Transparency in Deep Learning Using Hierarchical Prototypes

Christiaan van der Vlist
christiaan.vandervlist@student.uva.nl

Albert Harkema
albert.harkema@student.uva.nl

Anna Langedijk
annalangedijk@gmail.com

Hinrik Snær Guðmundsson
hinriksnaer@gmail.com

ABSTRACT

Neural Networks are highly effective and widely used algorithms for classification. However, it is often difficult to interpret why these models make certain predictions. Earlier work uses the notion of prototype layers to allow for easier, visual interpretation of the network's predictions. We extend this prototype model with a hierarchical prototype model, introducing sub- and superprototype layers. These layers enable the model to visualize a number of superprototypes equal to the number of superclasses while simultaneously allowing the model to infer and visualize the latent subclasses present in the data. This extension does not sacrifice accuracy, achieving 99% accuracy on an MNIST classification task. We have found that this model is indeed able to find and visualize general superprototypes and more specific subprototypes. Ultimately, we argue that this model can also be used in the pipeline of debiasing data and subsequent predictions.

1 INTRODUCTION

The importance of transparency in Machine Learning is rapidly increasing. As Machine Learning algorithms grow in prominence in society, being able to explain the decision making process of these algorithms in an intelligible manner has become more important. This is true both from a legal perspective (such as with the GDPR[1]) as from a human perspective – interpretability engenders trust in users, even when it makes mistakes [5], which is especially important when people rely on the algorithm to keep them healthy and safe, as is the case with medical AI or self-driving cars. Furthermore, an explainable algorithm allows us to better detect biases and spurious relationships present in the training data, for instance due to data leakage [8]. However, most modern Machine Learning algorithms rely on artificial neural networks which use nonlinearities to capture hidden relations between inputs and outputs. This makes explaining model output in a satisfactory manner a difficult task [1].

A number of different approaches have been proposed to address the inherent lack of transparency prevalent in widely used neural network architectures. One example of those approaches is the explanation technique LIME [18], which provides local (i.e. for a single input) explanations of what features are important for the classification. When applied to images, for example, it highlights which pixels weighed heavily in its decision-making. This technique is binary – a feature was either a relevant factor or not. Other examples of methods that focus on explanation through feature highlighting have been recently proposed by Lundburg & Lee [15]. One drawback of the methods of improving explanation discussed so far is that they require new models to be trained in order to make explanation possible. A method that does not have this issue is Integrated Gradients [20], which also does feature highlighting but can be applied to existing models.

A more specific method of explaining why a model classifies inputs in a certain way is by visualizing what a model considers to be an archetypal example of a given class. Activation Maximization (AM) [6] is such a method: for each class it provides a hypothetical input that would produce the highest activation for that class. However, from a human point of view, such inputs tend to be dissimilar to actual inputs from that class [6]. This goes against our intuition that model explanations should be interpretable as well as accurate. To resolve this dissimilarity, Li et al. [12] propose a representation learning architecture for a *prototype classifier*. This architecture classifies by measuring distance between input and prototypes, which are defined as points that represent a class in a latent space. These prototypes are learned through regularization such that they are close to encoded inputs corresponding to that class. Li et al. simultaneously train this classifier and an autoencoder [7] that transforms inputs to the latent space of the prototypes. After training has finished, the autoencoder can be used to visualize the prototypes by decoding them.

In this paper, we first of all attempt to reproduce the results attained by the architecture introduced by Li et al. [12]. Secondly, we extend the architecture with hierarchical prototypes. This hierarchy is composed of two levels: superprototypes and subprototypes. Subprototypes are close to encoded inputs (similar to prototypes in the original architecture), while superprototypes act as descriptors for the subprototype clusters in the latent space. For example, when training on the MNIST dataset, subprototypes can find different ways of writing the same digit (as prototypes do in [12]), while superprototypes come to represent the average way numbers are written. By visualizing the sub- and superprototypes an encoded input is closest to, we gain an extra layer of interpretability for the classification process. When the number of superprototypes is equal to the number of classes, a fixed activation pattern can be used to force the detection of all classes. The subprototypes will in this case take care of intraclass variation. The network automatically learns which classes need more subprototypes. This solves an issue in the original architecture [12], namely, that even when the number of prototypes exceeds the number of classes, there are cases where not all classes are represented by a prototype (for an example, see Figure 6 in the Appendix).

In the next section, we will briefly explain the original, nonhierarchical architecture. This is followed by a detailed description of our hierarchical prototype network. In Section 3, we describe and justify our choice of hyperparameters. Results for both reproduction and the new architecture are examined in Section 4, followed by a

[1]As described in https://www.cyberadviserblog.com/2018/01/analysis-article-29-working-party-guidelines-on-automated-decision-making-under-gdpr (retrieved on January 16, 2020)

discussion of both the benefits and shortcomings of (hierarchical) prototype networks. Finally, Section 6 discusses some broader implications within the field of transparancy and fairness in artificial intelligence.

2 METHOD

We kept most of the original architecture from [12] intact, only adding to it to work with hierarchical prototypes. The inputs are encoded and their distances to the subprototypes and superprototypes are calculated for two separate classification tasks, one for sub- and superprototypes respectively. Subprototypes are trained in exactly the same way as prototypes in the original architecture – that is, by forcing them to be similar to encoded input through regularization. Likewise, superprototypes are forced to be similar to subprototypes.

2.1 Original architecture

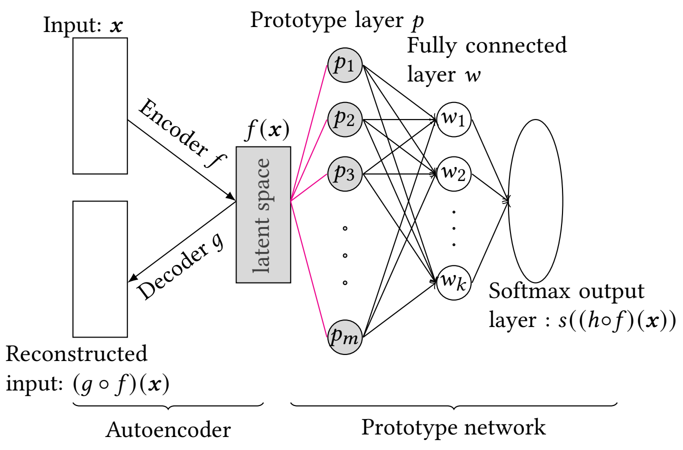


Figure 1: Architecture of the standard prototype model
Gray nodes represent images encoded in latent space. The purple lines indicate the similarity between the input images and prototypes.

The original architecture from [12] is comprised of two parts: an autoencoder [7] and a prototype network. These are shown schematically in Figure 1. The autoencoder's encoder, f, takes a p-dimensional input $\mathbf{x}$ and transforms it to a q-dimensional latent space. The decoder, g, takes a q-dimensional input and transforms it back into p-dimensional space.

The prototype network, h, takes a q-dimensional input and outputs K probabilities, one for each class. This network itself consists of a prototype layer $p : \mathbb{R}^q \rightarrow \mathbb{R}^m$. It is followed by a fully-connected linear layer $w : \mathbb{R}^m \rightarrow \mathbb{R}^K$ with learnable weights (unless $m = K$, see further down). Finally, there is a softmax layer $s : \mathbb{R}^K \rightarrow \mathbb{R}^K$. The prototype layer contains m learnable q-dimensional prototype vectors. When given an input, this layer calculates the squared L^2 distance between a single input and each of the prototype vectors. These m distances are then pushed through w and s to get the classification. In case $m = K$, w is a negative identity matrix $-I_{K \times K}$ instead.

The loss is comprised of a sum of four terms: Cross-entropy for the classification, the reconstruction error of the autoencoder, and two interpretability terms R_1 and R_2. R_1 is defined as the mean of the minimum of distances between some prototype and the current input images. It forces prototypes to be similar to at least one data point in the batch. R_2 is defined as the mean of the minimum of distances between some datapoint in latent space and the prototypes. This term forces encoded images to be close to at least one prototype.

The final loss function is as follows:

$$L((f, g, h), D) = \lambda_{\text{class}} E(h \circ f, D) + \lambda_R R(g \circ f, D) + \lambda_1 R_1 + \lambda_2 R_2 \quad (1)$$

Where λ_{class}, λ_R, λ_{R_1} and λ_{R_2} are hyperparameters that can be used to adjust the importance of the different terms.

2.2 Hierarchical prototype network

In the hierarchical version of the prototype network, the single prototype layer is replaced by a subprototype layer $p : \mathbb{R}^q \rightarrow \mathbb{R}^m$ of length $m > K$ and a superprototype layer $P : \mathbb{R}^q \rightarrow \mathbb{R}^k$, where k is the number of superprototypes. These two prototype layers both have their own classification output, denoted h_{sub} and h_{sup}. When k is equal to the number of classes K, no intermediate linear layer is learned. Instead, its output is fixed to the negative identity matrix as before. The new layered prototype network is visualized in Figure 2.

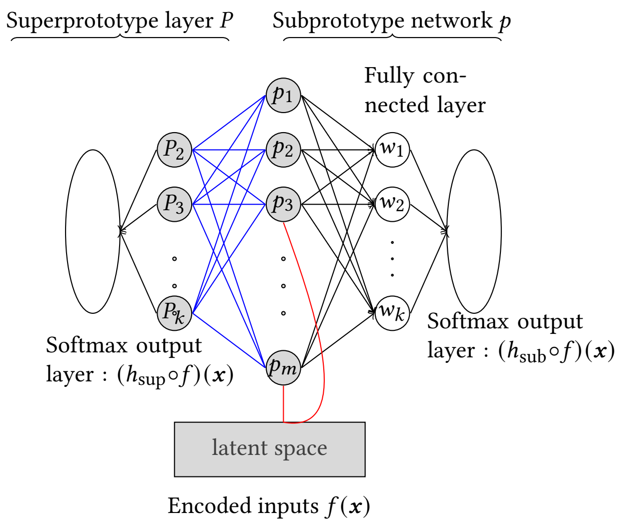

Superprototype layer P
Subprototype network p
Fully connected layer
P_2
P_3
P_k
p_1
p_2
p_3
p_m
w_1
w_2
w_k
Softmax output layer : $(h_{\text{sup}} \circ f)(\boldsymbol{x})$
Softmax output layer : $(h_{\text{sub}} \circ f)(\boldsymbol{x})$
latent space
Encoded inputs $f(\boldsymbol{x})$

Figure 2: Architecture for a subprototype network
The autoencoder is not visualized to avoid reduncancy. Gray nodes represent images encoded in latent space. Red lines indicate the similarity between the input images and the subprototypes. In contrast, blue lines indicate the similarity between the superprototypes and subprototypes. These colors also hold in Equation 2. Notice that there are two classifiers, as opposed to the single classifier in Figure 1.

We adjust the loss function to deal with the new layer. Error terms R_1 and R_2 force the subprototypes to be similar to at least one input data point and vice versa, as before. Two new terms, R_3 and R_4 are introduced to enforce similarity for a superprototype to at least one subprototype and vice versa. This highlights the hierarchicality of our network. The final loss term is:

$$\begin{aligned} L((f,g,h),D) = {} & \lambda_{sub} E_{sub}(h_{\text{sup}} \circ f, D) \\ & + \lambda_{sup} E_{sup}(h_{\text{sup}} \circ f, D) + \lambda_R R(g \circ f, D) \\ & + \lambda_1 R_1 + \lambda_2 R_2 + \lambda_3 R_3 + \lambda_4 R_4 \end{aligned} \tag{2}$$

Where R_3 and R_4 describe the connection between subprototypes and superprototypes, as R_1 and R_2 do in the original paper:

$$R_3(\boldsymbol{P}_1, \dots, \boldsymbol{P}_K, \boldsymbol{p}_1, \dots, \boldsymbol{p}_m) = \frac{1}{K} \sum_{k=1}^{K} \min_{j \in [1,m]} ||\boldsymbol{P}_k - \boldsymbol{p}_j||_2^2 \tag{3}$$

$$R_4(\boldsymbol{P}_1, \dots, \boldsymbol{P}_K, \boldsymbol{p}_1, \dots, \boldsymbol{p}_m) = \frac{1}{m} \sum_{j=1}^{m} \min_{k \in [1,K]} ||\boldsymbol{P}_k - \boldsymbol{p}_j||_2^2 \tag{4}$$

Again, all λ's are hyperparameters that can be used for adjusting the importance of the various error terms.

Furthermore, by using the superprototypes to classify inputs and taking the cross-entropy error of this task into account in the loss function, we ensure that the superprototypes remain useful for the main task – that is, classification.

3 EXPERIMENTAL SETUP

To test the original architecture, we apply it on the MNIST dataset to see how it performs at classifying handwritten digits in comparison to [12], using a near-identical[2] experimental setup. Thereafter, we apply our hierarchical prototype network to MNIST to see whether it will produce specialized subprototypes and generalized superprototypes.

Because our first goal is to reproduce the results attained by Li et al. [12], we used the same autoencoder and hyperparameters that are used in this paper. The encoder consists of four convolutional layers of shapes $14 \times 14 \times 32$, $7 \times 7 \times 32$, $4 \times 4 \times 32$ and $2 \times 2 \times 10$ respectively. The latent space is thus of size $2 \times 2 \times 10 = 40$. The decoder consists of four identically sized convolutional layers in the opposite direction. The activation function for all layers of the autoencoder is the sigmoid function.

Next, the authors set a number of hyperparameters: the number of prototype vectors, the size of the latent space and four hyperparameters λ_{class}, λ_{R}, λ_1 and λ_2 for adjusting the ratio between the reconstruction error, the R_1 term and the R_2 term respectively. To allow for reproducibility, we used values $\lambda_{\text{class}} = 20$ and $\lambda_{\text{R}} = \lambda_1 = \lambda_2 = 1$ as in [12].

Adjusting the λ hyperparameters of our loss function allows us to balance the individual loss terms. We set $\lambda_{\text{sub}} = \lambda_{\text{sup}} = 20$, similar to the nonhierarchical approach. By selecting values for λ_{sub} and λ_{sup} that are proportionally higher than our other regularization terms, we enforce that our cross-entropy loss terms are of the same magnitude as the other loss terms. When the terms are of different orders of magnitude, in this case the cross-entropy loss will be

[2]Since the original supplementary material did not include a random seed, it was not possible to use the exact same setup.

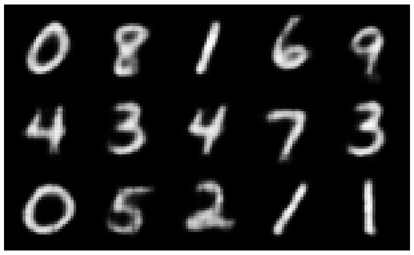


Figure 3: Original prototypes after 1500 epochs

smaller, the model will tend to optimize for the loss terms with the largest value, sacrificing the smaller loss terms [4]. Furthermore, we set $\lambda_1 = \lambda_2 = \lambda_3 = \lambda_4 = 1$. Assigning the same value to each of these terms ensures that prototypes will resemble our input data. Finally, we set our λ_R to 1 to ensure good reconstruction at a minimum loss.

For both architectures, we used the Adam optimizer [9] and a batch size of 250 for training. We selected a low learning rate of 0.0001 and trained over 1500 epochs to allow our prototypes to form gradually. The choice of batch size and number of epochs is done in accordance with [12]. By opting for a lower learning rate instead of a higher one, the set of prototypes is more likely to find representations for all classes in the dataset, instead of being heavily influenced by initialization and the subset of data that our model encounters during the early stages of training. To maintain reasonable accuracy while still achieving interpretability, we apply the data augmentation technique *elastic deformation* [19] to every batch, following the same training setup as described in [12] where a Gaussian filter of standard deviation 4 and scaling factor of 20 is used for the displacement field.

4 RESULTS

We ran two different versions of the model: the first is the same as that of Li et al. [12] so that we can report on the original paper's reproducibility, the second is the version that runs the newly proposed hierarchical version of this model. We will begin by discussing the reproduced results of the original paper. After that, we will discuss the results of the hierarchical model.

4.1 Reproduced Results

After training our implementation of the original prototype model for 1500 epochs with 15 prototypes and a learning rate of 0.0001 as done by Li et al. [12], we achieved an accuracy of 98.8%. This corroborates the results of the original paper. Moreover, it shows that the addition of a prototype layer does not significantly reduce the accuracy that the model is able to achieve [12].

To visualize the prototypes, they can be passed through the decoder. These are shown in Figure 3. As in the original paper [12], the prototypes resemble authentic handwritten digits. This further confirms that we can reproduce the results of the original paper, namely that we can generate meaningful and interpretable prototypes.

4.2 Hierarchical Results

For the hierarchical model, after 1500 epochs, the superprototype and the subprototype classification network respectively achieved a 98.9% accuracy and a 99.0% accuracy on the MNIST test set. These accuracies are nearly identical to the ones found in our implementation of the original prototype paper and to the accuracy reported by Li et al. [12]. This points towards our hierarchical prototype model being equally as capable of making accurate predictions on the MNIST as the other two models are.

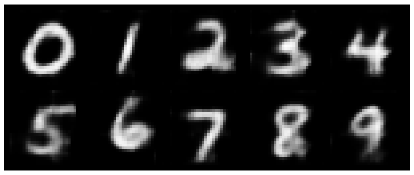


Figure 4: Superprototypes after 1500 epochs with weights set to negative identity.

Next, we visualized the learned superprototypes as shown in Figure 4. By fixing the weight matrix to the negative identity matrix, we ensure that the model learns one prototype that resembles a distinct class within our data for every class [12]. Because of the fixed weight matrix, the superprototypes shown in Figure 4 each represent a different digit corresponding to our expectation. The learned superprototypes are crisp and easily interpretable representations of handwritten digits.

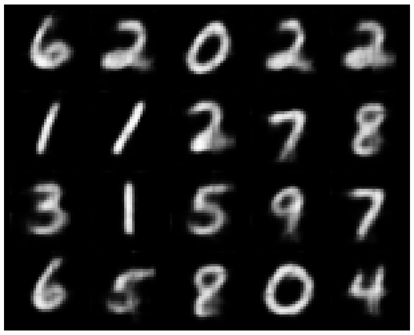


Figure 5: 20 subprototypes after 1500 epochs with learnable weights

We trained the superprototypes simultaneously with the subprototypes. Again, we visualized these subprototypes as shown in Figure 5. Because the weight matrix connecting the 20 neuron subprototype layer to the 10 class output layer has learnable parameters, the network itself learns how many subprototypes it requires for every class. Consequently, some interesting subprototypes show up. The model finds four different visualisations of a 2. One explanation is that it learns to model a number of different ways a 2 can be written, whether it is more straight or has a loop connecting the bottom part to the main diagonal. For the 1s, our model learns 3 different representations, mostly variations in how much they are tilted to the right. The variations in the 6s follow a similar pattern.

This can also be seen in the weights connecting the subprototypes to the output layer, as shown in Table 1 in the appendix. The larger the negative weight, the more it activates for that particular class which corresponds to the MNIST digit [12]. From Table 1 it becomes clear that most weights have the highest activation for the corresponding digit. However, some weights show that the model is not as sure about every subprototype. For instance, the subprototype for a 2 in the fifh row of Table 1 is uncertain about the weights, with the model deciding between a 1, 4, 5 and the correct 2. This may be because those digits all activate in the same area of an image as does the 2. Moreover, we also have the 2 in the eighth row of Table 1 which the model mistakes for a 7. Again, this makes sense because this prototype resembles both a 2 and a 7. This may be because these two digits both consist of a diagonal and a (near) horizontal line at the top.

5 DISCUSSION

The original paper [12] introduced the notion of prototypes to make neural networks more transparent. Indeed, we have seen that these prototypes converge to representations that explain why a neural network makes certain predictions. In this paper we ran the model proposed by the original paper and used the exact same hyperparameters. Subsequently, we extended the architecture of the original paper by introducing the hierarchical concept of subprototypes. This extension has a number of advantages, namely, adding a notion of hierarchy to the network allows for more explainability because we can also discover interclass and intraclass variability of the uncovered classes. This may help researchers further explain why the neural network makes certain predictions.

Furthermore, having multiple prototype layers subdivided in super- and subprototype layers gives our network the flexibility to determine by itself, given the fixed number of total super- and subprototypes, how many subclasses should be discerned from the dataset. This flexibility is constrained in the original paper in which the learned prototypes are forced to be both more general and more specific at the same time [12]. Additionally, a potential problem with the original prototype network is that it is not guaranteed to learn a representation for each class, as shown in Figure 6 in the Appendix. There, the model only learns a single hybrid representation of a 3 and an 8. Our main goal, however, is to provide clear prototypes that allow for better interpretation of why the model makes certain predictions. By fixing the weights for the superprototypes to the negative identity matrix we can ensure that we will at least find a representation for each class. In some additional experiments, we downsampled a single class of digits (9) during training. For this imbalanced data, the nonhierarchical model did not learn a prototype for 9, as seen in Figure 7. However, as shown in Figure 8, the negative identity matrix forces a hierarchical model to learn a prototype for the underrepresented class.

In contrast to the fixed superprototype layer, having learnable weights for the linear layer connecting the subprototype layer to its corresponding softmax layer, we can still learn the intraclass variability in connection to our superprototypes. With this we can effectively counteract the problem of not finding representations for each class, reducing potential adverse effects on the predictive and interpretative abilities of the model.

However, we also uncovered a number of disadvantages. Firstly, our proposed architecture comes with a number of extra hyperparameters related to the separate loss terms in our loss function that need to be tuned. Generally, there are no efficient ways of performing this optimization [2], so adding more hyperparameters comes at the cost of significantly increased computation time. This added computational cost can be alleviated by using random search instead of full grid search [2].

Secondly, the number of prototypes remains a hyperparameter to be tuned. The addition of subprototypes introduces a plethora of new combinations of the numbers of prototypes per class to be tuned in order to find the right number of (sub)prototypes. Expediting the search for the best number of (sub)prototypes remains an open research question.

Thirdly, the prototypes add explainability, but do not allow for any further post hoc analysis. This entails that we can not simply use this model on previously created models currently used for decision making to explain its predictions. However, if models such as ours that add explainability as an integral part of their architecture become more prevalent, networks used for predictions may in the future always include a component that provides this explainability, counteracting the need for post hoc analysis.

Fourthly, the prototypes do not give a feature by feature basis on which one could base an analysis. The prototypes themselves show what a typical class looks like but do not necessarily show what specific features it looks for in order to perform the classification, because the distance to a prototype does not model lower level features. Other architectures have been proposed to incorporate these other features[16]. Prototypes may provide sufficient transparency for visual tasks such as image classification. They may however be insufficient when interpretability is heavily dependent on low level features.

Finally, the distance measure used to define the distance between the input images and the decoded images is the L^2 norm. Although this similarity measure is widely used, it is not very suitable for image data because small changes to the image such as translations, may result in a very large error even though the result is good to human eyes [11]. In turn, poor encodings and decodings could potentially impact the prototype layer's ability to learn meaningful representations from the data. Research is being done into finding different similarity measures [11]. However, discussing these measures is beyond the scope of this paper.

6 BROADER IMPLICATIONS

Our primary goal is to provide an architecture that is both accurate and interpretable, which places our architecture within the domains of Transparency and Accountability. Another domain that has been getting more attention recently, however, is that of Fairness. Algorithms can be considered fair if they do not give disparate treatment to a disadvantaged group. Fairness criteria that are supposed to effect this are, according to Liu et al. [13], "clearly intended to protect the disadvantaged group", even though they are not always succesful in improving the well-being of the this group [13].

An important part of fairness is determining which algorithms are actually fair and which are biased. Bringing these biases to light in existing models has led, for example, to reduced disparity in hiring processes [17]. Exposing biases is something that interpretable models aid in. In the case of the architecture of Li et al. [12] and our own architecture, for example, prototypes provide archetypical examples of data classes. When applied to, say, hiring data, prototypes show if the archetypical person that gets hired tends to be of a certain gender, race, or other sensitive category. If a model is found to be biased, appropriate action can be taken.

On the other hand, our architecture is incapable of downplaying or removing certain information from the input. This is necessary if the architecture is to be used for fair classification that does not rely on sensitive information in the input. Simply removing features explicitly related to this sensitive information is not enough, since the information can be inferred from other correlated features [3].

Both interclass and intraclass differences may be used to check for biases in the model or the data it is trained on. A certain class may be underrepresented when it has few subprototypes, or may have a much higher variance in its subprototypes. Although these variations are not related to fairness when it comes to MNIST digits, they may become relevant when a prototype network is trained on data containing sensitive information.

A variant of the variational autoencoder architecture [10] proposed by Louizos et al. [14] is capable of transforming input to a latent space in such a way that information can be factored out of the latent representation, while maintaining as much information as possible that is useful for classification. One potential way of making the architecture we propose in this paper fairer is by replacing the autoencoder with a Variational Fair Autoencoder. The prototypes would also need to be changed from points to distributions in the latent space. We believe that this will result in prototypes that are unbiased toward the sensitive information.

7 CONCLUSION

In this paper we have discussed a new neural network architecture proposed by Li et al. [12] in light of increased calls for more transparency in deep learning models. This architecture involves the learning of prototypes to allow for easier interpretation of predictions. We have succeeded in reproducing the results attained by Li et al. [12] and therefore we award the results reproduced badge[3] to the paper. Furthermore, we have extended their architecture by replacing the prototypes with hierarchical prototypes[4]. These hierarchical prototypes add another layer of explainability to the model, with superprototypes showing the "class average" while subprototypes capture intraclass differences. In doing so, we can visualize several layers consisting of more intricate features allowing for better interpretation of why our network is making certain predictions. We have shown that we can enforce our model to learn representations for all superclasses while simultaneously learning the subclasses without sacrificing our model's predictive abilities. Ultimately, we have also demonstrated ways in which our model's increased interpretability may help in other domains such as fairness, where our model can be used to uncover potentially biased predictions in order to counteract these unwanted biases.

[3]https://www.acm.org/publications/policies/artifact-review-badging
[4]The code that was used for reproducing the original paper and our extension can be found at: https://github.com/Sasafrass/PrototypeDL

REFERENCES

[1] J. M. Benítez, J. L. Castro, and I. Requena. 1997. Are Artificial Neural Networks Black Boxes? *IEEE Transactions on Neural Networks* (1997).

[2] J. Bergstra and Y. Bengio. 2012. Random search for hyper-parameter optimization. *Journal of Machine Learning Research* 13 (2012), 281–305.

[3] T. Calders and I. Žliobaitė. 2013. *Discrimination and Privacy in the Information Society. Data Mining and Profiling in Large Databases.* Springer, Chapter Why Unbiased Computational Processes Can Lead to Discriminative Decision Procedures, 43–57.

[4] F. Chollet. 2017. *Deep Learning with Python.* Manning, Chapter Advanced Deep-Learning Best Practices, 241.

[5] M. T. Dzindolet, S. A. Peterson, R. A. Pomranky, L. G. Pierce, and H. P. Beck. 2003. The role of trust in automation reliance. *International Journal of Human-Computer Studies* (2003).

[6] D. Erhan, Y. Bengio, A. Courville, and P. Vincent. 2009. Visualizing higher-layer features of a deep network. *Technical Report 1341* (2009).

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

[8] S. Kaufman, S. Rosset, and C. Perlich. 2011. Leakage in data mining: formulation, detection, and avoidance. In *Knowledge Discovery and Data Mining (KDD).*

[9] D. P. Kingma and J. Ba. 2014. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980.* (2014).

[10] D. P. Kingma and M. Welling. 2013. Auto-Encoding Variational Bayes. *arXiv:1312.6114* (2013).

[11] Sønderby S. K. Larochelle H. Larsen, A. B. L. and O. Winther. 2015. Autoencoding beyond pixels using a learned similarity metric. *arXiv:1512.09300.* (2015).

[12] O. Li, H. Liu, C. Chen, and C. Rudin. 2018. Deep learning for case-based reasoning through prototypes: A neural network that explains its predictions. In *Thirty-Second AAAI Conference on Artificial Intelligence.*

[13] T. L. Liu, S. Dean, E. Rolf, M. Simchowitz, and M. Hardt. 2018. Delayed Impact of Fair Machine Learning. In *Proceedings of the 35th International Conference on Machine Learning.*

[14] C. Louizos, K. Swersky, Y. Li, M. Welling, and R. Zemel. 2016. The Variational Fair Autoencoder. *ICLR 2016* (2016).

[15] S. M. Lundberg and S. Lee. 2017. A Unified Approach to Interpreting Model Predictions. In *NeurIPS 2017.*

[16] David Alvarez Melis and Tommi Jaakkola. 2018. Towards robust interpretability with self-explaining neural networks. In *Advances in Neural Information Processing Systems.* 7775–7784.

[17] I. D. Raji and J. Buolamwini. 2019. Actionable Auditing, Investigating the Impact of Publicly Naming Biased Performance Results of Commercial AI Products. In *AIES '19: Proceedings of the 2019 AAAI/ACM Conference on AI, Ethics, and Society.* 429–435.

[18] M. T. Ribeiro, S. Singh, and C. Guestrin. 2016. 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.* 1135—-1144.

[19] P. Y. Simard, D. Steinkraus, and J. C. Platt. 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)*, Vol. 2.

[20] M. Sundararajan, A. Taly, and Q. Yan. 2017. Axiomatic Attribution for Deep Networks. In *ICML 2017.*

APPENDIX

Contributions

Reproduction was a fully combined effort on which we worked almost exclusively together. One day Anna was able to finish the reproduction except for one minor bug which was resolved the next day by Albert. For the hierarchical architecture we first did a lot of theorycrafting together for potential architectures. Chris came up with the idea of the architecture that we are currently using, after Hinrik and Anna had worked hard on an implementation that turned out not to be a fruitful endeavour. When the current architecture seemed not to work out, it was due to Chris and Hinrik's perseverance that they found out about a very elusive bug. Later that evening, Albert was able to resolve the bug again. Subsequently, everybody contributed to the paper. Anna did a lot of great work on the LaTeXlayouts and the images of the architectures. Together we wrote and revised all of the subsections in the paper and we have made sure throughout the entire project to distribute the work equally. Last full revisions to the paper were done by Albert and Chris while Anna and Hinrik made sure to refactor the code.

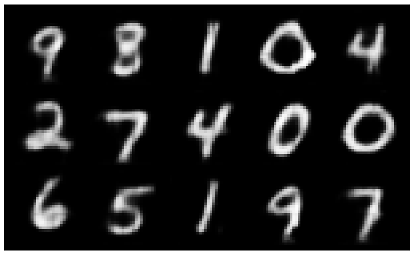


Figure 6: Reproduced prototypes with seed set to 9. Notice that there is no prototype learned for class 3.

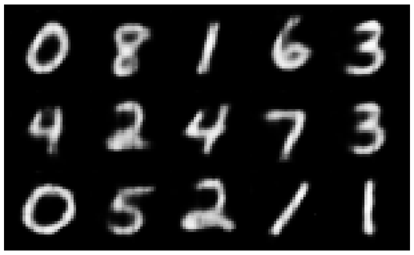


Figure 7: When downsampling the class 9, a nonhierarchical model with default parameters does not learn a prototype for 9.

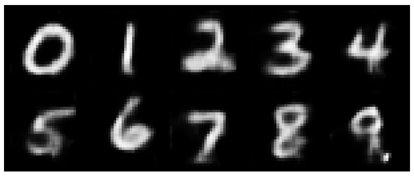


Figure 8: When downsampling the class 9, a hierarchical model with default parameters does learn a decent prototype for 9.

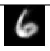
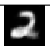
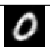
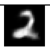
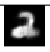
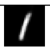
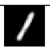
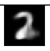
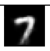
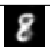
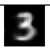
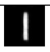
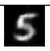
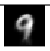
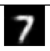
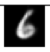
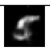
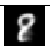
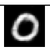
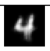
		0	1	2	3	4	5	6	7	8	9
1		-0.016	0.470	0.413	-0.069	-0.155	0.311	-0.859	-0.519	0.035	0.356
2		0.293	-0.118	-0.708	0.183	-0.538	0.206	0.491	0.046	-0.068	-0.417
3		-1.443	0.313	-0.028	-0.107	0.302	0.086	-0.278	0.504	-0.121	-0.140
4		0.267	-0.306	-0.820	-0.321	0.227	0.122	-0.094	0.762	0.657	-0.604
5		0.106	-0.266	-0.193	0.322	-0.262	-0.206	0.027	-0.040	-0.012	0.678
6		-0.189	-0.509	-0.047	0.213	0.412	0.122	-0.002	-0.327	0.276	0.433
7		0.152	-0.939	0.109	-0.334	-0.376	0.067	0.507	0.221	-0.156	0.473
8		0.019	-0.044	-0.261	0.441	0.392	-0.208	-0.382	-0.399	-0.148	0.329
9		-0.100	-0.117	0.290	-0.313	0.237	-0.300	-0.001	-0.576	0.533	-0.204
10		0.082	0.458	0.147	-0.351	0.495	-0.223	0.109	0.021	-0.757	-0.128
11		0.113	-0.025	0.480	-2.275	0.023	0.237	0.290	0.086	0.255	0.390
12		0.227	-0.722	0.142	0.234	0.212	0.411	0.173	0.496	0.343	-0.622
13		0.070	0.275	0.208	0.112	0.183	-0.779	-0.036	0.303	0.387	-0.331
14		0.339	0.022	0.083	0.231	-0.452	0.121	0.023	0.050	0.096	-1.713
15		0.139	0.209	0.008	0.302	-0.166	0.090	0.237	-1.548	-0.515	0.424
16		0.211	0.112	0.189	-0.035	-0.091	-0.204	-0.684	0.116	0.098	0.006
17		0.221	-0.229	-0.265	0.294	0.666	-1.335	0.103	0.464	-0.314	0.136
18		-0.419	0.625	0.138	0.235	-0.063	0.479	-0.278	0.101	-1.400	0.426
19		-1.296	-0.239	-0.162	0.710	0.120	0.166	-0.104	-0.010	0.097	-0.213
20		0.347	0.313	-0.157	0.158	-1.659	0.235	0.079	-0.102	0.140	0.368

Table 1: Weight matrix for the subprototypes in the hierarchical model with default settings. Minimum weights (maximum activation) are highlighted in blue.