3D Object Recognition using Convolutional Neural Networks with Transfer Learning between Input Channels

Luís A. Alexandre *

Department of Informatics and Instituto de Telecomunicações
Univ. Beira Interior, Covilhã, Portugal
luis.alexandre@ubi.pt

Abstract. RGB-D data is getting ever more interest from the research community as both cheap cameras appear in the market and the applications of this type of data become more common. A current trend in processing image data is the use of convolutional neural networks (CNNs) that have consistently beat competition in most benchmark data sets. In this paper we investigate the possibility of transferring knowledge between CNNs when processing RGB-D data with the goal of both improving accuracy and reducing training time. We present experiments that show that our proposed approach can achieve both these goals.

Keywords: 3D object recognition, transfer learning, convolutional neural networks

1 Introduction

The use of RGB plus depth (RGB-D) data has been increasing recently due to the availability of cheap cameras that produce this type of data. The possible applications of RGB-D data are multiple, but among the many possibilities we can cite the use for robotic vision as a means to allow a robot to better perceive its surrounding environment, recognizing both the type of environment [1] it is currently located in and the objects [2] that are within it.

Deep learning [3] has also receive strong attention lately due to its capacity to create multilevel representations of the data and allowing for a possibly more efficient way to solve some problems [4]. Several approaches have been put forward in this area: deep belief networks [5], stacked (denoising) autoencoders [6] and convolutional neural networks (CNNs) [7, 8] are among some of these.

CNNs have been shown to be particularly adapted to process image data, since they are inspired in the way the human visual system works [9] and have shown these capabilities by winning several international competitions involving object recognition [8].

* This work was partially financed by FEDER funds through the "Programa Operacional Factores de Competitividade - COMPETE" and by Portuguese funds through "FCT - Fundação para a Ciência e a Tecnologia" in the framework of the project PTDC/EIA-EIA/119004/2010 and PEst-OE/EEI/LA0008/2013.

In this paper we investigate the possibility of making transfer learning while training CNNs for solving object recognition tasks using RGB-D data. The main idea is to use four independent CNNs, one for each channel, instead of using a single CNN receiving the four input channels, and train these four independent CNNs in a sequence, instead of training them in parallel, and using the weights of a trained CNN as starting point to train the other CNNs that will process the remaining channels. We compare this approach against training four CNNs in parallel and also against a single CNN that processes the four input channels simultaneously and conclude that our proposal not only saves training time but it also increases the recognition accuracy.

The paper is organized as follows: the next section presents some related work; section 3 contains the detailed explanation of our proposal; section 4 presents the experiments and the last section lists our conclusions.

2 Related Work

2.1 Convolutional Neural Networks

A convolutional neural network (CNN) is a type of deep neural network (DNN) inspired in the human visual system, used for processing images. These were originally proposed by Fukushima [10] and latter also developed by LeCun [7]. A CNN has several stages (typically two or three), each composed of two layers: the first layer does a convolution of the input image with a filter and the second layer downsamples the result of the first layer, using a pooling operation. These stages build increasingly abstract representations of the input pattern: the first might be sensitive to edges, the second to corners and intersections and so on. The idea is that these representations become both more abstract and more invariant as the pattern data goes through the CNN. The output of the last stage is usually a vector (not an image) that is fed to a multi-layer perceptron (MLP) that produces the final network output, usually a class label.

The possibility of using CNNs for processing RGB-D data was investigated in [11]. In that paper the authors combined both a CNN with a recursive neural network and obtained state-of-the-art results on an RGB-D data set.

2.2 Transfer Learning with CNNs

Transfer learning (TL) consists in the use of knowledge acquired solving a source problem to facilitate the resolution of a target problem. This can take many different forms [12]. In the case of neural networks, one way to do TL is to reuse layers from the source problem to solve the target problem. These layers can be reused as they are or they can be fine-tuned.

The possibility of doing transfer learning on deep neural networks (DNNs) has been investigated before. In [13], the authors proposed to train a DNN on a given problem and reuse it by fine-tuning only the last layer, while keeping the remaining weights unchanged. They compare the results against randomly initialized DNNs and also against fine-tuning the last 2 layers, last 3 and so on. They focused on character recognition and concluded that transfer learning is viable in this task, since it allows for faster training and smaller classification errors.

In [14], the authors addressed the transfer learning between deep networks used as classifiers of digit images. They considered cases where only the set of class labels, or only the data distribution, changed from source to target problem. They concluded that reusing networks trained for a different label set led to better results than reusing networks trained for a different data distribution. In particular, reusing a network trained for more classes on a problem with less classes was beneficial for virtually any amount of training data.

While these previous studies showed that in general one can achieve some improvement in terms of accuracy and / or overall processing time for different configurations of both learning architectures and data sets by transfer learning, none of these approaches considers the possibility put forward in the current paper of transferring learned networks between channels while processing RGB-D data.

3 Our Proposal

The straightforward way to apply a CNN to RGB-D data is simply to train it to process images with four channels (three color + one depth).

An alternative (which we will also investigate in the experiments section) is to use one CNN to process each channel independently and then combine the decisions of each network to obtain the final classification result. This combination can be done in many different ways and we will explore two common possibilities in the experiments.

When training one CNN for each channel, it is obvious that the task each of these networks is learning to perform is very similar (specially among the three color channels). In this paper we propose to take advantage of the similarity of tasks that are performed on each input channel during training and avoid starting to train a CNN from scratch for each channel given that we can reuse the trained weights from one initial trained channel to become the starting point for training the remaining networks that will process the other channels.

Formally, consider a color image $I(x, y)$ that is composed of three color channels, such that $I(x, y) = (R(x, y), G(x, y), B(x, y))$ and each channel is a mapping from the set of image coordinates to the set $D_c = \{0, \ldots, 255\}$.

The RGB-D data adds to the color image a depth image, that is usually a 16 bit grayscale image, $D(x, y)$, that maps the image coordinates to the set $D_d = \{0, \ldots, 2^{16}-1\}$.

A CNN trained on one of these channels is learning a function from D_c^m or D_d^m, where m is the number of pixels, to the set of class labels, $y = \{0, 1, \ldots, C-1\}$, for a C-class classification problem.

Some of the approaches that we test imply a combination of the outputs of several CNNs. For this, we used two combination methods, that we now describe. Let us represent the output of neuron j of network k by $y_{j,k}$, and consider n classifiers and a problem with C classes. The decision of a CNN k corresponds to the output with largest value: $d_k = \text{argmax}_{j=1,\ldots,C} y_{j,k}$ The combination using the majority vote is given by:

$$i = \arg \max_{j=1,\ldots,C} \sum_{k=1}^{n} \mathbb{1}_{\{d_k=j\}}$$

where i is the class label obtained through the combination and $\mathbb{1}_{\{d_k=j\}}$ is the indicator function that gives 1 when d_k is equal to j and zero otherwise.

The second combination method is the maximum of the mean of the outputs produced by each CNN. The decision using this approach is the class

$$i = \arg \max_{j=1,\ldots,C} \frac{1}{n} \sum_{k=1}^{n} y_{j,k} = \arg \max_{j=1,\ldots,C} \sum_{k=1}^{n} y_{j,k}$$

4 Experiments

We used an open source CNN implementation, `cuda-convnet`[1], that takes advantage of graphics processing units (GPUs). All experiments were run on Fedora 17, using a nVidia GeForce 680 GPU.

4.1 The Data Set

We used a subset of the large data set of 3D point clouds from [15]. The original data set contains 300 objects from 51 different categories captured on a turntable from three different camera poses. We used 48 objects representing 10 categories. The training data contain clouds captured from two different camera views, and the validation and test data contains clouds captured using a third different view.

The training set has a total of 946 clouds, the validation set has 240 clouds and the test set contains 475 clouds. The data used to train consisted on two 2D images per point cloud, one with the RGB data and another with the depth information (16 bit data). These images were re-scaled to squares with 64 pixel sides. Figure 1 shows some example images of the dataset objects and Fig. 2 shows the corresponding depth maps. It is easy to understand from these examples that a CNN that uses only the depth maps will have a more difficult task than a CNN using a color channel.

4.2 Network Architecture and Training

All networks use three stages (three convolution plus sub-sampling layers) plus a final one hidden layer MLP. The convolution layers use 32 filters and the sub-sampling layers use the $relu$ function. Full description of all paramenters can be found in the configuration files available online[2].

The starting value of the learning rates of the convolution layers and MLP layer and some of the other meta-parameters (L_2 weight decay for convolutional and MLP layers) were obtained by doing a grid search using the depth data (the one that had worst performance) and were kept for all channels. Other parameters (number and sizes

[1] `code.google.com/p/cuda-convnet`

[2] Since there are too many parameters to present here, the configuration files used can be obtained online: `www.di.ubi.pt/~lfbaa/pubs/IAS-13.zip`. The lists with the names of the files used in the training, validation and test sets are also there. This allows for our experiments to be reproduced.

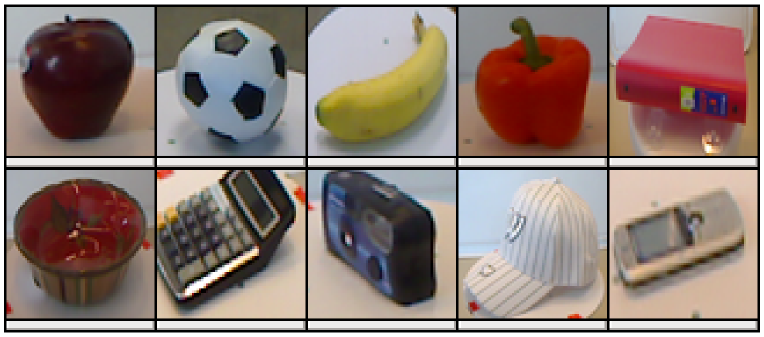

Fig. 1. One sample RGB image of each of the 10 object categories used in the experiments. Note that in each category, there are usually more than 3 or 4 different objects that can differ significantly between each other.

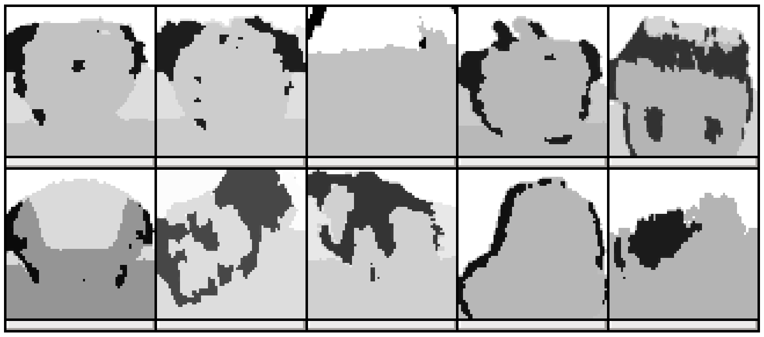

Fig. 2. One sample depth image (equalized for display purposes) of each of the 10 object categories used in the experiments. These sample images correspond to the RGB images presented in Fig. 1.

of convolutional filters, initial values on the weights and bias, momentum, etc.) used the default values. This grid search used only the data from the training and validation sets.

The training of all networks was done according to the following algorithm:

```
1. restart = 0
2. while restart < 3 do
   (a) train for 10 epochs
   (b) evaluate the validation error, VE
   (c) if VE increased for two consecutive evaluations:
       – restart = restart + 1
       – divide the value of the learning rate of the convolution layers by 10
       – go back to the weights of the network used 20 epochs before
3. evaluate the test error and stop.
```

Note that the goal of these experiments was not to obtain the best possible result in this dataset but to illustrate other possible ways to train a CNN to deal with RGB-D data, namely, using transfer learning. Even so, the presented results are comparable with the color descriptor results obtained in [2], and are better than the results obtained with the grayscale descriptors. Note also that in that paper, the validation set was not used since for the methods presented there, no meta-parameter search was performed.

4.3 Two Baselines

We consider two baseline results against which we compare the transfer learning proposal.

For the first baseline, named RGBD in table 1, we consider a single CNN that receives all four channels as input.

As a second baseline, we consider four independent CNNs, each one processing one of the input channels. The output of these four networks is then combined through simple majority vote (marked with 'maj' in table 1) and also through the maximum of the mean value of the output of the last layer of the CNNs (marked with 'mean' in table 1), to yield the final decision.

4.4 Transferring Learned Networks

We train, as in baseline 2, four networks, one for each input channel. But in this case we will transfer the weights from the network that had the smallest error among the 40 trained with individual channels. These weights are used as a starting point for the training (fine-tuning) of the networks from the other three channels. This is done instead of using random weights. The individual decisions are then combined by the same two approaches used in baseline 2.

Since the data used in the CNN that processes the depth data is different in its nature (different range of values and also different type of underlying data distribution) from the color channels, we also tested the combination of the channel that had smallest error (R) with the G and B channels learned with TL and the depth channel without transfer learning, since the weights transferred from a color channel network could be inadequate to serve as a starting point to train the depth channel CNN. The outputs of these networks are also combined with the same two approaches as before.

4.5 Discussion

The results obtained in the experiments are presented in table 1.

Regarding the error rates, we see that the best result was obtained with a transfer learning approach (R,G+TL,B+TL,D+TL mean), reducing by 1.07% in 29.87% (a 3.6% reduction) the error obtained with the RGBD approach (single CNN, first row of table 1) and also being faster than the RGBD approach by more than 22%.

When training individual CNNs for each channel we found that the best results were obtained using channel R. So we transferred the weights of the network trained with only the R channel data that achieved the smallest error (as we trained 10 networks for each channel) to be used as a starting point to train new networks for the other three remaining channels. These are listed in table 1, with a + TL in front of the channel name.

These networks, that used the weights of the best R channel network, achieved better results than the originally trained networks for the case of the two remaining color channels (B and G) and slightly worst results for the depth channel. The standard deviations of their errors show that they all produced very similar final results.

In the three combination settings, the decision obtained with the maximum of the mean was always better than the decision obtained with majority voting. These decision fusion approaches using the maximum of the mean were also always better than using a single CNN to process the four channels, both in terms of error rates and also in terms of the time spent.

The combination of the channel that had smallest error (R) with the G and B channels learned with TL and the depth channel without transfer learning (R,G+TL,B+TL,D mean) did not outperform the R,G+TL,B+TL,D+TL mean, showing that even if the original CNN trained for the depth channel without transfer learning was used (that had smaller error than the D + TL CNN), the combination performance did not beat the combination using the R channel plus the other three TL networks.

The best overall time was obtained when fusing the four independently trained CNNs without using TL: an improvement of more than 29% over the single CNN.

Regarding the time, this approach can be use in real-time applications. The fastest time was around 500s, but this includes training, validation and testing for 10 repetitions of the experiment. Even if we consider this to be the time for testing only and since we used near 500 point clouds in the test set, this gives around 0.1 seconds per point cloud. But note that the training time is much longer than the test time, so if the networks were to be trainined off line, the test speed would be substantially higer than this value.

5 Conclusions

In this paper we studied alternative approaches to train a CNN with RGB-D data.

First we proposed the use of four independently trained CNNs, one for each channel that were then combined to produce a decision by two different methods. We concluded that using one of the combination methods we can obtain both smaller error and faster training time than using the single CNN for all channels.

Then we proposed the use of transfer learning between the CNNs trained on each channel. This again showed better results than the original single CNN approach, both in terms of error and time, and also better than in the first combination proposal in terms of error but not in terms of time.

So we conclude that it is advantageous to split RGB-D data into four separate data sets, train CNNs on each channel and fuse their results. Better results in terms of error rates can even be achieved if one transfers the weights from the CNN with smallest error and uses them as a starting point for the training of the CNNs of the remaining channels, implementing a transfer learning approach.

Table 1. Average (and standard deviation) of the error and time used on 10 repetitions for each of the evaluated approaches. The best results for error and time spent are in bold.

Approach	Error [%]	Time[s]
RGBD	29.87 (3.30)	714.60 (191.95)
Channel R	32.15 (3.17)	136.50 (36.05)
Channel G	44.02 (2.40)	131.60 (41.79)
Channel B	55.62 (5.58)	110.10 (24.83)
Channel D	65.85 (0.77)	126.30 (17.16)
R,G,B,D maj	36.72 (4.31)	**504.50** (119.83)
R,G,B,D mean	29.58 (1.66)	**504.50** (119.83)
Channel G + TL	37.47 (0.00)	166.60 (0.70)
Channel B + TL	43.58 (0.00)	95.10 (0.57)
Channel D + TL	66.32 (0.00)	157.70 (0.67)
R,G+TL,B+TL,D+TL maj	33.45 (0.75)	555.90 (37.99)
R,G+TL,B+TL,D+TL mean	**28.80** (0.61)	555.90 (37.99)
R,G+TL,B+TL,D maj	32.63 (0.79)	524.50 (54.48)
R,G+TL,B+TL,D mean	29.01 (0.63)	524.50 (54.48)

References

1. Pronobis, A., Mozos, O.M., Caputo, B., Jensfelt, P.: Multi-modal semantic place classification. I. J. Robotic Res. **29**(2-3) (2010) 298–320
2. Alexandre, L.A.: 3D descriptors for object and category recognition: a comparative evaluation. In: Workshop on Color-Depth Camera Fusion in Robotics at the IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS), Vilamoura, Portugal (October 2012)
3. Bengio, Y.: Learning deep architectures for AI. Foundations and Trends in Machine Learning **2**(1) (2009) 1–127
4. Le Roux, N., Bengio, Y.: Deep belief networks are compact universal approximators. Neural computation **22**(8) (2010) 2192–2207
5. Hinton, G.E., Osindero, S., Teh, Y.W.: A fast learning algorithm for deep belief nets. Neural computation **18**(7) (2006) 1527–1554
6. Vincent, P., Larochelle, H., Bengio, Y., Manzagol, P.A.: Extracting and composing robust features with denoising autoencoders. In: Machine Learning, Proceedings of the Twenty-Fifth International Conference (ICML 2008), Helsinki, Finland, June 5-9, 2008. (2008) 1096–1103
7. LeCun, Y., Bengio, Y.: Convolutional networks for images, speech, and time-series. In Arbib, M.A., ed.: The Handbook of Brain Theory and Neural Networks, MIT Press (1995)
8. Ciresan, D.C., Meier, U., Schmidhuber, J.: Multi-column deep neural networks for image classification. In: 2012 IEEE Conference on Computer Vision and Pattern Recognition, Providence, RI, USA (June 2012) 3642–3649
9. Filipe, S., Alexandre, L.A.: From the human visual system to the computational models of visual attention: A survey. Artificial Intelligence Review (January 2013) 1–47
10. Fukushima, K.: Neocognitron: A self-organizing neural network model for a mechanism of pattern recognition unaffected by shift in position. Biological Cybernetics **36**(4) (1980) 193–202

11. Socher, R., Huval, B., Bath, B., Manning, C., Ng, A.: Convolutional-recursive deep learning for 3d object classification. In Bartlett, P., Pereira, F., Burges, C., Bottou, L., Weinberger, K., eds.: Advances in Neural Information Processing Systems 25. (2012) 665–673
12. Pan, S.J., Yang, Q.: A survey on transfer learning. IEEE Transactions on Knowledge and Data Engineering **22**(10) (2010) 1345–1359
13. Ciresan, D.C., Meier, U., Schmidhuber, J.: Transfer learning for latin and chinese characters with deep neural networks. In: The 2012 International Joint Conference on Neural Networks, Brisbane, Australia (June 2012) 1301–1306
14. Amaral, T., Sá, J., Silva, L., Alexandre, L.A., Santos, J.: Improving performance on problems with few labelled data by reusing stacked auto-encoders. In: submitted. (2014)
15. Lai, K., Bo, L., Ren, X., Fox, D.: A Large-Scale hierarchical Multi-View RGB-D object dataset. In: Proc. of the IEEE International Conference on Robotics & Automation (ICRA). (2011)