MASTER THESIS

Extending explainability of generative classifiers with prototypical parts

AUTHOR

Michelle Peters
M-CS, Data Science & Technology

SUPERVISORS

Meike Nauta, M.Sc.
University of Duisburg-Essen
University of Twente

Prof. Dr. Christin Seifert
University of Duisburg-Essen
University of Twente

Dr. Ing. Gwenn Englebienne
University of Twente

Department of Data Management & Biometrics
Faculty of Electrical Engineering, Mathematics and Computer Science (EEMCS)
University of Twente

May 2022

UNIVERSITY OF TWENTE.

Extending explainability of generative classifiers with prototypical parts

Michelle Peters
Faculty of Electrical Engineering, Mathematics and Computer Science
University of Twente
May, 2022

Abstract

The field of Explainable Artificial Intelligence is a response to the increase in use of Artificial Intelligence and lack of insights into the reasonings behind a model's decisions. Prototypical parts have been suggested as explanations that are easy to interpret by humans, and generative models as being able to give more insight into the classification of Out-of-Distribution data. We introduce a model, ProtINN, that combines a generative model with prototypical part explanations, to give insightful explanations on its predictions. ProtINN segments the input images, and uses a pre-trained Invertible Neural Network to get the segments' activations. These activations are used for clustering to create prototypes, and to calculate the similarity between the prototypes and the segments of the to-be classified images. These similarity scores are then translated to class prediction scores by a single linear layer. To explain the class prediction scores, we present visualisations which show the prototypes, similarity scores, prediction layer weights, and input image segments. Our experiments on the ImageNet dataset show that, while we have some loss in accuracy, we gain a lot of insights into the predictions with our visualisations.
Code: https://github.com/Michiexb/ProtINN

1 Introduction

With the growing importance of deep neural networks for decision making, the interest in interpretability of those networks has increased. There is research on both the building of interpretable models and methods [3, 13, 9, 12, 7], as well as on what an explanation should look like according to social science studies [20]. Explanations on a neural network's decisions are important for finding bias, and for creating trust [9]. If someone uses a neural network to make high stakes decisions, it might be useful for them to see how the network came to its conclusion, so that they can validate the decision, or maybe even learn from it.

There are two ways to add these explanations to a model; post-hoc, and intrinsically. Post-hoc explanations try to uncover the reasons for a prediction without knowing the inner workings of the model [17]. Models that by themselves have interpretable elements are called intrinsically interpretable models [7]. While intrinsically interpretable models might have to compromize on prediction accuracy [9], an important advantage is that they give insights into the actual decision making rather than an approximation, and are thus less likely to mislead the user [17].

Humans often use case-based reasoning to solve problems. This is the method of solving a new problem by looking at similar past cases [1]. Similar objects or cases can be represented by a so-called prototype [14]. For image data, a prototype is an image representing an entire class or a subset of a class. Prototypical parts, sometimes also called prototypes [22, 6], are images representing only a part of the class object. The prototypical parts of a car might be wheels, mirrors, and headlights.

Most Explainable Artificial Intelligence (XAI) research has focused on discriminative classifiers, because these have always been more accurate than generative classifiers when it comes to complex data [24, 4]. Recently, Mackowiak et al. [18] have created a generative classifier that is on par with the state-of-the-art discriminative classifiers. They show that the generative model can provide an explanation for some cases that are not always explainable by a discriminative model [18]. Figure 1 [18] shows how a discriminative classifier cannot tell the difference between a case where an image has similar scores for multiple classes because all classes are relevant, and the case where neither class is relevant. This is because a discriminative model only models the boundaries between classes, while a generative model learns the data distribution of each class (see Figure 2) [32].

Prototypical methods for discriminative models visualize their prototypes by showing an image of their training data which is closest to the prototype in the latent space of the model. If a generative model is able to create sensical images from latent space representations that do not belong to an existing image of the given dataset, the actual prototypes can be visualized. To sum up, a generative model with prototypical learning has the following advantages:

- The model is intrinsically interpretable.
- Prototypes could be generated rather than sampled from the training data.
- Well modelled generative classifiers can be more representative of the data than their discriminative counterparts.
- Out-of-Distribution data can be recognized as such.

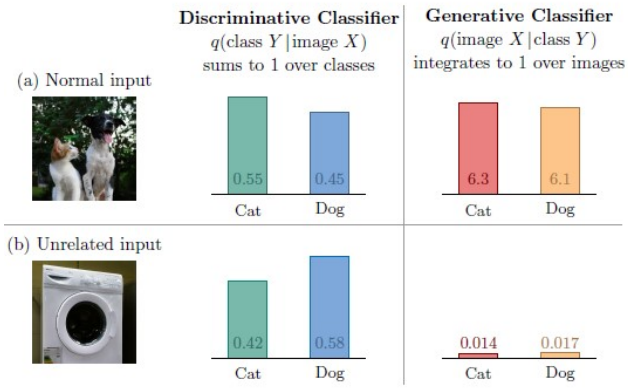


Figure 1: Example of the main advantage of a generative classifier compared to a discriminative classifier: More insight into Out-of-Distribution data predictions. Image from Mackowiak et al. [18].

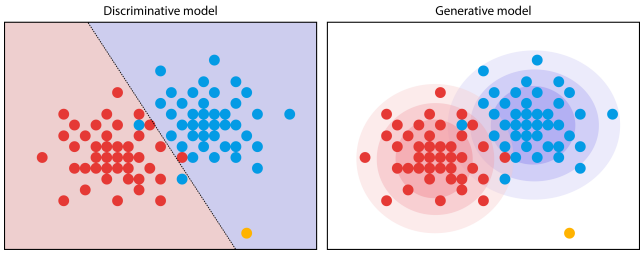


Figure 2: Difference between a discriminative model and a generative model. A discriminative model only models the boundaries between the classes. A generative model models the data distribution. A new data points (yellow) that lies far away from both data distributions might not be recognized as such by a discriminative model.

Therefore, we introduce ProtINN; a model that combines the use of prototypes and the Invertible Neural Network (INN) of Mackowiak et al. [18]. This architecture is an intrinsically interpretable model that can provide explanations for situations that cannot be properly explained by discriminative models. We aim to answer the following research questions:

- How can a generative classifier and prototypical learning be combined to develop an intrinsically interpretable model?
- How does the new interpretable model compare to the used black box INN, considering accuracy, and interpretability for Out-of-Distribution data?

To answer the first question, we develope an architecture that combines approaches from several studies. We then evaluate this model to answer the second question.

We first discuss some existing research on explanations, prototypes and generative models in Section 2. Then, in Section 3, we describe the architecture of ProtINN in detail. Section 4 discusses some important features for the visualisation design, and Section 5 explains the setup that we used for our experiments. The results of these experiments are discussed in Section 6, and conclusions and future work can be read in Section 7.

2 Related work

There has been a lot of research within the field of XAI, with many different types of explanations [3, 9, 12, 7, 13]. The choice for a specific type depends on multiple factors, which we discuss in Section 2.1. In Section 2.2, we discuss different studies that use prototypical explanations, and in Section2.3 and 2.4 discuss a state-of-the-art generative model and studies that have combined generative models and prototypes.

2.1 Considerations for explanations

Within explanation methods, there are some aspects that influence which type of explanations is suitable. Firstly, it is important to decide on whether the explanation should be global, i.e. explain the model as a whole, or local, i.e. explain the decisions for a specific input. Secondly, the amount of time that the user has to look at the results and understand them influences the level of complexity and detail that the explanation can encompass. And lastly, the nature of the user expertise should be taken into account [9]. Hohman et al. [12] organize Artificial Intelligence (AI) users into three non-mutually exclusive groups: Model developers and builders, model users, and non-experts. Model developers and builders might need the explanations for debugging, and would need a different type of explanation than users that do not have a strong understanding of the inner workings of neural networks. Model users have some understanding of neural networks, and need to be able to perform tasks such as training smaller-scale models. Non-experts have no prior knowledge on machine learning.

Explanations aimed at non-experts are not only useful for them, but also for model users and model builders [12]. Although these more specialized groups would also need additional information that might not be useful or even understandable by non-experts, the less detailed explanations could still give them some initial insights. Therefore, we will aim our explanations at non-experts, creating a basis that is useful for all types of users.

Miller [20] conducted a social science study, and found four key features for AI explanations. First, explanations should be contrastive. This means not just explaining why a certain class was predicted, but also why another was not. The second feature is that explanations should be selected. Often, a classification decision is based on many reasons, but only a few, if possible the most important ones, should be shown to the user. Thirdly, it is also important for explanations that they should not just show probability numbers. Probabilities don't matter to most people, but only the reasons why a certain class has a higher probability than another. Lastly, explanations should be social. They need to be part of a conversation or interaction. Creating conversations in a classifier model is difficult, but interaction can quite easily be added to the explanations. We will keep this in mind when creating the visualizations for the classification results.

2.2 Prototypes

A few explanation methods that are often used for image data are Features Importance, Salient Masks, and Prototype Selection [9].

With Features Importance, the set of features that is used by the model serves as the explanation, together with the features' weights. This can give insight into whether the model looks at objects, patterns, colours, etcetera. Then, it gives a scale of how important each feature is for the classification by showing the weight.

Salient Masks highlight the parts of the image that were most important for the image's classification. This can be useful if decisions are sometimes based on smaller parts of an image.

A prototype is an object that represents a set of similar objects. This prototype can either be an instance of the set it represents, or a new instance that summarizes (part of) the set [9]. This explainability method works well for image data, since images, and thus the prototypes, can be viewed directly.

By combining these methods into a prototypical parts model, we can combine the benefits of each. By using prototypical parts instead of full-image prototypes, and giving weights to those parts for the classes, we create a model that uses image parts as features. The weights for these prototypes, together with some link between the input image and the prototypes, then show the relevance of the different parts in the input image for the classification.

There have been multiple studies on prototypical explanations [6, 10, 15, 16, 22, 27, 29].

Chen et al. [6] have created an intrinsically interpretable deep network architecture, called the Prototypical Part Network (ProtoPNet), which uses prototypical parts to explain the network's decisions. During training, the model creates a set number of prototypes per class and calculates weights between them. To classify an image, the model searches for parts in the image that are similar to the class-specific prototypes and combines these similarities and the weights to retrieve probabilities for the classes. The prototypes are visualized by the patch of an image from the training dataset that is most similar to the prototype in latent space. Figure 3 shows a simplified version of the ProtoPNet architecture.

Nauta et al. [22] have made a follow-up on ProtoPNet, called ProtoTree, by adding an easy-to-interpret global explanation in the shape of a decision tree. In this tree, each node contains a prototypical part, and based on the similarity score between a prototypical part and the patch in the input image which is most similar to the prototype, the image will go to either the right or left child node or leaf. Decision trees, specifically small ones, are easily interpretable by humans, but also discriminative in nature [9]. For a generative model we can thus not use a decision tree for classification, without changing the nature of the results to discriminative.

ProtoPShare is also based on ProtoPNet, but shares its prototypes among different classes, rather than having class-specific prototypes [27]. Similar to ProtoPNet, the model has a seperate layer for obtaining prototypes during training. For each class, the number of prototypes with non-zero weights, is the same.

Singh et al. [29] have created another model based on ProtoPNet; NP-ProtoPNet. This model does not only show which prototypes are similar to parts of the input image, but also which prototypes are not similar to any part. This way, the model also explains why another class was not predicted. Because of the addition of this negative reasoning, this model has a better performance than ProtoPNet [29].

We will base our architecture design on the design of ProtoPNet [6], but not use class-specific prototypes, just like ProtoPShare [27], use positive as well as negative reasoning like NP-ProtoPNet [29], and use the easily interpretable ProtoTree as inspiration for the visualization design.

Hase et al. [10] have created a model with prototypical learning, which uses hierarchical prototypes to classify a given image at every level in a predefined taxonomy. These prototypes are not prototypical parts, but full-image prototypes combined with heatmaps to show the focus. Since we do not focus on hierarchy or full-image prototypes, we do not consider this model for our design.

The Bayesian Case Model (BCM) learns prototypes that best represent clusters in the dataset by joint intergerence on cluster labels, prototypes and important features [15]. These prototypes are full images from the training dataset and are shown together with two important features, such as colour and shape, to explain a cluster. The model is only used on simple datasets with few features, such as emoticons with different colours and facial expressions and the Handwritten Digit dataset. Since we will use a more complex dataset as well as prototypical parts, we do not draw inspiration from this model.

Ghorbani et al. [8] have created a method called Automatic Concept-based Explanations (ACE), which uses prototypical parts to globally explain a class. Although this method only gives a global explanation of the classes rather than local explanations about the classification of a specific image, we can use this method with some adjustments as the basis for our local explanations. This will be explained in more detail in Section 3.2. The ACE method uses Simple Linear Iterative Clustering (SLIC) [2] to create superpixels for image segmentation, because compared to other superpixel algorithms, SLIC has a small runtime while producing high boundary recall and a low undersegmention error [23]. SLIC segmentation adapts K-Means clustering to generate the superpixels and reduces the number of distance calculations by limiting the search space [2]. The size of this search space is defined by the approximate size of the superpixels. Besides location, the distance between two pixels is also based colour, leading to sensical segments.

A potential risk of segmenting the images before forwarding them through a pretrained model, is that the model is not trained on segments, but rather on full images. For the model used in ACE [8], however, this was not a problem. We will therefore also use SLIC as our segmentation method for the creation of our prototypical parts.

2.3 State-of-the-art generative model

Most research in XAI has been focused on discriminative models, since generative models are difficult to model well.

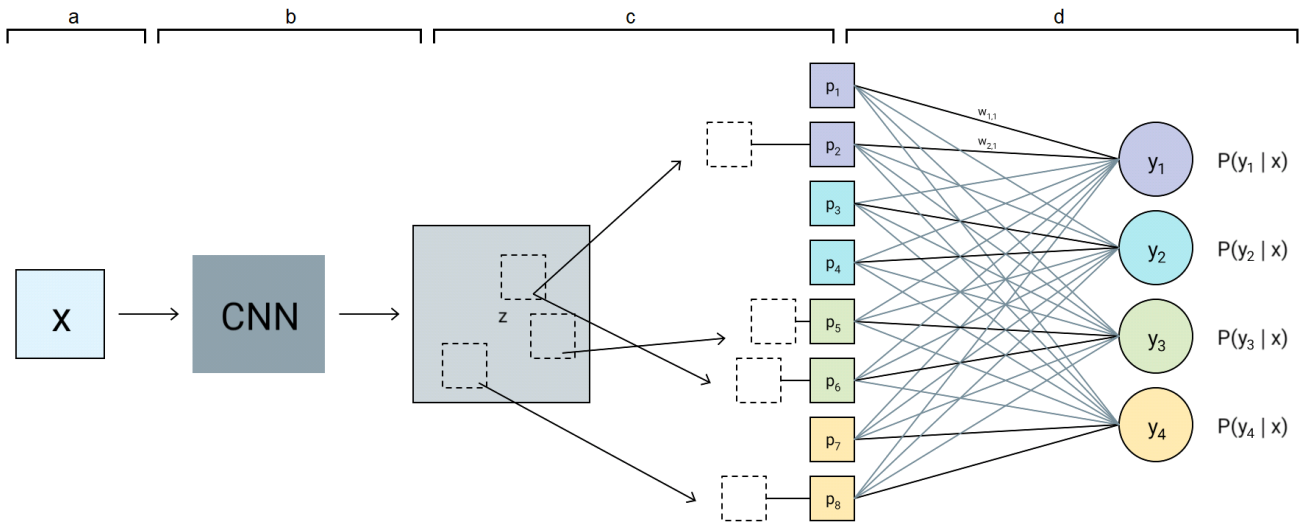


Figure 3: Architecture of ProtoPNet. a) The input image. b) The input is forwarded through the CNN. c) Patches of the resulting latent space for the image are compared to the class-specific prototypical parts. d) These similarity scores are combined with the weights to retreive the class probabilities.

This usually results in low classification accuracy, especially for fine-grained images [30].

Recently, Mackowiak et al. [18] developed a new generative model that is competitive to discriminative models. This model consists of an Invertible Neural Network with an affine coupling block architecture, that is built to resemble the well-performing discriminative ResNet-50 architecture [11] as closely as possible. It has a high accuracy for the fine-grained image dataset ImageNet. Since generative classifiers can have some great advantages to discriminative ones, this model is interesting to combine with the existing explainability methods for discriminative models. For example, it can give more insights for cases where the image is Out-of-Distribution.

We will use this model of Mackowiak et al. [18] for our architecture, as it is state-of-the-art and there are no good alternatives when it comes to generative classifiers.

2.4 Generative models and prototypes

Some research has been conducted on the combination of generative models and prototypes [15, 31]. However, these usually do not focus on using the prototypes for classification, and even less work with fine-grained image data. Li et al. have added additional explainability to a generative model by using prototypes [16], but added a discriminative component to their model. They used the encoder part of an AutoEncoder to create the latent space that connects to the prototype classifier network, and the decoder to translate the prototype latent spaces into images. The prototype classifier network starts with a prototype layer, followed by a fully connected layer, and at the end a softmax layer, which means that the model is still trained in a discriminative manner. As they can't benefit from all advantages of a fully generative classifier, their results could still be improved. Our model will be fully generative, to ensure that our model not only gains the generative benefits when creating the prototypes, but also presents more insights with its class prediction scores.

3 ProtINN

In this section, we present the design of our model. We start with a short overview, and go into more details for each part in the next subsections.

3.1 Model overview

A visual overview of the model can be seen in Figure 4. The training phase (Figure 4a) focuses on creating prototypes and training a mapping from these prototypes to the classes of the dataset. The classification phase (Figure 4b) is used to classify any image and give a visual explanation on the reason for the classification.

Training phase. In order to classify images with prototypes, we first need to create those prototypes and find their relation to the classes. This is done in the first stage with the following steps:

- **a.** Segmentation of the input data (Section 3.2).
- **b.** Forward pass through the INN base model (Section 3.4).
- **c.** Prototype generation (Section 3.5).
- **d.** Training the prediction layer (Section 3.6).

Classification phase. After all steps of the training phase, the model is ready to be used for classification. This happens as follows:

- **a.** Segmentation of the input data (Section 3.2).
- **b.** Forward pass through the INN base model (Section 3.4).

c. Similarity calculation (Section 3.7).
d. Forward pass through the prediction layer (Section 3.6).
e. Visualization of the prediction output (Section 4).

3.2 Segmentation

Since we use prototypical parts, rather than full-image prototypes for our explanations, we first segment the input images. Similar to ACE [8], we use SLIC superpixels [2] for the segmentation. In ACE, the images are segmented at three different resolutions: 15 segments per image for the larger objects, 50 segments per images, and 80 segments per image for patterns. To save on runtime and memory, we segment our images at 15 and 50 segments per image. Since the superpixels all have different shapes and sizes, but a neural network is always trained for one specific input shape and size, the segments need to be resized and padded. In ACE, the segments are padded with a shade of grey. Since the model is not trained on a dataset with grey backgrounds, we instead use the bounding box of the segment to crop our segment to a rectangle, which is then resized to 224x224 to fit the base model's input requirements. A downside of using patches instead of padding the segments with a single colour, is that some parts of the image might appear in more patches, depending on the shapes of the segments created by SLIC. See Appendix C for some results of experiments for the different types of segment padding.

3.3 Base model

We use IBINN, an Invertible Neural Network by Mackowiak et al. [18] as our base model to retrieve the latent space of the input images.

IBINN uses an IB objective to balance the training of their model between invertibility and classification. Given some features Z of a network, inputs X, and ground-truth outputs Y, the IB loss is calculated by using the mutual information I, and balanced with $\hat{\beta}$:

$$\mathcal{L}_{\text{IB}} = I(X, Z) - \hat{\beta} I(Y, Z) \tag{1}$$

A model with $\hat{\beta}$ = 0 only focuses on invertibility during training, while a model with $\hat{\beta} = \infty$ only focuses on the prediction accuracy. Since, by design, an INN preserves all information of the original image, using an IB objective does not automatically work. Therefore, very low noise is added to the inputs when training the model [18].

For $\hat{\beta}$ = 8, Mackowiak et al. [18] mention an accuracy of 74.59% for their model trained on the ImageNet dataset. However, the checkpoint file of this fully trained model is not available. Instead, a file for a shortly trained model with an accuracy of 0.12% is available. Training this model for an additional 20 epochs gives an accuracy of 67.85%, which is sufficient for our research.

3.4 Retrieving latent space

An INN needs to have the same number of input as output features in order for the data to be invertible, but in order to generalize the data for classification, fewer layers are required. Therefore, for the last layers, IBINN splits the data into two parts: z_{fc} and z_{conv}. The model learns to put the relevant information for classification into z_{fc} and the rest of the information that is only needed for the reproduction of the image in z_{conv}. We use the output of z_{fc} to cluster the data, since this part contains the generalized information.

3.5 Prototype generation

To create the prototypes, we cluster the latent representations of the training data segments, as was done to create the concept groups in ACE [8]. We use Mini-Batch K-Means clustering, an adaptation of the popular K-Means clustering algorithm, since this method is really fast, even on large datasets [28]. After clustering, we remove clusters that contain less than 15 segments since we argue that such a small cluster does not serve as a significant representation for the data, given a dataset of 1300 images per class.

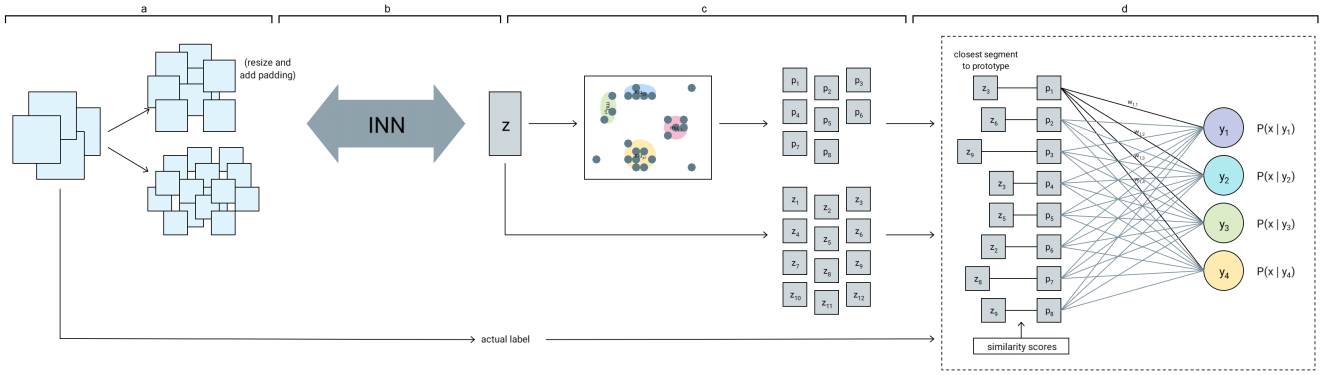
K-Means returns the cluster centers, which we use as the prototypes. Since K-Means clustering needs to know the number of clusters beforehand, we run our experiments for different numbers of clusters, and compare the prediction accuracies and runtime to find out which number of clusters gives a good compromise between the two. We only do this for a data subset of 20 classes. To visualize the prototypes, we test whether invertion can give sensical images for not-real datapoints, by using interpolation. As can be seen in Figure 5, where interpolation is performed between two images with 15 steps, even a single step already results in a completely noisy image. More invertibility experiments can be found in Appendix B. Since the invertibility cannot be used to visualize the prototypes, we will use the training image segment which is closest to the cluster mean in latent space. Since this image segment will then be shown as the prototype in the explanation visualization, we will also use the latent space for this image for the similarity calculations. Otherwise, the explanation might be confusing when a test image segment seems very similar to the shown prototype but a low similarity score is shown.

3.6 Prediction layer

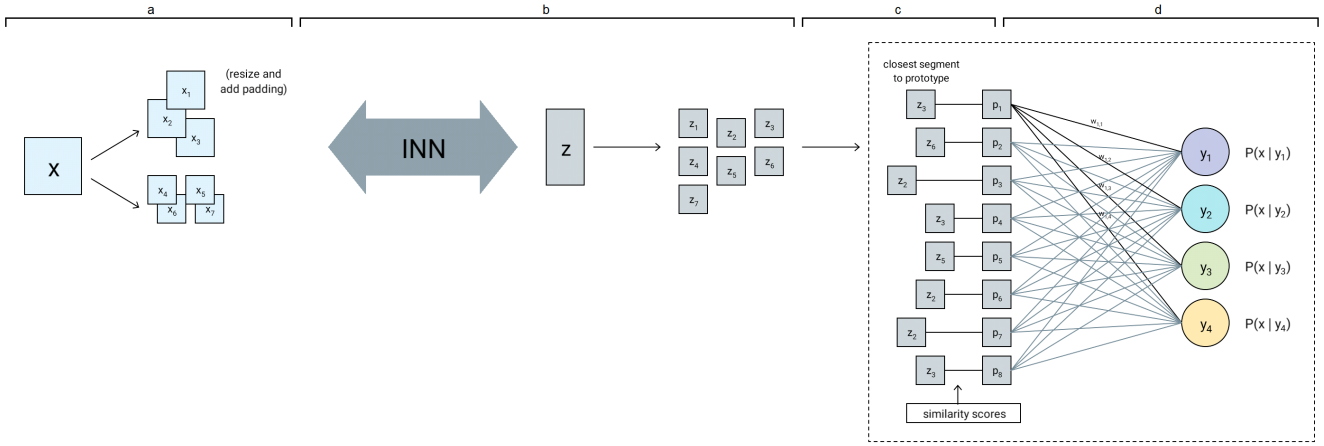
When we have the prototypes, we need to map their relations to the classes. We do this by training a single linear layer without bias, since this keeps the model interpretable. In the end, we want to be able to give a full image as input for our architecture. Therefore, we also need a mapping from the full image to the prototypes. For this, we use similarity scores. For each training image, we segment the image and get the segments' latent representations. For each existing prototype, we then find the input image's segment that is closest to it in latent space, and the similarity score between the two (explained further in Section 3.7) is then used as the input for the prediction layer's node belonging to that prototype. With these similarity mappings as input, and the actual labels of the full images as target, we train this prediction layer until convergence.

3.7 Similarity scores

The similarity scores represent to which extent a prototype is present in the input image. These scores range from zero



(a) Training phase: a) The input is segmented at multiple sizes. b) The segments are forwarded through the INN. c) The resulting latent spaces for all training image segments are clustered. These clusters result in prototypes. d) The weights between the prototypes and classes are trained.



(b) Classification phase: a) The input is segmented at multiple sizes. b) The segments are forwarded through the INN. c) The resulting latent spaces of the image segments are compared to the prototypes. d) These similarity scores are forwarded through the prediction layer to retrieve class prediction scores. The visualization step (e) is not shown in this image.

Figure 4: Architecture of the model

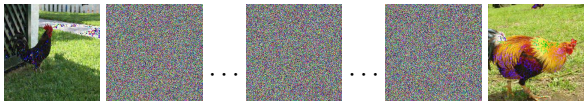


Figure 5: Interpolation in latent space between two images with 15 steps on $[z_{\text{fc}}, z_{\text{conv}}]$, inverted back to image.

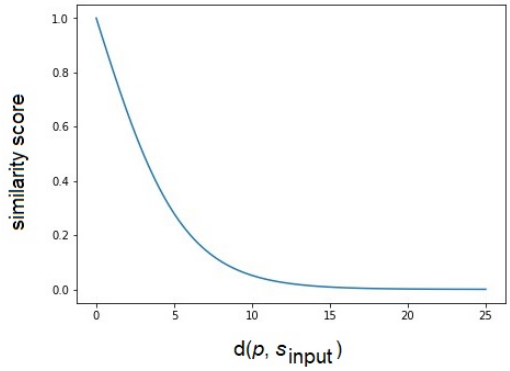


Figure 6: Plot of the similarity score in Equation 2, with $\text{d}(p, s_{\text{furthest}}) = 5$

to one, where zero means that the prototype is most likely not present, and one means that it very likely is. To transform the distances between the prototype and input segment to similarities ranging from zero to one, we use Equation 2, where $\text{d}(p, s_{\text{input}})$ is the Euclidean distance between the prototype and the given input segment, and $\text{d}(p, s_{\text{furthest}})$ the Euclidean distance between the prototype and the furthest training segment in the prototype's cluster. We use this last term to normalize the score with respect to the size of the cluster; segments that are close to the cluster center get a similarity score close to one, while segments that are further away than the cluster's furthest training segment get a similarity score close to zero. We multiply the distance of the furthest segment by 1.1 to ensure that input segments that are near the border of the cluster are also seen as clearly belonging to that cluster. Figure 6 shows an example plot of the equation, where the distance of the furthest segment in the cluster is five.

$$\text{sim. score} = \tanh\left(-\frac{\left|\text{d}(p, s_{\text{input}})\right|}{1.1 * \text{d}(p, s_{\text{furthest}})}\right) + 1 \qquad (2)$$

4 Visualization Design

For this visualization, we will look at the most important features for explanations, according to a social science study by Tim Miller [20]:

- Contrastivity: explain the event relative to an event that did not occur.
- Selectivity: don't show the complete reasoning, but only a few important causes.
- No probabilities: only the reasons for the difference in probabilities is important.
- Social: people need to be able to interact with or discuss the explanations.

To create this desired interactivity, we use Panel Holoviz [25] in python to create the visualizations. This package makes it possible to create interactivity for Matplotlib visualizations. With this interactivity, the user can choose which classes to compare. We will show some visualizations to demonstrate that our architecture can provide insightful explanations, but our goal is not to find the best visualizations, but rather only show some possibilities.

5 Experimental Setup

This section discusses the setup we used for our experiments, which includes an explanation on the used dataset, the model and training parameters, and which metrics we use to evaluate our model.

5.1 Dataset

We use the ImageNet Large Scale Visual Recognition Challenge 2012 (ILSVRC2012) dataset [26] for our experiments. This dataset consists of 1000 object classes of fine-grained images, and our base model is already trained on this dataset. Each class in the ImageNet dataset contains 1300 training images and 50 validation images, and the dataset has a total of 100.000 test images [26]. These images are hand-labelled photographs collected from several search engines. Some of the classes in this dataset very clearly consist of different objects, such as cars, while other classes like dogs and volcanos are less object oriented. For the test images, the labels are not published, since the dataset is part of a challenge. Therefore, we will use the validation set for testing instead. The used INN requires both the training and validation data to be in the ImageFolder format. Therefore, we place the validation images, which come in a single directory along with a text file containing the image labels, in class-specific directories.

5.2 Model parameters

The prediction layer of our model is a single layer without bias. For our experiments, we use 20 classes, and 400 clusters or prototypes, giving us 400 input features and 20 output features. This results in $400 * 20 = 8000$ weights. Since the model is trained without bias, the model only consists of these 8000 parameters.

5.3 Hyperparameters

To control the training process of our prediction layer, we need to set the hyperparameters.

As criterion for our training loss, we use the sum of the mean squared error between the output and the target of all items in the training batch, which we want to minimize.

We use stochastic gradient descent with a learning rate of 0.1 for our optimizer. To prevent exploding gradients, we use gradient clipping with a maximum norm of 15 after each training batch.

After each epoch, we use the same criterion for calculating the loss on the evaluation data as we use for the training loss. When the evaluation loss has stopped decreasing for 10 epochs, we reduce the learning rate with a factor 0.1. When measuring this descrease in evaluation loss, we use a threshold of 10^{-4}, meaning that a decrease smaller than this threshold will not be considered as a decrease.

When the change in evaluation loss has been lower than 10^{-5} for 5 consecutive epochs, we consider the model to have converged and stop the training process. To prevent the model from training endlessly when the model does not converge, we set the maximum number of epochs at 400.

5.4 Evaluation metrics

We evaluate the prediction performance of our model by calculating the accuracy. Even though a main advantage of a generative model is the fact that the model can recognize multiple classes in a single image, for this accuracy we only look at the class with the single highest prediction score and whether that's the same class as the actual label.

We also want to evaluate the explanations of the model. Nauta et al. [21] have proposed a new AI explanation framework; Co-12, a list of 12 explanation quality properties. Some of these include correctness, compactness, and completeness. Correctness describes how truthful an explanation is with regards to the predictive model that it's trying to explain. Compactness considers the size of the explanation. And completeness describes to which extent the predictive model is explained. These last two should be in balance. An explanation should not be too big to understand, but should also not be too incomplete.

These three properties can be measured with Incremental Deletion. For this, important features according to the explanation are removed one by one, to observe the change in prediction [21]. Correctness is measured by the decrease in prediction accuracy when removing the most important feature. Compactness measures how many features need to be deleted, in order of the explanation's shown importance, to change the prediction. If all features that need to be deleted for this change are shown in the explanation, the explanation can be called output-complete.

Since we only use a single linear layer without bias to make the predictions, removing the weight between a prototype and a class, will always decrease the output score for that class with exactly weight*similarity. Obscuring a seemingly important part of an input image will have a similar effect. However, the exact change in output is dependend on the similarity scores of the next most similar segments for all prototypes for which the changed image part was most similar before the perturbation. Therefore, our explanation is inherently correct.

We can show importance of a feature in different ways for local or global visualizations. For local explanations, importance of a feature is dependent on to what extent that feature is present in the image and how much it therefore contributes to the prediction score. For a global explanation, there are no similarity scores, so the importance of a feature is determined by the weight. Therefore, we perform Incremental Deletion for measuring compactness twice: 1) the weights are sorted by highest weight*similarity score, since this value determines how important an image segment was for the prediction, 2) we sort only by the weights, since this value determines how important the prototype is for the class.

The number of features shown by our explanation can be easily adjusted, so we can use the results from the Incremental Deletion experiment to determine the number of features needed for an explanation that's almost always output-complete.

Since better insight into predictions for Out-of-Distribution data is one of the main advantages of a generative classifier, we will also look at the models ability to recognize multiple classes within a single image, as well as images that do not belong to any of the classes that the model was trained on. To demonstrate multiple class recognition, we will use an image from the dataset which contains two classes, as well as augment an image by adding a prototype with a high weight for a class with an initially low prediction score for the given image.

6 Results

In this section, we discuss our model's performance, the created explanations and the results of our evaluations. Figure 7, 8 and 9 show global explanations for the model, and Figure 10 shows a local explanation for a given image.

6.1 Predictive performance

We trained our model on a subset of Imagenet with the following 20 classes: 'slide rule', 'web site', 'green lizard', 'balance beam', 'gong', 'radio telescope', 'Granny Smith', 'ladle', 'television', 'cliff', 'bullfrog', 'file', 'basketball', 'pizza', 'cicada', 'Bouvier des Flandres', 'cuirass', 'toilet seat', 'rapeseed', and 'moped'.

Training until convergence gives us a model accuracy of 72.3%, and Figure 7 shows a bar chart with the prediction accuracy per class, as well as which other class the class is mostly misclassified as and how often. IBINN has a prediction accuracy of 82.4% on the same 20 classes.

6.2 Interpretability

Global explanations Figure 7 gives a quick insight into which classes the classification of the model is best and worst for. We see that class 'balance beam', for example, is very often misclassified as 'basketball'. We can look into why this happens by looking at other visualizations. Figure 8 shows a visualization for which two classes can be selected to compare. This visualization shows that both classes ('basketball' and 'balance beam') have some prototypes with body parts in their top 10, and the fifth most important prototypes for both are the same, but with a higher weight for the 'basketball' class. The model bias towards 'basketball' rather than 'balance beam' can be seen in Figure 9, which shows that the total sum of the weights for 'basketball' is higher than that of 'balance beam', mostly due to a higher sum of positive weights.

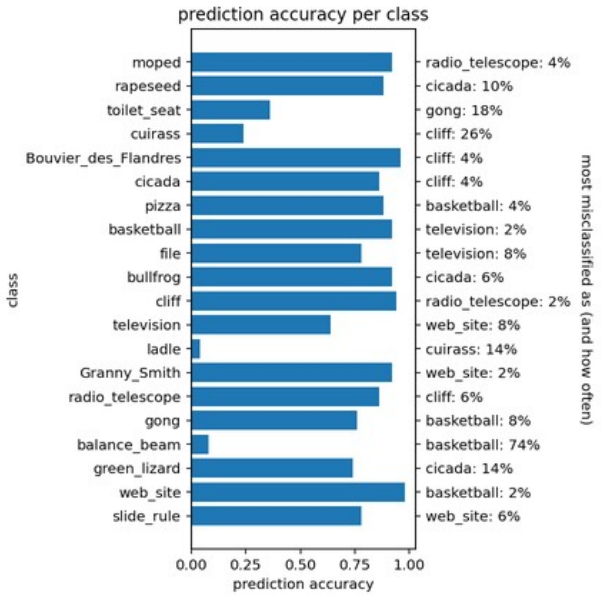


Figure 7: Global explanation which shows the prediction accuracy per class, and for each class which class it is mostly misclassified as, followed by the number of times this has happened.

Local explanations To demonstrate the use of the local explanations, we take an image of class 'balance beam' which is misclassified as 'basketball'. Figure 10 shows that the image not only has a high score for 'basketball', but also for its actual class. The bottom plot in the figure shows that the reason for the high 'basketball' score is most likely due to all the body parts in the image, which are important for 'basketball' images. Even though body parts are also important for class 'balance beam', the weights for the most contributing 'basketball' prototypes are higher than the weights of the 'balance beam' prototypes.

Incremental Deletion We run Incremental Deletion on our model for the subset of 20 classes. The results are shown in Table 1. By looking at the mean values, we can assume that only the one or two most important features are usually needed to explain a prediction. But even the maximum of four features will probably not compromize the compactness of the explanation too much, while making sure that, at least for the evaluation dataset, the explanation is always output-complete.

Out-of-Distribution detection We demonstrate the capability of our model to detect Out-of-Distribution data in Figure 11 and Figure 12. Figure 11a from the Imagenet evaluation dataset [26] shows an image of class 'green lizard' and its prediction scores by our model. Our model clearly classifies the image as the correct class. Figure 11b shows the same image, but augmented by adding the most important prototype of the class 'slide rule' to the image before the segmentation process, since this class has a very low prediction

	sorted by sim $* w$	sorted by w
mean	1.337	1.354
std.	0.620	0.632
min	1	1
max	4	4

Table 1: Results of Incremental Deletion: Number of most important features that need to be removed to change the prediction outcome, where feature importance can be measured by sim $* w$ or only w.

score for the original image. This image has approximately the same prediction scores as the the original image, except for the class 'slide rule', which has a very high prediction score. This shows that our model recognizes both the green lizard, as well as the slide rule in one image.

To quantify this, we select 5 images per class, thus 100 images in total, to which we add one of the five segments closest to the cluster centroid of the most important prototype for one of the five classes with the lowest prediction score. For 36% of the images, the prediction score of the added class has exceeded the prediction score of at least one other class that was previously higher. The average number of classes that the selected class has surpassed is -0.93. Some additional augmented images and their change in prediction can be found in Appendix H.

Figure 12 shows two images from the Imagenet evaluation dataset [26], followed by the predictions for those image by our own model, as well as the predictions of the black-box IBINN model. IBINN outputs the predictions as distances, which are shown in the third row (Figure 12e and 12f), where the image is classified as the class with the lowest distance. These scores are all very high, and therefore need to be zoomed in on, to be able to clearly see the prediction differences. Since the distances for different images are not normalized to the same scale, we can only inspect the differences in class scores per image, and not between the different images. We use Equation 3 to invert these distances to something similar to our similarity scores, while highlighting the differences between the scores for the different classes. Image 12g and 12h show these transformed scores for the two used images.

$$\text{score}_i = -\text{dist}_i + \max(\text{dist}) \tag{3}$$

Figure 12a shows an collage-image of class 'Granny Smith', which also contains images of pizza. Figure 12c shows that our model recognizes both 'Granny Smith' and 'pizza' in the image. IBINN also recognizes both classes in the image (see Figure 12g). To show that our model can also recognize images from classes that the model is not trained on as Out-of-Distribution data, we use an image of class 'mushroom' (Figure 12b). Figure 12d shows that our model can indeed tell that the image most likely does not belong to any of the classes. Since the IBINN scores have no good normalization over all classes, Figure 12h only shows that there are not really any classes that are a lot more likely than other classes, but not whether the image falls under all or none of the classes (note that the y-axis only goes to 17.5,

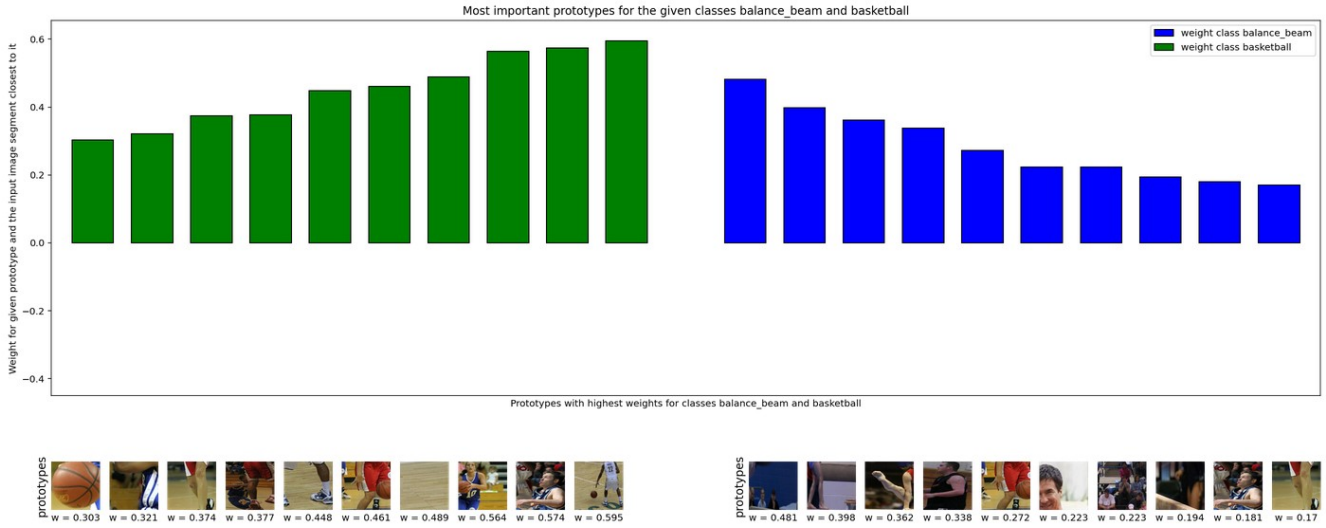


Figure 8: Global explanation for comparing most important prototypes for two selected classes; ’balance beam’ and ’basketball’.

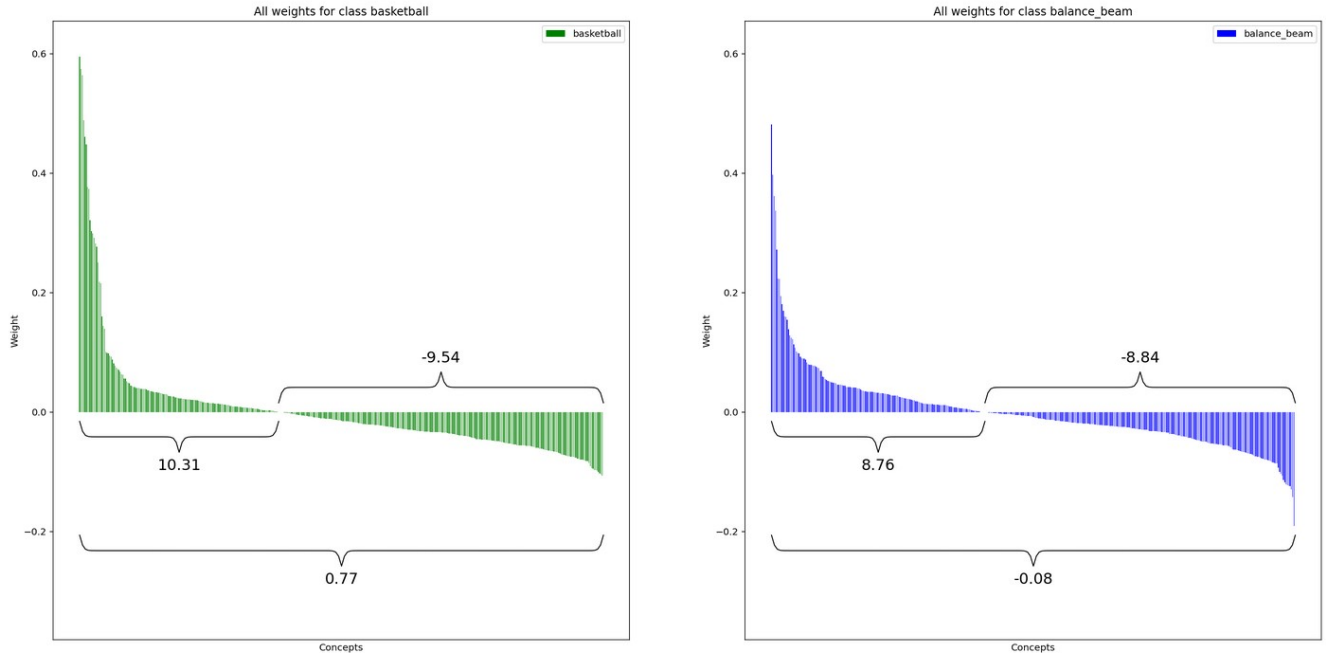


Figure 9: Global explanation for comparing total sum of positive and negative weights of the prototypes for two selected classes; ’balance beam’ and ’basketball’. The model is thus, compared to ’balance beam’, biased towards ’basketball’.

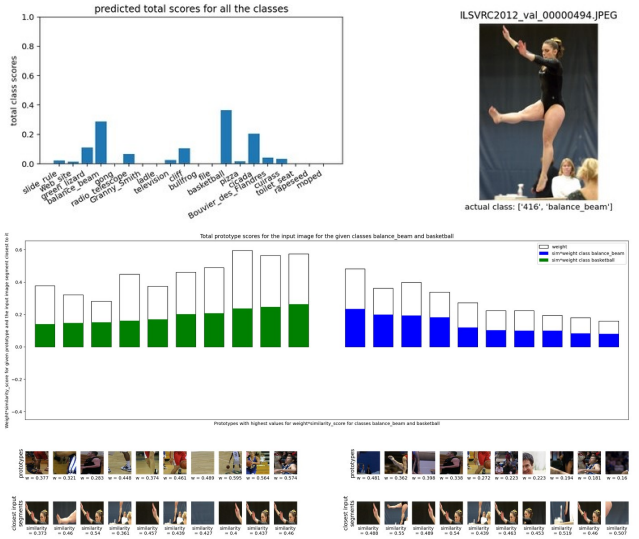


Figure 10: Local explanation of a given image (top right), with class prediction scores (top left), and the weights and contribution scores of the most contributing prototypes and their closest segment for two selected classes (bottom).

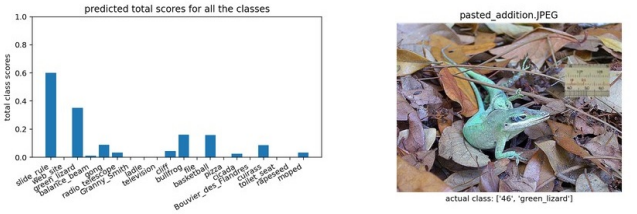
which is the difference between the highest and lowest prediction score, but not an indication of how likely the class is present in the image). IBINN can thus only give insights into the presence of a class compared to other classes, but not compared to other images, while ProtINN can distinguish between all classes being present and all classes not being present in the input image.

7 Conclusion

In this paper, we presented an architecture that is intrinsically interpretable, and can give both global and local explanations. The architecture can show the difference between images that are Out-of-Distribution due to it containing multiple classes or none of the classes, by showing the prediction scores. The explanations can also give insights into why an image was misclassified by showing the most contributing prototypes and their closest input image segments for two classes that can be chosen by the user. By choosing the expected class and the predicted class, the user can deduce what caused the misclassification. By using Incremental Deletion, we discovered that the explanations only need to show a maximum of four prototypes to be output-complete. The accuracy of our model is lower than that of the black box INN that we use in our architecture. We thus lose a bit of prediction accuracy, but gain additional interpretability on why an image was or was not classified as a certain class, and on whether an Out-of-Distribution image belongs to multiple classes or none, while the black box INN fails to distinguish between these two situations due to how its prediction scores are normalized.

In the future, user studies could be used to test the user interpretability of the visualizations, as well as to create better visualization designs.

(a) The original image has the highest prediction score for its actual class; green lizard, and a low score for class 'slide rule'.



(b) The image augmented with the most important prototype for the class 'slide rule', has high scores for both 'green lizard', as well as 'slide rule'.

Figure 11: By adding the most important prototype for the class 'slide rule', the prediction score for this class increases drastically, while the scores for the other classes stay approximately the same.

To decrease model complexity, and thus increase interpretability, the model could be trained to decrease the number of non-zero weights. This way, only a few prototypes will be active per class, which could drastically decrease the number of prototypes in the class explanations. The model could also be trained to only contain positive weights. The influence of negative weights can now be confusing for the user, when trying to figure out why a certain prediction was or was not made.

We also hope that INN's with better invertibility become available, to be able to use the actual cluster centers as prototypes, since invertibility was one of the main advantages to using an INN for our architecture. It would also be interesting to see whether replacing the INN with a CNN with high prediction accuracy would work in our architecture, and whether this would yield a better performance for our model. Another option for re-adding invertibility would be to replace the INN with an Auto-Encoder (AE), since an AE trains an encoder to generalize the image information, and a decoder to invert that generalized information back to an image. Invertibility on cluster means rather than actual images would still need to be tested.

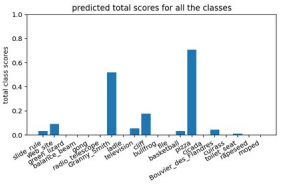
References

[1] Aamodt, A., and Plaza, E. 1994. Case-based reasoning: Foundational issues, methodological variations, and system approaches. *AI Communications* 7(1):39–59.

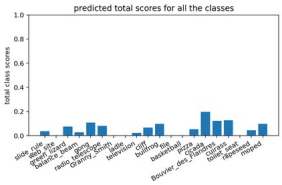
[2] Achanta, R.; Shaji, A.; Smith, K.; Lucchi, A.; Fua, P.; and Süsstrunk, S. 2012. Slic superpixels compared to state-of-the-art superpixel methods. *IEEE Trans-*

(a) Image from Imagenet that is actually a collage of pizza images and an apple image. Label in Imagenet is 'Granny Smith'.

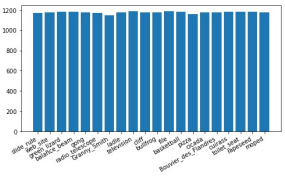
(b) Image of class 'mushroom'. The model is not trained for this class.



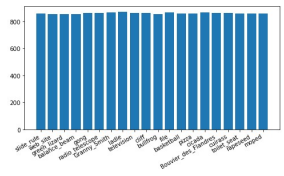
(c) Prediction scores by our model for (a).



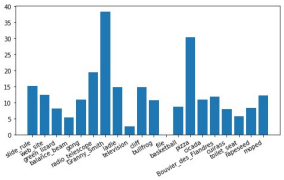
(d) Prediction scores by our model for (b).



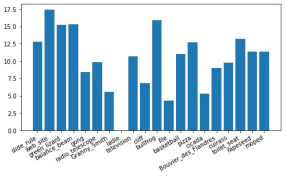
(e) Distances to classes by IBINN for (a).



(f) Distances to classes by IBINN for (b).



(g) Distances by IBINN transformed to scores with -dist+max(dist) for (a).



(h) Distances by IBINN transformed to scores with -dist+max(dist) for (b).

Figure 12: Two Out-of-Distribution images from the Imagenet dataset [26]. (a) has actual label 'Granny Smith', which is one of the classes that the model is trained on, but the image also contains pizza, which is also one of the trained-on classes. (b) has actual label 'mushroom', which is not part of the trained-on classes.

actions on Pattern Analysis and Machine Intelligence 34(11):2274–2282.

[3] Adadi, A., and Berrada, M. 2018. Peeking inside the black-box: A survey on explainable artificial intelligence (xai). *IEEE Access* 6:52138–52160.

[4] Bishop, C. M., and Lasserre, J. 2007. Generative or discriminative? getting the best of both worlds. *Oxford University Press* 3–24.

[5] Brock, A.; Donahue, J.; and Simonyan, K. 2019. Large scale gan training for high fidelity natural image synthesis. *arXiv:1809.11096 [cs, stat]*. arXiv: 1809.11096.

[6] Chen, C.; Li, O.; Tao, D.; Barnett, A.; Rudin, C.; and Su, J. K. 2019. This looks like that: Deep learning for interpretable image recognition. In Wallach, H.; Larochelle, H.; Beygelzimer, A.; d'Alché-Buc, F.; Fox, E.; and Garnett, R., eds., *Advances in Neural Information Processing Systems*, volume 32. Curran Associates, Inc.

[7] Das, A., and Rad, P. 2020. Opportunities and challenges in explainable artificial intelligence (xai): A survey. *arXiv:2006.11371 [cs]*. arXiv: 2006.11371.

[8] Ghorbani, A.; Wexler, J.; Zou, J. Y.; and Kim, B. 2019. Towards automatic concept-based explanations. *Advances in Neural Information Processing Systems* 9273–9282.

[9] Guidotti, R.; Monreale, A.; Ruggieri, S.; Turini, F.; Giannotti, F.; and Pedreschi, D. 2018. A survey of methods for explaining black box models. *ACM Computing Surveys* 51(5).

[10] Hase, P.; Chen, C.; Li, O.; and Rudin, C. 2019. Interpretable image recognition with hierarchical prototypes. *The Seventh AAAI Conference on Human Computation and Crowdsourcing (HCOMP-19)* 32–40.

[11] He, K.; Zhang, X.; Ren, S.; and Sun, J. 2016. Deep residual learning for image recognition. In *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 770–778. IEEE.

[12] Hohman, F.; Kahng, M.; Pienta, R.; and Chau, D. H. 2019. Visual analytics in deep learning: An interrogative survey for the next frontiers. *IEEE Transactions on Visualization and Computer Graphics* 25(8):2674–2693.

[13] Islam, S. R.; Eberle, W.; Ghafoor, S. K.; and Ahmed, M. 2021. Explainable artificial intelligence approaches: A survey. *arXiv:2101.09429 [cs]*. arXiv: 2101.09429.

[14] Keane, M. T., and Kenny, E. M. 2019. *How Case-Based Reasoning Explains Neural Networks: A Theoretical Analysis of XAI Using Post-Hoc Explanation-by-Example from a Survey of ANN-CBR Twin-Systems*, volume 11680 of *Lecture Notes in Computer Science*. Springer International Publishing. 155–171.

[15] Kim, B.; Rudin, C.; and Shah, J. 2014. The bayesian case model: A generative approach for case-based reasoning and prototype classification. *Advances in Neural Information Processing Systems 27 (NIPS 2014)* 1–9.

[16] Li, O.; Liu, H.; Chen, C.; and Rudin, C. 2018. Deep learning for case-based reasoning through prototypes: A neural network that explains its predictions. *Proceedings of the AAAI Conference on Artificial Intelligence* 32(1).

[17] Lipton, Z. C. 2018. The mythos of model interpretability: In machine learning, the concept of interpretability is both important and slippery. *Queue* 16(3):31–57.

[18] Mackowiak, R.; Ardizzone, L.; Kothe, U.; and Rother, C. 2021. Generative classifiers as a basis for trustworthy image classification. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, 2971–2981.

[19] Manning, C. D.; Raghavan, P.; and Schütze, H. 2008. *Introduction to Information Retrieval*. Cambridge, UK: Cambridge University Press.

[20] Miller, T. 2019. Explanation in artificial intelligence: Insights from the social sciences. *Artificial Intelligence* 267:1–38.

[21] Nauta, M.; Trienes, J.; Pathak, S.; Nguyen, E.; Peters, M.; Schmitt, Y.; Schl ötterer, J.; van Keulen, M.; and Seifert, C. 2022. From anecdotal evidence to quantitative evaluation methods: A systematic review on evaluating explainable ai. *arXiv preprint arXiv: 2201.08164*.

[22] Nauta, M.; van Bree, R.; and Seifert, C. 2021. Neural prototype trees for interpretable fine-grained image recognition. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, 14933–14943. IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2021 ; Conference date: 19-06-2021 Through 25-06-2021.

[23] Neubert, P., and Protzel, P. 2012. Superpixel benchmark and comparison. *Proc. of Forum Bildverarbeitung* 205–218.

[24] Ng, A., and Jordan, M. 2001. On discriminative vs. generative classifiers: A comparison of logistic regression and naive bayes. In Dietterich, T.; Becker, S.; and Ghahramani, Z., eds., *Advances in Neural Information Processing Systems*, volume 14. MIT Press.

[25] Rudiger, P. 2018. Panel holoviz. `https://panel.holoviz.org` (Oct. 2018).

[26] Russakovsky, O.; Deng, J.; Su, H.; Krause, J.; Satheesh, S.; Ma, S.; Huang, Z.; Karpathy, A.; Khosla, A.; Bernstein, M.; Berg, A. C.; and Fei-Fei, L. 2015. ImageNet Large Scale Visual Recognition Challenge. *International Journal of Computer Vision (IJCV)* 115(3):211–252.

[27] Rymarczyk, D.; Struski, L.; Tabor, J.; and Zieliński, B. 2021. Protopshare: Prototypical parts sharing for similarity discovery in interpretable image classification. In *Proceedings of the 27th ACM SIGKDD Conference on Knowledge Discovery & Data Mining*, KDD '21, 1420–1430. New York, NY, USA: Association for Computing Machinery.

[28] Sculley, D. 2010. Web-scale k-means clustering. *Proceedings of the 19th International Conference on World Wide Web* 1177–1178.

[29] Singh, G., and Yow, K.-C. 2021. These do not look like those: An interpretable deep learning model for image recognition. *IEEE Access* 9:41482–41493.

[30] Ulusoy, I., and Bishop, C. M. 2006. *Comparison of Generative and Discriminative Techniques for Object Detection and Classification*, volume 4170 of *Lecture Notes in Computer Science*. Springer Berlin Heidelberg. 173–195.

[31] Wu, N.; Sun, S.-C.; Zou, Y.; and Yu, Y. 2020. Imbalanced question classification using generative prototypes. *2020 5th International Conference on Robotics and Automation Engineering (ICRAE)* 206–210.

[32] Wu, Y.-h.; Guo, J.; and Liu, G. 2009. Improved discriminative training for generative model. *The Journal of China Universities of Posts and Telecommunications* 16(3):126–130.

class code	class name
n04238763	slide rule
n06359193	web site
n01693334	green lizard
n02777292	balance beam
n03447721	gong
n04044716	radio telescope
n07742313	Granny Smith
n03633091	ladle
n04404412	television
n09246464	cliff
n01641577	bullfrog
n03337140	file
n02802426	basketball
n07873807	pizza
n02256656	cicada
n02106382	Bouvier des Flandres
n03146219	cuirass
n04447861	toilet seat
n11879895	rapeseed
n03785016	moped

Table 2: Classes used for the experiments.

A Dataset classes

Table 2 shows the 20 classes from ImageNet that were used for the experiments, including their class codes.

B Invertibility and choosing IBINN model

We first inspect the invertibility of the IBINN models, by forwarding an image through the network, and then passing the results backwards through the network. This results in the images shown in Figure 13. For $\hat{\beta} = 16$ and $\hat{\beta} = 32$, the images are completely noisy, while the images for $\hat{\beta} = 1, 2, 4$ and 8 all seem to have roughly the same quality. We run the evaluation code by Mackowiak et al. [18], where we notice that we do not get the same results as stated in their paper. Table 3 gives an overview of the different results. This is likely due to incorrect checkpoint files for the models having been uploaded, which means that there are no fully trained models for us to use. Instead, we choose to train the $\hat{\beta} = 8$ model for an additional 20 epochs, which gives an accuracy of 67.85%. We choose $\hat{\beta} = 8$, because the inverted image seems very similar in quality as $\hat{\beta} = 1, 2$ and 4, but it is supposed to have a higher accuracy when trained correctly.

We use interpolation, as often used to assess GAN models [5], to test whether our retrained IBINN model can also invert latent spaces that are not direct outputs from existing images. We perform interpolation between two images from the Imagenet dataset, for our retrained $\hat{\beta} = 8$ model, and the not fully trained $\hat{\beta} = 8$ and $\hat{\beta} = 2$ models for comparison. We use 15 steps in the interpolation to see whether a latent space close to an existing image can provide a sensical image when a bigger change in latent space cannot. For each model, interpolation is done on both z_{fc} and z_{conv}

$\hat{\beta}$	Accuracy Mackowiak	Our accuracy
1	67.30%	25.30%
2	71.73%	0.56%
4	73.69%	0.21%
8	74.59%	0.12%
16	75.54%	0.76%
32	76.18%	36.81%

Table 3: Accuracy for the different values for $\hat{\beta}$ as stated by Mackowiak et al. [18] and as found by us by using their evaluation code.

and for only z_{fc} where all features in z_{conv} are set to zero, since the prototypes in our model will only consist of z_{fc}. Figure 14 shows our interpolation results. We see that our retrained model can only produce sensical images for invertion of the latent space of the actual image, while the not fully trained models do still produce somewhat sensical images. However, since the prediction accuries of these models are below 1%, we cannot use these models. When z_{conv} is set to zeros, even the latent spaces of the original images do not invert back to sensical images. For the untrained $\hat{\beta} = 2$ model, the invertion of z_{fc} for all steps in the interpolation results in images that are not completely noisy, but at least contain some information about the colour of the image.

We use the not fully trained $\hat{\beta} = 2$ model to test other possibilities to use as z_{conv} for future studies, if an INN with both sufficient prediction accuracy and invertibility might become available. Figure 15 shows a latent space inversion, where z_{conv} is taken from the latent space of a random segment, while z_{fc} is taken from a cluster center. This shows that z_{conv} contains almost all detail information about an image, since the noisy colour background comes from the cluster center, while the grass segment lying on top of it comes from the random segment. It could be a possibility to, instead of a random segment, use z_{conv} from the medoid. However, it might not have a big advantage compared to only using the cluster medoid without the actual cluster center.

C Segment types

The segments that are generated by SLIC Superpixels are often not rectangular. Since an INN needs rectangular input of a specific size, we need to add padding to the created superpixels. In ACE [8], the superpixels are padded with the default zero value of the model, which is a specific shade of grey. Another possibility is to blur the surroundings of the superpixel within its bounding box to keep some of the surroundings information, but remove details. A third option is to use the bounding box of the superpixel to cut out a rectangular image patch, which thus not only shows the superpixel but also some of its surroundings. Some examples for these three types of segments can be found in Figure 16.

To decide which segment type to use, we run some experiments on a small subset of the dataset, which consists of 8 classes (list can be found in Table 4), to compare the performance of the different models. These models are trained on

Figure 13: Results of forwarding images through IBINN, and then passing the output through the network in reverse, for the different available $\hat{\beta}$ models.

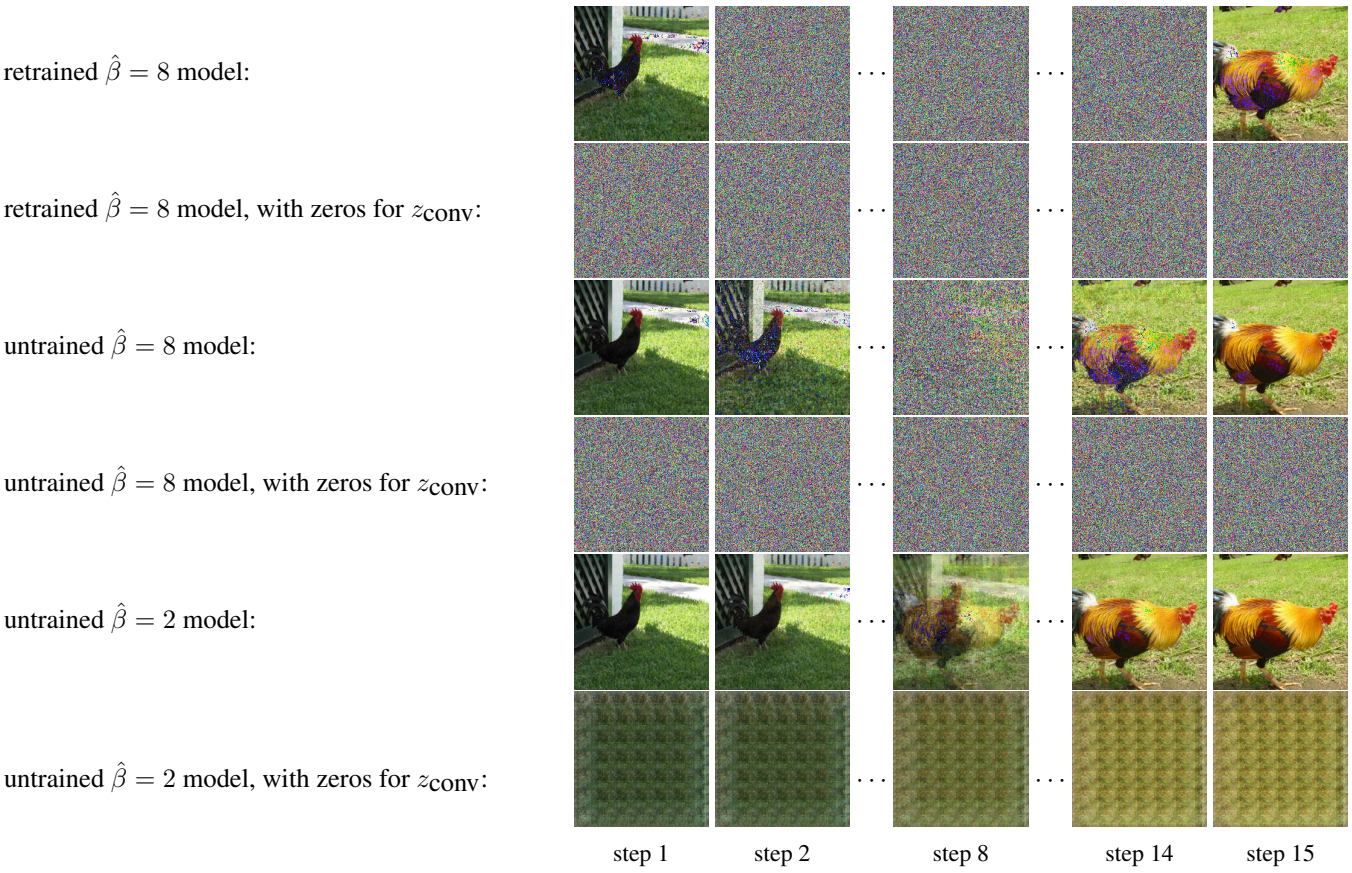


Figure 14: Interpolation results, for our retrained $\hat{\beta} = 8$ model, and the not fully trained (named 'untrained') $\hat{\beta} = 8$ and $\hat{\beta} = 2$ models. Interpolation is done between two images from the Imagenet dataset, with 15 steps. For each model, interpolation is done on both z_{fc} and z_{conv} and for only z_{fc} where all features in z_{conv} are set to zero.

Figure 15: Inversion of latent space through IBINN, where z_{fc} is a cluster center and z_{conv} is that of the latent space of a random segment.

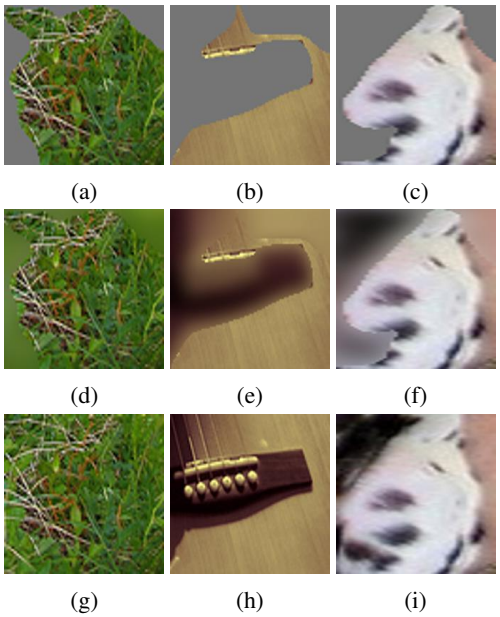


Figure 16: Different segment types for 3 example superpixels. The first row (a-c) shows segments where the bounding box of the superpixel is filled with grey, as used in ACE [8]. In the second row (d-f), segments where the bounding box is filled with a blurred version of the segment patch are shown. The third row (g-i) shows segments where the bounding box of the found superpixel is used to cut out a rectangular patch from the image.

class code	class name
n02106662	German shepherd
n02110341	Dalmation
n02123045	Tabby cat
n02123597	Siamese cat
n02676566	Accoustic guitar
n03272010	Electric guitar
n03445924	Golf kart
n04285008	Sports car

Table 4: Classes used for experiments for different segment types.

Segment type	Accuracy	Epochs
Grey background	57.67%	173
Blurred background	65.75%	309
Rectangular patches	82.50%	309

Table 5: Average accuracies and number of epochs until convergence of 3 runs, for three types of segments. Run on 8 classes, with 200 clusters.

class labels, with the number of clusters set to 200.

Table 5 shows the results from these experiments. The patch segments give significantly better results than the other two segments, and the blurred background segments give better accuracy than grey background segments. We hypothesize that this is because grey background segments are the least similar to the images that the black box INN was trained on, and patches are the most similar. Because patch segments have the highest accuracy, we will use these for the rest of our experiments.

D Target types

Since the blackbox IBINN model was already trained on the dataset, we hoped that using the knowledge from the model could help improve or speed up the training of our prediction layer. Therefore, we did some experiments where we trained the model on different types of targets to see whether using the IBINN output together with, or instead of, the actual labels would indeed be better than using only the labels.

We train the model on the following targets: 1) the actual label, 2) only the IBINN prediction scores, 3) on a multiplication of the two, which gives 0 for everything except the actual label, which gets the prediction score of IBINN instead of 1. 4) A weighted balance between the label and IBINN prediction.

Table 6 shows that training on the actual label reaches the highest accuracy, with the weighted target of 0.4*IBINN-output + 0.6*label as a close second. Since training on the actual label not only has a slightly higher accuracy, but also uses less epochs to reach this, we use this as the target to train our model on. Thus, our hypothesis that using the IBINN model's pretrained knowledge could improve or speed up the training of our model seems incorrect.

Target type	Accuracy	Epochs
l	**72,3**	**195**
bb	48	87
bb*l	63,5	526
0.5*bb + 0.5*l	66	253
0.4*bb + 0.6*l	**71**	**239**
0.2*bb + 0.8*l	66,5	75

Table 6: Evaluation accuracies for models trained with 400 clusters, with different targets. Epochs illustrates the number of epochs needed to reach convergence. 'bb' stands for the IBINN blackbox output, and 'l' for the actual label.

Clusters	Accuracy	Epochs
50	59,5	311
100	64,4	766
200	68,0	480
400	**72,3**	**195**
800	**76,3**	***799***

Table 7: Evaluation accuracies for models trained on different numbers of clusters. All models are trained on the actual labels. Epochs illustrates the number of epochs needed to reach convergence.

E Number of clusters

K-Means clustering needs a set number of desired clusters as input. Since we do not know the perfect number of clusters needed for our dataset, we run experiments for 50, 100, 200, 400 and 800 clusters to find the best number. Table 7 shows the results of these experiments, and Figure 17 shows that there is a relation between the accuracy and the number of clusters. More clusters leads to a higher accuracy. However, we also need to keep in mind the size of our explanations, which will drastically increase when increasing the number of clusters, as well as the memory usage and runtime which will also increase with an increase in clusters. Therefore, we choose to run the experiments for our paper with 400 clusters, which gives a high accuracy of 72.3%, and only uses 195 epochs to reach this.

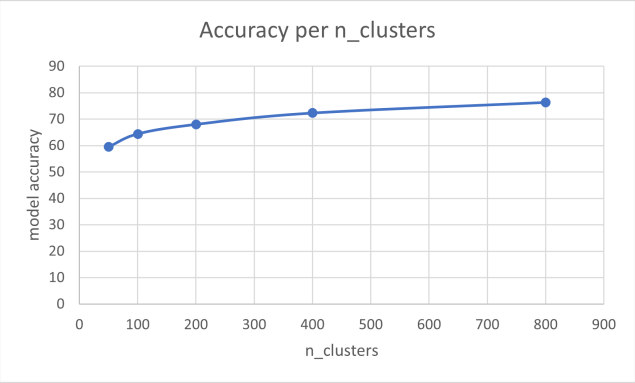


Figure 17: Relation between accuracy and number of clusters.

Centroid/medoid	Normalized	d	Accuracy
Centroid	no	-	8.75%
Centroid	yes	1	8.75%
Centroid	yes	2.5	12.50%
Medoid	yes	2.5	16.75%

Table 8: Prediction accuracy for the model with loss-based weights for different normalizations. In case of normalization, Equation 6 is used.

Weight	Accuracy
Count	43.25%
Equation 5	61.0%

Table 9: The prediction accuracy for the count-based weights model, with and without normalization.

F Weight calculation

Besides training weights through a linear layer, we also try creating weights based on already present information. For this, we use the 8-class data subset from Table 4, with 400 prototypes. We calculate weights based on: 1) the IBINN prediction loss for the cluster center, 2) the number of segments of a given class in the cluster, 3) tf-idf; where we use a cluster as the document and a class as the term. For 1) and 2), we also try some normalizations on the loss and segment count to see whether that improves the prediction accuracy.

Since IBINN is already trained on the data, we explore whether we can use its prediction results for the cluster centroids or medoids as weights. We do this for only the prediction loss, as well as the loss normalized by the size of the cluster. We use Equation 4, where we use the distance between the cluster centroid and the most outlying segment that belongs to that cluster for the cluster's $size$. d is an arbitrary value to scale the cluster size down, which ensures that only prototypes with smaller losses to the class get a high weight. Table 8 presents the results of the models with loss-based weights.

$$w = \exp(-\text{loss}) * \exp(-\text{size}/d) \tag{4}$$

We also try weights, where the weight between a prototype and a class is based on the number of segments of that class in that cluster. For the normalization, we use Equation 5 where the first part creates a function as seen in Figure 18. The numbers are found by looking at the plot and the observed class counts for the prototypes. For these weights, we also use $\exp(-\text{size}/d)$ with $d = 2.5$ to scale down the number of large weights. Table 9 shows the prediction accuracy for the count-based weights model, with and without normalization.

$$w = 1/(1 + \exp(-\text{count}/8 + 6)) * \exp(-\text{size}/d) \tag{5}$$

We also calculate weights based on the 'term frequency-inverse document frequency' (tf-idf), shown in Equation 6 [19], where tf is the number of segments of a given class in the given cluster, df is the number of clusters that contain

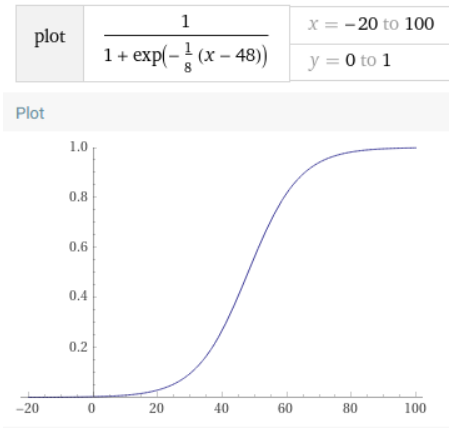


Figure 18: Plot of the first part of Equation 5, which is used for normalizing the count-based weights.

segments of the class, and N is the total number of classes. Using the results from this equation as weights, we get an evaluation accuracy of 22.0%.

$$\text{tf-idf} = \text{tf} * \log(N/\text{df}) \tag{6}$$

The tf-idf based weights result in a predicton accuracy of 22.0%.

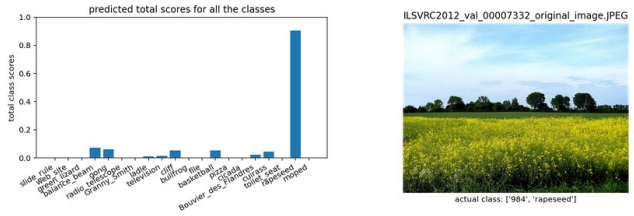
Because of the low prediction accuracies in these experiments, we choose to train the weights as described in Section 3.6 of the paper.

G Blurring most important segment

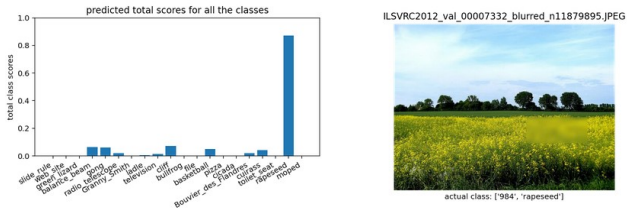
Our local visualisations show which prototype contributed most towards the high prediction of the class with the highest score. The input image segment that is shown as closest to this prototype, can be interpreted as stating that that prototype is present in the image. We test whether blurring this segment will result in a different class getting the highest prediction score. We use the same 20 classes from Table 2, with 5 images per class randomly selected from the evaluation data, resulting in 100 images overall. This experiment results in only 27% getting a different class with the highest prediction than before the blurring. This might be due to the second closest segment often being really similar to the first closest segment, and thus having a very small change in similarity score for the prototype after blurring the segment. Figure 19 and 20 show instances where the predictions do not change to another class after blurring the most important segment.

H Adding important segments of other classes

Figure 21 and 22 show some extra examples of important segments belonging to another class than the originally highest predicted class being added to the image, and how that affect their prediction scores. Figure 21a shows an image of a balance beam with low scores for all classes. By adding the most important prototype for the class 'Granny Smith',

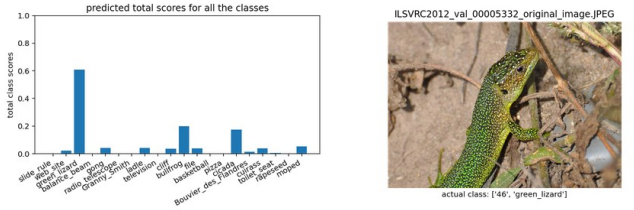


(a) An image of class 'rapeseed', which scores really high for its actual class.

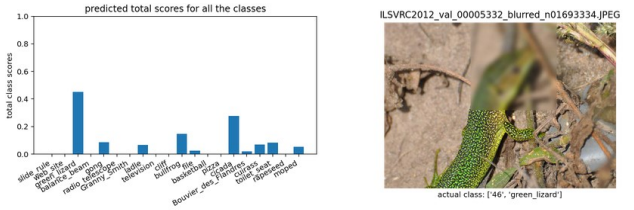


(b) An image of class 'rapeseed' where the segment, which is closest to the most contributing prototype for its prediction as 'rapeseed', is blurred. The prediction scores seem identical.

Figure 19: Blurring a part of this 'rapeseed' image does not seem to influence the prediction scores. This is probably due to many segments of the image being similar to the blurred image, and thus having minimal change in the similarity scores of the prototypes that the segment used to be closest to.

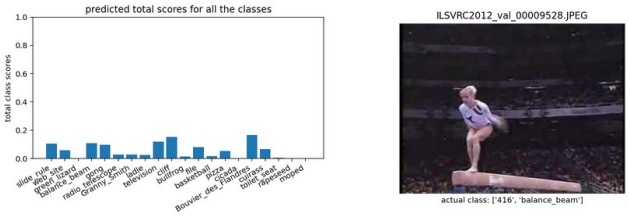


(a) An image of class 'green lizard', which scores really high for its actual class.

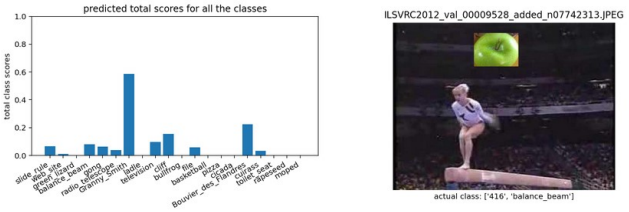


(b) An image of class 'green lizard' where the segment, which is closest to the most contributing prototype for its prediction as 'green lizard', is blurred. The prediction score for 'green lizard' has decreased, but is still higher than that of the other classes.

Figure 20: Blurring an important part of this 'green lizard' image decreases the prediction score for that class, but not enough to change the highest prediction to another class.



(a) The original image does not have the highest prediction score for its actual class; balance beam, and has low scores for all classes.



(b) The image augmented with the most important prototype for the class 'Granny Smith' has a high score for that class, and still low scores for all the other classes.

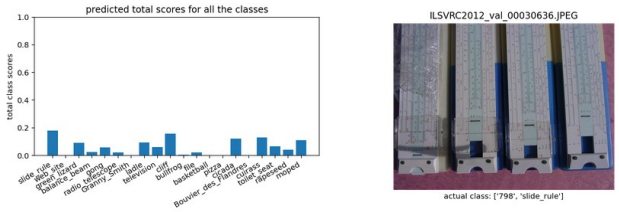
Figure 21: By adding the most important prototype for the class 'Granny Smith', the prediction score for this class increases drastically, while the scores for the other classes stay approximately the same.

the prediction score for this class increases drastically, while the scores for the other classes stay approximately the same (Figure 21b). The original image in Figure 22a has the highest prediction score for its actual class; slide rule, but not significantly higher than the scores for the other classes. The prediction scores for both 'basketball' and 'pizza' are very close to zero. Adding the most important prototype for the class 'basketball' has an increased score for that class, but still lower than that for 'slide rule' (Figure 22b), but adding the most important prototype for the class 'pizza' gives a much higher score for 'pizza' than for all other classes (Figure 22c).

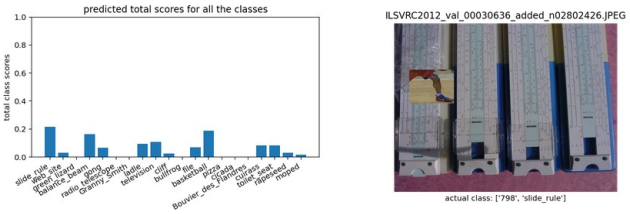
I Runtime and memory usage

In this section, we show some statistics about the produced code. We show the memory usage for some large and important functions, and the longest runtimes. Table 10 shows the memory usage for the tasks in the training phase of the model, Table 11 shows that for the performance evaluation, and Table 12 shows the memory usage for the interpretability evaluation, which blurs and augments images as explained in Appendices G and H.

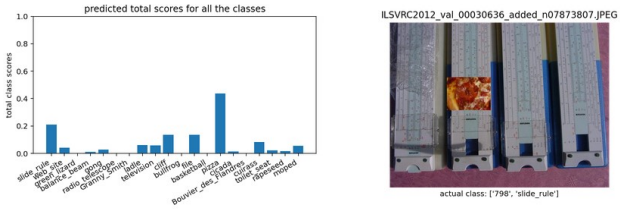
When running our code on a NVIDIA RTX A6000 GPU, the runtime for segmenting the images is around 1.5 minutes for 50 images. For 20 classes with 1300 images each, this sums to around 13 hours of segmentation. When running the training phase for 20 classes with pre-segmented ImageNet images (1300 per class), the runtime can be around 6.5 hours, of which 2.5 is for getting the segment activations, and 2.5 for creating the 400 clusters.



(a) The original image has the highest prediction score for its actual class; slide rule, but not significantly higher than the scores for the other classes. The prediction scores for both 'basketball' and 'pizza' are very close to zero.



(b) The image augmented with the most important prototype for the class 'basketball' has an increased score for that class, but still lower than that for 'slide rule'.



(c) The image augmented with the most important prototype for the class 'pizza' has a much higher score for that class than for all other classes.

Figure 22: Adding the most important prototype for the class 'basketball' has less impact on the final prediction than adding the most important prototype for the class 'pizza'.

Memory usage	Task
303.1 MiB	loading segments as dataset
3480,5 MiB	loading IBINN model
163,234.3 MiB	getting segment activations
78.8 MiB	saving activations to dictionary
763.1 MiB	creating concepts / clusters
554.4 MiB	saving concept dictionary to pickle file
157.4 MiB	getting similarity scores
5625.5 MiB	getting evaluation segment activations
469.4 MiB	getting evaluation similarity scores
6.5 MiB	training prediction layer until convergence
175,379.9 MiB	complete training phase with pre-segmented images

Table 10: Memory usage of the important or large functions in the training phase.

Memory usage	Task
14 MiB	loading segments as dataset
6424.5 MiB	loading eval segment activations
679.6 MiB	loading concepts
149.0 MiB	getting similarity scores
0.4 MiB	loading prediction layer
0.9 MiB	running prediction
7535.8 MiB	complete performance evaluation with pre-segmented images

Table 11: Memory usage of the important or large functions of the predictive performance evaluation.

Memory usage	Task
1171.7 MiB	segmenting and loading as dataset
3316.5 MiB	loading IBINN model
202.4 MiB	getting segment activations
654.9 MiB	loading concepts
105.9 MiB	getting similarity scores
158.8 MiB	adding blur and prototypes to images
162.6 MiB	getting predictions of new images
6026.9 MiB	complete interpretability evaluation

Table 12: Memory usage of the important or large functions in the interpretability evaluation.