A Machine Learning Tutorial for Operational Meteorology. Part II: Neural Networks and Deep Learning

RANDY J. CHASE,[a,b,c] DAVID R. HARRISON,[b,d,e] GARY M. LACKMANN,[f] AND AMY MCGOVERN[a,b,c]

[a] *School of Computer Science, University of Oklahoma, Norman, Oklahoma*
[b] *School of Meteorology, University of Oklahoma, Norman, Oklahoma*
[c] *NSF AI Institute for Research on Trustworthy AI in Weather, Climate, and Coastal Oceanography, University of Oklahoma, Norman, Oklahoma*
[d] *Cooperative Institute for Severe and High-Impact Weather Research and Operations, University of Oklahoma, Norman, Oklahoma*
[e] *NOAA/NWS/Storm Prediction Center, Norman, Oklahoma*
[f] *Department of Marine, Earth, and Atmospheric Sciences, North Carolina State University, Raleigh, North Carolina*

(Manuscript received 4 November 2022, in final form 22 April 2023, accepted 1 May 2023)

ABSTRACT: Over the past decade the use of machine learning in meteorology has grown rapidly. Specifically neural networks and deep learning have been used at an unprecedented rate. To fill the dearth of resources covering neural networks with a meteorological lens, this paper discusses machine learning methods in a plain language format that is targeted to the operational meteorological community. This is the second paper in a pair that aim to serve as a machine learning resource for meteorologists. While the first paper focused on traditional machine learning methods (e.g., random forest), here a broad spectrum of neural networks and deep learning methods is discussed. Specifically, this paper covers perceptrons, artificial neural networks, convolutional neural networks, and U-networks. Like the Part I paper, this manuscript discusses the terms associated with neural networks and their training. Then the manuscript provides some intuition behind every method and concludes by showing each method used in a meteorological example of diagnosing thunderstorms from satellite images (e.g., lightning flashes). This paper is accompanied with an open-source code repository to allow readers to explore neural networks using either the dataset provided (which is used in the paper) or as a template for alternate datasets.

KEYWORDS: Radars/Radar observations; Satellite observations; Forecasting techniques; Nowcasting; Operational forecasting; Machine learning

1. Introduction

In the previous part of this tutorial series Chase et al. (2022, hereafter Part I) provided a survey of many of the most common traditional machine learning techniques that a meteorologist might encounter. This included: linear regression, logistic regression, naïve Bayes, decision trees, random forest, gradient boosted trees, and support vector machines. Beyond discussing the formulation of the methods, Part I also discussed the general terms associated with machine learning and provided an end-to-end machine learning example to detect lightning flashes within satellite and radar images. In this manuscript we continue our explanation and tutorial of supervised machine learning techniques by discussing a rapidly expanding category of machine learning known as *neural networks* and *deep learning*.

While neural networks can be viewed similarly to the other methods described in Part I (i.e., an empirical tool for making predictions and classifications), there are numerous nuances and different terms associated with neural networks that motivate their own detailed discussion. Furthermore, given the accelerated growth of neural networks (cf. Fig. 1e in Part I) and recent impressive demonstrations of neural networks achieving similar forecasting performance to numerical weather prediction (e.g., Weyn et al. 2020; Rasp and Thuerey 2021; Ravuri et al. 2021; Espeholt et al. 2022; Keisler 2022; Lam et al. 2022; Bi et al. 2022; Nguyen et al. 2023; Chen et al. 2023), the meteorological literature could benefit from a neural network–specific *plain language* discussion and simple meteorological example.

This paper follows the same organization as Part I. Section 2 provides an introduction to neural network–based machine learning methods and defines common neural network terms. Section 3 discusses how the neural network methods discussed in section 2 can be applied to a meteorological example. Section 4 summarizes this paper. The specific neural network types covered in this manuscript are perceptrons, artificial neural networks, convolutional neural networks, and "U"-shaped networks (U-Net).

2. Neural network methods and common terms

This section introduces many of the common terms that meteorologists would encounter while reading about or using output from neural networks. The goal of this paper is to provide readers with the intuition behind the different neural network methods as well as introduce common terms used within neural networks so that readers can become familiar with them. This section will dive deeper than Part I's corresponding section in

Denotes content that is immediately available upon publication as open access.

Corresponding author: Randy J. Chase, randy.chase@colostate.edu

DOI: 10.1175/WAF-D-22-0187.1

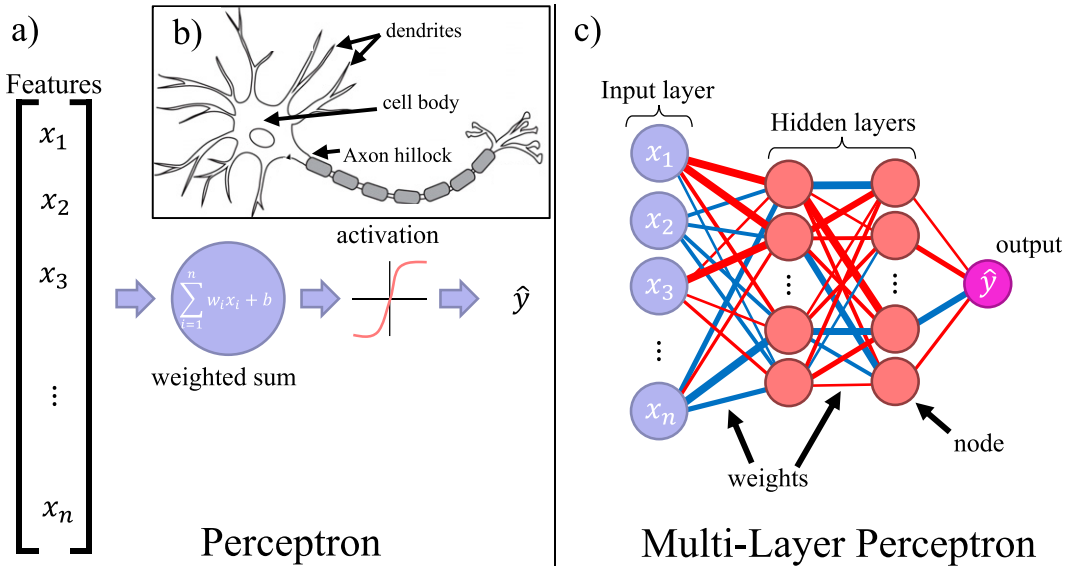


FIG. 1. Schematic of (a) a perceptron, (b) a biological neuron adapted from Henley (2021), and (c) a multilayer perceptron.

order to remove some of the mystery of the more complex mathematical nature of neural networks and hopefully achieve the same level of intuition as the traditional methods.

Before describing the various types of neural networks, also known as different *architectures*, we first define neural networks as: the group of machine learning methods that use a network of trainable weights that are organized in a structure that loosely resemble a biological brain. The name *neural network* comes from the analogy of how the information is passed in a biological brain and more specifically across neurons. Simply, a biological brain observes some information, which is then processed by a neuron and passed along a series of connections to numerous other neurons resulting in a thought or action.

Another common term that is used with neural networks is *deep learning*. While deep learning is often perceived as a synonym of neural networks by new users, it is actually a specific subset of neural networks. Since there are many different definitions of what exactly deep learning is, deep learning is defined here as a neural network that contains a minimum of two or more *hidden layers*,[1] though often involves many more than two layers (e.g., from tens to hundreds). This deep learning definition can be interpreted as a minimum complexity requirement for a neural network to be considered deep learning.

a. *Architectures of neural networks*

1) THE PERCEPTRON

The first architecture of neural networks came from McCulloch and Pitts (1943) in their formulation of a *logical neuron*, called later a *perceptron* (and referred to later in this document as nodes). A schematic of a perceptron is in Fig. 1a. The perceptron has some input data (i.e., features), which are altered by weights and aggregated (i.e., summed). Then the aggregated value is passed through an *activation function* which determines the output of the perceptron. This is similar to a biological neuron (Fig. 1b; Henley 2021), where information is passed to the neuron from the dendrites, aggregated at the cell body, passed through the axon hillock function, and then results in some output of the neuron.

Mathematically, the perceptron is

$$f(x) = \sigma\left(\sum_{i=1}^{i=n} w_i x_i + b\right), \tag{1}$$

where w_i are the weights, x_i are the n total input features, b is the bias, and σ is the activation function. Equation (1) will look familiar to those who read Part I because in essence it is the same as linear and logistic regression [Eq. (1) in Part I]. In fact, Eq. (1) is exactly logistic regression if the activation function is the sigmoid function. The only difference is how the weights w are determined, which is discussed later (section 2b). Since it is effectively the same as logistic regression, the perceptron is used in a similar manner. For example, we could use the same input features as Part I (e.g., minimum brightness temperature) to determine if there were any lightning flashes in a satellite image. Given the limited representational capacity of a perceptron, their application in the meteorological literature has been limited. One meteorological example can be found in Kim et al. (2013), where a perceptron is used to remove chaff[2] and clutter from radar data.

[1] Hidden layers are layers that do not directly interact with the input or output of a neural network. These are discussed more later.

[2] Military aircraft countermeasure for heat-seeking missiles.

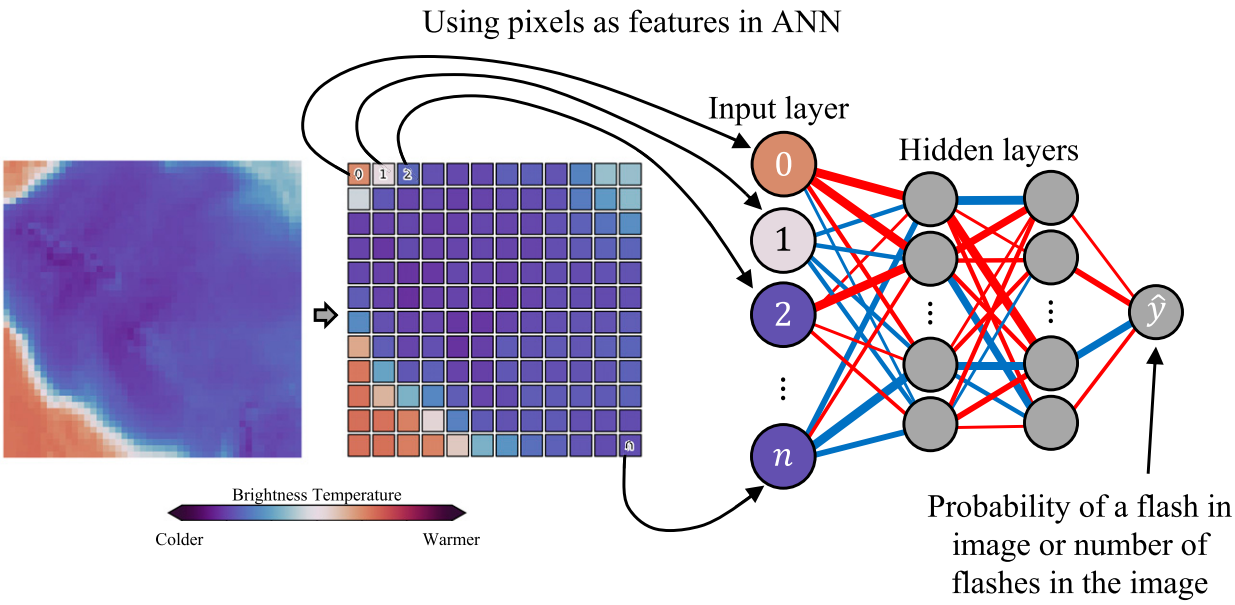


FIG. 2. Schematic of using each image pixel as input features to a multilayer perceptron [also known as an artificial neural network (ANN)]. The leftmost image is the infrared brightness temperature. The second image is the same brightness temperature image but coarsened for visualization purposes.

2) MULTILAYER PERCEPTRON (ARTIFICIAL NEURAL NETWORK)

Akin to how many neurons make up a brain, the second type of neural network is an extension of the single perceptron, which includes multiple perceptrons and multiple layers of multiple perceptrons (Rumelhart et al. 1986). This *multilayer perceptron* network is also known as an artificial neural network (ANN; Fig. 1c). Similar to the single perceptron, the data flows from the input layer (i.e., input data) to each of the perceptrons (hereafter *nodes*) through an activation function. The resulting information is then passed to all of the nodes in the next layer and so on until it reaches the output layer (i.e., where the final prediction is made). Any layer of nodes that are between the input and output are known as *hidden layers*. Mathematically, the multilayer network is usually summarized by the following:

$$\hat{y} = f(\mathbf{x}; \theta), \tag{2}$$

where $\hat{y}$ is the output of the neural network, f is the neural network that is a function of the input data $\mathbf{x}$, and has parameters (i.e., weights and biases) θ. Like the perceptron, the same features from the Part I data example can be plugged in as the input layer (Fig. 1c). Alternatively, ANNs can efficiently handle images where each pixel can be used as a feature (Fig. 2). Both methods are shown in the meteorology example in section 3.

The use of ANNs has been much more popular in meteorology than a single perceptron. Initial applications of ANNs in meteorology date back to the 1990s, which included short-term forecasts of rain (Kuligowski and Barros 1998); road temperatures (Shao 1998); significant thunderstorms (McCann 1992); damaging winds (Marzban and Stumpf 1998); and even tornadoes (Marzban and Stumpf 1996). More recent examples include the following: short-term forecasting of solar irradiance (McCandless et al. 2016); building radar retrievals of snowfall (Chase et al. 2021); and forecasting tropical cyclone intensity (Cloud et al. 2019; Xu et al. 2021).

Before continuing to the next type of neural networks, a popular neural network-based tool should be mentioned: *self-organizing maps* (SOM; Kohonen et al. 1997). Self-organizing maps are neural networks, but they are an *unsupervised* machine learning method. Recall the discussion of supervised and unsupervised machine learning in Part I, where unsupervised learning is machine learning on unlabeled data and thus SOMs focus on clustering data without human prescribed classes. For example, SOM have been used to classify severe storm environments (Anderson-Frey et al. 2017; Katona and Markowski 2021), organize synoptic weather patterns in context of warm precipitation events (Wang et al. 2019), and auto classify near-proximity soundings to supercells (Nowotarski and Jensen 2013). While unsupervised clustering applications are useful, they are not the focus of these two manuscripts (Part I and this part are focused on supervised learning) and likely deserve to have their own dedicated manuscript discussing all unsupervised techniques (e.g., principal component analysis, *K*-means clustering).

3) CONVOLUTIONAL NEURAL NETWORK

While applications of ANNs can be impressive, an additional advancement to neural networks was introduced by LeCun et al. (1989) named convolutional neural networks (CNNs). As the name implies, these neural networks use *convolutions* where a convolution is a function that processes an image by systematically altering the image with a small window called a *kernel* or *filter*. Graphically a convolution is shown in Fig. 3. An image is

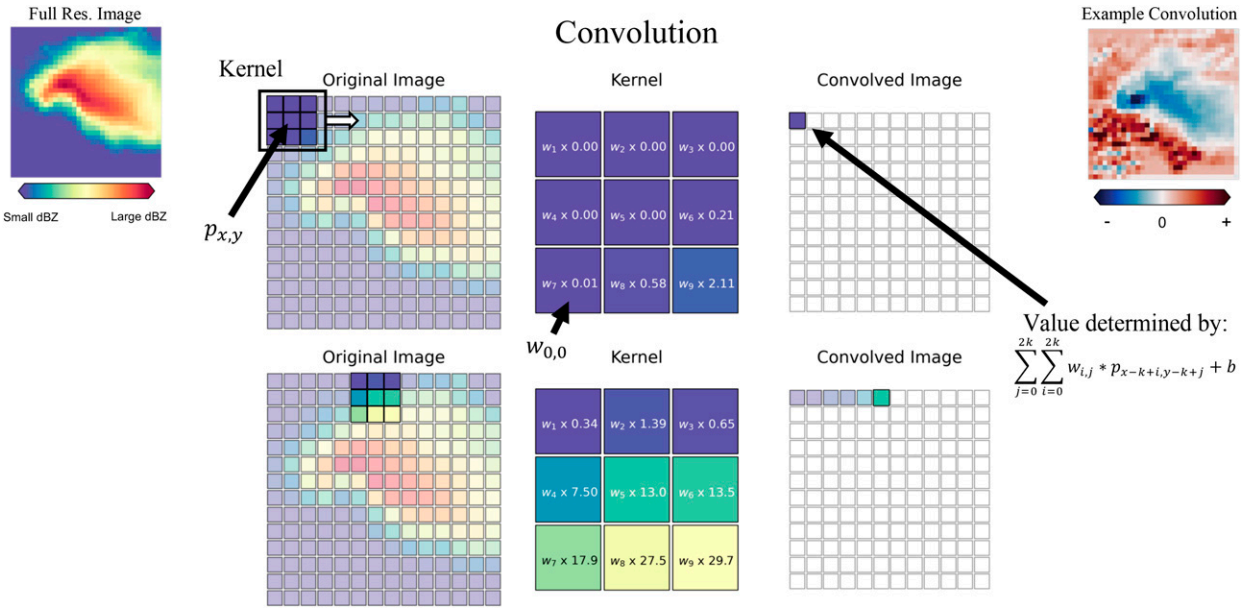


FIG. 3. Convolution graphic. The original hook echo radar reflectivity is located in the top-left corner. The convolution at step 0 is shown in the top row while the convolution at step 6 is shown in the bottom row. Note these images are coarsened for visualization purposes (i.e., can see the pixels). An animation of this convolution can be found in the Notebook 6 in the code repository. The result of the full convolution is shown in the top right, blues are negative, reds are positive, and they are unitless.

convolved/filtered by moving this kernel through the image. The kernel is made up of weights (center of Fig. 3), much like the nodes in an ANN, which are used to create a weighted sum that is a convolved image (also known as a feature map). Note that the weighted sum is passed through an activation function, as was done in the ANN. The mathematical expression of a convolution for some pixel p with coordinates x, y ($p_{x,y}$) is

$$p_{x,y} = \sigma\left(\sum_{j=0}^{j=2k}\sum_{i=0}^{i=2k} w_{i,j}p_{x-k+i,y-k+j} + b\right), \quad (3)$$

where $w_{i,j}$ is a scalar value (i.e., weight) that is learning during training at the at the ith and jth coordinate, $p_{x-k+i,y-k+j}$ is the pixel value, k is the floor (i.e., rounded down) of half the kernel size,[3] b is a scalar constant (also known as a bias term), and σ is an activation function (e.g., see sigmoid in Part I). This equation is then repeated for all pixels in the image. For visual learners, we encourage readers to check out the animated images in Lagerquist et al.'s (2020b) supplemental material[4] as well as Notebook 6 in the accompanying code with this manuscript.[5] You might notice that the convolution equation does not work for the edge of an image (i.e., negative indices do not make sense in this context). The fix for the edges of the image is to *pad* (i.e., add) a row of zeros on all edges of the image.

The idea of a convolution is probably very abstract, so let us consider an in depth example of how one could work. Figure 4a shows a classic radar "hook" echo (Fujita 1958). The data are from Lagerquist et al. (2020b) where the goal is to determine if the storm in the radar image will produce a tornado in the next hour. Before jumping into the CNN, first consider how a human would extract information from a radar image that might be useful for determining if a tornado will occur. One thought could be that we could have meteorologists go through thousands of images and encode the hook (i.e., 0 for no hook, 1 for hook), but that would be labor intensive and subjective. Another thought would be to take the max reflectivity of this image. That could work, since stronger storms have stronger updrafts and stronger reflectivity, which could be more likely to create a tornado, but maximum reflectivity is likely too simple. This thought activity should have illustrated that the optimal choices of data to extract are not trivial and since a machine learning model can only be as good as the predictors it is given, determining skillful inputs (i.e., features) is vital.

One of the main benefits of a CNN is that it will extract relevant features (i.e., patterns in input data) automatically from the data it is provided in order to optimize performance. Thus, there is no need for a human to manually identify important patterns in the images. Furthermore, since the CNN is using these convolutional windows, spatial information is automatically encoded into the features. The CNN does this feature extraction through the learning of the weights of the kernels. Sometimes these kernels are referred to as *filters* which is likely a more apt description of them. The kernels *filter* the features from the image. How the specific weights are learned is discussed in the following section (section 2b),

[3] For the example in Fig. 3, the kernel size is 3.
[4] https://journals.ametsoc.org/view/journals/mwre/148/7/mwrD190372.xml?tab_body=supplementary-materials.
[5] https://github.com/ai2es/WAF_ML_Tutorial_Part2/blob/main/jupyter_notebooks/Notebook6_Convolutions.ipynb.

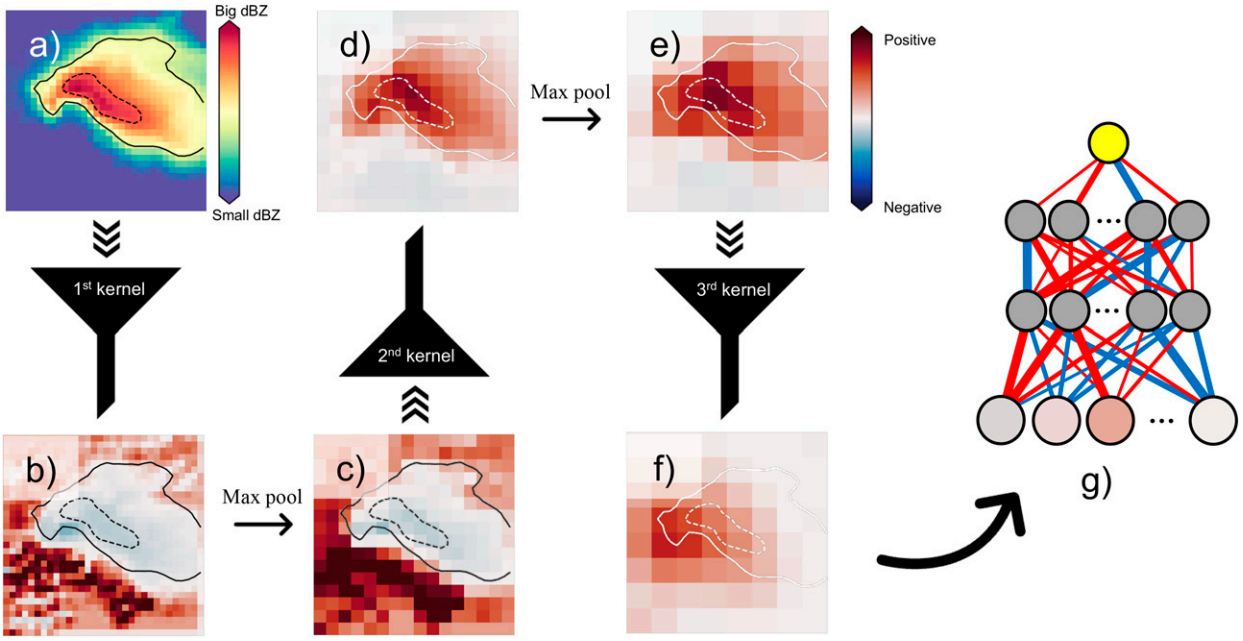


FIG. 4. A schematic showing how the learned kernels/filters from a CNN extracts features. (a) The same hook echo example from Fig. 3, but the 25- and 50-dBZ contours are drawn. (b) Colors are the normalized reflectivity values of the output of the first convolution, where reds are positive and blues are negative (unitless). The same storm contours are included from (a). (c) The result of pooling the image in (b). (d) As in (b), but taking the convolution of (c). (e) Result of the pooling in (d). (f) As in (d), but taking the convolution of (e). (g) An ANN that takes the pixels of (f) as input.

but know that the CNN tries multiple filters which result in some amount of error (e.g., truth–ML prediction). This error is then used to inform the CNN which filters work better than others and how to tweak the filters to get better performance (i.e., less error).

While the autoextraction of relevant features is a benefit of CNNs, it can also lead to unexpected results. A nonmeteorology example is from Lapuschkin et al. (2019) where the machine learning task is the classification of images with classes of dog, cat, horse, etc. Lapuschkin et al. (2019) showed that the CNN was using the copyright of images as a dominant predictor of the horse images. The Lapuschkin et al. (2019) example illustrates how vital the interrogation of the decisions of a CNN, and more broadly all machine learning methods, is. The interrogation of machine learning methods include explainable artificial intelligence (XAI) methods, which are discussed in section 3f.

Back to the hook echo example (Fig. 4a). One of the learned filters is shown in Fig. 4b, which appears to filtering out the storm location. But notice, that after a single convolution, we are still stuck with the same scenario from before: how do we extract information from the new image? (Fig. 4b). To answer this question, several more convolutions and many filters are typically used with an additional layer, called a *pooling layer*, in between convolution layers. A pooling layer is a way of reducing the dimensionality of the image, which ultimately allows the CNN to distill high resolution information into useful features. One can view pooling as making an image a lower resolution, like converting high resolution precipitation maps from 1-km horizontal grid spacing to a more regional scale such as 20 km. The intuition behind pooling layers can be thought of as summarizing the key findings of a scientific paper. The pooling layers boil down the most vital information in the paper (image), representing it in a smaller space (less pixels). Pooling is done similarly to the convolution kernel (i.e., uses a window), but has static weights which either take an average value (i.e., average pooling) or passes the maximum value through (i.e., maximum pooling). The typical size of a pooling kernel is two by two, which effectively halves the dimensions of the image. In the hook echo example, the result of the pooling is apparent as the grid becomes coarser and the CNN focuses (i.e., large values) in on the hook echo location (Fig. 4f).

In the process of summarizing data (i.e., pooling), there is less space for information to be stored (i.e., less pixels). Thus, in the CNN the number of filters (i.e., kernels/filters) typically increases with depth in the CNN (Fig. 5). Drawing on the same scientific paper analogy, imagine at the beginning of the network the CNN has only one filter and it writes a full-page summary on one key finding of the paper with plenty of detail. After another pooling layer, the full-page summary gets summarized further into one paragraph. Another pooling layer results in a sentence, and finally another layer leaves the filter with one word. The analogy here keeps the number of filters the same (i.e., one). If instead the CNN has access to more filters as it goes deeper (i.e., more pooling layers), the CNN can then summarize different aspects of the key findings, enhancing the total extracted information by the CNN.

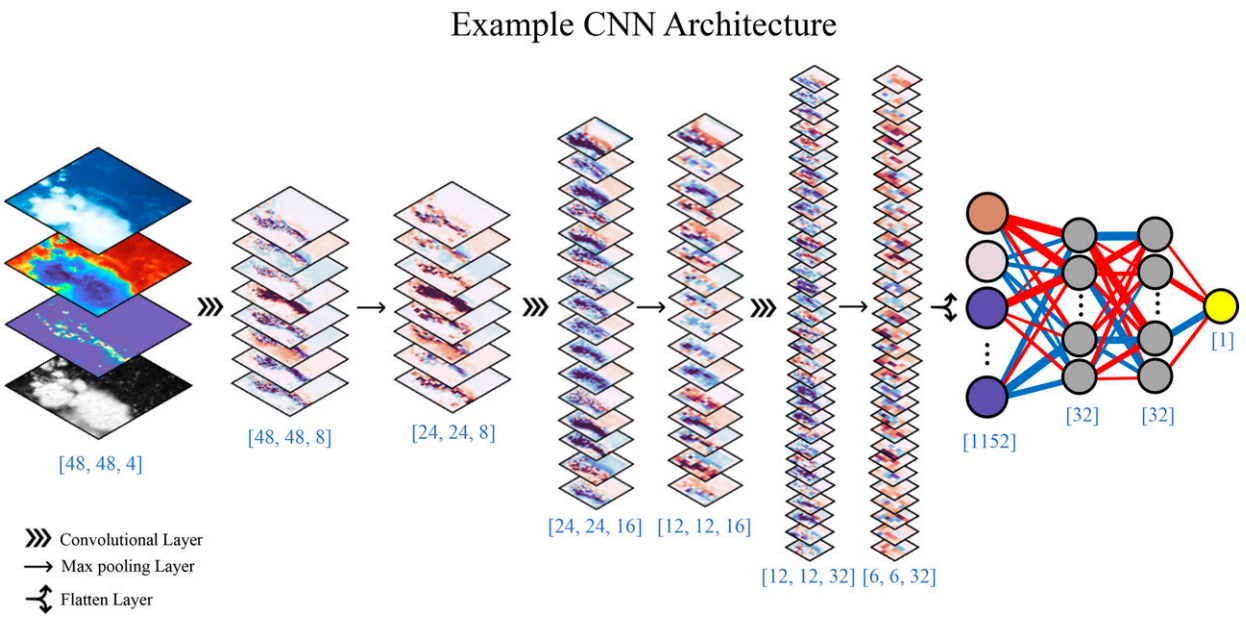


FIG. 5. Example CNN architecture. The different arrow indicators symbolize different layer types; see the legend in the bottom-left-hand corner. The blue bracketed text is the size of the images ([*x* dimension, *y* dimension, channel/feature dimension]) or vector (i.e., dense layers). This is the exact architecture for the best performing CNN in section 3.

From the discussion thus far, it is likely not clear how many layers of convolutions-pooling are needed for a machine learning task. The exact number of convolutions and pooling is problem dependent, which is usually determined through hyperparameter tuning (i.e., trying a bunch of different number of layers). Furthermore, for some problems a CNN without pooling layers might work better. This can also be tested with hyperparameter tuning (i.e., train a model with pooling and without pooling). One way to anticipate the number of convolutional and pooling layers is needed could be to consider receptive fields, where the maximum pattern size that a CNN can encode can be anticipated (see Ebert-Uphoff and Hilburn 2020, for discussion on receptive fields).

After the convolutional layers and their corresponding pooling layers an ANN is usually appended on to the end (Fig. 4g). In other words, after the final convolutional layer, the images are reshaped into a one-dimensional vector (like Fig. 2) and passed into the ANN. An example of a CNN architecture that is used in the data example (section 3d) is shown in Fig. 5.

Convolutional neural networks are an emerging technique in the meteorological literature that can do complex tasks. Examples include the following: detecting fronts in reanalysis data (Lagerquist et al. 2019, 2020a); estimating tropical cyclone intensity for satellite data (Chen et al. 2019; Griffin et al. 2022); determining if a storm will produce severe hail (Gagne et al. 2019); automatically classifying strongly rotating storms in numerical weather prediction data (Molina et al. 2021); and identifying intense convection in satellite imagery (Cintineo et al. 2020; Lee et al. 2021). While all of the discussion thus far has been focused on two dimensional convolutions and images, the idea can be extended to work on one dimensional data (e.g., a temperature profile) and to full three-dimensional volumes (e.g., numerical weather prediction output) or as time as a third dimension. The only change to go from a two dimensional convolution to a one- or three-dimensional convolution is the shape of the kernel. Both one-dimensional (e.g., Stock 2021; Harrison et al. 2022) and three-dimensional convolutions have been used in meteorological applications (e.g., Lagerquist et al. 2020b; Zhou et al. 2020; Kamangir et al. 2021; Justin et al. 2022).

4) U NETWORK

Even though the ANNs and CNNs described above can do skillful meteorological tasks, their architecture is best suited to do a single output (i.e., one-dimensional output) like diagnosing how many lightning flashes are in a satellite image or labeling a radar image as a squall line or supercell. An advancement to neural network architectures came from Ronneberger et al. (2015), where an architecture named *U Network (U-Net)* was introduced. Originally designed to label and track biological cells in microscope imagery, this method lends itself to doing a valuable task called *image-to-image translation*. Image-to-image translation is an example of taking some input image like infrared brightness temperature and translating it into a map of lightning data. The primary advantage of U-Nets is it will produce an image with a similar shape to its inputs.

An example U-Net is shown in Fig. 6. The name U-Net stems from the general U shape the network is built in. To be clear, a U-Net is a specific type of CNN, so it contains the same makeup of convolutional layers and pooling layers of a CNN, but the U-Net differs in that it contains a series of upsampling or unpooling layers [i.e., opposite of the pooling,

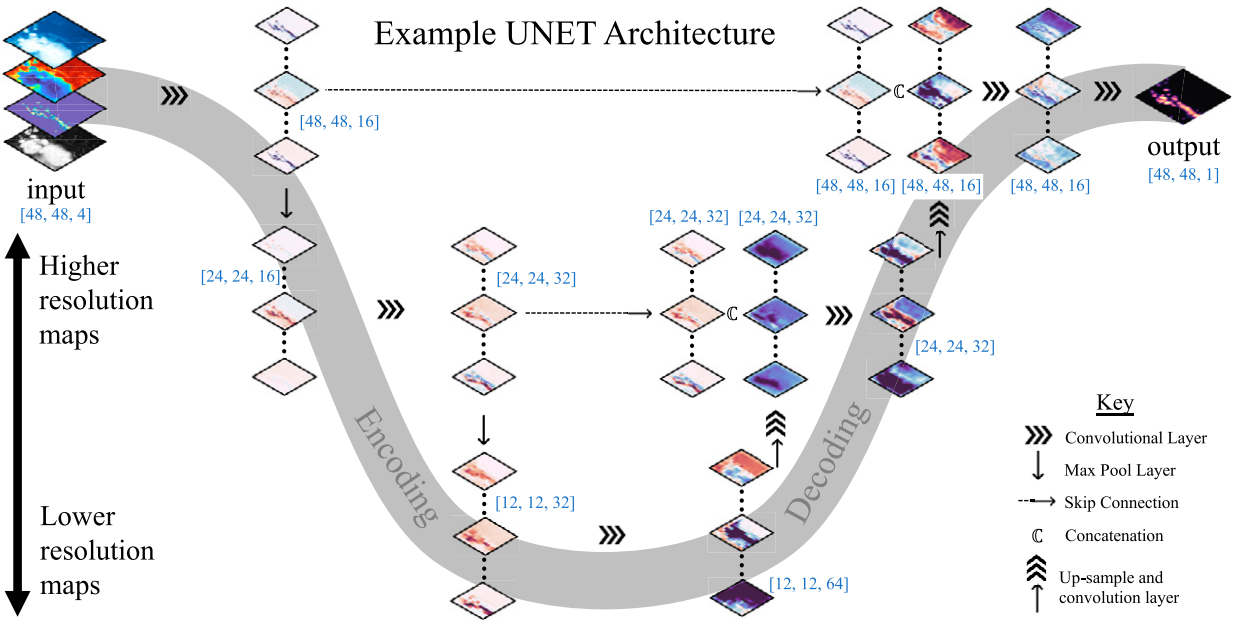


FIG. 6. Example U-Net architecture. As in Fig. 5, the different arrow indicators symbolize different layer types, and the legend is in the bottom-right-hand corner. The red–blue colored images in the middle are the convolved images. Only three kernels are shown for space reasons. The blue bracketed text is the size of the images ([*x* dimension, *y* dimension, channel/feature dimension]). This is the exact architecture for the best performing U-Net in section 3.

increasing the resolution through some interpolation technique (e.g., nearest neighbor)] instead of the ANN added to the end of the CNN that was shown in Fig. 5.

Each step down the left-hand side in the U (Fig. 6) symbolizes the pooling reduction of image resolution. Then at the bottom of the U, instead of doing additional pooling or flattening of data to be fed into an ANN (like a CNN), the data are upsampled (i.e., resampled to include more pixels using an interpolation method like nearest neighbor) and convolved. Then the new higher resolution images are concatenated (i.e., combined) with the same shaped images from the left-hand side of the architecture (see matrix sizes in Fig. 6) and passed through a convolution, but this time the number of filters is halved, as opposed to the number of filters doubling on the left side of the U. The concatenations from one side of the architecture to the other are called *skip-connections*. The process of upsampling and concatenating is repeated until you reach the original input image shape. The left side of the U is often called the *encoding branch* while the right side is often called the *decoding branch*.

The intuition behind U-Nets is similar to CNNs where convolutions are used to extract spatial information. The added complexity of a U-Net beyond a CNN allows a machine learning method to produce a whole map (i.e., matrix) of predictions, instead of a single pixel or value (i.e., scalar). This is extremely useful for meteorological datasets since often in meteorology, users are interested in spatial distributions of variables (e.g., dryline location). Given that forecasters have deemed timeliness an important property of machine learning meteorological tools (Harrison et al. 2022), the production of a map from U-Nets is helpful because a CNN trained to do the same task as the U-Net will require N more iterations (e.g., more time) to produce the same map, where N is the number of pixels in the map.

Examples of U-Nets in meteorology include automatic detection of cyclones in satellite imagery (Kumler-Bonfanti et al. 2020), translating geostationary satellite data into radar data (Hilburn et al. 2021), short-term forecasts of lightning (Zhou et al. 2020; Cintineo et al. 2022) and convection (Lagerquist et al. 2021), labeling bow echoes within model data (Mounier et al. 2022), and downscaling (i.e., statistically increasing the resolution) of coarse numerical weather prediction data (Sha et al. 2020a,b).

5) SUMMARY OF ALL MACHINE LEARNING METHODS

By this point in the paper series (Part I and Part II combined) there have been discussions about a total of 11 machine learning methods. Thus, in order to organize and summarize these various techniques and their distinctions, Fig. 7 is provided. In the graphic there is a brief summary of each method, some strengths, and some weaknesses.

b. How to train neural networks

1) LEARNING THE WEIGHTS

To determine the weights of a neural network (i.e., θ) the method is similar to the traditional machine learning methods mentioned in Part I. More specifically, the training data are used to learn the weights of the machine learning model such that the loss (i.e., error or cost) is minimized through the use of derivatives (i.e., gradients). While this simplified intuition works well for the traditional machine learning models

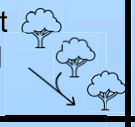
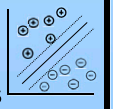
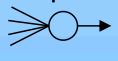
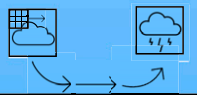
Method	Summary	Strengths	Weaknesses
Linear Regression 	A simple and fast-training linear fit to a dataset which is used for *regression* tasks	Fast, simple and easy to understand	Linear, sensitive to outliers
Logistic Regression 	A simple and fast-training linear fit to a dataset which is used for *classification* tasks	Fast and simple	Simple
Decision Trees 	A simple and fast-training algorithm that emulates a flow chart	Fast, simple and easy to understand	Hard cutoffs, limited skill, easily overfit
Naïve Bayes 	A simple and fast-training classification algorithm that follows (Bayesian) statistics	Fast, simple and easy to understand	Must choose the distribution and assume feature independence
Random Forest 	A collection (i.e., forest; ensemble) of decision trees trained on random subsets of features and data	Ensemble method that can account for non-linear relationships	Harder to *explain* compared to Decision trees
Gradient Boosted Trees 	A collection of decision trees that are successively trained to reduce the *loss* of the data	Can account for non-linear relationships	Harder to *explain* compared to Decision trees and Random Forest
Support Vector Machines 	Like linear and logistic regression but maximizes a margin and can handle non-linear data	Will maximize distance between classes and can generalize beyond linear	Can be slow to converge, harder to explain than linear and logistic regression
Perceptron 	Like logistic regression but can use any *activation function*	Fast and simple	Doesn't generalize well to complex problems
Artificial Neural Network (ANN) 	A network of perceptrons which can be one or more layers	Very flexible and can capture non-linear behavior	*Hparam* search required, harder to explain compared to *traditional* methods, data intensive
Conv. Neural Network (CNN) 	Small filters (i.e., convolutions) are used to extract features from data. Then the extracted data is often passed through an artificial neural network	Can automatically extract spatial-temporal features, very flexible	*Hparam* search required, harder to explain compared to *traditional* methods and ANN, data intensive
U-Network (U-NET) 	A specific CNN structure used for *image-to-image* translation tasks	Can automatically extract spatial-temporal features, very flexible, outputs an image	*Hparam* search required, harder to explain compared to *traditional* methods, ANN, and CNN, data intensive

FIG. 7. Summary graphic describing all methods discussed in both parts of this tutorial paper. The complexity of the methods increases farther down the table. Hparam is an abbreviation for hyperparameter tuning.

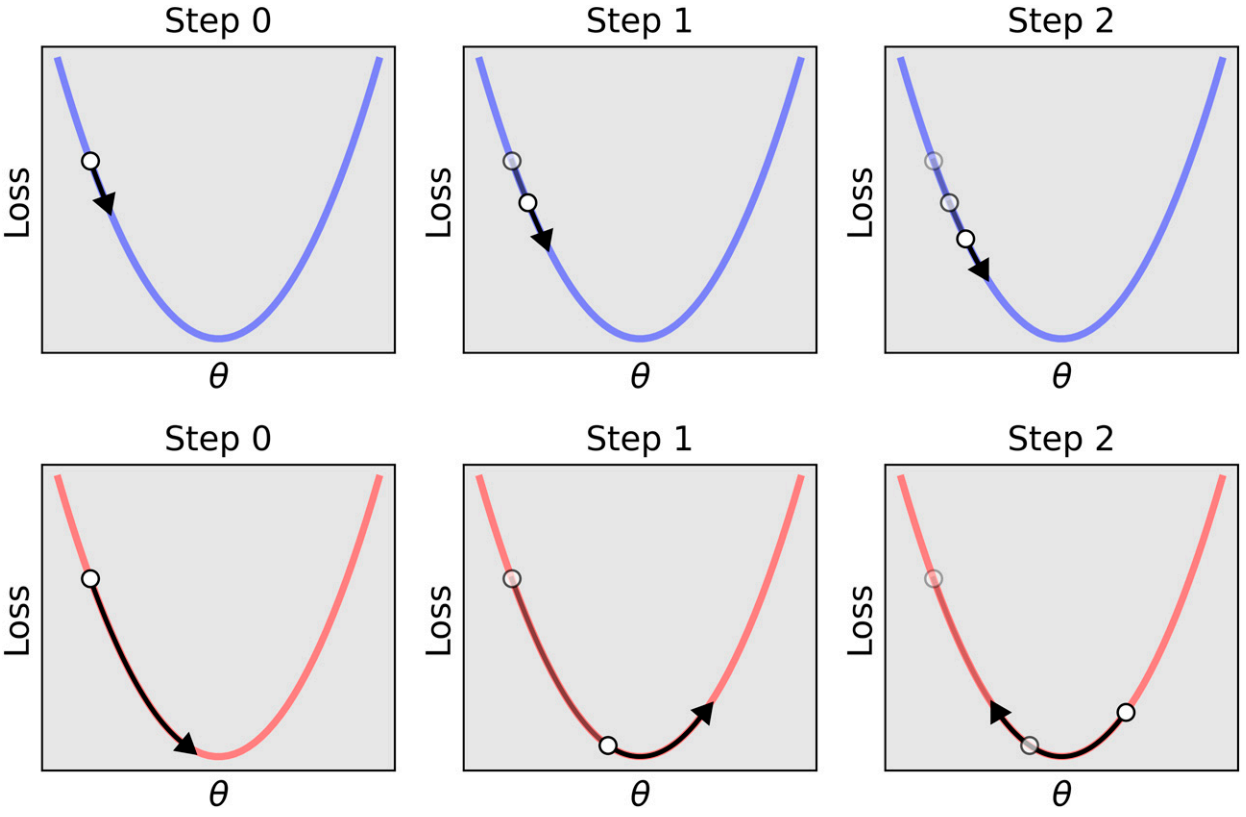


FIG. 8. Schematic depicting gradient descent. (top) The blue colored lines are using a smaller learning rate (η) than the (bottom) red colored lines. Arrows depict where loss will be after a gradient step. The θ is the symbol representing the neural network current parameters (i.e., weights and biases).

described in Part I, neural networks require a bit more description so that readers can navigate common vocabulary and methods that would be found in a paper describing a meteorological neural network.

Before Rumelhart et al. (1986), a roadblock with neural networks was the efficient and timely training of a neural network with more than a few neurons (i.e., computation took too long). As a solution, Rumelhart et al. (1986) introduced an algorithm named *backpropagation* to solve for the weights of an ANN. Backpropagation works by sequentially feeding each training data example through the network, calculating the error, and then calculating the change in error with respect to each of the weights, also known as the *gradient* (i.e., derivative of *loss* with respect to θ). Readers can think about this gradient as the quantitative amount to change the weights in the network such that error on that example is reduced. After the gradient is calculated the algorithm adjusts the weights of the network by following a *gradient descent* step:

$$\theta_{i+1} = \theta_i + \eta \frac{d(\text{loss})}{d\theta}, \tag{4}$$

where θ_{i+1} are the updated weights, θ_i is the previous weights, η is the *learning rate*, and $d(\text{loss})/d\theta$ is the gradient of the error. The learning rate is a scalar value (e.g., 10^{-3}), which tells the algorithm how large of a step to take.

To help illustrate this algorithm, consider the top row example visualized in Fig. 8. Envision the loss function on some dataset is a parabola and the neural network at the start (before any training) has a loss marked by the circle marker in the subplot labeled Step 0. After seeing a data example and calculating the gradient, the algorithm takes a step (the size of the step is determined by the learning rate) which results in the updated and lower loss in the Step 1 subplot. This is repeated, resulting in the subplot labeled Step 2. Eventually with enough steps the loss should be at a minimum (bottom of the parabola). The top row of Fig. 8 depicts an appropriate learning rate for this example. In scenarios where the learning rate is too large, like the bottom row in Fig. 8, this algorithm could end up overshooting the minimum and never converging to the smallest loss. Conversely if the learning rate is too small (not shown), the algorithm will take too long to converge on the minimum loss. For these reasons, the learning rate is a hyperparameter that is varied (i.e., multiple values like 0.01, 0.001, and 0.0001 are tried) when training neural networks. Recall from Part I 1 that hyperparameters are human designated choices in a machine learning models configuration (e.g., the number of nodes) or training procedure (e.g., the learning rate) that are varied.

Unfortunately in practice, calculating the gradient on every single training example can be too costly since the typical

number of training examples is on the order of thousands to millions and the number of weights in a neural network can be similarly large. Thus, a trick around this is to use something called *stochastic gradient descent*. The idea is instead of calculating the gradient and updating the weights after seeing each example, a random *batch* (i.e., small collection, subset) of examples is used to estimate the gradient which is then used to inform the algorithm how to change the weights. The size of the batch, like the learning rate, is another hyperparameter of neural networks. The new procedure is then, select one random batch, send the batch of data through the network, calculate the loss, calculate the mean gradient of the batch and update weights [i.e., take step according to Eq. (4)]. This sequence is repeated until all training data have been sent through the neural network. After the entire training dataset has been passed through the network, the network has been trained for one *epoch*. Neural networks are often trained for many epochs (e.g., 50, 100, 1000, etc.) usually until the loss does not change much (i.e., changes less than 10^{-6})[6] or when overfitting is detected.

In practice, stochastic gradient descent is just one method of *optimizing* a neural network. Other *optimizers* can be used to train neural networks, but for the sake of this tutorial, they all generally follow the same steps as stochastic gradient descent. The names of other popular optimizers a meteorologist might encounter are the following: the adaptive moment estimation (Adam; Kingma and Ba 2017) and root-mean-square propagation (RMSprop; Hinton et al. 2012).

2) Loss functions

Just like the traditional machine learning methods, neural networks can be used for both categories of supervised machine learning: classification and regression. The primary differences between a neural network for classification and a neural network for regression is which loss function is optimized and what output activation is chosen (i.e., activation of the last node or layer). For classification, typical loss functions include *binary cross-entropy* and *categorical cross-entropy* accompanied with a sigmoid [see Fig. 3 and Eq. (6) in Part I] or softmax (a variant of sigmoid) output activation function for binary and multiclass classification tasks, respectively. Meanwhile for regression, common loss functions include mean absolute error and mean squared error accompanied with a linear output activation. More sophisticated loss functions can be used, like the fractions skill score (Roberts and Lean 2008), and are an active area of research within machine learning for meteorology (Ebert-Uphoff et al. 2021; Lagerquist and Ebert-Uphoff 2022).

3) Regularization and overfitting

Neural networks can often contain hundreds, thousands or even millions of trainable parameters. While this enables neural networks to be very flexible, it can also enable the network to overfit to the training data very easily. Thus, there are some specialized methods that can help prevent overfitting (i.e., regularize) the neural network. A popular method of regularization is called *dropout* (Srivastava et al. 2014). Dropout is where neurons within a layer of the network are randomly turned off (set to 0) in the training process. The neurons that are turned off are changed after each batch, but the percentage of neurons turned off in the layer is constant over the training time and is a hyperparameter choice (e.g., 10% of neurons). Then when the model is used in *inference mode* (i.e., prediction time) the dropout layers are not used, meaning all the neurons are used. The intuition behind dropout is that if random neurons are turned off during training, the network is forced to learn redundant pathways and cannot simply memorize a single pathway through the network for each example it sees.

A second regularization method commonly used is called *data augmentation*. Data augmentation are synthetic alterations made to the training data. These alterations include things like random rotations, random flips (up–down or left–right or both), and adding random noise. The reason this is done is because adding these slight alterations provides the neural network with slightly different examples to learn from, which in turn makes your neural network model more resistant to overfitting and more robust to things like measurement errors. Data augmentation is also a way to increase your training sample size without having to actually add more data (see Lagerquist et al. 2020b, for a meteorological example using data augmentation).

A third method of regularization is called *batch normalization* (Ioffe and Szegedy 2015). Batch normalization, as the name suggests, normalizes the values of a batch of data within the neural network. The reason for this stems from the use of batches themselves, which are needed for timely training of neural networks. Because the training process randomly selects a batch of data to estimate the gradient from, that batch of data is not guaranteed to have properties that are well suited for stable training, like being normally distributed. Thus, to assure that training goes as smoothly as possible, batch normalization layers can be inserted after any layer in a neural network.

4) Hardware

A meteorologist will likely encounter discussions of what hardware (i.e., computer details) is being used to do the neural network training. This discussion comes from the issue that training a neural network can be computationally very slow on a normal computer [i.e., central processing units (CPU)]. As a way to speed things up, the open-source neural network software packages, named Tensorflow (Abadi et al. 2015) and PyTorch (Paszke et al. 2019), have built their software to allow users to utilize a computer chip called a graphical processing unit (GPU). The GPU enables the calculation of the convolution of an image and the gradients to be much faster, which ultimately accelerates training. While there are many different types of GPUs and CPUs and many different neural network tasks, in general a GPU can often reduce training time by a factor of 2–10. The Google Colab notebooks (see data availability section) that accompany this

[6] This can be user defined, but 10^{-6} is a common choice.

manuscript leverage the freely available GPUs provided by Google in the cloud.

3. Neural network application and discussion

a. Problem statements

Here we restate the machine learning problem statements explored in this paper. We again apply the Storm Event Imagery (SEVIR; Veillette et al. 2020) dataset to two main tasks: 1) Does this image contain a thunderstorm? and 2) How many lightning flashes are in this image? For more information about the SEVIR data see Part I. We assume the GOES Lightning Mapper (GLM) observations are unavailable and we need to use the other measurements (e.g., infrared brightness temperature) as features to estimate if there are lightning flashes (i.e., classification), and how many of them are there (i.e., regression). Both tasks 1 and 2 are centered on using machine learning models having a singular (i.e., one dimensional) output. As we mentioned in section 2a(4), U-Nets offer more than a single output (i.e., two dimensional), as they recreate an entire image as an output. Thus, the problem statements for the U-Net application are then as follows: 1b) Label the pixels in this image where there are lightning flashes; and 2b) For each pixel, diagnose the number of flashes in that pixel.

b. Data

Before jumping into the results of the trained neural networks, we want to emphasize an intersection between neural networks and the traditional methods discussed in Part I. The discussion from Part I (their section 3b) regarding data curation applies to neural networks as well. Specifically, a dataset used to develop a neural network must also be split into independent subsets for training, validation, and testing the model. Thus, to follow Part I we use the same datasets with a slight alteration. While the original SEVIR dataset is primed for successful machine learning, the size of the raw dataset (approximately one terabyte in storage size) is cumbersome for the intents of a tutorial and could not be used on most personal computers. Thus, as an effort to make this dataset more accessible as a tutorial, we have reduced its size. To do this, we first reduced all images to have the same resolution of the gridded lightning data (48 × 48 pixels; approximately 8 km × 8 km pixels). Figure 9 shows an example of the full resolution visible image and its corresponding low resolution version. After the images were resampled, one random continuous hour (12 images) of the four total hours (48 images) for each storm event is kept. Since we are keeping the same number of storm events, we keep the training, validation and testing data splits the same as Part I, which were 1 January 2017–1 June 2019 for training and split every other week in the rest of 2019 into the validation and test sets. Doing both of these resamplings of SEVIR results in a more manageable dataset (approximately two gigabytes in storage size), while also preserving 60 000 training samples and about 12 000 validation and test samples. We name this subset of SEVIR: *sub-SEVIR*, and the location of the dataset can be found in the data availability section.

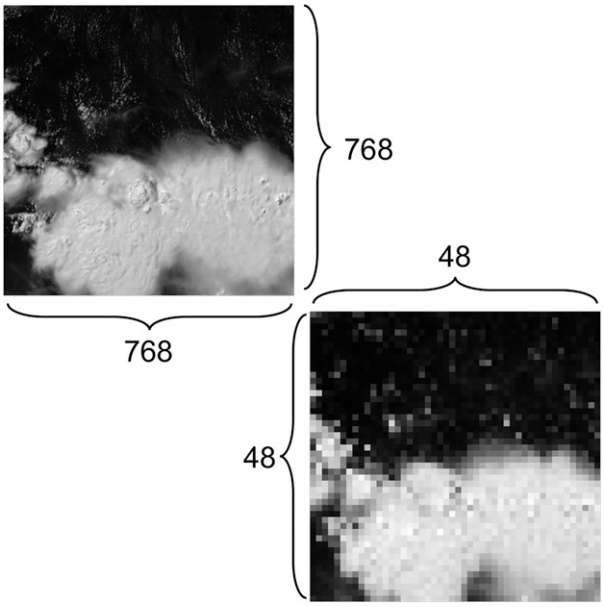


FIG. 9. Example of (top left) the full resolution visible image and (bottom right) its reduced resolution in sub-SEVIR. The numbers correspond to the number of pixels along each dimension.

Owing to reduced resolution, the sub-SEVIR dataset contains different information. Given our goal of comparing the neural network models of this paper to the machine learning methods of Part I, we must re-extract the same features from sub-SEVIR. Specifically, we extract the following percentiles: 0, 1, 10, 25, 50, 75, 90, 99, 100. These percentiles are then used as input features for retraining the traditional machine learning methods and to serve as a baseline comparison with trained neural networks.

c. Training the networks

After reading the section on how to train neural networks (section 2b), the reader might notice that there are numerous hyperparameters for neural networks. In Part I, the traditional machine learning models shown were trained with the default hyperparameter choices as defined by the scikit-learn Python package (scikit-learn; Pedregosa et al. 2011). The idea of default hyperparameter choices does not necessarily exist with neural networks. Thus, it is good practice for those training neural networks to run some sort of hyperparameter search (i.e., vary a bunch of the parameters) because users are not guaranteed to get good performance with some starting parameter choices. For example, recall the discussion about choosing a learning rate in section 2b(1).

All the trained neural network models shown here are the result of a hyperparameter search. We conducted 100 random hyperparameter configurations for each neural network trained and systematically varied things like the number of layers, the number of neurons, the loss etc. In the end we chose one of the 100 models to show in the following section. These were chosen based on their performance in the validation set. For readers interested in the exact hyperparameter choices we varied to

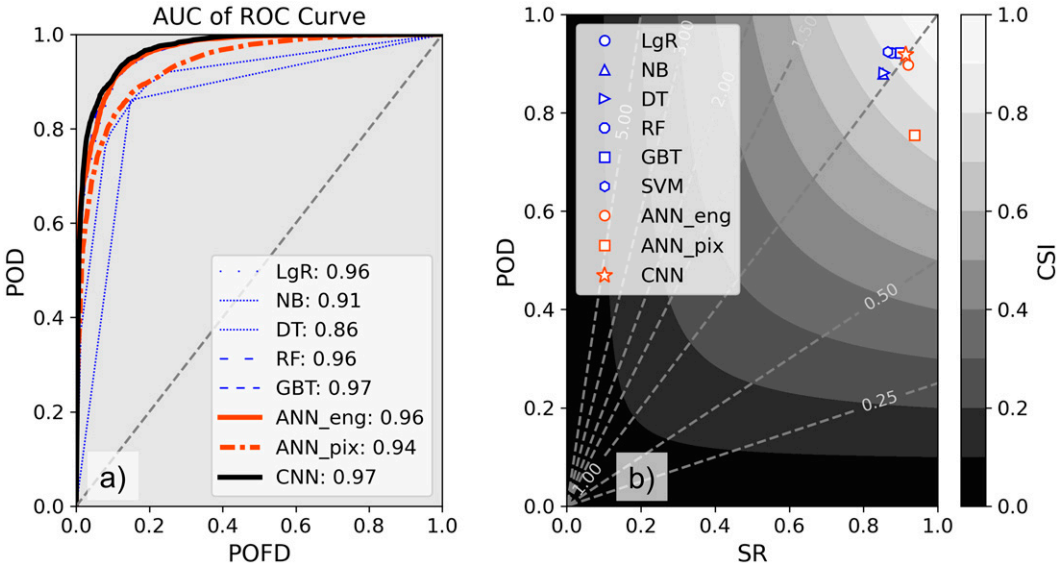


FIG. 10. Classification metrics (a) ROC curve diagram. All thin blue lines are the traditional machine learning methods from Part I [logistic regression (LgR), naïve Bayes (NB), decision tree (DT), random forest (RF), and gradient boosted decision tree (GBT). The thick lines are the neural networks trained (ANN_{eng}: ANN trained with engineered features; ANN_{pix}: ANN trained using pixels as features). The values in legend are the AUC values. (b) The performance diagram for each machine learning model (Roebber 2009).

find the best performing models, see Figs. A1–A3 in the appendix.

d. *Classification*

The first machine learning task we consider is 1) to classify a SEVIR image if it has at least one lightning flash within it. To serve as a comparison, all of the traditional machine learning methods from Part I were retrained on the sub-SEVIR dataset. Their performance on the validation dataset is shown as the thin blue lines and blue markers in Fig. 10.

For task 1, we also trained three neural networks. The first is an ANN trained with the same input features (i.e., the table of percentiles extracted from each image) as the traditional machine learning models (red solid line and red circle Fig. 10). This ANN trained on the engineered features (i.e., the percentiles of the image) effectively reproduces the performance of the best traditional machine learning methods (e.g., gradient boosted trees). Since there is a relatively similar performance of random forest, gradient boosted trees, and ANN, it would be better to use the tree-based methods over the ANN operationally for this task. This is suggested because tree-based methods are less complex and thus more *interpretable* (cf. Fig. 1 in Flora et al. 2022a). Using a less complex and more interpretable model also provides a better opportunity to meet the *consistency* point made by Murphy (1993).

The second neural network trained is another ANN but this time it was trained using each pixel as a feature (e.g., Fig. 2). The reason a second ANN is trained is to see if the ANN could learn important features on its own, without a domain scientist (i.e., meteorologist) extracting pertinent information (i.e., the percentiles from the satellite images). While the pixel trained ANN has generally good skill (AUC $>$ 0.9, CSI $>$ 0.7 red; dash-dot line Fig. 10a; red square Fig. 10), the result is worse than all other methods discussed so far (Fig. 10b).

The last neural network trained for task 1 is a CNN. To be explicit, recall that the CNN uses the raw images as inputs and convolves them to extract features. The result of training a CNN on the sub-SEVIR data provides one of the best performing machine learning methods (black line Fig. 10a; red star Fig. 10), matching the skill of the gradient boosting trees and the ANN trained on the engineered features. Note that the CNN only marginally outperforms the other methods on the performance diagram and is likely not a significant difference.

It might be surprising to see that the ANNs do not substantially outperform the tree based methods on this task despite the added complexity of neural networks and their training. This is a common pitfall for machine learning users. In fact, there is growing evidence that the tree-based methods can often outperform neural networks and deep learning on *tabular data* (i.e., data contained in a spreadsheet; Shwartz-Ziv and Armon 2022). A distinction is made between tabular and nontabular datasets here because spatial details can contain substantial information for the machine learning task and is not always easily quantified into a tabular dataset. For example, consider assessing a storm's tornadic potential. While using composite radar reflectivity as a feature could be useful (e.g., strong reflectivity value means a strong storm), there is likely more information contained in the shape of the radar echo (e.g., is there a hook echo?). Thus, given the amount of

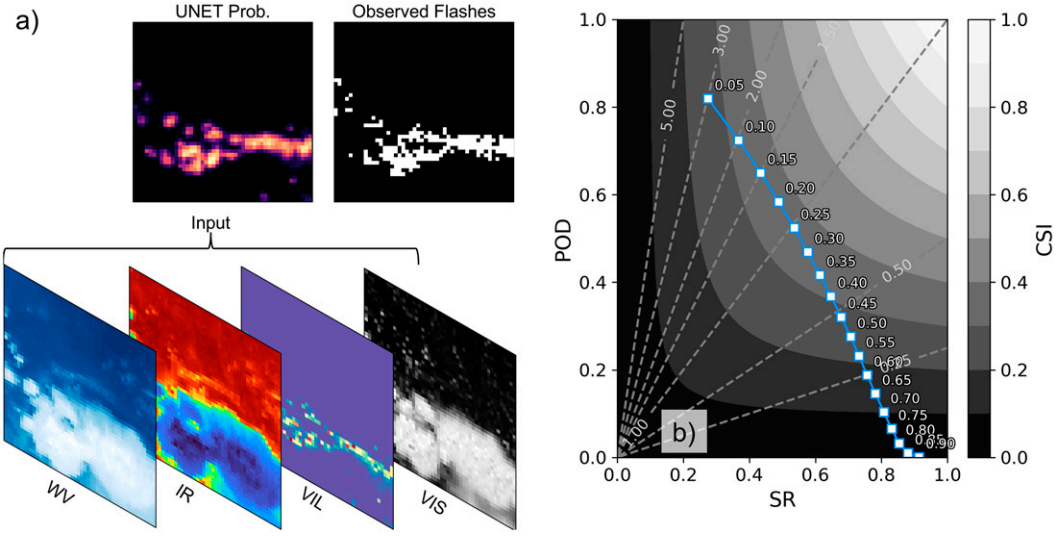


FIG. 11. Trained U-Net results for classification. (a) Example input and output of the U-Net. (b) Performance diagram for the U-Net evaluated on every pixel. The numbers next to the markers show the probability threshold to classify a pixel as containing lightning or not.

additional effort required to explore the hyperparameters in neural networks, our suggestion is that if you have a tabular dataset, start with random forest and gradient boosted trees for your machine learning model. Often times this will result in a useful machine learning model without the headache of doing a large hyperparameter search or needing specialized computers (i.e., GPUs). Otherwise, if you have a spatial dataset (e.g., radar images) and you are unsure of what features to extract, then the extra effort of CNNs could be beneficial.

Moving beyond the single output models, a U-Net for classification is also trained and evaluated. Recall that a U-Net (e.g., Fig. 6) outputs a map with the same shape as the input images. In other words, a U-Net tasked with problem statement 1b (in section 3a) produces an output map where each pixel is assigned a probability of it containing lightning. An example output is shown in Fig. 11a. A U-Net can be evaluated similarly to the previous models and a performance diagram for the trained U-Net is shown in Fig. 11b. Note on Fig. 11b that instead of a single marker the figure shows a line with many markers. This is because the threshold for deciding if a pixel is labeled as no lightning or lightning is varied from zero to one at 0.05 increments. This is done because a model could potentially get better results if a probability threshold other than 0.5 is used (which was shown in Fig. 10), which is the case for this U-Net. Note that this could also be done with the all of the other machine learning methods (except support vector machines) shown in Fig. 10b, but the threshold of 0.5 generally works well for those models.

Comparing Figs. 10b and 11b, initially it seems like the U-Net is performing worse than the ANNs and CNN because the line on Fig. 11b is well below the location of all other models in Fig. 10b. That being said, it is unfair to compare the two sets of performance statistics because the U-Net is being evaluated on every single pixel rather than on the image as a whole. Given the added complexity in problem statement 1b, the U-Net performance is encouraging with CSI values of 0.36 when using a probability threshold of 0.25. This offset from the probability threshold of 0.5 happens frequently in meteorology and can be mostly attributed to rare phenomena and the training dataset being imbalanced.

It might not seem like the lightning flashes are rare, but if you consider the total number of pixels that contain lightning, they make up less than one percent of the total amount of pixels. Thus, given the number of no-lightning pixels far outweigh the lightning pixels, the U-Net will learn this natural distribution and skew its output to account for the more likely outcome. The result is that on the performance diagram, a lower probability threshold can perform better than using the default 0.5. While altering the probability threshold for pretrained models can improve performance, other mitigation techniques can be taken and are focused on adjusting the ratio of nonzero pixels to zero pixels in the training dataset.

One way to adjust the ratio of pixels is by subsampling the 48×48 pixel images into smaller *patches* (e.g., 24×24) and only train on patches that have a larger proportion of nonzero pixels. This tends to work well but is more resource intensive because patching the data requires the user to then *stitch* the patches back together while using the model output. Another way would be to adjust or change the loss function to weight the classes differently. By default most loss functions weight all classes equally. There are ways to adjust the loss function and tell your machine learning model that the rare classes are more important than the training data suggests. For examples of custom loss functions see Ebert-Uphoff et al. (2021). Alternatively, one could do both subsampling and a differently weighted loss function. These alterations can be considered part of the hyperparameter tuning of the U-Net training. A meteorological example of exploring various U-Net training procedures can be found in Mounier et al. (2022), where a U-net is used to identify bow echoes. Note that weighted loss

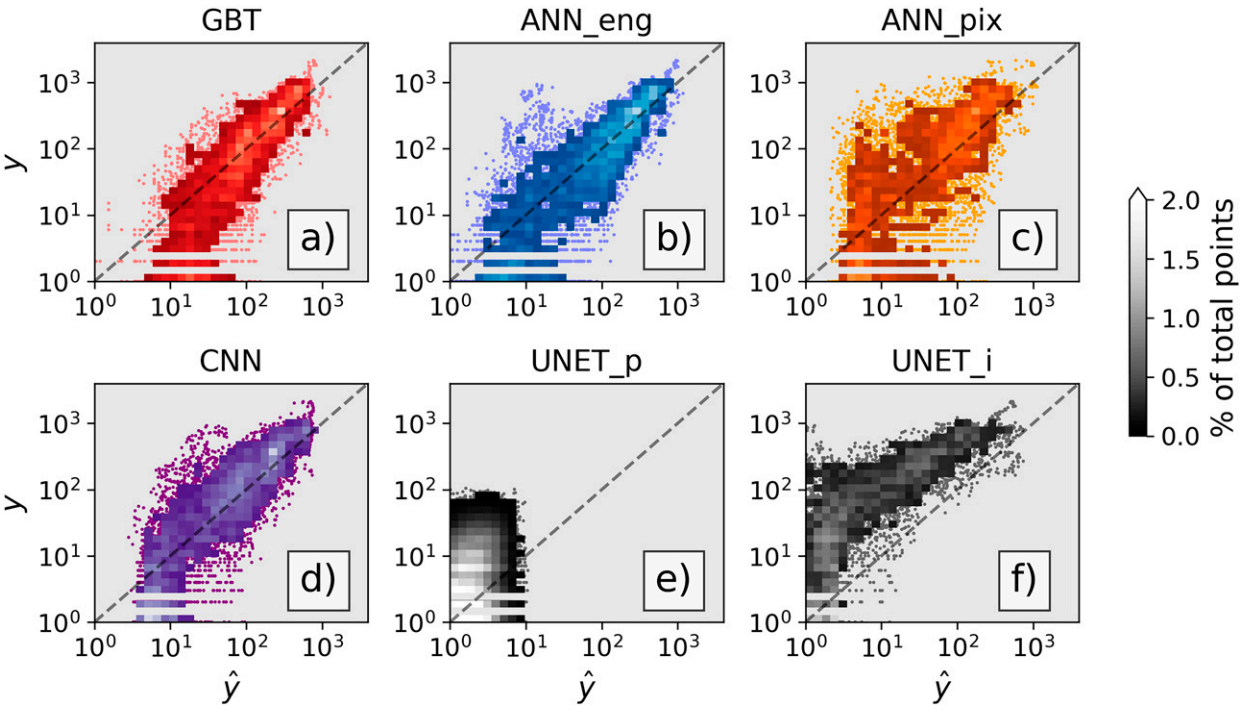


FIG. 12. One-to-one diagrams with all regression methods trained on sub-SEVIR. The x axis is the machine learning model prediction ($\hat{y}$) and the y axis is the truth (y): (a) gradient boosted trees; (b) artificial neural network using the tabular data; (c) artificial neural network using the pixel data; (d) convolutional neural network; (e) U-Net evaluated on every single pixel in the images; and (f) the U-Net evaluation, but on the sum of all lightning flashes in an image.

functions and resampling the training data is not exclusive to U-Nets. These methods can be explored for all neural networks.

e. *Regression*

Considering task 2, the goal of the machine learning is to now determine the number of flashes that are occurring in a SEVIR image. Like the previous section, the goal is to compare the neural network methods presented in this paper to the traditional machine learning methods of Part I. To make the comparison more concise, we only show the best performing regression model trained on the sub-SEVIR dataset, which was the gradient boosted trees. Recall that for this regression task, only data examples that had more than one flash in them were used as the training data. The performance of the gradient boosted tree on the validation dataset is shown in Fig. 12a and the red bar in Fig. 13.

For task 2, a similar suite of neural networks as the classification task are trained and their performance is characterized in the same way as regression in Part I. The first neural network trained is the ANN using the engineered features as inputs. Akin to the results of the classification task, this ANN achieves similar performance to the gradient boosted trees. The ANN has a high density of points that follow the diagonal in Fig. 12b and has a mean absolute error, root-mean-square error, and R^2 values very close to the gradient boosted tree (blue bar Fig. 13). That being said, the bias of the ANN is larger than the bias of the gradient boosted tree (Fig. 13).

A second neural network trained is an ANN using the pixels as features. It is clear that this model has issues. The points on the one-to-one plot are more spread out and not highly concentrated along the diagonal (oranges Fig. 12c). All metrics are worse compared to the ANN and gradient boosted tree trained on the engineered features. This result is very similar to the classification model, where the model has some skill but performance is considerably worse when the ANN has to learn what features are important based on the pixels as input.

A third network trained is a CNN. The CNN achieves similar performance to the ANN trained on the engineered features and the gradient boosted tree. The points are more densely aligned along the diagonal in Fig. 12d (purples), and the quantitative metrics (purple bar Fig. 13) are effectively the same as the ANN, but it does have worse bias (Fig. 13a). Thus, like the classification task the CNN was able to extract relevant features to make a skillful designation of the number of lightning flashes in the image.

A regression U-Net is also trained and evaluated. Instead of determining the probability of lightning in each pixel, the designation of the regression U-Net is the number of flashes in each pixel. A similar problem occurs with the regression U-Net as with the classification U-Net when trying to compare the U-Nets to the other neural networks. Consider the pixelwise evaluation of the regression U-Net (Figs. 12e and 13). The U-Net has a clear underestimation of the number of flashes compared to the observed flashes and yet the mean absolute value, bias, and root-mean-square error are close to

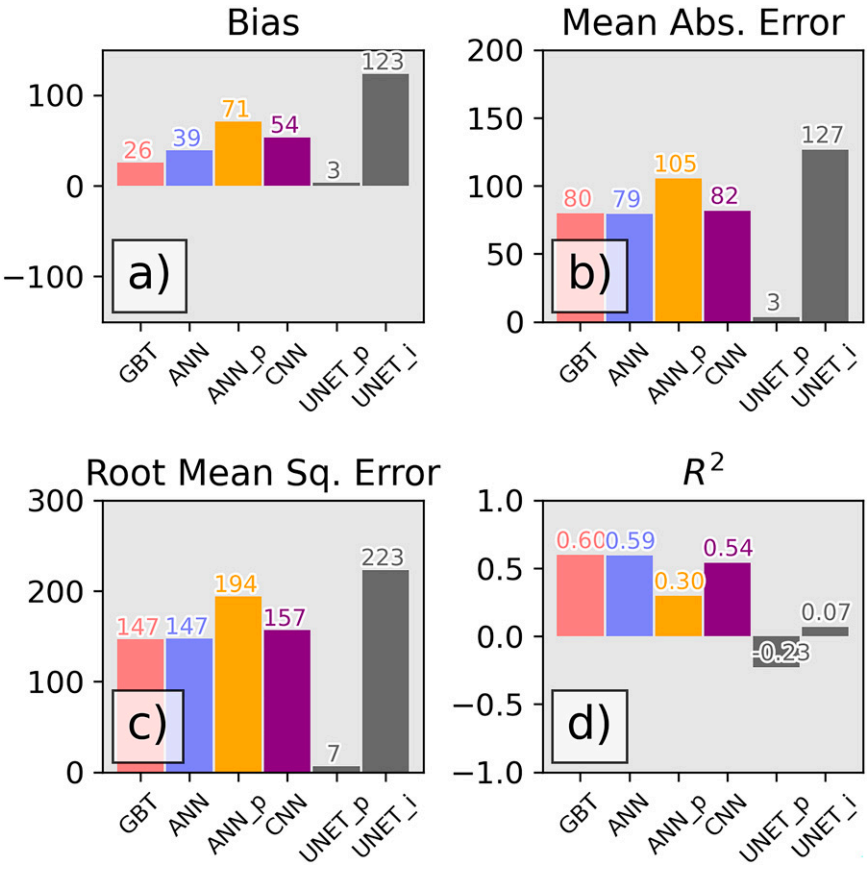


FIG. 13. Metric bar charts with all regression methods trained on sub-SEVIR.

zero. This evaluation might initially seem contradictory, but the pixelwise distribution of flashes is two orders of magnitude smaller than the image wise number of flashes (e.g., mean pixel number of flashes is 3 while mean image number of flashes is 150). Since the magnitude is smaller, the metrics are correspondingly smaller. Thus, the comparison of this U-Net to the other neural networks is not necessarily fair.

As an alternative evaluation, the sum of U-Net predicted flashes across all pixels in an image can be calculated. The sum of all flashes in the images results in approximately an order of magnitude offset in the designation (Fig. 12f). Like the underestimation of the U-Net in the classification example, the regression underestimation probably occurs because skew of the distribution of pixels with lightning and pixels without lightning. The regression example is further compounded by the strong left skew (i.e., toward zero flashes) in the distribution of pixels with lightning flashes. The previously discussed mitigation techniques for the classification U-Net can also be applied to regression (e.g., changing the loss, patching etc.).

f. Explainable artificial intelligence

As mentioned in the motivation of Part I, machine learning methods are often seen as *black boxes* where the user cannot see what the machine learning is using to make its decisions and predictions. To combat the opaqueness of machine learning methods we present two methods, permutation importance and accumulated local effects, which can be applied to the traditional machine learning methods that made the black box more transparent. Here we show something similar but applied to neural networks. In machine learning, the methods used to *explain* a machine learning output are commonly referred to as explainable artificial intelligence (XAI). The XAI field is a place of active research development and readers can see Flora et al. (2022a,b) for additional discussion of XAI techniques for the traditional machine learning methods and Mamalakis et al. (2022a,b) for XAI techniques for neural networks. Know that the following discussion and examples only show the XAI techniques applied to CNNs, but these techniques can be applied to all the neural networks discussed in section 3.

1) PERMUTATION IMPORTANCE

The first XAI method shown here is the same as Part I, permutation importance (Breiman 2001; Lakshmanan et al. 2015). We show this method because it is a powerful method that can help users understand which inputs to the machine learning model are most important. Also, we choose to show this technique because of how flexible the method is to be used on any machine learning method.

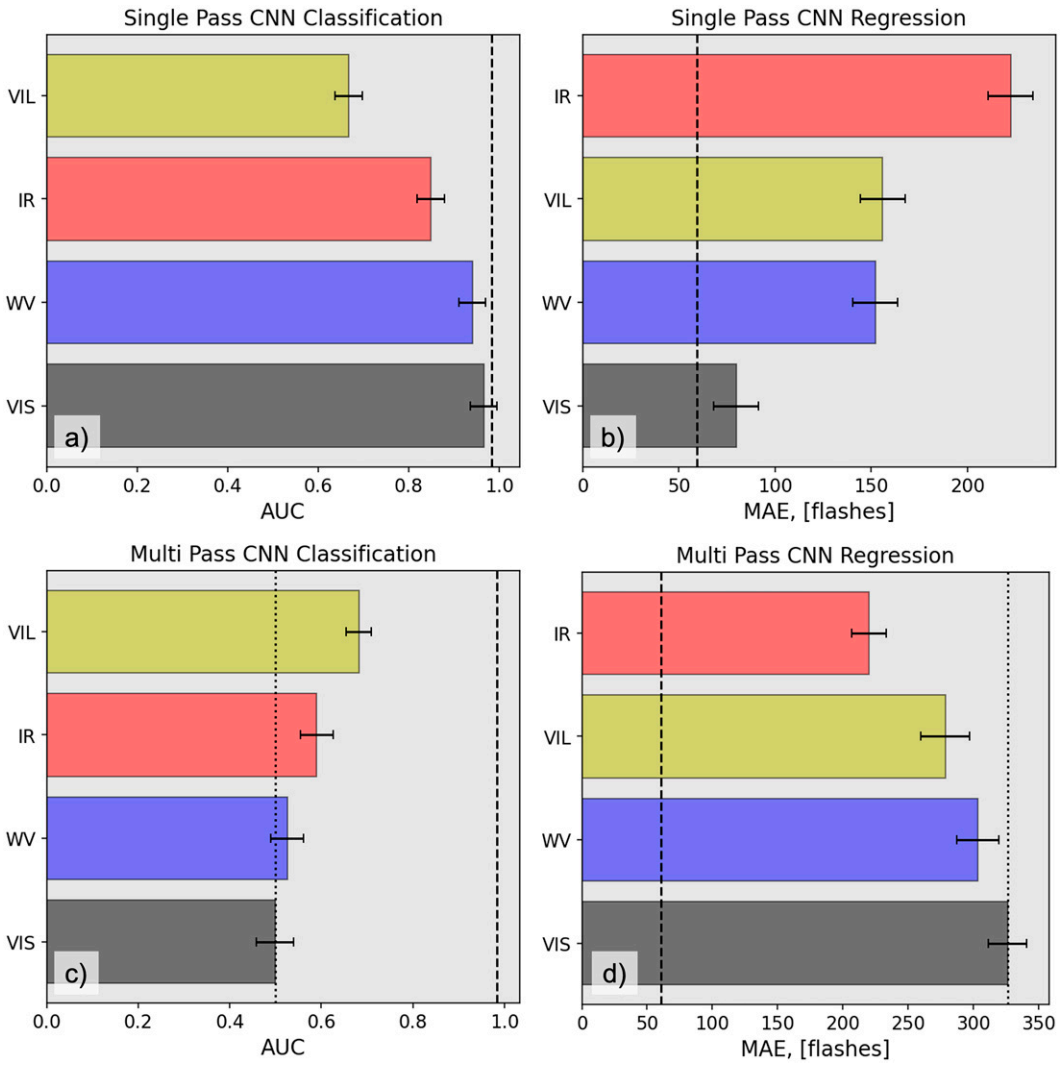


FIG. 14. Permutation importance results of the convolutional neural networks for both classification and regression. The yellow bar is the vertically integrated liquid, the red bar is the infrared, blue is the water vapor, and black is the visible. (a),(b) The single-pass backward results and (c),(d) multipass backward results. (left) Classification and (right) regression. All results are computed on 30 random samples of 250 images from the training dataset. The dashed line is the original score before shuffling any features, while the dotted line is the final score for shuffling all variables (only multipass).

The general procedure is the same as discussed in Part I [section 3c(1)]. Input features are shuffled one by one, such that the change in the desired metric quantifies the feature's importance to the machine learning model (i.e., single pass). Since we are doing this technique on images, the difference from Part I is that first the pixels within an image are shuffled, then the order of images is shuffled to properly make the input features random. From there the procedure is exactly the same as in Part I.

Figure 14 is an example of permutation importance applied to both the CNN for classification and the CNN for regression. The interpretation of this figure is the same as Part I, but now features are grouped according to the variable from which they originated. For example, the single pass result shows that the vertically integrated liquid is the most important feature for diagnosing if an image has at least one flash in it, while the infrared channel is the most important feature for determining the number of flashes in an image. Remember that the most important variable is identified by how much skill is reduced when shuffling that specific variable while keeping the other variables unchanged. Figure 14a shows the vertically integrated liquid reduces AUC by about 0.4 (i.e., from an AUC near 1 to an AUC near 0.6), and Fig. 14b shows the MAE increases from about 50 to more than 200 flashes when the infrared is shuffled. The multipass method (Figs. 14c,d) also shows that the vertically integrated liquid and infrared are the most important features for the classification and regression task, respectively. Recall from Part I that to interpret the

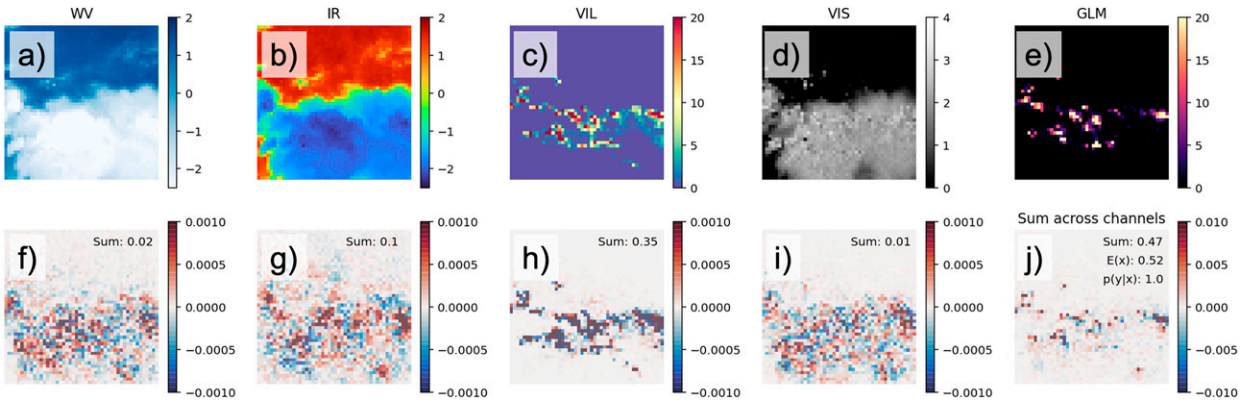


FIG. 15. SHAP values for an example in the SEVIR dataset (19 Aug 2019). (a) Water vapor, (b) infrared, (c) vertically integrated liquid, and (d) visible. Note that values for (a)–(d) are scaled and are thus unitless. (e) GOES Lightning Mapper number of flashes in each pixel. (f)–(i) SHAP values for each respective channel of the input. The sum of all SHAP values in the image is annotated in the top-right corner. (j) Sum of all SHAP values across the channels with the total sum, expected value, and the ML output is written in the top-right corner.

multipass result the successive reduction in skill is how to see the most important features. Thus, notice that the reduction in skill from the dashed black line in Figs. 14c and 14d is largest compared to the change from the top bar to the second bar, and the second bar to the third bar etc. For this example, the multipass method shows the same result as the single pass method, but this is not always the case.

These designated important features make sense meteorologically. Vertically integrated liquid can be interpreted similarly to radar reflectivity. If one knows where to look at an image where there is no radar reflectivity measured, it would be simple to say there is no lightning in the image. Meanwhile, since the regression task is evaluated on only examples that have at least one flash, the model is leaning more heavily on infrared. This could be because the amount of cold cloud tops in an image is plausibly related to how much lightning is in the image (e.g., more updrafts can lead to more clouds which could lead to more lightning), but further testing would need to be done to confirm or deny this explanation of the machine learning reasoning.

2) DEEP SHAP

The second XAI method we discuss is called Deep SHAP (Lundberg and Lee 2017), which estimates Shapley values (i.e., SHAP values; Shapley 1953) that quantify the effect each input feature contributes to the total machine learning model output. SHAP values are calculated using a branch of mathematics called game theory, which enables the SHAP values to consider interactions between features (e.g., water vapor is correlated to infrared brightness temperature) while also allowing timely computation. While understanding how SHAP values are exactly calculated can be complicated, their interpretation is relatively straightforward and has some attractive properties.

Consider an example of SHAP values for the classification CNN model on one of the examples (Fig. 15). We can see in this example that there is deep convection in the bottom half of the image, characteristic of cold cloud tops (Figs. 15a,b), large vertically integrated liquid values (Fig. 15c) and lots of observed lightning (more than 1000 flashes in this 5-min observation; Fig. 15e). Using Deep Shap, the estimated SHAP values for each feature are shown in the corresponding image below the input data (Figs. 15f–j). The way to interpret SHAP values are that negative values (blue colors in Figs. 15f–j) have negative attribution, or contribute negatively to the output (i.e., evidence against lightning in the image), while positive values (red colors in Figs. 15f–j) have positive attribution (i.e., evidence for lightning in the image).

A general interpretation of the SHAP values in Fig. 15 is that the ML model is using pixels where there are clouds for its output (i.e., SHAP colors show up where cloud is). While this might seem like an unimportant result, it is never guaranteed that the ML model will use logical decision techniques. There have been notable examples in the computer science literature where the ML identifies unexpected parts of an image to do its output, like a copyright symbol or a company logo (Lapuschkin et al. 2019). It is then encouraging that the ML is considering the clouded part of the image to diagnose if there is lightning within it. Another interpretation from the SHAP values is that the clouded region contributes both positively and negatively to determining if there is lightning in this image. This decision making process is not expected but could be a result of the ML task of determining if there is at least one flash in the image and not determining *where* in the image the lightning is. Since the ML task is for the entire image, then the SHAP values should also be interpreted more holistically where the sum of the SHAP values across the clouded area can be compared against the sum outside the clouded area, where the sum is larger in the clouded region (i.e., more red than blue).

The summation of SHAP is enabled by its additive formulation. By design the SHAP values, when added to the expected value (i.e., mean output from all images) results in the output of ML model. This additive property enables more

than the discussion above of the clouded and nonclouded region, but also the relative importance of every input channel to the output of the ML. For example, consider the channel wise SHAP sums in the top right corner of Figs. 15f–i. The SHAP values for vertically integrated liquid are the largest sum with a value of 0.35, followed by infrared brightness temperature with a value of 0.1, and then water vapor and visible with values of 0.02 and 0.01, respectively. This is a similar result to the permutation importance result which provided evidence that the vertically integrated liquid is the most important input variable. Last if you consider some of the SHAP values in a pixelwise sense (Fig. 15j), the SHAP values mainly outline the edges of the vertically integrated liquid input channel.

The additive property of SHAP values can be extended beyond this *local* (i.e., one sample or case) explanation. The SHAP values can be summed across all dataset examples to get a similar *global* explanation to what permutation importance gave us. The channelwise sum across all examples in the validation dataset are shown in Fig. 16. The result is the same as permutation importance, showing that the vertically integrated liquid is most important, followed by the infrared brightness temperature, water vapor brightness temperature and then visible reflectance. It is encouraging to get the same result from two different XAI methods, which builds confidence in the end result.

While the SHAP discussion has been centered on the classification task, the same analysis can be done on the regression task but is not done here for brevity. Similarly, the discussion in this paper has been focused on neural networks, but SHAP can also be applied to the traditional methods of Part I. For more examples of SHAP being used in the meteorological research readers can look over these references: Gensini et al. (2021), Griffin et al. (2022), Mamalakis et al. (2022a,b), van Straaten et al. (2022), and Flora et al. (2022a,b).

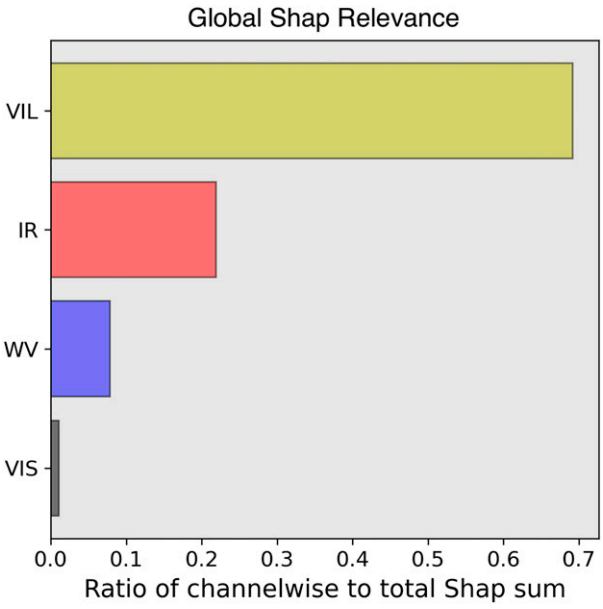


FIG. 16. Global SHAP ratio on the validation dataset. The channelwise ratio (i.e., sum across each input variable) of all SHAP values to the total SHAP sum. These SHAP values were evaluated on the entire validation dataset.

4. Summary

This manuscript is the second of a pair of machine learning tutorial papers designed for the operational meteorological community. The main focus of this paper was the plain language discussion of neural networks. More specifically the neural networks discussed included artificial neural networks (ANNs; i.e., multilayer perceptrons), convolutional neural networks (CNNs), and U-shaped networks (U-Net). Similar to Part I of this tutorial series (i.e., Chase et al. 2022), the goal of this paper was to provide an overview of the many terms involved in neural networks while also providing entry level intuition of each method and their training procedures. Furthermore, the same simple meteorological example using the Storm Event Imagery dataset (SEVIR; Veillette et al. 2020) to identify lightning presence and amount was reconducted with the neural network methods to allow for direct comparison of all machine learning methods discussed in both parts of the series. The following list explicitly summarizes the results of this paper:

1) Discussed the various nuances and terms associated with neural networks (section 2)
2) Discussed three different neural network architectures in detail (section 2a)
3) Demonstrated a classification and regression task to diagnose the presence and number of lightning flashes in a satellite image (sections 3d and 3e)
4) Showed two explainable artificial intelligence techniques applied to a CNN (sections 3d and 3e)
5) Released python code to conduct all steps and examples in this manuscript (see the *data availability statement*)

As technology continually advances, unprecedented meteorological measurements and simulations will continue to occur. For example, the GOES-R series of geostationary satellites provides 0.5-km grid spacing of visible imagery that was only previously obtainable from polar orbiting satellites (e.g., MODIS). Another example includes the growing efforts to begin global simulations of weather using convective-allowing horizontal grid spacing (e.g., less than 4 km; Stevens et al. 2019). With these improved measurements and simulations come daunting increases of dataset sizes and then potentially information overload (i.e., too much data to use). Thus, it is imperative that meteorologists are familiar with tools that can reduce their individual burden. Machine learning is poised to handle the future terabytes/petabytes of meteorological data and potentially can provide valuable tools for meteorologists to make trustworthy and well-informed data-driven decisions.

Acknowledgments. This material is based upon work supported by the National Science Foundation under Grant ICER-2019758, supporting authors RJC and AM. Author

DRH was provided support by NOAA/Office of Oceanic and Atmospheric Research under NOAA–University of Oklahoma Cooperative Agreement NA21OAR4320204, U.S. Department of Commerce. The scientific results and conclusions, as well as any views or opinions expressed herein, are those of the authors and do not necessarily reflect the views of NOAA or the Department of Commerce. We want to acknowledge the work put forth by the authors of the SEVIR dataset (Mark S. Veillette, Siddharth Samsi, and Christopher J. Mattioli) for making a high-quality free dataset. We would also like to acknowledge the open-source Python community for providing their tools for free. Specifically, we acknowledge Google Colab (Bisong 2019), Anaconda (Anaconda 2020), scikit-learn (Pedregosa et al. 2011), Pandas (Wes McKinney 2010), Numpy (Harris et al. 2020), and Jupyter (Kluyver et al. 2016).

Data availability statement. As an effort to accelerate the use and trust of machine learning within meteorology we have supplied a github repository with a code tutorial of a lot of the same things discussed in this paper. The latest version of github repository can be located at https://github.com/ai2es/WAF_ML_Tutorial_Part2. If you are interested in the version of the repository that was available at time of publication, please see the zendo archive of version 1 (https://zenodo.org/record/7011372). The original github repo for SEVIR is located at https://github.com/MIT-AI-Accelerator/neurips-2020-sevir.

APPENDIX

Hyperparameter Tuning Specifics

All the models shown in the paper are the result of a fairly extensive hyperparameter search. Each of the following figures contains the different hyperparameters that were varied. Note that only 100 models were trained for each model type (e.g., ANN regression was one model), so it is very possible that not all possible hyperparameter solution sets were run. Figures A1–A3 are for the ANN, CNN, and U-Net, respectively, and red indicates the best configuration choice for regression, blue indicates the best configuration choice for classification, and purple means the best configuration choice for both model types. The best configurations were determined by the best performance on the validation dataset.

Artificial Neural Network Hyperparameters	
Hyperparameter	Value
Number of layers	[1, **2**, **3**, 4]
Number of neurons	[4,**8**,16,**32**]
Activation function	[**relu**,sigmoid,tanh]
dropout range	[0.05,0.5]; (**0.182**,**0.120**)
optimizer	[**adam**,adagrad,**sgd**,rmsprop]
batchnorm	[**on**,**off**]
batch size	[32,64,**128**,256,**512**]
loss (reg. only)	[MSE,**MAE**]

FIG. A1. The hyperparameters for the artificial neural networks.

Convolutional Neural Network Hyperparameters	
Hyperparameter	Value
Number of conv. layers	[1, **2**, **3**]
Conv. kenel size	[**3x3**,**5x5**,7x7]
Kernel activation function	[**relu**,sigmoid,tanh]
Number of kernels	[**4**,**8**,16,32]
Number of dense layers	[**1**, **2**, 3,4]
Number of neurons	[**4**,8,16,**32**]
Activation function	[relu,**sigmoid**,tanh]
dropout range	[0.05,0.5]; (**0.3886**,**0.250**)
optimizer	[**adam**,adagrad,**sgd**,rmsprop]
batchnorm	[**on**,off]
batch size	[32,**64**,**128**,256,512]
loss (reg. only)	[MSE,**MAE**]

FIG. A2. The hyperparameters for the convolutional neural networks.

U-Network Hyperparameters	
Hyperparameter	Value
Number of conv. layers	[**1**,2, 3]
Conv. kenel size	[**3x3**,5x5, **7x7**]
Kernel activation function	[relu,**leaky-relu**,**elu**,prelu]
Number of kernels	[4,8,**16**,**32**]
Depth of U-Net	[1,2, **3**, **4**]
optimizer	[**adam**,adagrad,sgd,rmsprop]
batchnorm	[**on**,**off**]
batch size	[32, **64**,128,256,512]
loss (reg. only)	[MSE,**MAE**]

FIG. A3. The hyperparameters for the U-Network.

REFERENCES

Abadi, M., and Coauthors, 2015: TensorFlow: Large-scale machine learning on heterogeneous systems. Accessed 17 May 2023, https://www.tensorflow.org/.

Anaconda, 2020: Anaconda software distribution. Anaconda Inc., accessed 17 May 2023, https://docs.anaconda.com/.

Anderson-Frey, A. K., Y. P. Richardson, A. R. Dean, R. L. Thompson, and B. T. Smith, 2017: Self-organizing maps for the investigation of tornadic near-storm environments. *Wea. Forecasting*, **32**, 1467–1475, https://doi.org/10.1175/WAF-D-17-0034.1.

Bi, K., L. Xie, H. Zhang, X. Chen, X. Gu, and Q. Tian, 2022: Pangu-weather: A 3D high-resolution model for fast and accurate global weather forecast. arXiv, 2211.02556v1, https://doi.org/10.48550/arxiv.2211.02556.

Bisong, E., 2019: Google Colaboratory. *Building Machine Learning and Deep Learning Models on Google Cloud Platform*, Apress, 59–64, https://doi.org/10.1007/978-1-4842-4470-8_7.

Breiman, L., 2001: Random forests. *Mach. Learn.*, **45**, 5–32, https://doi.org/10.1023/A:1010933404324.

Chase, R. J., S. W. Nesbitt, and G. M. McFarquhar, 2021: A dual-frequency radar retrieval of two parameters of the snowfall particle size distribution using a neural network. *J. Appl. Meteor. Climatol.*, **60**, 341–359, https://doi.org/10.1175/JAMC-D-20-0177.1.

——, D. R. Harrison, A. Burke, G. M. Lackmann, and A. McGovern, 2022: A machine learning tutorial for operational meteorology. Part I: Traditional machine learning. *Wea. Forecasting*, **37**, 1509–1529, https://doi.org/10.1175/WAF-D-22-0070.1.

Chen, B.-F., B. Chen, H.-T. Lin, and R. L. Elsberry, 2019: Estimating tropical cyclone intensity by satellite imagery utilizing convolutional neural networks. *Wea. Forecasting*, **34**, 447–465, https://doi.org/10.1175/WAF-D-18-0136.1.

Chen, K., and Coauthors, 2023: FengWu: Pushing the skillful global medium-range weather forecast beyond 10 days lead. arXiv, 2304.02948v1, https://doi.org/10.48550/arxiv.2304.02948.

Cintineo, J. L., M. J. Pavolonis, J. M. Sieglaff, A. Wimmers, J. Brunner, and W. Bellon, 2020: A deep-learning model for automated detection of intense midlatitude convection using geostationary satellite images. *Wea. Forecasting*, **35**, 2567–2588, https://doi.org/10.1175/WAF-D-20-0028.1.

——, ——, and ——, 2022: ProbSevere LightningCast: A deep-learning model for satellite-based lightning nowcasting. *Wea. Forecasting*, **37**, 1239–1257, https://doi.org/10.1175/WAF-D-22-0019.1.

Cloud, K. A., B. J. Reich, C. M. Rozoff, S. Alessandrini, W. E. Lewis, and L. D. Monache, 2019: A feed forward neural network based on model output statistics for short-term hurricane intensity prediction. *Wea. Forecasting*, **34**, 985–997, https://doi.org/10.1175/WAF-D-18-0173.1.

Ebert-Uphoff, I., and K. Hilburn, 2020: Evaluation, tuning, and interpretation of neural networks for working with images in meteorological applications. *Bull. Amer. Meteor. Soc.*, **101**, E2149–E2170, https://doi.org/10.1175/BAMS-D-20-0097.1.

——, R. Lagerquist, K. Hilburn, Y. Lee, K. Haynes, J. Stock, C. Kumler, and J. Q. Stewart, 2021: CIRA guide to custom loss functions for neural networks in environmental sciences—Version 1. arXiv, 2106.09757v1, https://doi.org/10.48550/arxiv.2106.09757.

Espeholt, L., and Coauthors, 2022: Deep learning for twelve hour precipitation forecasts. *Nat. Commun.*, **13**, 5145, https://doi.org/10.1038/s41467-022-32483-x.

Flora, M., C. Potvin, S. Handler, and A. McGovern, 2022a: Comparing explanation methods for traditional machine learning models. Part I: An overview of current methods and quantifying their disagreement. arXiv, 2211.08943, https://arxiv.org/abs/2211.08943.

——, ——, ——, and ——, 2022b: Comparing explanation methods for traditional machine learning models. Part II: Quantifying model explainability faithfulness and improvements with dimensionality reduction. arXiv, 2211.10378, https://arxiv.org/abs/2211.10378.

Fujita, T., 1958: Mesoanalysis of the Illinois tornadoes of 9 April 1953. *J. Meteor.*, **15**, 288–296, https://doi.org/10.1175/1520-0469(1958)015<0288:MOTITO>2.0.CO;2.

Gagne, D. J., S. E. Haupt, D. W. Nychka, and G. Thompson, 2019: Interpretable deep learning for spatial analysis of severe hailstorms. *Mon. Wea. Rev.*, **147**, 2827–2845, https://doi.org/10.1175/MWR-D-18-0316.1.

Gensini, V. A., C. Converse, W. S. Ashley, and M. Taszarek, 2021: Machine learning classification of significant tornadoes and hail in the United States using ERA5 proximity soundings. *Wea. Forecasting*, **36**, 2143–2160, https://doi.org/10.1175/WAF-D-21-0056.1.

Griffin, S. M., A. Wimmers, and C. S. Velden, 2022: Predicting rapid intensification in North Atlantic and eastern North Pacific tropical cyclones using a convolutional neural network. *Wea. Forecasting*, **37**, 1333–1355, https://doi.org/10.1175/WAF-D-21-0194.1.

Harris, C. R., and Coauthors, 2020: Array programming with NumPy. *Nature*, **585**, 357–362, https://doi.org/10.1038/s41586-020-2649-2.

Harrison, D., A. McGovern, C. Karstens, I. L. Jirak, and P. T. Marsh, 2022: Winter precipitation-type classification with a 1D convolutional neural network. *31st Conf. on Weather Analysis and Forecasting (WAF)/27th Conf. on Numerical Weather Prediction (NWP)*, Houston, TX, Amer. Meteor. Soc., J11.4, https://ams.confex.com/ams/102ANNUAL/meetingapp.cgi/Paper/394420.

Henley, C., 2021: *Foundations of Neuroscience*. Michigan State University Libraries, https://openbooks.lib.msu.edu/neuroscience/.

Hilburn, K. A., I. Ebert-Uphoff, and S. D. Miller, 2021: Development and interpretation of a neural-network-based synthetic radar reflectivity estimator using GOES-R satellite observations. *J. Appl. Meteor. Climatol.*, **60**, 3–21, https://doi.org/10.1175/JAMC-D-20-0084.1.

Hinton, G., N. Srivastava, and K. Swersky, 2012: Neural networks for machine learning—Lecture 6a: Overview of mini-batch gradient descent. University of Toronto, 31 pp., https://www.cs.toronto.edu/~tijmen/csc321/slides/lecture_slides_lec6.pdf.

Ioffe, S., and C. Szegedy, 2015: Batch normalization: Accelerating deep network training by reducing internal covariate shift. *Proc. 32nd Int. Conf. on Machine Learning*, Vol. 37, Lille, France, PMLR, 448–456, https://proceedings.mlr.press/v37/ioffe15.html.

Justin, A. D., C. Willingham, A. McGovern, and J. T. Allen, 2022: Toward operational real-time identification of frontal boundaries using machine learning: A 3D model. *21st Conf. on Artificial Intelligence for Environmental Science*, Houston, TX, Amer. Meteor. Soc., 3.3, https://ams.confex.com/ams/102ANNUAL/meetingapp.cgi/Paper/395669.

Kamangir, H., W. Collins, P. Tissot, S. A. King, H. T. H. Dinh, N. Durham, and J. Rizzo, 2021: FogNet: A multiscale 3D CNN with double-branch dense block and attention mechanism for fog prediction. *Mach. Learn. Appl.*, **5**, 100038, https://doi.org/10.1016/j.mlwa.2021.100038.

Katona, B., and P. Markowski, 2021: Assessing the influence of complex terrain on severe convective environments in northeastern Alabama. *Wea. Forecasting*, **36**, 1003–1029, https://doi.org/10.1175/WAF-D-20-0136.1.

Keisler, R., 2022: Forecasting global weather with graph neural networks. arXiv, 2202.07575v1, https://doi.org/10.48550/arxiv.2202.07575.

Kim, Y. H., S. Kim, H.-Y. Han, B.-H. Heo, and C.-H. You, 2013: Real-time detection and filtering of chaff clutter from single-polarization Doppler radar data. *J. Atmos. Oceanic Technol.*, **30**, 873–895, https://doi.org/10.1175/JTECH-D-12-00158.1.

Kingma, D. P., and J. Ba, 2017: Adam: A method for stochastic optimization. arXiv, 1412.6980v9, https://doi.org/10.48550/arxiv.1412.6980.

Kluyver, T., and Coauthors, 2016: Jupyter notebooks—A publishing format for reproducible computational workflows. *Positioning and Power in Academic Publishing: Players, Agents and Agendas*, F. Loizides and B. Schmidt, Eds., IOS Press, 87–90.

Kohonen, T., S. Kaski, and H. Lappalainen, 1997: Self-organized formation of various invariant-feature filters in the adaptive-subspace SOM. *Neural Comput.*, **9**, 1321–1344, https://doi.org/10.1162/neco.1997.9.6.1321

Kuligowski, R. J., and A. P. Barros, 1998: Experiments in short-term precipitation forecasting using artificial neural networks. *Mon. Wea. Rev.*, **126**, 470–482, https://doi.org/10.1175/1520-0493(1998)126<0470:EISTPF>2.0.CO;2.

Kumler-Bonfanti, C., J. Stewart, D. Hall, and M. Govett, 2020: Tropical and extratropical cyclone detection using deep learning. *J. Appl. Meteor. Climatol.*, **59**, 1971–1985, https://doi.org/10.1175/JAMC-D-20-0117.1.

Lagerquist, R., and I. Ebert-Uphoff, 2022: Can we integrate spatial verification methods into neural-network loss functions for atmospheric science? arXiv, 2203.11141v2, https://doi.org/10.48550/arxiv.2203.11141.

——, A. McGovern, and D. Gagne, 2019: Deep learning for spatially explicit prediction of synoptic-scale fronts. *Wea. Forecasting*, **34**, 1137–1160, https://doi.org/10.1175/WAF-D-18-0183.1.

——, J. T. Allen, and A. McGovern, 2020a: Climatology and variability of warm and cold fronts over North America from 1979 to 2018. *J. Climate*, **33**, 6531–6554, https://doi.org/10.1175/JCLI-D-19-0680.1.

——, A. McGovern, C. R. Homeyer, D. J. Gagne II, and T. Smith, 2020b: Deep learning on three-dimensional multiscale data for next-hour tornado prediction. *Mon. Wea. Rev.*, **148**, 2837–2861, https://doi.org/10.1175/MWR-D-19-0372.1.

——, J. Q. Stewart, I. Ebert-Uphoff, and C. Kumler, 2021: Using deep learning to nowcast the spatial coverage of convection from *Himawari-8* satellite data. *Mon. Wea. Rev.*, **149**, 3897–3921, https://doi.org/10.1175/MWR-D-21-0096.1.

Lakshmanan, V., C. Karstens, J. Krause, K. Elmore, A. Ryzhkov, and S. Berkseth, 2015: Which polarimetric variables are important for weather/no-weather discrimination? *J. Atmos. Oceanic Technol.*, **32**, 1209–1223, https://doi.org/10.1175/JTECH-D-13-00205.1.

Lam, R., and Coauthors, 2022: GraphCast: Learning skillful medium-range global weather forecasting. arXiv, 2212.12794v1, https://doi.org/10.48550/arxiv.2212.12794.

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**, 1096, https://doi.org/10.1038/s41467-019-08987-4.

LeCun, Y., B. Boser, J. Denker, D. Henderson, R. Howard, W. Hubbard, and L. Jackel, 1989: Handwritten digit recognition with a back-propagation network. *Advances in Neural Information Processing Systems*, D. Touretzky, Ed., Vol. 2, Morgan-Kaufmann, 396–404, https://proceedings.neurips.cc/paper/1989/file/53c3bce66e43be4f209556518c2fcb54-Paper.pdf.

Lee, Y., C. D. Kummerow, and I. Ebert-Uphoff, 2021: Applying machine learning methods to detect convection using Geostationary Operational Environmental Satellite-16 (GOES-16) Advanced Baseline Imager (ABI) data. *Atmos. Meas. Tech.*, **14**, 2699–2716, https://doi.org/10.5194/amt-14-2699-2021.

Lundberg, S. M., and S.-I. Lee, 2017: A unified approach to interpreting model predictions. *Advances in Neural Information Processing Systems: Proceedings of the First 12 Conferences*, I. Guyon et al., Eds., Vol. 30, Curran Associates, Inc., https://proceedings.neurips.cc/paper_files/paper/2017/file/8a20a8621978632d76c43dfd28b67767-Paper.pdf.

Mamalakis, A., E. A. Barnes, and I. Ebert-Uphoff, 2022a: Investigating the fidelity of explainable artificial intelligence methods for applications of convolutional neural networks in geoscience. arXiv, 2202.03407v2, https://doi.org/10.48550/arxiv.2202.03407.

——, I. Ebert-Uphoff, and E. Barnes, 2022b: Explainable artificial intelligence in meteorology and climate science: Model fine-tuning, calibrating trust and learning new science. *xxAI—Beyond Explainable AI*, A. Holzinger et al., Eds., Springer International Publishing, 315–339, https://doi.org/10.1007/978-3-031-04083-2_16.

Marzban, C., and G. J. Stumpf, 1996: A neural network for tornado prediction based on Doppler radar-derived attributes. *J. Appl. Meteor.*, **35**, 617–626, https://doi.org/10.1175/1520-0450(1996)035<0617:ANNFTP>2.0.CO;2.

——, and ——, 1998: A neural network for damaging wind prediction. *Wea. Forecasting*, **13**, 151–163, https://doi.org/10.1175/1520-0434(1998)013<0151:ANNFDW>2.0.CO;2.

McCandless, T. C., G. S. Young, S. E. Haupt, and L. M. Hinkelman, 2016: Regime-dependent short-range solar irradiance forecasting. *J. Appl. Meteor. Climatol.*, **55**, 1599–1613, https://doi.org/10.1175/JAMC-D-15-0354.1.

McCann, D. W., 1992: A neural network short-term forecast of significant thunderstorms. *Wea. Forecasting*, **7**, 525–534, https://doi.org/10.1175/1520-0434(1992)007<0525:ANNSTF>2.0.CO;2.

McCulloch, W. S., and W. Pitts, 1943: A logical calculus of the ideas immanent in nervous activity. *Bull. Math. Biophys.*, **5**, 115–133, https://doi.org/10.1007/BF02478259.

McKinney, W., 2010: Data structures for statistical computing in Python. *Proc. Ninth Python in Science Conf. (SCIPY 2010)*, Austin, TX, SciPy, 56–61, https://doi.org/10.25080/Majora-92bf1922-00a.

Molina, M. J., D. J. Gagne, and A. F. Prein, 2021: A benchmark to test generalization capabilities of deep learning methods to classify severe convective storms in a changing climate. *Earth Space Sci.*, **8**, e2020EA001490, https://doi.org/10.1029/2020EA001490.

Mounier, A., L. Raynaud, L. Rottner, M. Plu, P. Arbogast, M. Kreitz, L. Mignan, and B. Touzé, 2022: Detection of bow echoes in kilometer-scale forecasts using a convolutional neural network. *Artif. Intell. Earth Syst.*, **1**, e210010, https://doi.org/10.1175/AIES-D-21-0010.1.

Murphy, A. H., 1993: What is a good forecast? An essay on the nature of goodness in weather forecasting. *Wea. Forecasting*, **8**, 281–293, https://doi.org/10.1175/1520-0434(1993)008<0281:WIAGFA>2.0.CO;2.

Nguyen, T., J. Brandstetter, A. Kapoor, J. K. Gupta, and A. Grover, 2023: ClimaX: A foundation model for weather and climate. arXiv, 2301.10343v2, https://doi.org/10.48550/arxiv.2301.10343.

Nowotarski, C. J., and A. A. Jensen, 2013: Classifying proximity soundings with self-organizing maps toward improving supercell and tornado forecasting. *Wea. Forecasting*, **28**, 783–801, https://doi.org/10.1175/WAF-D-12-00125.1.

Paszke, A., and Coauthors, 2019: PyTorch: An imperative style, high-performance deep learning library. *Advances in Neural Information Processing Systems 32*, H. Wallach et al., Eds., Curran Associates, Inc., 8024–8035.

Pedregosa, F., and Coauthors, 2011: Scikit-learn: Machine learning in Python. *J. Mach. Learn. Res.*, **12**, 2825–2830.

Rasp, S., and N. Thuerey, 2021: Data-driven medium-range weather prediction with a Resnet pretrained on climate simulations: A new model for WeatherBench. *J. Adv. Model. Earth Syst.*, **13**, e2020MS002405, https://doi.org/10.1029/2020MS002405.

Ravuri, S., and Coauthors, 2021: Skilful precipitation nowcasting using deep generative models of radar. *Nature*, **597**, 672–677, https://doi.org/10.1038/s41586-021-03854-z.

Roberts, N. M., and H. W. Lean, 2008: Scale-selective verification of rainfall accumulations from high-resolution forecasts of convective events. *Mon. Wea. Rev.*, **136**, 78–97, https://doi.org/10.1175/2007MWR2123.1.

Roebber, P., 2009: Visualizing multiple measures of forecast quality. *Wea. Forecasting*, **24**, 601–608, https://doi.org/10.1175/2008WAF2222159.1.

Ronneberger, O., P. Fischer, and T. Brox, 2015: U-net: Convolutional networks for biomedical image segmentation. arXiv, 1505.04597v1, https://doi.org/10.48550/arxiv.1505.04597.

Rumelhart, D. E., G. E. Hinton, and R. J. Williams, 1986: Learning representations by back-propagating errors. *Nature*, **323**, 533–536, https://doi.org/10.1038/323533a0.

Sha, Y., D. J. Gagne II, G. West, and R. Stull, 2020a: Deep-learning-based gridded downscaling of surface meteorological variables in complex terrain. Part I: Daily maximum and minimum 2-m temperature. *J. Appl. Meteor. Climatol.*, **59**, 2057–2073, https://doi.org/10.1175/JAMC-D-20-0057.1.

——, ——, ——, and ——, 2020b: Deep-learning-based gridded downscaling of surface meteorological variables in complex terrain. Part II: Daily precipitation. *J. Appl. Meteor. Climatol.*, **59**, 2075–2092, https://doi.org/10.1175/JAMC-D-20-0058.1.

Shao, J., 1998: Improving nowcasts of road surface temperature by a backpropagation neural network. *Wea. Forecasting*, **13**, 164–171, https://doi.org/10.1175/1520-0434(1998)013<0164:INORST>2.0.CO;2.

Shapley, L. S., 1953: A value for n-person games. *Contributions to the Theory of Games II*, H. W. Kuhn and A. W. Tucker, Eds., Princeton University Press, 307–317.

Shwartz-Ziv, R., and A. Armon, 2022: Tabular data: Deep learning is not all you need. *Info. Fusion*, **81**, 84–90, https://doi.org/10.1016/j.inffus.2021.11.011.

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**, 1929–1958.

Stevens, B., and Coauthors, 2019: DYAMOND: The DYnamics of the atmospheric general circulation modeled on non-hydrostatic domains. *Prog. Earth Planet. Sci.*, **6**, 61, https://doi.org/10.1186/s40645-019-0304-z.

Stock, J., 2021: Using machine learning to improve vertical profiles of temperature and moisture for severe weather nowcasting. M.S. thesis, Dept. of Computer Science, Colorado State University, 95 pp.

van Straaten, C., K. Whan, D. Coumou, B. van den Hurk, and M. Schmeits, 2022: Using explainable machine learning forecasts to discover subseasonal drivers of high summer temperatures in western and central Europe. *Mon. Wea. Rev.*, **150**, 1115–1134, https://doi.org/10.1175/MWR-D-21-0201.1.

Veillette, M., S. Samsi, and C. Mattioli, 2020: SEVIR: A storm event imagery dataset for deep learning applications in radar and satellite meteorology. *Advances in Neural Information Processing Systems*, H. Larochelle et al., Eds., Vol. 33, Curran Associates, Inc., 22 009–22 019.

Wang, J., X. Dong, A. Kennedy, B. Hagenhoff, and B. Xi, 2019: A regime-based evaluation of southern and northern Great Plains warm-season precipitation events in WRF. *Wea. Forecasting*, **34**, 805–831, https://doi.org/10.1175/WAF-D-19-0025.1.

Weyn, J. A., D. R. Durran, and R. Caruana, 2020: Improving data-driven global weather prediction using deep convolutional neural networks on a cubed sphere. *J. Adv. Model. Earth Syst.*, **12**, e2020MS002109, https://doi.org/10.1029/2020MS002109.

Xu, W., K. Balaguru, A. August, N. Lalo, N. Hodas, M. DeMaria, and D. Judi, 2021: Deep learning experiments for tropical cyclone intensity forecasts. *Wea. Forecasting*, **36**, 1453–1470, https://doi.org/10.1175/WAF-D-20-0104.1.

Zhou, K., Y. Zheng, W. Dong, and T. Wang, 2020: A deep learning network for cloud-to-ground lightning nowcasting with multisource data. *J. Atmos. Oceanic Technol.*, **37**, 927–942, https://doi.org/10.1175/JTECH-D-19-0146.1.