Finding Regions of Uncertainty in Learned Models: An Application to Face Detection

Shumeet Baluja[1,2]

[1] Justsystem Pittsburgh Research Center
4616 Henry Street
Pittsburgh, Pa. 15213
[2] School of Computer Science
Carnegie Mellon University
Pittsburgh, Pa. 15213

Abstract. After training statistical models to classify sets of data into predetermined classes, it is often difficult to interpret what the models have learned. This paper presents a novel approach for finding examples which lie on the decision boundaries of statistical models trained for classification. These examples provide insight into what the model has learned. Additionally, they can provide candidates for use as additional training data for improving the performance of the statistical models. By labeling the examples which lie on the decision boundaries, we provide information to the model in the regions in which it is most uncertain. The approaches presented in this paper are demonstrated on the real-world vision-based task of detecting faces in cluttered scenes.

1 Introduction

After training statistical models, such as artificial neural networks or decision trees, to classify a set of data into predetermined classes, it is often difficult to interpret what the models have learned. However, knowing how a model makes its decisions and which features it considers salient not only provides insights into what the model encodes, but also provides guidelines on how to improve the model's performance. In this paper, we present a novel technique to efficiently find examples which lie on the decision boundaries of learned models. We also show how these examples can be used for continuing the training of the models.

To demonstrate the techniques, we explore the task of face detection in cluttered scenes. The system used in this study is based upon the neural network-based system described in [7]. This task is chosen because (1) after the face detection networks are trained, we would like to understand what features they encode; (2) training the networks is a very slow process because finding good training examples is difficult. Both of these problems can be addressed with the methods described in this paper.

The next section describes the face detection system. Section 3 describes the techniques for finding examples on the decision boundaries. Section 4 shows how these examples can be used for continuing the model's training. Finally, Section 5 closes this paper with conclusions and directions for future research.

2 A Face Detection System

The system employed in this study is based on the neural network-based system presented in [7]. A neural network is used as a filter that receives as input a 20x20 pixel region of the image, and generates an output ranging from 1 to -1, signifying the presence or absence of a face, respectively. To detect faces anywhere in an image, the filter is applied at every location in the image. To detect faces larger than the window size, the input image is repeatedly reduced in size (by subsampling), and the filter is applied at each size. For the work presented here, we apply the filter at every pixel position in the image, and scale the image down by a factor of 1.2 for each step in the pyramid; this is the approach used in [7]

The filtering algorithm is shown in Fig. 1. First, a preprocessing step, of histogram equalization and lighting correction (see [7, 8]) is applied to each window of the image. These steps help account for lighting variation and expand the range of intensities in the window. This compensates for differences in camera input gains, in addition to improving contrast in some cases.

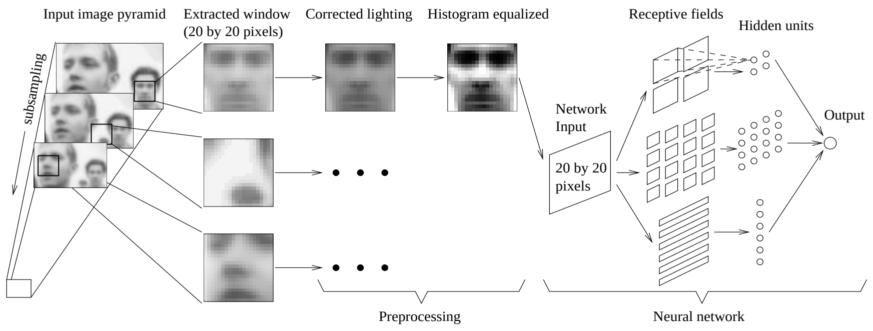

Fig. 1. The Face Detection System, based on [7]

The preprocessed window is then passed through a neural network (NN). The architecture of the NN is shown in the right half of Fig. 1. There are three types of hidden units: 4 which look at 10x10 pixel subregions, 16 which look at 5x5 pixel subregions, and 6 which look at overlapping 5x20 pixel horizontal stripes of pixels. More details about the architecture and motivations for its use can be found in [7]. The network has a single, real-valued output, which indicates whether or not the window contains a face.

To train the neural network to serve as an accurate filter, a large number of face and non-face images are needed. Nearly 1050 face examples were gathered. The images contained faces of various sizes, orientations, positions, and intensities. The eyes, tip of nose, and corners and center of the mouth of each face were labelled manually. These points were used to normalize each face to the same scale, orientation, and position [7]. Fifteen face examples are generated for the training set from each original image, by randomly rotating the images (about their center points) up to 10°, scaling between 90% and 110%, translating up to half a pixel, and mirroring. Each 20x20 window in the set is then preprocessed (by applying histogram equalization and lighting correction). A few example images are shown in Fig. 2.

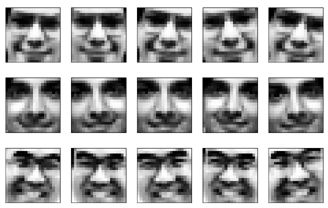

Fig. 2. Example face images, randomly mirrored, rotated, translated, and scaled by small amounts.

Practically any image can serve as a non-face example because the space of non-face images is much larger than the space of face images. However, collecting a "representative" set of non-faces is difficult. Instead of collecting the images before training is started, the images are collected during training in the following manner, adapted from [8, 5]:

1. Create an initial set of non-face images by generating 1000 random images. Apply the preprocessing steps to each of these images.
2. Train a neural network to produce an output of 1 for the face examples, and -1 for the non-face examples. The training algorithm is standard error backpropagation with momentum [6]. On the first iteration of this loop, the network's weights are initially random. After the first iteration, we use the weights computed by training in the previous iteration as the starting point.
3. Run the system on an image of scenery *which contains no faces.* Collect subimages in which the network incorrectly identifies a face (an output activation ≥ 0).
4. Select up to 1000 of these subimages at random, apply the preprocessing steps, and add them into the training set as negative examples. Go to step 2.

Some examples of non-faces that are collected during training are shown in Fig. 3. Note that some of the examples resemble faces. The presence of these examples forces the neural network to learn a sharp boundary between face and non-face images. For the networks trained in this study, we used a set of 285 images of scenery for collecting negative examples in the bootstrap manner described above. A typical training run selects approximately 10,000 non-face images from the approximately 200,000,000 subimages that are available at all locations and scales in the training scenery images.

The results for the networks used in this study are comparable to those used in [7]. The networks were tested on a large set of images [3]. Two networks were trained, starting with random initial weights. The performance of these networks is given in Table 1.

[3] The test images are available from `http://www.cs.cmu.edu/~har/faces.html`

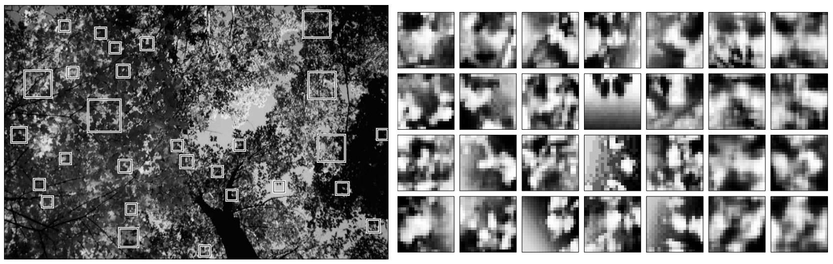

Fig. 3. During training, the partially-trained system is applied to images of scenery which do not contain faces (left). Any windows in the image detected as faces (expanded on the right) are errors, which can be added into the set of negative training examples.

Table 1. Detection and error rates for a test set which consists of 130 images and contains 507 frontal faces. It requires the system to examine a total of 83,099,211 20x20 pixel windows. Two independent networks are trained.

	Number Faces Missed	Detection Rate	False Positives	False Positive Rate
Network 1	45/507	91.1%	829	1/100240
Network 2	51/507	89.9%	693	1/136294

3 Finding Uncertain Examples

In the previous section, we described the training algorithm for the face detection networks. To find negative training examples (examples which represent "non-faces"), the network is scanned over scenery images which are known to contain no faces. Any location where the network detects a face is labeled as a negative example, and added to the training set. Although this is an effective procedure for finding negative examples, it is extremely slow. As the network is trained, the number of examples that must be seen before a single negative example is found rapidly increases. As can be seen in Fig. 4, after the network is trained from the examples from only a few of the initial scenery images, *often more than* 10^5 *examples must be seen before a single false-positive is found.*

The same method that is used to find false-positives can be used to find examples for which the network is most uncertain. Uncertainty is defined in terms of the network's output. The network's output is a real-value between -1 (indicating non-face) and $+1$ (indicating face); therefore, an input which gives an output near 0.0 indicates the network is uncertain of the classification. Finding input examples for which the network is uncertain is useful for two reasons. First, these examples provide samples on the decision boundaries; they reveal where the network does not perform well. Second, once labeled, these examples can be used for continuing training since it is known that the network is uncertain about their classification. This is a form of *active learning*, in which the learner requests information about uncertain points [1, 4].

There are at least three methods by which the uncertain examples can be found. First, similarly to the training procedure, the network can be serially

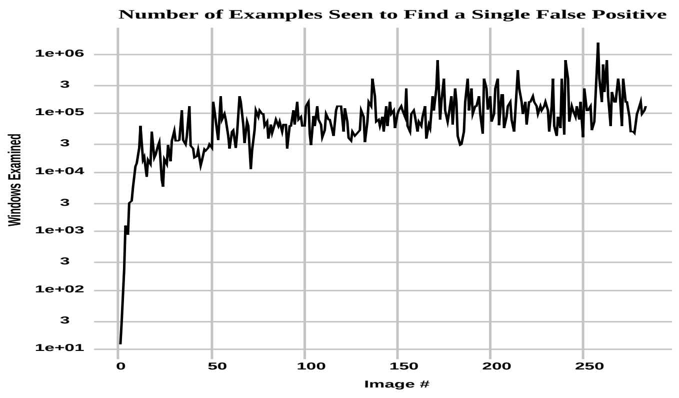

Fig. 4. The number of examples which must be examined before a single false positive (negative example) is added into the training set. X Axis: Scenery image presentation. Y Axis: Number of 20x20 windows examined.

scanned over scenery images until an input window is found for which the network outputs a value of 0.0. As described above, this method is extremely slow.

A second procedure is to use a backpropagation gradient descent procedure similar to that used for training neural networks; a cursory overview is given here. Given a 20x20 image of random pixel values as input, a forward pass of the input to the output is done (as in the standard backpropagation algorithm). If this output does not match the target (of 0.0), an error is passed back to the hidden units and then to the input layer. However, instead of using this error to update the network's weights, as would be done with the standard backpropagation algorithm, *the values of the inputs* are updated. Given a network in which the weights are frozen, this will move the inputs towards values that will reveal a network output of 0.0. This method has successfully been used to find inputs which the network maps to output of 0.0. However, the drawback to this method is that the resulting inputs can lie in a subspace that will never be encountered by the face detector network when used on real images. The reason for this is that the face detector system uses pre-processing steps, such as histogram equalization and lighting correction, which place all inputs in a limited subspace of possible inputs. Therefore, using this method to obtain inputs which the network maps to a value of 0.0 may not give a meaningful representation of the decision boundaries since the resulting inputs may violate invariants which are created by the pre-processing methods. In order to get meaningful samples, the pre-processing step must be accounted for when new inputs are found. It is difficult to account for the constraints and the non-linearities of the preprocessing steps in the backpropagation procedure.

The third approach is to use stochastic search techniques to find 20x20 images that the network maps to an output value of 0.0. However, two issues must still be resolved: the first is how to stay within the subspace of valid images, and the second is how to evaluate each image. To evaluate a new image, termed X, it is first projected into the space of valid images. This is done by applying all of the pre-processing steps used in training the face detection networks, yielding an image X'. X' is used as input into the trained network, and a forward propagation step is done, yielding the network's output, $NN(X')$. The difference between $NN(X')$ and the target output (0.0) is calculated. The larger the difference, the lower the evaluation of the image. Once an example is found for which the network outputs a value of 0.0, X' is returned. Given this evaluation procedure, which ensures that the evaluation is based on the mapping of the image to a valid subspace, we now describe the search procedures used.

3.1 An Optimization Approach to Finding Uncertain Examples

The search for an input image for which the trained network outputs a value of 0.0 is formulated as a combinatorial search problem. Three stochastic search procedures were examined: random mutation stochastic hill-climbing, population-based incremental learning (PBIL) [2], and genetic algorithms. A brief description of each is given below.

Rather than initializing the search algorithms randomly, all of the search techniques examined improved performance by being initialized with false-positives that were found through the neural-network training procedure (such as those found in Figure 3). For random mutation hill-climbing, this initialization meant that the starting point was an example selected from the set of negative examples in the training set. The hill-climbing procedure then stochastically perturbed pixels in the example (magnitude changes of up to 70% were allowed). The solution was evaluated as described previously (by performing the appropriate pre-processing, and then using it as input to the trained neural network and measuring the output). Moves to better or equal states were accepted. After 1000 consecutive non-accepted moves, the hill-climbing was restarted with another randomly selected image from the set of negative examples.

A complete description of the PBIL algorithm is beyond the scope of this paper; however, some of the basics can be found in [2]. In a manner similar to the hill-climbing procedure, a single image was used to initialize the search. A binary *modification vector* was created. The vector specified an amount to scale each pixel in the image. Each variable in this vector was represented with 5 bits. After applying the modification vector to the image, the image was pre-processed and used as input to the network. Compared to previous implementations of PBIL, the learning rate was set higher than normal (0.3), with a population size of 100. Additionally, whenever a new modification vector was created that performed better than any seen earlier, it was immediately used to update PBIL's probability vector.

Two modifications are made to the standard genetic algorithm to customize it for this domain. The first is the selection of the initial population. In many standard genetic algorithms, the initial population is selected randomly from the uniform distribution. However, using this initialization, finding even a single example which the network mapped to a value of 0.0 proved to be extremely slow. This is not surprising since the trained network already discriminates between faces and non-faces quite well (as described in Section 2). Therefore, there may be relatively few images, with respect to the possible images in the manifold spanned by images which have been appropriately pre-processed, which cause a trained network to output a positive response. To alleviate this problem, the population was initialized with examples chosen from the set of 20x20 windows which were used as negative examples (these were the examples gathered as false positives during the training procedure described in Section 2). Since these images were in the training set, it is unlikely that these images will themselves cause the network to output a value of 0.0; however, the hope is that the GA will find combinations of these images that will.

The second modification to the standard GA is in the crossover operator. One-point and two-point recombination operators are designed to exploit locality in typical one-dimensional parameter encodings. We use an analogous operator for two-dimensional images. Two points, (X1,Y1) and (X2,Y2), are randomly chosen. The rectangle defined by these two points is the area that is swapped between the parents. See Fig. 3.1 for an example. [3] provides a discussion of two-dimensional crossover operators.

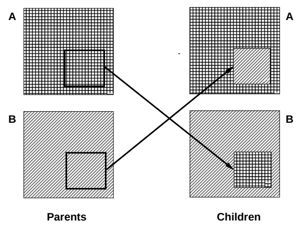

Fig. 5. 2D-Crossover.

With these two modifications, a standard steady-state genetic algorithm [9] was used with fitness proportional selection. It was empirically found that no mutation operator was required. A population size of 50 was used. By using this initialization procedure and this crossover operator, the number of evaluations required to find an image which the network mapped to an output of 0.0 was approximately 600, on average.

In the experiments performed, the genetic algorithm found the target in approximately 7-10% fewer evaluations than hill-climbing. PBIL, with the high learning rate, was able to find solutions approximately 20-30% faster than the GAs. However, due to the computational expense, only a few experiments were tried. Further experimentation should be conducted to fully understand the benefits of one method over another.

Some of the results of the search procedure are shown in Fig. 6. Note that the 60 images shown were found in a total of approximately 36,000 evaluations (forward passes through the network). In contrast, recall that when using the standard training procedure described in Section 2, by the end of training, it often took between 100,000 to 1,000,000 evaluations to find *a single* false-positive (see Fig. 4) by serially scanning over scenery images.

As can be seen from the images in Fig. 6, many of the images have dark patches where eyes would be located. The cheek, mouth, and nose areas are more variable. This indicates that the network is more sensitive to the presence of eyes than to the presence of the other features. This has been empirically

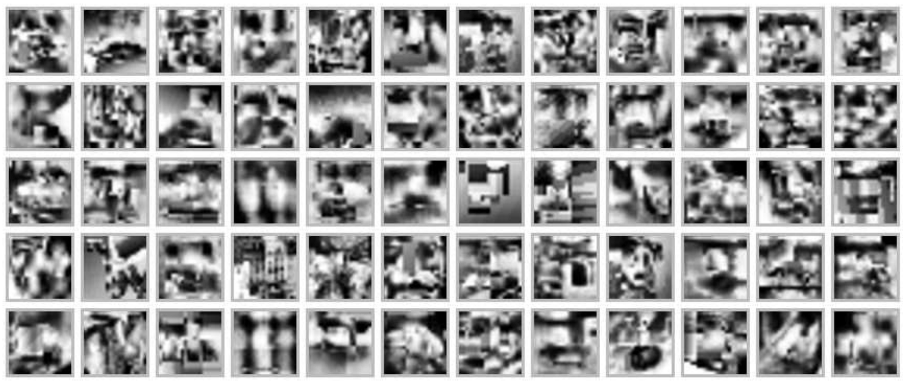

Fig. 6. 60 20x20 typical images obtained by the optimization procedures. A trained face detection neural network outputs a value close to 0.0 for each of the images, indicating that it is uncertain of their classification (face or non-face).

verified in other studies [7].

4 Using Uncertain Examples for Training

Despite the "eye-features" in some of the images in Fig. 6, when compared to the images of actual faces (see Fig. 2), it is clear that many of these images can be labeled as negative examples and then added into the training set for continued training. However, a few of the images look sufficiently close to faces that using them as examples of non-faces may confuse the network. Rather than using all of the found examples for training (as negative examples), we can select only those which are *most unlike* faces. To determine similarity to face examples, we measure the distance of each image to each of the known positive training examples (examples of which are shown in Fig. 2). Note that similarity between two examples is measured simply by the summed absolute difference between corresponding pixels. Alternatively, instead of finding images which are "far" from labeled face examples, we can find images which are "close" to labeled *non-face* examples. We can measure the distance of each image to each of the negative training examples (the type shown in Fig. 3). This last measure also has the benefit of selecting images which are similar to those that will be seen in real images, since the negative examples were selected from real images.

Both of these orderings are valid for the task. The ordering used is a combination of both. The first ordering ranks how similar each image is to the face examples; *the least similar is given the smallest rank and the most similar is given the largest rank.* The second ordering, ranks how similar the images are to the non-face examples; *here, the most similar is given the smallest rank and the least similar the largest rank.* The images with the smallest *summed* rank are used for training, see Fig. 7.

Table 2 shows the results of continuing the training of the network with the extra examples. We compare the performance to the original network by adjusting the detection threshold to give the same detection rates, and then

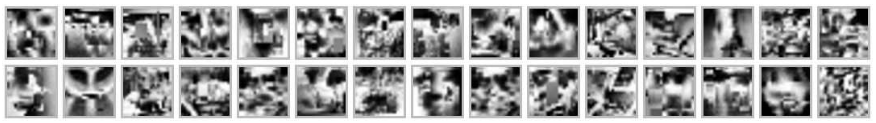

Fig. 7. Combined ordering of 30 20x20 images, each image was classified as a face by the network. Top-Left: smallest combined rank.

examining the number of false positives. The last column in the table shows the improvement (in terms of the reduction in the number of false positives) over the original network. For each network, 10,000 new examples were generated as described in this paper. The top $N\%$ were used for training after sorting them according to the combined metric described in the previous paragraph. The table shows the results of varying N. The same test set described in Table 1 is used.

Table 2. Results after training with found examples. Note the improvements in performance with respect to the original networks described in Table 1. Unfortunately, performance improvements do not always scale with the amount of extra training data.

	N	Faces Missed	Detect Rate	False Pos.	False Pos. Rate	% Improvement
Network 1	0.5%	45/507	91.1%	789	1/105322	4.8%
Network 1	2%	45/507	91.1%	769	1/108061	7.2%
Network 1	10%	45/507	91.1%	717	1/115898	13.5%
Network 2	0.5%	51/507	89.9%	643	1/146893	7.2%
Network 2	2%	51/507	89.9%	684	1/138088	1.2%
Network 2	10%	51/507	89.9%	640	1/147581	7.6%

5 Conclusions and Future Research

This paper has demonstrated an efficient and effective method for finding examples which lie on the decision boundaries of learned classification models. Although demonstrated with neural networks, the techniques are general, and can be used with any type of statistical model. The found examples represent the areas in which the model is uncertain; therefore, with labels, they may be used to provide useful information to the learner. In the domain of face detection, we were able to improve the performance of state-of-the-art models using the examples to continue training.

In order for this method to be successfully applied to a new domain, there must either be a procedure to search only in the space of meaningful examples, or to map all inputs into this range. For example, if it is known that all of the expected input examples span only a small range of all possible inputs, finding uncertain examples outside of this range is not useful, since they may never be encountered in practice. In the face detection domain, this problem was alleviated by beginning the search for uncertain examples from examples which had been encountered by the system, and ensuring that all of the generated inputs matched the constraints of inputs which would be seen in practice. For example, all of the images found by the search heuristics are deemed uncertain by the network *after* being pre-processed with histogram-equalization and lighting correction. Therefore, these examples meet the constraints which would be maintained in all of the false-positive images found by the system through serially scanning over the non-face training images. Further, by using the sorting metrics to select uncertain images that are most like the false positives found from the serial scanning procedure, we help to ensure that these images provide a training signal that helps the classification performance of the network when tested on real images.

There are at least two immediate directions for future research. The first is to combine the ranking procedures described in Section 4 with the evaluation function used for search. By modifying the evaluation function to include extra terms, it should be possible to search for examples which the network not only maps to a value close to 0.0, but are also dissimilar to known faces, and similar to known non-faces. One method for implementing this may be to penalize solutions which are distant from the negative examples in the training set; this will ensure that the examples found are close to those that are encountered in practice. Second, it will be interesting to determine the extent that it is possible to repeat the entire training process. For example, after training on examples for which the network is uncertain, it should be possible to find the examples which the newly trained model is uncertain, and repeat the training process.

References

1. L. Atlas, D. Cohn, and R. Ladner. Training connectionist networks with queries and selective sampling. In *Advances in Neural Information Processing Systems 2*. Morgan Kaufmann, 1989.
2. Shumeet Baluja. Genetic algorithms and explicit search statistics. In M. Mozer, M. Jordan, and T. Petsche, editors, *Advances in Neural Information Processing Systems (NIPS) 9*. MIT Press, Cambridge, MA, 1998.
3. H. M. Cartwright and S. P. Harris. The application of the genetic algorithm to two-dimensional strings: The source apportionment problem. In *Fifth International Conference on Genetic Algorithms*. Morgan Kaufmann, 1989.
4. D. Cohn, Z. Ghahramani, and M.I. Jordan. Active learning with statistical models. In *Advances in Neural Information Processing Systems 7*. MIT Press, 1995.
5. Harris Drucker, Robert Schapire, and Patrice Simard. Boosting performance in neural networks. *International Journal of Pattern Recognition and Artificial Intelligence*, 7(4):705–719, 1993.
6. John Hertz, Anders Krogh, and Richard G. Palmer. *Introduction to the Theory of Neural Computation*. Addison-Wesley, Reading, MA., 1991.
7. Henry A. Rowley, Shumeet Baluja, and Takeo Kanade. Neural network-based face detection. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 20:23–38, January 1998.
8. Kah-Kay Sung. *Learning and Example Selection for Object and Pattern Detection*. PhD thesis, MIT AI Lab, January 1996. Available as AI Technical Report 1572.
9. D. Whitley and T. Starkweather. Genitor ii: A distributed genetic algorithm. *JETAI*, 2, 1990.

This article was processed using the LaTeX macro package with LLNCS style