Self Driving Car Nanodegree Project 2

Michael Person

February 26, 2017

1 Introduction

In this project a deep neural network was built, trained, and deployed using Tensorflow as part of Udacitys Autonomous Car Nanodegree Program [1]. The purpose of this project was to gain familiarity with large datasets, network architecture, optimization algorithms, hyperparameter selection, model generalization, and internal model state visualization. Each one of these items will be addressed by discussing the methods implemented with explanations given for why they were selected.

2 Dataset Summary and Exploration

The German Traffic Sign dataset was chosen to train this neural network [4]. The dataset consists of 34799 training images, 4410 validation images, and 12630 testing images spread over 43 classes. Each image is 32 by 32 pixels and is a standard 3 channel color image. As seen in Figure 1, Figure 2, and Figure 3 each three of the portions of the dataset were roughly equally distributed meaning the network is likely to train for each class to be predicted correctly.

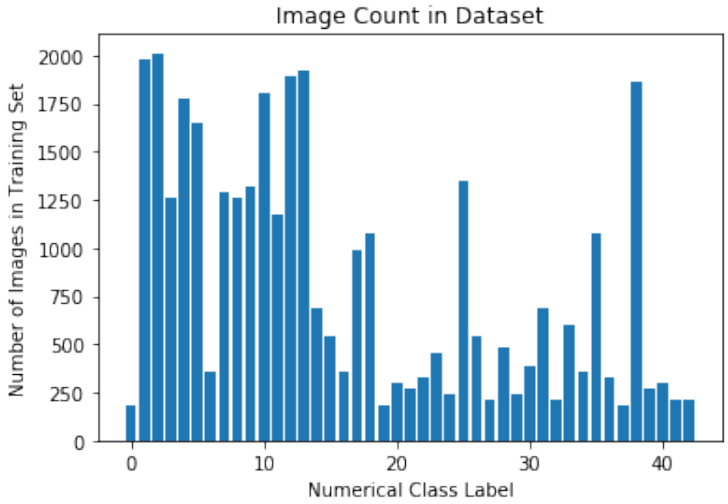

Figure 1: Training Images in dataset

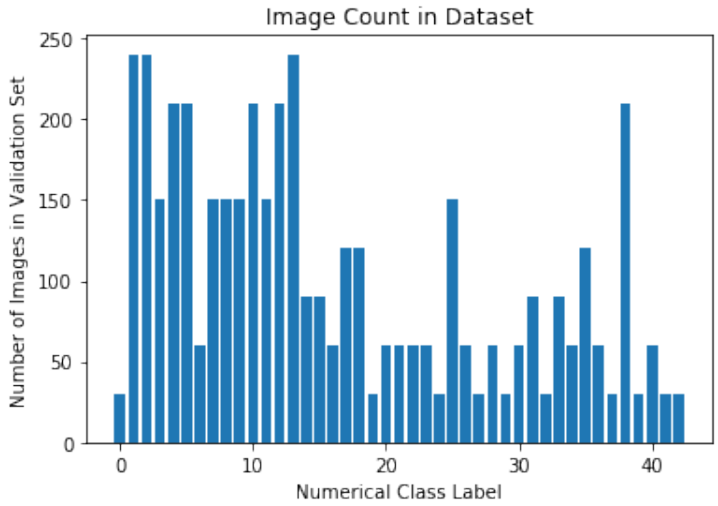

Figure 2: Validation Images in dataset

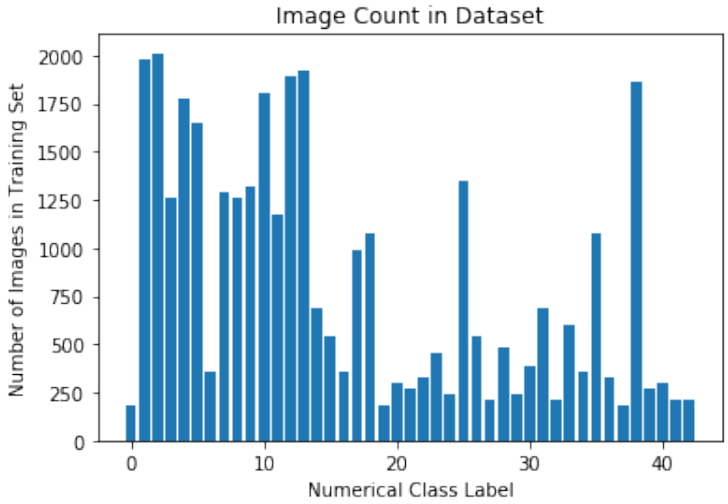

Figure 3: Testing Images in dataset

3 Data Augmentation

Aggressive data augmentation was undergone in order to combat overfitting due to the relatively small dataset. Most of the data augmentation ideas were taken directly from Krizhevsky et al. [2]. I employed the same two types of data augmentation, spacial and color based. This data augmentation increased the training set by a factor of 5 which significantly helped test accuracy and generalization.

3.1 Spatial Augmentation

The first spacial method that was used was a random horizontal and vertical shift. The range of movement for the image was 20 pixels, equating to ± 10 pixels. The next method that was used was a rotation of the image and the range was 90° equating to $\pm 45°$. The third method was to mirror the image around the middle of the vertical axis so the image would be horizontally flipped. The last spacial method that was used was the translated pictures were then rotated.

The reasoning behind these spacial methods were that an image should have the same classification regardless of where in the image certain features are found. The rotation augmentation is more subtle because an image should have the same classification regardless of what the orientation

of the found features are but within reason. Take for example a person, it is still a person if it is upside down but much more descriptive power may be needed to recognize that it is still a person even if it is not standing on the ground which is why a maximum rotation of $\pm 45^{\circ}$ was chosen.

3.2 Color Based Augmentation

The color based augmentation was not only inspired but directly taken from Krizhevsky et al. [2]. The method employs a variation of PCA where the common eigenbasis of the RGB pixel intensity values are found and then stochastic noise is added to each pixel in proportion to the representation of that color eigen basis. The mathematical formula can be seen as follows:

$$\begin{bmatrix} R'_{i,j} \\ G'_{i,j} \\ B'_{i,j} \end{bmatrix} = \begin{bmatrix} \mathbf{v}_1 & \mathbf{v}_2 & \mathbf{v}_3 \end{bmatrix} \begin{bmatrix} \alpha_1 \lambda_1 \\ \alpha_2 \lambda_2 \\ \alpha_3 \lambda_3 \end{bmatrix} + \begin{bmatrix} R_{i,j} \\ G_{i,j} \\ B_{i,j} \end{bmatrix} \tag{1}$$

Where λ and $\mathbf{v}$ are the eigenvalue, eigenvector pair for each RGB channel and α is the stochastic noise. The eigenvalues and eigenvectors were calculated via each images correlation matrix. The correlation matrix was found by flattening each R, G, and B channel into a single dimension vector and then appending them together to form a N^2 by 3 matrix which outputted the correct dimensionality of a 3 by 3 correlation matrix.

The reasoning for implementing this alteration method is that an image will maintain the same classification in different lighting and with small variations in the exact pixel intensities.

3.3 Preprocessing

Surprisingly, no preprocessing was used for the training of this network. I found that since this network took relatively little time to train (< 1 hour) and reached very high levels of accuracy that there was no need to reduce the memory of the images. The preprocessing would reduce the descriptive representation of the $0-255$ pixel intensity values by either normalization or subtracting the mean image pixel activity which would allow for higher batch sizes at reducing the amount of information that could be gleamed by the network. The weight's size were held in check via adding a regularizer to the loss function and therefore reducing the pixel's magnitude would hurt more than help the network.

4 Design and Learning of Model

4.1 Model Architecture

The model is almost entirely taken from LeCun et al. and can be seen in Figure 4 [5]. This architecture was chosen because of the relative small complexity of the dataset which required few learned features in order to correctly label each image.

My architecture did include minor details that helped aid generalization and accuracy though. The first layer was the start convolutional layer with filters of size 5x5x6 and stride of 1. I then added in a localized response normalization term from [2] in order to penalize high weights in order to reduce overfitting. Then the features were passed through a standard ReLU (Rectified Linear Unit) activation function. Next a max pooling was applied with kernel's of size 2x2 and stride 1.

Then the above layers were applied again in the same order including all dimensions and strides. The output of the second max pooling layer was then flattened and passed through two fully connected layers each with a ReLU activation unit and dropout in order to combat overfitting. The output of the second fully connected layer was fed into a a last fully connected layer to constitute the final output of the network.

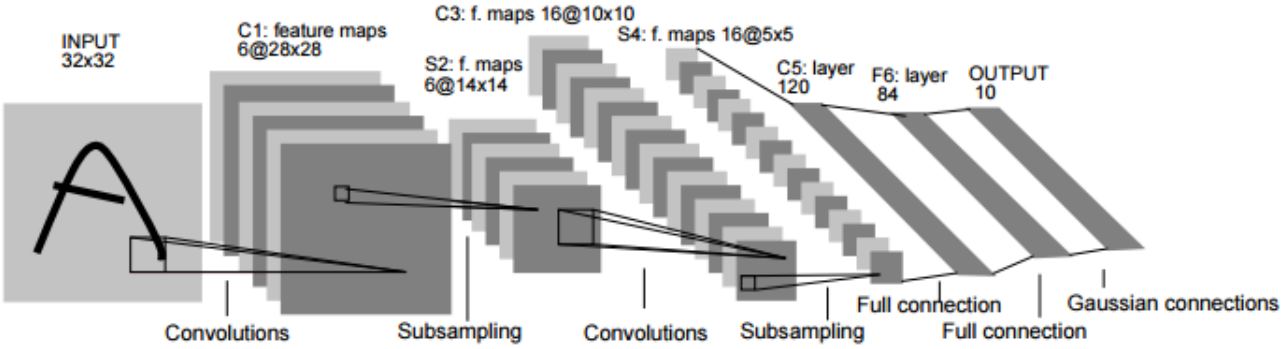

Figure 4: Model Architecture

4.2 Optimization Method

The optimization method used was ADAM [3]. Adam has proved to be a very effective learning algorithm in order to optimize the mean cross entropy of our validation set.

4.3 Hyperparameter Selection

The main hyperparameters selected were the batch size, number of epochs, and the learning rate. A batch size of 256 was used since the images were small and not memory intense and therefore could be trained in large numbers and also for many epochs without spending a large amount of training time. The number of epochs was chosen to be 100 because learning seemed to approach an asymptote by that point without the loss of generalization against the test set. The learning rate was chosen to be 0.0001 because it allowed for high convergence in the 100 epoch window. A lower learning rate may have been used with a much high number of training epochs if time had not been an issue.

4.4 Learning

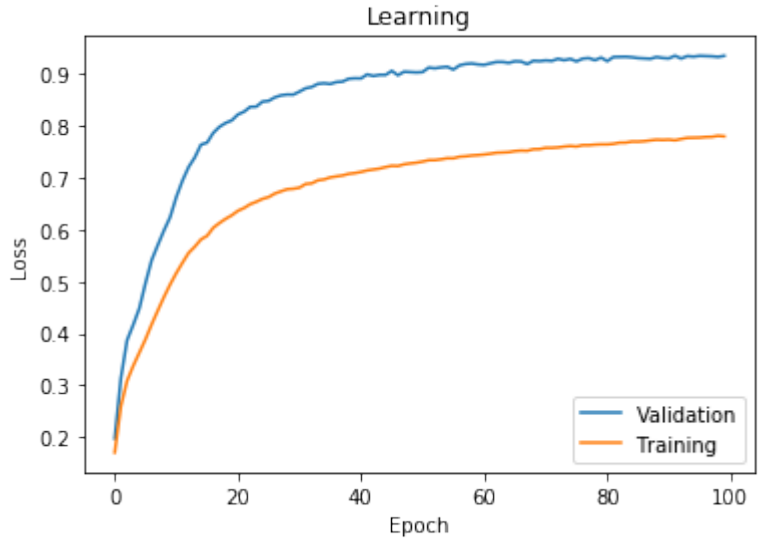

Figure 5: Learning Curve

The learning can been seen to steadily increase until it reaches what appears to be an asymptote. You can start to see the oscillations in the loss function because it is bouncing between points at

the bottom of the valley of the objective function which suggests that the learning would benefit from choosing a lower learning rate.

5 Results

5.1 Testing Results

My model achieved a testing accuracy of 91.6%. I attribute most of this to the augmented dataset and the large number of epochs that the model was able to train for. The breakdown of correct inferences can be seen in Figure 6 and the breakdown of wrong inferences can be seen in Figure 7.

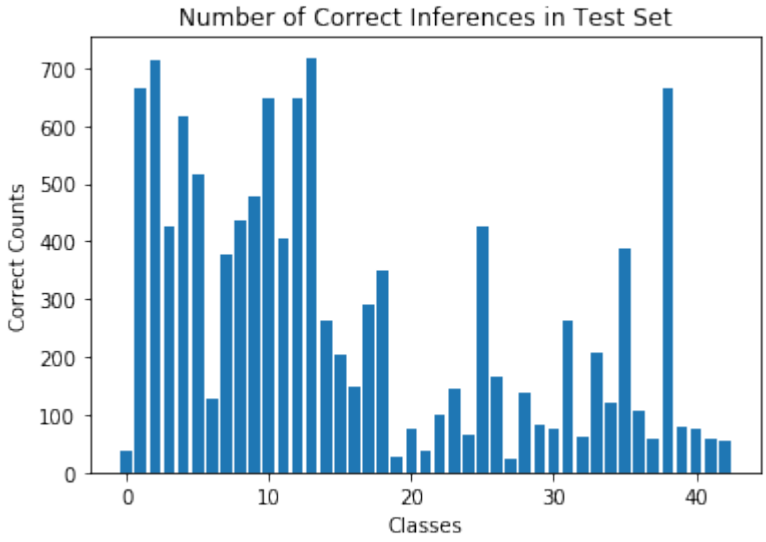

Figure 6: Correct Inferences on Testing Set

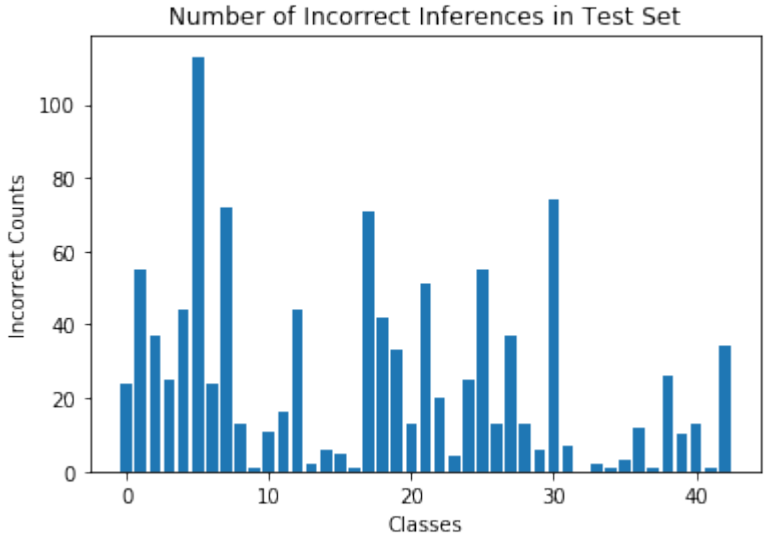

Figure 7: Wrong Inferences on Testing Set

5.2 Internet Results

Eight images were pulled from the internet of German traffic signs and were fed through the network. The test accuracy of the internet set was 62.5%. The breakdown of correct and wrong inferences can be seen in Figures 8 and 9 respectively. From a purely qualitative point of view,

there does not seem to be any correlation between the correct inferences based upon class between the test and internet set but this could be due to the small number of samples in the set.

One of the major reasons for the low accuracy on the internet set is that every single image had a watermark on it. This is difficult to see in the resized image to fit the rest of the dataset but this watermark would definitely have influenced the features that the network used to classify each image.

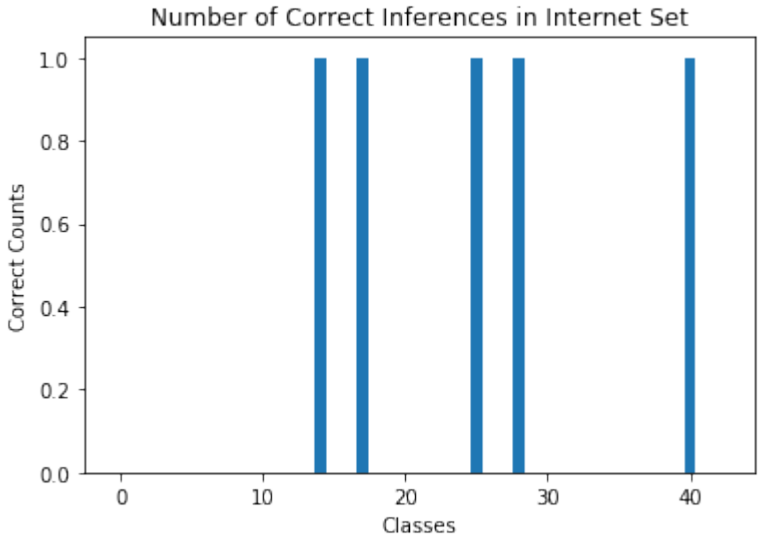

Figure 8: Correct Inferences on Internet Set

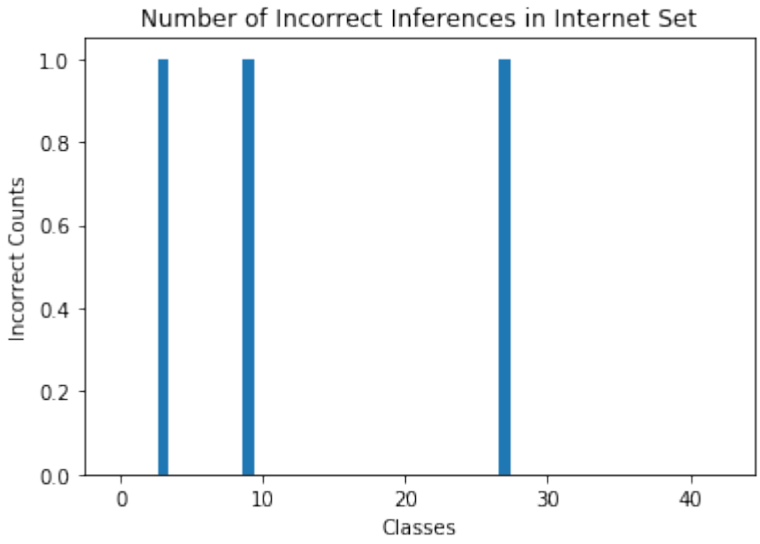

Figure 9: Wrong Inferences on Internet Set

5.3 Softmax Probabilities

The top 5 softmax probabilities of each of the images in the internet set were included in the code. Half of the images were classified with an assurance of over 85% while three were at less than 50%.

5.4 Inference Time

The time of inference for a single forward pass through the network was 0.5 microseconds. This is a very fast time and makes this an ideal network for deployment on an autonomous vehicle.

Autonomous vehicles must make decisions at no slower than human speeds (roughly 100,000 microseconds) and this image classifier would only make up a small part of the driving process. A more complex network with more convolutional layers could be added to increase the representational power of the network but this added complexity would also increase the inference time dramatically. It was decided that this network was a good middle ground between accuracy and plausibility.

6 Visualize the Models Learned Weights

The visualization of my network provided valuable insight into the inner workings of the network. It can clearly be seen in the 0th, 1st, 4th, and 5th feature map of the first layer that the network has learned feature detectors and is picking out features in the input image as seen in 10. Since the network is so small, it is impossible to determine any features by the second layer as seen in 11.

Due to the small network architecture and seeing what the internal network state is, it can be concluded that adding another convolutional layer could be beneficial to the representation power and accuracy of this classifying network without adding irredeemable compute time. This however was not explored because a network like this would most likely never be deployed on an actual autonomous vehicle because you would need an algorithm before this that would give region proposals that could then be fed into this network. Or a semantic segmentation algorithm could be used or the vastly popular RNN or any of its "fast" counterparts.

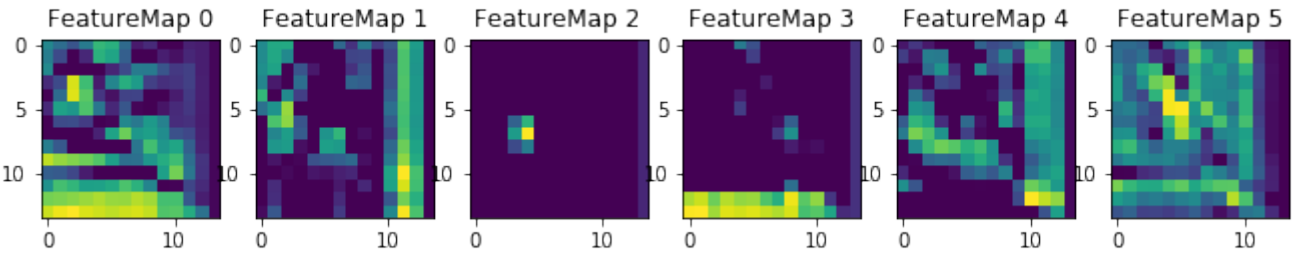

Figure 10: Layer 1 Visualization

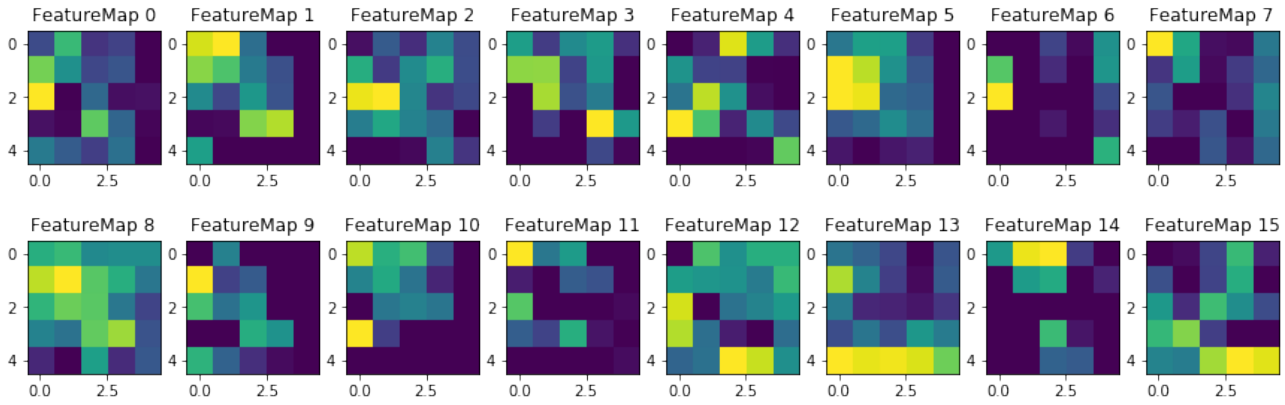

Figure 11: Layer 2 Visualization

References

[1] Martín Abadi, Ashish Agarwal, Paul Barham, Eugene Brevdo, Zhifeng Chen, Craig Citro, Greg S. Corrado, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Ian Goodfellow, Andrew Harp, Geoffrey Irving, Michael Isard, Yangqing Jia, Rafal Jozefowicz, Lukasz

Kaiser, Manjunath Kudlur, Josh Levenberg, Dan Mané, Rajat Monga, Sherry Moore, Derek Murray, Chris Olah, Mike Schuster, Jonathon Shlens, Benoit Steiner, Ilya Sutskever, Kunal Talwar, Paul Tucker, Vincent Vanhoucke, Vijay Vasudevan, Fernanda Viégas, Oriol Vinyals, Pete Warden, Martin Wattenberg, Martin Wicke, Yuan Yu, and Xiaoqiang Zheng. TensorFlow: Large-scale machine learning on heterogeneous systems, 2015. Software available from tensorflow.org. URL: http://tensorflow.org/.

[2] Geoffrey E. Hinton Alex Krizhevsky, Ilya Sutskever. Imagenet classification with deep convolutional neural networks. URL: http://papers.nips.cc/paper/4824-imagenet-classification-with-deep-convolutional-neural-networks.pdf.

[3] Jimmy Lei Ba Diederik P. Kingma. Adam: A method for stochastic optimization. URL: https://arxiv.org/pdf/1412.6980.pdf.

[4] J. Stallkamp, M. Schlipsing, J. Salmen, and C. Igel. Man vs. computer: Benchmarking machine learning algorithms for traffic sign recognition. *Neural Networks*, (0):–, 2012. URL: http://www.sciencedirect.com/science/article/pii/S0893608012000457, doi:10.1016/j.neunet.2012.02.016.

[5] Yoshua Bengio Patrick Haffner Yann LeCun, Leon Bottou. Gradient based learning applied to document recognition. URL: http://yann.lecun.com/exdb/publis/pdf/lecun-01a.pdf.