From Vision to Grasping: Adapting Visual Networks

Rebecca Allday, Simon Hadfield, and Richard Bowden

Centre for Vision, Speech and Signal Processing, University of Surrey, UK
{R.Allday, S.Hadfield, R.Bowden}@surrey.ac.uk

Abstract. Grasping is one of the oldest problems in robotics and is still considered challenging, especially when grasping unknown objects with unknown 3D shape. We focus on exploiting recent advances in computer vision recognition systems. Object classification problems tend to have much larger datasets to train from and have far fewer practical constraints around the size of the model and speed to train. In this paper we will investigate how to adapt Convolutional Neural Networks (CNNs), traditionally used for image classification, for planar robotic grasping. We consider the differences in the problems and how a network can be adjusted to account for this. Positional information is far more important to robotics than generic image classification tasks, where max pooling layers are used to improve translation invariance. By using a more appropriate network structure we are able to obtain improved accuracy while simultaneously improving run times and reducing memory consumption by reducing model size by up to 69%.

Keywords: Robotic Grasping, Machine Learning, CNNs, SqueezeNet, AlexNet

1 Introduction

In the field of robotics, grasping is a fundamental yet challenging problem. It is considered even more challenging when attempting to generalise grasps to previously unseen objects using just sensor data rather than heuristics or 3D models. Robotic grasping requires solving several problems in the different stages of grasping. Initially, these could include object segmentation and object recognition, in order to find an object in a possibly cluttered environment and determine what the object is. Then we need a method for grasping the given object. In the simplest case, these grasps may be estimated from a 3D model, which itself is a challenging task. However, depending on the sensor modalities, the challenges are further exacerbated by the need to simultaneously reason about the object's shape and optimal grasping points. Finally there is the problem of motion planning and robust execution of the proposed grasp. In this paper we will be looking at a key aspect of this pipeline, specifically finding a planar grasp configuration in the challenging case where we do not know the shape of the object.

In recent years deep learning with CNNs has emerged within computer vision and related fields as the dominant approach to simultaneously learn feature representations and classifiers. This means that instead of engineering an explicit 3D representation for objects, we can learn an alternative intermediate representation. Networks such as AlexNet [7], GoogLeNet [14] and SqueezeNet [5] have shown how deep learning can be used to deliver excellent results for image classification on large object datasets such as ImageNet. However, whilst deep networks perform extremely well for tasks such as image classification, and have been successfully applied to robotics applications, we should consider the difference in application when designing and training them.

In order for robotic grasping to be useful we need to be able to make decisions about how to grasp objects quickly, so large deep neural networks which take a long time to process data are not practical. Furthermore, online learning where the robot iterates through stages of exploration and learning phases by definition require the learning phase to be fast. Unlike image classification we cannot farm data from the internet, grasping databases take a long time to collect because example grasps must be individually executed and recorded, whether that be via a simulator or a real robot. Avoiding large data collection phases would also be an advantage in robotics especially since data may need to be collected every time new hardware such as grippers are used. In addition to this, many robotics systems may have space constraints when it comes to storing models, sometimes requiring them to be on an embedded system, so smaller models would be required. The controllers for these systems may also not have the power that a desktop computer would. All of these problems mean we need to adapt CNNs used for general vision tasks to robotics.

In this paper we will look at how to adapt networks designed for visual recognition to the task of robotic grasping. In Section 2 we review the related work. Then in Section 3 we describe in more detail the problem of robotic grasping we address and how we intend to adapt our visual networks. Section 4 evaluates the proposed changes to the network architecture. Finally, we conclude in Section 5 and discuss the possibilities of future work.

2 Related Work

There are many different approaches to tackle robotic grasping, the survey by Bohg et al. [1] splits them into analytic and data-driven methods. Analytic methods tend to use geometric and dynamic criteria to form grasp maps to find force closure grasps. Alternatively data-driven methods rank sampled grasps from the grasp space based on some specified metrics. This is the area we will be focusing on, it can be divided into 3 subsections:

- Known objects: Requires use of object recognition and pose estimation to retrieve a suitable grasp from an experience database. Often these methods have access to geometric models of the objects.
- Familiar objects: Assumes any object encountered is similar to another object and uses a measure of similarity (shape, colour, texture) to find a possible grasp.
- Unknown objects: Does not assume any access to models or grasp experience. Translates features of an object directly to ranking grasp candidates.

Data-driven methods often use existing knowledge of grasps. This can be as a direct way of mapping grasps onto a known or familiar object. Alternatively existing knowledge can be used as training data for a machine learning method to learn general rules for mapping features of objects to grasps. We will be focusing on methods using machine learning in this paper.

Learning for grasping has been attempted with many different types of data including 2D images [13], 3D data [4,2] and multi-modal data combining both of these [8,12]. Both El-Khoury and Sahbani [2] and Pelossof [10] use superquadratics to approximate an object and then train an artificial neural network network (ANN) and Support Vector Machine (SVM) respectively. The drawback for both of these approaches is that they require access to accurate 3D models of the objects. More work has been done recently on learning grasp configurations from RGB images directly. Levine et al. [9] train CNNs to learn a mapping straight from raw images to torques at the robot's motors, however

they only use a few hundred examples to train a deep network. Pinto et al. [11] focus on this problem collecting around 40,000 random trial and error grasp attempts to adapt a CNN based on AlexNet. Given an image patch, the output of their CNN predicts the likelihood of a successful planar grasp at the center of the patch in 18 discreet angles (0°, 10°, ... 170°), making it 18-way binary classification problem. Whilst the amount of data they have collected is an improvement on many previous grasping datasets, it is not a significant amount when compared to the number of parameters in the network which is pre-trained on over 1 million images.

Within the context of deep learning for computer vision, Inandola et al. [5] created SqueezeNet which is designed to have far fewer parameters in comparision to AlexNet whilst maintaining accuracy. Their motivation for this was to increase efficiency in training and decrease the size of deployed models. They achieved this by the use of Fire modules which are comprised of a squeeze convolution layer which feed into an expand convolution layer. Fire modules only have 1x1 and 3x3 filters in their convolution layers keeping the number of parameters to a minimum. This makes it a suitable architecture for embedded systems and problems where there is a reduced volume of training data, however it has as of yet not been applied to the problem of robotic grasping.

3 Approach

We define the problem as predicting a successful planar grasp configuration (x, y, θ), where (x, y) is the center point of the grasp and θ is the angle of the gripper, from an image of an object centred at (x, y). We use CNNs to predict whether a grasp configuration results in a successful grasp for a given image. The input into our CNN will be an image patch centred on the center point of the grasp. Whilst learning for grasping is often treated as a continuous problem, aiming to find the optimal (x, y, θ), this is difficult because there are many possible successful grasp options for each object and CNNs tend to perform better at classification problems. Therefore similarly to Pinto et al. [11] we make our problem discrete by setting the output of our network to be N likelihoods. Each of which predicts if the object in the center of the patch is graspable at $\theta_n = \frac{(n-1)\pi}{N}$ radians, where $n \in \{1, ..., N\}$. Thus we can think of this as an N-way binary classification problem. A smaller N decreases the number of parameters in the network but leaves us with a much coarser angle selection which could impact the accuracy of the grasp.

We have N outputs to our CNN but for each image patch input, a robotics system can only trial one of the angles corresponding to a single output. This means our input image patch only has a single label, $l_n \in \{0, 1\}$, corresponding to attempted angle θ_n, whereas the network has N binary output layers. In order to integrate training with robot control our loss layers must be able to block different back propagation paths depending on the decision as to which angle the robot has tried. This is done using a softmax layer followed by an adapted multinomial logistic loss layer on each of the N outputs. Each adapted multinomial logistic loss layer only calculates a loss if the label for an image states that the angle corresponding to that output has been attempted.

Given a single $n \in \{1, ..., N\}$ the output from the softmax layer gives the probability of angle θ_n being successful as

$$p_n = \frac{e^{x_1^n}}{\sum_{k \in \{0,1\}} e^{x_k^n}}, \tag{1}$$

where x_k^n are the outputs from the n^{th} binary output layer. The loss for the n^{th} output is then given as

$$L_n = \begin{cases} -l_n log(p_n) - (1 - l_n) log(1 - p_n) & \text{if } \exists\, l_n \\ 0 & \text{otherwise.} \end{cases} \tag{2}$$

Then the gradient of the loss being fed back to the output layer is

$$\frac{\partial L_n}{\partial p_n} = \begin{cases} -\frac{l_n}{p_n} + \frac{1-l_n}{1-p_n} & \text{if } \exists\, l_n \\ 0 & \text{otherwise.} \end{cases} \tag{3}$$

We can see that, as required, this blocks back propagation on the layers where a label, l_n, does not exist indicating the angle θ_n has not been attempted.

Based on the differences between robotic grasping and computer vision applications we train adapted visual CNNs, based upon both AlexNet and SqueezeNet architectures. Here we describe the adaptations we propose in further detail.

3.1 Network adaptation

Translation Invariance A key difference between the problem of object classification and grasping is that object classification only cares about what an object is, not its position in the image. For robotic grasping, the precise position of an object and its grasp points are important. In visual neural networks, max pooling layers are often used to improve the translation invariance when classifying objects. They are also used to reduce the size of the data going through the network to make it more manageable for training and testing. For robotic grasping it is likely that this damages the specificity of the grasps. Therefore it may be advantageous to avoid spatial accumulation in the later stages of visual networks to reduce the effect of translation invariance while keeping the early pooling layers to keep the network size manageable.

Reduced Feature Complexity Another issue is that visual networks for classification tend to be used to classify hundreds if not thousands of different objects. Conversely, in this task we have simplified the problem down to N binary classifiers (one for each angle). The problem is far simpler than general classification in that it has fewer outputs. We also tend to have far less data available for grasping problems making it infeasible to train complex networks with many parameters. We can combat this partially by using pre-trained convolution layers on a dataset such as ImageNet, but the weights of these networks will still need to be adapted to the task at hand.

When a network is too complex for the given task information may be spread sparsely through the network, with some parts being unused. Here we give an example of this. When we convolve an input image I_{in} with the i_{th} convolution layer filter $conv_i$ we get $I_{out} = conv_i * I_{in}$. We can also apply a Rectified Linear Unit (ReLU) layer to an image with the function $I_{out} = max(I_{in}, 0)$. When we combine these functions across two convolution layers both followed by ReLU layers we get

$$I_{out} = max(conv_{i+1} * max(conv_i * I_{in}, 0), 0). \tag{4}$$

If the network is too complex for the given task it is possible for the network learn to give $conv_i * I_{in} > 0$ for all elements. In this case the first maximum function becomes redundant giving

$$I_{out} = max(conv_{i+1} * conv_i * I_{in}, 0). \tag{5}$$

Now due to the linear nature of the convolutions, $conv_{i+1} * conv_i$ can be combined into a single convolution $conv'_i$, meaning we can write

$$I_{out} = max(conv'_i * I_{in}, 0). \tag{6}$$

This shows that if the network is too complex then it is possible for two sets of ReLU and convolution combinations can give the same output as a single set.

Our suggestion is to create a simpler network with less convolution layers which may be better constrained by the problem, making it more accurate and more efficient. We do this by removing convolution layers from later in visual networks and randomly initialising the last remaining convolution layer whilst keeping the others pre-trained. This is because the penultimate convolution layer is specialised to feed into the final layer. Random re-initialisation of this penultimate layer ensures it encodes the penultimate and final stages for the task at hand.

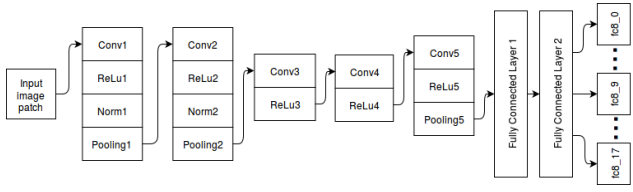
4 Results

For our experiments we will be using the data set collected by Pinto et al. [11] for planar grasping. A Baxter robot by Rethink Robotics with a two fingered parallel gripper was used to collect random trial and error grasp attempts on a table of objects. For a single grasp attempt, a random point is selected 25cm above the region of interest containing an object. This forms the grasp point and a random angle in the range $(0, \pi)$ is chosen as the gripper angle. For each trial, the grippers are instructed to close until the motors are stalled, this force on the motors tells us that the gripper has stopped before full closure so is grasping an object. The grasp configuration for each attempt is recorded along with whether the grasp was a success or a failure depending on the readings from the force sensors in the grippers. This data assumes that we split the angles between $(0, \pi)$ into 18 angle bins so choosing $N = 18$ from Section 3. Therefore our labels will consist of which angle was attempted and whether is was a success.

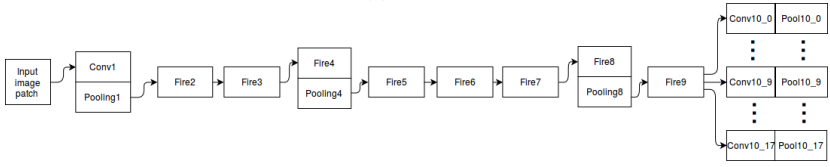
Deep learning requires vast quantities of data, especially if networks have more parameters than are truly necessary. To help artificially increase the amount of data the image patches are rotated by random θ_r and this is added to the angle label. Another problem with trial and error data sets in grasping is that there are overwhelmingly more failure examples than successful examples. This bias can cause problems when training neural networks since they can improve their accuracy by always predicting a failure. To avoid this in our experiments we use balanced numbers of successful and failed grasp attempts when training our networks by sub-sampling the failure cases. The bias can then be added back in later by using a higher threshold on the output probability of a successful grasp.

The networks which will be training are modified versions of CNNs normally used for image classification. The two networks we will use are AlexNet and SqueezeNet. AlexNet has 5 convolution layers with max pooling after convolution layers 1, 2 and 5. AlexNet then has two fully connected layers and a final fully connected layer for classification. For this application the final fully connected layer is removed and replaced with 18 binary fully connected layers, as shown in Figure1a. SqueezeNet is made up of Fire modules which consist of a *squeeze* convolution layer and an *expand* convolution. As you can see in Figure1b, SqueezeNet is comprised of a standalone convolution layer

and then 8 fire modules (fire2-9), and then another standalone convolution layer. Max pooling layers are included following the first convolution layer, fire4, and fire 8, whilst a global average pooling layer is used to convert the output from the final convolution layer to a suitable output. Unless otherwise stated when using these networks, we initialised the parameters of the convolution layers in AlexNet and the first convolution layer and the Fire modules in SqueezeNet with models pre-trained on ImageNet.



(a) AlexNet



(b) SqueezeNet

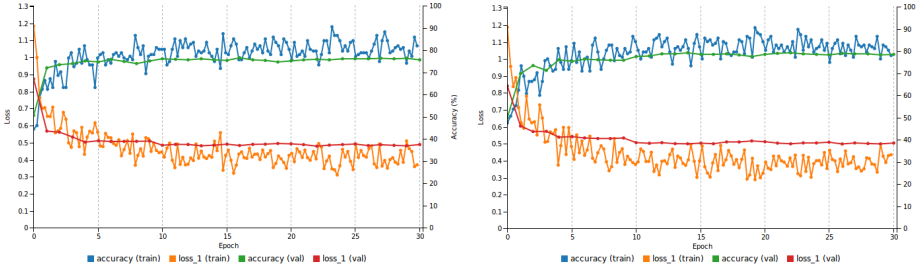
Fig. 1: Visual networks to be adapted for robotics

In order to decrease the time taken training CNNs we initially chose to take attempts from one of the 18 angle bins and train our network with a single binary output layer. This allows us to see any effect on learning before evaluating on the full dataset.

4.1 Evalutation of Translation Invariance

Evaluation of Removing Pooling layers The first experiment uses AlexNet with a single binary output trained on the data from angle 0 as the base and compares this to removing the final pooling layer in the network, Pooling5. These were both trained with an initial learning rate of 0.0001 for 30 epochs. The results of training two network configurations can be seen in Figure 2. The original network achieved 75.72% accuracy on the validation set after 30 epochs of training. In comparison the network without the final pooling layer achieved 78.60%. This small change to our network increased our accuracy by 2.88%. As stated earlier, we believe this is because we do not lose position based information which is necessary to accurately determine successful grasp points on an object. We were unable to remove earlier pooling layers from AlexNet as this changes the data size, rendering the pre-trained network unusable.

Conversely, the fully convolutional structure of SqeezeNet allows us to attempt removing more pooling layers earlier in the network. As a base we trained SqueezeNet in the same way we trained the base AlexNet achieving 75.48% final accuracy on the validation data. We can see in Table 1 that removing the later pooling layers for this network did not improve the accuracy, however removing the first pooling layer, Pooling1, did. However, this was at a large cost of the time taken to train the network since the data being passed through the network was much larger meaning this is not a practical solution especially if we wish to use this system for online training.



(a) Base AlexNet (b) AlexNet without Pooling5

Fig. 2: Graphs to show training of AlexNet with and without the final max pooling layer.

Evaluation of Reduced Feature Complexity To test the removal and random initialisation of later convolution layers we again used standard SqueezeNet with a single binary output training on angle 0 and all Fire modules pre-trained on ImageNet. We remove a single Fire module from the end of the network and reinitialise the final remaining Fire module. We repeat this until we have only 6 Fire modules remaining. We chose to use Xavier initialisation for stability in this set of results. We found it was difficult to avoid the exploding gradient problem using Gaussian initialisation when layers were removed. In Table 2 we see that without removing and layers we can improve accuracy by just randomly initialising Fire9, this probably helps the network learn more specifically for the task of grasping rather than object recognition. We can see that removing further layers in this network does not increase the accuracy. However, the file size for a single model does decrease by 69% whilst still having an accuracy of 71.63%. This is extremely beneficial in embedded systems and robotics where the space for large network models is often not available.

Table 1: Comparing the effect of removing max pooling layers from SqueezeNet

Pooling Layer Removed	Validation Accuracy	Time Taken for 30 epochs
-	75.48%	5 min 16 sec
Pooling8	73.79%	5 min 57 sec
Pooling4	71.95%	6 min 47 sec
Pooling1	75.96%	42 min 24 sec

Table 2: Comparing the effect of removing Fire modules from SqueezeNet and randomly initialising the final remaining Fire module.

Number of Fire Modules	Randomly Initialised Module	Validation Accuracy	Size of Model
9	-	74.03%	2.9MB
9	Fire9	77.16%	2.9MB
8	Fire8	73.79%	2.1MB
7	Fire7	73.07%	1.4MB
6	Fire6	71.63%	0.9MB

We took a similar approach to evaluating this solution with AlexNet. First we randomly initialised conv5 without removing any other layers, then we removed conv5 and randomly initialised conv4, then removed conv4 and randomly initialised conv3 and finally removed conv3 and randomly initialised conv2. The results to these experiments can be seen in Table 3. We can see that all of the networks we trained with less than 5 convolution layers in this experiment achieve a higher accuracy on the validation set than

the original AlexNet. The most impressive aspect of these results is that we can see that for this problem a much shallower network - for example one with only two convolution layers - is able to achieve accuracy higher than a deeper network. However, we find that removing layers in fact increases the size of the model. This is due to dimensions of the final convolution layer increasing when we remove layers meaning the number of weights to connect to the first fully connected layer increases. SqueezeNet avoided this problem due to it's fully convolutional nature.

Table 3: Comparing the effect of removing convolution layers from AlexNet and randomly initialising the final remaining layer.

Total Convolution Layers	Randomly Initialised Layer	Validation Accuracy	Size of Model
5	-	75.72%	0.2GB
5	Conv5	75.24%	0.2GB
4	Conv4	77.64%	1.1GB
3	Conv3	76.92%	1.1GB
2	Conv2	77.16%	0.7GB

Cross-Learning Between Angles The final set of experiments we ran used the full 18 binary layer outputs to see how our changes can affect the whole system. Firstly we trained our SqueezeNet base with all the angles, which gave an average of 86.39% accuracy on the validation data across all the angles. We see an improvement here compared to the single angle problem, with the accuracy of angle θ_0 being 74.03% on the single angle compared to 86.92% for θ_0 when trained with all the angles. This indicates that whilst the individual angles can be treated as separate problems, combining them with shared features leads to cross-training significantly improving the performance.

Next we ran the experiment on SqueezeNet with a randomly initialised final Fire module, this gave an average accuracy of 87.51% showing an improvement on our base network consistent with our single angle experiments. We also attempted removing the final layer and randomly initialising the penultimate Fire module giving an average of 85.01% accuracy but also reducing the model size by 24%. This shows that we can still maintain a high performing network with fewer parameters.

Finally, we trained our base AlexNet with the full set of output layers. This gave us an average accuracy of 80%. Despite the greatly improved results from removing a pooling layer on the single angle problem, removing the final pooling layer with the full network gave an average accuracy of 78.44%. Removing the final convolution layer and randomly initialising the penultimate layer gave an average accuracy of 78.93%. The reason for this discrepancy is likely to be that our methods excel with limited data. The full angle problem has 18 times more data than the single angle problem, but does not have 18 times more parameters. It is likely that in the 18 angle case our networks would out perform the base network in situations where less data is available such as for online learning robotics applications.

5 Conclusion

In this paper, we have proposed steps for adapting networks designed for computer vision tasks to robotic grasping. We have shown that due to the differences in these tasks it is vital for robotics researches to consider the network architectures to improve accuracy, obtain smaller models and improve training efficiency. The fact that the exact position of an object is far more important for accurate grasping means that decreasing

translational invariance in our networks helped improve the accuracy by 2.88% when using AlexNet. We also saw that by reducing the number of parameters being trained we were able to achieve improved accuracy in this problem over the base SqueezeNet while also obtaining a 69% reduction in model size. These methods can be used to create space efficient networks with improved accuracy which can be used in robotic control systems.

Given smaller networks and faster training times we feel this could lend itself to online learning systems. It would be interesting to explore the results of using these methods with reinforcement learning in a full end to end system. In the future we want to further investigate other advanced architectures developed, such as Residual Networks (ResNets)[3], 3D convolution systems [15] and Recurrent Neural Networks [6], to not only adapt vision networks to the task of robotics but to find which provided the best base architecture for this problem.

Acknowledgements
This work was supported by Tesco Labs, with particular thanks to Paul Wilkinson for his support.

References

1. Bohg, J., Morales, A., Asfour, T., Kragic, D.: Data-driven grasp synthesis - A survey. CoRR abs/1309.2660 (2013)
2. El-Khoury, S., Sahbani, A.: Handling objects by their handles. In: IROS Workshop on Grasp and Task Learning by Imitation (2008)
3. He, K., Zhang, X., Ren, S., Sun, J.: Deep residual learning for image recognition. CoRR abs/1512.03385 (2015)
4. Huebner, K., Kragic, D.: Selection of robot pre-grasps using box-based shape approximation. In: 2008 IEEE/RSJ International Conference on Intelligent Robots and Systems. pp. 1765–1770 (Sept 2008)
5. Iandola, F.N., Moskewicz, M.W., Ashraf, K., Han, S., Dally, W.J., Keutzer, K.: Squeezenet: Alexnet-level accuracy with 50x fewer parameters and <1mb model size (2016)
6. Jaeger, H.: Tutorial on training recurrent neural networks, covering bppt, rtrl, ekf and the" echo state network" approach (2002)
7. Krizhevsky, A., Sutskever, I., Hinton, G.E.: Imagenet classification with deep convolutional neural networks. In: Advances in Neural Information Processing Systems (2012)
8. Lenz, I., Lee, H., Saxena, A.: Deep learning for detecting robotic grasps. The International Journal of Robotics Research 34(4-5), 705–724 (2015)
9. Levine, S., Finn, C., Darrell, T., Abbeel, P.: End-to-end training of deep visuomotor policies. CoRR abs/1504.00702 (2015)
10. Pelossof, R., Miller, A., Allen, P., Jebara, T.: An svm learning approach to robotic grasping. In: Robotics and Automation, 2004. Proceedings. ICRA '04. 2004 IEEE International Conference on. vol. 4, pp. 3512–3518 Vol.4 (April 2004)
11. Pinto, L., Gupta, A.: Supersizing self-supervision: Learning to grasp from 50k tries and 700 robot hours. arXiv:1509.06825 (2015)
12. Redmon, J., Angelova, A.: Real-time grasp detection using convolutional neural networks. CoRR abs/1412.3128 (2014)
13. Saxena, A., Driemeyer, J., Ng, A.Y.: Robotic grasping of novel objects using vision. The International Journal of Robotics Research 27(2), 157–173 (2008)
14. Szegedy, C., Liu, W., Jia, Y., Sermanet, P., Reed, S.E., Anguelov, D., Erhan, D., Vanhoucke, V., Rabinovich, A.: Going deeper with convolutions. CoRR abs/1409.4842 (2014)
15. Tran, D., Bourdev, L.D., Fergus, R., Torresani, L., Paluri, M.: C3D: generic features for video analysis. CoRR abs/1412.0767 (2014)