CIFAR-10 - Object Recognition in Images

Sebastian Hagn SEBASTIAN.ANTON.HAGN@EST.FIB.UPC.EDU

Johan Nordin JOHAN.ERIK.OLOF.NORDIN@EST.FIB.UPC.EDU

Abstract

This report describes how a neural network with one hidden layer is trained for classifying images from the CIFAR-10 data set. Firstly different preprocessing methods are used to boost the classification rate. Secondly the network's parameters are optimized to tune the performance given the preprocessed data.

Keywords: Artificial neural network, Object Recognition, Preprocessing, CIFAR-10, Principal component analysis

1. Introduction

Object recognition is an important part of computer vision. Humans can easily recognize and classify objects in images, but for machines it is usually more difficult. This is partially explained by the fact that the architecture of the human brain is different from the architecture of a computer Freeman and Skapura (1991). Neural networks are "designed to model the way in which the brain performs a particular task" Haykin (2009) and they have significantly improved state-of-the-art performance in various domains, such as object recognition, speech recognition, drug discovery and so on LeCun et al. (2015).

In this project, we aim at successfully train a feed-forward neural network on the CIFAR-10 data set and thereby gaining understanding about object recognition with neural networks. We check how preprocessing of the input data and the tuning of the network's parameters as the number of hidden neurons affect the network's overall classification performance. Namely, we will use MATLAB and the Neural Network Toolbox to create a network, train, validate and test the performance of the network.

In order to have a common understanding of what object recognition is about, in section 2 we will give a brief overview on related previous work. Having this basic understanding, in section 3 we will then describe the CIFAR-10 dataset. Followed by an explanation of our methodology in section 4 and how the data set has been preprocessed in section 5. Moreover, in section 6, we describe the how the Artificial Neural Network (ANN) have been implemented and which parameters that are being used. After that, we will present the training and validation results in section 7. Eventually, in section 8, the report will be summarized in a conclusion.

2. Related work

The CIFAR-10 and its relative the CIFAR-100 are both labelled subsets of the 80 million tiny images data set and they are well established data sets used in object recognition Krizhevsky (2016); Kaggle (2016). As the MNIST data set CIFAR-10 is widely used and to evaluate the performance of classification algorithms LeCun and Cortes (2016). Latest research methods show an accuracy as high as 96,5% on the data set. All the neural networks which are achieving such high classification rates are usually deep and/or convolutional networks and are using techniques as dropout Hinton et al. (2012); Benenson (2016) and they take at least several hours for training even when using modern GPUs nagadomi (2015).

3. Dataset

The CIFAR-10 is an open source data set and consists of 60,000 32x32 color images used for object recognition Krizhevsky (2016). There are 50,000 images in the training set and 10,000 in the test set. In addition, there are 10 object classes in total and one image belongs exclusively to a certain class and there are no intersections among the 10 classes. The label classes are namely airplane, automobile, bird, cat, deer, dog, frog, horse, ship and truck. Using the CIFAR-10 data set for this project has many advantages. The data set is well documented and has been used many times in scientific research.

A color image usually consist of three separate channels: red, green and blue. Therefore, in the data set each image 32x32 is stored as a row with 3x1024 elements and the whole dataset is represented as a n*m matrix, where m is the number of dimensions and n is the number of images.

Moreover, in the data set each image is identified through a vector that contains the associated index for the category that the image belongs to. This information was encoded into binary variables in a target variable matrix. One target vector consists out of only zero values except for a 1 in the element which represents the category of this instance.

4. Methodology

Our procedure to train the ANN can be divided into two steps. First we focused on the preprocessing of the data. We used a fixed feedforward network with one hidden layer with 20 neurons. The training function was scaled conjugate gradient backpropagation and cross-entropy was used. The latter are the default values for the ”patternnet” provided by the MATLAB software Inc. (2015). While keeping this network topology fixed we applied different preprocessing methods on the data. Then we trained ten networks with the manipulated data and took the average test classification rate. If the preprocessing method improved the classification rate we kept it and continued to try another method. In the second step we focused on optimizing the network structure and did not modify the input data in this step. We tried to find optimal values for the number of neurons as well as the hyper-parameters.

5. Preprocessing

As described we tried different preprocessing methods and evaluated which influence it had on the classification rate. For conducting these tests we did not use the whole training data set of 50000 instances, but only one data batch which contains around 10000 samples. This reduced the training time significantly and allowed a faster process iteration, especially due to the fact that we had to train several networks to get an average estimate of the performance. We chose to use the second data batch because its class distribution comes closest to the even class distribution in the whole training data set. The network used 70% of the data set for training and 15% for validation. The remaining part was used to determine its performance. Through preprocessing we intended to reduce the needed computational resources for training the network as well as helping the network to learn more general representation of the data.

The CIFAR-10 data set consist of color images and an important question was whether if it was necessary to maintain the color representation or not. The information may not be important in some algorithms and exactly what was the benefit of keeping the color representation was unclear. However, if we converted the images to grayscale, which would be taking the mean over the three channels, we will definitely lose information. Moreover, in images color may be related to different objects. For example, flowers generally consist of a different set of colors than airplanes. For this reason, we determined not to convert the color images to grayscale.

5.1. Centering

Centering the data is a common preprocessing method that implies subtracting the mean across every individual feature in the data. In our case we transformed the data so the rows corresponds to the pixels and the columns to images and centered the values per image.

5.2. Principal component analysis

Principal component analysis (PCA) is a method which is often used for dimensionality reduction and/or feature extraction. It can be defined as the "as the orthogonal projection of the data onto a lower dimensional linear space [...] such that the variance of the projected data is maximized" Bishop (2007). PCA is especially suited for our case because in natural images adjacent pixels are highly correlated and will therefore allow the data to be approximated with lower dimensions Ng et al. (2013). Although the images in the data set are quite small, they still have a large number of dimensions and training a neural network that use all the dimensions can be computationally burden and take excessive time. That is the reason why in our case a reduction of dimensions has an exceptionally big impact on the training of the neural network. To decrease the number of dimensions even further we kept only the components that hold 99% of the variance.

5.3. Whitening transformation

Whitening is another preprocessing technique that can be used if the data is redundant, e.g. the neighbouring pixel values often are very similar in images. This type of redundancies is what we want to remove but at the same time we dont want to lose useful information.

However, the property of whitened data is that it has a zero mean and the principal components are normalized to unit variance. Therefore, it can be obtained by perform PCA and then normalize the variances of the principal components by dividing them by their standard deviations.

5.4. Image preprocessing

When all the different methods described above had been tested and we still wanted to improve the results, we tried with some traditional image processing methods. As mentioned in de Andrade; Goodfellow et al. (2013), Global contrast normalization (GCN) has proved to be a successful preprocessing method. The goal was to process the data set in the image domain and remove unnecessary information using methods such as histogram normalization and median filters. If it turn out to be successful, the plan was to use it together with other preprocessing methods.

A fair assumption to make about the images in the data set is they are collected during different occasions and conditions. Therefore, the intensity levels in the images can vary and it is possible that they are not in the same range. To make these intensity levels equal or almost equal, we used a technique called histogram equalization. The idea with histogram equalization is to stretch and redistribute the original values to use the entire range of the discrete intensity levels. This creates a more uniform representation of the data set.

Moreover, the images in the data set are small and it is therefore hard to tell whether an image contains noise or not. For this reason, a median filter was implemented to remove the outliers in the images, pixel values that potentially could provide the network with misleading information.

Both of this operations were performed in the image domain. This means that each image in the data set first was transformed from one dimensional arrays to three dimensional matrices, which is the standard representation for color images. Then after being processed in the image domain they were transformed back to the initial representation.

6. Artificial Neural Network

After finishing the manipulation of the input data we kept the best obtained configuration and started optimizing the network itself. We chose beforehand the training and performance functions based on theoretical knowledge. The network structure and the hyper-parameters were then optimized by experiments.

We used the following parameters for the network. Other parameters were subject to change in the experiments.

Training function: Scaled conjugate gradient
Performance function: Crossentropy
Transfer function (hidden layer): Tansig
Transfer function (output layer): Softmax

6.1. Network structure

A neural network can be seen as a collection of parallel units that are connected together in the form of a directed graph. Furthermore, a multilayer neural network consists of a layer of input units, one or more layers of hidden units, and one output layer of units, as shown in Figure 1.

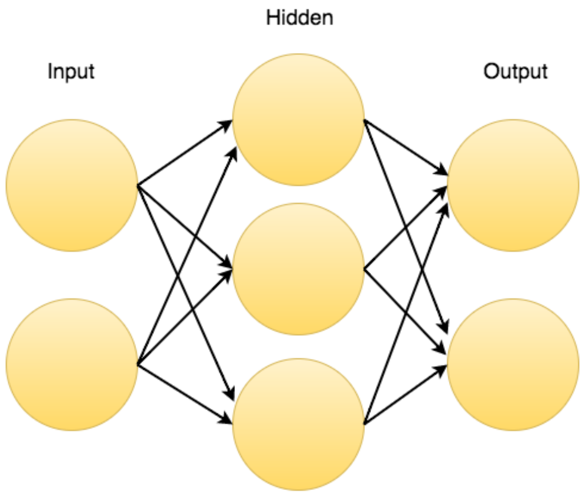

Figure 1: A simple neural network with one hidden layer.

While the number of input- and output units of a network are dependent of the data set. The number of input units are set by the number of the dimensions of the data set, in our case the number of pixels. The number of output units are tied to the number of categories.

However, the number of hidden units is not simply related to any known properties of the data set. The number of hidden units defines the total number of weights in the network, that could be considered as the number of degrees of freedom Duda et al. (2000). For this reason, the number of hidden units should not exceed the total number of training points.

To find the optimal number of hidden units is a complex problem that depends on many factors. Too few hidden units will generally create high training errors due to under-fitting. On the other hand, too many hidden units will result in low training errors, but will make the training slow, and will result in poor generalization and lead to overfitting. A possible solution to avoid over-fitting is regularization.

In addition to the number of hidden units it is also possible to experiment with the number of hidden layers of the network. The hidden layers can be interpreted as feature detectors, recognizing different patterns in the data. However because tuning a neural network requires a certain amount of experience which we did not have, we did not use more than one hidden layer because it would significantly the number of possible configurations. Thus we only tried different number of neurons.

6.2. Training function

We would have preferred to use the LevenbergMarquardt method because it combines the speed of the GaussNewton algorithm and the stability of the steepest descent method Haykin (2009); Wilamowski (2011). But processing only one data batch would already have needed

Matlab to acquire an array of 28,3 GB which goes way beyond both our notebooks RAM sizes. So we used the Scaled conjugate gradient method instead. It not only requires less memory but it is also suitable for a broad variety of problems and in particular networks with a large number of weights Inc. (2016).

The basic idea of training a neural network for classification problems, is that the network should learn mapping the inputs values to the outputs values for different patterns. The neural network is given pairs of input values and the corresponding target values. Therefore, it is called supervised learning method because the correct result is presented to the network. Moreover, a neural network learns patterns by adjusting its weights and how this is done is defined by the training algorithm.

6.3. Activation function

The default activation function for the hidden layer of the "patternnet" was the tansig activation function which we had no intention to replace. A useful property of neural networks used for classification problems are that the continuous outputs can interpreted as probabilities. We took advantage of this feature by using the Softmax activation function in the output layer. The Softmax activation function is a generalization of the Sigmoid activation function, that can be used when there is more than two output classes.

6.4. Performance function

We chose to use cross-entropy to measure the network's performance instead of a cost function based on the mean squared error. Using cross-entropy avoids the learning slowdown occurring when using a quadratic cost function Nielsen (2015). The performance goes also hand in hand with the matching activation function. Since we have a multi-class classification task and Softmax outputs a multi-class cross-entropy function is the suitable choice Bishop (2007).

6.5. Hyper-parameters

Besides the choice of the general structure of the network the optimization of the different hyper-parameters has a huge impact on a network's performance. Nielsen (2015) describes a strategy how to find reasonable values for these parameters. First suggestion is to simplify the problem and/or the network structure to get feedback on the experiments quicker. This makes it possible to try a lot more values for the hyper-parameters. As soon as the network's performance shows a significant improvement the optimization should focus to improve it further by individually adjusting one parameter. E.g. only the value of the learning rate will be changed while keeping the other parameters fixed. When an optimal learning rate is found, the next parameter for instance the regularization or the number of hidden neurons is optimized. This is done for each parameter. After that a new iteration can begin because all the changed parameters probably will also lead to new optimal values for the other parameters.

7. Results and Discussion

Table 1 shows the classification rate that we obtained by applying the different preprocessing methods.[1] The baseline are 32,6% which was the classification rate when the unchanged data was fed to the network.

Table 1: Classification rate obtained with different preprocessing methods

Applied technique	**Classification rate**	**Comments**
none (Raw data)	32,6%	
Centering	36,1%	
Standardizing	36,69%	
Contrast normalization	23,99%	on images reconstructed from the raw data
Median filter	27,36%	on images reconstructed from the raw data
PCA	22,1%	on standardized data
PCA	27,75%	on standardized data, 99% of total variation kept
processpca	22,37%	on standardized data
processpca	38,79%	on standardized data, components with at least 0,1% of variation are kept
processpca	41,11%	on standardized data, components with at least 0,4% of variation are kept

7.1. Centering and Standardization

Due that centering the input data is a standard procedure and it is useful in most cases we expected some improvement which also happened. The simple technique improved the classification rate by almost 4% to 36,1%. On top of that standardizing brings a small improvement.

7.2. Contrast normalization and Median filter

Neither the histogram normalization or median filter methods would prove successful. Regarding the histogram normalization, it may be because the method has been applied wrong and that it therefore did not lead to the desired result. We would subsequently find out that if the histogram normalization are made for each color channel separately, it will lead to inaccuracies. This mistake can partly attributable to that we wanted to keep the color representation of images to not lose information but also that we did not have enough knowledge about the method.

As for the median filter it was an experiment rather than something we had evidence that it worked. Also this method lead to worse classification results.

1. The number of maximum validation failures was increased from 6 to 20 and kept at this level.

7.3. Principal component analysis

At first we used the function "pca" to perform a PCA of the standardized data. We trained the neural network with all the obtained principal components as well as the number of components so still 99% of the variation in the data was explained. However the performance dropped a lot which was the contrary what we expected. Omitting the dimensions which explains only little variation in the data boosts the classification rate. Probably these dimensions irrelevant and/or noisy data and concentrating on fewer more changing features helps to discriminate the classes.

Because of the counter-intuitive results and assuming we did a fault on our part we tried another approach. This time we used the function "processpca" which is also a part of MATLABs Neural Network Toolbox. It processes a rows of matrix so that the rows become uncorrelated (which corresponds to the pixel in our case). When using all components both methods perform similar. But keeping only components that hold at least 0,1% of the variation with processpca, brings an 2% improvement compared to the standardized data. Further increasing the threshold up to 0,4% leads to a slight increase in performance to 41,11%.

This made us curious if the number of components is the decisive factor why we got better results with processpca than pca. So we used the first 84 components from the pca-method (which corresponds to the 0,1% threshold when using processpca) and we got 37,62% similar results with the pca. So it was not a wrong implementation as we thought, rather we did not decrease the number of dimensions enough.

7.4. Hyper-parameters

After we improved the classification rate by preprocessing the input data, we focused on the network structure and its parameters. The most obvious aspect is of course the number of hidden neurons. But as mentioned in section 6 a high number in hidden neurons will allow the network to fit the data arbitrary well which will lead to low training error, but a poor generalization. Therefore some form of regularization is needed. We will do this by changing the regularization parameter of the crossentropy function. This parameter states the fraction that is associated with minimizing the weights in the network and thus avoids overfitting. The figure 2 shows the impact of the regularization parameter. Whereas in the left diagram nothing keeps the network of mimic the data, on the right the constraint keeps the test-error from rising up again by preventing overfitting.

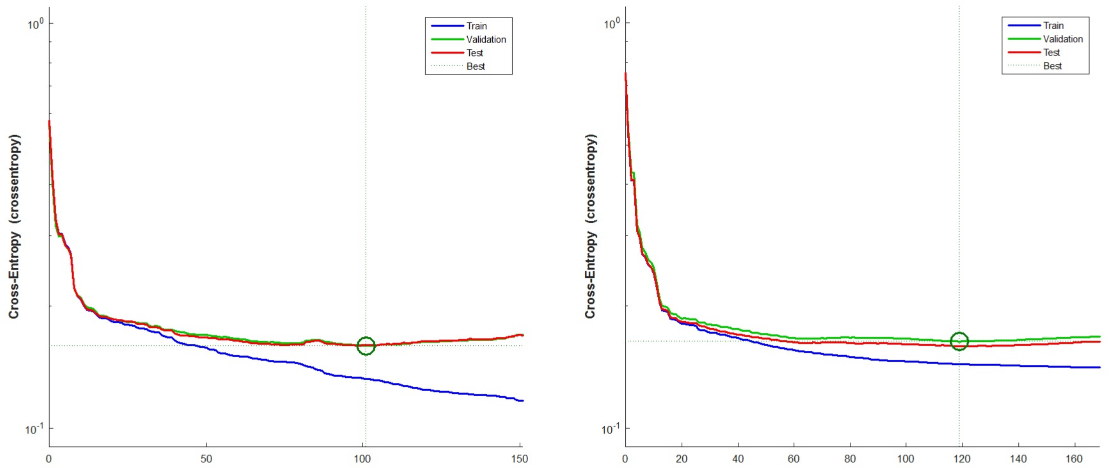

Figure 2: 100 neurons without and with weight constraint

We first chose to evaluate increased number of neurons up to 100 neurons which is according to Hastie et al. (2009) a typical range. As the table 2 shows got the classification rate on the test data even without a constraint slightly better with an increasing number of neurons. Adding a constraint on the weights then improved it further. The best results were obtained with the most neurons. There exist many rules of thumb which give advice on how to choose the number of hidden neurons Sarle (2002). Some of them would have suggested that the number of hidden neurons should not surpass the input data's dimensions. But as we see on the classification rate (on the test data) it performs quite well. Also our experiments suggest that more., e.g. 200, neurons would have improved the performance further. We kept to 100 neurons because with the number of neurons also the computation time rises significantly.

Table 2: Classification rate by changing number of hidden neurons and regularization

Hidden neurons	n=20				n=30			
Regularization	r=0	r=0.01	r=0.05	r=0.1	r=0	r=0.01	r=0.05	r=0.1
Classification	41,11%	41,93%	41,16%	42,63%	41,4%	42,83%	42,84%	39,59%

Hidden neurons	n=50				n=100			
Regularization	r=0	r=0.01	r=0.05	r=0.1	r=0	r=0.01	r=0.05	r=0.1
Classification	42,47%	42,29%	43,57%	43,5%	42,34%	42,35%	43,77%	44,81%

After finding these final parameters[2] we trained the neural network on the whole training data and then predicted the classes for the test data batch which achieved a classification

2. Because we use the scaled conjugate gradient backpropagation the learning rate, which otherwise would have been another very decisive parameter, had not to be adjusted.

rate of 48,59%. The increase of about 4% compared to the model on only the second data batch is probably due to the larger number of instances.

8. Conclusions

In this paper we have conducted experiments on the CIFAR-10 data set to gain understanding and knowledge about how to train a neural network for an object recognition task. We have worked in a structured but investigative manner through the project, due to that our knowledge in the field were limited in the beginning. Moreover, we have seen the importance of preprocessing the data as PCA and how various parameters impact on the classification performance.

We have also seen that the image-wise preprocessing method we applied did not yield any performance gains. For the median filter this could be partly due the small image size and the histogram normalization may not work correctly if applied on different colour channels separately. But it also shows that methods that human think should be helpful for the task, not necessarily turn out to have the desired impact in practice. Because of ANN are black-box methods, their training is and will continue to be a very experience-based task. Rules of the thumb try to soften the learning curve but do not apply every time, so for beginners there often only remains a trial-and-error approach.

References

Rodrigo Benenson. Classification datasets results, 2016. URL http://rodrigob.github.io/are_we_there_yet/build/classification_datasets_results.html.

Christopher M. Bishop. *Pattern Recognition and Machine Learning*. Springer, 2007.

Anderson de Andrade. Best practices for convolutional neural networks applied to object recognition in images.

R.O. Duda, P.E. Hart, and D.G. Stork. *Pattern Classification*. Wiley, 2000. ISBN 9781118586006.

J.A. Freeman and D.M. Skapura. *Neural Networks: Algorithms, Applications, and Programming Techniques*. Addison-Wesley, 1991.

Ian J Goodfellow, David Warde-Farley, Mehdi Mirza, Aaron Courville, and Yoshua Bengio. Maxout networks. *arXiv preprint arXiv:1302.4389*, 2013.

Trevor Hastie, Robert Tibshirani, and Jerome Friedman. *The Elements of Statistical Learning*. Springer Series in Statistics. Springer New York Inc., 2009.

Simon Haykin. *Neural networks and learning machines*. Prentice Hall/Pearson, 2009.

Geoffrey E. Hinton, Nitish Srivastava, Alex Krizhevsky, Ilya Sutskever, and Ruslan R. Salakhutdinov. Improving neural networks by preventing co-adaptation of feature detectors. *CoRR*, pages 1–18, 2012.

The MathWorks Inc. patternnet, 2015. URL https://de.mathworks.com/help/nnet/ref/patternnet.html.

The MathWorks Inc. Choose a multilayer neural network training function, 2016. URL https://de.mathworks.com/help/nnet/ug/choose-a-multilayer-neural-network-training-function.html.

Kaggle. Cifar-10 - object recognition in images, 2016. URL https://www.kaggle.com/c/cifar-10.

Alex Krizhevsky. Cifar-dataset, 2016. URL https://www.cs.toronto.edu/~kriz/cifar.html.

Yann LeCun and Corinna Cortes. The mnist database of handwritten digits, 2016. URL http://yann.lecun.com/exdb/mnist/.

Yann LeCun, Yoshua Bengio, and Geoffrey Hinton. Deep learning. *Nature*, 521:436444, 2015.

nagadomi. Code for kaggle-cifar10 competition. 5th place., 2015. URL https://github.com/nagadomi/kaggle-cifar10-torch7.

Andrew Ng, Jiquan Ngiam, Chuan Yu Foo, Yifan Mai, and Caroline Suen. Ufldl tutorial, 2013. URL http://ufldl.stanford.edu/wiki/index.php/UFLDL_Tutorial.

Michael Nielsen. Improving the way neural networks learn, 2015. URL http://neuralnetworksanddeeplearning.com/chap3.html.

Warren S. Sarle. comp.ai.neural-nets faq, part 3 of 7: Generalization - section - how many hidden units should i use?, 2002. URL http://www.faqs.org/faqs/ai-faq/neural-nets/part3/section-10.html.

Bogdan Wilamowski. *The industrial electronics handbook*. CRC Press, 2011.