Out-Of-Sample Recognition: The Ignorance of Machines

Christopher M. Bender
Department of Computer Science
The University of North Carolina
Chapel Hill, NC 27599-2200
bender@cs.unc.edu

Samuel D. George
Department of Computer Science
Department of Pharmacology
The University of North Carolina
Chapel Hill, NC 27599-2200
sdgeorge@cs.unc.edu

Abstract

All machine learning algorithms are trained on a set of representative data and then deployed on a new set of unseen data. The strength of the model is assessed on its ability to respond to unseen data. Implicit, is the assumption that the unseen data belongs to the same parent set as the training data (i.e. in the context of classification, that the data belongs to one of the expected classes). When this assumption is violated, the algorithm is used in an unintended manner and will produce unpredictable results. This work explores several methods of estimating when an example belongs to the same distribution as the training set (and the model can be trusted) and when it belongs to a different distribution. In particular, we explore the use of likelihood estimators(1) to assess the similarity of an example to the training set and construct a one-class neural network(2) by borrowing ideas from the linear one-class SVM(3).

1 Introduction

When a system is deployed into the wild it can encounter a variety of inputs that were out of scope of the original design or the available training examples. It is an unfortunate reality that designers cannot practically account for all possible situations and that sufficient training data cannot be sampled or labelled. This design and input limitation is exacerbated by the extreme (and unjustified) over-confidence(4) neural networks exhibit when faced with examples that violate the implicit assumptions in the design and training procedures. A viable solution to this problem presents several opportunities for improving the field of machine learning. For example, models that have been pre-trained on a set of data could be used in another environment with a much higher level of confidence. Often given a new data set, only an architecture should be reused because behavior using previous training can be erratic. If a consistent confidence metric existed for judging new data's relation to previous data, models could be reused with greatly reduced training on slightly different new data. A good confidence metric could also be used to develop intelligent training methods where a set of data could be compiled that effectively covered the space of a desired class based on confidence. This could greatly reduce the required number of samples in a given class required to accurately cover the desired sample space. Overall, a solution to this problem could have a vast impact on the field of machine learning.

2 Related Work

A variety of methods to address this problem have been proposed with varying levels of success. A comprehensive review of the literature shows that most methods fail to reach better than a 80% recognition of images that contain unseen classes(5). The current state-of-the-art is a method called Out-Of-Distribution Detector for Neural Networks(4) (ODIN) and approaches the 80% recognition level in some cases. This method learns a simple rescaling and thresholding of the softmax output prediction in a network to decide if an example is out-of-distribution. However, ODIN loses accuracy as the number of classes grows, achieving an accuracy closer to 70% on the CIFAR-100 dataset. Other methods have attempted thresholding estimates of the log-likelihood from each class, similar to our ultimate approach. In one particular case, PixelCNN++(6) was used to estimate the likelihood, but, when tested in the same manner as above, it failed to perform better than random guessing(5; 6). Many other methods exist, including direct binary classification of in vs out of set(5), the introduction of a null class(7), the use reconstruction error from pre-trained variational autoencoder(8) (VAE) to derive a metric for confidence, and the application of a Generative Adversarial Network's (GAN) discriminator(9). The VAE-based method approaches the accuracy of ODIN in terms of performance on large datasets, but its accuracy still only approaches 70% as the number of classes grows(5). GAN based methods struggle from the typical difficulties associated with GAN training and the requirement that a separate GAN be trained for each known class.

3 Datasets

For the sake of simplicity and familiarity, we limit our analysis to labelled image datasets. This will allow us to differentiate between in-distribution and out-of-distribution examples by only training on a subset of the total classes. To this end, we use the CIFAR-10(10) and COCO(11) datasets. Sections 3.1 and 3.2 discuss the particulars of each dataset and they were segregated to create in-distribution and out-of-distribution sets.

3.1 CIFAR-10

CIFAR-10 is a collection of 32×32 RGB images over ten distinct classes. In our initial experiments, we extracted the "dog" and "cat" classes as in-distribution and used the remaining classes to assess out-of-distribution, creating a 20%-to-80% split. Our results in this case failed unless we partially cheated by including all the classes as part of the code (but not in-distribution) training process. Eventually, we inverted the training split by retaining the first eight classes as in-distribution and the remaining two (ships and trucks) as out. The data was broken down into 4,000 training examples, 1,000 validation examples, and 1,000 testing examples (per class).

3.2 COCO

COCO is a data-set maintained by Microsoft that was specifically designed to be used for solving bounded box object recognition problems. All images are class labeled. We extracted the classes of "horse" and "airplane" to use for our training set for in-distribution and then used "book," "cake," "car," "cat," "dog," "laptop," "pizza," and "zebra" as our examples of out-distribution data. Images in the data set are of variable size and scale. All images were shrunk and padded to 331×331 RGB images. Within each set 75% of the data was used for training, while 25% of the data was used for testing and validation.

4 Methods and Results

All of our testing methods utilized an encoded representation of the inputs. This is not a necessary requirement for out-of-distribution testing; however, in the original space, image data is massively redundant. The high levels of redundancy can make it difficult for a network to key in on important global parameters due to the local operations inherent in convolutions and autoregressive conditioning. We use two methods to develop the image codes: an autoencoder and the [learned] features from a convolutional classifier.

4.1 Autoencoder

The first method we use to generate the input encodings is an autoencoder(12). The autoencoder is trained to minimize the L2 loss between the input image and the output reconstruction. The codespace is restricted to be either a 32 or 128 dimensional vector. In some of our tests an additional classification task is imposed from the reduced dimension code in the hope of making the code more class-dependent, see Sec. 4.4.1 for additional details.

Ideally, the autoencoder is trained using only the retained, in-distribution data; however, some experiments were conducted where the autoencoder was trained using all of the data. This partial cheat is somewhat justifiable in the case where we have large volumes of data but only some of the data has labels. This is one of the primary advantages of the autoencoder method: it operates unsupervised and can incorparate a variety of data that might otherwise have to be excluded.

Figure 1 demonstrates the autoencoder input images and reconstruction when the code dimension is set to 32. The data is preprocessed to be between zero and one. The autoencoder architecture is composed of several convolutional (and transpose convolutional) layers activated with a leaky relu. The layers on both sides of the code are composed of a fully-connected layer that is also activated with leaky relu. All of the convolutional layers are interspersed with batch-normalization layers. The final reconstruction layer is activated with a sigmoid layer. In all cases, the autoencoder is trained prior to any other networks that use the code for out-of-distribution estimation.

Figure 1: Autoencoder input images and reconstruction

4.2 Progressive Neural Architecture Search (PNAS) Features

The PNASNet-5_Large_331 network is one of the pre-trained ImageNet classifiers maintained by Google as an official part of TensorFlow(13). This model has been proven to have an accuracy on ImageNet of 96.2% Top-5 Accuracy and a 82.9% Top-1 Accuracy. It represents the state-of-the-art in image recognition and has a structure where the last layer post convolution is a 1×4028 array. (14)

We consider the last output layer from this network to be the raw image encoding. Our theory is that this layer contains an accurate representation of the patterns present in the original image in a far more condensed space. Assuming this code is a good representation of the higher level features in the image, it makes for an ideal starting place for our likelihood estimators. We will refer to this last layer as the PNAS Feature Layer.

4.3 One-Class Neural Networks

Linear one-class support vector machines(3) (SVMs) have been used extensively to assess if an example is out-of-distribution. However, this method suffers from the same primary limitation inherent in all SVMs — limited input dimensionality. Recently, the ideas used in one-class SVM has been extended to to neural networks(2) with seemingly impressive (nearly perfect!) performance. Sadly, we could not reproduce the results described and further investigation indicated some oddities in problem setup and execution. For the sake of thoroughness, we briefly discuss the formulation of the problem, the advantages of the one-class neural network over the one-class SVM, and present our results.

The original one-class SVM solves the optimization problem

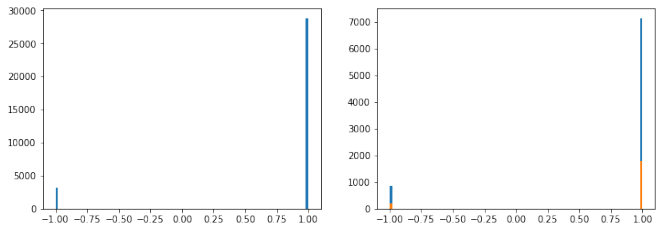
$$\min_{w,r} \frac{1}{2}||w||_2^2 + \frac{1}{\nu} \cdot \frac{1}{N} \sum_{n=1}^{N} \max\left(0, r - \langle w, \Phi(X_n) \rangle\right) - r \tag{1}$$

where w is a projection vector, r is an offset distance, ν is a hyper parameter between zero and one that controls the strength between the projection norm and the offset, N is the number of examples, Φ is the SVM kernel, and X_n is the nth input. The formulation is easily extended to the case of neural networks by modifying Eq. 1 such that kernel is replaced with a parameterized nonlinearity, i.e.

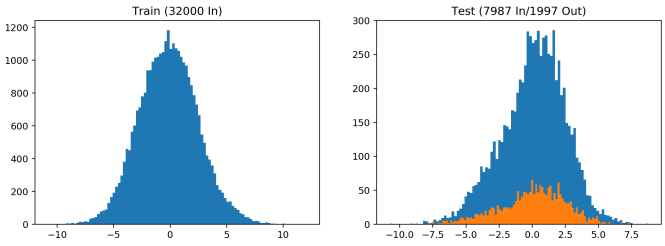
$$\min_{w,r,\theta} \frac{1}{2}||w||_2^2 + \frac{1}{\nu} \cdot \frac{1}{N} \sum_{n=1}^{N} \max\left(0, r - \langle w, f_\theta(X_n) \rangle\right) - r \tag{2}$$

where f is the nonlinear function parameterized by θ. Note, this function can be constructed of an arbitrary number of layers. In some cases, it may be necessary to include an additional regularization term on the parameters as in (2).

There are a two primary advantages of the one-class neural network over the one-class SVM. The neural network formulation allows the nonlinear transform to be learned through its parameters. This means that the input features can be adjusted to a greater degree than is possible with an SVM. Additionally, since the input features are themselves extracted from a learned code-space, the one-class neural network can adjust not just the activation on the input features/codes but could be made to change the compressed representation all the way to the original input image.



(a) One-Class SVM Class Decision



(b) One-Class Neural Network Signed-Distance Distribution

Figure 2: One-Class Decisions (Blue in-distribution, orange out)

Figure 2 illustrates the histogram of the in- and out-of-distribution examples through the one-class SVM (Fig. 2a) and the one-class neural network (Fig. 2b). In both cases, the left figure contains the in-distribution training data and the right figure contains the test data, with the out-of-distribution data illustrated in orange. Both methods failed. Figure 2a shows that the one-class SVM overfit to the in-distribution training data and declares approximately 80% of the data in-distribution all the time. Figure 2b demonstrates that the one-class neural network failed to learn any worthwhile features at all and does little better than a coin flip.

The results hold regardless of the hyperparameters chosen in the one-class neural network training proceedure. We attempted three different nonlinear activations (linear, sigmoid, and relu), three different values of ν (0.1, 0.5, and 0.9), two different code dimensions (32 and 128), attempted the optimization with and without weight optimization, and with and without adding a joint classifier to the original autoencoder training process. In all cases, the performance was bounded between 45 and 55 % accuracy. The total one-class network consisted of two fully-connected layers with an intermediate dimension equal to half the input dimension. Despite our failure to solve the problem, the method has merit and it seems premature to discount the idea.

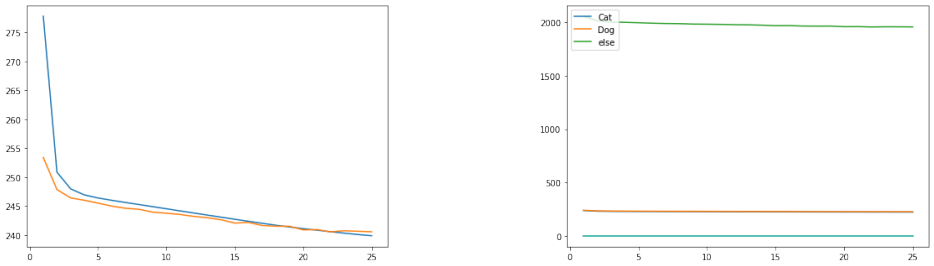
4.4 Likelihood Estimation

Likelihood estimators are trained to predict the underlying distribution for the training data and then either, assess the probability that a given input comes from the approximated distribution or to generate new, novel samples. The former process seems like an ideal tool for estimating if an example is out-of-distribution; if the example is dissimilar to the training set, it will have a low likelihood. Likelihood estimators are generally broken down into two categories, autoregressive and flow models. Other authors (see (5) for a comprehensive list) have attempted to use these models directly on the input images. Unfortunately, even the state-of-the-art Pixel CNN++ likelihood estimator fails to achieve sufficiently impressive results on the out-of-distribution problem. Instead, we attempt a likelihood estimation of an encoded representation of the input. Additionally, instead we use the extremely advanced transformational autoregressive networks(1) to estimate the likelihoods. This method combines the typical autoregressive and flow models to overcome the limitations inherent to the separate methods.

4.4.1 Autoencoder Likelihood

Our original efforts utilized a vanilla autoencoder trained on the in-distribution classes to create the 128 dimensional codespace. These experiments used CIFAR-10 and included cats and dogs as the in-distribution set and used the remaining classes as the out-of-distribution datasets. Once the autoencoder was trained, the [fixed] codes were used to train a TAN. This method proved to be too simple. We found that adding a shallow classifier based on the codes and training this jointly with the autoencoder reconstruction improved the separability of the codes and allowed us to successfully train the likelihood estimator. Additionally, we projected the data into an eigensubspace of dimension ten.

Figure 3a shows the negative log-likelihood returned by the TAN during training (blue) and testing (orange) averaged over all classes. In this case, we see that the negative log-likelihood monotonically decreases with epoch for both training and testing. However, it's difficult to draw conclusions from the average negative log- likelihood. Figure 3b splits the testing likelihood out by class. The figure indicates that cat and dog (the in-distribution set) likelihoods are easily separable from the out-of-distribution classes.



(a) Average negative log-likelihood

(b) Test negative log-likelihood per class

Figure 3: Negative log-likelihood estimate versus epoch

Unfortunately, we had to cheat a bit to get this result. In particular, the autoencoder and the joint classifier were trained using all of the data, including the out-of-distribution data. For this method to be valid, we cannot use all of the out-of-distribution training data. Some out-of-distribution data may be appropriate since it is not unreasonable to have a small sample of unknown unknowns, but it is impossible to have a thorough set of all possible unknowns. One of the main advantages of this technique is that it is completely unsupervised. Neither the autoencoder nor the TAN require any labels to function. Introducing the classifier to help separate the codes undoes this advantage. The next steps in this analysis method would be to start removing some of the unknowable information until we have arrived at a more realistic training method. The primary difficulty will be in constructing a code space that generalizes well to unseen in-distribution examples without generalizing to out-of-distribution examples. Alternatively, we could explore methods that do not rely on a lower-dimensional encoding and attempt to assess the distribution based on the full input space.

4.4.2 Conditional TAN Classifier

As a method to estimate likelihoods for in-class samples as a metric to identify out-of-sample data, we assembled what we call a Conditional TAN Classifier.

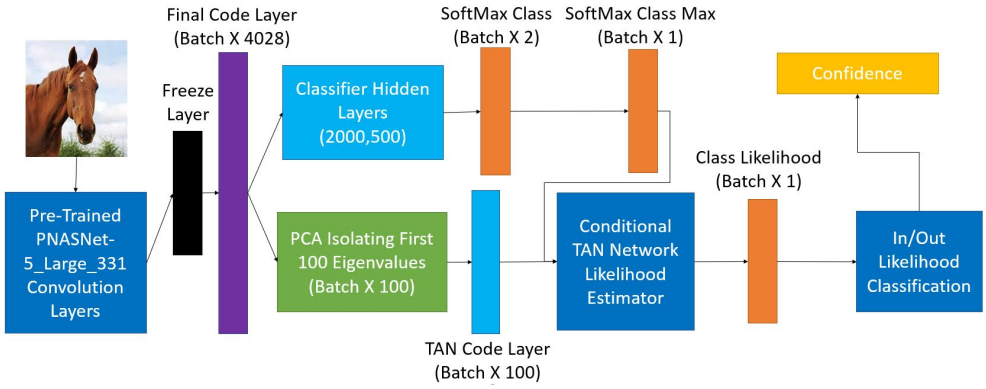


Figure 4: Conditional TAN Classifier Network Structure

Using the last layer post-convolution from the PNAS network, we first trained a binary classifier to recognize the two in-class samples of "horses" and "airplanes." We used a straightforward DNN approach for this with two dense layers (2000 & 500) followed by a binary SoftMax layer representing to which class the images belonged. Looking at several batches we see our classifier has a very high accuracy for distinguishing "horses" and "airplanes."

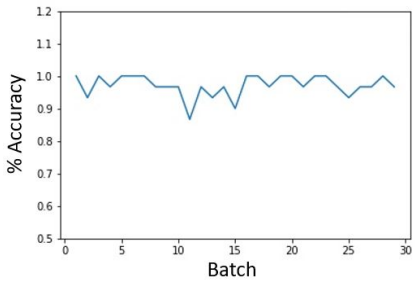


Figure 5: PNAS Classifier Accuracy

Next we implemented a PCA-Like data space reduction on the last layer post-convolution by subtracting the mean from each batch and then multiplying by a 4028×100 learned matrix. This produced a 1×100 code (TAN Code Layer). We then multiplied that code by the transpose of the learned matrix and added back the batch mean. This generated a 1×4028 reconstruction layer. This learned matrix was then trained on the L2-loss between the original 1×4028 encoding and the 1×4028 reconstruction layer. Looking at the original feature layer versus the reconstruction we see that our encoding is capturing large patterns in the data, but is failing to capture some more nuanced features.

After creating the TAN Code Layer to represent the features present in the PNAS Feature Layer, we fed these encodings into a Conditional TAN. During the training the TAN is conditioned on the correct label for the class. During testing, the TAN is conditioned on the class identified by the max SoftMax class. Looking at the performance of a validation set on the TAN we see that likelihood is maximized in as few as five epochs of the data.

Finally, we tested the architecture using a mix of the two in-class categories and eight out-class categories. Here, we see that the distribution of in-class likelihoods is distinct from the out-class likelihoods. Looking at the average likelihood for each class we see that most of the overlapping likelihood values in the distribution come from the high likelihoods in the out-class category for "zebra." This makes sense, given zebras are a type of horse.

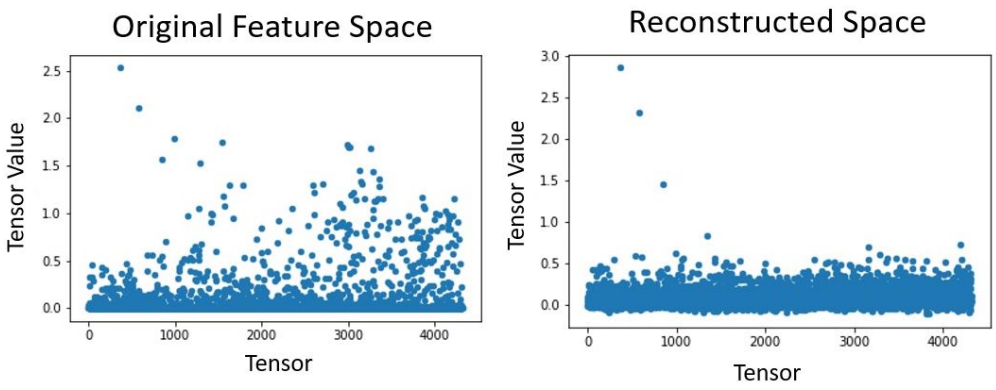
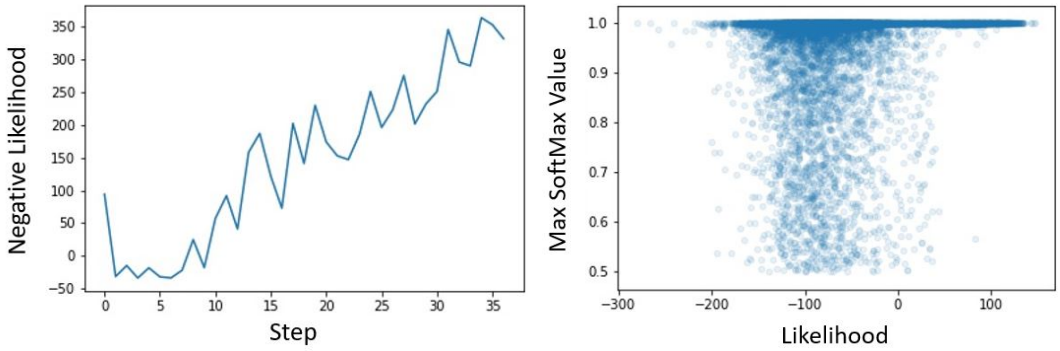


Figure 6: PCA-Like encoding of PNAS Code Layer



(a) TAN Training

(b) Likelihood vs SoftMax Correlation

Figure 7: Likelihood comparisons

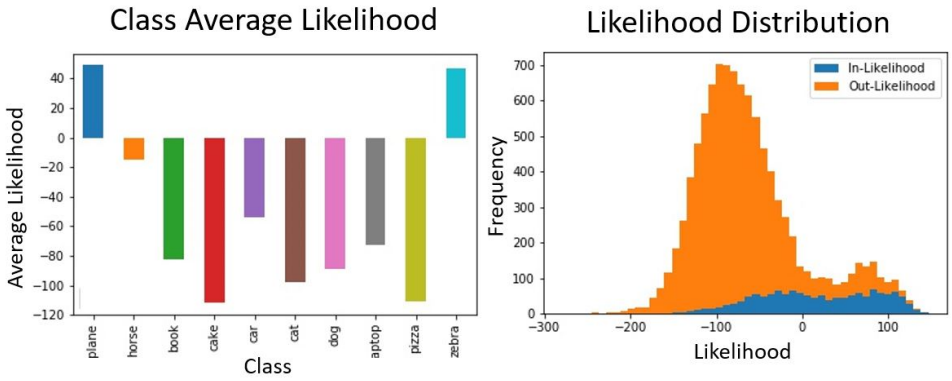


Figure 8: Conditional TAN Likelihood Distribution (Plane and Horse are in-class)

Looking at the correlation between the max SoftMax Value for a given class and its likelihood value we see that likelihood provides a much stronger indication of in-vs-out classification then any method the just used SoftMax as a threshold.

If we choose a metric for classifying samples as in-vs-out of class based on likelihood we are able to produce several very accurate models that predict in-vs-out of class with an accuracy higher than even the state-of-the-art methods in the field.

Overall, the Conditional TAN Classifier shows that likelihoods can be used to recognize in-class versus out-class data with the exception of classes that closely resemble the classes seen in training (“horses” vs “zebras”). While further testing of variable data sets is required to fully flesh out the

Table 1: Conditional TAN Out-vs-In Accuracy

Threshold Metric	Accuracy
Mean Lowest In-Sample Class	85.6%
Median Lowest In-Sample Class	85.2%
Quarter Percentile from Lowest In-Sample Class	74.7%

abilities of these likelihood compared to the state-of-the-art methods, in our test cases, this method proves to be very effective.

5 Discussion

While our methods, present an interesting avenue for solving the out-of-sample problem, several interesting points should be mentioned. First, the need for an accurate representation of patterns in images is still needed. Training an autoencoder outright, did not appear to provide a good representation of higher level features and this is likely the reason the likelihood estimator failed under these conditions. Second, our pre-trained model provided a much better feature encoding, but that encoding was the result of training on a large 1,000 image space. If we used this network without pre-training and it had only been taught to recognize the two distinct classes we trained on, it is unclear if the high level feature encoding would be meaningful once condensed and used to estimate likelihood. Finally, likelihood methods that are able to deal with much higher dimensional data may be a crucial necessity for solving the out-of-sample problem. Overall, if each of the levels of our architecture could be improved incrementally, our method may provide a great solution to out-of-sample recognition in the future.

6 Conclusion

Neural networks and machine learning are becoming common place in our society. So far, these systems have largely been deployed to improve quality of life or inform human decision makers but it is only a matter of time before these systems can make decisions that could suddenly change a person's life, e.g. self driving cars. Before we put that level of trust in our machines, we need to teach them to recognize their own ignorance. Significant strides have been made to this end(5) but more work is still required. We have demonstrated several possible avenues worth consideration in both supervised and unsupervised domains. Unfortunately, while our efforts indicate some potential, additional tests against other baseline datasets are required for a true comparison and several of the methods are incomplete.

Acknowledgments

We would like to thank Dr. Oliva for providing a simple interface for the Tensorflow(13) implementation of the TAN architecture. This considerably simplified our efforts and helped to focus on the problem at hand.

Special thanks to all our fans out there. We couldn't do it without you.

Code

https://github.com/SamuelDGeorge/out_of_sample

References

[1] J. Oliva, A. Dubey, M. Zaheer, B. Poczos, R. Salakhutdinov, E. Xing, and J. Schneider, "Transformation autoregressive networks," in *Proceedings of the 35th International Conference on Machine Learning* (J. Dy and A. Krause, eds.), vol. 80 of *Proceedings of Machine Learning Research*, (Stockholmsmässan, Stockholm Sweden), pp. 3898–3907, PMLR, 10–15 Jul 2018.

[2] R. Chalapathy, A. K. Menon, and S. Chawla, "Anomaly detection using one-class neural networks," *CoRR*, vol. abs/1802.06360, 2018.

[3] L. M. Manevitz, M. Yousef, N. Cristianini, J. Shawe-taylor, and B. Williamson, "One-class svms for document classification," *Journal of Machine Learning Research*, vol. 2, pp. 139–154, 2001.

[4] S. Liang, Y. Li, and R. Srikant, "Enhancing the reliability of out-of-distribution image detection in neural networks," *ICLR*, 2017.

[5] *Hidden for review*, "Does your model know the digit 6 is not a cat? a less biased evaluation of "outlier" detectors," 2018.

[6] T. Salimans, A. Karpathy, X. Chen, and D. P. Kingma, "Pixelcnn++: A pixelcnn implementation with discretized logistic mixture likelihood and other modifications," in *ICLR*, 2017.

[7] A. Bendale and T. E. Boult, "Towards open set deep networks," in *2016 IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2016, Las Vegas, NV, USA, June 27-30, 2016*, pp. 1563–1572, 2016.

[8] D. P. Kingma and M. Welling, "Auto-encoding variational bayes," *International Conference on Learning Representations*, vol. abs/1312.6114, 2013.

[9] T. Schlegl, P. Seeböck, S. M. Waldstein, U. Schmidt-Erfurth, and G. Langs, "Unsupervised anomaly detection with generative adversarial networks to guide marker discovery," in *Information Processing in Medical Imaging - 25th International Conference, IPMI 2017, Boone, NC, USA, June 25-30, 2017, Proceedings*, pp. 146–157, 2017.

[10] A. Krizhevsky, V. Nair, and G. Hinton, "Cifar-10 (canadian institute for advanced research),"

[11] T.-Y. Lin, M. Maire, S. Belongie, J. Hays, P. Perona, D. Ramanan, P. Dollár, and C. L. Zitnick, "Microsoft coco: Common objects in context," in *Computer Vision – ECCV 2014* (D. Fleet, T. Pajdla, B. Schiele, and T. Tuytelaars, eds.), (Cham), pp. 740–755, Springer International Publishing, 2014.

[12] G. E. Hinton and R. R. Salakhutdinov, "Reducing the dimensionality of data with neural networks," *Science*, vol. 313, no. 5786, pp. 504–507, 2006.

[13] M. Abadi, A. Agarwal, P. Barham, E. Brevdo, Z. Chen, C. Citro, G. S. Corrado, A. Davis, J. Dean, M. Devin, S. Ghemawat, I. Goodfellow, A. Harp, G. Irving, M. Isard, Y. Jia, R. Jozefowicz, L. Kaiser, M. Kudlur, J. Levenberg, D. Mané, R. Monga, S. Moore, D. Murray, C. Olah, M. Schuster, J. Shlens, B. Steiner, I. Sutskever, K. Talwar, P. Tucker, V. Vanhoucke, V. Vasudevan, F. Viégas, O. Vinyals, P. Warden, M. Wattenberg, M. Wicke, Y. Yu, and X. Zheng, "TensorFlow: Large-scale machine learning on heterogeneous systems," 2015. Software available from tensorflow.org.

[14] C. Liu, B. Zoph, J. Shlens, W. Hua, L. Li, L. Fei-Fei, A. L. Yuille, J. Huang, and K. Murphy, "Progressive neural architecture search," *CoRR*, vol. abs/1712.00559, 2017.