Improving Compositionality of Neural Networks by Decoding Representations to Inputs

Anonymous Author(s)
Affiliation
Address
email

Abstract

In traditional software programs, we take for granted how easy it is to debug code by tracing program logic from variables back to input, apply unit tests and assertion statements to block erroneous behavior, and compose programs together. But as the programs we write grow more complex, it becomes hard to apply traditional software to applications like computer vision or natural language. Although deep learning programs have demonstrated strong performance on these applications, they sacrifice many of the functionalities of traditional software programs. In this paper, we work towards bridging the benefits of traditional and deep learning programs by jointly training a generative model to constrain neural network activations to "decode" back to inputs. Doing so enables practitioners to probe and track information encoded in activation(s), apply assertion-like constraints on what information is encoded in an activation, and compose separate neural networks together in a plug-and-play fashion. In our experiments, we demonstrate applications of decodable representations to out-of-distribution detection, adversarial examples, calibration, and fairness — while matching standard neural networks in accuracy.

1 Introduction

Traditional hand-written computer programs are comprised of a computational graph of typed variables with associated semantic meaning. This structure enables practitioners to interact with programs in powerful ways (even if they are not the author) — such as debug code by tracing variables back to inputs, apply unit tests and assertions to block against erroneous behavior, and compose separate programs together for more complex functionality. This compositionality and flexibility has enabled software to permeate every industry and application. Even still, traditional software has its limitations: it is difficult to hand-write programs to classify images or extract sentiment from natural language. For these functionalities, deep learning and neural networks [53] have become the dominant approach [29, 2, 60].

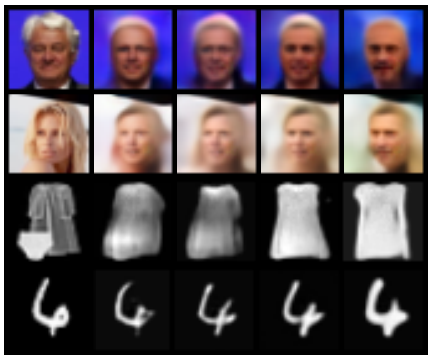


Figure 1: Decoding activations to images for misclassified examples.

This distinction has manifested in a budding shift from the classical stack of Python, C++, etc. to "software 2.0" [26, 63, 48], which is built on neural network weights. While neural networks have made impressive progress on complex tasks, switching to software 2.0 comes at a sacrifice of many of the desirable properties of "software 1.0" programs. The closest approximation to a variable in a neural network is an activation. It is difficult to understand a neural network's computation from an activation value, and there is little to associate an activation

Submitted to 35th Conference on Neural Information Processing Systems (NeurIPS 2021). Do not distribute.

with semantic meaning. A practitioner cannot impose unit tests on neural networks, let alone write assertion statements to constrain valid logic: checking the values that an activation takes is usually not enough to gauge correctness nor meaning. Moreover, given multiple neural networks (e.g. one for classification and one for segmentation), composing them together requires retraining from scratch. Consequently, neural networks deployed in practice are brittle and require maintenance [54].

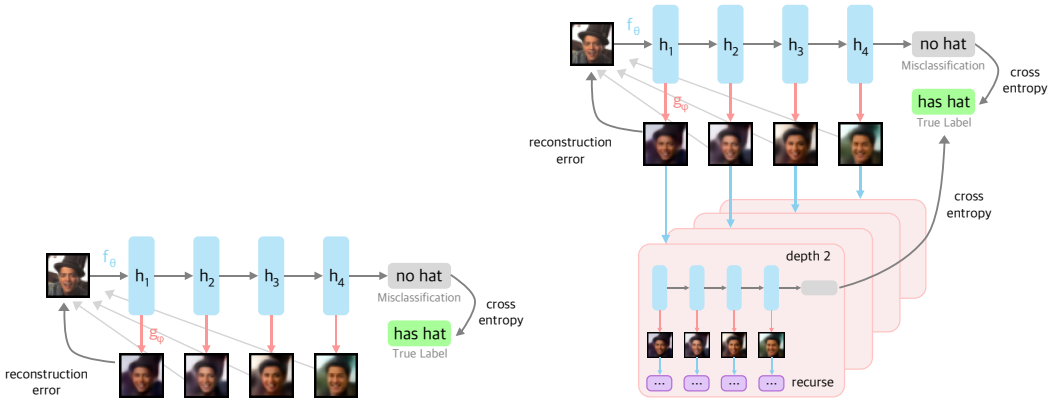
In this paper, we work towards a "software 1.5", where the overarching goal is to bridge the expressivity of deep learning with the engineering practicality of traditional software. Our desiderata is to construct neural networks whose activations can be decoded to semantic meanings — taking a step towards enabling deep learning practitioners to probe the model by tracing activations back to inputs, apply assertion-like constraints on what information is stored in an activation, and also compose neural network modules together in new and useful ways.

To start, we propose to train a neural network classifier jointly with a generative model whose role is to map the classifier's activations back to the input, approximating invertibility. This is only an approximation as activations deeper into the network drop information, meaning decoded inputs — while semantically related to the original input — are themselves new. This joint training optimizes the classifier not only to be highly performant, but constrains its activations to a manifold that the generative model is able to decode. Consequently, the neural network's entire computation is constrained to a "decodable" vector space as all activations can be mapped back to the input space. Continuing the software analogy, we are enforcing our neural network activations to have the same "type" as training examples. It is this insight that enables us to treat our neural networks more like traditional software, where practitioners can analyze and repurpose decoded activations.

In our experiments, we make progress towards each of the aforementioned features of software 1.0 that we could not do with software 2.0: probing our "variables" (i.e. activations) for meaning, constraining model logic with "assertions", and composing neural network modules together. In doing so, we will explore applications of decodable neural networks to out-of-distribution detection, adversarial examples, calibration, and fairness — a representative collection of engineering features necessary for real world applications of deep learning. Throughout this, we show that decodability comes at low cost as we find equivalent accuracy as a standard neural network.

2 Background

Neural Networks We will focus on supervised neural networks for classification, although it is simple to extend to regression tasks. We denote a neural network by f_θ where θ represents its trainable parameters. A neural network f_θ maps an example x to predicted probabilities over K classes. Typically, a neural network is composed of L blocks. For example, if f_θ is a multi-layer perception, each block is a linear layer followed by an non-linearity. If f_θ is a ResNet [21], each layer is a residual block. In computing its prediction, a neural network f_θ produces activations $\{h_1, \ldots, h_L\}$ where h_l is the output of the l-th block.



(a) Decodable NN (b) Recursively Decodable NN

Figure 2: *Decodable Neural Networks*: (left) A generative model g_ϕ is jointly trained with a classifier f_θ to map activations to inputs. The images shown are actual reconstructions from a trained network that misclassified whether a celebrity is wearing a hat. (right) A recursively decodable neural network.

To train a neural network, we solve an optimization function of the form

$$\mathcal{L}_{\text{nn}}(x;\theta) = \log p(y|f_\theta(x)) + \beta\Omega(x;\theta) \tag{1}$$

using stochastic gradient descent. In Equation 1, the notation x denotes an input example and y its label. The function Ω represents an auxiliary objective such as a regularizer. The hyperparameter $\beta > 0$ is used to scale the auxiliary loss. For classification, $\log p(y|\cdot)$ is cross entropy.

Generative Models Broadly, we are given a latent variable H and an observed variable X. A generative model g_ϕ captures the distribution $p(x|h)$, mapping samples of the latent code h to samples of the observed variable x. For a deep generative model, g_ϕ is parameterized by a neural network. In the simplest case where g_ϕ is an autoencoder, we are also given an encoder q that maps observed samples x to latent codes h. Then, we optimize the objective

$$\mathcal{L}_{\text{ae}}(x;\phi) = \log p(x|g_\phi(q(x))) \tag{2}$$

where for continuous x, the expression $\log p(x|h)$ is the log Gaussian probability with mean h and fixed variance (or equivalently, mean squared error). More complex generative models might impose a prior distribution on h [27], an adversarial discriminator [35, 18], or invertibility of g_ϕ [44].

3 Neural Networks with Decodable Activations

In Section 1, we motivated our desiderata for activations that can be mapped back to the input space. We now propose to do this by "inverting" a neural network at a particular activation. Concretely, given an input x, a neural network f_θ, we can compute an activation h by computing the forward pass $f_\theta(x)$. Then, suppose we could construct the set:

$$S(h) = \{\hat{x} : \hat{x} \sim p_{\mathcal{D}}, f_\theta(\hat{x}) = h\} \tag{3}$$

where $p_{\mathcal{D}}$ represents the empirical distribution that x is sampled from. Now, any input $\hat{x} \in S(h)$ is a valid mapping of h back to the input space. (Trivially, $x \in S(h)$.) We call this $\hat{x}$ a *decoding* of the activation h. Unfortunately, in practice we do not know how to construct such a set $S(h)$.

But, we do know how to (approximately) sample from $S(h)$. We can train a generative model g_ϕ to reconstruct the original input x from neural network activations h (recall $x \in S(h)$). Then, we can sample $x \sim g_\phi(h)$, which we know how to do, to sample from $S(h)$. If we jointly train this generative model g_ϕ with the neural network f_θ, we can optimize for activations that both solve the classification task and decode back to inputs at once.

So far, we have only considered a single activation despite our neural network having L such activations. Although one could train a separate generative model per layer, we found it efficient and performant to share weights ϕ across all layers. Our joint objective is:

$$\mathcal{L}(x;\theta,\phi) = \log p(y|f_\theta(x)) + \beta\frac{1}{L}\sum_{l=1}^{L}\log p(x|g_\phi(h_l)), \qquad h_1, ..., h_L \text{ come from } f_\theta(x) \tag{4}$$

where the first term is cross entropy as in Equation 1 and the second term is a reconstruction error (Equation 2) averaged over layers. Comparing Equation 4 to Equation 1, we can interpret the generative loss as a regularization term Ω. Intuitively, in a standard neural network, the activations $h_1, \ldots, h_L$ are unconstrained, chosen only to minimize the loss. However, in Equation 4, the network f_θ is constrained to produce activations that the generative model g_ϕ can invert. We call this setup a *Decodable Neural Network*, or DecNN. See Figure 2a for an illustration.

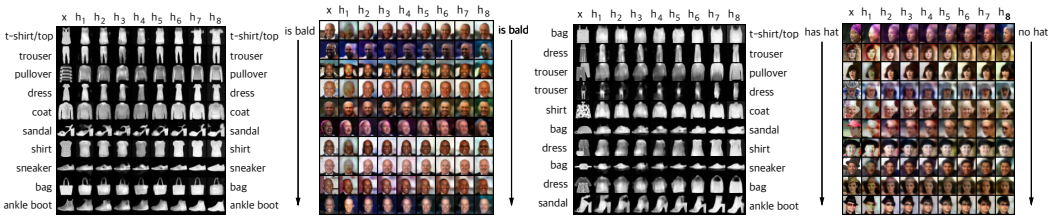
After training, given a new example x^*, we first compute hidden activations $h_1^*, \ldots h_L^*$ through $f_\theta(x^*)$. For each activation h_l^*, we can decode it to an input using $g_\phi(h_l^*)$. In the remaining sections, we show how DecNN enables us to probe what information is stored in an activation (Section 4), compose neural networks modules (Section 5), and constrain what a network learns (Section 6).

4 Probing Decodable Activations

For a classifier, the Data Processing Inequality [7, 4] tells us that information about the input x embedded in activation h_l decreases as l increases. More formally, the mutual information follows a sequence of inequalities $\mathcal{I}(X;H_1) \geq \cdots \geq \mathcal{I}(X;H_L) \geq \mathcal{I}(X;Y)$ where X, H_l, and Y are random variables representing the inputs, activations, and outputs, respectively. In practice, this theory leads us to expect "useless" information to be dropped deeper into the network, while "useful" information is kept to solve the classification task.

In a traditional software program, debugging code amounts to creating breakpoints and printing values to understand what information a variable contains. For a neural network, we similarly wish to probe what information an activation captures. While many algorithms have been developed for specific architectures [55, 1, 34], they require a nontrivial amount of additional computation. With DecNN, probing activations is more straightforward. Since decoded activations have the same type as inputs (which we assume are human interpretable), we can directly visualize information. If we examine all L activations at once, we can track what information is dropped through the network.

Table 1a shows test accuracies comparing DecNN to an standard network, finding no performance loss. Given this, Figure 3 shows randomly chosen examples taken from DecNNs trained on FashionMNIST and CelebA (see Appendix for full figure). In each subfigure, the leftmost image is from the dataset while the remaining eight images are decoded activations (f_θ is an MLP with $L = 8$ layers).



(a) (b) (c) (d)

Figure 3: *Visualizing activations*: the left two images shows decoded activations when the model correctly classified examples; the right two shows misclassifications.

The left two images show decoded activations for correctly-classified examples whereas the right two images show mis-classified examples. For correctly-classified examples, we observe that the decoded activations tends towards class prototypes. Decoded images from the 7th or 8th activation lose details (e.g. patterns, logos, or shoe straps disappear from clothing). The top row of Figure 3a presents a good example: the tank-top (a rare form of the t-shirt) is projected to a more prototypical t-shirt. We also find that mis-classified examples morph into images of the incorrect class. The seventh row of Figure 3c morphs the two articles of clothing that compose the dress to build a shirt. The seventh row of Figure 3d show the celebrity's hat transform into a background color.

As a practitioner, it is useful to understand what features a neural network is attending to in making a prediction, or to understand what features make it susceptible to making an error. Decodable activations take a step towards "printing activations" for debugging.

5 Composing Decodable Neural Networks

In traditional software, given two functions f_1 and f_2, we can easily compose them together by $f_1(f_2(x))$ for some input x. With this as inspiration, we wish to similarly compose neural networks together. But given two image classifiers, it is not clear how to do so. *How do we tie outputs from one neural network to inputs for the other?* Both networks expect images as input.

Since decoded activations have the same type as inputs, we are free to treat decoded activations as inputs for another model. In other words, we can easily compose neural networks together using decoded activations as a conduit: given two models f_1 and f_2, and an input x, we can compute $f_2(g_\phi(h))$ where h is derived from $f_1(x)$ and use it to optimize a joint objective. In the next two subsections, we explore two kinds of neural network compositions: one by composing a model with itself and the other by composing a model with a pretrained one.

5.1 Recursive Self-Composition

Our first composition will be to compose a DecNN model with itself. But if we can do that, we can also recursively compose a DecNN model with itself infinitely:

Given a decoded activation $g_\phi(h_l^{(0)})$ for any layer l, we can feed it as input back into the classifier, $f_\theta(g_\phi(h_l^{(0)}))$ and maximize agreement with the true label y. Intuitively, if $g_\phi(h_l)$ is a good reconstruction of the input x (i.e. $g_\phi(h_l)$ and x belong to the same set $S(h_l)$), then a good classifier f_θ should predict the correct label with it as input. Now, we also observe that the computation $f_\theta(g_\phi(h_l^{(0)}))$

produces L additional activations, $h_1^{(1)}, \ldots, h_L^{(1)}$ where we use the superscript to represent the number of compositions. We can repeat this exact process with the same intuition, decoding $h_l^{(1)}$ and passing it to f_θ to produce $h_l^{(2)}$. Computational limits aside, this can be repeated ad infinitum.

One interpretation of this procedure is as data augmentation: we saw in Section 4 that decoded activations share features relevant to the prediction task (e.g. generated celebrities wear hats) but vary other irrelevant features (e.g. face shape, hair color, etc.). Notably, these generated images are not in the original dataset. Thus, training a model using decoded activations as inputs is similar to augmenting the initial dataset. The only difference is that gradients are propagated back through recursion to the inputs, which is also a form of regularization.

Admittedly, what we have described so far is intractable. Besides recursing infinitely, each activation produces L more activations, creating a tree (with root node x) with a large branching factor L. In practice, we make two adjustments — First, we limit the recursion to a max depth D. Second, instead of constructing the full graph of activations, which would have $O(L^D)$ nodes, we randomly sample "paths" in the graph. Specifically, starting at the root, we compute $\{h_1^{(0)}, \ldots, h_L^{(0)}\}$ through $f_\theta(x)$ and randomly choose an integer $l_0 \in [1, L]$. Then, we compute $\{h_1^{(1)}, \ldots, h_L^{(1)}\}$ through $f_\theta(g_\phi(h_{l_0}^{(0)}))$ i.e. we classify the l_0-th decoded activation from depth 0. Again, we randomly pick l_1 to compute $\{h_1^{(2)}, \ldots, h_L^{(2)}\}$ through $f_\theta(g_\phi(h_{l_1}^{(1)}))$ and repeat until depth D.

Building on Equation 4, we can write this "recursive" objective as:

$$\hat{\mathcal{L}}(x;\theta,\phi) = \log p(y|f_\theta(x)) + \sum_{d=1}^{D} \alpha^d \log p(y|f_\theta(g_\phi(h_{l_{d-1}}^{(d-1)}))) + \beta \sum_{d=1}^{D} \alpha^d \log p(x|g_\phi(h_{l_{d-1}}^{(d-1)})) \quad (5)$$

where the sequence of integers $l_0, \ldots l_{D-1}$ are each sampled uniformly from $[1, L]$. We introduce a new hyperparameter $\alpha \in [0, 1]$ that geometrically downweights later depths (if $\alpha = 0$, Equation 5 reduces to maximum likelihood). The first term in Equation 5 is the usual classification objective. The second term is a sum of cross entropy terms that encourage the decoded activation at depth d to predict the correct label through f_θ. The third and final term is a sum of reconstruction losses that encourage the decoded activation at depth d to approximate the original input x.

We call this model a "Recursively Decodable Neural Network", abbreviated ReDecNN. Note that ReDecNN is a special case of an ensemble network. That is, every path sampled in the activation tree by picking $l_0, \ldots l_{D-1}$ builds a legitimate classifier. There are $O(L^D)$ such classifiers implicit in the ReDecNN. We can interpret optimizing Equation 5 as training individual classifiers that are Monte Carlo sampled from the tree every gradient step. See Figure 2b for an illustration.

Our goal in the next few paragraphs is to utilize this ensemble to measure uncertainty.

Method	MNIST	Fashion	CelebA
Standard	97.1 (0.11)	87.7 (0.19)	90.8 (<0.1)
DecNN	**97.8** (0.17)	**88.8** (0.19)	90.8 (<0.1)
ReDecNN	97.5 (0.20)	88.1 (0.16)	90.8 (0.16)
Dropout	95.9 (0.13)	80.7 (0.24)	88.5 (0.12)
MC-Dropout	91.3 (0.14)	78.6 (0.23)	87.8 (0.19)
BayesNN	96.3 (0.51)	87.0 (0.36)	88.2 (0.22)
Ensemble	97.6 (0.11)	88.4 (0.12)	**90.9** (0.11)

(a) Classification performance on test set.

Method	MNIST	Fashion	CelebA
ReDecNN	0.869	0.795	**0.692**
MC-Dropout	**0.878**	**0.818**	0.657
BayesNN	0.537	0.574	0.587
Ensemble	0.531	0.559	0.605

(b) ROC-AUC of separating correctly classified and misclassified examples using Eq. 6 as a score. The stdev. for all entries over 3 runs are <0.01.

Measuring Uncertainty As a first step, we compare the performance of ReDecNN to several baselines: a standard neural network, a neural network with dropout, two kinds of Bayesian neural networks — Monte Carlo dropout [15], abbreviated MC-Dropout, and weight uncertainty [6], abbreviated BayesNN — and a naive ensemble network where we train D copies of the standard neural network with different initializations. See supplement for training details. Table 1a shows accuracies over a held-out test set, averaged over three runs. While we observe a drop in accuracy in Bayesian NNs, we observe equal performance of ReDecNN to a standard neural network.

The next step is to compare the quality of model uncertainty. For the ReDecNN, for a given input x, we measure uncertainty as follows: sample N different classifiers from the activation tree, and make

predictions with each classifier on x. The uncertainty is the entropy of the predictions:

$$\text{Uncertainty}(x) = -\sum_{c=1}^{K} \hat{p}(y=c) \log \hat{p}(y=c) \tag{6}$$

where $\hat{p}(y=c)$ is the empirical probability (e.g. normalized count) of predicting class c out of N classifiers. A higher uncertainty metric would represent more disagreement between classifiers. We can compute the same metric for a naive ensemble, as well as for MC-Dropout (where repeated calls drop different nodes) and BayesNN (by sampling weights from the learned posterior).

We want to test if the model's uncertainty is *useful*. One way is to correlate the uncertainty with when the model makes prediction errors. It would be useful if the model was less uncertain on correct predictions and more uncertain on incorrect ones. A practitioner could then use uncertainty to anticipate when a model might make a mistake. Table 1b reports the area under the ROC curve, or ROC-AUC of separating correctly and incorrectly classified examples in the test set (using uncertainty as a score). A higher ROC-AUC, closer to 1, represents a better separation. We find that ReDecNN is competitive with MC-Dropout, even out performing it on CelebA. In Section A.3, we include examples of images with low and high uncertainty.

Out-of-Distribution Detection A second application of uncertainty is detecting out-of-distribution (OOD) inputs. Given an input that is far from the training distribution, a model is likely to make a mistake. Instead, the model should recognize its uncertainty, and refuse to make a prediction. There is a rich literature on detecting OOD inputs using downstream computation on a trained model [22, 31, 30, 68, 23]. Unlike those works, we study OOD detection using uncertainty alone.

	MNIST			FashionMNIST			CelebA		
	ReDecNN	MC	Ensemble	ReDecNN	MC	Ensemble	ReDecNN	MC	Ensemble
Adversarial (FGSM)	0.787	**0.817**	0.540	0.711	**0.760**	0.589	-	-	-
OOD (MNIST)	-	-	-	0.793	**0.864**	0.501	**0.647**	0.548	0.591
OOD (FashionMNIST)	0.812	**0.888**	0.534	-	-	-	**0.641**	0.572	0.599
OOD (CelebA)	0.893	**0.943**	0.675	0.753	**0.793**	0.704	-	-	-
Corrupt (Mean)	0.727	**0.785**	0.566	0.676	**0.691**	0.606	**0.686**	0.569	0.632
Corrupt (Stdev)	0.075	0.113	0.087	0.072	0.108	0.106	0.038	0.017	0.016

Table 2: ROC-AUC of predicting which examples are out-of-distribution (OOD). We vary OOD examples to be adversarial, corrupted, or from a different dataset. Standard deviation for all entries are < 0.01 over three runs.

We explore three types of OOD examples: (1) adversarial examples crafted for a single classifier using FGSM [19], (2) examples from a different dataset (e.g. train on MNIST and consider FashionMNIST as OOD) as done in [31], and (3) and corrupted examples by 14 image transformations (e.g. adding pixel noise or rotating an image), borrowed from [39].

Table 2 reports the ROC-AUC of separating inlier examples taken from the test split of the dataset the models were trained on, and outlier examples. From Table 2, we observe ReDecNN is just under MC-Dropout, but while MC-Dropout achieves these results at the cost of classification accuracy, ReDecNN does not (Table 1a). Furthermore, we find ReDecNN generalizes better to CelebA, a more complex image dataset, where it outperforms MC-Dropout (and other baselines).

OOD: Wearing a Hat	
Method	ROC-AUC
ReDecNN	**0.604** (<0.01)
MC-Dropout	0.519 (<0.01)
BayesNN	0.510 (0.01)
Ensemble	0.526 (<0.01)

OOD: Blond Hair	
Method	ROC-AUC
ReDecNN	**0.593** (<0.01)
MC-Dropout	0.502 (<0.01)
BayesNN	0.508 (0.01)
Ensemble	0.509 (<0.01)

OOD: Bald	
Method	ROC-AUC
ReDecNN	**0.615** (<0.01)
MC-Dropout	0.502 (<0.01)
BayesNN	0.508 (<0.01)
Ensemble	0.503 (<0.01)

Table 3: We hold out a group of CelebA attributes, such as those wearing a hat, when training. We compute the ROC-AUC of labeling the held-out group as OOD (mean and stdev. over 3 runs).

Focusing on CelebA, we study a domain-specific OOD challenge by holding out all images with positive annotations for a single attribute as out-of-distribution. Here, inlier and outlier examples are very similar in distribution. We report ROC-AUCs in Table 3 for three different held-out attributes: is the celebrity in the image "wearing a hat", has "blonde hair", or "is bald". As this is a more challenging problem, the performance is lower than in Table 2. But while MC-Dropout, BayesNN, and a naive ensemble all perform near chance, ReDecNN is consistently near 0.6.

Method	MNIST	Fashion	CelebA
Standard	1.18 (0.10)	0.56 (0.14)	2.75 (0.13)
ReDecNN	**0.33** (0.05)	0.21 (0.03)	**1.92** (0.13)
Dropout	0.63 (0.09)	0.48 (0.09)	2.51 (0.13)
MC-Dropout	0.41 (0.06)	**0.13** (0.03)	2.99 (0.15)
BayesNN	0.91 (0.20)	0.18 (0.03)	2.41 (0.19)
Ensemble	1.80 (0.12)	0.89 (0.07)	3.12 (0.12)

Table 4: We compare the expected calibration error (ECE) of ReDecNN to a standard neural network with various regularization and ensembling.

Calibration A third application of uncertainty we explore is calibration. Standard neural networks are notoriously over-confident and are incentived in training to predict with extreme probabilities (e.g. 0.99). A model with useful uncertainty would make predictions with proper probabilities. Although many calibration algorithms exist [41, 20], we want to measure how calibrated each model is out-of-the-box to compare the model uncertainties.

Table 4 reports the expected calibration error, or ECE [40], which approximates the difference in expectation between confidence accuracy using discrete bins. A lower number (closer to 0) is a more calibrated model. We observe that while a standard neural network has a relatively high ECE, dropout, MC-Dropout, BayesNN, and ReDecNN all reduce ECE. In two of three datasets, ReDecNN achieves the lowest ECE whereas MC-Dropout achieves the lowest in the third. These results should be viewed in parallel to Table 1a, which measures "sharpness". Otherwise, we can trivially reduce ECE by ignoring the input.

5.2 Composing with Pretrained Models

Apart from self-composition, we can also compose our neural networks with off-the-shelf pretrained models as a form of regularization or distillation. Suppose we have a pretrained model m that maps an input x to a class prediction. We can modify the ReDecNN objective as follows:

$$\tilde{\mathcal{L}}(x;\theta,\phi) = \hat{\mathcal{L}}(x;\theta,\phi) + \sum_{d=1}^{D} \alpha^d \log p(f_\theta(g_\phi(h_{l_{d-1}}^{(d-1)}))|m(g_\phi(h_{l_{d-1}}^{(d-1)}))) \tag{7}$$

where $\hat{\mathcal{L}}$ is as defined in Equation 5. The additional term acts as a divergence, bringing the classifer's predictions close to those of the pretrained model. A similar edit can be made to the DecNN. We revisit Section 5 experiments now using two pretrained models — a linear classifier or a ResNet-18 classifier. For each, we optimize Equation 7 and compute performance on OOD detection.

Method	MNIST None	MNIST Linear	MNIST ResNet	CelebA None	CelebA Linear	CelebA ResNet
Accuracy	97.50 (0.20)	97.17 (0.12)	**98.30** (0.24)	90.80 (<0.1)	89.76 (0.11)	**91.28** (0.13)
Misclassfication	0.869 (<0.01)	**0.904** (<0.01)	0.742 (<0.01)	0.692 (<0.01)	**0.732** (<0.01)	0.615 (<0.01)
Adversarial	0.787 (<0.01)	**0.805** (0.01)	0.729 (<0.01)	-	-	-
OOD (MNIST)	-	-	-	0.647 (<0.01)	**0.693** (<0.01)	0.609 (<0.01)
OOD (Fashion)	0.812 (<0.01)	**0.845** (<0.01)	0.650 (<0.01)	0.641 (<0.01)	**0.673** (<0.01)	0.585 (<0.01)
OOD (CelebA)	0.893 (<0.01)	**0.928** (<0.01)	0.862 (<0.01)	-	-	-
Corruption (Mean)	0.727 (<0.01)	**0.790** (<0.01)	0.694 (<0.01)	0.686 (<0.01)	**0.708** (<0.01)	0.616 (<0.01)
Calibration (ECE)	0.334 (0.05)	**0.296** (0.01)	0.385 (0.02)	**1.927** (0.13)	1.937 (0.04)	3.002 (0.10)

Table 5: OOD detection using uncertainty for ReDecNN composed with pretrained models.

Table 5 compares the results of incorporating a linear model to a residual network to no pretrained model (None). Using a linear model, we observe a slight drop in test accuracy but an increase across all OOD experiments and calibration. In fact, these new results rival or surpass MC-Dropout from Table 2. Conversely, with ResNet-18, we observe a 1 point increase in test accuracy but notable drops in OOD and calibration results. These polarizing outcomes show two use cases of pretrained models: using simpler models allow for regularization that trades off accuracy for robustness while more complex models encourage distillation, making the opposing tradeoff.

6 Constraining Decodable Computation

In traditional software, assertion statements are a popular way to make sure the program is executing as expected. Ideally, we would have the same functionality in neural networks: the ability to specify what information an activation should *not* capture. This has a close relationship with fairness in machine learning [36], where we have knowledge of sensitive or protected attributes that should not be abused in solving a task. Our desired "assertion statement" for neural networks should proactively block the representations from utilizing the protected attributes in prediction.

Suppose we have access to a pretrained classifier m that predicts the assignment for a K-way protected feature. Then, building on Section 5.2, we can optimize our classifier to ignore information about the protected feature through m. To do this, we define the objective:

$$\tilde{\mathcal{L}}(x;\theta,\phi) = \hat{\mathcal{L}}(x;\theta,\phi) + \sum_{d=1}^{D} \alpha^d \log p(m(g_\phi(h_{l_{d-1}}^{(d-1)}))|\frac{1}{K}\mathbf{1}_K) \tag{8}$$

where $\mathbf{1}_K$ is a K-dimensional vector of ones. That is, Equation 8 encourages the decoded activation $g_\phi(h_{l_{d-1}}^{(d-1)})$, when passed through the protected classifier m, to predict chance probabilities i.e. have no information about the protected attribute. A high performing classifier trained in this manner must have solved the prediction task without abusing the protected attribute(s).

To test this, we extract two attributes from CelebA: "attractiveness" and "pale skin". As the vast majority of CelebA images contain celebrities with pale skin, naively training a binary classifier for "attractiveness" could lead to a discrepancy in performance between images of celebrities with and without pale skin. Indeed, Table 6 shows a 8 point difference in F1 and a 9 point difference in average precision (AP) in classifying attractiveness between the two groups. The second row of Table 6 provides a baseline ReDecNN optimized without knowledge of the protected attribute. We find a similar discrepany between groups as to a standard neural network. In the third row, we evaluate a ReDecNN that was optimized to ignore the "pale skin" attribute using a pretrained classifier for pale skin. Critically, we find much more balanced F1, AP, and ECE across the two groups.

Model	F1 (Pale / Not Pale)	AP (Pale / Not Pale)	ECE (Pale / Not Pale)
Standard	0.843/0.776 (<0.01 / <0.01)	78.55/69.36 (0.05 / 0.14)	2.22/2.94 (0.01 / 0.12)
ReDecNN (Eq. 4)	0.845/0.780 (<0.01 / 0.01)	78.28/70.34 (0.07 / 0.14)	2.05/2.69 (0.04 / 0.08)
ReDecNN (Eq. 8)	**0.833/0.806** (<0.01 / 0.01)	**77.36/76.27** (0.09 / 0.14)	**2.06/2.12** (0.07 / 0.09)

Table 6: We use a pretrained model on CelebA that predicts pale skin to optimize the activations of a second classifier to ignore skin color when predicting attractiveness.

7 Generalizing to other Modalities

While we have focused on image classification, the proposed approach is more general and can be extended to other modalities. We apply the same decodable representations to speech, in particular utterance and action recognition. We compare DecNN and ReDecNN to the same baselines, measuring accuracy and uncertainty through similar experiments as we did for images.

Method	Acc	Misclass.	OOD	ECE
Standard	94.5 (0.1)	-	-	-
DecNN	93.8 (0.2)	-	-	-
ReDecNN	93.4 (0.2)	0.766	0.705	**0.225** (0.1)
MC-Dropout	82.1 (0.4)	**0.788**	0.745	0.429 (0.2)
BayesNN	91.6 (0.7)	0.545	0.518	1.039 (0.4)
Ensemble	**96.3** (0.1)	0.529	0.506	1.103 (0.2)

(a) AudioMNIST

Method	Acc	Misclass.	OOD	ECE
Standard	42.4 (0.4)	-	-	-
DecNN	41.4 (0.7)	-	-	-
ReDecNN	41.2 (0.5)	**0.642**	**0.605**	0.523 (0.2)
MC-Dropout	34.5 (0.7)	0.629	0.562	**0.515** (0.2)
BayesNN	40.3 (1.2)	0.523	0.500	0.918 (0.1)
Ensemble	**44.1** (0.1)	0.541	0.522	1.189 (0.5)

(b) Fluent Speech Commands

Table 7: Performance on speech classification. If not specified, stdev. is <0.01 over three test runs.

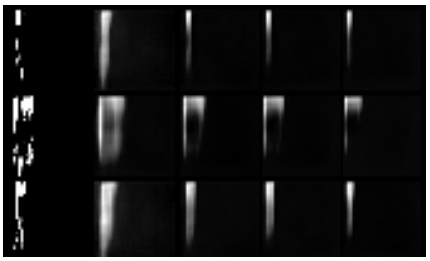


Figure 4: Decoding activations to images for Fluent spectrograms.

We utilize the AudioMNIST [5] and Fluent speech commands [33] datasets, the former being a corpus of 60k recorded utterances of the digits 0 through 9 and the latter a corpus of 100k recordings of 97 speakers interacting with smart-home appliances. Each utterance demarcates one of six actions. Audio waveforms are preprocessed to log Mel spectrograms, outputting a 32 by 32 dimensional matrix, as done in [61]. Figure 4 shows the input on the left-most column along with the 2nd, 4th, 6th, and 8th decoded activations for 3 random test examples. We can observe information being dropped through the network, although the primary structure is preserved. Table 7 reports the findings, where like the image experiments, we

find ReDecNN has strong performance on OOD detection and calibration in return for only a small drop in performance compared to a standard neural network.

8 Related Work

Probing Neural Networks A number of works seek to probe neural network representations post-training to understand the underlying computation. Some propose to do this through distillation to simpler models [49], visualizing gradients [55, 42, 10], estimating feature importance [1] and node importance [3], while others leverage influence functions [28] or natural language [38]. Most relevant to our work is an approach to understand CNN representations by inverting them through reconstruction [34]. Unlike these approaches, we are not proposing computation post-hoc. Rather, we are interested in *optimizing* neural networks such that their computation is more easily probed by mapping representations back to the input domain. There are some works that optimize for interpretability [64, 51], such as regularizing a network's predictions to agree with a decision tree. However, these methods have difficulty generalizing to richer modalities, such as images.

Model Robustness and Fairness Out-of-distribution detection [32, 30, 68, 52, 24, 23], selective classification [16, 17], adversarial perturbation detection [65, 14, 37, 43], and neural network calibration [20, 67, 66, 40, 47] each have a rich subfield of algorithms. While task-specific methods likely outperform our results, we are excited about representations that enable uncertainty in a way that is useful — requiring no task information nor additional algorithms. For model fairness, our approach shares similarities to [56] where the authors maximize the mutual information of representations with the input while minimizing the mutual information with the protected attribute. Our approach is an approximation of this where we treat the pretrained classifier for the protected attribute as a proxy for mutual information with the sensitive attribute.

Invertible and Ensemble Networks The DecNN is akin to invertible generative models [9, 45] as DecNN is training a generative model to "invert" the classifier up to a hidden layer. However, unlike invertible flows, DecNN does not impose an explicit prior on the activations (i.e. latent variables), and further, DecNN has a supervised objective. Our proposed recursive network, ReDecNN, has similarities to Bayesian neural networks [15, 6] and ensemble networks [62] — two baselines we compare against for evaluating model uncertainty. However, we find ReDecNN to be easier to train than doing inference over weights, and cheaper in parameter count than naive ensembles.

9 Limitations and Future Work

We explored building neural networks with decodable representations, inspired by both the difficulty of maintaining, composing, and debugging modern neural networks, as well as the gap between traditional software and deep learning in usability. By enforcing decoded activations to have the same "type" as the input, we were able to visualize information embedded in an activation, compose neural networks together in useful ways for out-of-distribution detection and calibration, and impose real-world constraints on what information a network uses to learn. We are optimistic that this is only the start and look to future research to explore more applications of decodable representations.

We discuss a few limitations. First, our approach is bottlenecked by the quality of the generative model. Without a good reconstruction, optimization will be intractable. However, in light recent work [59, 58, 8, 25, 57], this is becoming less of a problem as new generative models surface. Second, we only explored feedforward classifiers in our experiments for simplicitly. Our approach extends naturally to residual and transformer blocks, and future research could explore this direction. Third, optimizing recursive networks, although parameter efficient, costs more compute as backpropagation is more expensive. Future work could explore weight sharing across activations (not just depth).

On a broader scale, one of the challenges of practical deep learning is understanding model computation. High dimensional vectors with nonlinearities are not easily interpreted by humans and as a result, deploying and maintaining neural networks is a constant challenge. While a significant amount research is dedicated to "white-boxing" neural networks, we view our work as an alternative approach: rather than try to understand what a model has learned after training, we seek to build models that bake in a way to understand and utilize network activations in a human medium. From a different perspective, by its very nature, decodable representations expose data through every activation. For sensitive, personal, or secure data, using DecNN essentially reveals that information to every actor that interacts with the model, facing privacy concerns.

References

[1] Sebastian Bach, Alexander Binder, Grégoire Montavon, Frederick Klauschen, Klaus-Robert Müller, and Wojciech Samek. On pixel-wise explanations for non-linear classifier decisions by layer-wise relevance propagation. *PloS one*, 10(7):e0130140, 2015.

[2] Dzmitry Bahdanau, Kyunghyun Cho, and Yoshua Bengio. Neural machine translation by jointly learning to align and translate. *arXiv preprint arXiv:1409.0473*, 2014.

[3] David Bau, Bolei Zhou, Aditya Khosla, Aude Oliva, and Antonio Torralba. Network dissection: Quantifying interpretability of deep visual representations. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 6541–6549, 2017.

[4] Normand J Beaudry and Renato Renner. An intuitive proof of the data processing inequality. *arXiv preprint arXiv:1107.0740*, 2011.

[5] Sören Becker, Marcel Ackermann, Sebastian Lapuschkin, Klaus-Robert Müller, and Wojciech Samek. Interpreting and explaining deep neural networks for classification of audio signals. *CoRR*, abs/1807.03418, 2018.

[6] Charles Blundell, Julien Cornebise, Koray Kavukcuoglu, and Daan Wierstra. Weight uncertainty in neural network. In *International Conference on Machine Learning*, pages 1613–1622. PMLR, 2015.

[7] Thomas M Cover. *Elements of information theory*. John Wiley & Sons, 1999.

[8] Prafulla Dhariwal and Alex Nichol. Diffusion models beat gans on image synthesis. *arXiv preprint arXiv:2105.05233*, 2021.

[9] Laurent Dinh, Jascha Sohl-Dickstein, and Samy Bengio. Density estimation using real nvp. *arXiv preprint arXiv:1605.08803*, 2016.

[10] Dumitru Erhan, Yoshua Bengio, Aaron Courville, and Pascal Vincent. Visualizing higher-layer features of a deep network. *University of Montreal*, 1341(3):1, 2009.

[11] Piero Esposito. Blitz - bayesian layers in torch zoo (a bayesian deep learing library for torch). https://github.com/piEsposito/blitz-bayesian-deep-learning/, 2020.

[12] et al. Falcon, WA. Pytorch lightning. *GitHub. Note: https://github.com/PyTorchLightning/pytorch-lightning*, 3, 2019.

[13] William Falcon and Kyunghyun Cho. A framework for contrastive self-supervised learning and designing a new approach. *arXiv preprint arXiv:2009.00104*, 2020.

[14] Reuben Feinman, Ryan R Curtin, Saurabh Shintre, and Andrew B Gardner. Detecting adversarial samples from artifacts. *arXiv preprint arXiv:1703.00410*, 2017.

[15] Yarin Gal and Zoubin Ghahramani. Dropout as a bayesian approximation: Representing model uncertainty in deep learning. In *international conference on machine learning*, pages 1050–1059. PMLR, 2016.

[16] Yonatan Geifman and Ran El-Yaniv. Selective classification for deep neural networks. *arXiv preprint arXiv:1705.08500*, 2017.

[17] Yonatan Geifman and Ran El-Yaniv. Selectivenet: A deep neural network with an integrated reject option. In *International Conference on Machine Learning*, pages 2151–2159. PMLR, 2019.

[18] Ian J Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, and Yoshua Bengio. Generative adversarial networks. *arXiv preprint arXiv:1406.2661*, 2014.

[19] Ian J Goodfellow, Jonathon Shlens, and Christian Szegedy. Explaining and harnessing adversarial examples. *arXiv preprint arXiv:1412.6572*, 2014.

[20] Chuan Guo, Geoff Pleiss, Yu Sun, and Kilian Q Weinberger. On calibration of modern neural networks. In *International Conference on Machine Learning*, pages 1321–1330. PMLR, 2017.

[21] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 770–778, 2016.

[22] Dan Hendrycks and Kevin Gimpel. A baseline for detecting misclassified and out-of-distribution examples in neural networks. *arXiv preprint arXiv:1610.02136*, 2016.

[23] Dan Hendrycks, Mantas Mazeika, and Thomas Dietterich. Deep anomaly detection with outlier exposure. *arXiv preprint arXiv:1812.04606*, 2018.

[24] Dan Hendrycks, Mantas Mazeika, Saurav Kadavath, and Dawn Song. Using self-supervised learning can improve model robustness and uncertainty. In *Advances in Neural Information Processing Systems*, pages 15663–15674, 2019.

[25] Jonathan Ho, Ajay Jain, and Pieter Abbeel. Denoising diffusion probabilistic models. *arXiv preprint arXiv:2006.11239*, 2020.

[26] Andrej Karpathy, Nov 2017.

[27] Diederik P Kingma and Max Welling. Auto-encoding variational bayes. *arXiv preprint arXiv:1312.6114*, 2013.

[28] Pang Wei Koh and Percy Liang. Understanding black-box predictions via influence functions. In *International Conference on Machine Learning*, pages 1885–1894. PMLR, 2017.

[29] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. Imagenet classification with deep convolutional neural networks. *Advances in neural information processing systems*, 25:1097–1105, 2012.

[30] Kimin Lee, Kibok Lee, Honglak Lee, and Jinwoo Shin. A simple unified framework for detecting out-of-distribution samples and adversarial attacks. In *Advances in Neural Information Processing Systems*, pages 7167–7177, 2018.

[31] Shiyu Liang, Yixuan Li, and R Srikant. Enhancing the reliability of out-of-distribution image detection in neural networks. In *International Conference on Learning Representations*, 2018.

[32] Shiyu Liang, Yixuan Li, and Rayadurgam Srikant. Enhancing the reliability of out-of-distribution image detection in neural networks. *arXiv preprint arXiv:1706.02690*, 2017.

[33] Loren Lugosch, Mirco Ravanelli, Patrick Ignoto, Vikrant Singh Tomar, and Yoshua Bengio. Speech model pre-training for end-to-end spoken language understanding. *arXiv preprint arXiv:1904.03670*, 2019.

[34] Aravindh Mahendran and Andrea Vedaldi. Understanding deep image representations by inverting them. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 5188–5196, 2015.

[35] Alireza Makhzani, Jonathon Shlens, Navdeep Jaitly, Ian Goodfellow, and Brendan Frey. Adversarial autoencoders. *arXiv preprint arXiv:1511.05644*, 2015.

[36] Ninareh Mehrabi, Fred Morstatter, Nripsuta Saxena, Kristina Lerman, and Aram Galstyan. A survey on bias and fairness in machine learning. *arXiv preprint arXiv:1908.09635*, 2019.

[37] Jan Hendrik Metzen, Tim Genewein, Volker Fischer, and Bastian Bischoff. On detecting adversarial perturbations. *arXiv preprint arXiv:1702.04267*, 2017.

[38] Jesse Mu and Jacob Andreas. Compositional explanations of neurons. *arXiv preprint arXiv:2006.14032*, 2020.

[39] Norman Mu and Justin Gilmer. Mnist-c: A robustness benchmark for computer vision. *arXiv preprint arXiv:1906.02337*, 2019.

[40] Mahdi Pakdaman Naeini, Gregory Cooper, and Milos Hauskrecht. Obtaining well calibrated probabilities using bayesian binning. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 29, 2015.

[41] Alexandru Niculescu-Mizil and Rich Caruana. Predicting good probabilities with supervised learning. In *Proceedings of the 22nd international conference on Machine learning*, pages 625–632, 2005.

[42] Chris Olah, Alexander Mordvintsev, and Ludwig Schubert. Feature visualization. *Distill*, 2017. https://distill.pub/2017/feature-visualization.

[43] Tianyu Pang, Chao Du, Yinpeng Dong, and Jun Zhu. Towards robust detection of adversarial examples. In *NeurIPS*, 2018.

[44] George Papamakarios, Eric Nalisnick, Danilo Jimenez Rezende, Shakir Mohamed, and Balaji Lakshminarayanan. Normalizing flows for probabilistic modeling and inference. *arXiv preprint arXiv:1912.02762*, 2019.

[45] George Papamakarios, Theo Pavlakou, and Iain Murray. Masked autoregressive flow for density estimation. *arXiv preprint arXiv:1705.07057*, 2017.

[46] Guim Perarnau, Joost Van De Weijer, Bogdan Raducanu, and Jose M Álvarez. Invertible conditional gans for image editing. *arXiv preprint arXiv:1611.06355*, 2016.

[47] John Platt et al. Probabilistic outputs for support vector machines and comparisons to regularized likelihood methods. *Advances in large margin classifiers*, 10(3):61–74, 1999.

[48] Christopher Ré. Software 2.0 and snorkel: beyond hand-labeled data. In *Proceedings of the 24th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, pages 2876–2876, 2018.

[49] Marco Tulio Ribeiro, Sameer Singh, and Carlos Guestrin. " why should i trust you?" explaining the predictions of any classifier. In *Proceedings of the 22nd ACM SIGKDD international conference on knowledge discovery and data mining*, pages 1135–1144, 2016.

[50] Robin Rombach, Patrick Esser, and Bjorn Ommer. Network-to-network translation with conditional invertible neural networks. *Advances in Neural Information Processing Systems*, 33, 2020.

[51] Andrew Ross and Finale Doshi-Velez. Improving the adversarial robustness and interpretability of deep neural networks by regularizing their input gradients. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 32, 2018.

[52] Chandramouli Shama Sastry and Sageev Oore. Detecting out-of-distribution examples with in-distribution examples and gram matrices. *arXiv preprint arXiv:1912.12510*, 2019.

[53] Jürgen Schmidhuber. Deep learning in neural networks: An overview. *Neural networks*, 61:85–117, 2015.

[54] David Sculley, Gary Holt, Daniel Golovin, Eugene Davydov, Todd Phillips, Dietmar Ebner, Vinay Chaudhary, Michael Young, Jean-Francois Crespo, and Dan Dennison. Hidden technical debt in machine learning systems. *Advances in neural information processing systems*, 28:2503–2511, 2015.

[55] Ramprasaath R Selvaraju, Michael Cogswell, Abhishek Das, Ramakrishna Vedantam, Devi Parikh, and Dhruv Batra. Grad-cam: Visual explanations from deep networks via gradient-based localization. In *Proceedings of the IEEE international conference on computer vision*, pages 618–626, 2017.

[56] Jiaming Song, Pratyusha Kalluri, Aditya Grover, Shengjia Zhao, and Stefano Ermon. Learning controllable fair representations. In *The 22nd International Conference on Artificial Intelligence and Statistics*, pages 2164–2173. PMLR, 2019.

[57] Jiaming Song, Chenlin Meng, and Stefano Ermon. Denoising diffusion implicit models. *arXiv preprint arXiv:2010.02502*, 2020.

[58] Yang Song and Stefano Ermon. Generative modeling by estimating gradients of the data distribution. *arXiv preprint arXiv:1907.05600*, 2019.

[59] Yang Song, Jascha Sohl-Dickstein, Diederik P Kingma, Abhishek Kumar, Stefano Ermon, and Ben Poole. Score-based generative modeling through stochastic differential equations. *arXiv preprint arXiv:2011.13456*, 2020.

[60] Ilya Sutskever, Oriol Vinyals, and Quoc V Le. Sequence to sequence learning with neural networks. *arXiv preprint arXiv:1409.3215*, 2014.

[61] Alex Tamkin, Mike Wu, and Noah Goodman. Viewmaker networks: Learning views for unsupervised representation learning. *arXiv preprint arXiv:2010.07432*, 2020.

[62] Sean Tao. Deep neural network ensembles. In *International Conference on Machine Learning, Optimization, and Data Science*, pages 1–12. Springer, 2019.

[63] Pete Warden, Nov 2017.

[64] Mike Wu, Michael Hughes, Sonali Parbhoo, Maurizio Zazzi, Volker Roth, and Finale Doshi-Velez. Beyond sparsity: Tree regularization of deep models for interpretability. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 32, 2018.

[65] Weilin Xu, David Evans, and Yanjun Qi. Feature squeezing: Detecting adversarial examples in deep neural networks. *arXiv preprint arXiv:1704.01155*, 2017.

[66] Bianca Zadrozny and Charles Elkan. Obtaining calibrated probability estimates from decision trees and naive bayesian classifiers. In *Icml*, volume 1, pages 609–616. Citeseer, 2001.

[67] Bianca Zadrozny and Charles Elkan. Transforming classifier scores into accurate multiclass probability estimates. In *Proceedings of the eighth ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 694–699, 2002.

[68] Ev Zisselman and Aviv Tamar. Deep residual flow for out of distribution detection. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 13994–14003, 2020.

Checklist

1. For all authors...
   (a) Do the main claims made in the abstract and introduction accurately reflect the paper's contributions and scope? **[Yes]**
   (b) Did you describe the limitations of your work? **[Yes, please refer to Section 9.]**
   (c) Did you discuss any potential negative societal impacts of your work? **[Yes, please refer to Section 9.]**
   (d) Have you read the ethics review guidelines and ensured that your paper conforms to them? **[Yes]**
2. If you are including theoretical results...
   (a) Did you state the full set of assumptions of all theoretical results? **[No theoretical results]**
   (b) Did you include complete proofs of all theoretical results? **[No theoretical results]**
3. If you ran experiments...
   (a) Did you include the code, data, and instructions needed to reproduce the main experimental results (either in the supplemental material or as a URL)? **[Yes, we have included anonymized code in the supplement. All data used is public.]**
   (b) Did you specify all the training details (e.g., data splits, hyperparameters, how they were chosen)? **[Yes, all choices are specified in detail in the Appendix.]**
   (c) Did you report error bars (e.g., with respect to the random seed after running experiments multiple times)? **[Yes, please see the tables and captions.]**
   (d) Did you include the total amount of compute and the type of resources used (e.g., type of GPUs, internal cluster, or cloud provider)? **[Yes, please refer to Appendix.]**
4. If you are using existing assets (e.g., code, data, models) or curating/releasing new assets...
   (a) If your work uses existing assets, did you cite the creators? **[Yes, for code we borrowed from public implementations, we cite the authors and include URL references in the Appendix.]**
   (b) Did you mention the license of the assets? **[Yes, please refer to Appendix.]**
   (c) Did you include any new assets either in the supplemental material or as a URL? **[Yes, we include our source code.]**
   (d) Did you discuss whether and how consent was obtained from people whose data you're using/curating? **[Not applicable.]**
   (e) Did you discuss whether the data you are using/curating contains personally identifiable information or offensive content? **[Not applicable.]**
5. If you used crowdsourcing or conducted research with human subjects...
   (a) Did you include the full text of instructions given to participants and screenshots, if applicable? **[No human subjects experiments]**
   (b) Did you describe any potential participant risks, with links to Institutional Review Board (IRB) approvals, if applicable? **[No human subjects experiments]**
   (c) Did you include the estimated hourly wage paid to participants and the total amount spent on participant compensation? **[No human subjects experiments]**