GRASP

Ching-an Wu
Andreas Rosenback

cawu@kth.se
rosenba@kth.se
KTH Royal Institute of Technology

Abstract. The aim of this project is to implement a Convolutional Neural Network (CNN) to be function as an evaluation function which can be used to predict the success of a robot grip. The reason to use a CNN instead of the original evaluation function is that the evaluation time can be improved. In this project, we construct two convolutional neural networks. One is a vanilla CNN, and the other is inspired by the GQ-CNN. Both models are trained on a synthetic dataset.

Keywords: CNN, robot grip, GQ-CNN

1 Introduction

In industry, robot grasping is a high-demanding task due to its generic applications. However, it requires many aspects of techniques to solve such a problem, such as the control of the robot arm and the sensing of the target objects to be grasped. Regarding the sensing of different objects, the traditional object recognition methods often require long computation time which is hard to use in real time. Recent work has shown that deep neural network can be used in planning the grip strategy after being trained on the real world images or even the synthetic point cloud data[1].

In this project, we are aiming to approximate an evaluation function with a convolutional neural network model. The dataset that is used has two classes (cage relevant and cage irrelevant) that are heavily unevenly distributed. Questions we ask ourselves include: How to augment the dataset to make learning more efficient? How to deal with heavily imbalanced data? After two different network models are built, how to assess different architectures? Why is one better than the other?

2 Background

The outcome of the task of grasping an object for a robot can, in the most basic case, be divided into two classes. Either it succeeds with its grasp or it fails with it. These two classes are denoted as cage relevant and cage irrelevant where cage relevant is the successful case and cage irrelevant is the unsuccessful one.

A convolutional neural network is a type of neural network and is described in the book Deep Learning[2]. This kind of neural network uses different different kind of mathematical operations in different layers on the input data. The different layers that are commonly used are pooling layers, convolutional layers, fully connected layers and the input can for example be the pixels of an image as in this project.

There are many different kinds of architectures and applications of convolutional neural networks. One succesful convolutional neural network is the Grasp Quality Convolutional Neural Network (GQ-CNN) trained on Dex-Net 2.0[1] which achieves a 99% precision on 40 novel objects. In this paper, a grasp were defined as planar position, angle and depth of a gripper relative to an RGB-D sensor. The experiments were carried out with an ABB YuMi robot that used a parallel jaw grasp. Their model has two inputs with the first one being the image and the second one being the depth. At the beginning of the model the input with the image goes into the model, which starts with two convolutional layers followed by one max pooling layer. After that it has two more convolutional layers and in the end it has 3 fully connected layers. It also has one additional fully connected layer with the additional information of planar position, angle and depth.

One way to reduce overfitting is to generate more data by using label preserving transformations[3]. The transformations mentioned in the paper is image translations and horizontal reflections. Another transformation is rotation as mentioned in [4]. These transformations are done by applying a displacement field on the image such that for each pixel a new location for it is calculated. It is also mentioned that the affine transformations greatly improved their results on the MNIST database.

3 Problem statement

This project sets out to explore how good a CNN architecture can perform on grasping objects given an evaluation function. One vanilla CNN is used as a baseline and one more complex model is tried out to see how it will perform as an approximate evaluation function for determining a grasp given an unseen image. The reason we approximate the evaluation function by using a CNN model instead of the evaluation function itself is that the traditional evaluation function is hard to use in real time due to its long computational time. To apply in the real-time application of the robot arms such as ABB YuMi, the computation time is of the main concern. As a result, our main task in this project is twofold. First, the data generating function is given by our project supervisors. However, the data produced by this function is heavily uneven. Thus, We should try to deal with the imbalanced data. Second, we are going to build a CNN model which can approximate the traditional evaluation function.

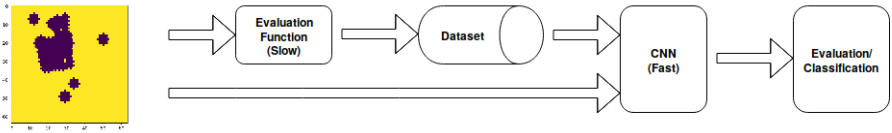


Fig. 1: The pipeline

4 Theory

4.1 Convolution

A convolution is a mathematical operation and as described in the book, Deep Learning [2], the most basic case has the following formula:

$$s(t) = \int x(a)w(t-a)da$$

and is commonly written as:

$$s(t) = (x * w)(t)$$

where $*$ denotes the convolution operation. In convolutional network terminology w is kernel, s is output or sometimes called feature map and x is input. A convolution is a mathematical operation and for the 2-dimensional problem, which an image can be described as, the operation can be written as the following:

$$S(i,j) = (I * K)(i,j) = \sum_m \sum_n I(m,n)K(i-m, j-n).$$

with input image I and kernel K. Due to its commutativity, it can also have the following form:

$$S(i,j) = (K * I)(i,j) = \sum_m \sum_n I(i-m, j-n)K(m,n).$$

Sparse weights is one important concept that a convolutional neural network possess which makes it efficient. It has the property of sparse weights because in the network the kernel is usually smaller than the input which makes it so that all the input units does not interact with all the output units. This will reduce the number of parameters required and is more memory efficient compared to a traditional neural network where all the input units interacts with all the output units.

In the convolutional layer of a convolutional neural network, multiple kernels are applied to the input. This is done through element-wise product between the input matrix and the kernel matrix and sum results together. The kernel matrix is typically smaller than the input matrix. A single kernel would be able to extract a single kind of feature. By using many different kernels many features,

which can be corners and edges for example, can be extracted from the input[2] and this results in the output of multiple feature maps[5]. The feature maps that are produced are then used in subsequent layers to extract higher order features. Because convolutional neural networks are able to extract features with the convolutional layer and the sparse weight representation makes it memory efficient. These properties makes it good for image classification.

4.2 Max pooling

In order to achieve spatial invariance, pooling layers can be used[6]. One such function is Max pooling, which produces an output that is the maximum value within a rectangular neighbourhood[2]. By applying Max pooling to the input the resolution of the output will be lower than the resolution of the input[6]. In [6] it is shown that Max pooling is superior to subsampling for capturing invariances in imagelike data. It is also described that using overlapping pooling windows does not increase recognition rate.

4.3 Local Response Normalization

The idea of utilizing Local Response Normalization (LRN) comes from [1]. This layer helps improve the generalization of the model. The concept of LRN is inspired by the mechanism called "lateral inhibition" in neurobiology. The mechanism describes the ability that an excited neuron suppresses the activity of its neighboring neurons. Based on this concept, LRN layer tends to form a local maxima and increase the contrast in an area of interest so as to increase the sensory perception of the model. According to [3], the response-normalized activity $b^i_{x,y}$ is given by the expression:

$$b^i_{x,y} = a^i_{x,y} / \left(k + \alpha \sum_{j=max(0,i-n/2)}^{min(N-1,i+n/2)} (a^j_{x,y})^2 \right)^{\beta}$$

where $a^i_{x,y}$ represents the activity of a neural located in (x, y) and applied by kernel i after the ReLU nonlinearity. N is the total number of kernels, and n denotes the number of adjacent kernels which are taken into account. The parameter k is a bias term that can be tuned by using a validation set. The parameters α and β are also determined by using the validation set.

4.4 F-measure

F-measure is an evaluation metric that is based on combining precision and recall where precision (P) and recall (R) can be defined in terms of true positives (TP), false positives (FP) and false negatives (FN) as[7]:

$$P = \frac{TP}{TP + FP}$$

$$R = \frac{TP}{TP + FN}$$

The formula for F-measure looks like the following[8]:

$$F = \frac{(\beta^2 + 1) \times R \times P}{\beta^2 \times P + R}$$

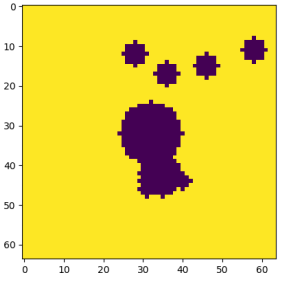
Depending on the value of β it is possible to control the balance between precision and recall. $\beta = 1$ gives the $F1$-score[9] which is the harmonic mean of precision and recall.

5 Approach

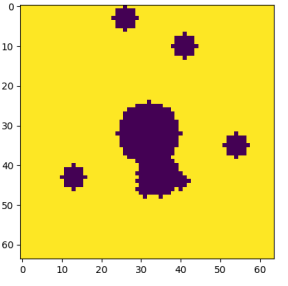
The approach of the this project will be discussed in this section. The topics that will be covered is architecture, TensorFlow, the input pipeline, the data generation, the data augmentation, the conversion to TFRecords.

5.1 Dataset

The synthetically generated dataset contains two classes with the labels cage relevant and cage irrelevant. The first one is when it is a successful robotic grasp and the second an unsuccessful one. Two examples of two images belonging to the two different classes can be seen in Fig. 2. In Fig. 2a is an image belonging to the cage irrelevant class while in Fig. 2b is an image belonging to the cage relevant class.



(a) The data set with the most probability mass

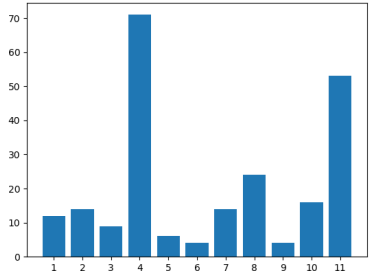


(b) The data set with the least probability mass

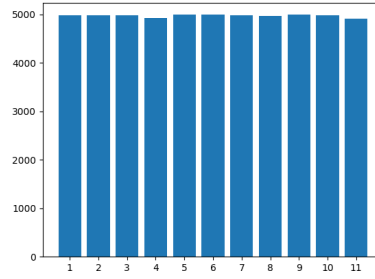
Fig. 2

When the images are generated the grippers are placed randomly in the image and therefore the probability that an image will have the label cage relevant is

much lower than cage irrelevant. This will make the dataset heavily unevenly distributed. In Fig. 3 is two plots of how many images there are of each object belonging to the two classes. Fig. 3a is showing the cage relevant images. There are 11 different objects and for each object it is shown how many different images that object has in this dataset. Object 4 has most with 70 different images belonging to the cage relevant class while both object 6 and 9 has the least with 5 images each. Fig. 3b shows a similar plot but for the cage irrelevant case. In this plot all of the objects have around 5000 images belonging to the cage irrelevant class.



(a) The number of cage relevant images distributed between 11 different objects



(b) The number of cage irrelevant images distributed between 11 different objects

Fig. 3

5.2 Architecture

In this project, we construct two CNN models. One is a vanilla CNN model which is constructed by mainly following the tutorial on the official Tensorflow website[10]. The other is a deeper CNN model which is inspired by the Grasp Quality Convolutional Neural Network (GQ-CNN)[1].

The first model, as shown in Fig. 4, contains six layers with weights. They are two convolutional layers with each followed by a max pooling layer and two fully-connected layers at the end. The only difference we make comparing to the architecture proposed in the tutorial is that we add a dropout layer after the the first fully-connected layer in order to decrease the speed of overfitting of the model and hoping to increase the generalization. For the second model, we increased the number of convolutional layers to four and use only one max pooling layer instead of two. Moreover, we add one normalizing layer called local response normalization layer (LRN) after every two convolutional layers. The motivation we utilize this normalization layer is described in the Theory part of this report.

The second model, which is shown in Fig. 5, contains 7 layers in total. The design of the architecture is inspired by GQ-CNN[1]. There are four convolutional layers with one max pooling layer placed in the middle, followed by two fully-connected layers. Response-normalization layers follow the second and the fourth convolutional layers. The ReLU non-linearity is placed after each convolutional layer and the first fully-connected layer. The dropout layer is placed after the first fully-connected layer in order to avoid overfitting.

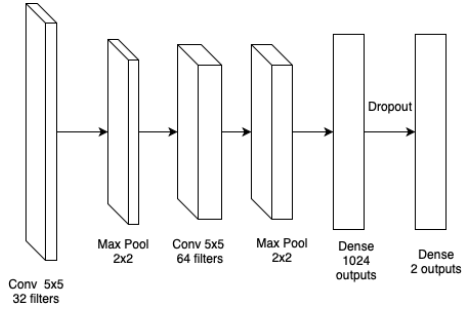


Fig. 4: The vanilla CNN model (first model)

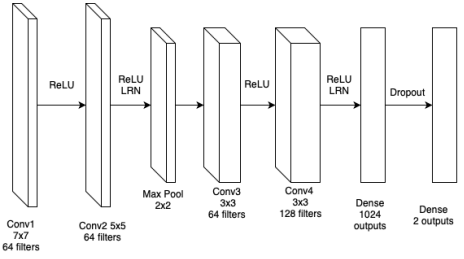


Fig. 5: The second model

5.3 TensorFlow

In this project, the TensorFlow[10] framework was used for building the models. TensorFlow is an interface that is used to express machine learning algorithms. It has its own file format called TFRecord which is a binary format. TensorFlow allows the user to build the models, train and evaluate them in the same framework, which streamlines the workflow with the pipeline.

5.4 Pipeline

The pipeline consists of 4 primary steps which are data generation, data augmentation, converting to TFRecords and input the TFRecords to the models. More details about the different stages of the pipeline will be given in the following sections below and an overall picture of the pipeline will be described here. The first step is to generate dataset that is going to be used for training and testing. Because of how the data is generated the probability of generating a cage-irrelevant is much higher than that of generating a cage-relevant and therefore the dataset will be heavily unevenly distributed. The next step is to deal with the uneven dataset that has been generated. The way it is dealt with is by augmenting the cage-relevant class of the dataset and it is done in two ways. The first way being oversampling of the minor class and the second by label preserving image transformations. After the dataset is augmented it is converted into TFRecords. The last step of the pipeline is to feed the TFRecords into the models.

5.5 Data generation

The synthetic data is generated as 2 channel images that have a width and height of 64 pixels. In each picture a random object represented by a point cloud and four grippers also represented by point clouds are generated at random locations where the object is in the first channel and the grippers in the second channel. The data is written into text files. Each line in the text-file contains one sample.

5.6 Data augmentation

Because the dataset is heavily unevenly distributed,several data augmentation techniques have been used. They were carried out after the data generation was done. The ones that has been used in this project is oversampling of the minor class and the transformation techniques, rotation, translation and flipping of images. Rotation, translation and flipping has also been combined to produce even more data. The oversampling and the transformation of data is done in slightly different stages of the pipeline. The oversampling is done immediately after the data generation, while the transformation of the images is done at the same time as the conversion from text-file to TFRecords.

The label preserving transformation techniques that were used on the images were rotation, translation and flipping. For rotation, the images were rotated by 90, 180 and 270 degrees. Thus, with the original image there are four different rotations used. For flipping, the images were flipped vertically and therefore there are two images with the original one. The translations were done by moving the images by one pixel to the left, right, up and down. It was also done for the combinations of left, up and left, down and right, up and right, down. With the original image this gives eight different translations of the image. By combining these three techniques, it is possible to generate $4 * 2 * 9 = 72$ new images.

The oversampling was done by first splitting the dataset into cage-relevant and cage-irrelevant images and $count = C$ the amount of cage-irrelevant images. Then the cage-relevant images was sampled until there were $C/72$ in order to get an evenly distributed dataset after the image transformations have been used on the cage-relevant images. After the oversampling is done, everything is written to a new text-file.

Adding noise to the image was also a technique that was considered but it does not produce a new image and therefore does not contribute to help with the imbalance of the dataset and therefore omitted. Adding noise to the synthetic input image would simulate a real image[11] which would make the network more robust.

5.7 Converting to TFRecords

In the step of converting to TFRecords the text-file with all the cage-irrelevant and all the cage-relevant images are read. It is then converted to TFRecords by using the TensorFlow API and saved to a tfrecord-file.

6 Experiments

We evaluate our models by using the synthetic preprocessed dataset mentioned above. In the beginning of the experiments, we train both the first and second model on the same balanced dataset in order to compare their performance. The criteria of performance is based on the accuracy and the f1 score on the test set. Fig.4 shows the architecture of the first vanilla convolutional neural network. This architecture mainly follows a tutorial that we found on a website. Fig.5 shows the architecture of the second model. The design of the second model is inspired by GQ-CNN[1]. Basically, the vanilla CNN acts as a baseline which we can compare the second model based on its performance.The further description of architecture can be found in Section 5.2 Architecture. The training dataset we used contains approximately 20,000 data point, and each data point is an synthetic image of an object with a particular gripper topology. Each model setting, as shown in 1, is trained on the same training set and valuated on the same validation set. The scores are the result based on an unseen test set.

7 Results

In this section the results of the different models are presented. In Table 1 the max and average values of accuracy, precision and f1-score is shown for each of the different models. In the table it can be seen that the first model is the one that performs worst. The second model has a value of above 0.8 for all of the different evaluation metrics. By removing both dropout and LRN it can be seen that the values for the metrics drops to below 0.8 in each except Max precision. The second model with dropout is slightly better than the second model with LRN.

Table 1: Accuracy, precision and f1-score of each different model settings

Scores	first model	second model	second model without LRN	second model without dropout	second model without LRN and dropout
Max accuracy	0.563	0.945	0.945	0.945	0.711
Avg accuracy	0.480	0.901	0.896	0.875	0.558
Max precision	0.563	0.896	0.908	0.907	0.917
Avg precision	0.485	0.833	0.820	0.800	0.651
Max f1-score	0.720	0.945	0.952	0.951	0.584
Avg f1-score	0.648	0.888	0.909	0.905	0.375

8 Conclusions

Based on the result stated in the previous sections, we could come to the conclusions as following:

- Training on a unevenly distributed dataset would impede the learning capability of model. To be more specific, the network model would overfit on the major class of the dataset, which leads to the final predict is always the same (the label of the major class).
- By increasing the depth of model, the learning capability seems to be improved if we compare the performance between the first and the second model.
- Both LRN and dropout layers have significant influence in the performance of model. Dropout layer can prevent the network from overfitting. LRN performs a form of lateral inhibition which aids generalization. Combining LRN and dropout does not improve the results a lot. In Table 1 it can be seen that the second models results are not improved a lot compared to using either dropout or LRN.

When we evaluate the models on different datasets, the performance can vary to a significant degree, meaning that the models are still overfitting to the training sets somehow. The potential reason might be that we augment our dataset by oversampling the minor class of the imbalanced data. From the result, we believe it would be good to try other data augmentation techniques such as undersampling.

References

1. Mahler, J., Liang, J., Niyaz, S., Laskey, M., Doan, R., Liu, X., Ojea, J.A., Goldberg, K.: Dex-net 2.0: Deep learning to plan robust grasps with synthetic point clouds and analytic grasp metrics. arXiv preprint arXiv:1703.09312 (2017)
2. Goodfellow, I., Bengio, Y., Courville, A.: Deep Learning. MIT Press (2016) `http://www.deeplearningbook.org`.
3. Krizhevsky, A., Sutskever, I., Hinton, G.E.: Imagenet classification with deep convolutional neural networks. In: Advances in neural information processing systems. (2012) 1097–1105
4. Simard, P.Y., Steinkraus, D., Platt, J.C.: Best practices for convolutional neural networks applied to visual document analysis. In: null, IEEE (2003) 958
5. LeCun, Y., Bottou, L., Bengio, Y., Haffner, P.: Gradient-based learning applied to document recognition. Proceedings of the IEEE **86**(11) (1998) 2278–2324
6. Scherer, D., Müller, A., Behnke, S.: Evaluation of pooling operations in convolutional architectures for object recognition. In: Artificial Neural Networks–ICANN 2010. Springer (2010) 92–101
7. Davis, J., Goadrich, M.: The relationship between precision-recall and roc curves. In: Proceedings of the 23rd international conference on Machine learning, ACM (2006) 233–240
8. Chinchor, N.: Muc-4 evaluation metrics. In: Proceedings of the 4th conference on Message understanding, Association for Computational Linguistics (1992) 22–29
9. Sasaki, Y., et al.: The truth of the f-measure. Teach Tutor mater **1**(5) (2007) 1–5
10. Abadi, M., Agarwal, A., Barham, P., Brevdo, E., Chen, Z., Citro, C., Corrado, G.S., Davis, A., Dean, J., Devin, M., Ghemawat, S., Goodfellow, I., Harp, A., Irving, G., Isard, M., Jia, Y., Jozefowicz, R., Kaiser, L., Kudlur, M., Levenberg, J., Mané, D., Monga, R., Moore, S., Murray, D., Olah, C., Schuster, M., Shlens, J., Steiner, B., Sutskever, I., Talwar, K., Tucker, P., Vanhoucke, V., Vasudevan, V., Viégas, F., Vinyals, O., Warden, P., Wattenberg, M., Wicke, M., Yu, Y., Zheng, X.: TensorFlow: Large-scale machine learning on heterogeneous systems (2015) Software available from tensorflow.org.
11. Johns, E., Leutenegger, S., Davison, A.J.: Deep learning a grasp function for grasping under gripper pose uncertainty. In: Intelligent Robots and Systems (IROS), 2016 IEEE/RSJ International Conference on, IEEE (2016) 4461–4468
12. Szegedy, C., Ioffe, S., Vanhoucke, V., Alemi, A.A.: Inception-v4, inception-resnet and the impact of residual connections on learning. In: AAAI. Volume 4. (2017) 12