This Looks Like *That There*: Interpretable neural networks for image tasks when location matters

Elizabeth A. Barnes,[a] Randal J. Barnes,[b] Zane K. Martin,[a] and Jamin K. Rader,[a]

[a] *Department of Atmospheric Science, Colorado State University, Fort Collins, CO, USA.*

[b] *Civil, Environmental, and Geo- Engineering, University of Minnesota, Minneapolis, MN, USA.*

Corresponding author: Elizabeth A. Barnes, eabarnes@colostate.edu

ABSTRACT: We develop and demonstrate a new interpretable deep learning model specifically designed for image analysis in earth system science applications. The neural network is designed to be inherently interpretable, rather than explained via *post hoc* methods. This is achieved by training the network to identify parts of training images that act as prototypes for correctly classifying unseen images. The new network architecture extends the interpretable prototype architecture of a previous study in computer science to incorporate absolute location. This is useful for earth system science where images are typically the result of physics-based processes, and the information is often geo-located. Although the network is constrained to only learn via similarities to a small number of learned prototypes, it can be trained to exhibit only a minimal reduction in accuracy compared to non-interpretable architectures. We apply the new model to two earth science use cases: a synthetic data set that loosely represents atmospheric high- and low-pressure systems, and atmospheric reanalysis fields to identify the state of tropical convective activity associated with the Madden-Julian oscillation. In both cases, we demonstrate that considering absolute location greatly improves testing accuracies when compared to a location-agnostic method. Furthermore, the network architecture identifies specific historical dates that capture multivariate, prototypical behaviour of tropical climate variability.

SIGNIFICANCE STATEMENT: Machine learning models are incredibly powerful predictors, but are often opaque "black boxes". The how-and-why the model makes its predictions is inscrutable — the model is not interpretable. We introduce a new machine learning model specifically designed for image analysis in earth system science applications. The model is designed to be inherently interpretable and extends previous work in computer science to incorporate location information. This is important because images in earth system science are typically the result of physics-based processes, and the information is often map based. We demonstrate its use for two earth science use cases and show that the interpretable network exhibits only a small reduction in accuracy compared to black box models.

1. Introduction

Machine learning has been identified as an innovative, under-explored tool for furthering understanding and simulation of the Earth system (Balmaseda et al. 2020; Irrgang et al. 2021; National Academies of Sciences Engineering and Medicine 2020). Artificial neural networks (as a type of supervised machine learning) have emerged as a powerful tool for extracting nonlinear relationships amidst noisy data, and thus are particularly suited to this endeavor. However, a major criticism of the use of neural network models for scientific applications is that they are "black boxes." Scientists typically want to know why the model reached the decision that it did. The benefit of explaining the decision-making process of a model goes beyond that of satisfying curiosity: explanation can assist users in (1) determining if the model is getting the right answers for the right reasons (e.g. Lapuschkin et al. 2019), (2) controlling and improving the machine learning approach (e.g. Keys et al. 2021), and (3) discovering new science (e.g. Toms et al. 2020; Barnes et al. 2020). Effective explanations also increase user confidence.

Because researchers are driven by the desire to explain the decision-making process of deep learning models, a large variety of *post hoc* explainability methods have been developed (e.g. Buhrmester et al. 2019; Barredo Arrieta et al. 2020; Samek et al. 2021). By *post hoc*, we mean methods in which a deep learning model has already been trained and the user attempts to explain the predictions of the black box model after the predictions have been made. Although *post hoc* explainability methods have demonstrated success across many scientific applications (including earth system science, e.g. McGovern et al. (2019); Toms et al. (2020); Davenport and Diffenbaugh

(2021)), they are not without their drawbacks. *Post hoc* explainability methods do not exactly replicate the computations made by the black box model. Instead, through a set of assumptions and simplifications, these methods quantify some reduced version of the model (e.g. Montavon et al. 2018) and, thus, do not explain the actual decision-making process of the network. Furthermore, the explanations are not always reliable (Kindermans et al. 2019). Different explanation methods can produce vastly different explanations of the exact same black box model (Mamalakis et al. 2021, 2022). Even if the explanation is reliable, at times the output of the explainability method itself requires extensive deciphering by the scientist to understand the result (e.g. Mayer and Barnes 2021; Martin et al. 2021; Barnes et al. 2020). Rudin (2019) discusses in detail many of these potential issues with explainable machine learning methods and suggests that we should instead be using machine learning models that are inherently "interpretable". That is, instead of trying to explain black box models, we should be creating models where the decision-making process is interpretable by design.

Chen et al. (2019) present an example of one type of interpretable neural network, the *prototypical part network* (ProtoPNet). The ProtoPNet hinges on training a neural network to identify patches of the training images that act as "prototypes" for correctly classifying unseen images. The idea for the ProtoPNet stems from the need to define a form of interpretability that works the way a scientist might describe their way of thinking. In their specific application, Chen et al. (2019) focus on classifying images of birds by their species. A scientist may classify a new bird image by comparing it to representative examples of each species (i.e. species prototypes) and choosing the prototype that most resembles the image, i.e *this looks like that*. In this way, the network is inherently interpretable in that the *actual* decision-making process can be linked to specific features of the bird in the input image and their similarity to a relatively small number of species-specific prototypes that are directly drawn from the training set. For bird species identification, Chen et al. (2019) demonstrate that the ProtoPNet learns prototypes that represent distinguishing features such as the red head of a red-bellied woodpecker, or the bright blue wing of a Florida jay.

Images in earth system science are typically the result of physics-based processes, and the information is often geo-located. Thus, unlike the ProtoPNet of Chen et al. (2019) which does not care *where* the bird's wing is in the image, the location of specific earth system features can be critical to the final task (although this is certainly not always the case, e.g. identification of cloud

types from satellite imagery; Rasp et al. (2019)). For example, the mere presence of a low-pressure system on a weather map is not enough to know where it will rain. Instead, the location of the low — *where* it is — is also vital for this task. Similarly, identifying the presence of a strong El Niño requires not only warm sea-surface temperatures, but specifically warm sea-surface temperatures in the tropical equatorial east Pacific (e.g. Philander 1983). Here, we extend the ProtoPNet of Chen et al. (2019) to consider absolute location in the interpretable prototype architecture, which we call the ProtoLNet ("Prototypical Location Network"). We demonstrate that considering absolute location greatly improves the network accuracy (ProtoLNet rather than ProtoPNet) for two earth science use cases. The first use case, the idealized quadrants use case (Section 3), applies the ProtoLNet to a synthetic data set that loosely represents high- and low-pressure systems where the need for location information is readily apparent. The second use case applies the ProtoLNet to over 100 years of atmospheric reanalysis fields to identify the state of tropical convective activity associated with the Madden-Julian oscillation (MJO; Madden and Julian 1971, 1972; Zhang 2005). The MJO use case (Section 4) provides a real, geophysical example of how the ProtoLNet relies on location information to make its predictions and demonstrates how the learned prototypes can be viewed as prototypical behaviour of transient climate phenomena.

2. Network Design & Training

As discussed in the introduction, the ProtoLNet is largely based on the ProtoPNet of Chen et al. (2019). We describe the network architecture below, highlighting where our ProtoLNet diverges from the ProtoPNet of Chen et al. (2019). We then describe the training procedure in detail.

a. ProtoLNet architecture

The ProtoLNet is designed to classify images by comparing latent patches of the input image to prototypical latent patches learned from the training set, all while explicitly considering the location within the image of the similar latent patches. Throughout, we use the word "patch" to refer to a group of neighboring pixels within the input image, and "latent patch" to refer to a latent representation of a patch that is computed via a series of convolutional and pooling layers within the convolutional neural network. In this section, we first provide a general overview of

the ProtoLNet architecture from start to finish, and then go into more detail about each step in subsequent paragraphs, ending with the training process.

The ProtoLNet architecture (Fig. 1) is very similar to that of the ProtoPNet, and starts with a base convolutional neural network (CNN) chosen by the user that takes-in an image as input. As discussed more in Section c, this base CNN may be a pre-trained network, or a newly initialized network with randomized weights. The CNN is followed by two 1×1 convolutional layers that act to restructure the dimensions of the CNN output to be consistent with the subsequent prototype layer. It is within the prototype layer that the interpretable learning is done. The network is trained to learn representative latent patches within the training set specific to each class, termed *prototypes*, which provide evidence for the image belonging to a particular class. That is, when the input image has a patch whose latent representation *looks like that* prototype, it is labeled as belonging to the prototype's associated class. This is done by computing the similarity of each prototype to the latent patches of the input image. Unique to our ProtoLNet, these similarity scores are scaled by a learned, prototype-specific location scaling grid so that similarities to the prototypes are only important for certain locations within the input image. The maximum scaled similarity score across the latent patches for each prototype is then computed. These scores are connected to the output via a fully connected layer, and the weighted scores are summed for each output class to produce a total number of "points" for each class. The class with the highest number of points is then identified as the predicted class.

As will be discussed in detail in Section c, the ProtoLNet learns the convolutional kernels within the two 1×1 convolution layers, the prototypes, the location scaling grid, and the final fully connected weights (pink components in Fig. 1). The user must specify the number of prototypes specific to each output class. For the use cases presented here, we choose an equal number of prototypes for each class, so if there are n classes and p prototypes per class, then there are $m = n * p$ total prototypes. A critical aspect of the architecture is that *each prototype is assigned to only one class* since it is used as evidence that a particular sample belongs its class.

Each sample is pushed through the extended CNN, which results in an output "quilt" of latent patches. To introduce some general notation, the quilt has shape $a \times b \times D$, where $a \times b$ is the new image shape after undergoing pooling in the base CNN, and D corresponds to the number of convolutional kernels chosen by the user. Each prototype vector ($\mathbf{p}$) then has shape $1 \times 1 \times D$. To

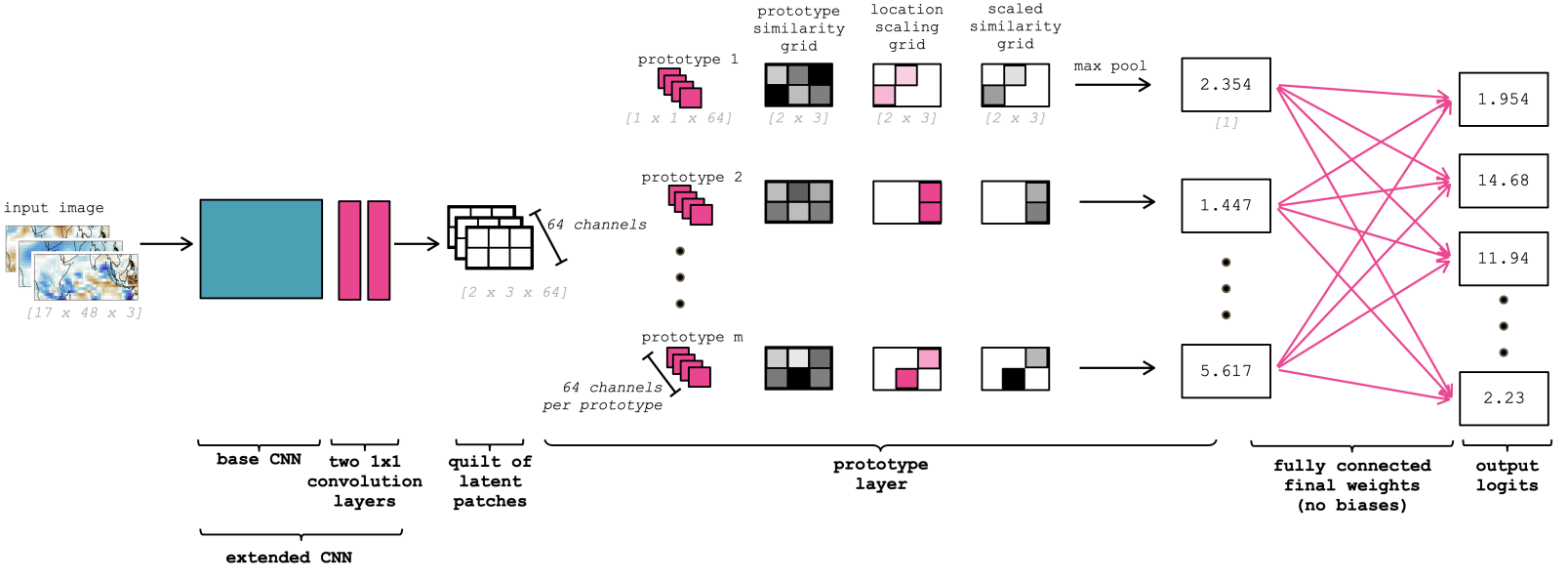


FIG. 1. Schematic depicting the ProtoLNet architecture. Example and internally consistent dimensions of the tensors at each step are given in grey brackets, although the specific dimensions vary for each use case. Pink colors denote components of the network that are trained (learned), while gray and black colors denote components that are directly computed. The weights within the base CNN (blue shading) can either be trained or frozen.

simplify our discussion, from here forward we will drop the general notation and instead use the specific dimensions (denoted in gray) of the example shown in Fig. 1. That is, $a = 2$, $b = 3$, and $D = 64$.

For the example in Fig. 1, a latent patch has shape $1 \times 1 \times 64$, and the quilt of latent patches output by the extended CNN has shape $2 \times 3 \times 64$. Because the input image has already potentially undergone multiple convolutional and pooling layers within the extended CNN, these latent patches *do not* represent a single pixel of the input image, but instead are a latent representation of some larger patch within the input image. Similar to the latent patches, each of the m learned prototypes are a latent representation of some larger region of the input image. Each prototype has the same shape as a latent patch: $1 \times 1 \times 64$. The similarity score for a prototype $\mathbf{p}$ and a latent patch $\mathbf{z}$ is computed as a function of the distance between these two vectors (i.e. the L_2 norm of the difference). The greater the distance between, the lower the similarity score. Following Chen et al. (2019), we compute

$$\text{SimilarityScore} = \log\left(\frac{\|\mathbf{z}-\mathbf{p}\|_2^2+1}{\|\mathbf{z}-\mathbf{p}\|_2^2+\epsilon}\right) \approx \log\left(1+\frac{1}{(\text{distance})^2}\right) \tag{1}$$

where $\| \|_2^2$ is the squared L_2 norm and ϵ is a small number, there to guard against divide-by-zero problems. Applying this similarity metric to a quilt of latent patches results in m 2×3 similarity grids, one for each prototype. The values within these grids thus quantify how much that latent patch of the input *looks like* each prototype.

In the original ProtoPNet, at this point the maximum similarity within each similarity grid is computed for each prototype. However, unique to our ProtoLNet — and indeed the novelty of this work — is that we scale each prototype's similarity grid by a location-specific value learned by the network. This step rescales the similarities such that similarities in certain locations are accentuated and similarities in other locations are muted. To follow this paper's title, it isn't enough for *this* latent patch (at any location) to look like *that* prototype. Instead, *this* latent patch must look like *that* prototype in only specific locations — *there*. This results in m location-scaled similarity grids, one for each prototype.

Once again following the architecture of the original ProtoPNet, we apply max pooling to each scaled similarity grid to obtain a single score for the maximum similarity (scaled by the location

scaling) between a prototype and the input image. These scores are then connected to the output layer via a fully connected layer with learned weights but zero bias. The choice of zero bias in the final fully-connected layer is essential for interpreting the prototypes as providing evidence for a particular class. With a zero bias, the final points contributing to each class are comprised only of a sum of location-scaled similarity scores multiplied by a final weight. The final weights layer is trained separately from the rest of the network. The layer is trained in such a way as to keep weights connecting prototypes with their associated class large, while minimizing the weights connecting prototypes with their non-class output units (see Section c). Finally, as is standard with a fully connected layer, the output values (weighted scores) contributing to each output unit are summed to produce a total number of points for each class. The class with the highest number of points is identified as the predicted class.

In the original ProtoPNet, there was no location scaling. Without this location scaling, the network is agnostic to *where* the input image looks most like each prototype. That is, the only thing of import is that the image looks like the prototype *somewhere*. Returning to the example of classifying bird images (as explored in Chen et al. (2019)), a prototype may correspond to a latent representation of the red head of a red-bellied woodpecker. The original ProtoPNet does not care whether a red head is found in the upper left or the upper right of the input image. Rather, the ProtoPNet just considers whether a red head is present at all. For our ProtoLNet presented here, the network is designed to take into consideration not only that a red head is found, but also *where* within the image the red head occurs. As we will show, this consideration of location can be highly beneficial in geophysical applications.

b. Choosing the base CNN

We envision three main approaches to choosing a base CNN. The first takes an existing CNN that has been previously trained to perform classification tasks. This CNN may already be performing well, but interpretability is desired. The user removes the output layer and fully connected layers of their existing CNN and then use the result as their base CNN for the ProtoLNet. In this approach, the ProtoLNet is used purely for interpretability of the original CNN.

The second approach to choosing a base CNN is to, once again, take a pre-trained CNN, remove the output and fully connected layers, and then use the result as the base CNN for the ProtoLNet.

Stage	Type	Base CNN	1x1 Layers	Prototypes	Location Scaling	Final Weights
1	train prototypes	frozen/**train**	**train**	**train**	**train**	frozen
2	replace prototypes	frozen	frozen	**replace**	frozen	frozen
3	train weights	frozen	frozen	frozen	frozen	**train**

FIG. 2. The three different stages of training the ProtoLNet.

The difference is that now the user allows the weights within the base CNN to be further refined during the ProtoLNet training in order to optimize the performance of the ProtoLNet. Allowing the base CNN weights to be updated implies that the user is no longer interpreting the same base CNN with which they started. However, if the goal is to create an interpretable network that is as accurate as possible, this may be a good approach. Furthermore, for image classification tasks, one might choose to use a CNN previously trained on a large dataset, e.g. VGG-19 (Simonyan and Zisserman 2014), as done by Chen et al. (2019).

The third approach to choosing a base CNN applies when no suitable pre-trained base CNN exists. In this case, the user must train the interpretable network from scratch. In this instance, there are two main choices. A separate base CNN could be trained, stripped of its final output and fully connected layers, and then appended to the ProtoLNet (as discussed above). Alternatively, one could initialize the base CNN with random initial weights and train it directly within the ProtoLNet architecture. We have tried both methods for the use cases explored here and found that they produced similar accuracies (although we acknowledge this may not always be the case). Here, we present results where we first pre-train a base CNN and then append it to the ProtoLNet, in order to provide a base accuracy with which to compare our ProtoLNet results.

c. ProtoLNet training

The training of the ProtoLNet is done in triads of stages (Fig. 2), largely following the original training approach of Chen et al. (2019). The first stage of training involves learning the prototypes by training the 1×1 layers, prototypes, location scaling grid, and the base CNN (if desired by the user; see Section b) at the same time. The final weights are frozen during this stage. The second stage of training involves replacing each prototype with the nearest latent patch within the training

samples of the same class. That is, stage 1 allows the network to learn *any* form of the prototype latent patch, and stage 2 replaces this prototype with the most similar training latent patch from the same class. In this way, the prototypes always directly correspond to a latent patch in one particular training sample. In the third stage of training, we freeze all elements of the ProtoLNet except for the fully connected final weights (pink arrows in Fig. 1), and the network learns them alone. These three stages are then cycled through multiple times (for our use cases, up to 5 times) for full training of the ProtoLNet.

Initialization: Prior to stage 1, the two 1×1 convolutional layers are initialized with random values drawn from a truncated normal distribution (He et al. 2015). The prototypes are initialized with random values drawn from a uniform distribution between 0.0 and 1.0, and the location scaling grid is initialized with ones everywhere (see Appendix B for additional details). The final weights (w) that connect a prototype with its assigned class are given an initial value of 1.0, and all other final weights are initialized to -0.5. The initialization of the base CNN was already discussed in Section b.

Stage 1: Training is performed via stochastic gradient descent with the Adam optimizer and batch size of 32. For the quadrants use case, the learning rate is set to 0.01 for every stage 1 cycle. For the MJO use case, the learning rate is also initially set to 0.01 but is reduced by an order of magnitude for the third cycle of stage 1 and every cycle thereafter. The network is trained with the standard cross-entropy loss (e.g. Géron 2019) added to two additional loss terms: the ClusterCost and SeparationCost. The cross-entropy loss penalizes the network for misclassifying the training samples. The ClusterCost encourages the network to construct prototypes such that training images have at least one latent patch with high similarity to a prototype of the correct class. The SeparationCost discourages the network from constructing prototypes such that training images have any latent patches with a high similarity to prototypes of the incorrect classes. Thus, the full stage 1 loss function takes the form

$$\text{Loss} = \text{CrossEntropy} + \beta_1 \text{ClusterCost} - \beta_2 \text{SeparationCost} \tag{2}$$

where β_1 and β_2 are coefficients chosen by the user. Full forms of the ClusterCost and SeparationCost, along with their coefficient values, are provided in Appendix C. For all use cases, we train in stage 1 for 10 epochs before moving to stage 2 of training.

Stage 2: This stage does not involve any iterative training but instead is direct computation. Specifically, the similarity scores are computed between each learned prototype from stage 1 and every latent patch of every training image of the same class. The prototype is then *replaced* by the training latent patch with the highest similarity. Note that this replacement process will nearly always reduce the accuracy of the network because it replaces the stage 1-optimized prototypes with something from the training set. However, this step is central to the interpretability of the ProtoLNet. By cycling through all three training stages multiple times, the network learns to perform well using the replaced prototypes from the training set.

Stage 3: The final weights $w_{k,j}$ connecting prototypes of class k to the output class j are learned via convex optimization, since all other layers are frozen. As a reminder, all $w_{k,j}$ for $k = j$ are initialized to 1.0, and the rest, $w_{k,j}$ for $k \neq j$, are initialized to -0.5. The weights are frozen for stages 1 and 2 of training. In stage 3, all other free parameters in the ProtoLNet are frozen, and the weights alone are trained to minimize the cross-entropy loss of the final output plus an additional L_1 regularization term evaluated on the weights $w_{k,j}$ for $k \neq j$. This additional loss term provides sparsity to the final model, i.e. $w_{k,j} \approx 0$ for $k \neq j$, which reduces the use of negative reasoning by the network ("this does *not* look like that"). See Singh and Yow (2021) for an exploration of the consequences when this sparsity requirement is relaxed. For the idealized quadrants use case, we set the regularization parameter to 0.5. For the MJO use case, it is set to 0.1. For all use cases, we train in stage 3 for 10 epochs. At that point, we either end training completely (i.e. we have the fully trained ProtoLNet), or we cycle through stages 1-3 again.

3. Use Case: Idealized Quadrants

As a first demonstration of the ProtoLNet, we construct an idealized synthetic test set to loosely represent the horizontal (latitude by longitude) spatial structures of geophysical anomalies. For example, the synthetic fields (or images) could represent idealized low- and high-pressure circulations. The anomaly fields are 100x100 pixels in size and are constructed by first initializing the field with random Gaussian noise. We then randomly add an additional anomaly value (uniformly distributed between 2 and 15) to the center of one or more of the four quadrants of each square field. Finally, we smooth each field with a Gaussian filter with standard deviation of 7 to make

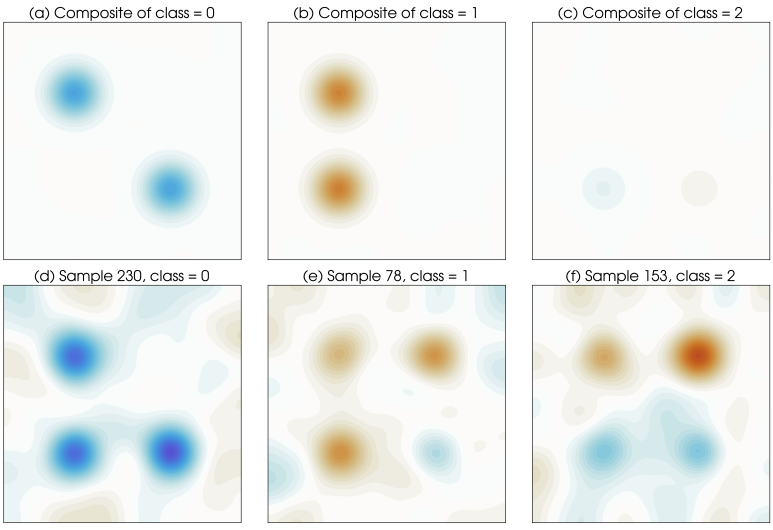


FIG. 3. The top three panels (a-c) show composites of all samples by class label for the idealized quadrants use case. The bottom three panels (d-f) exhibit one example sample for each class.

the fields look more like typical tropospheric pressure anomalies. Example samples are shown in Fig. 3.

The fields in the idealized data set are assigned labels based on the sign of the anomalies in each of the four quadrants of the sample (Fig. 3). Specifically, fields with negative anomalies in both the second and fourth quadrants are labeled class 0, fields with positive anomalies in both the second and third quadrants are labeled class 1, and all other fields are labeled class 2 (Fig. 3a-c). Fig. 3d-f show example samples for each class. As designed, sample #230 (labeled class 0) has negative anomalies in the second and fourth quadrants, sample #78 (labeled class 1) has positive anomalies in the second and third quadrants, and sample #153 (labeled class 2) does not achieve either of the requirements of classes 0 or 1. As will become clear, this idealized data set was designed such that the location of the different anomalies matters.

The synthetic data set has equally balanced classes by construction, with 3,000 samples for each of the three classes (9,000 samples total). This set is then randomly split such that 7,200 samples are used for training and 1,800 for testing. Prior to training, the input images are standardized by subtracting the mean and dividing by the standard deviation over all training pixels.

We task the ProtoLNet with ingesting a single input field and classifying it into one of the three classes, as depicted in Fig. 4. The network cannot simply identify the existence of negative

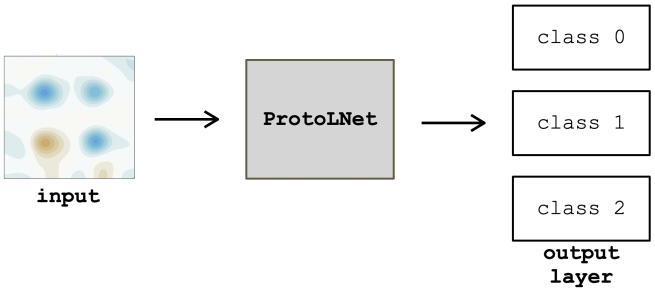


FIG. 4. Prediction setup for the idealized quadrants use case.

anomalies (in the case of class 0) or the existence of positive anomalies (in the case of class 1). Instead, it must consider the existence of different signed anomalies *and their location* within the input field. To illustrate this point, we trained a ProtoPNet where location is not considered (i.e. learning of the location scaling grid is turned off) and — unsurprisingly — the network fails with an accuracy of 32%, no better than random chance (i.e. 33%).

We first train a standard CNN to perform the classification task and act as our base CNN for the ProtoLNet. Details of the CNN architecture and training parameters are provided in Appendix A. Once the CNN is trained, we remove the final fully connected layer and output layer, and append the result to the ProtoLNet to become the base CNN (see Fig. 1). We assign 5 prototypes (with $D = 128$) to each output class, for a total of 15 prototypes. Using more prototypes than this yielded prototypes that rarely provided points for any sample. We cycle through the three stages of ProtoLNet training (Fig. 2) five times, freezing the base CNN for the first cycle of stage 1 but allowing it to train for all subsequent cycles of stage 1. Once fully trained, the ProtoLNet achieves an accuracy of 96%, a significant improvement over random chance and the ProtoPNet. For comparison, the base CNN achieves an accuracy of 98%. The ProtoLNet is not designed to outperform all alternative approaches. Instead, it is designed to provide interpretability with a minimal loss in accuracy.

The power of the ProtoLNet is that once trained, its decision-making process can be interpreted by the user. Three example predictions are shown in Fig. 5, along with their two "most winning" prototypes (i.e. prototypes that gave the most points to the winning class in each example) and the associated location scaling grids. To avoid any confusion, we want to clearly state that the "prototypes" outlined in colored boxes in Fig. 5(i),(iii) are not the prototypes themselves. The actual

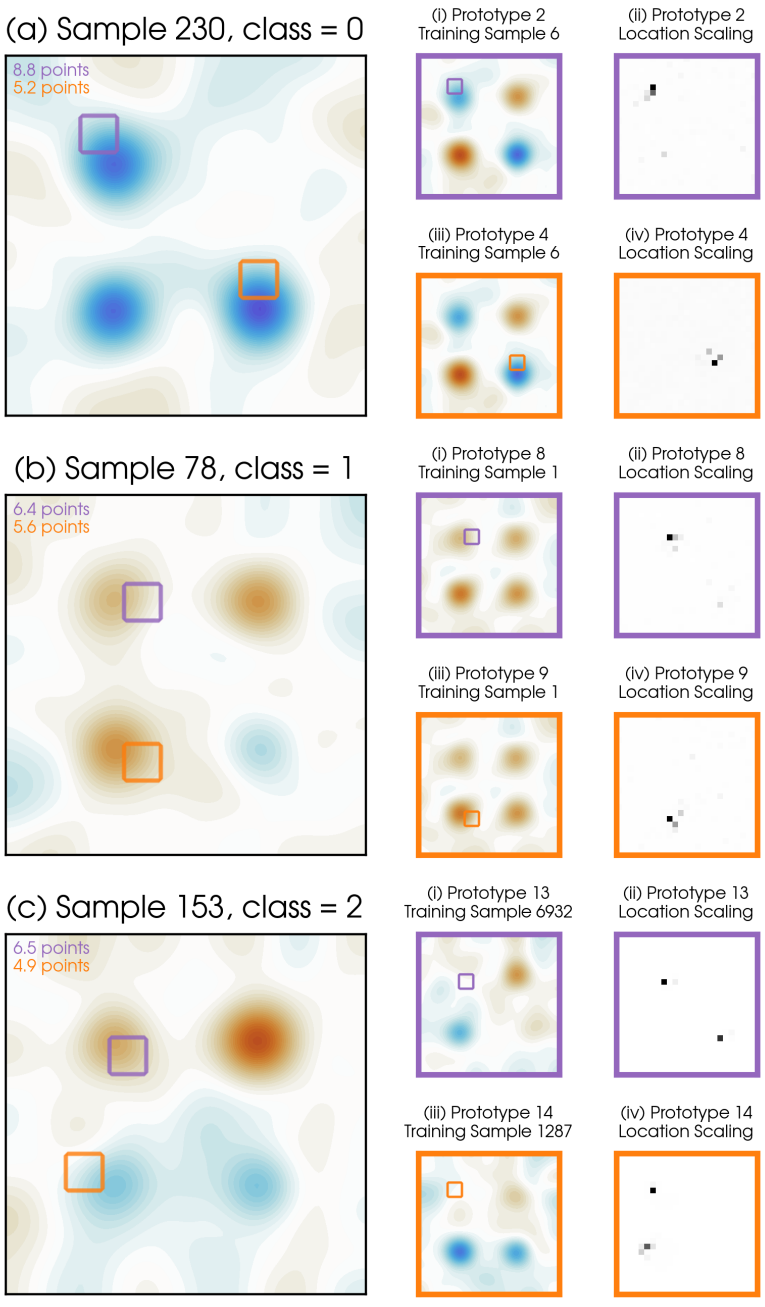


FIG. 5. Three example predictions by the network for the idealized quadrants use case, along with the two winning prototypes for each sample and the associated location scaling grid. For each of the three samples, there are two prototypes shown along with their associated location scaling grids. These are indexed as (i,iii) and (ii,iv), respectively.

prototypes are vectors of latent patches of size $1 \times 1 \times 128$ and would likely be incomprehensible since they capture the output of a series of complex convolutions, poolings, and nonlinear activations. Instead, we visualize the group of neighboring pixels of the training field that contribute to the prototype latent patch, often termed the "receptive field". In contrast, the location scaling panels in Fig. 5(ii),(iv) display the actual grids used in the prototype layer computation, which is why the squares are much larger than the pixels in the input field (i.e. the dimensions have been reduced to 25×25).

Consider Sample 230 (Fig. 5a), which the ProtoLNet correctly labeled as class 0. Prototypes 2 and 4 contributed the most points to a class 0 prediction, giving 8.8 and 5.2 points, respectively. Prototype 2 was drawn from training sample 6 and, more specifically, Prototype 2 represents a latent patch from the purple-boxed region of training sample 6 (Fig. 5(ai)). The location scaling grid for Prototype 2 (Fig. 5(aii)) shows that this prototype is highly relevant only when found in the upper-left corner of the field (dark gray and black pixels). Thus, the ProtoLNet identified high similarity between Prototype 2 and an upper-left patch of Sample 230. Or in other words, the ProtoLNet identified that sample 230 *looks like that* prototype *there*.

Prototype 4 (Fig. 5(aiii),(iv)) also contributed points to the correct prediction of class 0. Note that Prototype 4 was also drawn from training sample 6; coincidentally the same sample as Prototype 2. Looking at Prototypes 2 and 4 together, one can interpret that the network's decision-making strategy is to look for blue anomalies in the upper-left and bottom-right quadrants of the image — which is exactly how class 0 is defined. A similar interpretation can be found for sample 78 (Fig. 5b) with a class label of 1. The network identifies the class 1 sample by looking for positive anomalies in the upper-left and bottom-left quadrants.

The network's decision-making strategy is particularly interesting for Sample 153 with a label of class 2 (Fig. 5c). Prototype 13 corresponds to features associated with a weakly positive anomaly in the upper-left or bottom-right quadrants. From this, it appears that the network is ruling out a class 0 sample, which exhibits negative anomalies in these quadrants. Similarly, Prototype 14 corresponds to features associated with a weakly negative anomaly in the upper-left or bottom-left quadrants. That is, the network rules out a class 1 field that exhibits strong positive anomalies in these two quadrants. Fig. 5(cii),(iv) nicely demonstrates that the location scaling grid can highlight multiple locations throughout the field for the same prototype. The interpretability of the

ProtoLNet prediction thus allows for identification of the patches of the input field that were used to make the prediction, i.e. the patches whose latent representation most looks like class-specific prototypes learned during training.

One interesting observation is the sparsity of the location scaling grids in (Fig. 5) despite no explicit sparsity requirement in the loss function. This comes about due to the SeparationCost (Eq. 2) pushing the values of the location scaling grid to lower values in unimportant areas. Since the SeparationCost is subtracted in the loss function, and the location scaling values (s_k) appear in the denominator of the SeparationCost, the gradient of the loss function ultimately favors small location scaling values for unfavorable prototypes.

4. Use Case: MJO Phase Classification

We next apply the ProtoLNet architecture to earth system reanalysis fields. Specifically, the network is tasked with ingesting maps of atmospheric fields in the tropics and predicting the current phase of the Madden-Julian oscillation (MJO). The MJO is a large-scale, eastward propagating coupling between tropical wind and convection that oscillates on subseasonal (30-60 day) timescales (Madden and Julian 1971, 1972; Zhang 2005). Canonical MJO events form in the Indian Ocean, and propagate east into the western Pacific: the "phase" of the MJO describes roughly where it is in this life cycle.

The task of classifying the current phase of the MJO from maps of the tropics is chosen here to demonstrate the utility of our method to a relatively straightforward climate science task. Classification of MJO phase requires the network to identify coherent, multivariate tropical patterns on a particular (planetary) spatial scale, and the MJO's eastward propagation also requires the network to take advantage of spatial location in its decision making. Thus, while straightforward from a scientific perspective, the task of classifying MJO phase is well-suited as a demonstrative use-case for the ProtoLNet methodology. Toms et al. (2021) classified the state of the MJO to explore the utility of explainability methods, in contrast to our interpretable method, for earth system science applications.

We define MJO activity and phase using the "Real-time Multivariate MJO index" (RMM; Wheeler and Hendon (2004)). RMM is derived through an empirical orthogonal function (EOF) analysis of three variables: outgoing longwave radiation (OLR), 200 hPa zonal wind (u200) and

850 hPa zonal wind (u850). Each variable in RMM is pre-processed by removing the seasonal cycle (i.e. the all-time mean and first three harmonics of the annual cycle on each calendar day), and the previous 120-day mean of each day (to remove variability associated with longer timescales than the MJO). Variables are averaged from 15N-15S, and the leading two modes of the EOF analysis are used to define the MJO through two daily time series. Plotted on a 2-dimensional plane, the distance of a point from the origin represents the strength of the MJO (often called the RMM amplitude), and the phase angle describes the phase of the MJO, or where it is in its life cycle. Following Wheeler and Hendon (2004), when the MJO is active (e.g. above a certain amplitude threshold) we divide the RMM phase space into octants. Phases 1 and 2, for example, correspond to active MJO convection in the Indian Ocean. Phases 3 and 4 are associated with activity around the Maritime Continent, etc.. If the MJO is not active, we label it as Phase 0.

We define and track the MJO using ERA-20C reanalysis data (Poli et al. 2016), a reanalysis dataset than spans the entire twentieth century and provides a larger sample size than the observational record. From ERA-20C, we use daily OLR, u850, and u200 data from May 1, 1900 until December 31, 2010 to calculate the RMM index. RMM is calculated from the ERA-20C data following the methodology in Wheeler and Hendon (2004) discussed above, except that the full ERA-20C period is used to define the climatology, and the processed data are projected onto the *observed* EOF modes from Wheeler and Hendon (2004) (as opposed to the EOFs from the ERA-20C data). Over the period when the observed RMM index overlaps with our ERA-20C RMM index, the two indices have a correlation of approximately 0.9, indicating very good agreement in how the RMM index is formed.

The network input is composed of three channels of 17 latitudes by 105 longitudes of u200, u850, and OLR, representing the three geophysical variables that go into the computation of the MJO index (see Fig. 6). Thus, a single sample has shape $17 \times 105 \times 3$. The labels are set to be the phase of the MJO, with phase 0 representing days where the amplitude of the MJO is less than 0.5. We choose to train on all available data; thus, the classes are not equally balanced across phases (see Supp. Fig. S1), although they are similar.

Given that there is memory of the MJO phase from one day to the next, we divide the 1900-2010 data into training and testing via distinct years. Specifically, the testing data is all calendar days within the 22 randomly selected years: 1902, 1903, 1907, 1912, 1916, 1917, 1918, 1923,

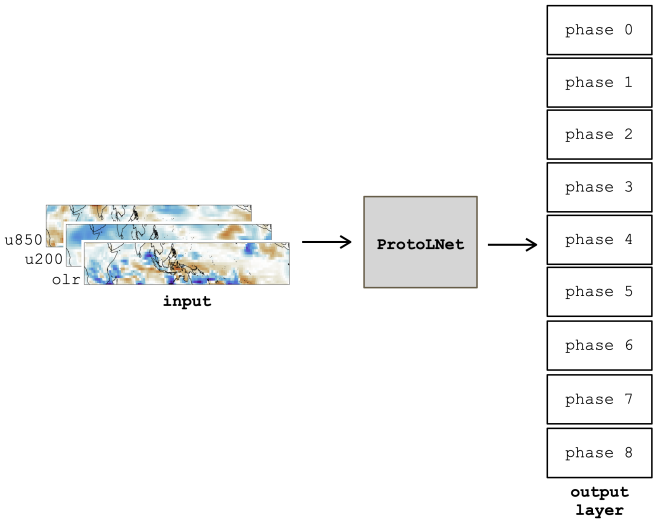


FIG. 6. Prediction setup for the MJO use case.

1935, 1937, 1941, 1945, 1946, 1949, 1953, 1961, 1965, 1976, 1992, 2007, 2008, and 2010. The training years comprise the remaining 89 years. (Results for other combinations of training/testing accuracies are given in Supp. Table S1.) This results in 32,387 training samples and 8,035 testing samples. The three input fields (channels) are converted to anomalies prior to analysis following a similar pre-processing as for the RMM computation. That is, the time-mean calendar-day seasonal cycle is subtracted from each gridpoint, and the mean of the previous 120 days is removed. Each variable is individually normalized by dividing it by its tropics-wide standard deviation. Then, immediately prior to training, the inputs are further standardized by the mean and standard deviation across all gridpoints and channels of the training set (via flattening the input fields).

We first train a standard CNN to perform the classification task and act as our base CNN for the ProtoLNet. Details of the CNN architecture and training parameters are provided in Appendix A. Once the CNN is trained, we remove the final fully connected layer and output layer, and append the result to the ProtoLNet to become the base CNN (see Fig. 1). We assign 10 prototypes (with $D = 64$) to each output class, which results in a total of 90 prototypes. Fewer than 90 reduced the accuracy, while using more than 90 did not improve the predictions. We cycle through the three stages of ProtoLNet training (Fig. 2) five times, freezing the base CNN for the first cycle of stage 1, but allowing it to train on all subsequent cycles of stage 1. Once fully trained, the ProtoLNet achieves a testing accuracy of 73% for classifying the phase of the MJO into one of nine classes

(random chance is approximately 11%), which is similar to the accuracy found in Toms et al. (2021) using a black box neural network. Supp. Fig. S2 shows that the ProtoLNet exhibits testing accuracies between approximately 70-80% across phases. A ProtoPNet, which does not consider location, never achieves an accuracy above 30%.

Interestingly, the base CNN upon which our ProtoLNet was trained converged to an accuracy of 58%, much lower than that of the subsequent ProtoLNet. We believe that the improved accuracy of the ProtoLNet is due to the regularizing nature of the prototype architecture. That is, the prototype approach constrains the network to focus on only a few latent features for phase identification, allowing it to converge on an appropriate decision-making strategy when the training data is limited (see discussion of additional experiments in Section 5). We believe that this may be an additional benefit of the prototype approach that is worthy of further investigation. With that said, Supp. Table S1 shows accuracies for the base CNN and ProtoLNet for six additional random seeds that set the model initialization and training/testing split. The ProtoLNet accuracies are incredibly robust across all seeds. While in two of the cases the base CNN achieved lower accuracies than the ProtoLNet (as in the setup shown here), the base CNN more often achieved a slightly higher accuracy than the ProtoLNet. Thus, it appears that the original accuracy of the base CNN does not solely dictate the resulting accuracy of the ProtoLNet.

An example of the interpretability of the ProtoLNet's prediction for testing sample 7591 is shown in Fig. 7. This sample corresponds to phase 2 of the MJO on October 14, 2008, and the three input fields (u200, u850, and olr) are displayed across the top row for that day. All anomalies are shown, but the shading outside of the prototype receptive field is muted in color. Note that the large-scale, enhanced convection of the western Indian Ocean (Fig. 7c) is a classic indication of a phase 2 MJO event, corresponding with a coupled wind response that shows upper-level easterlies (Fig. 7a), and lower-level westerlies (Fig. 7b) in the same region.

The network correctly classifies this sample as phase 2, and we can use the interpretability of the ProtoLNet to further explore why. Although multiple prototypes contributed to the winning number of points for the classification of sample 7591, it can be insightful to investigate the winning prototype (i.e. the prototype that contributes the most points). With multiple channels as input, the winning prototype for this sample (Prototype #20) is visualized as three different fields, one for each input variable (i.e. u200, u850, olr), as shown in Fig. 7d-f. Prototype 20 is a latent

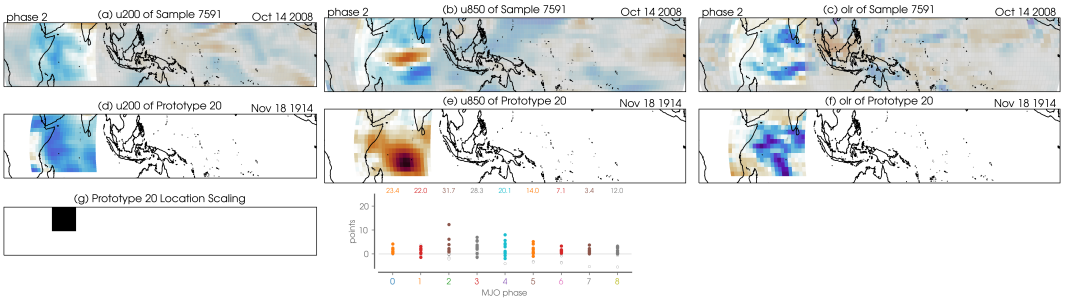
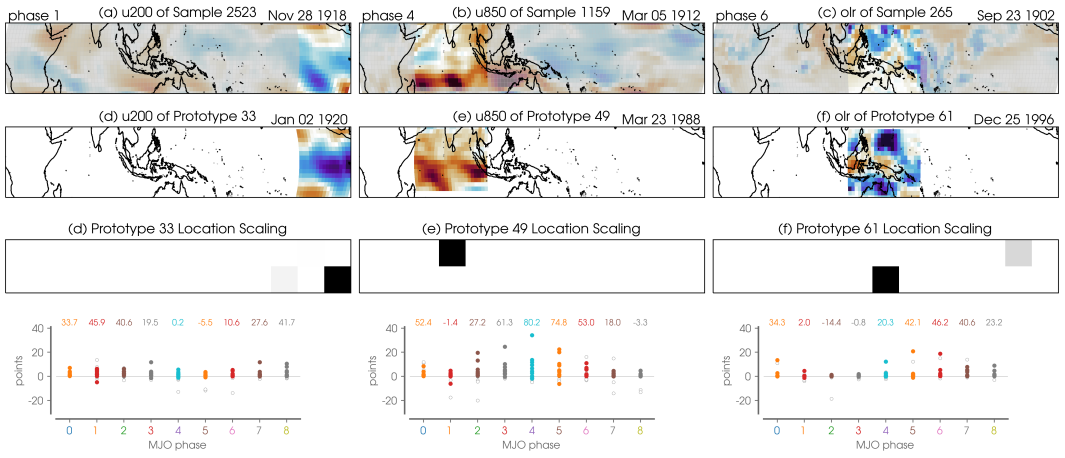


FIG. 7. One example prediction (testing sample 7591) by the ProtoLNet for the MJO use case, along with the winning prototype (prototype 20) and associated location scaling grid. The three columns denote the three input fields (i.e. u200, u850, olr). All anomalies are shown in panels (a)-(c) but the shading outside of the prototype patch is muted in color. The color scales are dimensionless with red shading denoting positive values and blue shading denoting negative values. The bottom middle panel show the points given to each class by each prototype, with the sum (i.e. total points) displayed along the top. Colored dots denote prototypes associated with the same class, and white dots denote contributions from prototypes of other classes.

patch corresponding to the state of the western Indian Ocean on November 18, 1914. The location scaling grid associated with Prototype 20 (Fig. 7g) highlights that similarities to this prototype are only heavily weighted when found at these longitudes. Thus, we see that the anomaly fields on October 14, 2008, for sample 7591 look a lot like those of Prototype 20, with upper-level easterlies, lower-level westerlies and enhanced convection over the western Indian Ocean. This provides evidence for why the network classified this sample as MJO phase 2.

Fig. 8 shows three additional (correctly predicted) testing samples and their winning prototypes, displaying only one geophysical field for each prediction to simplify the figure. Sample 2523 on November 28, 1918, is classified as phase 1, in part because its upper-level easterlies *look like* those of Prototype 33 from January 2, 1920 over the eastern Pacific (Fig. 8a,d). The lower-level westerlies over the Indian Ocean on March 5, 1912, *look like* those of phase 4 Prototype 49 from March 23, 1988 (Fig. 8b,e). Enhanced convection as seen by the OLR field east of the Maritime Continent on September 23, 1902 *looks like* that of phase 6 Prototype 61 (Fig. 8c,f).

As a summary of the MJO classification results, Fig. 9 displays the most frequently winning prototype for each phase of the MJO. A hallmark feature of the MJO is its eastward propagation, and Fig. 9 reveals the eastward progression of the prototypes (and associated location scaling



473 FIG. 8. As in Fig. 7, but for three additional example testing samples (one per column) displaying only one
474 geophysical field for each.

482 FIG. 9. The most frequently winning prototype for correctly classified testing samples by MJO phase. Each
483 column represents a different input variable; the fourth column displays the associated location scaling.

487 grids) starting in phase 2 and continuing to phases 7 and 8. That is, the ProtoLNet, with its
488 location-specific focus, has learned representative prototypes that move eastward with the known
489 progression of the MJO. Phase 1, however, does not appear to behave this way. Prototype 16 is
490 often the most-winning prototype for phase 1, but it is focused over the mid-Pacific rather than

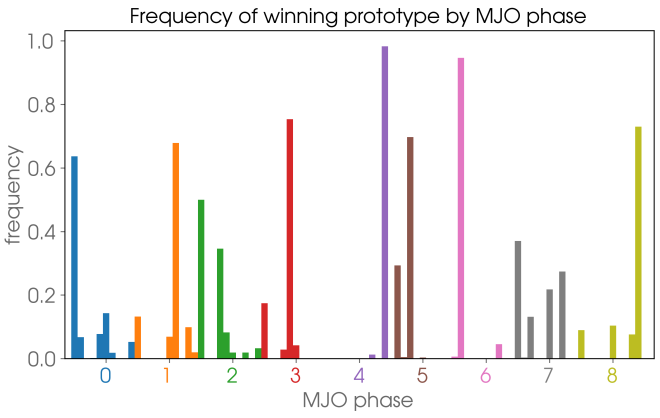


FIG. 10. The frequency that each prototype is the winning prototype (i.e. contributes the most points to the predicted class) for each correctly classified testing sample. Each phase has 10 possible prototypes; however, there are some prototypes that are never a winning prototype. They have frequency of zero.

the western Indian Ocean as one might expect (this is true for most of the Phase 1 prototypes; see Supp. Fig. S5). The reason why phase 1 prototypes tend to focus on this region is not clear, but we hypothesize the network may be focusing on wind signals in this region associated with a phase 1 event forming or a previous MJO event decaying. Further investigation is needed.

Fig. 10 shows a breakdown of how often (i.e. for how many testing samples) each prototype was the winning prototype. For example, Prototype 49 from March 23, 1988, is the most-winning prototype for phase 4, and it is the winning prototype for 98% of all correctly classified phase 4 testing samples. This suggests that this prototype is highly indicative of phase 4 MJO events. On the other hand, phase 7 has multiple prototypes that frequently earn the title of winning prototype. Thus, Prototype 70 (displayed in Fig. 9) should be interpreted as only one possible indicator of phase 7.

All 10 learned prototypes for each phase are provided in the Supp. Fig. S4-S12. Careful inspection shows that some of the learned prototypes come from the same training sample, indicating a particularly prototypical event. However, in cases where the prototypes come from the same training sample and have similar location scaling grids, there could be concern that this is a repeated prototype. Chen et al. (2019) discuss an additional "pruning" step in their ProtoPNet methodology, although it could also be that the CNN is identifying different aspects of the image that are prototypical. Either way, for this MJO use case the Supp. Fig. S4-S12 specify how often

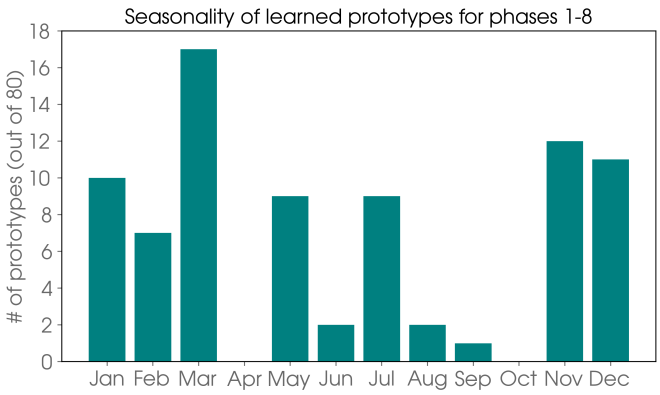


FIG. 11. Number of learned prototypes for MJO phases 1-8 (excluding phase 0, so out of 80 prototypes total) binned by month of the year of the training sample from which the prototype was drawn.

a particular prototype was the "winning" prototype and in all cases it is one specific prototype that wins-out over the rest which is why we are confident showing Fig. 10.

Fig. 11 shows the breakdown of the monthly distribution for all prototypes for active MJO phases 1-8. The network preferentially chooses prototypes from November-March when the MJO is known to be most active, however, prototypes from May and July are also learned, likely to capture the differences in MJO behavior across seasons (Zhang 2005). The monthly seasonality for all prototypes, including those for MJO phase 0, are shown in Supp. Fig. S3.

5. Discussion

The value of the ProtoLNet design is that interpretation of the network's decision-making process is baked into the architecture itself, rather than performed *post-hoc* like most explainable AI methods (Buhrmester et al. 2019; Barredo Arrieta et al. 2020; Samek et al. 2021). Although the network is constrained to only learn via similarities to a small number of learned prototypes, multiple use cases demonstrate that it can be trained to exhibit only a small reduction in accuracy compared to non-interpretable architectures (Chen et al. 2019; Singh and Yow 2021). Moreover, for our MJO use case, the ProtoLNet actually improved in accuracy over its base CNN. We hypothesize that this is because the ProtoLNet greatly reduces the search space possibilities, which allows the network to converge on a good prediction strategy given a limited sample size. One might think of this as a form of regularization, or instead, a form of physics-guided constraint (e.g. Beucler et al. 2021)

that forces the network to learn physically realizable evidence for each class. To further explore this hypothesis, we trained additional ProtoLNets for the idealized quadrants use case (Section 3), but with a much smaller training size (only 1,400 samples for training). In all cases, the ProtoLNets obtained higher testing accuracies — sometimes significantly higher — than their respective base CNNs (see results in Supp. Fig. S13). This is not to say that the ProtoLNet is categorically more accurate than a standard CNN. A more thorough exploration of the hyperparameter space could bring the base CNN accuracy up to that of the ProtoLNet. Instead, we just wish to highlight that with minimal tuning, the ProtoLNet was able to consistently achieve high accuracies with limited training data.

In addition to being interpretable, the ProtoLNet provides the benefit of learning a small subset of prototypical parts *from the training set* that reflect identifiable features for each output class. That is, each prototype is found "in the wild" and, thus, has a direct connection to a sample that has occurred. This should be distinguished from more standard architectures that learn complex latent representations and features that may never occur in reality. For the case of MJO phase classification, this means that the network can learn particular example MJO events that generalize across the observational record and reflect identifiable features for each specific MJO phase. Thus, although predicting the current phase of the MJO is routine from a scientific perspective, the ProtoLNet allows us to look back and identify specific dates that exhibit prototypical MJO phase behaviour, as shown in Fig. 9 and Fig. 11. Furthermore, it is straightforward to extend the interpretable ProtoLNet setup of Fig. 6 to ingest current atmospheric fields and predict the MJO phase at some lead time into the future.

As we have used it here, the ProtoLNet design learns localized prototypes from the input that provide evidence for a particular output class. This should be distinguished from the standard climate approach that composites the input fields over many samples for a single class, and thus results in a smooth averaged field (assuming there are enough samples to average out the noise). Such a composite field is computed pixel by pixel and as such, does not capture shared gradients or higher-level features that can be learned by the convolutional layers of the ProtoLNet. Finally, as discussed above, the ProtoLNet identifies prototypical behavior that has been realized in a training sample, while the composite field provides a smoothed, idealized picture that will likely never be observed.

The ProtoLNet is based on the ProtoPNet of Chen et al. (2019) which uses positive reasoning, i.e. *this looks like that*, to predict the correct class of an input image. Singh and Yow (2021) introduce a variation, the NP-ProtoPNet, which additionally includes negative reasoning, i.e. *this does not look like that*. Their argument is that by allowing negative reasoning, the network is able to better rule out incorrect classes and achieve accuracies on-par with the best performing black box models. It is straightforward to apply our location-scaling grid to a NP-ProtoPNet, which mainly involves relaxing the sparsity requirement of the final weights layer. However, by allowing both positive and negative reasoning, interpreting the model's decision making process may become significantly more difficult due to competing negative and positive point contributions to the final output classes. Thus, we chose to focus on positive reasoning for this study.

6. Conclusions

Driven by the desire to explain the decision-making process of deep learning models, a large variety of *post hoc* explainability methods have been developed (e.g. Buhrmester et al. 2019; Barredo Arrieta et al. 2020; Samek et al. 2021). However, these explainability methods come with their own challenges (Kindermans et al. 2019; Mamalakis et al. 2021) and recent work by Rudin (2019) and Chen et al. (2019) suggest that instead of trying to explain black box models, we should be creating models where the decision-making process is interpretable by design.

Here, we extend the interpretable ProtoPNet of Chen et al. (2019) to consider absolute location in the interpretable prototype architecture, which we term the ProtoLNet. The results of our work can be summarized by three main conclusions. (1) Considering absolute location in the ProtoLNet architecture greatly improves accuracy for the geophysical use cases explored here. (2) The ProtoLNet is interpretable in that it directly provides which prototypes are similar to different patches of an input image (i.e. *this looks like that*), and where these prototypes matter (i.e. *there*). (3) The network is able to learn specific historical dates that serve as multivariate prototypes of the different Madden-Julian oscillation phases.

This work serves as one example of an interpretable deep learning model specifically designed for earth system science applications (see also Sonnewald and Lguensat 2021). There is much more research to be done on the topic. For example, the incorporation of negative reasoning and extension to regression tasks could be beneficial for its use in earth science. Furthermore,

the interpretation and utility of the learned prototypes themselves, apart from the prediction task, leaves much to be explored. Thus, this work should be seen as merely a step in the direction of interpretable deep learning for earth science exploration.

Acknowledgments. This work was funded, in part, by the NSF AI Institute for Research on Trustworthy AI in Weather, Climate, and Coastal Oceanography (AI2ES) under NSF grant ICER-2019758. ZKM recognizes support from the National Science Foundation under Award No. 2020305. JKR recognizes support from the U.S. Department of Energy, Office of Science, Office of Advanced Scientific Computing Research, Department of Energy Computational Science Graduate Fellowship under Award No. DE-SC0020347.

Data availability statement. Once published, the code will be made available to the community via a permanent DOI on Zenodo. For peer-review, the code is available at `https://github.com/eabarnes1010/tlltt`. The ERA-20C is publically available at `https://www.ecmwf.int/en/forecasts/datasets/reanalysis-datasets/era-20c`.

APPENDIX A

Base CNN architectures and training

The base CNN for the idealized quadrants use case (Section 3) has two convolutional layers of 32 kernels each. Every convolutional layer is followed by an average pooling layer with kernel size 2×2 and a stride length of 2. The output of the final average pooling layer is flattened, and then fed into a final dense layer of 64 units which is fed into the final output layer of 3 units. The final output layer contains the softmax activation function which convert the outputs into confidences that sum to 1.0. The final dense layer is trained with dropout (Srivastava et al. 2014) at a rate of 0.4 to reduce overfitting. When the base CNN is appended to the ProtoLNet, the dropout rate is set to zero. That is, dropout is only used to reduce overfitting during the pre-training of the base CNN. The base CNN is trained with a fixed learning rate of 5e-5 for 12 epochs.

The base CNN for the MJO use case (Section 4) has three convolutional layers of 16 kernels each. Every convolutional layer is followed by an average pooling layer with kernel size 2×2 and a stride length of 2. The convolutional layers are trained with dropout at a rate of 0.4 to reduce overfitting. The output of the final average pooling layer is flattened, and then fed into a final dense layer of 32 units that is fed into the final output layer of 9 units. The final output layer contains the softmax activation function which converts the outputs into confidences that sum to 1.0. The final dense layer is trained with dropout at a rate of 0.2. When the base CNN is appended to the ProtoLNet, the dropout rates are set to zero. That is, dropout is only used to reduce overfitting

during the pre-training of the base CNN. The base CNN is trained with a fixed learning rate of 0.00017548 for 23 epochs.

The hyperparameters for these networks were explored using KerasTuner. We did not find the results to be overly sensitive to these choices.

APPENDIX B

Learning location scaling exponents

The location scaling values must be non-negative. Subsequently, we use a trick from Duerr et al. (2020) and learn the exponents of the location scaling, rather than the values themselves. That is, if s_k denotes the location scaling value for prototype $\mathbf{p}$ at latent patch k then

$$s_k = e^{\gamma_k}, \tag{B1}$$

where the free parameter γ_k is *learned* by the network during training. Thus, at initialization, all γ_k values are initialized to zero so that the location scaling grid (all s_k values) is initialized to a grid of ones.

APPENDIX C

Stage 1 loss function

The Stage 1 loss function is given by Equation 2. There are three components: the usual CrossEntropy, plus a ClusterCost, and minus a SeparationCost.

Consider a set of input samples and associated class labels $\{(\mathbf{x}_i, y_i) : i = 1, 2, \ldots, N\}$. The output from the extended CNN given sample $\mathbf{x}_i$ is a quilt of latent patches $\mathbf{z}_{ik}$, where k indexes the latent patches. For the architecture shown in Figure 1, $k \in \{1, 2, \ldots, 6\}$ because the quilt is 2×3. Let s_k denote the current location scaling value associated with latent patch k, and $\mathbf{P}_{y_i}$ denote the set of all prototypes belonging to class y_i. The ClusterCost is given by

$$\text{ClusterCost} = \frac{1}{N} \sum_{i=1}^{N} \left[\min_{\mathbf{p} \in \mathbf{P}_{y_i}} \min_{k} \frac{\|\mathbf{z}_{ik} - \mathbf{p}\|_2^2}{s_k + \epsilon} \right] \tag{C1}$$

where $\| \|_2^2$ is the squared L_2 norm and ϵ is a small number, there to guard against divide-by-zero problems.

The ClusterCost encourages training images to have at least one latent patch with high similarity to a prototype of the same class. The computation is based on Chen et al. (2019), but incorporates the location scaling grid introduced in this paper.

The SeparationCost discourages training images from having high similarity to prototypes belonging to the incorrect class. The computation is almost identical to that of the ClusterCost. The difference is that we minimize over the set of all prototypes that do not belong to class y_i.

$$\text{SeparationCost} = \frac{1}{N}\sum_{i=1}^{N}\left[\min_{\mathbf{p}\notin\mathbf{P}_{y_i}}\min_{k}\frac{\|\mathbf{z}_{ik}-\mathbf{p}\|_2^2}{s_k+\epsilon}\right] \tag{C2}$$

For the idealized quadrants use case, we set the ClusterCost coefficient $\beta_1 \approx 0.17$ (see code for all digits) and the SeparationCost coefficient $\beta_2 = \beta_1/10$. For the MJO use case $\beta_1 = 0.2$ and $\beta_2 = \beta_1/10$. Note the negative sign in front of the SeparationCost term in Equation 2 encourages the network to have larger separation (lower similarity) between samples and the prototypes from incorrect classes.

References

Balmaseda, M., and Coauthors, 2020: NOAA-DOE precipitation processes and predictability workshop. Tech. Rep. DOE/SC-0203; NOAA Technical Report OAR CPO-9., U.S. Department of Energy and U.S. Department of Commerce NOAA.

Barnes, E. A., B. Toms, J. W. Hurrell, I. Ebert-Uphoff, C. Anderson, and D. Anderson, 2020: Indicator Patterns of Forced Change Learned by an Artificial Neural Network. *Journal of Advances in Modeling Earth Systems*, **n/a (n/a)**, e2020MS002 195, https://doi.org/10.1029/2020MS002195.

Barredo Arrieta, A., and Coauthors, 2020: Explainable artificial intelligence (XAI): Concepts, taxonomies, opportunities and challenges toward responsible AI. *Inf. Fusion*, **58**, 82–115, https://doi.org/10.1016/j.inffus.2019.12.012.

Beucler, T., M. Pritchard, S. Rasp, J. Ott, P. Baldi, and P. Gentine, 2021: Enforcing analytic constraints in neural networks emulating physical systems. *Phys. Rev. Lett.*, **126 (9)**, 098 302, https://doi.org/10.1103/PhysRevLett.126.098302.

Buhrmester, V., D. Münch, and M. Arens, 2019: Analysis of explainers of black box deep neural networks for computer vision: A survey. *arXiv*, 1911.12116.

Chen, C., O. Li, D. Tao, A. Barnett, C. Rudin, and J. K. Su, 2019: This looks like that: Deep learning for interpretable image recognition. *Advances in Neural Information Processing Systems*, Curran Associates, Inc., Vol. 32.

Davenport, F. V., and N. S. Diffenbaugh, 2021: Using machine learning to analyze physical causes of climate change: A case study of U.S. midwest extreme precipitation. *Geophys. Res. Lett.*, https://doi.org/10.1029/2021gl093787.

Duerr, O., B. Sick, and E. Murina, 2020: *Probabilistic Deep Learning: With Python, Keras and Tensorflow Probability*. MANNING PUBN.

Géron, A., 2019: *Hands-on Machine Learning with Scikit-Learn, Keras, and TensorFlow*. O'Reilly UK Ltd.

He, K., X. Zhang, S. Ren, and J. Sun, 2015: Delving deep into rectifiers: Surpassing Human-Level performance on ImageNet classification. *2015 IEEE International Conference on Computer Vision (ICCV)*, 1026–1034, https://doi.org/10.1109/ICCV.2015.123.

Irrgang, C., N. Boers, M. Sonnewald, E. A. Barnes, C. Kadow, J. Staneva, and J. Saynisch-Wagner, 2021: Towards neural earth system modelling by integrating artificial intelligence in earth system science. *Nature Machine Intelligence*, **3 (8)**, 667–674, https://doi.org/10.1038/s42256-021-00374-3.

Keys, P. W., E. A. Barnes, and N. H. Carter, 2021: A machine-learning approach to human footprint index estimation with applications to sustainable development. *Environ. Res. Lett.*, **16 (4)**, 044 061, https://doi.org/10.1088/1748-9326/abe00a.

Kindermans, P.-J., S. Hooker, J. Adebayo, M. Alber, K. T. Schütt, S. Dähne, D. Erhan, and B. Kim, 2019: The (un)reliability of saliency methods. *Explainable AI: Interpreting, Explaining*

and Visualizing Deep Learning, W. Samek, G. Montavon, A. Vedaldi, L. K. Hansen, and K.-R. Muller, Eds., Springer International Publishing, Cham, 267–280, https://doi.org/10.1007/978-3-030-28954-6_14.

Lapuschkin, S., S. Wäldchen, A. Binder, G. Montavon, W. Samek, and K.-R. Müller, 2019: Unmasking clever hans predictors and assessing what machines really learn. *Nat. Commun.*, **10 (1)**, 1096, https://doi.org/10.1038/s41467-019-08987-4.

Madden, R. A., and P. R. Julian, 1971: Detection of a 40-50 day oscillation in the zonal wind in the tropical pacific. *Journal of Atmospheric Sciences*, **28 (5)**, 702 – 708, https://doi.org/10.1175/1520-0469(1971)028<0702:DOADOI>2.0.CO;2, URL https://journals.ametsoc.org/view/journals/atsc/28/5/1520-0469_1971_028_0702_doadoi_2_0_co_2.xml.

Madden, R. A., and P. R. Julian, 1972: Description of global-scale circulation cells in the tropics with a 40-50 day period. *Journal of Atmospheric Sciences*, **29 (6)**, 1109 – 1123, https://doi.org/10.1175/1520-0469(1972)029<1109:DOGSCC>2.0.CO;2, URL https://journals.ametsoc.org/view/journals/atsc/29/6/1520-0469_1972_029_1109_dogscc_2_0_co_2.xml.

Mamalakis, A., E. A. Barnes, and I. Ebert-Uphoff, 2022: Investigating the fidelity of explainable artificial intelligence methods for applications of convolutional neural networks in geoscience. 2202.03407.

Mamalakis, A., I. Ebert-Uphoff, and E. A. Barnes, 2021: Neural Network Attribution Methods for Problems in Geoscience: A Novel Synthetic Benchmark Dataset. *arXiv, 2103.10005*, 2103.10005.

Martin, Z. K., E. A. Barnes, and E. D. Maloney, 2021: Using simple, explainable neural networks to predict the madden-julian oscillation. *Earth and Space Science Open Archive*, 42, https://doi.org/10.1002/essoar.10507439.2, URL https://doi.org/10.1002/essoar.10507439.2.

Mayer, K. J., and E. A. Barnes, 2021: Subseasonal forecasts of opportunity identified by an explainable neural network. *Geophys. Res. Lett.*, https://doi.org/10.1029/2020gl092092.

McGovern, A., R. Lagerquist, D. John Gagne, G. E. Jergensen, K. L. Elmore, C. R. Homeyer, and T. Smith, 2019: Making the black box more transparent: Understanding the physical

implications of machine learning. *Bull. Am. Meteorol. Soc.*, **100 (11)**, 2175–2199, https://doi.org/10.1175/BAMS-D-18-0195.1.

Montavon, G., W. Samek, and K.-R. Müller, 2018: Methods for interpreting and understanding deep neural networks. *Digit. Signal Process.*, **73**, 1–15, https://doi.org/10.1016/j.dsp.2017.10.011.

National Academies of Sciences Engineering and Medicine, 2020: *Earth System Predictability Research and Development: Proceedings of a Workshop–in Brief*. The National Academies Press, Washington, DC, https://doi.org/10.17226/25861.

Philander, S. G. H., 1983: El niño southern oscillation phenomena. *Nature*, **302 (5906)**, 295–301, https://doi.org/10.1038/302295a0.

Poli, P., and Coauthors, 2016: Era-20c: An atmospheric reanalysis of the twentieth century. *Journal of Climate*, **29 (11)**, 4083 – 4097, https://doi.org/10.1175/JCLI-D-15-0556.1, URL https://journals.ametsoc.org/view/journals/clim/29/11/jcli-d-15-0556.1.xml.

Rasp, S., H. Schulz, S. Bony, and B. Stevens, 2019: Combining crowd-sourcing and deep learning to understand meso-scale organization of shallow convection. *arXiv*, 1906.01906.

Rudin, C., 2019: Stop explaining black box machine learning models for high stakes decisions and use interpretable models instead. *Nature Machine Intelligence*, **1 (5)**, 206–215, https://doi.org/10.1038/s42256-019-0048-x.

Samek, W., G. Montavon, S. Lapuschkin, C. J. Anders, and K.-R. Müller, 2021: Explaining deep neural networks and beyond: A review of methods and applications. *Proc. IEEE*, **109 (3)**, 247–278, https://doi.org/10.1109/JPROC.2021.3060483.

Simonyan, K., and A. Zisserman, 2014: Very deep convolutional networks for Large-Scale image recognition. *arXiv*, 1409.1556.

Singh, G., and K.-C. Yow, 2021: These do not look like those: An interpretable deep learning model for image recognition. *IEEE Access*, **9**, 41 482–41 493, https://doi.org/10.1109/ACCESS.2021.3064838.

Sonnewald, M., and R. Lguensat, 2021: Revealing the impact of global heating on north atlantic circulation using transparent machine learning. *J. Adv. Model. Earth Syst.*, **13 (8)**, https://doi.org/10.1029/2021ms002496.

Srivastava, N., G. Hinton, A. Krizhevsky, I. Sutskever, and R. Salakhutdinov, 2014: Dropout: a simple way to prevent neural networks from overfitting. *J. Mach. Learn. Res.*, **15 (1)**, 1929–1958.

Toms, B. A., E. A. Barnes, and I. Ebert-Uphoff, 2020: Physically Interpretable Neural Networks for the Geosciences: Applications to Earth System Variability. *Journal of Advances in Modeling Earth Systems*, https://doi.org/10.1029/2019MS002002.

Toms, B. A., K. Kashinath, D. Yang, and Prabhat, 2021: Testing the reliability of interpretable neural networks in geoscience using the Madden–Julian oscillation. *Geosci. Model Dev.*, **14 (7)**, 4495–4508, https://doi.org/10.5194/gmd-14-4495-2021.

Wheeler, M. C., and H. H. Hendon, 2004: An all-season real-time multivariate mjo index: Development of an index for monitoring and prediction. *Monthly Weather Review*, **132 (8)**, 1917 – 1932, https://doi.org/10.1175/1520-0493(2004)132<1917:AARMMI>2.0.CO;2, URL https://journals.ametsoc.org/view/journals/mwre/132/8/1520-0493_2004_132_1917_aarmmi_2.0.co_2.xml.

Zhang, C., 2005: Madden-julian oscillation. *Reviews of Geophysics*, **43 (2)**, https://doi.org/https://doi.org/10.1029/2004RG000158, URL https://agupubs.onlinelibrary.wiley.com/doi/abs/10.1029/2004RG000158, https://agupubs.onlinelibrary.wiley.com/doi/pdf/10.1029/2004RG000158.