Convolutional Neural Networks for Classification of Plankton Shadowgrams

Chris Curro, Harrison Zhao
Department of Electrical Engineering
The Cooper Union
New York, NY 10003
{curro,zhao3}@cooper.edu

Abstract

We present various convolutional neural network architectures, as well as a nearly end-to-end procedure for the classification of plankton shadowgrams. This processing contains preprocessing of the images, feature extraction, and classification. This work was done as a part of the National Data Science Bowl, 2015.

1 Introduction

As a part of an initiative to grow excitement about data science, Kaggle along with the Oregon State University Hatfield Marine Science Center, supported by Booz Allen Hamilton, organized the first ever National Data Science Bowl. This year's National Data Science Bowl challenged entrants to label a corpus of plankton images. We selected a convolutional neural network (CNN) approach to this problem due to CNN's increasingly impressive performance on object recognition benchmarks such as ImageNet [1–3].

1.1 Data source

The Hatfield Marine Science Center collected and labeled a large corpus of of images taken with their In Situ Ichthyoplankton Imaging System (ISIIS) [4]. This system produces a series of shadowgrams; because of the shadowgraph technique regardless of an object's distance to the sensor it appears same size. In other words there is a fixed conversion factor between units of length in the real world and pixels in the sampled images. When sampled, the images are parsed, analyzed, and segmented to discover regions of interest (ROI). The segmented ROI are then hand labeled as any of 121 classes. The classes include various species of plankton, as well as other classes such as "segmentation artifact" and "unknown."

1.2 Neural networks

1.2.1 Fully connected networks

Traditionally a neural network is a computational graph where nodes can be defined in the form:

$$\mathbf{y} = f\left(W\mathbf{x} + \mathbf{b}\right) \tag{1}$$

Where $\mathbf{x}$ is an input vector, W and b are parameters that define a linear relationship, and $f(\cdot)$ is an activation function. The activation function is generally a non-linear function so each node in the graph can perform a non-linear mapping from $\mathbf{x}$ to $\mathbf{y}$.

Nodes can be connected successively to one another in a feed-forward fashion to create increasingly complex representations of the input data. Nodes in these networks are generally referred to as layers, and can be represented diagrammatically as in Figure 1.

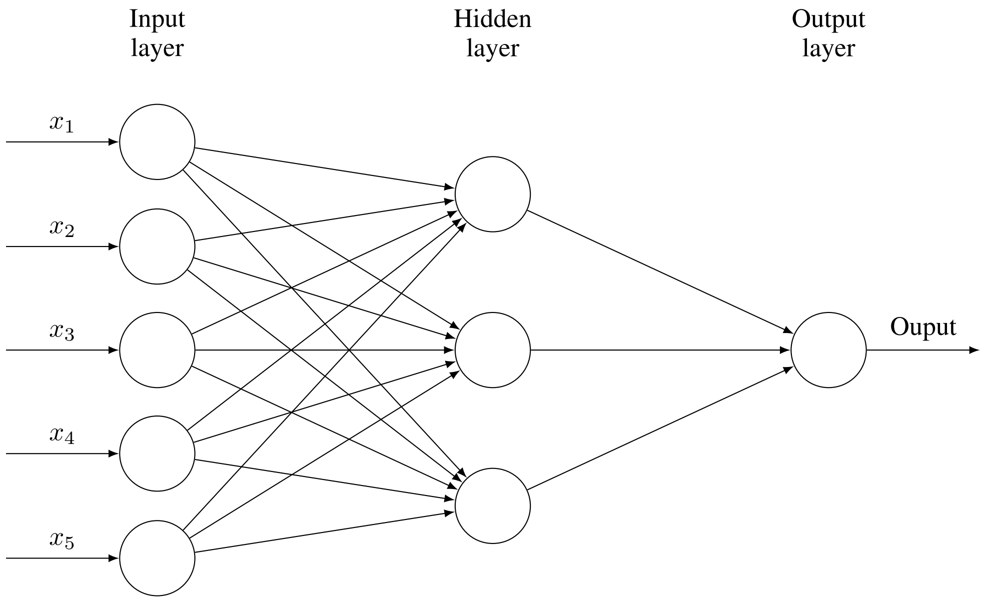

Figure 1: A single hidden layer feed-forward neural network. Each directed connection has a scaling factor w_{ij} associated with it; these entries make up the W matrix. Each circle, now referred to as a neuron, performs the sum and activation functions corresponding to the matrix multiply, and mapping by $f(\cdot)$ in Equation 1. Layers in the middle of the network are referred to as hidden layers because they are not directly observable — instead these are latent variables.

Figure 1 shows a feed-forward network with one hidden-layer. The outputs of the hidden layer are referred to as latent variables and are a new type of representation for the input data $\mathbf{x}$. With each hidden-layer added to the network increasingly complex data representations may be available, but the networks will also tend to over-fit their training data as the number of parameters increase. In order to combat over-fitting, and increase the generalization performance of the network new types of architectures have been proposed.

1.2.2 Convolutional neural networks

The first convolutional neural network was proposed by Fukushima in 1980 [5]. The idea of the network is that instead of creating a weight matrix W with parameters that correspond to every entry in $\mathbf{x}$, that there would be a kernel of some small size that would be swept across across $\mathbf{x}$. This process is essentially a convolution. Mathematically this process is:

$$\mathbf{y} = f(\mathbf{x} \star \mathbf{w}) \tag{2}$$

Where $\star$ represents a discrete convolution, $\mathbf{w}$ is a one-dimensional kernel and $f(\cdot)$ is an activation function. This method can be extended to work for two and three-dimensional data representations, for example:

$$Y = f(X \star W) \tag{3}$$

where Y, X and W are all matrices.

The generalization power of the convolutional architecture comes from the decreased number of parameters compared to the fully connected (also known as dense) layers described in Section 1.2.1. Convolutional neural networks are especially applicable and powerful for processing data where there is a temporal or spatial relationship between adjacent entries in $\mathbf{x}$ or X. The assumption of a convolutional neural network is that the input data has entries that are correlated with their neighboring entries and that these patterns may appear anywhere in the vector representation.

Some early work from LeCun et al. [6] demonstrated the power of convolutional architectures in image processing tasks, such as handwritten digit recognition, but many more advances were re-

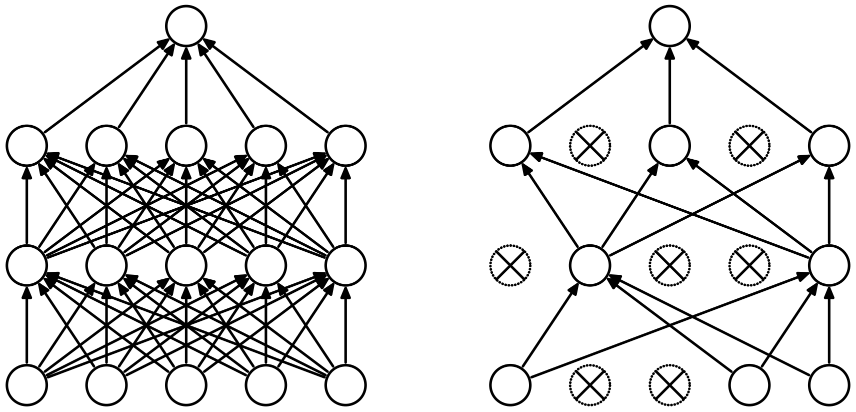

Figure 2: An ordinary fully connected neural network on the left, and the same network with dropout masks applied on the right. Diagram from Srivastava et al. [7].

quired before CNNs began achieving state of the art performance on a variety of image processing challenges.

1.2.3 Increased generalization performance

Dropout regularization Various teams have demonstrated that in deep networks neurons can learn complex co-adaptation schemes; that is deep neurons respond to "mistakes" in shallower neurons. In order to prevent co-adaptation, so that each neuron learns a meaningful representation, the dropout scheme has been proposed [7–9]. Dropout is a masking process. In order to apply dropout to a layer during training, a binary mask is applied across neurons. The mask is determined by sampling a Bernoulli distribution with a parameter p corresponding to the probability that a neuron will be masked. When a neuron is masked its output is considered fixed at zero. When the network is used for evaluation no masks are applied and all neurons are connected.

Weight Decay Krogh et al. adapted common regularization techniques that punish large weight parameters for neural network optimization algorithms [10]. They demonstrated that the weight decay can suppress any irrelevant components in the weight vector leading to the simplest solution for the learning problem, and that if the decay parameter is chosen correctly it can suppress the effects of static noise on the targets, which improves generalization.

Ensemble methods Geman et al. have shown that neural networks while having low bias can suffer from high variance [11]. The bias/variance trade-off is something that holds for single models. Because of this, increased generalization performance can be rendered by simply averaging different model's predictions. The ensembled predictions can be taken from models with different initializations as well as different architectures, both will realize a decreased variance in the effective final model.

1.2.4 Reducing training time

These networks are trained in an iterative fashion example-wise and with various optimization algorithms. Training large networks can take extraordinarily long times. This is both because iterating on a single example is slow, and because it can require an extremely large number of iterations to converge.

Mini-batch training Traditionally there were two different approaches to optimizing neural network parameters, the online approach and the batch approach. In the online approach, the parameters of the network are updated after each exposure to a training example and gradient calculation. In the batch approach, the network is exposed to all of the training examples, the gradients are accumulated and then the network's parameters are updated. This was long believed to be the better approach, as the accumulated and averaged gradient was more likely to be an estimate of the "true" gradient of the network towards an optimized solution. It was later shown that, while the batch mode may give a better estimate of the gradient, due to the noise and its stochastic nature, the online approach actually leads to a faster convergence time, in terms of number of examples [12]. This is because in the stochastic approach the optimizer is less likely to get stuck in local minima. An alternative approach that recently has become popular is the mini-batch approach. In this approach some number of examples, say N, are exposed to the network, the gradients are accumulated, and the parameters are updated. In this case N is much less than the total number of training examples available. This approach, with serial computational resources really only represents a decrease in training speed, because it is fairly similar to the batch approach. The reason this approach has become popular lately is that with the parallel resources afforded by modern high performance graphics processing units (GPUs) the increase in example-wise training time becomes a decrease in wall-time to train.

Nesterov descent Most networks are trained with a variant of stochastic gradient descent. A particular variant that is becoming more popular is Nesterov accelerated gradient descent [13–15] Nesterov accelerated gradient descent utilizes a particular variant of momentum to achieve a time constant of $1/t^2$ which represents one order of improvement over ordinary stochastic gradient descent. By adding momentum to the optimization, gradients from past steps are still accounted for in an amount corresponding to a scaling parameter. Momentum essentially steadies the gradient a little bit over time, so that the optimization algorithm is less likely to get trapped in local minima.

Rectified linear units Various activations have been proposed on the basis of similarity to the potential activations in actual biological neurons in human eyes. Recently the rectified linear unit (ReLU) has demonstrated significant performance increases for network generalization and increased training speed [9, 16]. The ReLU activation is defined as $\max(0, x)$. In other words a ReLU activation forwards along any positive inputs and sets and negative inputs to zero.

Intelligent parameter initialization Kaiming et al. have demonstrated an improved parameter initialization technique specifically designed for neurons with ReLU activations [3]. This approach derives a method that enables extremely deep models comprised of ReLU neurons to converge rather than stall, as they would with other initialization schemes. The initial parameters for the convolutional layers are drawn from a zero mean normal distribution with a standard deviation of $\sqrt{2/n_l}$. Where $n_l = k^2c$. This corresponds to the number of connections in the response for $k \times k$ kernels processing c input channels.

2 System description

The system while almost an end-to-end system has several ancillary stages. We take you through the stages here. First we offer some light preprocessing, including data augmentation through a series of affine transformations. Then we discuss the architecture of our best single model, as well as how we trained that model. Finally we discuss the ensembling methods used to improve generalization performance.

2.1 Preprocessing

Convolutional neural networks are a type of method that require extremely large datasets to generalize well. For example in the object recognition tasks undertaken by various authors [1–3] the dataset required to exceed human level performance consists of approximately 1 million images. In order simulate a larger dataset, we make sets of transformations that preserve the class labels. In this case we make use of affine transformations that can be represented by matrix arithmetic over the coordinate space. Our affine transformations include translation, scaling, rotation, shearing, and aspect ratio manipulation. Additionally we also employ random vertical and horizontal flips. Parameters for the affine transformations were empirically determined such that they preserve class labels. Additional preprocessing included inverting the source images, and rescaling them to 128×128 pixels, centering the region of interest according to its centroid, and thresholding pixel values within 1% of maximum intensity. This thresholding technique was employed because we noticed noise in the high intensity pixels due to the automatic segmentation algorithm used to produce the dataset.

Table 1: Below is the architecture for our best single model. All kernel sizes are given as $a \times b$ with the number of kernels denoted immediately afterwards. Strides, other than one, are indicated by $_{/s}$. All pooling layers are ordinary max pooling except for the final layer pool4 with is an adaptive max pooling layer that always outputs a matrix of fixed 4×4 size.

Layers		Activation
conv1_1	$7 \times 7, 64,_{/2}$	ReLU
pool1	$2 \times 2,_{/2}$	
conv2_1	$3 \times 3, 256$	ReLU
conv2_2	$3 \times 3, 256$	ReLU
pool2	$2 \times 2,_{/2}$	
conv3_1	$3 \times 3, 512$	ReLU
conv3_2	$3 \times 3, 512$	ReLU
pool3	$2 \times 2,_{/2}$	
conv4_1	$3 \times 3, 512$	ReLU
conv4_2	$3 \times 3, 512$	ReLU
pool4	4×4	
fc1	3000	SoftSign
fc2	121	SoftMax

2.2 Network architecture

The architecture for our best single model can be seen in Table 1.

2.3 Training

Our network was trained with Torch 7 [17] on two Nvidia Tesla K40's. We used Nesterov accelerated gradient descent with a momentum of 0.9. The initial learning rate was 0.01 with a scheduled 20% decay every five epochs (30,000 exposures per epoch.) The mini-batch size was 32. Weight decay was used; the regularization parameter was 1×10^{-4}. The network was optimized to minimize the negative log loss.

2.4 Prediction generation

An ensemble of 40 networks were used in the generation of the final prediction. Each network was trained from initializations according to the method devised by Kaiming et al. [3]. The specific architectures used were picked to maximize the variance between each model. The changes in architecture included the number of hidden layers, the inclusion of different activation functions, and the adjustment of the number units in the hidden layers. Additionally a random affine transformation was applied to each image before being propagated through the networks to further increase the variance; each affine transformation was different for each network. The predictions from each network were equally weighted in an arithmetic mean to produce the final predictions.

3 Results

Our local cross-validation results for individual models matched very well the Kaggle leaderboard indicating good, predictable, generalization performance. Our final position on the leaderboard is 29 out of 1024 indicating performance in the 97th percentile. Our final score is a 0.649712 negative log loss.

4 Conclusion

We have demonstrated a nearly end-to-end system for species classification of plankton shadowgrams with state of the art performance.

Acknowledgments

We would like to thank Professor Sam Keene for his advisement, as well as David Katz, Caleb Zulawski, and Ethan Lusterman for supporting us in their own special way.

References

[1] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "Imagenet classification with deep convolutional neural networks," in *Advances in Neural Information Processing Systems 25*, F. Pereira, C. Burges, L. Bottou, and K. Weinberger, Eds. Curran Associates, Inc., 2012, pp. 1097–1105. [Online]. Available: http://papers.nips.cc/paper/4824-imagenet-classification-with-deep-convolutional-neural-networks.pdf

[2] C. Szegedy, W. Liu, Y. Jia, P. Sermanet, S. Reed, D. Anguelov, D. Erhan, V. Vanhoucke, and A. Rabinovich, "Going deeper with convolutions," *CoRR*, vol. abs/1409.4842, 2014. [Online]. Available: http://arxiv.org/abs/1409.4842

[3] K. He, X. Zhang, S. Ren, and J. Sun, "Delving deep into rectifiers: Surpassing human-level performance on imagenet classification," *CoRR*, vol. abs/1502.01852, 2015. [Online]. Available: http://arxiv.org/abs/1502.01852

[4] J. Luo, "Frequently asked questions," Hatfield Marine Science Center, 2014. [Online]. Available: https://www.kaggle.com/c/datasciencebowl/data

[5] K. Fukushima, "Neocognitron: A self-organizing neural network model for a mechanism of pattern recognition unaffected by shift in position," *Biological Cybernetics*, vol. 36, no. 4, pp. 193–202, 1980. [Online]. Available: http://dx.doi.org/10.1007/BF00344251

[6] Y. LeCun, B. Boser, J. S. Denker, D. Henderson, R. E. Howard, W. Hubbard, and L. D. Jackel, "Backpropagation applied to handwritten zip code recognition," *Neural computation*, vol. 1, no. 4, pp. 541–551, 1989.

[7] N. Srivastava, G. Hinton, A. Krizhevsky, I. Sutskever, and R. Salakhutdinov, "Dropout: A simple way to prevent neural networks from overfitting," *Journal of Machine Learning Research*, vol. 15, pp. 1929–1958, 2014. [Online]. Available: http://jmlr.org/papers/v15/srivastava14a.html

[8] G. E. Hinton, N. Srivastava, A. Krizhevsky, I. Sutskever, and R. Salakhutdinov, "Improving neural networks by preventing co-adaptation of feature detectors," *CoRR*, vol. abs/1207.0580, 2012. [Online]. Available: http://arxiv.org/abs/1207.0580

[9] G. E. Dahl, T. N. Sainath, and G. E. Hinton, "Improving deep neural networks for lvcsr using rectified linear units and dropout," in *Acoustics, Speech and Signal Processing (ICASSP), 2013 IEEE International Conference on*. IEEE, 2013, pp. 8609–8613.

[10] A. Krogh and J. A. Hertz, "A simple weight decay can improve generalization," in *ADVANCES IN NEURAL INFORMATION PROCESSING SYSTEMS 4*. Morgan Kaufmann, 1992, pp. 950–957.

[11] S. Geman, E. Bienenstock, and R. Doursat, "Neural networks and the bias/variance dilemma," *Neural computation*, vol. 4, no. 1, pp. 1–58, 1992.

[12] D. R. Wilson and T. R. Martinez, "The general inefficiency of batch training for gradient descent learning," *Neural Netw.*, vol. 16, no. 10, pp. 1429–1451, Dec. 2003. [Online]. Available: http://dx.doi.org/10.1016/S0893-6080(03)00138-2

[13] Y. Nesterov *et al.*, "Gradient methods for minimizing composite objective function," 2007.

[14] X. Glorot and Y. Bengio, "Understanding the difficulty of training deep feedforward neural networks," in *International conference on artificial intelligence and statistics*, 2010, pp. 249–256.

[15] I. Sutskever, J. Martens, G. Dahl, and G. Hinton, "On the importance of initialization and momentum in deep learning," in *Proceedings of the 30th International Conference on Machine Learning (ICML-13)*, 2013, pp. 1139–1147.

[16] X. Glorot, A. Bordes, and Y. Bengio, "Deep sparse rectifier networks," in *Proceedings of the 14th International Conference on Artificial Intelligence and Statistics. JMLR W&CP Volume*, vol. 15, 2011, pp. 315–323.

[17] R. Collobert, K. Kavukcuoglu, and C. Farabet, “Torch7: A matlab-like environment for machine learning,” in *BigLearn, NIPS Workshop*, no. EPFL-CONF-192376, 2011.

[18] M. D. Zeiler and R. Fergus, “Visualizing and understanding convolutional networks,” *CoRR*, vol. abs/1311.2901, 2013. [Online]. Available: http://arxiv.org/abs/1311.2901