The Thirty-Second AAAI Conference
on Artificial Intelligence (AAAI-18)

Deep Learning for Case-Based Reasoning through Prototypes: A Neural Network that Explains Its Predictions

Oscar Li,[*1] **Hao Liu,**[*3] **Chaofan Chen,**[1] **Cynthia Rudin**[1,2]
[1]Department of Computer Science, Duke University, Durham, NC, USA 27708
[2]Department of Electrical and Computer Engineering, Duke University, Durham, NC, USA 27708
[3]Kuang Yaming Honors School, Nanjing University, Nanjing, China, 210000
runliang.li@duke.edu, 141242059@smail.nju.edu.cn, {cfchen, cynthia}@cs.duke.edu

Abstract

Deep neural networks are widely used for classification. These deep models often suffer from a lack of interpretability – they are particularly difficult to understand because of their non-linear nature. As a result, neural networks are often treated as "black box" models, and in the past, have been trained purely to optimize the accuracy of predictions. In this work, we create a novel network architecture for deep learning that naturally explains its own reasoning for each prediction. This architecture contains an autoencoder and a special *prototype layer*, where each unit of that layer stores a weight vector that resembles an encoded training input. The encoder of the autoencoder allows us to do comparisons within the latent space, while the decoder allows us to visualize the learned prototypes. The training objective has four terms: an accuracy term, a term that encourages every prototype to be similar to at least one encoded input, a term that encourages every encoded input to be close to at least one prototype, and a term that encourages faithful reconstruction by the autoencoder. The distances computed in the prototype layer are used as part of the classification process. Since the prototypes are learned during training, the learned network naturally comes with explanations for each prediction, and the explanations are loyal to what the network actually computes.

1 Introduction

As machine learning algorithms have gained importance for important societal questions, interpretability (transparency) has become a key issue for whether we can trust predictions coming from these models. There have been cases where incorrect data fed into black box models have gone unnoticed, leading to unfairly long prison sentences (e.g., prisoner Glen Rodriguez was denied parole due to an incorrect COMPAS score, Wexler, 2017). In radiology, lack of transparency causes challenges to FDA approval for deep learning products. Because of these issues, "opening the black box" of neural networks has become a debated issue in the media (Citron 2016; Smith 2016; Angwin et al. 2016; Westervelt 2017). Artificial neural networks are particularly difficult to understand because their highly nonlinear functions do not naturally lend to an explanation that humans are able to process.

In this work, we create an architecture for deep learning that explains its own reasoning process. The learned models naturally come with explanations for each prediction, and the explanations are loyal to what the network actually computes. As we will discuss shortly, creating the architecture to encode its own explanations is in contrast with creating explanations for previously trained black box models, and aligns more closely with work on prototype classification and case-based reasoning.

In the past, neural networks have often been designed purely for accuracy, with *posthoc* interpretability analysis. In this case, the network architecture was chosen first, and afterwards one aims to interpret the trained model or the learned high-level features. To do the interpretability analysis requires a separate modeling effort. One problem with generating explanations posthoc is that the explanations themselves can change based on the model for the explanation. For instance, it may be easy to create multiple conflicting yet convincing explanations for how the network would classify a single object, none of which are the correct reason for why the object was classified that way. A related issue is that posthoc methods often create explanations that do not make sense to humans. This means that extra modeling is needed to ensure that the explanations are interpretable. This happens, for instance, in the Activation Maximization (AM) approach, where one aims to find an input pattern that produces a maximum model response for a quantity of interest to the user (Erhan et al. 2009). Since the images from AM are not generally interpretable (they tend to be gray), regularized optimization is used to find an interpretable high activation image (Hinton 2012; Lee et al. 2009; van den Oord, Kalchbrenner, and Kavukcuoglu 2016; Nguyen et al. 2016). When we add regularization, however, the result is a combination of what the network actually computes and the extrinsic regularization. Given that the explanations themselves come from a separate modeling process with strong priors that are not part of training, we then wonder how we can trust the explanations from the posthoc analysis. In fact there is a growing literature discussing the issues mentioned above for AM (Montavon, Samek, and Müller 2017). For images, posthoc analysis often involves visualization of layers of a neural network. For instance, an alternative to AM was provided by Zeiler and Fergus (2014), who use deconvolution as a technique to visualize what a

[*]Contributed equally

convolutional neural network (CNN) has learned. Deconvolution is one method for decoding; our method can use any type of decoder to visualize the prototypes, including deconvolution. In addition, Zeiler and Fergus (2014) try to visualize parts of images that most strongly activate a given feature map, but they do not provide an explanation for how the network reaches its decision. In contrast, we build a reasoning process into our network and do not consider posthoc analysis in this work.

There are other works that also build interpretability into deep neural networks without using posthoc analysis. Pinheiro and Collobert (2015) design a network for weakly supervised image segmentation by training a classification network that extracts important pixels which could potentially belong to an object of some class. Lei, Barzilay, and Jaakkola (2016) propose a network architecture that extracts parts of an input as a rationale and uses the rationale for predictions. Both of these works build interpretability into neural networks by extracting parts of an input and focusing on those parts for their respective tasks. Our method differs in that we use case-based reasoning instead of extractive reasoning – our model explains its predictions based on similarity to prototypical cases, rather than highlighting the most relevant parts of the input; it is possible for their ideas to be combined with ours. Tan, Sim, and Gales (2015) and Wu et al. (2016) aim to improve the interpretability of activation patterns of feature maps in deep neural networks used for speech recognition. In contrast, our model does not aim to enforce a particular structure on feature maps – it allows flexibility in feature learning but introduces a special prototype layer for decision interpretation.

Our network is a form of *prototype classifier*, where observations are classified based on their proximity to a prototype observation within the dataset. For instance, in our handwritten digit example, we can determine that an observation was classified as a "3" because the network thinks it looks like a particular prototypical "3" within the training set. If the prediction is uncertain, it would identify prototypes similar to the observation from different classes, e.g., "4" is often hard to distinguish from "9", so we would expect to see prototypes of classes 4 and 9 identified when the network is asked to classify an image of a 9.

Our work is closely aligned with other prototype classification techniques in machine learning (Bien and Tibshirani 2011; Kim, Rudin, and Shah 2014; Priebe et al. 2003; Wu and Tabak 2017). Prototype classification is a classical form of case-based reasoning (Kolodner 1992); however, because our work uses neural networks, the distance measure between prototypes and observations is measured in a flexible latent space. The fact that the latent space is adaptive is the driving force behind its high quality performance.

The word "prototype" is overloaded and has various meanings. For us, a prototype is very close or identical to an observation in the training set, and the set of prototypes is representative of the whole data set. In other contexts, a prototype is not required to be close to any one of the training examples, and could be just a convex combination of several observations. In few-shot and zero-shot learning, prototypes are points in the feature space used to represent a single class, and distance to the protoype determines how an observation is classified. For example, ProtoNets (Snell, Swersky, and Zemel 2017) utilize the mean of several embedded "support" examples as the prototype for each class in few-shot learning. Li and Wang (2017) use a generative probabilistic model to generate prototypes for zero shot learning, which are points in the feature space. In both cases, prototypes are not optimized to resemble actual observations, and are not required to be interpretable (meaning that their visualizations will not generally resemble natural images), and each class can have only one prototype.

Our deep architecture uses an autoencoder (Hinton and Salakhutdinov 2006) to create a latent low-dimensional space, and distances to prototypes are computed in that latent space. Using a latent space for distance computation enables us to find a better dissimilarity measure than L^2 on the pixel space. Other works also use latent spaces, e.g., Salakhutdinov and Hinton (2007) conduct a soft k-nearest neighbors classification on the latent space of a restricted Boltzman machine autoencoder, although not for the aim of interpretability.

2 Methodology

2.1 Network Architecture

Let $D = \{(\mathbf{x}_i, y_i)\}_{i=1}^n$ be the training dataset with $\mathbf{x}_i \in \mathbb{R}^p$ and $y_i \in \{1, ..., K\}$ for each $i \in \{1, ..., n\}$. Our model architecture consists of two components: an autoencoder (including an encoder, $f : \mathbb{R}^p \to \mathbb{R}^q$, and a decoder, $g : \mathbb{R}^q \to \mathbb{R}^p$) and a prototype classification network $h : \mathbb{R}^q \to \mathbb{R}^K$, illustrated in Figure 1. The network uses the autoencoder to reduce the dimensionality of the input and to learn useful features for prediction; then it uses the encoded input to produce a probability distribution over the K classes through the prototype classification network h. The network h is made up of three layers: a prototype layer, $p : \mathbb{R}^q \to \mathbb{R}^m$, a fully-connected layer $w : \mathbb{R}^m \to \mathbb{R}^K$, and a softmax layer, $s : \mathbb{R}^K \to \mathbb{R}^K$. The network learns m prototype vectors $\mathbf{p}_1, ..., \mathbf{p}_m \in \mathbb{R}^q$ (each corresponds to a *prototype unit* in the architecture) in the latent space. The prototype layer p computes the squared L^2 distance between the encoded input $\mathbf{z} = f(\mathbf{x}_i)$ and each of the prototype vectors:

$$p(\mathbf{z}) = \begin{bmatrix} \|\mathbf{z} - \mathbf{p}_1\|_2^2, & \|\mathbf{z} - \mathbf{p}_2\|_2^2, & \dots & \|\mathbf{z} - \mathbf{p}_m\|_2^2 \end{bmatrix}^\top . \tag{1}$$

In Figure 1, the *prototype unit* corresponding to $\mathbf{p}_j$ executes the computation $\|\mathbf{z} - \mathbf{p}_j\|_2^2$. The fully-connected layer w computes weighted sums of these distances $Wp(\mathbf{z})$, where W is a $K \times m$ weight matrix. These weighted sums are then normalized by the softmax layer s to output a probability distribution over the K classes. The k-th component of the output of the softmax layer s is defined by

$$s(\mathbf{v})_k = \frac{\exp(v_k)}{\sum_{k'=1}^K \exp(v_{k'})} \tag{2}$$

where v_k is the k-th component of the vector $\mathbf{v} = Wp(\mathbf{z}) \in \mathbb{R}^K$.

During prediction, the model outputs the class that it thinks is the most probable. In essence, our classification algorithm is distance-based on the low-dimensional learned

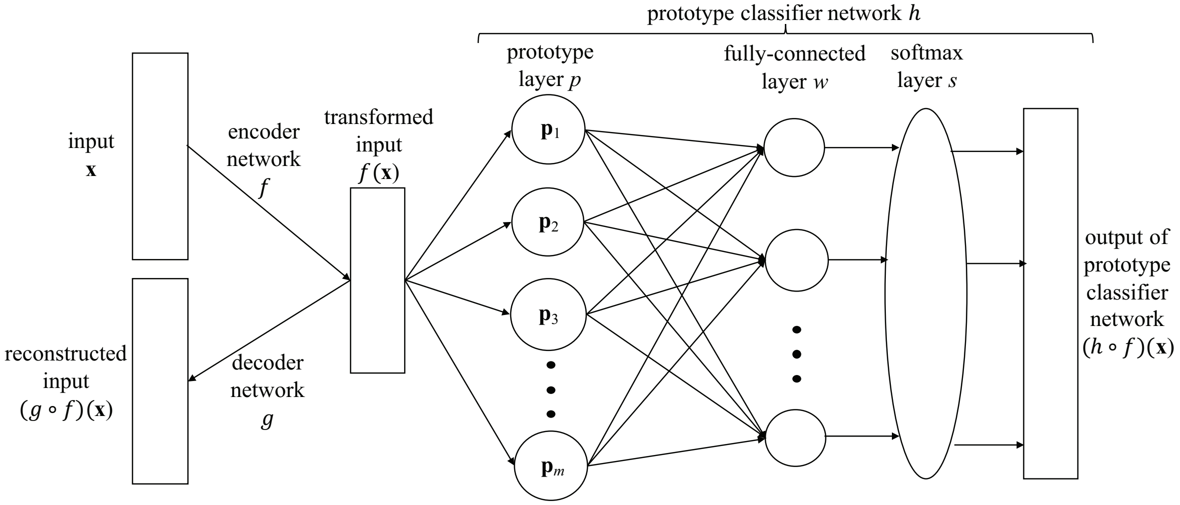


Figure 1: Network Architecture

feature space. A special case is when we use one prototype for every class (let $m = K$) and set the weight matrix of the fully-connected layer to the negative identity matrix, $W = -I_{K\times K}$ (i.e. W is not learned during training). Then the data will be predicted to be in the same class as the nearest prototype in the latent space. More realistically, we typically do not know how many prototypes should be assigned to each class, and we may want a different number of prototypes from the number of classes, i.e., $m \neq K$. In this case, we allow W to be learned by the network, and, as a result, the distances to all the prototype vectors will contribute to the probability prediction for each class.

This network architecture has at least three advantages. First, unlike traditional case-based learning methods, the new method automatically learns useful features. For image datasets, which have dimensions equal to the number of pixels, if we perform classification using the original input space or use hand-crafted feature spaces, the methods tend to perform poorly (e.g., k-nearest neighbors). Second, because the prototype vectors live in the same space as the encoded inputs, we can feed these vectors into the decoder and visualize the learned prototypes throughout the training process. This property, coupled with the case-based reasoning nature of the prototype classification network h, gives users the ability to interpret how the network reaches its predictions and visualize the prototype learning process without *posthoc* analysis. Third, when we allow the weight matrix W to be learnable, we are able to tell from the strengths of the learned weight connections which prototypes are more representative of which class.

2.2 Cost Function

The network's cost function reflects the needs for both accuracy and interpretability. In addition to the classification error, there is a (standard) term that penalizes the reconstruction error of the autoencoder. There are two new error terms that encourage the learned prototype vectors to correspond to meaningful points in the input space; in our case studies, these points are realistic images. All four terms are described mathematically below.

We use the standard cross-entropy loss for penalizing the misclassification. The cross-entropy loss on the training data D is denoted by E, and is given by

$$E(h\circ f, D) = \frac{1}{n}\sum_{i=1}^{n}\sum_{k=1}^{K} -\mathbb{1}[y_i = k]\log((h\circ f)_k(\mathbf{x}_i)) \quad (3)$$

where $(h \circ f)_k$ is the k-th component of $(h \circ f)$. We use the squared L^2 distance between the original and reconstructed input for penalizing the autoencoder's reconstruction error. The reconstruction loss, denoted by R, on the training data D is given by

$$R(g \circ f, D) = \frac{1}{n}\sum_{i=1}^{n} \|(g \circ f)(\mathbf{x}_i) - \mathbf{x}_i\|_2^2. \quad (4)$$

The two interpretability regularization terms are formulated as follows:

$$R_1(\mathbf{p}_1, ..., \mathbf{p}_m, D) = \frac{1}{m}\sum_{j=1}^{m} \min_{i\in[1,n]} \|\mathbf{p}_j - f(\mathbf{x}_i)\|_2^2, \quad (5)$$

$$R_2(\mathbf{p}_1, ..., \mathbf{p}_m, D) = \frac{1}{n}\sum_{i=1}^{n} \min_{j\in[1,m]} \|f(\mathbf{x}_i) - \mathbf{p}_j\|_2^2. \quad (6)$$

Here both terms are averages of minimum squared distances. The minimization of R_1 would require each prototype vector to be as close as possible to at least one of the training examples in the latent space. As long as we choose the decoder network to be a continuous function, we should expect two very close vectors in the latent space to be decoded to similar-looking images. Thus, R_1 will push the prototype vectors to have meaningful decodings in the pixel space. The minimization of R_2 would require every encoded training example to be as close as possible to one of the prototype vectors. This means that R_2 will cluster the training examples around prototypes in the latent space. We notice here that although R_1 and R_2 involve a minimization function that is not differentiable everywhere, these terms are differentiable almost everywhere and many modern deep learning libraries support this type of differentiation. Ideally, R_1 would take the minimum distance over the entire training set for every prototype; therefore, the gradient computation would grow linearly with the size of the training set. However, this would be impractical during optimization for a large dataset. To address this problem, we relax the minimization to be over only the random minibatch used by the Stochastic Gradient Descent (SGD) algorithm. For the other three terms, since each of them is a summation over the entire training set, it is natural to apply SGD to randomly selected batches for gradient computation.

Putting everything together, the cost function, denoted by L, on the training data D with which we train our network (f, g, h), is given by

$$\begin{aligned} L((f,g,h),D) = E(h \circ f, D) + \lambda R(g \circ f, D) \\ + \lambda_1 R_1(\mathbf{p}_1, ..., \mathbf{p}_m, D) \\ + \lambda_2 R_2(\mathbf{p}_1, ..., \mathbf{p}_m, D), \end{aligned} \quad (7)$$

where λ, λ_1, and λ_2 are real-valued hyperparameters that adjust the ratios between the terms.

3 Case Study 1: Handwritten Digits

We now begin a detailed walkthrough of applying our model to the well-known MNIST dataset. The Modified NIST Set (MNIST) is a benchmark dataset of gray-scale images of segmented and centered handwritten digits (Lecun et al. 1998). We used 55,000 training examples, 5,000 validation examples, and 10,000 testing examples, where every image is of size 28×28 pixels. We preprocess the images so that every pixel value is in $[0, 1]$. This section is organized as follows: we first introduce the architecture and the training details, then compare the performance of our network model with other noninterpretible network models (including a regular convolutional neural network), and finally visualize the learned prototypes, the weight matrix W, and how a specific image is classfied.

3.1 Architecture Details

Hinton and Salakhutdinov (2006) show that a multilayer fully connected autoencoder network can achieve good reconstruction on MNIST even when using a very low dimensional latent space. We choose a multilayer convolutional autoencoder with a symmetric architecture for the encoder and decoder to be our model's autoencoder; these types of networks tend to reduce spatial feature extraction redundancy on image data sets and learn useful hierarchical features for producing state-of-the-art classification results. Each convolutional layer consists of a convolution operation followed by a pointwise nonlinearity. We achieve downsampling in the encoder through strided convolution, and use strided deconvolution in the corresponding layer of the decoder. After passing the original image through the encoder, the network flattens the resulted feature maps into a code vector and feeds it into the prototype layer. The resulting unflattened feature maps are fed into the decoder to reconstruct the original image. To visualize a prototype vector in the pixel space, we first reshape the vector to be in the same shape as the encoder output and then feed the shaped vector (now a series of feature maps) into the decoder.

The autoencoder in our network has four convolutional layers in both the encoder and decoder. All four convolutional layers in the encoder use kernels of size 3×3, same zero padding, and stride of size 2 in the convolution stage. The filters in the corresponding layers in the encoder and decoder are not constrained to be transposes of each other. Each of the outputs of the first three layers has 32 feature maps, while the last layer has 10. Given an input image of dimension $28 \times 28 \times 1$, the shape of the encoder layers are thus: $14 \times 14 \times 32$; $7 \times 7 \times 32$; $4 \times 4 \times 32$; $2 \times 2 \times 10$, and therefore the network compresses every 784-dimensional image input to a 40-dimensional code vector ($2 \times 2 \times 10$). Every layer uses the sigmoid function $\sigma(x) = \frac{1}{1+e^{-x}}$ as the nonlinear transformation. We specifically use the sigmoid function in the last encoder layer so that the output of the encoder is restricted to the unit hypercube $(0, 1)^{40}$. This allows us to initialize 15 prototype vectors uniformly at random in that hypercube. We do not use the rectified linear unit (ReLU – Krizhevsky, Sutskever, and Hinton, 2012) in the last encoder layer because using it would make it more difficult to initialize the prototype vectors, as initial states throughout ${\mathbb{R}_{\geqslant 0}}^{40}$ would need to be explored, and the network would take longer to stabilize. We also specifically choose the sigmoid function for the last decoder layer to make the range of pixel values in the reconstructed output $(0, 1)$, roughly the same as the preprocessed image's pixel range.

3.2 Training Details

We set all the hyperparameters λ, λ_1, λ_2 to 0.05 and the learning rate to 0.0001. We minimize (7) as a whole: we do not employ a greedy layer-wise optimization for different layers of the autoencoder nor do we first train the autoencoder and then the prototype classification network.

Our goal in this work is not just to obtain reasonable accuracy, but also interpretability. We use only a few of the general techniques for improving performance in neural networks, and it is possible that using more techniques would improve accuracy. In particular, we use the data augmentation technique *elastic deformation* (Simard, Steinkraus, and Platt 2003) to improve prediction accuracy and reduce potential overfitting. The set of all elastic deformations is a superset of affine transformations. For every mini-batch of size 250 that we randomly sampled from the training set, we apply a random elastic distortion where a Gaussian filter of standard deviation equal to 4 and a scaling factor of 20 are used for the displacement field. Due to the randomness in the data augmentation process, the network sees a slightly different set of images during every epoch, which significantly reduces overfitting.

3.3 Accuracy

After training for 1500 epochs, our model achieved a classification accuracy of 99.53% on the standard MNIST training set and 99.22% on the standard MNIST test set.

To examine how the two key elements of our interpretable network (the autoencoder and prototype layer) affect predictive power, we performed a type of ablation study. In particular, we trained two classification networks that are similar to ours, but removed some key pieces in both of the networks. The first network substitutes the prototype layer with a fully-connected layer whose output is a 15-dimensional vector, the same dimension as the output from the prototype layer; the second network also removes the decoder and changes the nonlinearity to ReLU. The second network is just a regular convolutional neural network that has similar architectural complexity to LeNet 5 (Lecun et al. 1998). After training both networks using elastic deformation for 1500 epochs, we obtained test accuracies of 99.24% and 99.23% respectively. These test accuracies, along with the test accuracy of 99.2% reported by Lecun et al. (1998), are comparable to the test accuracy of 99.22% obtained using our interpretable network. This result demonstrates that changing from a traditional convolutional neural network to our interpretable network architecture does not hinder the predictive ability of the network (at least not in this case).

In general, it is not always true that accuracy needs to be sacrificed to obtain interpretability; there could be many models that are almost equally accurate. The extra terms in the cost function (and changes in architecture) encourage the model to be more interpretable among the set of approximately equally accurate models.

3.4 Visualization

Let us first discuss the quality of the autoencoder, because good performance of the autoencoder will allow us to interpret the prototypes. After training, our network's autoencoder achieved an average squared L^2 reconstruction error of 4.22 over the undeformed training set, where examples are shown in Figure 2. This reconstruction result assures us that the decoder can faithfully map the prototype vectors to the pixel space.

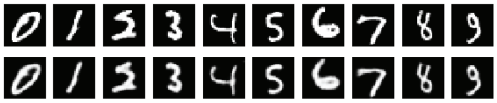


Figure 2: Some random images from the training set in the first row and their corresponding reconstructions in the second row.

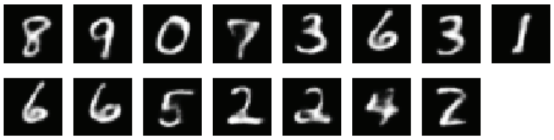


Figure 3: 15 learned MNIST prototypes visualized in pixel space.

We visualize the learned prototype vectors in Figure 3, by sending them through the decoder. The decoded prototype images are sharp-looking and mostly resemble real-life handwritten digits, owing to the interpretability terms R_1 and R_2 in the cost function. Note that there is not a one-to-one correspondence between classes and prototypes. Since we multiply the output of the prototype layer by a learnable weight matrix prior to feeding it into the softmax layer, the distances from an encoded image to each prototype have differing effects on the predicted class probabilities.

We now look at the transposed weight matrix connecting the prototype layer to the softmax layer, shown in Table 1, to see the influence of the distance to each prototype on every class. We observe that each decoded prototype is visually similar to an image of a class for which the corresponding entry in the weight matrix has a significantly negative value. We will call the class to which a decoded prototype is visually similar the *visual class* of the prototype.

The reason for such a significantly negative value can be understood as follows. The prototype layer is computing the dissimilarity between an input image and a prototype through the squared L^2 distance between their representations in the latent space. Given an image $\mathbf{x}_i$ and a prototype $\mathbf{p}_j$, if $\mathbf{x}_i$ does not belong to the visual class of $\mathbf{p}_j$, then the distance between $f(\mathbf{x}_i)$ and $\mathbf{p}_j$ will be large, so that when $\|\mathbf{p}_j - f(\mathbf{x}_i)\|_2^2$ is multiplied by the highly negative weight connection between the prototype $\mathbf{p}_j$ and its visual class, the product will also be highly negative and will therefore significantly reduce the activation of the visual class of $\mathbf{p}_j$. As a result, the image $\mathbf{x}_i$ will likely not be classified into the visual class of $\mathbf{p}_j$. Conversely, if $\mathbf{x}_i$ belongs to the visual class of $\mathbf{p}_j$, then when the small squared distance $\|\mathbf{p}_j - f(\mathbf{x}_i)\|_2^2$ is multiplied by the highly negative weight connection between $\mathbf{p}_j$ and its visual class, the product will not decrease the activation of $\mathbf{p}_j$'s visual class too much. In the end, the activations of every class that $\mathbf{x}_i$ does not belong to will be significantly reduced because of some non-similar prototype, leaving only the activation of $\mathbf{x}_i$'s actual class comparatively large. Therefore, $\mathbf{x}_i$ is correctly classified in general.

An interesting prototype learned by the network is the last prototype in Table 1. It is visually similar to an image of class 2; however, it has strong negative weight connections with class 7 and class 8 as well. Therefore, we can think of this prototype as being shared by these three classes, which means that an encoded input image that is far away from this prototype in latent space would be unlikely to be an image of 7, 8, or 2. This should not be too surprising: if we look at this decoded prototype image carefully, we can see that if we hide the tail of the digit, it would look like an image of 7; if we connect the upper-left endpoint with the lower-right endpoint, it would look like an image of 8.

Let us now look at the learned prototypes in Figure 3. The three prototypes for class 6 seem to represent different writing habits in terms of what the loop and angle of "6" looks like. The first and third 6's have their loops end at the bottom while the second 6's loop ends more on the side. The 2's show similar variation. As for the two 3's, the two prototypes correspond to different curvatures.

Let us look into the model as it produces a prediction for

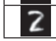
	0	1	2	3	4	5	6	7	8	9
	-0.07	7.77	1.81	0.66	4.01	2.08	3.11	4.10	-20.45	-2.34
	2.84	3.29	1.16	1.80	-1.05	4.36	4.40	-0.71	0.97	-18.10
	-25.66	4.32	-0.23	6.16	1.60	0.94	1.82	1.56	3.98	-1.77
	-1.22	1.64	3.64	4.04	0.82	0.16	2.44	-22.36	4.04	1.78
	2.72	-0.27	-0.49	-12.00	2.25	-3.14	2.49	3.96	5.72	-1.62
	-5.52	1.42	2.36	1.48	0.16	0.43	-11.12	2.41	1.43	1.25
	4.77	2.02	2.21	-13.64	3.52	-1.32	3.01	0.18	-0.56	-1.49
	0.52	-24.16	2.15	2.63	-0.09	2.25	0.71	0.59	3.06	2.00
	0.56	-1.28	1.83	-0.53	-0.98	-0.97	-10.56	4.27	1.35	4.04
	-0.18	1.68	0.88	2.60	-0.11	-3.29	-11.20	2.76	0.52	0.75
	5.98	0.64	4.77	-1.43	3.13	-17.53	1.17	1.08	-2.27	0.78
	1.53	-5.63	-8.78	0.10	1.56	3.08	0.43	-0.36	1.69	3.49
	1.71	1.49	-13.31	-0.69	-0.38	4.55	1.72	1.59	3.18	2.19
	5.06	-0.03	0.96	4.35	-21.75	4.25	1.42	-1.27	1.64	0.78
	-1.31	-0.62	-2.69	0.96	2.36	2.83	2.76	-4.82	-4.14	4.95

Table 1: Transposed weight matrix (every entry rounded off to 2 decimal places) between the prototype layer and the softmax layer. Each row represents a prototype node whose decoded image is shown in the first column. Each column represents a digit class. The most negative weight is shaded for each prototype. In general, for each prototype, its most negative weight is towards its visual class except for the prototype in the last row.

a specific image of digit 6, shown on the left of Table 2. The distances computed by the prototype layer between the encoded input image and each of the prototypes are shown below the decoded prototypes in Table 2, and the three smallest distances correspond to the three prototypes that resemble 6 after decoding. We observe here that these three distances are quite different, and the encoded input image is significantly closer to the third "6" prototype than the other two. This indicates that our model is indeed capturing the subtle differences within the same class.

After the prototype layer computes the 15-dimensional vector of distances shown in Table 2, it is multiplied by the weight matrix in Table 1, and the output is the unnormalized probability vector used as the logit for the softmax layer. The predicted probability of class 6 for this specific image is 99.99%.

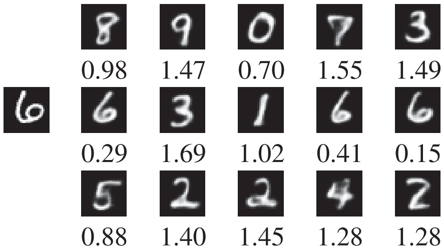

0.98 1.47 0.70 1.55 1.49
0.29 1.69 1.02 0.41 0.15
0.88 1.40 1.45 1.28 1.28

Table 2: The (rounded) distances between a test image 6 and every prototype in the latent space.

4 Case Study 2: Cars

The second dataset we use consists of rendered color images, each with $64 \times 64 \times 3$ pixels, of 3D car models with varying azimuth angles at 15° intervals, from -75° to 75° (Fidler, Dickinson, and Urtasun 2012). There are 11 views of each car and every car's class label is one of the 11 angles (see Figure 4). The dataset is split into a training set ($169 \times 11 = 1859$ images) and a test set ($14 \times 11 = 154$ images).

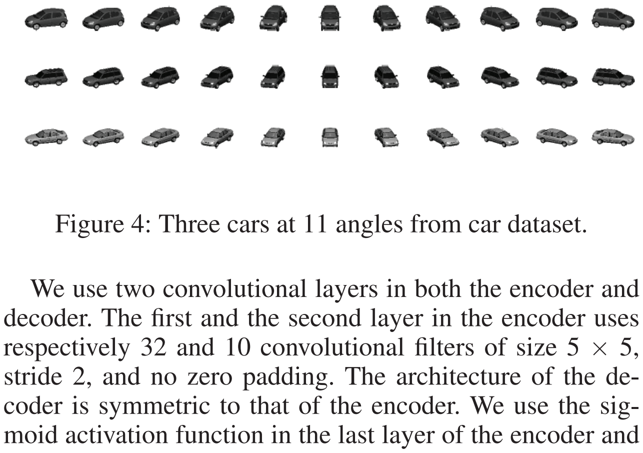


Figure 4: Three cars at 11 angles from car dataset.

We use two convolutional layers in both the encoder and decoder. The first and the second layer in the encoder uses respectively 32 and 10 convolutional filters of size 5×5, stride 2, and no zero padding. The architecture of the decoder is symmetric to that of the encoder. We use the sigmoid activation function in the last layer of the encoder and the decoder, and leaky ReLU in all other autoencoder layers. We set the number of our prototypes to be eleven, which is the same as the number of classes. Figure 5 shows the eleven decoded prototypes from our model. If we compare Figure 4 and Figure 5 in color, we can observe that the network has determined that the color of a car is not important in determining the angle, so all of the decoded prototypes are of the same "average" color. The learned weight matrix W is shown in Table 4 in the Supplementary Material. We compared our model to a network without the interpretable parts, in which we removed the decoder and replaced the prototype layer with a fully connected layer of the same size. The accuracies for these two models are shown in Table 3.

The result again illustrates that we do not sacrifice much accuracy when including the interpretability elements into the network.

	interpretable	non-interpretable
train acc	98.2%	99.8%
test acc	93.5%	94.2%

Table 3: Car dataset accuracy.

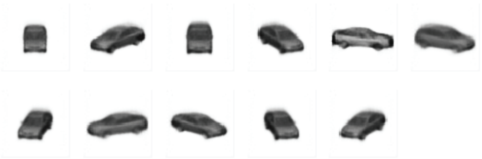


Figure 5: Decoded prototypes when we include R_1 and R_2.

We use this case study to illustrate the importance of the two interpretability terms R_1 and R_2 in our cost function. If we remove both R_1 and R_2, the decoded prototypes will not look like real images, as shown in Figure 6. If we leave out only R_1, the decoded prototypes will again not look like real observations, as shown in Figure 7. If we remove only R_2, the network chooses prototypes that do not fully represent the input space, and some of the prototypes tend to be similar to each other, as shown in Figure 8. Intuitively, R_1 pushes every prototype to be close to a training example in the latent space so that the decoded prototypes can be realistic, while R_2 forces every training example to find a close prototype in the latent space, thereby encouraging the prototypes to spread out over the entire latent space and to be distinct from each other. In other words, R_1 helps make the prototypes meaningful, and R_2 keeps the explanations faithful in forcing the network to use nearby prototypes for classification.

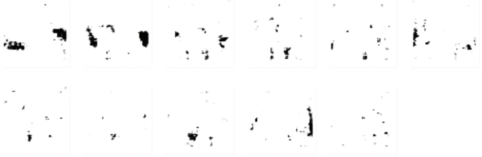


Figure 6: Decoded prototypes when we remove R_1 and R_2.

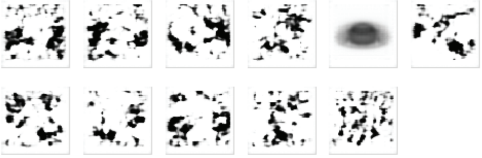


Figure 7: Decoded prototypes when we remove R_1.

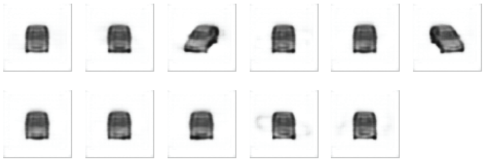


Figure 8: Decoded prototypes when we remove R_2.

5 Case Study 3: Fashion MNIST

Fashion MNIST (Xiao, Rasul, and Vollgraf 2017) is a dataset of Zalando's article images, consisting of a training set of 60,000 examples and a test set of 10,000 examples. Each example is a 28×28 grayscale image, associated with a label from 10 classes, each being a type of clothes item. The dataset shares the same image size and structure of training and testing splits as MNIST.

We ran the same model from Case Study 1 on this fashion dataset and achieved a testing accuracy of 89.95%. This result is comparable to those obtained using standard convolutional neural networks with max pooling reported on the dataset website (87.6-92.5% for networks that use similar architecture complexity as ours, Fashion-MNIST, 2017). The learned prototypes are shown in Figure 9. For each class, there is at least one prototype representing that class. The learned prototypes have fewer details (such as stripes, precence of a collar, texture) than the original images. This again shows that the model has recognized what information is important in this classification task – the contour shape of the input is more useful than its fine-grained details. The learned weight matrix W is shown in Table 5 in the Supplementary Material.

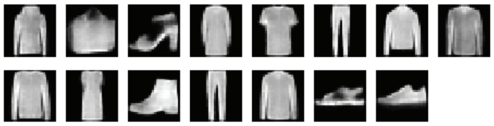


Figure 9: 15 decoded prototypes for Fashion-MNIST.

6 Discussion and Conclusion

We combine the strength of deep learning and the interpretability of case-based reasoning to make an interpretable deep neural network. The prototypes can provide useful insight into the inner workings of the network, the relationship between classes, and the important aspects of the latent space, as demonstrated here. Although our model does not provide a full solution to problems with accountability and transparency of black box decisions, it does allow us to partially trace the path of classification for a new observation.

We have noticed in our experiments that the addition of the two interpretability terms R_1 and R_2 tend to act as regularizers and help to make the network robust to overfitting. The extent to which interpretability reduces overfitting is a topic that could be explored in future work.

Supplementary Material and Code: Our supplementary material and code are available at this URL: https://github.com/OscarcarLi/PrototypeDL.

Acknowledgments: This work was sponsored in part by MIT Lincoln Laboratory.

References

Angwin, J.; Larson, J.; Mattu, S.; and Kirchner, L. 2016. Machine bias. https://www.propublica.org/article/machine-bias-risk-assessments-in-criminal-sentencing.

Bien, J., and Tibshirani, R. 2011. Prototype selection for interpretable classification. *Annals of Applied Statistics* 5(4):2403–2424.

Citron, D. 2016. (Un)fairness of risk scores in criminal sentencing. *Forbes, Tech section*.

Erhan, D.; Bengio, Y.; Courville, A.; and Vincent, P. 2009. Visualizing higher-layer features of a deep network. Technical Report 1341, University of Montreal. Also presented at the ICML 2009 Workshop on Learning Feature Hierarchies, Montreal, Canada.

Fashion-MNIST. 2017. Github repository website. https://github.com/zalandoresearch/fashion-mnist. Online; accessed September 7, 2017.

Fidler, S.; Dickinson, S.; and Urtasun, R. 2012. 3d object detection and viewpoint estimation with a deformable 3d cuboid model. In *Advances in Neural Information Processing Systems (NIPS) 25*. 611–619.

Hinton, G. E., and Salakhutdinov, R. R. 2006. Reducing the dimensionality of data with neural networks. *Science* 313(5786):504–507.

Hinton, G. E. 2012. A practical guide to training restricted boltzmann machines. In *Neural networks: Tricks of the trade*. Springer. 599–619.

Kim, B.; Rudin, C.; and Shah, J. 2014. The Bayesian case model: A generative approach for case-based reasoning and prototype classification. In *Advances in Neural Information Processing Systems (NIPS)*, 1952–1960.

Kolodner, J. 1992. An introduction to case-based reasoning. *AI Review*.

Krizhevsky, A.; Sutskever, I.; and Hinton, G. E. 2012. Imagenet classification with deep convolutional neural networks. In *Advances in Neural Information Processing Systems (NIPS) 25*. 1097–1105.

Lecun, Y.; Bottou, L.; Bengio, Y.; and Haffner, P. 1998. Gradient-based learning applied to document recognition. *Proceedings of the IEEE* 86(11):2278–2324.

Lee, H.; Grosse, R.; Ranganath, R.; and Ng, A. Y. 2009. Convolutional deep belief networks for scalable unsupervised learning of hierarchical representations. In *Proceedings of the 26th International Conference on Machine Learning (ICML)*, 609–616.

Lei, T.; Barzilay, R.; and Jaakkola, T. S. 2016. Rationalizing neural predictions. In *Proceedings of the 2016 Conference on Empirical Methods in Natural Language Processing (EMNLP)*.

Li, Y., and Wang, D. 2017. Zero-shot learning with generative latent prototype model. *CoRR* abs/1705.09474.

Montavon, G.; Samek, W.; and Müller, K. 2017. Methods for interpreting and understanding deep neural networks. *CoRR* abs/1706.07979.

Nguyen, A.; Dosovitskiy, A.; Yosinski, J.; Brox, T.; and Clune, J. 2016. Synthesizing the preferred inputs for neurons in neural networks via deep generator networks. In *Advances in Neural Information Processing Systems 29 (NIPS)*, 3387–3395.

Pinheiro, P. O., and Collobert, R. 2015. From image-level to pixel-level labeling with convolutional networks. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 1713–1721.

Priebe, C. E.; Marchette, D. J.; DeVinney, J. G.; and Socolinsky, D. A. 2003. Classification using class cover catch digraphs. *Journal of classification* 20(1):003–023.

Salakhutdinov, R., and Hinton, G. E. 2007. Learning a nonlinear embedding by preserving class neighbourhood structure. In *Proceedings of the Eleventh International Conference on Artificial Intelligence and Statistics, (AISTATS)*, 412–419.

Simard, P. Y.; Steinkraus, D.; and Platt, J. C. 2003. Best practices for convolutional neural networks applied to visual document analysis. In *Proceedings of the Seventh International Conference on Document Analysis and Recognition (ICDAR), Volume 2*.

Smith, M. 2016. In wisconsin, a backlash against using data to foretell defendants' futures. *New York Times*.

Snell, J.; Swersky, K.; and Zemel, R. S. 2017. Prototypical networks for few-shot learning. *CoRR* abs/1703.05175.

Tan, S.; Sim, K. C.; and Gales, M. 2015. Improving the interpretability of deep neural networks with stimulated learning. In *Proceedings of 2015 IEEE Workshop on Automatic Speech Recognition and Understanding (ASRU)*, 617–623.

van den Oord, A.; Kalchbrenner, N.; and Kavukcuoglu, K. 2016. Pixel recurrent neural networks. In *Proceedings of the 33nd International Conference on Machine Learning, (ICML)*, 1747–1756.

Westervelt, E. 2017. Did a bail reform algorithm contribute to this San Francisco man's murder? *National Public Radio, Law*.

Wexler, R. 2017. When a computer program keeps you in jail: How computers are harming criminal justice. *New York Times*.

Wu, C., and Tabak, E. G. 2017. Prototypal analysis and prototypal regression. *CoRR* abs/1701.08916.

Wu, C.; Karanasou, P.; Gales, M. J.; and Sim, K. C. 2016. Stimulated deep neural network for speech recognition. In *Interspeech*, 400–404.

Xiao, H.; Rasul, K.; and Vollgraf, R. 2017. Fashion-mnist: a novel image dataset for benchmarking machine learning algorithms. *CoRR* abs/1708.07747.

Zeiler, M. D., and Fergus, R. 2014. Visualizing and understanding convolutional networks. In *Proceedings of the European Conference on Computer Vision (ECCV)*, 818–833.