Robotic Grasp Pose Classification over IR-Depth Images Using Convolutional Neural Networks

Nedal Horany

MASTER THESIS

submitted to

The Department of

Computer Science, University of Bonn

Bonn, Germany

Supervisors: Prof. Dr. Sven Behnke and Dr. Seongyong Koo

September 2017

Declaration

I hereby declare and confirm that this thesis is entirely the result of my own original work. Where other sources of information have been used, they have been indicated as such and properly acknowledged. I further declare that this or similar work has not been submitted for credit elsewhere.

Bonn, Germany, September 19, 2017

Nedal Horany

Contents

Abstract

Given the image observations of a single object or occluded objects in a bin, a major challenge is predicting the best grasp under high gripper uncertainty. Recently various works adopted deep learning and computer vision methods in order to solve this problem. While most of these approaches focus on detecting and suggesting the best grasp given an image from infinite domain, this work focuses on binary classification of a single grasp pose using a data-driven approach, which outperforms state-of-the-art in grasp pose classification using deep Convolutional Neural Network (CNN). This is achieved by feeding the network with IR-Depth image and grasp pose (x, y, θ) corresponding to the center coordinates and the orientation of the grasp in the $2D$ image plane. The network is able to answer the question whether the object is graspable within the given pose. Labeling was done in two stages. Firstly: Surrounding graspable and ungraspable regions with bounding boxes using the depth image, taking into consideration the configuration of the the robot gripper. Secondly: Generating positive and negative grasp poses, based on random translation and rotation, which served later for training and validating. We compare between two trained models, one using IR-Depth and the other using only depth images. We show that IR can improve the overall grasp pose classification. Finally, we present a multi-grasp prediction heuristic, that uses the classifier over local regions to predict multiple high quality grasp poses over the entire image.

Chapter 1

Introduction

This master thesis studies grasp prediction problem for bin picking. The problem is concerned with a robotic arm equipped with parallel jaw-gripper, that needs to detect multiple graspable parts from a random heap of parts (see Figure 3.1), having specific texture and geometric shape using sensors data. One challenge is that these parts, so called chain links in Figure 3.2 occlude with each other and each can be grasped from different edges, depending on their posture in the bin. Which leaves us dealing with large number of possible grasps that we need to predict.

Reliable grasp prediction is essential task for integrating these robots in industry for the various applications e.g. parts feeding and sorting. The feeding of parts from bulk supplies to production lines is a common task in industrial automation. Usually it requires special devices such as vibratory bowl feeders, where it works very well for certain type of objects having specific shape features. However such a technique needs a specific ramp designs for each part, which limits their usage for mass production of identical parts. Bin picking is another possibility, where a robot arm with a camera and gripper lying above the bulk supply, it detects graspable parts from the sensed data by a grasp detection algorithm, using Inverse Kinematics and Motion Planning, the arm grasp the object and move it into the part feeder.

In this work, we focus on grasp detection task. Some works try to solve this problem using hand-designed features, given the CAD model of the object one could define a set of object characteristic features to determine

how and where the object can be grasped. However, these methods strongly rely on accurate models due the usage of the CAD models, which makes it difficult to apply on real scenarios where we have noisy sensors measurements. In addition, these approaches usually require enormous amount of time spent on hand engineering the features and it can't be generalized for novel objects. While other approaches used deep learning, where Convolutional Neural Networks (CNN) has resulted in outstanding performance on object detection and classification, by learning features directly from images with high computational power using the GPUs. This would save us the time and effort of hand-engineering the features.

CNN has hierarchical representation consisting of cascaded layers, where each layer uses the output of the previous layer. The multiple layers correspond to different levels of features learning, the more convolution layers we have the more complicated features the network will be able to learn to recognize. Such characteristic can help with extracting features from the images and learning non-linear transformations between those and the labels.

We adopted deep learning using CNN approach. In the contrary to other approaches trying to deep learn the best grasp from an image, we propose a queryable CNN, which is able to classify the graspability of any $3D$ grasp pose illustrated in Figure 3.3 over given IR-Depth image, we achieve that by feeding $3D$ vector directly into the fully connected layer. One important advantage obtained by this solution, is that it can be integrated easily with different applications and heuristics for determining grasps candidates.

1.1 Contributions

In this work, we contribute a new technique for learning queriable $3D$ grasp poses classifier over IR-Depth images based on Convolution Neural Network. We have generated a dataset of IR and Depth images and introduced a new technique for manual labeling which copes with our task goal. The dataset was generated in a way that allows us to achieve depth invariant trained models, which is able to classify grasp poses from different depth levels. We also compare between two trained models, the first model is trained using depth images, and the second model is trained using two channels images

for depth and IR. Furthermore, we show the contribution of IR channels in improving the overall grasp pose classification and detection. The proposed solution can be integrated in many applications and combined with different heuristics, this flexibility is desirable for robotic applications. We propose a heuristic that uses the learned classifier for predicting grasp poses, this uses a modified CNN architecture to speed up the evaluation of large amount of grasp poses on image at once.

1.2 Structure of this Thesis

Chapter 2 presents a survey of existing approaches that address the grasp detection problem. Each approach is briefly discussed and its limitations are pointed out. The chapter also introduces background for the problem, with work that solves same problem with different approach. We span known Convolutional Neural Networks architectures and the tasks it solves. Furthermore, we summarize briefly the different CNN components together with techniques used in this master thesis.

Our approach along with accompanying heuristics are presented in 3. We start by defining the target grasp pose, positive grasp pose and introducing a heuristic to find grasp ground truth over the depth images. This served as a baseline for the manual labeling. We explain the dataset generation technique from the labeled images, which served for training and validating the CNN. Afterwards, we introduce the CNN architecture, the training strategy and summarize the empirical results, while making performance comparison between two trained models. Finally, we propose an application using the grasp pose classifier, to predict multiple robust grasp poses over entire images.

We conclude the work of this master thesis in Chapter 4 and provide insights and thoughts for further development and improvements.

Chapter 2

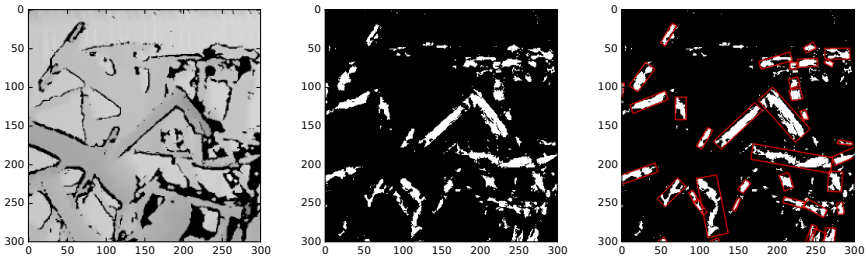
Related Work

2.1 Background

(a) Environment setup

(b) Chain link parts variations which is used to assemble chains conducting cables

In this section, we will provide a background of the problem and summarize briefly a work that solves the same problem using different approach.



(a) Depth map **(b)** Valley pixels **(c)** Graspable parts

Figure 2.2: Finding graspable parts from a depth map

The task is concerning robolink® WR manipulator[1] in Figure 2.1a, which need to detect graspable chain links part (Shown in Figure 2.1b) from a bin using SR300 RGBD camera mounted on the wrest, pick one graspable part and feed it into a slot.

In work of [Koo et al., 2017], they solved grasp pose detection for the same task using hand engineered features over the depth image. Their key idea is to traverse over the $3D$ image surface shown in Figure 2.2a, centering each pixel in a bounding box with size of the projected stroke at that point. Then it chooses pixels, which has no neighbors within the bounding box having depth values smaller than the depth at the center. Figure 2.2b shows the valleys which correspond to those pixels. Multiple graspable parts then computed using growth clustering between the connected local depth minima, then surrounding each cluster with minimal bounding box (red rectangles in Figure 2.2c) represented by its width, height centers coordinates and orientation. Reliable grasp pose then is determined by picking the cluster that has minimal average depth value over its pixels. The grasp pose position was determined as its corresponding bounding box center, with orientation orthogonal to the bounding box.

In their experiments, they have performed 20 bin picking attempts, for each the robot try to place the part on the table. Once it failed to detect the part on the table, it returns into the observation position and select the second highest graspable part. In 65% of the attempts the robot successfully

[1] http://www.igus.de/wpck/6076/robolink

managed to pick the part from the first attempt and 100% from the second attempt. Due to false positive grasp detection, typically caused form noise and incorrect depth measurements, resulting lower success rates during the first attempt. This work was motivated as an attempt for increasing the success rates from the first attempt, by proposing a solution for classifying grasp poses with high precision using extra additional source of information which is Infra-Red (IR) channel.

2.2 Deep Learning

Deep learning is subfield of machine learning concerned with large neural networks, simulating the functionality of the brain. It comes as a solution for allowing computers to learn from experience and understanding hierarchy of concepts from real world problems. The concepts hierarchy are related in terms of complexity, usually from complex to simpler. This is achieved by gathering knowledge from experience which avoids the need for a human to formally define and specify the knowledge that computer needs in order to solve some problem. It can be represented as a graph structure, where concepts are built on top of each other, resulting a deep graph with many layers which is the reason behind calling this approach deep learning [Goodfellow et al., 2016].

Deep learning has proved to be solving variety of tasks in image detection and speech recognition, that relatively can be considered hard for humans but straight forward for computers. This made the field very popular recently and adopted for variety of AI applications.

Deep learning can be divided into two main fields:

- **Supervised Learning** - When training a neural network on labeled dataset, where each example in the dataset associated to a corresponding label. The network should be able provide the target label for new inputs, by learning sort of function mapping between the input and target. When labels are classes the problem called classification. Alternatively, when the label space is continuous the problem is called regression.

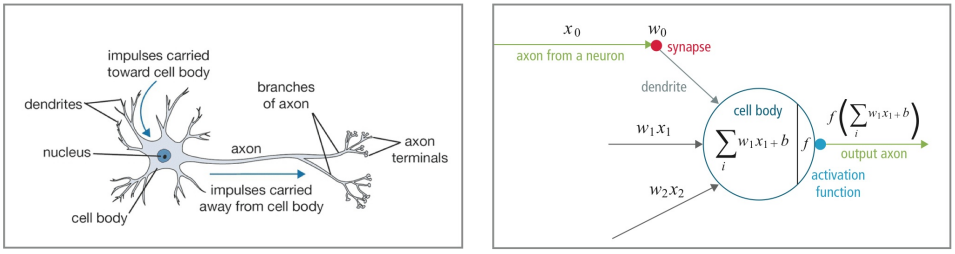
- **Unsupervised Learning** - Concerned neural network which learns from unlabeled dataset, usually large amount of data. The network is trained to learn representative features and patterns from the dataset for multiple purposes e.g. clustering. This method can be very useful when having large amount of unlabeled dataset and small amount of labeled dataset. The network is trained to get good features from the ulabeled dataset and if we are trying to solve classification task for example, then we take the learned features and apply on that a supervised learning using the labeled dataset to solve the task.

2.3 Convolutional Neural Networks

Neural networks has been inspired from the biological modeling of the neural systems of the brain illustrated in Figure 2.3. Each neuron is a computation unit which receives inputs and produce outputs to the next layer. Mathematically, a neuron can be translated into activation function f over a linear layer, which applies linear transformation on vector input x of dimension I, weights matrix A and outputs a vector with bias parameter b.

$$y = A \cdot x + b \tag{2.1}$$

$$y_i = \sum_{k=0}^{I} A_{i,k} \cdot x_k + b_i \tag{2.2}$$



(a) **(b)**

Figure 2.3: (a) illustrates biological model of neuron and (b) its mathematical model.

2.3.1 Activation or Non-Linear Functions

Neural networks can approximate complex functions including non-convex functions as a result of using non-linear activation layers [Agostinelli et al., 2014]. The three main activation functions illustrated in Figure 2.4:

1. **Sigmoid**: It has a dense representation by squashing a real value to fall between 0 and 1

$$\sigma(x) = \frac{1}{1 + e^{-x}} \tag{2.3}$$

   One known problem with sigmoid function is that its gradient vanishes whenever absolute x increases. Consequently, backpropogation algorithm will fail to update the neuron weights.

2. **Hyperbolic Tanget (Tanh)**: It has also a dense representation by squashing a real value to fall between -1 and 1

$$\tanh(x) = 2\sigma(2x) - 1 \tag{2.4}$$

   Like Sigmoid function its activation saturates, however it has a zero-centered output.

3. **Rectified Linear Unit (ReLU)**: It has a sparse representation by accepting only positive values

$$y = max(0, x) \tag{2.5}$$

   The neuron which this function is operating on is called ReLU following [Nair and Hinton, 2010], despite the simplicity of the function, it has been found to accelerate the convergence of stochastic gradient descent compared to the Tanh in [Krizhevsky et al., 2012a] by factor of 6. In the contrary to Sigmoid function, it doesn't have the problem with vanishing gradient and it requires less computation. However, it ignores negative values which can be undesirable for certain tasks.

2.3.2 Spatial Convolution

In this work, we are interested in learning spatial relationships contained in the features map (e.g. images). This can be achieved using convolution

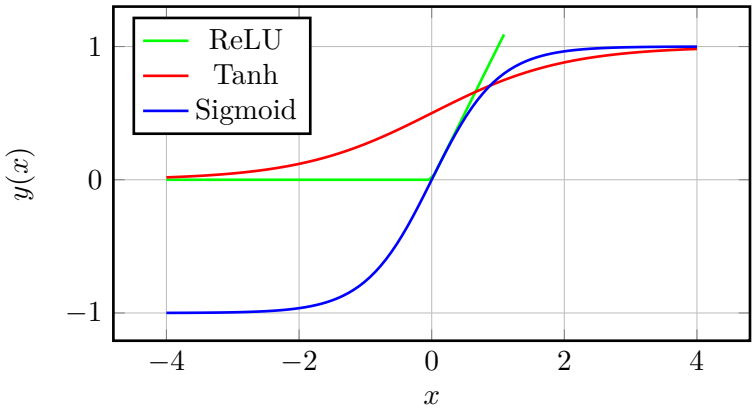


Figure 2.4: Illustrated graphs for three activation functions: ReLU, Tanh and Sigmoid

layer, unlike neural networks where the input is a vector, convolution layer learns a set of filters from multi-channeled image. The layer accepts an image of size $w \times h \times c$ corresponding respectively to height, width and to image channels number (e.g. $c = 3$ for RGB images. The layer can learn a set of k filters (kernel) of size $n \times n \times q$, where the filter dimensions are smaller from the image dimensions. For $w = h$, it produce k features maps of size $w - n + 1$, each is convolved with the input image (see Figure 2.5). The more the filter correlates with a region in the image, the more strong its corresponding location in the features map.

2.3.3 Spatial Pooling

Pooling layers help with down-sampling the input by aggregating chunks R, typically of size 2×2 into single value (see Figure 2.6).

$$y_R = P_{i \in R}(x_i) \tag{2.6}$$

Where P is a pooling function over region R. Max pooling is commonly used for this layer, which returns the maximum value in every region (chunk). It helps avoiding cancellation of negative values and prevents bluring in the result of the preceding activation layers. One important features of this layer, is that it decreases the dimension of the image and provides invariance between similar inputs.

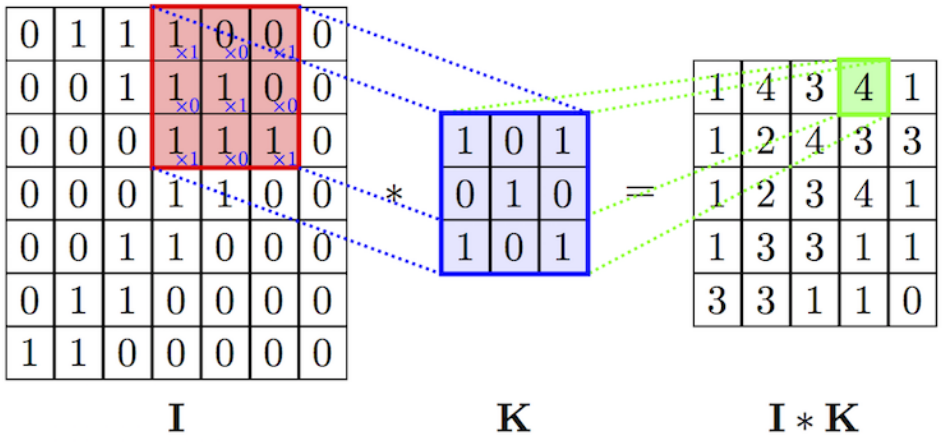


Figure 2.5: The figure illustrates the convolution operator with simple example.

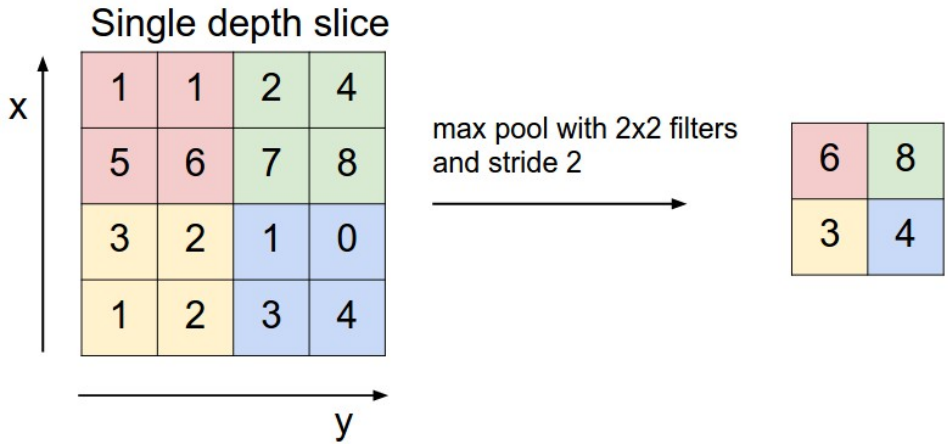


Figure 2.6: Left figure illustrates down-sampling image using pool layer, right figure show example for max-pooling

2.3.4 Batch Normalization

This layer helps the CNN to converge faster [Ioffe and Szegedy, 2015], which made it very popular in the recent deep learning works. It's a technique for maintaining zero mean and unit variance by input shifting. It helps the different trainable layers inputs to become comparable across features. Consequently, ensuring the network learning ability while using high learning rates and any initial values for the weights vector. Batch normalization ful-

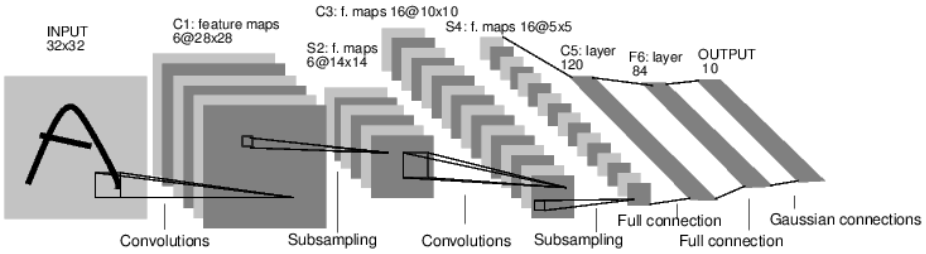


Figure 2.7: Architecture of LeNet-5.

fills the need of using bias vectors as well as it solves the problem concerning sigmoids and similar functions of having vanishing gradient.

2.4 CNN Architectures

Convolutional neural networks can be constructed in many ways using the above described layers, and its size may vary depending on the task it's trying to solve. The deeper the network the more it can solve complex problems accurately, however it requires more time for processing. It's rather difficult task to decide the correct trade-off between accuracy and speed. Fortunately, there are many works in this field suggesting different CNNs architectures, which usually can be a good baseline for similar tasks. In this section, we will describe briefly some known architectures which achieved high performance on its given task.

2.4.1 LeNet-5

This architecture is one of the leading successful applications of CNNs, it was developed in the 1990's for hand-written and machine printed digits recognition [LeCun et al., 2001]. It was the source of inspiration for the modern CNNs architectures.

In Figure 2.7 we can see their CNN architecture, it was constructed using 3 layers of convolution, average pooling and non-linear layers using hyperbolic tangent and sigmoid function, followed by fully connected layer as classifier. The network was trained on MNIST (database of handwritten digits) [LeCun and Cortes, 1998], for accelerating the training process they

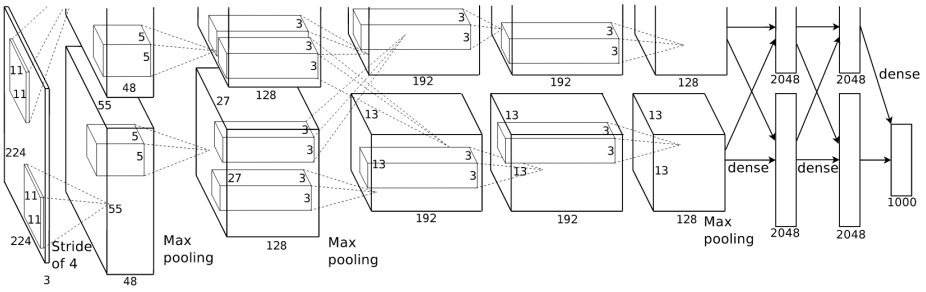


Figure 2.8: Architecture of AlexNet.

normalized the inputs using the method proposed in [Cun et al., 1991].

2.4.2 AlexNet

AlexNet [Krizhevsky et al., 2012b] achieved high accuracy rates on ImageNet Large-Scale Visual Recognition Challenge 2012 (ILSVRC12) [**deng2012largem**] challenge. The CNN architecture in Figure 3.4 is deeper compared to LeNet using 5 convolution layers, 3 max-pooling and 3 fully connected layers using ReLU for activation functions. In their work, they have adopted the method of overlapping max-pooling for decreasing the error rate. Moreover, introduced the method of stacking of convolution layers before using the pool layer and at that time, employing a recent regularization method in the fully connected layers to reduce overfitting. There method was implemented in CUDA and running on multiple GPUs, achieving low error-rates (15.3%) on ImageNet classification compared to second-best place (26.2%) .

2.4.3 VggNet

[Simonyan and Zisserman, 2014] introduced very deep CNNs, VggNet-16 (see Figure 2.9) and VggNet-18, which was submitted to ILSVRC14 challenge with around 140 million of parameters. The network ranked second-best place with 7.32% in top-5 error for image classification.

Their main idea is to decompose large convolution filter sizes e.g. 5×5 and 7×7 into multiple 3×3 convolutions with strides of 1 pixel on top of each other in order to emulate the effect of larger receptive fields. Consequently, using more convolutions and pooling layers for reducing input dimensions,

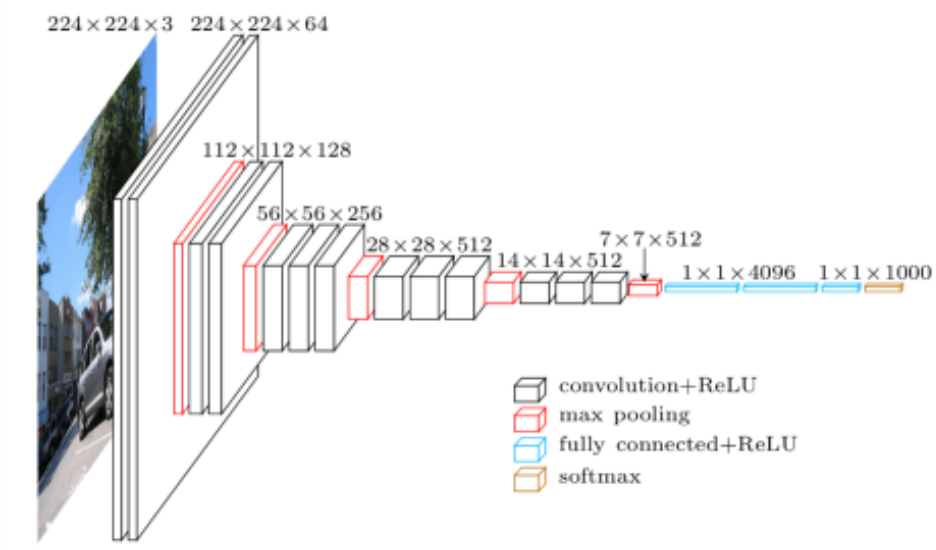
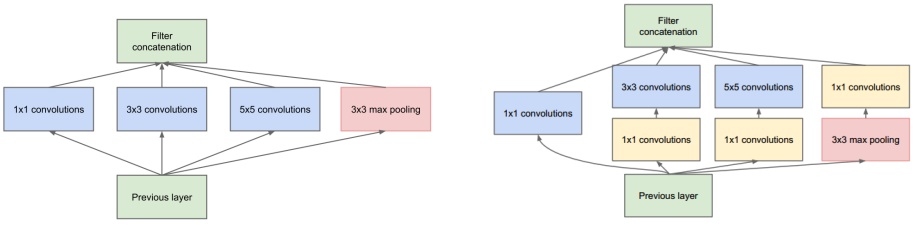


Figure 2.9: Architecture of Vgg-16 [Cord, 2016].

which makes the network much deeper.

2.4.4 GoogLeNet

GoogLeNet [Szegedy et al., 2014] is a 22 layers deep network which is the winner of ILSVRC14 challenge with 6.67% top-5 error. They proposed the Inception Module (see Figure 2.10), which has significantly reduced the number of parameters. They used 1×1 convolutions before 3×3 convolutions for reducing the input size while keeping the computational budget constant.



(a) Inception module (naive version)

(b) Inception module with dimension reductions

Figure 2.10: Inception module

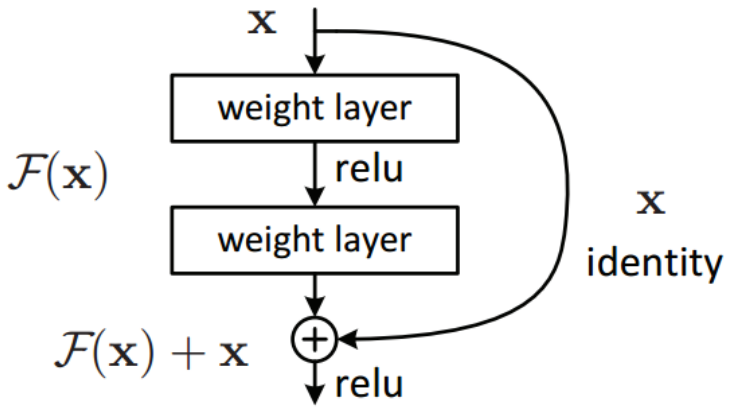


Figure 2.11: Building block for Residual learning

2.4.5 ResNet

ResNet [He et al., 2015] is the winner of ImageNet challenge ILSVRC15 [Russakovsky et al., 2015]. The network achieved low error rates 3.57% using what so called Deep Residual Learning framework. It's a special case of Feed-forward Highway LSTM network [Srivastava et al., 2015] (Long Short-Term Memory) but without gates.

Denoting g, t, h and $\mathcal{F}$ for non-linear differentiable functions. $\mathcal{F}(x) = g(x)x + t(x)h(x)$ is computed by each of the non-input layers of the Highway nets [Srivastava et al., 2015] given x from the previous layer. Figure 2.11 illustrates residual block in ResNet which uses $g(x) = 1$ and $t(x) = 1$ for feed-forward, where each residual layer calculates $\mathcal{F}(x) = x + h(x)$.

2.5 Grasp Prediction Using Deep Learning

Detecting optimal grasp poses from complex scenes, hand-engineered features still performs well, for example in dense clutter [Boularias et al., 2015], where features of the grasp are all points from the point clouds that may collide with the robotic hand through grasping, or multi-fingered grasping [Kopicki et al., 2016] by extracting 3D object surface features using point clouds, which is a composition of position, orientation and a vector describing the local curvatures. Nevertheless, for simple scenes CNNs has proven

significant improvement on grasp pose regression and classification. One challenge of using these deep learning methods is finding a relevant labeled data collection to exploit the full capacity of CNNs while avoiding overfitting.

The majority of these methods falls into one of the two categories:

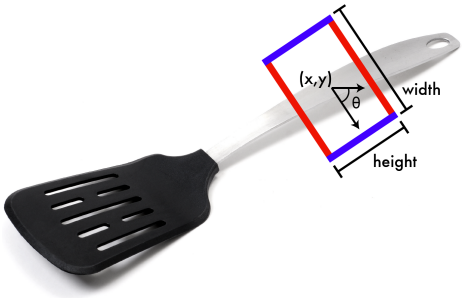
- Regression over an optimal grasp from the entire observation image or from individual patch
- Recasting grasping prediction problem as classification

The following sections, provide span over grasp detection methods divided into two main groups: **Grasp Regression** and **Grasp Classification**.

2.5.1 Prediction as Regression

Most of the works used manually human-labeled collection like *The Cornell Grasping Dataset* [Jiang et al., 2011], this dataset contains 1035 RGB-D images of 280 different graspable objects, some of them are shown in Figure 2.12a.

(a) A small set of object images from the Cornell Dataset.



(b) (x, y) is the grasp center, θ is the orientation with horizontal axis, the blue lines correspond to the gripper plate size and the red lines to the approximate start position of the parallel gripper.

Each image contains a single object taken from different orientations and with different postures. For each image the corresponding labels are a set of positive and negative grasping rectangles, defined as $5D$ vector in the image plane illustrated in Figure 2.12b. This representation is a lower dimensional

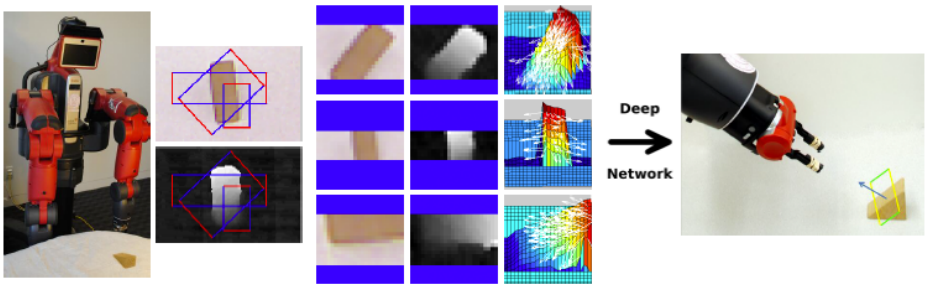


Figure 2.13: Two-staged model proposed for grasp detection in [Lenz et al., 2015]

of $7D$ grasp representation corresponding to $3D$ orientation, $3D$ location and gripper opening width. Claiming other representations like the $2D$ grasping point proposed in [Saxena et al., 2008] and pair of contact points for two fingered hand in [Le et al., 2010], aren't representative enough to predict $7D$ grasp, it used only to predict partial configuration while other dimensions are estimated separately.

[Lenz et al., 2015] proposes a grasp detection method by training two-staged CNN (see Figure 2.13) on Cornell dataset. First stage is fast and serve as pruner for unlikely grasp candidates. They trained a small deep network to search the space for possible grasps candidates using rectangle representation of the grasps and score them. Second stage it runs on top of the grasp candidates obtained from the previous stage and re-rank them according to their graspability using richer features space. While it focuses only on the part of the image contained within these rectangles. This was achieved using deeper neural network which is trained on 7 channels images of YUV, depth and computed $3D$ vector (x,y,z) correspond to the surface normal.

The performance was speeded up in [Angelova et al., 2015] by feeding the entire image. They propose three different models:

- Direct regression of the optimum grasp within the whole image
- Very similar to the first model, but it takes into consideration the object class
- Detects multi-grasps by dividing the image into $N \times N$ grid, then

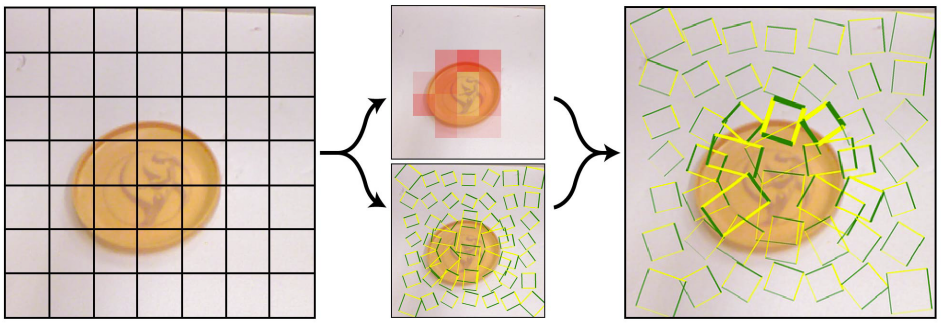


Figure 2.14

Figure 2.15: Third model proposed by [Angelova et al., 2015], left image contain the object divided into $N \times N$ grid. For each grid cell they predicted its optimal grasp and its score as shown in the middle stage. Then it outputs assigned score for each of the grasps candidates.

predicting optimum grasp per grid cell and the likelihood that the predicted grasp is feasible on that object as shown in Figure 2.14.

In order to avoid overfitting during the training, they picked random label every time the image was exposed to the network, the side-effect is that during optimizing the loss function, the network converges to the average good grasp. In this work, average good grasp over the regressed grasps is not necessary a good grasp.

[Kumra and Kanan, 2016] addresses the same problem using deeper CNN architecture than used in [(Angelova et al., 2015)]. They use a ResNet model [He et al., 2015] consists of 50 layers, which redefines state-of-the-art performance on Cornell dataset.

Above methods are not applicable in this work for the following reasons:

- Cornell dataset images contain centered single objects, whereas in this work, we are dealing with multiple objects lying in a random heap
- Cornell dataset images contain distinctive RGB colored objects from different categories, in this work the objects are the same and all have black color
- Direct grasp regression from an image or a patch underlying a strong assumption, that it contains a single good grasp. However, in this

work we could have scenarios where large number of different good grasps exist in one image. Additionally, such an assumption inherently require grasp existence within the image, which is defintely wrong for our problem.

2.5.2 Prediction as Classification

Convolutional Neural Networks has proven a strong performance on classification tasks like on ImageNet [Krizhevsky et al., 2012a]. In some of the methods they tried to reform the grasping training problem to classification task. For example in [Pinto and Gupta, 2016], they recast grasp regression into grasp classification of 18 different possible orientations with jumps of 10 degrees, centered at the patch center. They used patch size 1.5 larger than the projected gripper size to include more context. By random trials and errors in real-time experiments, they were able to collect a 50 thousand of grasp attempts, which used as prior grasping for training a CNN to classify graspable patches from an image, then to classify 18 possible orientations of grasp which is centered in the patch. During test they used random sampling of patches along Region of Interest and choosing top classified orientation for predicting the best grasp. Although collecting data with trial and error is less biased than human-labeled datasets, but it still an exhausting process consuming an enormous amount of time.

[Levine et al., 2016] proposes an online training method using multiple robots, executing grasping attempts simultaneously for couple of weeks. The key idea is to learn the motor commands directly from the images, this way the robot can coordinates with the scene during grasping attempt to achieve successful physical grasping, simulating hand-eye coordination for grasping.

Unlike the conventional methods used for collecting labeled data, where usually the training data consist of a grasp with rectangle representation like in [Angelova et al., 2015] or $3D$ grasp pose in [Pinto and Gupta, 2016] with assigned label. In their work they considered the the arm motion v_t in time frame T along with the images received from the cameras, until the robot closes the gripper (At time frame T). At each time step t the images I_t and the current pose p_t were recorded, resulting a training example $(I_t, p_T - p_t, l)$

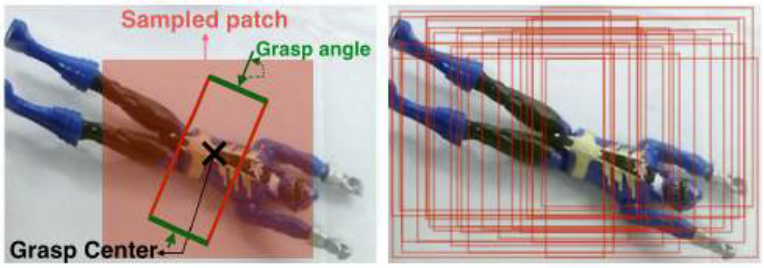


Figure 2.16: Left image shows a sampled ROI using Mixture of Gaussians (MOG) algorithm for reducing number of trials on empty spaces during collecting the data. A patch is sampled uniformly from the image space, where the grasp correspond to the patch center's coordinates with orientation randomly chosen in range $[0, \pi]$. The right image shows random sampling of patches along ROI during test time.

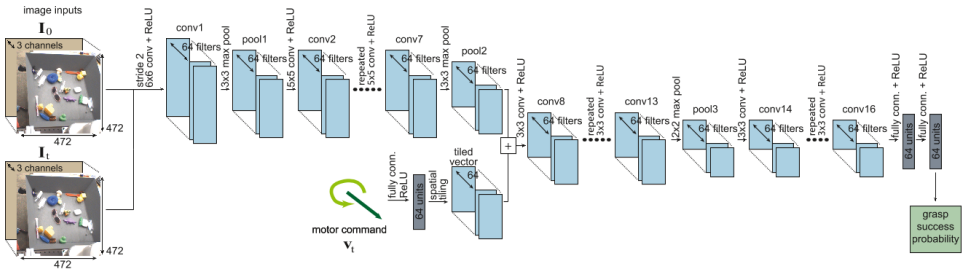
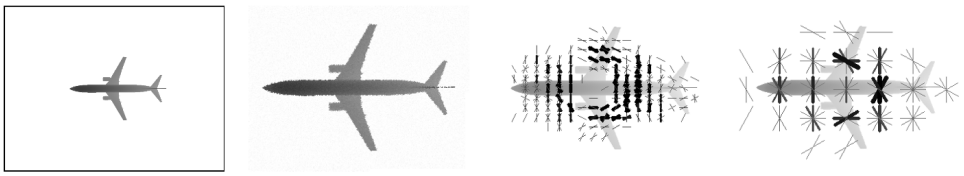


Figure 2.17: The CNN used for grasp prediction, the motor command is processed through fully connected layer and pointwise added to the result of pool2 after tiling in order to match dimensions. The result then is processed by more convolution layers and outputs grasp probability success using Sigmoid over the last layer.

where l is the evaluated grasp at time T and $p_T - p_t$ the motion vector. Every grasp attempt produce T new training samples. They have used specific CNN architecture illustrated in Figure 2.17 for feeding the motor command beside input images, which was source of inspiration for this work. Considering the time needed for collecting the data and the number of robots used for training, this method is not applicable for this work.

[Johns et al., 2016] proposes a new method in this field where they deep learn a grasp scoring function under gripper pose uncertainty using $3D$ representation of the grasp. They used physics simulations to collect their dataset taking into consideration depth information only. The key idea is to simulate



(a) Simulated depth image (b) Simulated depth image after adding the noise model (c) Visualisation of the grasp function (d) Visualisation of the grasp function at a coarser scale

Figure 2.18: An illustration of the grasp function learned through the training: (a) Shows the synthetic depth image, (b) shows scaled image after applying the noise model, (c) Visualize grasp function computed from physics simulation and (d) similar to (c) however with coarser poses distribution. The thickness of the grasp poses indicates the score of each pose.

robot grasping trials on $3D$ object meshes using the simulator for each possible grasp pose. Having observed depth image of size 640×480 , they learn scores of 8712 possible grasp poses, by discretizing the image space with 14 pixel in both axis corresponding to $1cm$ in the table surface, and the angle space by 30 degrees, Figure 2.18 shows the distribution of the scored grasp poses over the image by simulation. Each grasp pose was classified into one of the 5 classes corresponding to equally divided intervals in $[0, 1]$, indicating the likelihood of that object to be graspable with that pose.

This dataset was used for training a CNN to learn the grasp function. In their work's set-up they had a depth image of a single object. The CAD model of the object was required to perform such simulations. Where for complex scenes like in this work, given a random pile, it's intractable making similar simulations. Adding to this the amount of processing time for rendering and exploring all grasp attempts with different pile configurations. However this work was strongly inspired from their approach of learning a grasp score function. We use same grasp pose representation and similar CNN architecture but trained on different dataset. In the contrary to their work, our learned base classifier is not limited to specific discretization in the gripper pose space in $2D$. Our CNN architecture allows us to query a given $3D$ grasp poses directly from the input image, which is an advantage over their method.

Chapter 3

Grasp Prediction

3.1 Problem Description

In this work, we are dealing with a bin of random pile, with parts lying in random positions, all have same shape, texture and color which makes it indistictive by RGB channels. In addition, the parts occlude with each other as can be seen in Figure 3.1. This makes Infra-Red and depth perception using SR300 RGBD camera essential for features based learning solution.

Figure 3.1: The bin with parts lying in a random pile when supplied after production.

The parts can be grasped by parallel-jaw gripper from 9 different edges, depending on the part posture in the bin Figure 3.2. Grasps prediction can be performed globally by predicting grasps from the whole input image, or locally over a single region of interest. In this work, we adopted the second approach due to its flexibility, in such manner the proposed algorithm can

be used in different applications depending on the scene and robot arm limitations in terms of reachability.

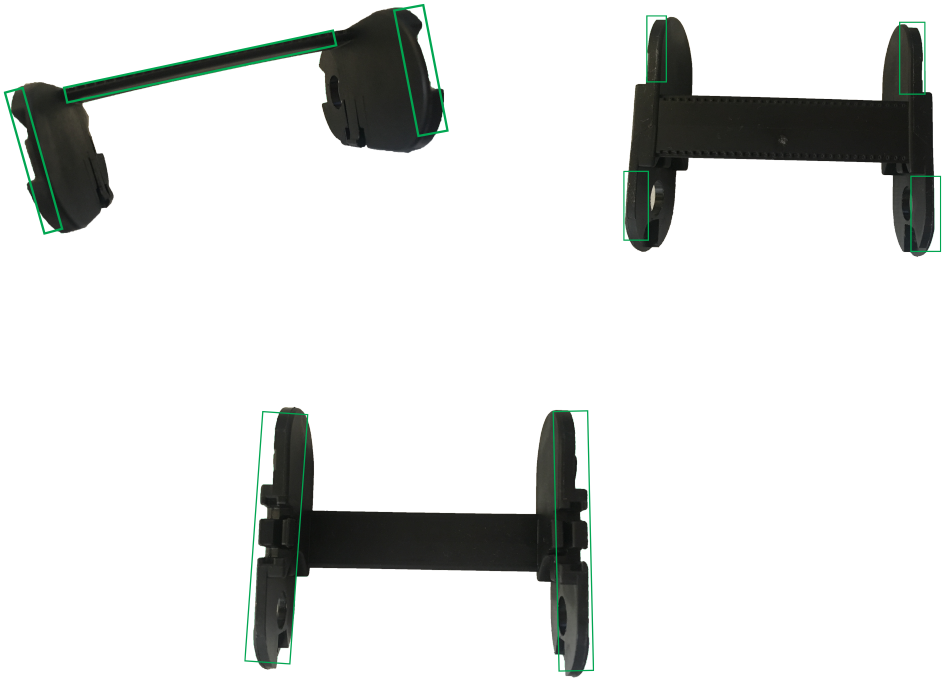


Figure 3.2: Figures above show the three different postures possibilities of the part in the bin, Green rectangles are bounding graspable regions with an orientation which is perpendicular to the longest edge.

Given the above, we needed to come up with a queriable algorithm which can classify graspability of grasp poses over IR-Depth image.Which can be formulated as the following:

Having I_{IR}, $\mathrm{I}_{\mathrm{Depth}}$ and grasp pose (x, y, θ) we need to find such an algorithm $\mathfrak{A}$:

$$\mathfrak{A}(\mathrm{I}_{\mathrm{IR}}, \mathrm{I}_{\mathrm{Depth}}, (x, y, \theta)) \rightarrow \text{Success or Failure}$$

3.2 Approach

The adopted approach is to train a CNN to classify grasp poses over IR-Depth images in a supervised learning manner. One challenge is acquiring a labeled dataset which represents real case scenarios. Thus, we need to collect a dataset and label it manually with positive and negative labels using dedicated annotator which is described in details in Section 3.4. The proposed CNN architecture has been constructed in a specific way as explained in Section 3.5 to receive beside the images a grasp pose as 3D vector which is fed directly to the Fully Connected Layer of the CNN.

3.2.1 Target Grasp Pose

This work focuses on the planar grasps only. A planar grasp is one where the grasp configuration is along and perpendicular to the workspace. Hence the grasp configuration lies in 3 dimensions grasp pose as (x, y, θ) illustrated in Figure 3.3, where (x, y) is corresponding to the gripper's center coordinates from left-top corner of the image, θ is the orientation of the gripper with the horizontal axis of the image plane. A transformation of the grasp pose from the image frame to the robot frame can be done in similar way used in [Koo et al., 2017].

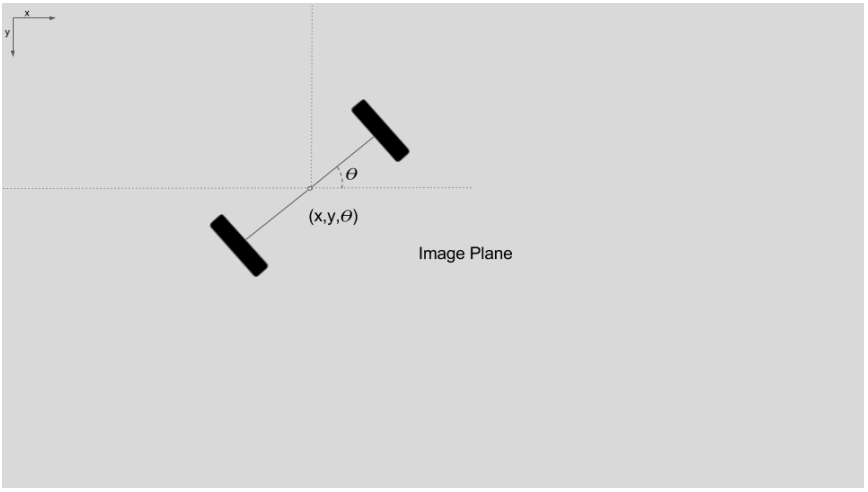


Figure 3.3: 3D Grasp Pose.

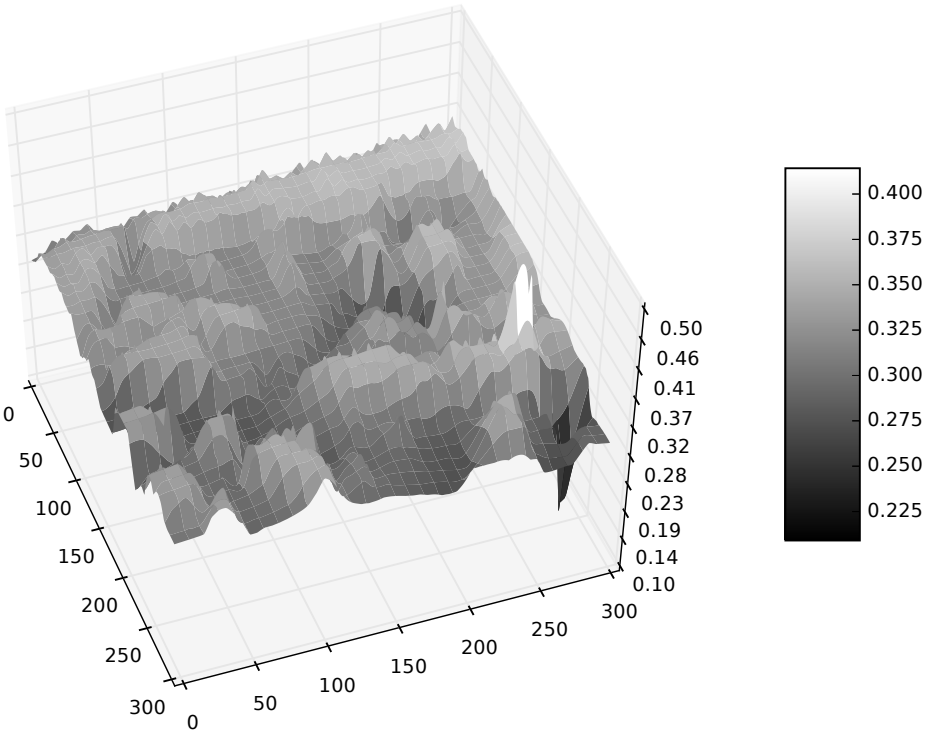


Figure 3.4: 3D visualization of the bin depth map.

3.2.2 Positive Grasp Pose

Given the depth map typology of width I_w and height I_h, graspable parts are graspable within a local minima for some region, which consist of a hill and two valleys from each side where the grippers plates can fit (see Figure 3.4). Having d as distance between the two plates, w, t as width and thickness of the plate, $d_{(x,y)}$ is depth value of the pixel at (x, y) and ϵ is the minimal depth required to be able to grasp the part. The idea becomes to observe whether the gripper plates can be placed on the depth map given the grasp center coordinates, orientation and plates dimensions.

For a grasp (x_0, y_0, θ), we will define the two rectangles corresponding to each of the grippers plates on the the x-y plane, denoted as $rect_1$ and $rect_2$. We will define first the set of pixels coordinates lying in a bounding box around (x, y) as the following:

$$\mathcal{B}(x,y) = \{(i,j) \in [0, I_w] \times [0, I_h] \mid |i - \lceil \frac{w}{2} \rceil| \leq x \text{ and } |j - \lceil \frac{t}{2} \rceil| \leq y\} \quad (3.1)$$

Having the following transformation function $\mathcal{T}$ of point (i,j) around (x,y) with orientation θ:

$$\mathcal{T}((i,j),\theta,(x,y)) \Longrightarrow (a,b) \quad (3.2)$$

$$a = \lceil (i-x)\cos\theta - (j-y)\sin\theta + x \rceil \quad (3.3)$$

$$b = \lceil (i-x)\sin\theta + (j-y)\cos\theta + y \rceil \quad (3.4)$$

The two rectangles defined as the following:

$$rect_1 = \{\mathcal{T}(i, j + \frac{d}{2}, \theta, x_0, y_0) \mid (i,j) \in \mathcal{B}(x_0, y_0)\} \quad (3.5)$$

$$rect_2 = \{\mathcal{T}(i, j - \frac{d}{2}, \theta, x_0, y_0) \mid (i,j) \in \mathcal{B}(x_0, y_0)\} \quad (3.6)$$

Given the grasp pose $g = (x_0, y_0, \theta)$ and depth $d_{(x_0,y_0)}$,
g is positive grasp iff

$$\forall (x,y) \in rect_1 \cup rect_2 \Rightarrow d_{(x_0,y_0)} \leq d_{(x,y)} - \epsilon \quad (3.7)$$

3.3 Dataset

Having dataset representing real scenarios from the target task is not trivial, taking into consideration the bin configuration variation during the picking process. The bin arrive from the bulk supply with different configuration of the parts, afterward every picking attempt can change the parts number and posture gradually, either by moving parts in case of failure or by removing parts after successful picking. It's not feasible to cover all cases, however the goal is to have representative dataset which the CNN can generalize from for the majority of the cases. This was achieved by capturing images of the bin after random shuffling of varying number of parts. Covering easy use cases, where the bin has low number of parts without occlusion, to more difficult

Distance between plates	2 cm
Width	2 cm
Thickness	0.5 cm
s	$\frac{1}{10\pi}$
ϵ	1 cm

Figure 3.5: Manipulator settings and parameters

cases where the parts number increase and occlude in a heap.

We used SR300 RGBD camera for collecting 100 images, each of size 640×480 , where camera is lying on the wrest of the robotic arm, with fixed height in such a way that it's orthogonal to the bin. Those images were served for labeling as described in Section 3.4.

In order to find positive grasp from images, we need to project gripper dimensions dimensions to the image plane in pixels. Given the gripper plates dimensions in the 3D sphere in Table 3.5, it can be done by estimating a mapping function $p : \mathbb{R}_+ \rightarrow \mathbb{R}_+$, using the depth image and extracting scaling parameter α. Knowing the part measurements, we have collected average depth measurements along one of the part edges and its the size in pixels from different images.
Given the pairs of depth and pixels measurements $[(d_0, p_0), .., (d_n, p_n)]$:

$$p(x) = \alpha \overline{d}_x \tag{3.8}$$

$$\alpha = \frac{p(x)}{\overline{d}_x} \rightarrow \hat{\alpha} = \sum_{i=0}^{n} \frac{p_i}{d_i} \tag{3.9}$$

$$\hat{p}(x) = \hat{\alpha} \overline{d}_x \tag{3.10}$$

$\overline{d}_x$ is the average depth value along x in the image, where x is the size in cm. This estimated function has served the algorithm for generating the ground truth based images for labeling purposes in the next section.

3.4 Labeling

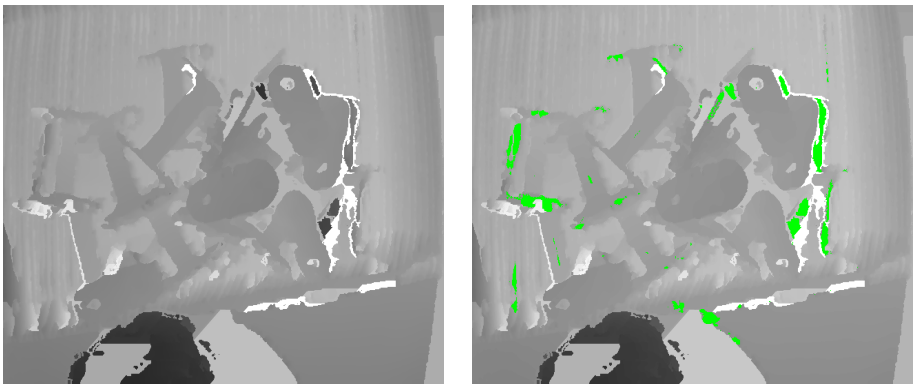
Labeling reliability is critical in order to exploit CNNs. Labeling strategy can determine the level of performance and generalization of the trained CNN on unseen scenarios. For grasp learning tasks, it becomes very challenging, since labeling is established over the images plane which unnecessarily reflect the outcome of the physical grasp. This could happen for many reasons e.g. the noise caused by the camera sensors and biased labeling caused by human errors, which is unavoidable in manual labeling.

We can overcome measurements errors to a certain degree by having large amount of data and having two kind of measurements like IR and depth. One can be considered when the other one is noisy. For reducing human error factor, a simple greedy algorithm was used to determine graspable pixels over the depth image, which was a base for the labeling process. Algorithm 3.1 traverse over all pixels of the synthetic depth image (see Figure 3.6a) and mark them as green, if it's positive grasp for some orientation. Since the parallel-jaw gripper is symmetric around the pivot, it's enough to traverse over $\theta \in [0, \pi]$, with discretization in the angle space using the parameter s. The output of the algorithm is a modified depth image, where green spots correspond to areas with high graspability likelihood as shown in Figure 3.6b.

Algorithm 3.1: Ground Truth Processor

```
1:  GTProcessor(I_depth, s)
2:      I_out ← I_Depth ;
3:      θ ← 0 ;
4:      for each pixel (x, y) in I_depth do
5:          while θ ≤ π do
6:              if IsPositive((x, y, θ), I_Depth) = TRUE then
7:                  I_out[x, y] = GREEN ;
8:                  break;
9:              end if
10:             θ ← θ + s ;
11:         end while
12:     end for
13:     return I_out ;
14: end
```

Having the output of Algorithm [3.1] alone is not enough for positive



(a) Synthestic depth image of a random pile.

(b) Output of the Algorithm 3.1 for $s = \frac{1}{10\pi}$, green areas indicate high liklihood of containing graspable edges.

Figure 3.6: Positive grasp ground truth processing using Algorithm 3.1 on Figure a.

grasp pose determination. Noisy measurement could affect the outcome of the algorithm by having false negative and false positive. At this point, a human interaction is used to determine true positive grasp poses and reduce false positive grasps. Given the original depth, IR images and the output of the algorithm, a person observes the three images and bound green regions using a rectangle shape. Graspable parts edges in the different images, correspond to these green regions. The bounding box has to be minimal and it's orientation determined by the green color spreading direction, while keeping in mind the possible graspable edges of the part in Figure 3.2 depending on its posture.

Beside positive labels, negative labels are required in supervised learning. Negative labels have as well a rectangle shape, it bounds areas on the image where it doesn't contain green pixels, or its orientation is biased significantly from the ground truth orientation. Negative labeling strategy covered both trivial and challenging cases. Trivial cases such as bin edges, areas which has no parts at all and part edges which is not one of the graspable edges in Figure 3.2. Challenging cases, where negative label is positioned near or between positive labels or it's bounding areas with green pixels, which from

the images it's clearly not graspable within any orientation.

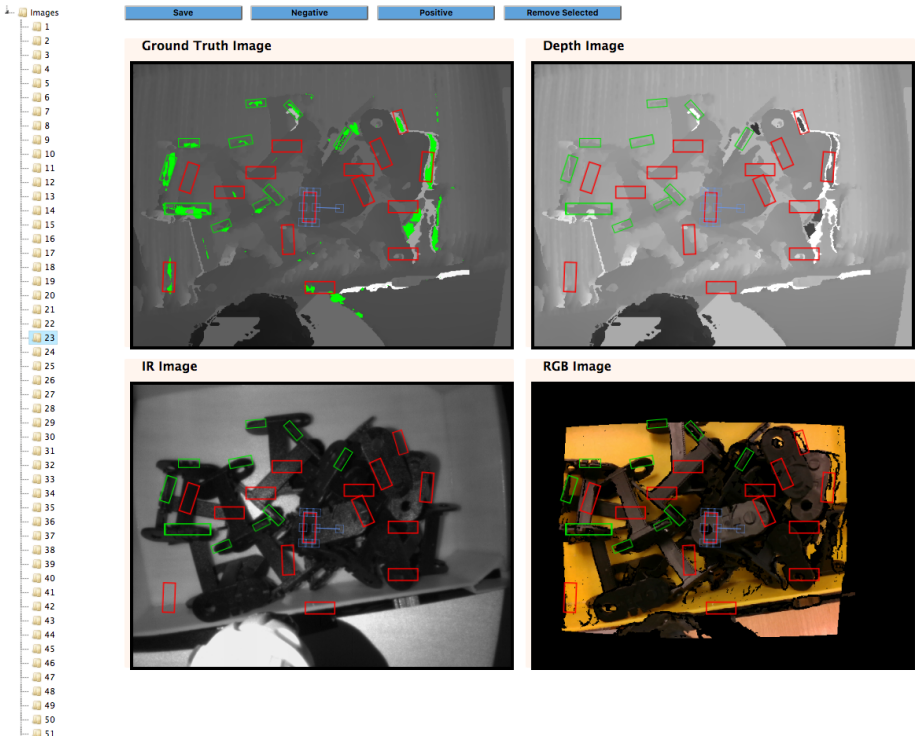


Figure 3.7: Web-based interface for labeling, using four synchronized HTML5 canvses for drawing the images and labels. With four different buttons for adding positive and negative labels on the canvases, saving and removing selected label.

This way we can manage to minimize the amount of false positive either by not including them in the positive labels or by surrounding them with negative labels. Moreover, we determine the orientation of the positive grasp as accurate as possible. In Figure 3.8 we have two examples of labeled images, green and red rectangles indicate for positive and negative labels respectively.

In order to annotate the images, a web based interface is built in JavaScript using Fabric.js library for drawing on canvases. The application loads the images names in a tree view and whenever a tree item is clicked, it loads and draws the four different images (IR, depth, RGB and depth with ground truth). The labels drawing is synchronized between all canvases, i.e any action on one of the canvases will be duplicated on other canvases. This feature

helps picking a clear view of the part in one of the images and make the labeling directly on it, then it's validated against the other images simultaneously.

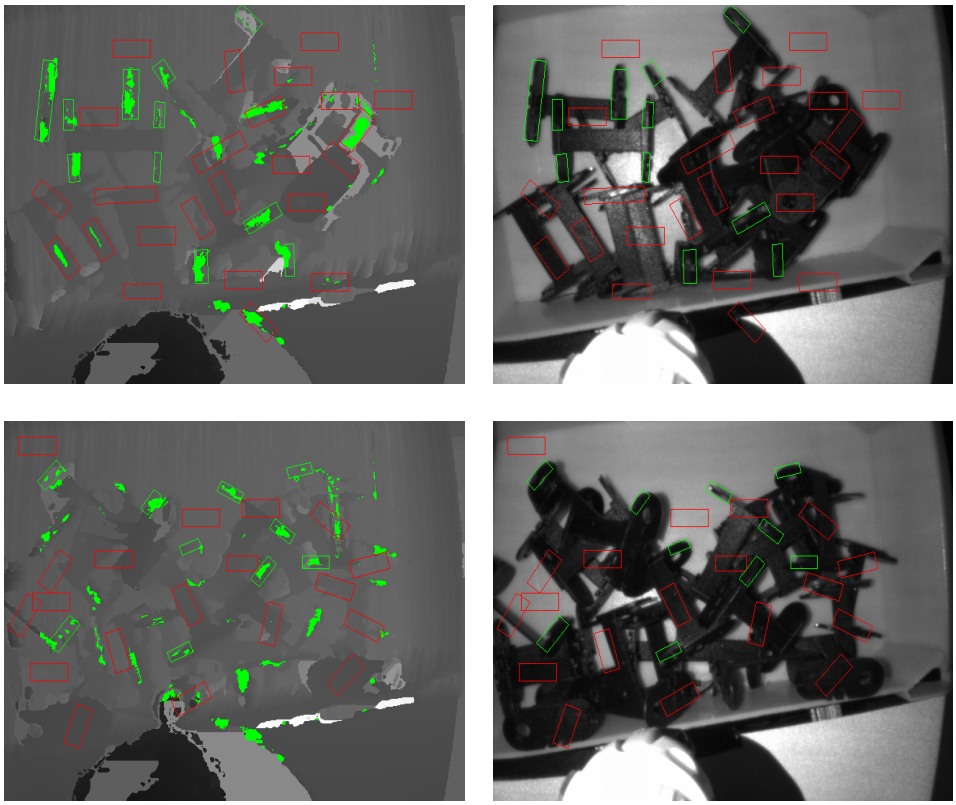


Figure 3.8: Two examples for labeled images. The figures on the left are the depth images with marked ground truth and on the right the IR images.

The user can draw green rectangles for positive grasps and red rectangle for negative grasps. These rectangles can be adjusted by size, position and orientation using dragging, rotating and resizing. The labels can be saved then into a JSON file as an array, each label is represented by four corner's coordinates and color.

3.4.1 Patch Size Calculation

In order to train the network on local regions, it requires extracting patches from the original image. The patch size is calculated as a function of the

depth value at the label's center and the projected gripper stroke size. This will allow the CNN to learn the features on small regions in the image and the it will be exposed to multi-scaled depth images. Having the stroke size s, estimated projection function 3.8 and valid depth value d, if the depth value is 0 at that point then we considered the average depth of larger receptive field (e.g. 7×7) at the label's center coordinates.

$$w = \hat{p}(2s) \tag{3.11}$$

In other words, the patch size is twice the projected gripper stroke size at label's center coordinates.

3.4.2 Region of Interest (ROI) Generation

From those labeled images, we are interested in Regions of Interest patches that contains single label regardless of its class. The idea is to center the label in a patch of size $w \times w$, then try to slide the patch in four directions left, bottom, top and right in the original image dimensions, while the label is still contained in the patch as illustrated in Figure 3.9. The reason behind this technique is mainly avoiding centered label patches, which limits the CNN to learn only centered grasp poses, when we are interested in learning classification of grasp poses over whole patch plane.

We should avoid having positive labels in the extreme edges of the patch, sense it doesn't contain enough surrounding information for determining the graspability of the label. However, in order to cover whole patch plane positions, for negative labels we won't use any safe margin, meaning that negative label can be located anywhere in the patch, while for positive labels we are more conservative, by giving safe margin enough to show the surrounding label's area.

We achieve that by using Algorithm 3.2, which receives as an input label's corners coordinates in the patch image plane, safe margin s, and it returns how much we can slide in the four directions, with a guarantee that the label is kept with margin of at least s from each of the image sides.

For every patch generation and after getting the margins, we randomly

Algorithm 3.2: Smart Margins Calculator

getSmartMargins(points, s)
  $minX \leftarrow min_x(points)$;
  $minY \leftarrow min_y(points)$;
  $maxX \leftarrow max_x(points)$;
  $maxY \leftarrow max_y(points)$;

  $top \leftarrow max(0, minY - s)$;
  $bottom \leftarrow max(0, patch_h - maxY - s)$;
  $left \leftarrow max(0, minX - s)$;
  $right \leftarrow max(0, patch_w - maxX - s)$;

  return top, left, bottom, right;
end

translate the labels center (x, y) within the range $[-left, right] \times [-top, bottom]$ using 3.4, the translated coordinates will serve as image crop center. Both depth and IR images are cropped using the algorithm 3.3, which guarantees the patch to stay within the original image frame, by sliding back overflowed patches and calculating actual crop center c_{new}. Then patches were resized to the CNN input size 224×224 and its corresponding label is translated and scaled accordingly.

Algorithm 3.3: Patch Cropper

getCroppedPatch(image, c)
  $c_{new} \leftarrow c$;

  $c_{new}[0] \leftarrow c_{new}[0] - min(c[0] - \frac{patch_w}{2}, 0)$;
  $c_{new}[0] \leftarrow c_{new}[0] - max(c[0] + \frac{patch_w}{2} - image_w, 0)$;
  $c_{new}[1] \leftarrow c_{new}[0] - min(c[1] - \frac{patch_h}{2}, 0)$
  $c_{new}[1] \leftarrow c_{new}[0] - max(c[1] + \frac{patch_h}{2} - image_h, 0)$;

  $croppedImage \leftarrow crop(image, c_{new}, patch_{size})$;
  return croppedImage, c_{new}
end

Algorithm 3.4: Random Patch Cropper

```
1:  getRandomPatch(I_Depth, I_IR, s, label)
2:      center ← label.center;
3:      top, left, bottom, right ← GetSmartMargins(label.points, s);
4:
5:      c'_x ← center_x + sample(−left, right);
6:      c'_y ← center_y + sample(−top, bottom);
7:
8:      cropped_depth, c_new ← GetCroppedPatch(I_Depth, c');
9:      cropped_ir, c_new ← GetCroppedPatch(I_IR, c');
10:
11:     label_new ← TranslateLabel(label, c_new);
12:
13:     return cropped_depth, cropped_ir, label_new;
14: end
```

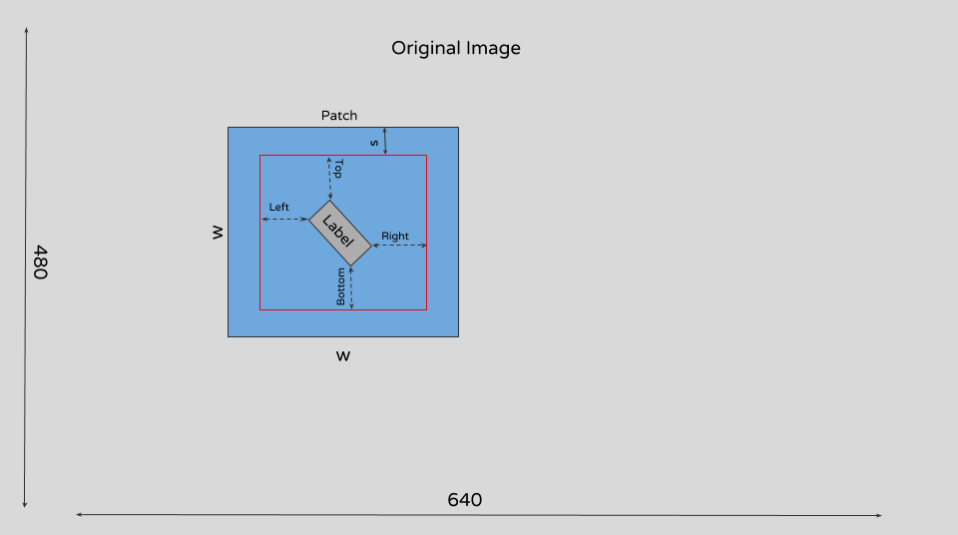


Figure 3.9: The figure demonstrates how to calculate margins from top, left, right and bottom of centered label given patch size $w \times w$ from image of size 640×480 and safe magin s.

3.4.3 Grasp Poses Generation

Grasp poses can be extracted easily given rectangle representation of the label. For negative label $\mathfrak{B}^-$ we can extract only negative grasp poses, which corresponds to all grasp poses contained in the label with any orientation

between $[0, \pi]$:

$$\text{Negative } = \{(x, y, \theta) | (x, y) \in \mathfrak{B}^- \text{ and } \theta \in [0, \pi]\}$$

For positive label $\mathfrak{B}^+$ with orientation ϕ and orientation error threshold ϵ, we can extract both positive and negative grasp poses in terms of orientation, as the following:

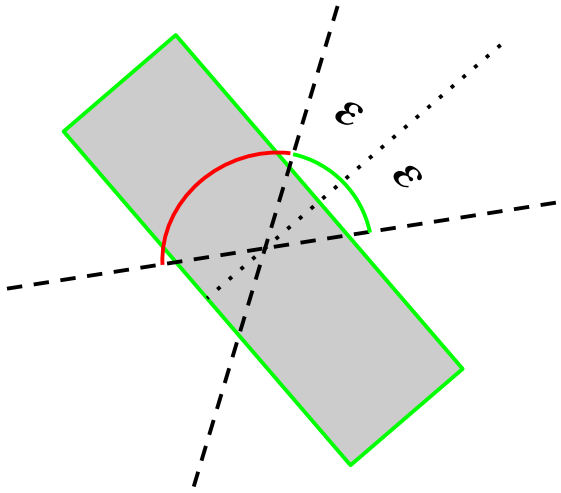


Figure 3.10: Positive and negative grasp pose orientation ranges for positive label, where the grasp pose coordinates are labels center. The dotted line shows the ground truth orientation which is perpendicular to the label, ϵ defines error magin from orientation ground truth.

$$\text{Positive } = \{(x, y, \theta) | (x, y) \in \mathfrak{B}^+ \text{ and } \theta \in [\phi + \frac{\pi}{2} - \epsilon, \phi + \frac{\pi}{2} + \epsilon]\}$$
$$\text{Negative } = \{(x, y, \theta) | (x, y) \in \mathfrak{B}^+ \text{ and } \theta \in [\phi, \phi + \frac{\pi}{2} - \epsilon] \vee [\phi + \frac{\pi}{2} + \epsilon, \phi + \pi]\}$$

Figure 3.10 illustrates the orientation ranges on positive label, for centered grasp poses. Similarly, we can generate grasp poses on any point contained in the label.

3.4.4 Training and Validation Datasets Generation

In order to decrease the variance between the images of same type either it's depth or IR, we normalized the images to fall between RGB scale $[0, 255]$. The labeled dataset contains in total 1849 labels, 660 of them are positive. We split the 100 labeled images in the dataset (see Figure 3.11) randomly into 85 images for training and 15 for validation.

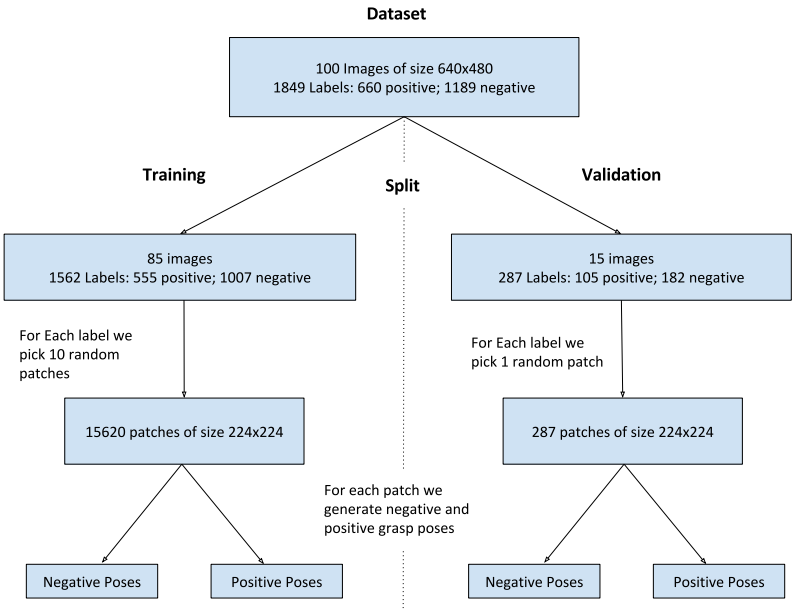
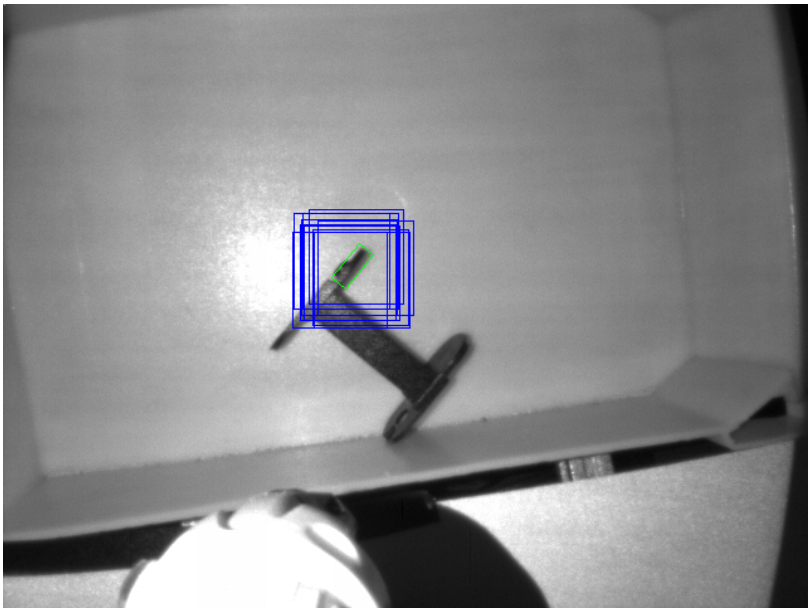
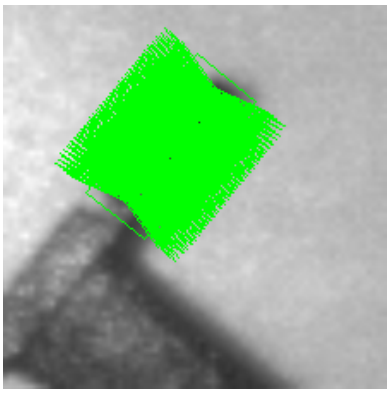


Figure 3.11: Ilustrating dataset splitting.

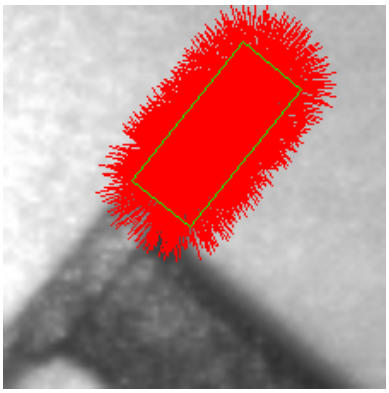
For each image we iterate through all labels and and generate ROI images using the Algorithm 3.4. From each label we generate 10 different patches (see Figure 3.12a) and a single patch otherwise. Having the scaled patch and label coordinates, we generate grasp poses as described in 3.4.3 with absolute orientation error threshold of $\frac{\pi}{12}$ (15 degrees) from the ground truth orientation, which is the orientation of the perpendicular to the labels longest edge from the horizontal axis. However, we discretized the translation and rotation space of grasp poses with 3 pixels and $\frac{\pi}{18}$ respectively, for tractability reasons. We have illustrated the generated grasp poses for positive label



(a) Example IR image containing single part, the green rectangle represent positive label and the blue rectangles are the different sampled ROIs before cropping and resizing. It has safe margin equal to 10% of its size, in order to include more context when training the network.



(b) Example for generated positive grasp poses



(c) Example for generated negative grasp poses

Figure 3.12: Figure a shows how we generate ROIs for positive labels, Figure b and Figure c illustrate patches with generated grasp poses after cropping and resizing one of the ROIs in Figure a. Grasp poses is represented as lines, where center and orientation of the line are the position and orientation of the grasp

over one of the generated patches in Figure 3.12c and Figure 3.12b.

3.5 CNN Architecture

Starting from a strong foundation for building the grasp classification and prediction system, will save lot of effort in exploring for proper CNN architecture. Where usually there is a trade off between accuracy and speed when choosing different CNN sizes, in terms of layers and parameters number. For training, we chose the network architecture illustrated in Figure 3.5, the network receive two inputs:

- IR and depth images as two channeled image
- 3D grasp pose (x, y, θ)

The architecture is very similar to the one used in [Johns et al., 2016], having same number of layers but using different Fully Connected layer setup. In contrary to their network which learns multi-scores classification over discretized space of coordinates in the image, we are rather interested only in binary classification. For feeding the second input into the network, our architecture is inspired from the method used in [Levine et al., 2016] for processing the motor command in a fully connected layer and integrate it with the results of the pooling. However, we used a simplified technique by concatenating the grasp pose to a flattened result of the last pool layer, before its processed using the fully connected layer.

The CNN input size is 224×224 with two channels for IR and depth values respectively, the images are processed with convolution layer with kernel size of 5 and stride of two pixels followed with max-pooling for reducing the input dimension. The next layer contains one convolution followed by max-pool, but using smaller kernel size of 3×3 for convolution. The last layer consist from 3 stacked convolutions with kernel size of 3×3, stride size of 1 followed by max-pooling layer. The result of the max-pooling is flattened and concatenated to the grasp pose, the result is served then as input to the three fully connected layers with $2D$ vector output, representing the scores of each of the two possible classes of grasp pose: Success and Failure. Each of the layers we applied batch normalizing and ReLU activation function.

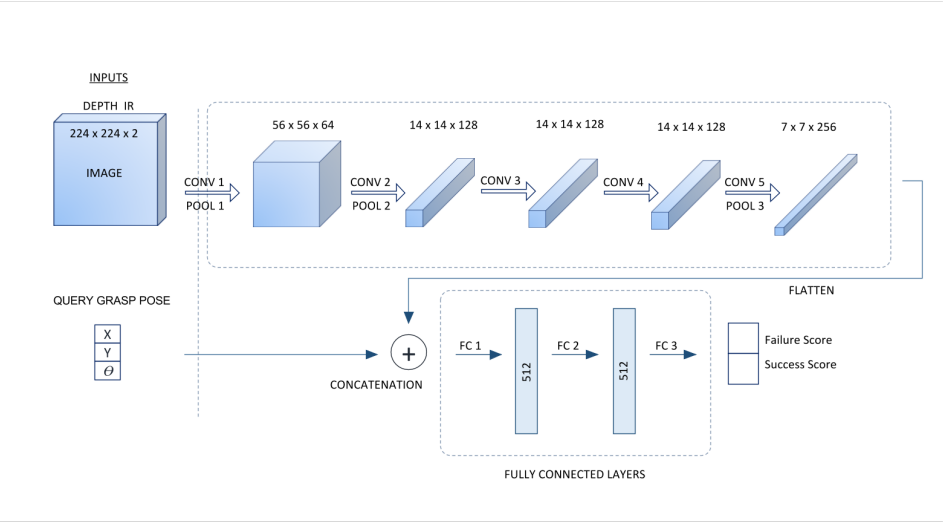


Figure 3.13: Training CNN architecture.

The network is implemented using tensorflow [Abadi et al., 2015], an open-source library developed initially by Google Brain engineers and researchers. The library provides wide functionality for conducting deep learning and machine learning research. It relies on concept of data flowing using graph representation for expressing computations, Where nodes represent the mathematical operations and edges are what so called tensors, represents multidimensional arrays. The architecture of tensorlfow is very flexible, different models can be served and evaluated simultaneously. One advantage of tensorflow is modularity, a model can be constructed from blocks or parts which can be used individually or with any desired combination, such feature can facilitate migration between models, e.g. between testing and training. The library is implemented in different programming languages and it can be easily configured to be deployed on CPUs and/or GPUs.

3.6 Training

3.6.1 Loss Function

The back propagation algorithm compute the gradients values, that are derived from implicit measure of the error function (Loss function). Hence,

choosing proper loss function can effect significantly weights calculation and the performance of the network. In our work, we use Softmax operation on top of the output layer which resolve confidence scores between 0 and 1, forming probability distribution along the classes number. One of the known loss functions is the MSE (Mean Square Error), which usually works better for regression than classification problems. When dealing with mutual exclusive classes (each entry is exactly in one class) problem, the more the result of the entries is close to 0 and 1, the gradient factor of MSE $(1 - v_{out})v_{out}$ get smaller and smaller resulting vanishing gradient. Cross-Entropy [Solla et al., 1988] is a good choice when dealing with proabibilties, it has proven better convergance on classification tasks and generally achieves better classification in terms of errors-rate [Golik et al., 2013].

Having two dimensional output of the network, we are interested in squashing its values into valid probability distribution using Softmax. For K dimensional vector output with values $o_1, .., o_K$, it calculates the probabilities vector $\sigma(o)$ as the following:

$$p_j = \sigma(o)_j = \frac{e^{o_j}}{\sum_{i=1}^{K} e^{o_i}} \text{ for } j = 1, .., K \tag{3.12}$$

Where $\sum_{j=1}^{K} p_j = 1$ and $0 \leq p_j \leq 1$. This is generalization of logistic regression, for two outputs o_1 and o_2, the probabilities can be written as the following:

$$p_1 = \frac{e^{o_1}}{e^{o_1} + e^{o_2}} = \frac{1}{1 + e^{o_2 - o_1}} \tag{3.13}$$

Having $p_i, (1 - p_i) \in \{0, 1\}$ the probabilities of some grasp pose on the i^{th} image and $\hat{p_i}, (1 - \hat{p_i}) \in [0, 1]$ its predicted probabilities as result of the softmax operation, we define the Cross-Entropy function as the following:

$$L_N = -\frac{1}{N} \sum_{i=1}^{N} p_i \cdot \log \hat{p_i} + (1 - p_i) \cdot \log (1 - \hat{p_i}) \tag{3.14}$$

The function is the average of Cross-Entropy between the real classes

and the output of the softmax layer, where N is the number of images in the batch.

3.6.2 Optimization Algorithm

Optimization algorithm is required in order to minimize or maximize some objective function w.r.t some parameters. Choosing optimization algorithm can impact the time for achieving good results. When the optimized function is differentiable w.r.t its parameters, it's relatively efficient to use gradient based algorithms. Since calculating partial derivatives has the same computational complexity as evaluating the function. There are many possible gradient based optimization algorithms one could use depending on the problem we are trying to solve [Ruder, 2016]. One of the commonly used algorithms is Adagrad [Duchi et al., 2011], which is well suited for sparse data since it adapts its learning rate by performing larger steps on infrequent parameters and smaller updates on frequent ones. Adadelta [Zeiler, 2012] is another optimization algorithm which is an extension of Adagrad. However, it monotonically decreases it's learning rate to reduce its aggressive. This is achieved by restricting the window of accumulated past calculated gradients, to a fixed size window, instead of accumulating all past squared gradients. The gradients are stored efficiently, by recursively defining the sum of the gradients as a decay average of all past squared gradients. In this work, we have used Adam (Adaptive Moment Estimation) optimizer [Kingma and Ba, 2014] which is recently adopted for multiple deep learning tasks. It computes adaptive learning rates for each parameter, in addition to storing an exponentially decaying average of past squared gradients like in Adadelta, Adam also stores an exponentially decaying average of past gradients:

$$m_t = \beta_1 m_{t-1} + (1 - \beta_1) g_t \tag{3.15}$$

$$v_t = \beta_2 v_{t-1} + (1 - \beta_2) g_t^2 \tag{3.16}$$

Where m_t is an estimate of the first moment, v_t is the estimate for second moment, g_t is the gradient at step t and β_1, β_2 are the moment

decays parameters. The biases were estimated :

$$\hat{m}_t = \frac{m_t}{1 - \beta_1^t} \tag{3.17}$$

$$\hat{v}_t = \frac{v_t}{1 - \beta_2^t} \tag{3.18}$$

Finally, they use these estimations for updating the parameters given learning rate η:

$$\theta_{t+1} = \theta_t - \frac{\eta}{\sqrt{\hat{v}_t} + \epsilon}\hat{m}_t \tag{3.19}$$

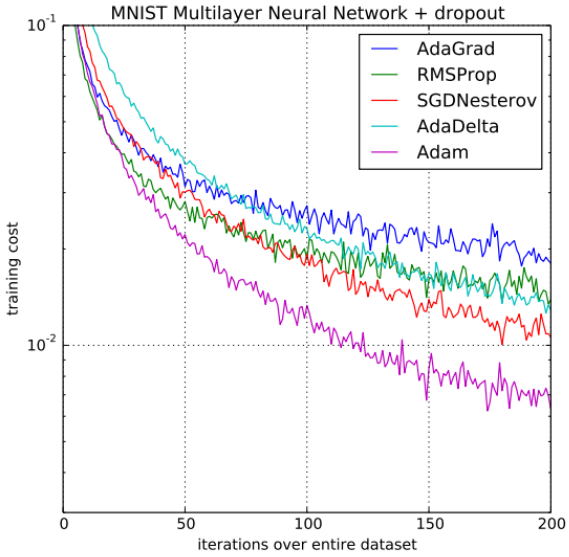


Figure 3.14: Convergence of Cross-Entopy cost function for multi-layer CNN using dropout stochastic regularization on MNIST dataset.

We used the recommended default values which are proposed in the paper and used by Tensorflow library:

β_1	0.9
β_2	0.999
ϵ	10^{-8}
η	10^{-3}

Adam optimizer has shown in Figure 3.14 better convergence than other optimization methods on MNIST dataset, where they used similar optimized function as the one we used for this work, which is the Cross-Entropy.

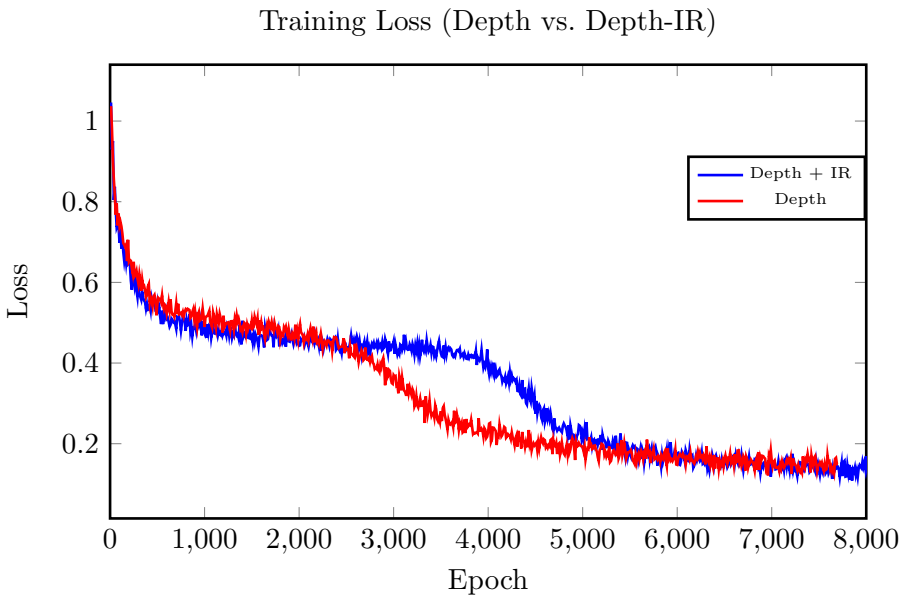
3.6.3 Training Strategy

The network is trained with a specific training strategy for avoiding overfitting on the training dataset. We used batch size of 200, which means at a time we serve the network 200 set of images and grasp poses corresponding to the images respectively. The batch is sampled from the training dataset randomly as the following:

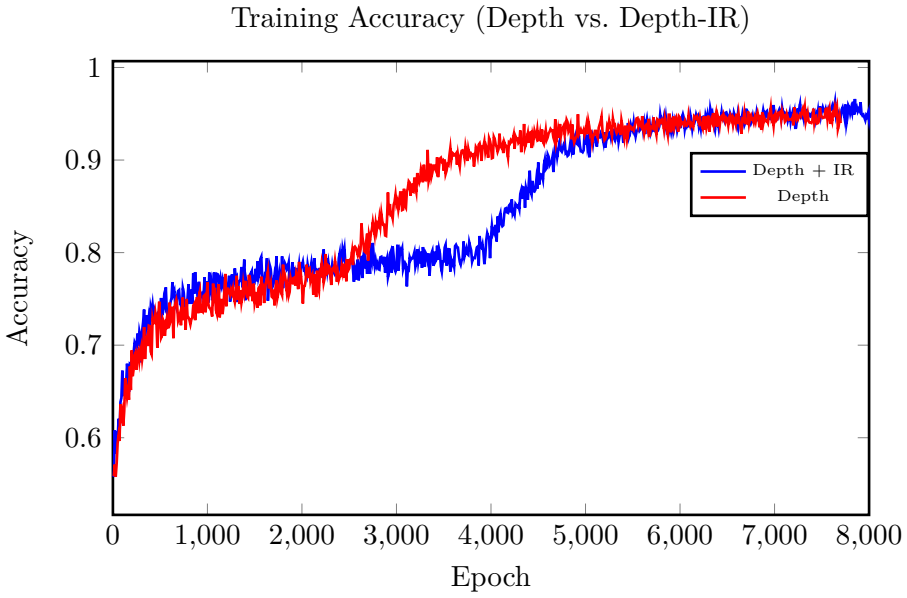
- Sample 200 different images from the training dataset
- For every image we flip a coin with equal chances, if the result is head we pick randomly positive grasp pose otherwise a negative grasp pose
- For each pair image and grasp pose we randomly pick rotation angle $\theta \in \{0, \frac{pi}{2}, \pi, \frac{3\pi}{2}\}$ and rotate both with θ
- We prepare one-hot encoded label l_{gt} to distinguish between the two labels, e.g. $[1, 0]$ or $[0, 1]$ for negative and positive grasp poses

Such batch sampling helps maintaining balanced amount of classes among the batch entries. Which is essential for classification tasks ifor avoiding biased predictions, specially when the classes are not equally distributed among the represented dataset similar to our problem, where the majority of the labels turned to be negative. In addition, we used random Dropout technique proposed in [Wager et al., 2013], over the last layer with 0.5 probability to reduce overfitting during the training, which has proved to achieve state-of-the-art results on many known datasets.

Weights initialization and learning rate determining become less critical due to the usage of batch normalization. We chose learning rate of value 0.001 and Adam optimizer for minimizing Cross-Entropy cost function which



(a) Illustrating loss function convergence for two models one using depth images only other one using Depth-IR images



(b) Illustrating batch accuracy score through the epochs.

Figure 3.15: Plotting loss and accuracy evaluation of single batch every 10 epochs.

was defined previously. We have trained two different models using the same CNN architecture first model using depth images only and the second model using both depth and IR images. For both models the loss function converged very well on the training dataset. In Figure 3.15a we can see that the first model converged faster than second model, one reason for this can be due to using smaller input channel size, which speeded up the learning process. Both models achieved similar accuracy rates on the batches toward the end of the training as we can see in Figure 3.15b.

Accuracy metric is calculated over a batch of size N, having the label l_{gt_i} and the predicted label l_{pred_i} for image i in the batch, as the following:

$$\text{Accuracy} = \frac{1}{N}\sum_{i=1}^{N} I_{correct}(l_{gt_i}, l_{pred_i}) \tag{3.20}$$

where

$$I_{correct}(l_{gt_i}, l_{pred_i}) = \begin{cases} 1 & \text{if } \underset{j\in\{0,1\}}{argmax}(l_{gt_i}) = \underset{j\in\{0,1\}}{argmax}(l_{pred_i}) \\ 0 & \text{otherwise.} \end{cases}$$

The equation above calculates the average of correctly predicted classes over the batch using the indicator $I_{correct}(l_{gt_i}, l_{pred_i})$, it has value 1 iff the predicted class of the grasp pose $grasp_i$ on $image_i$ is equal to its label class.

For evaluating the models performance during the training on unseen images, we sampled every 200 epochs a 20 batches from validation set in similar way to the training however without rotating the images. Having the accuracy measure alone, doesn't give sufficient indication on the classifier performance. Thus, we evaluated additional two measures: Precision and Recall. Similar to accuracy definition, we will define precision and recall metrics:

$$TP = \frac{1}{N}\sum_{i=1}^{N} I_{tp}(l_{gt_i}, l_{pred_i}) \tag{3.21}$$

$$FP = \frac{1}{N}\sum_{i=1}^{N} I_{fp}(l_{gt_i}, l_{pred_i}) \tag{3.22}$$

$$FN = \frac{1}{N}\sum_{i=1}^{N} I_{fn}(l_{gt_i}, l_{pred_i}) \tag{3.23}$$

where

$$I_{tp}(l_{gt_i}, l_{pred_i}) = \begin{cases} 1 & \text{if } \underset{j\in\{0,1\}}{argmax}(l_{gt_i}) = \underset{j\in\{0,1\}}{argmax}(l_{pred_i}) = 1 \\ 0 & \text{otherwise.} \end{cases}$$

$$I_{fp}(l_{gt_i}, l_{pred_i}) = \begin{cases} 1 & \text{if } \underset{j\in\{0,1\}}{argmax}(l_{gt_i}) = 0 \text{ and } \underset{j\in\{0,1\}}{argmax}(l_{pred_i}) = 1 \\ 0 & \text{otherwise.} \end{cases}$$

$$I_{fn}(l_{gt_i}, l_{pred_i}) = \begin{cases} 1 & \text{if } \underset{j\in\{0,1\}}{argmax}(l_{gt_i}) = 1 \text{ and } \underset{j\in\{0,1\}}{argmax}(l_{pred_i}) = 0 \\ 0 & \text{otherwise.} \end{cases}$$

Then the precision and recall metrics are calculated as follows:

$$Precision = \frac{TP}{TP + FP} \tag{3.24}$$

$$Recall = \frac{TP}{TP + FN} \tag{3.25}$$

Figure 3.16 shows the evaluated scores on two models, In Figure 3.16b we can see the evaluated scores when training the network on depth images. The trendline of the three scores is monotonically increasing, however it has negative peaks which shows that the network wasn't able to learn the grasp poses on some sampled images due to using of the depth channel only. Figure 3.16a shows the evaluated scores using Depth-IR images, the graphs is smoother and having less peaks. Clearly, we can notice that recall rates graph gained from using Depth-IR images is higher than precision. Conversely, when we used depth images only the recall maintained higher than precision and both relatively high. Which leads us to the conclusion that training the network on depth images indeed managed to reduce the False Positive rates over the grasp poses, but it missed many possible good

Table 3.1: Results summary of the two models over the positive labels in the validation dataset.

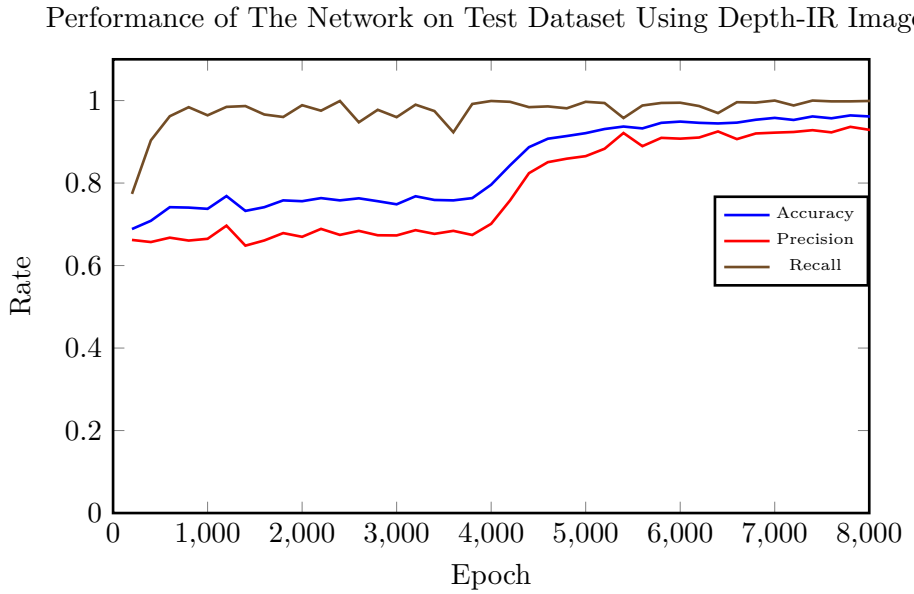
Model	TH. (≥)	Prec.	Rec.	Acc.	Error (Degrees)
Depth-IR	0.5	57%	84%	86%	39
	0.7	62%	84%	89%	31
	0.9	65%	84%	90%	30
Depth	0.5	58%	55%	87%	29
	0.7	60%	57%	87%	29
	0.9	61%	57%	88%	29

ones on the way in the contrary to the second model, which managed to reduce False Positive and the True Negative rates alike.

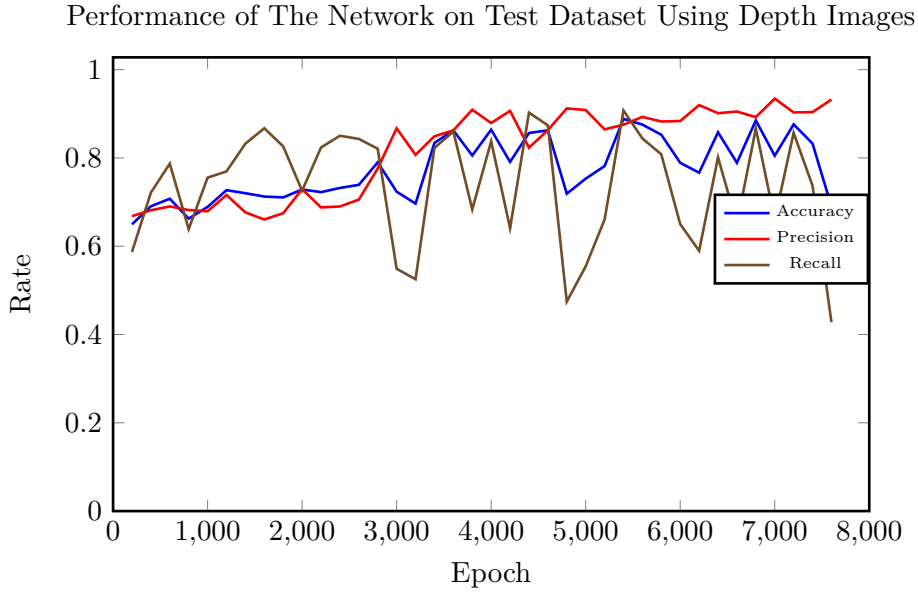
3.7 Empirical Results

In order to have some base line for comparison between the different models, we have evaluated them over all labels in the validation set. We evaluated each label class individually, since we generate from negative labels only negative grasp poses, precision and recall become meaningless, and calculating the accuracy would be sufficient for evaluating the performance. In the contrary, positive labels contain both classes of grasp poses, where grasp pose is negative in terms of orientation and not position (see Figure 3.10). Thus, we could evaluate in addition to precision and recall, the orientation error average from the ground truth orientation, over the false positive predictions. This provide an indication for the orientation accuracy of the models on the validation set.

Table 3.1 summarize the evaluation results over patches containing positive labels in the validation dataset. The scores are the average scores over the patches. It includes three rounds of evaluations of the two models, each round was evaluated with different threshold on the success confidence score from the networks' output. We simply consider the prediction to be negative if the success confidence score below some threshold p. For Depth-IR model, increasing the threshold maintained same recall rates, which means in average it didn't lose its sensitivity in classifying the orientation. However



(a) Evaluted scores through CNN training using Depth-IR images.



(b) Evaluted scores through CNN training using depth images

Figure 3.16: Evaluting precision, recall and accuracy through training every 200 epochs for 20 sampled batches from the validation dataset

Table 3.2: Accuracy summary of the two models over the negative labels in the validation dataset.

Model	Acc.
Depth-IR	97%
Depth	99%

accuracy and precision were increased and consequently, the orientation error of the false positive predictions has decreased. It achieved up to 65% precision with 84% recall and minimal orientation error of 30 degrees. In the contrary, the model trained on depth images, changing the threshold hasn't much affect on the evaluation results, however in average it scored lower results from the other model. It achieved up to 61% precision with relatively low recall up to 57% and orientation error of 29 degrees. It shows that in average, the trained model over Depth-IR images has learned to classify grasp poses, in terms of orientation, better than the model trained over depth images.

Table 3.2 summarize accuracy results over patches containing negative labels in the validation dataset. The model trained on depth images was able to classify negative grasp poses better than the other model, 99% of precision compared to 97% when training on Depth-IR images. Which shows that using the additional IR channel could increase in average the amount of false negative grasp poses for non-graspable regions.

3.8 Enhanced CNN Architecture

One drawback of the architecture in Figure 3.13, it requires processing the image for every single grasp pose evaluation, which is inefficient, specially when trying to iterate over the grasp poses space on the same image. Instead we would like to allow evaluation of batches of grasp poses without the cost of unnecessary processing of the image on every evaluation. The idea is to change the architecture in such a way, it receives as input a batch of grasp poses to evaluate on single image. This can be achieved easily using

Tensorflow library, by tiling the last Pool layer as the number of grasp poses in the batch, then each grasp is concatenated to the flattened result of the Pool and processed in the fully connected layer similar to the previous architecture. This increased significantly the time required to evaluate multiple grasp poses, since tiling operation is relatively faster than to re-processing the image in the convolutions layers.

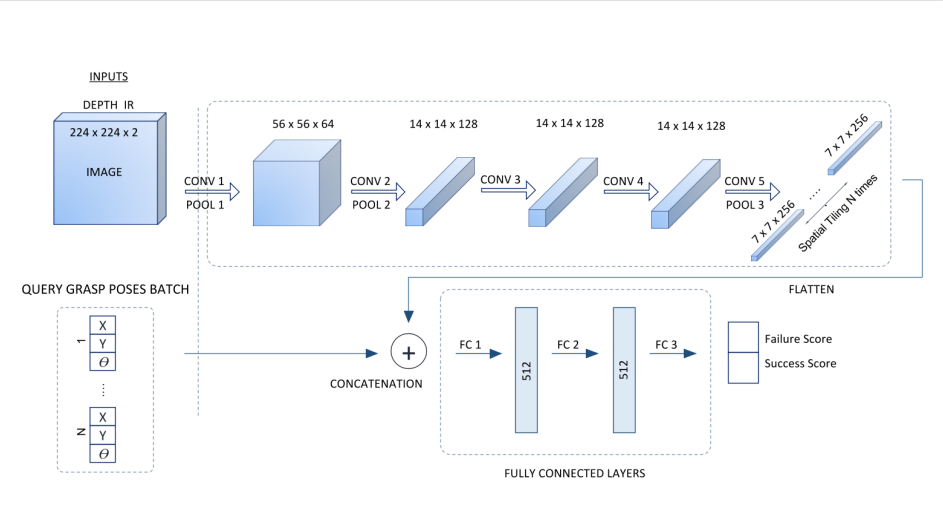


Figure 3.17: CNN architecture for validaition.

3.9 Multiple Grasp Pose Prediction

Our approach provide high level of flexibility, since we don't need to process the whole image in order to predict grasp poses. It can be used to predict/-classify local regions (Regions of Interest) on the image, which is very useful for robotic applications. For example, it can be adapted to the robotic arm configuration by ignoring regions on the image where the arm can't reach. Alternatively, it can use different heuristics for identifying regions on the image with high likelihood of containing graspable parts, and then use our approach to classify or predict grasp pose candidates.

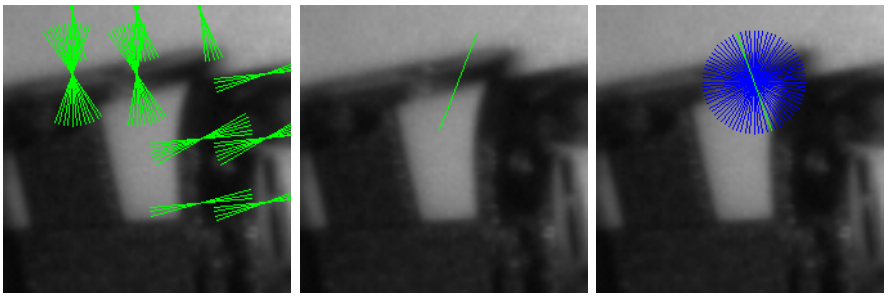
In this section, we will provide one possible heuristic which uses the classifier for predicting multiple grasp poses over the image. The idea is to divide the sensed images into smaller patches images to match the CNN

input size using sliding window approach. For each patch we traverse over the gripper $3D$ poses space and predict a single high quality grasp pose.

Having the sensed images using the camera, we generate fixed ROI-s centers over the image plane, for each center we crop and resize a patch using the method described in Section 3.4.2, resulting set of images patches. Then we generate uniformly grasp poses over the 224×224 patch with certain descretizaiton in x,y and θ, which will be evaluated using our trained models. The outcome of the classification the trained network, may contain false positive grasp poses as shown in Figure 3.18a, however those usually are near some graspable regions with inaccurate grasp position. In order to eliminate its effect, we could use couple of methods like neighbor clustering. For each cluster we calculate it's average orientation, then grasp poses are determined by the clusters centers and its average orientation. However, we used a simpler method by calculating the average over the positively classified grasp poses to yield a single robust predicted grasp pose. When all grasp poses candidates are negatively classified, it indicates that the patch doesn't include graspable parts. Otherwise, we will have set of N positively classified $3D$ grasp poses $pos_1, ..., pos_N$ over some patch, we calculate the average pose pos_{avg}:

$$pos_{avg} = (x_{avg}, y_{avg}, \theta_{avg}) = \frac{1}{N} \sum_{i=0}^{N} pos_i \tag{3.26}$$

Average grasp can be unreliable, specially when a patch contains multiple graspable parts, that can be grasped in different orientations (see Figure 3.18a), the average grasp would be then lying somewhere in between the graspable parts. Another case, when the part is graspable at the average grasp position (x_{avg}, y_{avg}), but with orientation different than the average grasp orientation θ_{avg} as shown in Figure 3.18b. These issues can be solved by generating a new set of poses candidates $\{(x_{avg}, y_{avg}, \theta_{new}) | \theta_{new} \in [0, \pi]\}$ illustrated in Figure 3.18c, which comes as a step for reconsidering new orientations on the average grasp position. The set is re-evaluated as a single batch using the network, we pick then grasp pose with the highest success score. This would allow us re-evaluating pos_{avg}, without missing a potential



(a) Positive predictions **(b)** Average grasp pose **(c)** Best grasp pose

Figure 3.18: Illustrates the outcome of three steps over the patch. Figure a illustrates the positive grasp poses predictions; Figure b shows the average grasp pose pos_{avg}; Figure c illustrates the proposed new grasp poses (blue lines) having the same centers' position as pos_{avg}, but with different orientations. The grasp pose with highest success score after re-evaluation is colored as green line.

good grasp for having biased average orientation θ_{avg}.

The choice in the granularity of discretization either in generating ROIs centers from the image space or in the grasp pose generation, can be a compromise between precision in the target pose predictions, and tractability of the application, specially when applying it in real-world experiments. Choosing coarser scale lead to faster evaluation, since we will have less number of patches and grasp poses. However, it could come with a price of yielding poor predictions and missing potential graspable parts.

We have applied the proposed heuristic over the images on the validation set. Given the images of size 640×480 , we sampled fixed ROI-s centers over the image plane with 40 pixels distance between them, with padding of 20 pixels from the image sides. Then we have generated uniformly grasp poses over the patch image by discretization of 50 pixels in position and 5 degrees in rotation. The task now becomes to predict, from each of the 165 patches, a single high quality grasp pose from the 900 possible grasp poses. We have used the modified CNN architecture in Figure 3.17, which allows us to evaluate 900 grasp poses at once. The final predicted grasp pose on a patch was translated back into the original image coordinates.

We chose to illustrate the results on IR images over depth images because it provides clearer view of the parts, the predicted grasp poses were repre-

sented as green lines to indicate its position (lines center) and orientation. Figure 3.19 shows the predicted grasps (green lines) on the IR image using IR-Depth trained model, but without using the re-evaluation step described above. We can see some of the grasp poses are lying on some graspable part (for example the left-top part in the image), but with wrong orientation, those grasp pose predictions can be misclassified in the re-evaluation step, without reconsidering the new orientations. In Figure 3.20, it shows the predicted grasps using the same model and image with the re-evaluation step. It managed to correct orientation for part of the predicted grasp poses, however it contains few false positive grasp poses, like the ones on the bin edges and on the table. It was possible to minimize those, by increasing the threshold over the predicted success score to 0.7, as shown in Figure 3.21. But it didn't have noticeable change in the calculation of the average grasp orientation, due to the fact that orientations are symmetric around its ground truth, for positively predicted grasp poses. Thus, the average has kept more or less the same after filtering them. Figure 3.6a shows predicted grasp poses using the depth model on the very same scene. The image has less false positive grasp poses, however it missed many good grasp poses. This is explainable from looking at the performance results on validation set, the depth model had relatively low recall, and it tends to miss positive grasp poses.

Table 3.3 summarize the average time during evaluation. We can see the it takes an average of 0.138 seconds to evaluate the grasp poses over a single patch without re evaluation, 0.28 seconds with re-evaluation, since we need to do an extra processing on the predictions and then evaluate another 36 new grasp poses, however the re-evaluation step would occur only if we have positive predictions over a certain patch. This explains why it takes around 28.3 seconds to evaluate 165 patches, which is less than the average time obtained from multiplying number of patches with the average time required per patch with re-evaluation.

Unfortunately, we couldn't evaluate the performance of the proposed heuristic quantitatively, since the dataset was partially labeled and it makes it difficult in determining the ground truth of predicted grasp poses. We hope in the future we will be able to evaluate this heuristic using robot experiments.

Table 3.3: This table summarize the average time needed to evaluate 900 grasp poses, served as a single batch using the CNN illustrated in Figure 3.17 using IR-Depth model, running on 4 GeForce® GTX TITAN Black GPUs. We split the time evaluation into 3 categories: single patch without re-evaluation step, single patch with evaluation step and the time required for evaluation the poses on 165 patches.

	Time (average)
Single Patch (without re-evaluation)	0.138 seconds
Single Patch (with re-evaluation)	0.28 seconds
165 Patches (with re-evaluation)	28.3 seconds

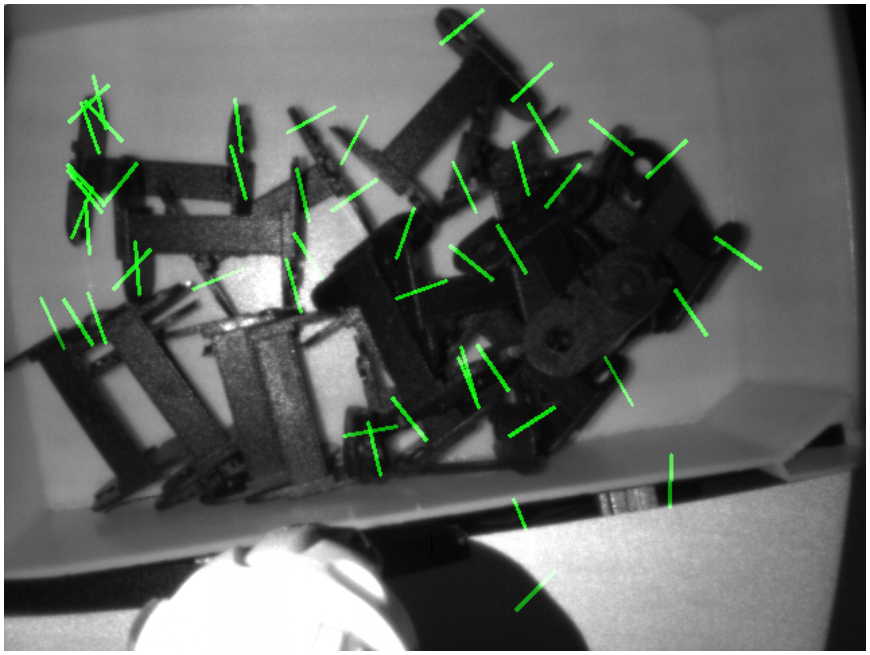


Figure 3.19: Multigrasp predictions using the average grasp without re-evaluating step using IR-Depth model.

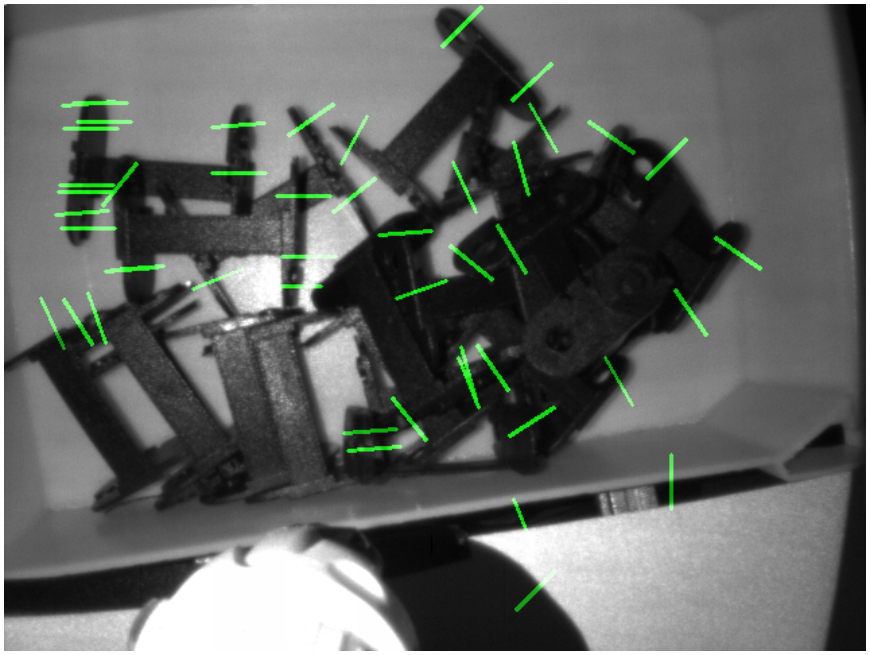


Figure 3.20: Multigrasp predictions using the average grasp after re-evaluating step using IR-Depth model

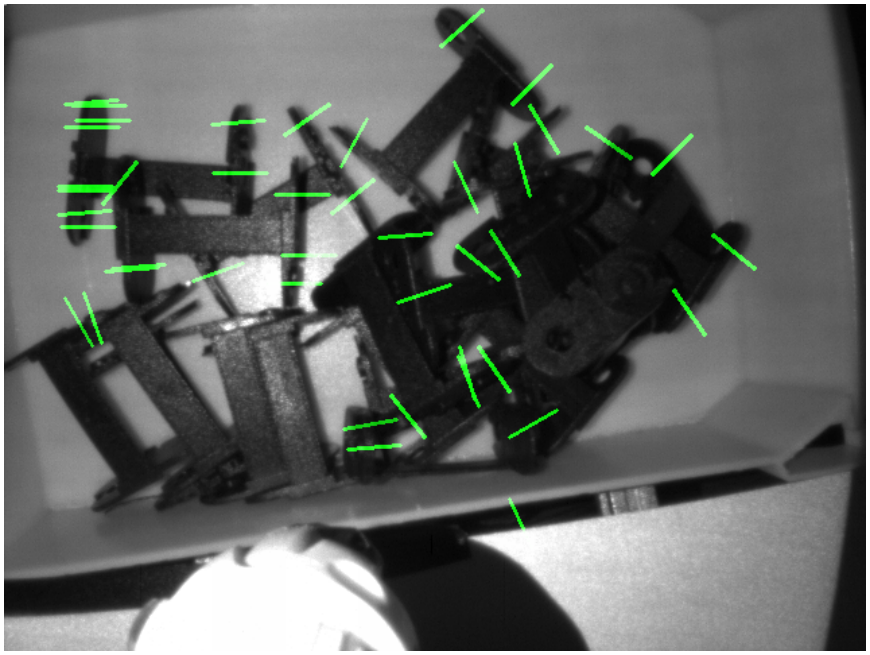


Figure 3.21: Multigrasp predictions using the average grasp after re-evaluating step using IR-Depth model, and applying 0.7 threshhold on success score

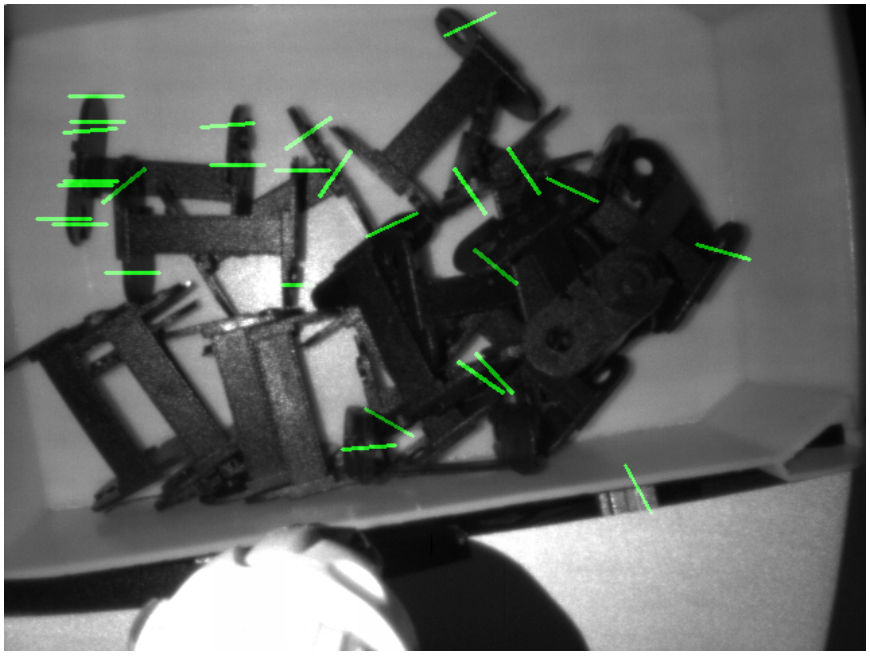


Figure 3.22: Multigrasp predictions using the average grasp after re-evaluating step using Depth model

Chapter 4

Outlook

4.1 Summary

In this work, we presented a state-of-the-art method for querying $3D$ grasp poses graspability over IR-Depth images using deep learning. Our contribution is a learned base grasp pose classifier, that works on local regions in the image. We have shown how to generate large dataset of patches and grasp poses from a relatively small set of images. This dataset has served for training two models, one using only depth images and the second using IR-Depth images. Training the CNN with IR images in addition to depth, has improved the overall grasp poses classification. This classifier provides flexible solution, it can be used in different heuristic either for classifying or predicting grasp poses. One example is our proposed multi-grasp prediction heuristic, which uses our classifier for multiple grasp predictions over the entire image. Our multi-grasp prediction heuristic uses an enhanced CNN architecture to serve large batch of grasp poses at once efficiently. With average grasp over uniformly sampled grasp poses from the patch plane, it is able to identify non-graspable regions with few false positive predicted grasp poses. Additionally, it provides a robust high quality predicted grasp poses thanks to the re-evaluation step.

4.2 Further Work

Our multi-grasp prediction heuristic has the ability to predict multiple grasps over the entire image. One drawback of this heuristic, is that it uses a sliding window approach to traverse over local regions, this can be speeded up by focusing only on regions with high likelihood of containing graspable parts. We also used average grasp pose over the patch to determine high quality grasp, which was re-evaluated using the classifier, for determining its graspability. Instead, neighbor clustering can be used, in turn average grasp poses over clusters would yield multiple grasps predictions instead of one, and it will fulfill the need for re-evaluation step. There is more work to be done on filtering the predicted grasp poses, and choosing the best one. For example, best grasp can correspond the grasp pose with minimal depth value. We couldn't quantitatively evaluate our multi-grasp prediction heuristic, because our dataset is partially labeled. Thus, further robot experiments are required for proper evaluation. For future work, we will compare the performance of our multi-grasp prediction heuristic with the method used in [Koo et al., 2017] for grasp detection. Moreover, we will work on integrating our classifier with their method, it can be used for providing continuous feedback over local regions, while the manipulator approaches the part, for determining the grasp pose in terms of position and orientation.

Bibliography

Abadi, Martín et al. (2015). *TensorFlow: Large-Scale Machine Learning on Heterogeneous Systems*. Software available from tensorflow.org. URL: http://tensorflow.org/.

Agostinelli, Forest et al. (2014). "Learning activation functions to improve deep neural networks". In: *arXiv preprint arXiv:1412.6830*.

Angelova, Anelia et al. (2015). "Real-Time Pedestrian Detection with Deep Network Cascades". In: *Proceedings of the British Machine Vision Conference 2015, BMVC 2015, Swansea, UK, September 7-10, 2015*, pp. 32.1–32.12. DOI: 10.5244/C.29.32. URL: http://dx.doi.org/10.5244/C.29.32.

Boularias, Abdeslam et al. (2015). "Learning to Manipulate Unknown Objects in Clutter by Reinforcement". In: *Proceedings of the Twenty-Ninth AAAI Conference on Artificial Intelligence, January 25-30, 2015, Austin, Texas, USA*. Pp. 1336–1342.

Cord, Matthieu (2016). *Deep CNN and Weak Supervision Learning for visual recognition*. URL: https://blog.heuritech.com/2016/02/29/a-brief-report-of-the-heuritech-deep-learning-meetup-5/.

Cun, Yann Le et al. (1991). "Eigenvalues of covariance matrices: Application to neural-network learning". In: *Physical Review Letters* 66.18, pp. 2396–2399. ISSN: 0031-9007. DOI: 10.1103/PhysRevLett.66.2396.

Duchi, John et al. (2011). "Adaptive subgradient methods for online learning and stochastic optimization". In: *Journal of Machine Learning Research* 12.Jul, pp. 2121–2159.

Golik, Pavel et al. (2013). "Cross-entropy vs. squared error training: a theoretical and experimental comparison." In: *Interspeech*. Vol. 13, pp. 1756–1760.

Goodfellow, Ian et al. (2016). "Deep Learning". In: http://www.deeplearningbook.org. MIT Press, pp. 1–2.

He, Kaiming et al. (2015). "Deep Residual Learning for Image Recognition". In: *CoRR* abs/1512.03385. URL: http://arxiv.org/abs/1512.03385.

Ioffe, Sergey and Christian Szegedy (2015). "Batch normalization: Accelerating deep network training by reducing internal covariate shift". In: *arXiv preprint arXiv:1502.03167*.

Jiang, Yun et al. (2011). "Efficient grasping from RGBD images: Learning using a new rectangle representation". In: *IEEE International Conference on Robotics and Automation, ICRA 2011, Shanghai, China, 9-13 May 2011*, pp. 3304–3311. DOI: 10.1109/ICRA.2011.5980145. URL: http://dx.doi.org/10.1109/ICRA.2011.5980145.

Johns, Edward et al. (2016). "Deep Learning a Grasp Function for Grasping under Gripper Pose Uncertainty". In: *CoRR* abs/1608.02239. URL: http://arxiv.org/abs/1608.02239.

Kingma, Diederik P. and Jimmy Ba (2014). "Adam: A Method for Stochastic Optimization". In: *CoRR* abs/1412.6980. URL: http://arxiv.org/abs/1412.6980.

Koo, Seongyong et al. (2017). "Robolink Feeder: Reconfigurable Bin-Picking and Feeding with a Lightweight Cable-Driven Manipulator". In: *13th IEEE International Conference on Automation Science and Engineering (CASE), Xi'an, China, August 2017.*

Kopicki, Marek Sewer et al. (2016). "One-shot learning and generation of dexterous grasps for novel objects". In: *I. J. Robotics Res.* 35.8, pp. 959–976.

Krizhevsky, Alex et al. (2012a). "ImageNet Classification with Deep Convolutional Neural Networks". In: *Advances in Neural Information Processing Systems 25: 26th Annual Conference on Neural Information Processing Systems 2012. Proceedings of a meeting held December 3-6, 2012, Lake Tahoe, Nevada, United States.* Pp. 1106–1114. URL: http://papers.nips.cc/paper/4824-imagenet-classification-with-deep-convolutional-neural-networks.

Krizhevsky, Alex et al. (2012b). "ImageNet Classification with Deep Convolutional Neural Networks". In: ed. by F. Pereira et al., pp. 1097–1105.

URL: http://papers.nips.cc/paper/4824-imagenet-classification-with-deep-convolutional-neural-networks.pdf.

Kumra, Sulabh and Christopher Kanan (2016). "Robotic Grasp Detection using Deep Convolutional Neural Networks". In: *CoRR* abs/1611.08036. URL: http://arxiv.org/abs/1611.08036.

Le, Quoc V et al. (2010). "Learning to grasp objects with multiple contact points". In: *Robotics and Automation (ICRA), 2010 IEEE International Conference on*. IEEE, pp. 5062–5069.

LeCun, Y. et al. (2001). "Gradient-Based Learning Applied to Document Recognition". In: *Intelligent Signal Processing*. IEEE Press, pp. 306–351.

LeCun, Yann and Corinna Cortes (1998). *The MNIST database of handwritten digits*.

Lenz, Ian et al. (2015). "Deep learning for detecting robotic grasps". In: *I. J. Robotics Res.* 34.4-5, pp. 705–724. DOI: 10.1177/0278364914549607. URL: http://dx.doi.org/10.1177/0278364914549607.

Levine, Sergey et al. (2016). "Learning Hand-Eye Coordination for Robotic Grasping with Deep Learning and Large-Scale Data Collection". In: *CoRR* abs/1603.02199. URL: http://arxiv.org/abs/1603.02199.

Nair, Vinod and Geoffrey E Hinton (2010). "Rectified linear units improve restricted boltzmann machines". In: *Proceedings of the 27th international conference on machine learning (ICML-10)*, pp. 807–814.

Pinto, Lerrel and Abhinav Gupta (2016). "Supersizing self-supervision: Learning to grasp from 50K tries and 700 robot hours". In: *2016 IEEE International Conference on Robotics and Automation, ICRA 2016, Stockholm, Sweden, May 16-21, 2016*, pp. 3406–3413. DOI: 10.1109/ICRA.2016.7487517. URL: http://dx.doi.org/10.1109/ICRA.2016.7487517.

Ruder, Sebastian (2016). "An overview of gradient descent optimization algorithms". In: *CoRR* abs/1609.04747. URL: http://arxiv.org/abs/1609.04747.

Russakovsky, Olga et al. (2015). "ImageNet Large Scale Visual Recognition Challenge". In: *International Journal of Computer Vision (IJCV)* 115.3, pp. 211–252. DOI: 10.1007/s11263-015-0816-y.

Saxena, Ashutosh et al. (2008). "Robotic grasping of novel objects using vision". In: *The International Journal of Robotics Research* 27.2, pp. 157–173.

Simonyan, Karen and Andrew Zisserman (2014). "Very Deep Convolutional Networks for Large-Scale Image Recognition". In: *CoRR* abs/1409.1556. URL: http://arxiv.org/abs/1409.1556.

Solla, SA et al. (1988). "ªAccelerated Learning in Layered Neural Networks, º Complex Systems, vol. 2". In:

Srivastava, Rupesh Kumar et al. (2015). "Highway Networks". In: *CoRR* abs/1505.00387. URL: http://arxiv.org/abs/1505.00387.

Szegedy, Christian et al. (2014). "Going Deeper with Convolutions". In: *CoRR* abs/1409.4842. URL: http://arxiv.org/abs/1409.4842.

Wager, Stefan et al. (2013). "Dropout training as adaptive regularization". In: *Advances in neural information processing systems*, pp. 351–359.

Zeiler, Matthew D. (2012). "ADADELTA: An Adaptive Learning Rate Method". In: *CoRR* abs/1212.5701. URL: http://arxiv.org/abs/1212.5701.