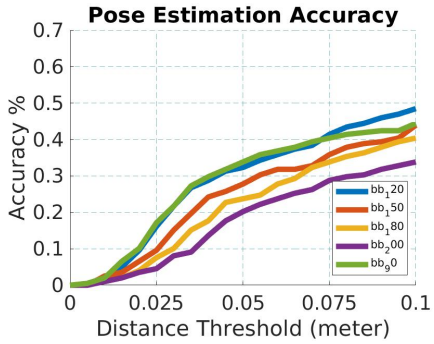



Figure 5.10: Accuracy SIFT-based first stage with ICP

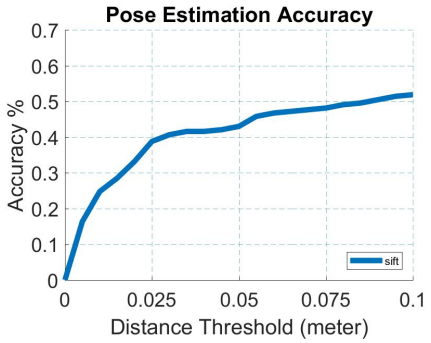



Figure 5.11: Accuracy SIFT-based first stage with Particle Filter

# Chapter 6

# Conclusions

This work presents a convolutional neural network analysis within a two-stage pipeline applied on robotic grasp where object detection are used as an input for pose estimation. In particular, eleven different networks architecture has been implemented through the Faster-RCNN object detector model, followed by a simple pose estimation method, represented by the Iterative Closest Point algorithm. The network architectures used are characterized by different depth and complexity: from the smaller MobileNet to the very deep VGG (with configurations 11,13,16,19), passing through AlexeNet and different configuration of ResNet. Observing the final value of accuracy obtained with each model, this work shows that a smaller CNN model can generate comparable grasping results as a larger CNN model. For a mobile manipulation robot, this represents the opportunity of having an embedded lightweight CNN model to save energy and runtime while still achieving high performance in robot grasping tasks.

Moreover, CNNs' power consumption and runtime measurements has been performed with two different system, a powerful GPU and a embedded board for machine learning application, due to the lack of this type of analysis in research works.

Furthermore, this work shows that relying solely on the CNN to make hard decisions for scene perception tasks leaves several open and vulnerabilities, especially when dealing with complex scenes. A further two-stage pipeline, called *SAND*, has been developed; which introduce a generative pose estimation method for the second stage. The first stage, instead, is composed by Pyrimad-CNN model, that is able to generate heatmap for the generative method. All the network architectures used for Faster-RCNN are also implemented in Pyramid approach, showing that SAND pipeline reaches high accuracy w.r.t the first pipeline. Moreover, using a two-stage SAND approach that avoids hard thresholding of the CNN output, improves the

robustness of object manipulation in complex scenes.

These complex scenes have some similarities to scenarios used for malicious attacks. Indeed, three types of malicious attacks have emulated through com- plex scenes, such as light condition, changes in objects aspect and overlay part of the object. Also in this analysis, collected results still demonstrate the SAND approach gets more robust than Baseline approach.

# Bibliography


[1] Besl, P. J., and McKay, N. D. Method for registration of 3-d shapes. In *Sensor Fusion IV: Control Paradigms and Data Structures* (1992), vol. 1611, International Society for Optics and Photonics, pp. 586–607.

[2] Carlini, N., and Wagner, D. Towards evaluating the robustness of neural networks. *arXiv preprint arXiv:1608.04644* (2016).

[3] Ciocarlie, M., Hsiao, K., Jones, E. G., Chitta, S., Rusu, R. B., and Şucan, I. A. Towards reliable grasping and manipulation in household environments. In *Experimental Robotics*. Springer Berlin Heidelberg, 2014, pp. 241–252.

[4] Collet, A., Martinez, M., and Srinivasa, S. S. The moped framework: Object recognition and pose estimation for manipulation. *The International Journal of Robotics Research* (2011), 0278364911401765.

[5] Courbariaux, M., Hubara, I., Soudry, D., El-Yaniv, R., and Bengio, Y. Binarized neural networks: Training deep neural networks with weights and activations constrained to+ 1 or-1. *arXiv preprint arXiv:1602.02830* (2016).

[6] Csurka, G., Dance, C., Fan, L., Willamowski, J., and Bray, C. Visual categorization with bags of keypoints. In *Workshop on statistical learning in computer vision, ECCV* (2004), no. 1-22, Prague, pp. 1–2.

[7] Everingham, M., Van Gool, L., Williams, C. K., Winn, J., and Zisserman, A. The pascal visual object classes (voc) challenge. *International journal of computer vision 88*, 2 (2010), 303–338.

[8] Fassold, H., and Rosner, J. A real-time gpu implementation of the sift algorithm for large-scale video analysis tasks. In *Real-Time Image and Video*

*Processing 2015* (2015), vol. 9400, International Society for Optics and Photonics, p. 940007.

[9] Fischler, M. A., and Bolles, R. C. Random sample consensus: a paradigm for model fitting with applications to image analysis and automated cartography. In *Readings in computer vision*. Elsevier, 1987, pp. 726–740.

[10] Girshick, R. Fast r-cnn. *arXiv preprint arXiv:1504.08083* (2015).

[11] Girshick, R., Donahue, J., Darrell, T., and Malik, J. Rich feature hierarchies for accurate object detection and semantic segmentation. In *Computer Vision and Pattern Recognition (CVPR), 2014 IEEE Conference on* (2014), IEEE, pp. 580–587.

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

[13] Gualtieri, M., ten Pas, A., and Jr., R. P. Category level pick and place using deep reinforcement learning. *CoRR abs/1707.05615* (2017).

[14] Han, S., Pool, J., Tran, J., and Dally, W. Learning both weights and connections for efficient neural network. In *Advances in neural information processing systems* (2015), pp. 1135–1143.

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

[16] He, S. J., Zhao, S. T., Bai, F., and Wei, J. A method for spatial data registration based on pca-icp algorithm. In *Advanced Measurement and Test III* (8 2013), vol. 718 of *Advanced Materials Research*, Trans Tech Publications, pp. 1033–1036.

[17] Hinterstoisser, S., Lepetit, V., Ilic, S., Holzer, S., Bradski, G., Konolige, K., and Navab, N. Model based training, detection and pose estimation of texture-less 3d objects in heavily cluttered scenes. 548–562.

[18] Hinton, G. E., Srivastava, N., Krizhevsky, A., Sutskever, I., and Salakhutdinov, R. R. Improving neural networks by preventing co-adaptation of feature detectors. *arXiv preprint arXiv:1207.0580* (2012).

[19] Howard, A. G., Zhu, M., Chen, B., Kalenichenko, D., Wang, W., Weyand, T., Andreetto, M., and Adam, H. Mobilenets: Efficient convolutional neural networks for mobile vision applications. *CoRR abs/1704.04861* (2017).

[20] Huang, L., Yang, Y., Deng, Y., and Yu, Y. Densebox: Unifying landmark localization with end to end object detection. *arXiv preprint arXiv:1509.04874* (2015).

[21] Huang, S., Papernot, N., Goodfellow, I., Duan, Y., and Abbeel, P. Adversarial attacks on neural network policies. *arXiv preprint arXiv:1702.02284* (2017).

[22] Iandola, F. N., Moskewicz, M. W., Ashraf, K., Han, S., Dally, W. J., and Keutzer, K. Squeezenet: Alexnet-level accuracy with 50x fewer parameters and <1mb model size. *CoRR abs/1602.07360* (2016).

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

[24] Kurakin, A., Goodfellow, I., and Bengio, S. Adversarial examples in the physical world. *arXiv preprint arXiv:1607.02533* (2016).

[25] Lai, K., Bo, L., and Fox, D. Unsupervised feature learning for 3d scene labeling. In *Robotics and Automation (ICRA), 2014 IEEE International Conference on* (2014), IEEE, pp. 3050–3057.

[26] Lai, K., Bo, L., Ren, X., and Fox, D. A large-scale hierarchical multi-view rgb-d object dataset. In *Robotics and Automation (ICRA), 2011 IEEE International Conference on* (2011), IEEE, pp. 1817–1824.

[27] LeCun, Y., Bengio, Y., and Hinton, G. Deep learning. *nature 521*, 7553 (2015), 436.

[28] Lin, D., Talathi, S., and Annapureddy, S. Fixed point quantization of deep convolutional networks. In *International Conference on Machine Learning* (2016), pp. 2849–2858.

[29] Long, J., Shelhamer, E., and Darrell, T. Fully convolutional networks for semantic segmentation. In *Proceedings of the IEEE conference on computer vision and pattern recognition* (2015), pp. 3431–3440.

[30] Lowe, D. G. Distinctive image features from scale-invariant keypoints. *International journal of computer vision 60*, 2 (2004), 91–110.

[31] Morris, D. D. A pyramid cnn for dense-leaves segmentation. *arXiv preprint arXiv:1804.01646* (2018).

[32] Narayanan, V., and Likhachev, M. Discriminatively-guided deliberative perception for pose estimation of multiple 3d object instances. In *Proceedings of Robotics: Science and Systems* (AnnArbor, Michigan, June 2016).

[33] Nathan Silberman, Derek Hoiem, P. K., and Fergus, R. Indoor segmentation and support inference from rgbd images. In *ECCV* (2012).

[34] Panda, P., and Roy, K. Explainable learning: Implicit generative modelling during training for adversarial robustness. *arXiv preprint arXiv:1807.02188* (2018).

[35] Papazov, C., Haddadin, S., Parusel, S., Krieger, K., and Burschka, D. Rigid 3d geometry matching for grasping of known objects in cluttered scenes. *The International Journal of Robotics Research* (2012), 0278364911436019.

[36] Papernot, N., McDaniel, P., Goodfellow, I., Jha, S., Celik, Z. B., and Swami, A. Practical black-box attacks against machine learning. In *Proceedings of the 2017 ACM on Asia Conference on Computer and Communications Security* (2017), ACM, pp. 506–519.

[37] Redmon, J., Divvala, S., Girshick, R., and Farhadi, A. You only look once: Unified, real-time object detection. *arXiv preprint arXiv:1506.02640* (2015).

[38] Ren, S., He, K., Girshick, R., and Sun, J. Faster r-cnn: Towards real-time object detection with region proposal networks. In *Advances in Neural Information Processing Systems* (2015), pp. 91–99.

[39] Sermanet, P., Eigen, D., Zhang, X., Mathieu, M., Fergus, R., and LeCun, Y. Overfeat: Integrated recognition, localization and detection using convolutional networks. *arXiv preprint arXiv:1312.6229* (2013).

[40] Shekhar, R., and Jawahar, C. Word image retrieval using bag of visual words. In *Document Analysis Systems (DAS), 2012 10th IAPR International Workshop on* (2012), IEEE, pp. 297–301.

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

[42] Sivic, J., Russell, B. C., Efros, A. A., Zisserman, A., and Freeman, W. T. Discovering objects and their location in images. In *Computer Vision, 2005. ICCV 2005. Tenth IEEE International Conference on* (2005), vol. 1, IEEE, pp. 370–377.

[43] Sucan, I. A., Moll, M., and Kavraki, L. E. The open motion planning library. *IEEE Robotics & Automation Magazine 19*, 4 (2012), 72–82.

[44] Sudderth, E. B., Torralba, A., Freeman, W. T., and Willsky, A. S. Learning hierarchical models of scenes, objects, and parts. In *Computer Vision, 2005. ICCV 2005. Tenth IEEE International Conference on* (2005), vol. 2, IEEE, pp. 1331–1338.

[45] Sui, Z., Zhou, Z., Zeng, Z., and Jenkins, O. C. Sum: Sequential scene understanding and manipulation. *arXiv preprint arXiv:1703.07491* (2017).

[46] Sui, Z., Zhou, Z. Y., and Jenkins, O. C. Never mind the bounding boxes, here's the sand filters. *arXiv preprint arXiv:1808.04969* (2018).

[47] Sze, V., Chen, Y.-H., Yang, T.-J., and Emer, J. S. Efficient processing of deep neural networks: A tutorial and survey. *Proceedings of the IEEE 105*, 12 (2017), 2295–2329.

[48] Szegedy, C., Liu, W., Jia, Y., Sermanet, P., Reed, S., Anguelov, D., Erhan, D., Vanhoucke, V., and Rabinovich, A. Going deeper with convolutions. *arXiv preprint arXiv:1409.4842* (2014).

[49] Szegedy, C., Zaremba, W., Sutskever, I., Bruna, J., Erhan, D., Goodfellow, I., and Fergus, R. Intriguing properties of neural networks. *arXiv preprint arXiv:1312.6199* (2013).

[50] ten Pas, A., Gualtieri, M., Saenko, K., and Platt, R. Grasp pose detection in point clouds. *The International Journal of Robotics Research 36*, 13-14 (2017), 1455–1473.

[51] Ten Pas, A., and Platt, R. Localizing handle-like grasp affordances in 3d point clouds. In *Experimental Robotics* (2016), Springer, pp. 623–638.

[52] Uličnỳ, M., Lundström, J., and Byttner, S. Robustness of deep convolutional neural networks for image recognition. In *International Symposium on Intelligent Computing Systems* (2016), Springer, pp. 16–30.

[53] Varley, J., Weisz, J., Weiss, J., and Allen, P. Generating multi-fingered robotic grasps via deep learning. In *2015 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)* (Sept 2015), pp. 4415–4420.

[54] Wong, J. M., Kee, V., Le, T., Wagner, S., Mariottini, G.-L., Schneider, A., Hamilton, L., Chipalkatty, R., Hebert, M., Johnson, D. M., et al. Segicp: Integrated deep semantic segmentation and pose estimation. In *Intelligent Robots and Systems (IROS), 2017 IEEE/RSJ International Conference on* (2017), IEEE, pp. 5784–5789.

[55] Xie, S., Girshick, R., Dollár, P., Tu, Z., and He, K. Aggregated residual transformations for deep neural networks. In *Computer Vision and Pattern Recognition (CVPR), 2017 IEEE Conference on* (2017), IEEE, pp. 5987–5995.

[56] Yang, J., Jiang, Y.-G., Hauptmann, A. G., and Ngo, C.-W. Evaluating bag-of-visual-words representations in scene classification. In *Proceedings of the international workshop on Workshop on multimedia information retrieval* (2007), ACM, pp. 197–206.

[57] Yang, J., Lu, J., Batra, D., and Parikh, D. A faster pytorch implementation of faster r-cnn. *https://github.com/jwyang/faster-rcnn.pytorch* (2017).

[58] You, Y., Zhang, Z., Hsieh, C., and Demmel, J. 100-epoch imagenet training with alexnet in 24 minutes. *CoRR abs/1709.05011* (2017).

[59] Zheng, S., Song, Y., Leung, T., and Goodfellow, I. Improving the robustness of deep neural networks via stability training. In *Proceedings of the ieee conference on computer vision and pattern recognition* (2016), pp. 4480–4488.

[60] Zhu, C., Han, S., Mao, H., and Dally, W. J. Trained ternary quantization. *arXiv preprint arXiv:1612.01064* (2016).