The Virtues of Peer Pressure: A Simple Method for Discovering High-Value Mistakes

Shumeet Baluja, Michele Covell, Rahul Sukthankar

Google Research

Abstract. Much of the recent success of neural networks can be attributed to the deeper architectures that have become prevalent. However, the deeper architectures often yield unintelligible solutions, require enormous amounts of labeled data, and still remain brittle and easily broken. In this paper, we present a method to efficiently and intuitively discover input instances that are misclassified by well-trained neural networks. As in previous studies, we can identify instances that are so similar to previously seen examples such that the transformation is visually imperceptible. Additionally, unlike in previous studies, we can also generate mistakes that are significantly different from any training sample, while, importantly, still remaining in the space of samples that the network should be able to classify correctly. This is achieved by training a basket of N "peer networks" rather than a single network. These are similarly trained networks that serve to provide consistency pressure on each other. When an example is found for which a single network, S, disagrees with all of the other $N-1$ networks, which are consistent in their prediction, that example is a potential mistake for S. We present a simple method to find such examples and demonstrate it on two visual tasks. The examples discovered yield realistic images that clearly illuminate the weaknesses of the trained models, as well as provide a source of numerous, diverse, labeled-training samples.

1 Introduction

The recent rapid resurgence of interest in deep neural networks has been spurred by state of the art performance on vision and speech tasks. However, despite their impressive performance, the deeper architectures require enormous amounts of labeled data and are surprisingly fragile. Additionally, because the training is done through following derivatives in high-dimensional spaces and often through ad-hoc architectures and input groupings, the results can be unintelligible with surprising error modes [9, 13, 16].

The three problems of needing vast amounts of training data, being brittle, and being unintelligible are interrelated. In this paper, we address them by finding *high-value* mistakes that the network makes. A high-value mistake is one where the network is expected to perform correctly and does not – e.g., the input lies in the space of realistic inputs for the task and yet is misclassified. Finding where these mistakes are likely to occur yields insight into the computation of the network. These mistakes can also be used to further augment the training set to mitigate the brittle decision boundaries that may have been created with only the original training examples. Figure 1 provides a visual example of high-value and low-value mistakes for the familiar MNIST dataset.

It shows examples that are classified perfectly by a trained network, examples that are misclassified, but should be classified well (high-value mistakes), and examples that are not classified well, but are outside the set of examples the network is trained on (low-value mistakes). If too many low-value mistakes are used for retraining the networks, or any machine learning classification tools, they face the problem of artificial concept drift [15]; the training examples no longer become representative of the actual problem.

Fig. 1: (Left) 49 digits that the network classified correctly. (Middle) Examples of *high-value* mistakes generated by our method. They are digits that should be classified correctly, since the images are of the same type as the training images. (Right) Examples of *low-value* mistakes. They all contain individual pixel-based noise which, though small, is not representative of the types of images the network was designed to recognize.

Through the last three decades, a variety of approaches have been explored to understand the performance of neural networks and to generate new examples based on their encoded computations. In a broad sense, the approach taken here is *behaviorist* in nature. We do not look at the internal states of the network to determine how it will perform; instead, we observe how the network responds to different inputs. Similar approaches were described in [1, 7]. Alternative approaches interpret the internal states of a neural network by discovering what the hidden units and layers encode [3, 13, 14].

The remainder of the paper is structured as follows. Section 2 details the proposed method for finding high-value mistakes; a simple, two-dimensional, example is provided to concretely demonstrate the algorithms. Section 3 describes experiments on the 10-digit MNIST database. Section 4 describes experiments with state-of-the-art networks trained to classify real-world images into 1,000 categories. Section 5 presents further experiments to illuminate limitations of the approach. Finally, Section 6 concludes the paper and presents directions for future work.

2 Using Peer Networks to Find Inconsistencies in Training

The underlying decision boundaries inferred by well trained classification models depend on a number of factors: the training examples used, the learning algorithms employed, the exact architecture and form of the trained models, and the proper use of

validation sets and hold-out sets, to name a few. With neural networks, further unpredictability in the decision boundaries arises because the training is sensitive to the order in which the examples are presented, the initial weights, and the numerous hyper-parameters associated with typical training algorithms such a backpropagation.

With this in mind, consider the simple two-class problem of classifying points on a checkerboard: does a given point fall on a red of black square? This is a version of the X-OR problem often used with neural networks; see Figure 2.

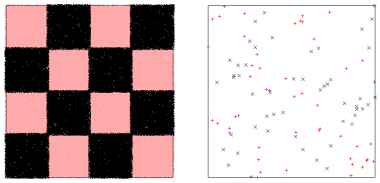

Fig. 2: (Left) A toy two-dimensional, two-class, classification problem. (Right) 200 randomly selected samples used for training a network (100 from each class).

To concretely show the different learning boundaries created, five neural networks with 10, 12, 14, 16 & 18 units per hidden-layer were trained on 200 sample points. Each network had two inputs (x and y coordinates), 2 fully-connected hidden layers, and 1 output classification node. The inferred classification boundaries are shown in Figure 3.

Fig. 3: Decision boundaries of the five networks trained on the toy task in Figure 2. Close inspection reveals many differences.

Though this variability may be initially considered a limitation in training, an enormous amount of research has already shown the benefits of using these individual networks as members of an *ensemble classifier* (e.g., [4]). In ensemble methods, the outputs of multiple networks are combined through any number of voting schemes to output the final classification. In contrast to typical ensemble methods, this study uses the variability in networks to find potential mistakes.

In this simple example, for any target network, A, we scan through valid values of x & y to find points where A disagrees with all of its peers (the other 4 networks). Such points are good candidates for new training examples for network A because all of the other similarly trained networks were able to classify them correctly (see Figure 4). The labels for these points can either be provided as a supervisory signal (analogous to an

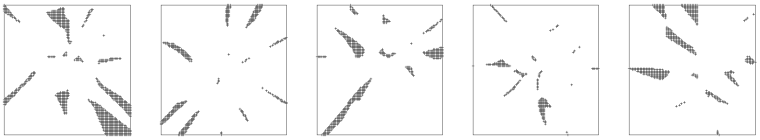

Fig. 4: Sample points where each network disagrees with *all* of its peers. Note that each falls near the decision boundaries shown in Figure 3.

active learning approach where the learner requests labels for selected instances [2]) or can be supplied by the consistent labeling generated by the peers.[1]

The toy 2-D example allows complete exploration of the input space. However, for more realistic problems, such as image classification one cannot consider the set of all potential input images. Not only is the space of possible images too large, but the set of all images is vastly dominated by unrealistic pixel combinations (e.g., white noise) that will not appear either during training or inference.

Instead, to find high-value-examples, we must either have a set of promising candidates (e.g., a repository of unlabeled data) or we can synthesize extra instances from known good images. With both of these approaches, we use the consistent classification of the peer networks as a filter to select images. For the MNIST task, we take the latter approach: we search the space of transformations on image X for an input image X' where for a given network A and $\forall$ networks $p_i, p_j, p_k \in$ peer networks $\{B,\ C,\ D,\ E\}$, we attempt to find examples X' that simultaneously:

$$\min\{\text{DIST}(p_i(X'), p_j(X'))\} \quad \text{and} \quad \max\{\text{DIST}(A(X'), p_k(X')\},$$

where $\text{DIST}(p_i(X'), p_j(X'))$ is the magnitude of the difference in classifier outputs between networks p_i and p_j when applied to X' (this is minimized to ensure consistency between peers). $\text{DIST}(A(X'), p_k(X'))$ is the magnitude of the difference between the target network, A, and its peers (maximized to ensure A and its peers are different).

How do we find this image? Naively sampling random images to find one that maximizes/minimizes these two objectives simultaneously would be prohibitively slow. Instead, we search through the space of potential images by stochastically perturbing an example digit to satisfy the above constraints. The algorithm in its simplest form is described in the procedure below.

Illustrating using the MNIST example, we start by randomly selecting a digit image from the training set (seed). We also randomly generate a small affine transformation (e.g., small rotation, translation, stretch). If applying this transform to the seed increases the difference between the output activation of the target network and its peers (with the peers still agreeing on their prediction), then we keep the transformed image and use it as the seed for the next iteration. We compose transforms in this manner until we either find a sequence of transforms that causes a misclassification by the target network (with peers still classifying correctly) or until we try too many transforms for the given seed — in which case we restart with a new seed image drawn from the training set.

[1] This raises the question: The worst performing network in the set will likely see improvement using this method, but will *all* the networks? This is an immediate area for future research.

- Start with seed image, X. If X is not classified in the same class, C, for all of the networks, find a different seed.
- done = False
- while (not done):
  - Generate a candidate, X', with a stochastically chosen constrained image transform on X.
  - If X' is not classified in the same class, C, by all the peers, reject X'. *(this ensures consistency of peers).*
  - Else: Measure network-A activation w/input X' for class C: $A(X')_C$.
    If $(A(X')_C < A(X)_C)$ then accept X' ; $X \leftarrow X'$
    Else: Reject X'.
  - done = network-A classifies X' sufficiently differently than peers do. e.g., minimize $A(X')_C$ where $\forall P \in$ Peers, $P(X') = C$.

Note for labeled data: In the cases where we know the label L of initial image X, we simply ensure that $C = L$ throughout the run.

Alternatives to next-ascent hillclimbing include more sophisticated search heuristics. Evolutionary algorithms such as genetic algorithms, population-based incremental hillclimbing or other evolutionary strategies may be used to search the space of image transforms; see [10]. Like hillclimbing, these quickly search discrete spaces without the need for derivatives [1, 7].

In the next section, we apply the proposed algorithm to networks trained to recognize MNIST digits. To keep our experiments reproducible, we use simple hillclimbing as the search mechanism. Because of space restrictions, we only note there that we have also successfully applied several other evolutionary algorithms on this task with similar qualitative results, although the efficiency varied greatly depending on the specific algorithm and search heuristics.

3 Result Set I: The MNIST Digit Database

For the experiments with MNIST [8], 60,000 digit images were used for training five different networks. Other than the number of hidden units in the networks (10, 20, 30, 40 & 50), the networks were identical and trained identically. Samples from the MNIST database are shown in Figure 1.

Figure 5 shows a typical hillclimbing session to obtain mistake images. Within five transformations (starting from the top), the target network misclassifies the digit '9' as a '3' with high confidence, yet its peer networks confidently predict the correct class for all transformations. Note that the sequence of constrained transforms results in an image that is still unambiguously a '9' for humans.

This image is a high-value mistake because the target network strongly misclassifies it while its four peers are confident in the correct prediction. Based on the sequence of transforms that generated this mistake, it appears as though the target network may be susceptible to small amounts of shrinking, to which its peers are robust. Adding this

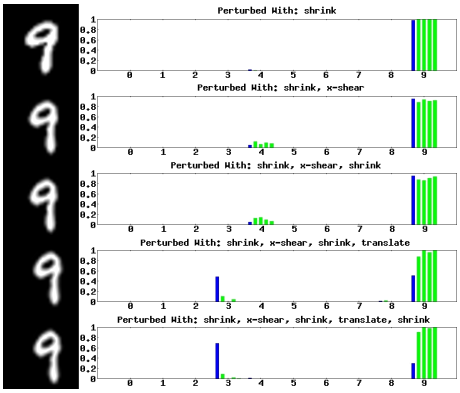

Fig. 5: A typical hillclimbing session. The leftmost of the bars in each group (blue) is the output activation of the target network. The four other bars (green) show the peers' activations. Top row: the '9' is correctly and confidently classified by all networks (all activations are close to 1.0). Bottom row: within five operations, the target network misclassifies the '9' as a '3', but its peers continue classifying it correctly with high confidence.

mistake to the target network's training set may help to make it more robust to such transforms. Additional examples are shown in Figure 6.

On average, only five small transforms were required to change a training image into one that is incorrectly classified. However, many more candidate images were evaluated by the networks in the process of finding this sequence of five moves. This is typical with stochastic search; the other candidates were rejected because they did not improve the objective function.

To give an indication of how much effort it takes to find a potential mistake, in Table 1, the results with over 1,000 trials per network are shown. In the first row, network A was chosen as the one for which errors were found, and networks B, C, D, and E were chosen as its peers. For each of the 1,000 trials, a random digit was selected as a seed and next-ascent-stochastic hillclimbing was used to search the space of affine

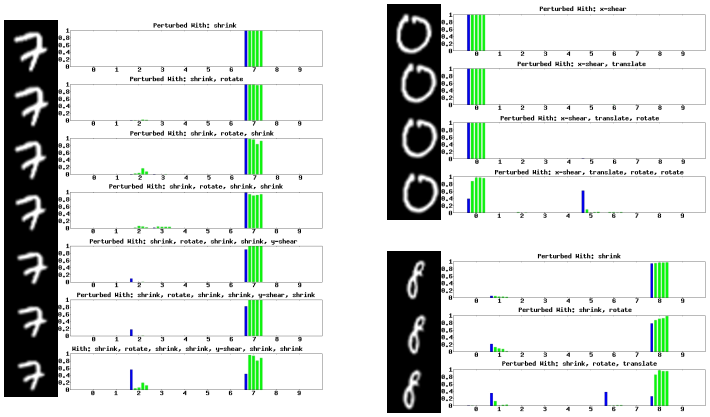

Fig. 6: Three hillclimbing sessions. Left: '7' mistaken as '2' in 7 image transformations. Right-Top: '0' mistaken as '5' in 4 moves. Right-Bottom: '8' mistaken as both '1' and '6' within only 3 moves. Note that in all of these examples, the peers still correctly identified the digit, while the target network did not.

Table 1: Inferences and moves required to find a high-value mistake per network

Target network	Hidden Nodes	% Successful Trials	# moves (average over successful trials)	# forward inferences (average over successful trials)
A	10	63%	5	507
B	20	25%	5	640
C	30	9%	6	852
D	40	5%	6	801
E	50	7%	6	819

image transformation. 63% of the trials resulted in a mistaken classification (Figure 5, Figure 6). These 630 sequences are labeled successful. Trials in which 250 consecutive transformations were tested without improvement were marked as *un*successful.

The last column shows the number of forward inferences needed on the successful trials through the five networks. On average, 507 forward inferences were required per successful trial. As there are a total of five networks, this is approximately 101 forward inferences through each. This means that, for the successful trials, on average, 101 image transformations were also created and tested through the hillclimbing procedure before the transformation sequence that led to a high-value mistake was found.

The next rows in Table 1 repeat the same process for each of the other networks. For example, in the last row network E is considered the primary network and its peers are networks A, B, C, and D. This is the largest network of the ones tried, with 50 hidden units. Note that the procedure has a lower success rate in terms of mistakes found and requires more forward inferences. In general, the more robust the target network, the harder it is to find errors that other, worse performing, networks do not also make. Interestingly, although the hillclimbing procedure had to search longer, the average length of the transform sequence needed to generate a high-value mistake only rose to six.

4 Results Set II: Deep Vision Networks

We present some early results on applying peer pressure on the ImageNet [11] dataset. We trained five convolutional neural networks on the ImageNet training set to classify a given image into one of 1000 categories (synsets). Unlike in the MNIST experiments in Section 3, where the peers had slightly different architectures, all of the GoogLeNet networks employed the same GoogLeNet architecture [12]. GoogLeNet consists of a series of *Inception modules* shown in Figure 7 (Top) connected in the very deep structure, illustrated in Figure 7 (Bottom).

The input layer was a 224×224 RGB image. The inputs automatically subtracted the mean from the image as the first normalization. All units, including those in the Inception modules used rectified linear units (ReLU) as the nonlinearity. Given the challenge of propagating gradients through its significant depth (27 layers), GoogLeNet employs auxiliary classifiers, shown as side branches ending in "SoftMax" units in Figure 7, that connect to intermediate layers and boost the loss signal to provide data-driven regularization. For additional details about GoogLeNet, see [12].

The networks were explicitly trained to be robust to common image transformations, such as cropping and rotation by sampling crops from the original image at various sizes (8%–100%), aspect ratios (3/4–4/3) and photometric distortions [6]. Some of the transforms that can be handled are shown in Figure 8 (LEFT). Note the severe shearing, large black borders, rotation and color and intensity variations.

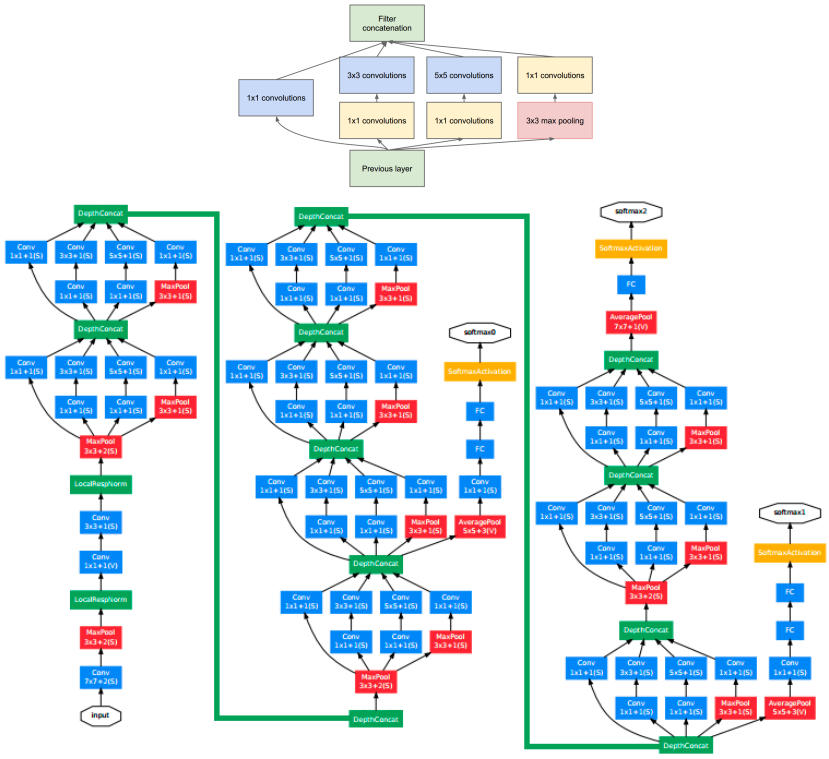

Fig. 7: (Top) A single Inception module. (Bottom) The GoogLeNet image classification network is built from many such modules. In this diagram, the pairs of "DepthConcat" units that are connected by heavy green lines are shown as pairs to make the diagram easier to fold — there is a single "DepthConcat" unit at each of these fold locations. Diagrams adapted from [12] with permission.

With such robust training, we initially believed that it would be difficult to find images that cause a single network to fail without also impacting the accuracy of its peers. However, in practice, this proved to not be the case. See Figure 8 (RIGHT). Each network had unique failure modes despite the observed resistance to numerous image transformations. Network 1 (shown) was often sensitive to lighting cues. Other networks' failures sometimes exhibited differences to individual classes; for example, varying degrees of robustness were exhibited with camera images than with hammer-

head sharks. Because of space restrictions, we only show the failure modes for a single network; however, all networks proved susceptible to the this mistake generation method, enabling us to quickly generate high-value training images for each network.

Fig. 8: (LEFT) All five networks are successfully trained to be robust to many severe transformations. The original image and the 8 transformed images are all classified correctly by all five networks. Note the variations in rotation, lighting, crop, the presence of black borders, etc. (RIGHT) For each column - left: original image, right: transformed image that the hillclimbing procedure found that caused an error for the target network. In this case, Network 1 (shown) was very sensitive to lighting conditions especially when coupled with small rotations or crops. Note that all the peers *correctly* classified these samples; thereby making these high-value mistakes for Network 1.

5 Limitations, Discussion and Further Experiments

In this section, we present two extra experiments conducted in the design of the algorithm. They are included to elucidate both the limitations and potential of our work.

One of the implicit assumptions throughout this paper has been that whatever is done for a network A with peers B, C, D, and E can be equivalently done for network B with peers A, C, D, and E. Thus, each network can serve as a peer for the other networks. In the next example, we seek a seed image from the training set, X, that can be transformed such that every network, in turn, will make a mistake on it while its peers maintain the correct classification. Of course, the sequence of transforms required will be different to drive each network to an error. Two such input images and their transformations are shown in Figure 9. Note, in general, we do not need to use the same image as a seed for different networks; however, this demonstrates that it is possible.

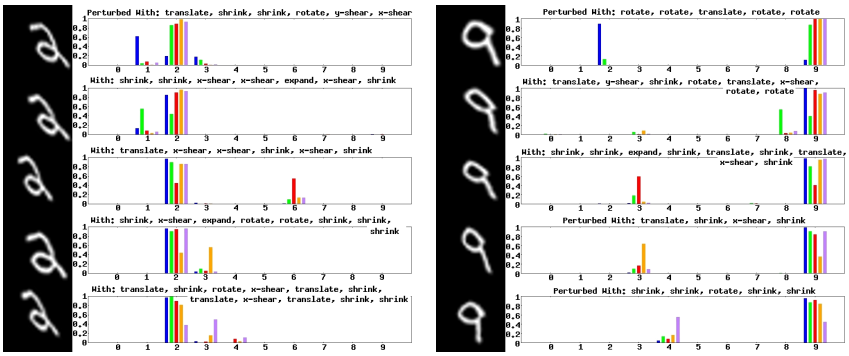

Fig. 9: (Left) The number '2' is mistaken by network A as a '1', by network B as '1', network C as '6', network D as '3' and network E as '3'. To find an error for each network, different transform sequences are found. (Right) Similar results for '9'.

Finally, it is important to explore the limitations of the proposed approach. In the experiments described above, we used small affine image transforms as the perturbation operator. These generate images that look reasonable by using the peer networks to ensure that radical transforms are suppressed (e.g., successively rotating a '6' until it becomes a '9'). This suggests a natural question: can peer pressure ensure that we only generate realistic images as mistakes?

We replaced the affine transforms in the proposed method with simple pixel swaps. Applying a sequence of pixel swaps to a seed image quickly yields noisy images that fall outside the domain of realistic inputs for this task. Nguyen et al. [9] examined similar effects (without the use of peers) in the extreme: white noise images that were classified incorrectly by trained networks. Unlike our study which attempts to find images that the network may encounter in practice, [9] showed that images far outside the training sets may be classified incorrectly with high confidence. We wanted to see whether peer pressure alone would suppress such images from being generated as valuable mistakes.

Unfortunately, as shown in Figure 1 (right), given the pixel swap operator, our method is able to find unrealistic images that are correctly classified by peers and misclassified by the target network. If the underlying image transformations are small, it is possible to "poke holes" in the target network's training in the precise locations where

they have little effect on the peers. Naively adding such images to the training set would be inadvisable as they are unlikely to help on the original task.

The benefit of peers is not lost, however. Peers make it more difficult to create such a mistake. We can examine the effect of peers by attempting to find a mistake in the network A with no peers, with 1 peer, etc., up to 4 peers. See Table 2.

Table 2: Effects of peers on difficulty of finding unrealistic mistakes

Number of peers	Successes (of 300)	#number of transforms
0	299	36
1	145	37
2	96	39
3	72	40
4	57	41

As can be seen in the table, without peers, 299/300 hillclimbing attempts transformed the seed image into a low-value mistake. This means that almost every run generated an unrealistic instance. The average number of pixel swaps it took to find the mistake was 36. For comparison, recall that the average number of affine transforms with 4 peers for MNIST was 5–6 moves. When peers are added, the number of successful trials drops sharply. With 4 peers, only 57 of 300 trials were successful in finding a mistake, and the average number of swaps increased to 41. This indicates what we hoped: having peers decreases the likelihood of escaping a reasonable input space.

6 Conclusions & Future Work

The successes of neural networks in real-world tasks has increased the need for finding ever increasing amounts of training data as well as finding insights into what the network has learned, and importantly, where it will fail. In this paper, we have shown that by training multiple "peer" networks to solve the same task, the inherent, unique, biases that are learned in the networks can be used to find mistakes that each network should have classified correctly. The mistakes lie within the space of examples that can be reasonably expected to be seen; thereby yielding valuable insights into the limitations of the trained models.

There are five immediate directions for future work. The most pressing is extending this work by augmenting the training of the networks with the generated high-value mistakes. To accomplish this, many decisions need to be made; for example, how are the new images weighted, and how frequently should they be introduced into training? Second, the effect of increasing the number of peers should be studied. Does adding more peers increase reliability at the expense of mistake coverage? Third, if all the networks were trained by using each others as peers, would they eventually reach a steady state and how soon would this happen? Does this then mimic the effects of using the ensemble for voting, and does this effectively become a technique for ensemble

compression [5]. Fourth, instead of using stochastic search, if appropriate derivatives can be created for an input transformation layer, that will yield an alternate way to find meaningful mistakes. Finally, only the simplest search algorithm was employed here; other multi-point search algorithms should also be evaluated for increased efficiency.

Acknowledgments

The authors gratefully acknowledge Christian Szegedy, George Toderici, and Jon Shlens for their discussions, code and help with GoogLeNet experiments.

References

1. Baluja, S.: Finding regions of uncertainty in learned models: An appliation to face detection. In: Parallel Problem Solving from Nature-1998. Springer (1998)
2. Cohn, D.A., Ghahramani, Z., Jordan, M.I.: Active learning with statistical models. Journal of Artificial Intelligence Research 4 (1996)
3. Erhan, D., Bengio, Y., Courville, A., Vincent, P.: Visualizing higher-layer features of a deep network. Tech. Rep. 1341, Université de Montréal (2009)
4. Hansen, L.K., Salamon, P.: Neural network ensembles. IEEE Transactions on Pattern Analysis and Machine Intelligence 12(10) (1990)
5. Hinton, G.E., Vinyals, O., Dean, J.: Distilling the knowledge in a neural network. In: NIPS Deep Learning Workshop (2014)
6. Howard, A.G.: Some improvements on deep convolutional neural network based image classification (2013), arXiv:1312.5402
7. Kindermann, J., Linden, A.: Inversion of neural networks by gradient descent. Parallel Computing 14(3) (1990)
8. LeCun, Y., Cortes, C., Burges, C.: The MNIST database of handwritten images (1998), http://yann.lecun.com/exdb/mnist/
9. Nguyen, A., Yosinksi, J., Clune, J.: Deep neural networks are easily fooled: High confidence predictions for unrecognizable images. In: Computer Vision and Pattern Recognition (2015)
10. Pelikan, M., Sastry, K., Cantú-Paz, E.: Scalable optimization via probabilistic modeling: From algorithms to applications, vol. 33. Springer Science & Business Media (2006)
11. Russakovsky, O., Deng, J., Su, H., Krause, J., Satheesh, S., Ma, S., Huang, Z., Karpathy, A., Khosla, A., Bernstein, M., Berg, A.C., Fei-Fei, L.: ImageNet large scale visual recognition challenge (2014), arXiv:1409.0575
12. Szegedy, C., Liu, W., Jia, Y., Sermanet, P., Reed, S., Anguelov, D., Erhan, D., Vanhoucke, V., Rabinovich, A.: Going deeper with convolutions (2014), arXiv:1409.4842
13. Szegedy, C., Zaremba, W., Sutskever, I., Bruna, J., Erhan, D., Goodfellow, I., Fergus, R.: Intriguing properties of neural networks (2013), arXiv:1312.6199
14. Touretzky, D.S., Pomerleau, D.A.: What's hidden in the hidden layers. Byte (1989)
15. Tsymbal, A.: The problem of concept drift: Definitions and related work. Tech. Rep. 106, Computer Science Department, Trinity College Dublin (2004)
16. Zeiler, M.D., Fergus, R.: Visualizing and understanding convolutional networks. In: Computer Vision–ECCV 2014, pp. 818–833. Springer (2014)