Learning Physics through Images: An Application to Wind-Driven Spatial Patterns

M. Giselle Fernández-Godino\textsuperscript{1,*}, Donald D. Lucas\textsuperscript{1,+}, and Qingkai Kong\textsuperscript{1}

\textsuperscript{1}Lawrence Livermore National Laboratory, 7000 East Ave, Livermore, CA 94550, United States
\textsuperscript{*}fernandez48@llnl.gov
\textsuperscript{+}lucas26@llnl.gov

ABSTRACT

For centuries, scientists have observed nature to understand the laws that govern the physical world. The traditional process of turning observations into physical understanding is slow. Imperfect models are constructed and tested to explain relationships in data. Powerful new algorithms are available that can enable computers to learn physics by observing images and videos. Inspired by this idea, instead of training machine learning models using physical quantities, we trained them using images, that is, pixel information. For this work, and as a proof of concept, the physics of interest are wind-driven spatial patterns. Examples of these phenomena include features in Aeolian dunes and the deposition of volcanic ash, wildfire smoke, and air pollution plumes. We assume that the spatial patterns were collected by an imaging device that records the magnitude of the logarithm of deposition as a red, green, blue (RGB) color image with channels containing values ranging from 0 to 255. In this paper, we explore deep convolutional neural network-based autoencoders to exploit relationships in wind-driven spatial patterns, which commonly occur in geosciences, and reduce their dimensionality. Reducing the data dimension size with an encoder allows us to train regression models linking geographic and meteorological scalar input quantities to the encoded space. Once this is achieved, full predictive spatial patterns are reconstructed using the decoder. We demonstrate this approach on images of spatial deposition from a pollution source, where the encoder compresses the dimensionality to 0.02% of the original size and the full predictive model performance on test data achieves an accuracy of 92%.

Introduction

Spatial patterns influenced by wind-driven dynamics are common in geosciences. Examples include algal blooms in the ocean surface\textsuperscript{1}, Aeolian sand dunes\textsuperscript{2}, and atmospheric dispersion plumes of air pollution\textsuperscript{3}, volcanic ash\textsuperscript{4}, and wildfire smoke\textsuperscript{5}, among others. Images and maps of spatial patterns collected from satellites and other remote sensing platforms encode the physical relationships between the prevailing wind conditions and resulting patterns. Note that the term spatial pattern is used to refer to an associated plume at a particular time, or an integral over time, after the source release, that is, there is no time variation involved. These images (snapshots) can be used to build data-driven models that predict new spatial patterns given wind inputs. A predictive model fully trained on images implicitly contains the relevant physical processes without relying on a mathematical model that has parameterizations, approximations, and other assumptions. Another advantage is the speedup in predictions compared with physics-based models, which discretize space into many cells and solve expensive mathematical equations. Figure 1 shows aerial pictures of some examples of these atmospheric plumes. Satellite images are a great way to obtain image data of these events, however, using these images to build machine learning models has not been exploited yet and this work is the starting point towards this direction.

For the past six and a half decades\textsuperscript{6}, the meteorological community has relied upon principal component analysis (PCA) for analyzing and decomposing features in spatial patterns. If most of the variability in a set of spatial patterns can be explained by a relatively small number of principal components, the dimensionality of the spatial problem can be greatly reduced. Effective dimensional reduction facilitates additional avenues for research and analysis of spatial patterns. For example, Higdon et al., 2008\textsuperscript{7} showed how computationally expensive Bayesian calibration and inference methods can be applied to images or data from complex physics models after first reducing the dimensionality using principal components. In Francom et al., 2019\textsuperscript{8}, principal components are applied to spatiotemporal plume data before training a statistical model to emulate wind-driven atmospheric transport. Despite their success, a known issue associated with these orthogonal approximations is their inability to model non-linear problems. Recent advances in computational power and data availability have facilitated the path for the scientific community towards big data science and deep learning models. An advantage of these models is the speedup and equivalent accuracy in predictions that can be achieved if compared with high fidelity simulations or experiments, doing so with comparable accuracy. In recent years, deep learning techniques such as convolutional neural networks (CNNs), have been
Recent examples demonstrate how autoencoders are actively used as feature extractors in plasma physics\textsuperscript{17} and seismology\textsuperscript{18}. The novelty of our work is the use of images and pixel information, instead of physical quantities, for predicting spatial patterns. In this paper, we show that an autoencoder consisting of multiple CNN layers with a relatively small number of hyperparameters ($\approx 50,000$) can be leveraged for predicting the spatial patterns associated with plumes blowing in different directions and originating from different locations. Moreover, unlike the Modified National Institute of Standards and Technology (MNIST) dataset of grayscale images (single channel) of handwritten digits\textsuperscript{19} commonly used for benchmarking autoencoders, our color images (three channels) include spatial patterns that are short and wide or long and narrow, based on the wind conditions, and translated so that a wide range of fractions of the spatial pattern remains inside the figure, adding an extra challenge. The difference of our dataset, however, lies in the fact that instead of ten objects (digits from zero to nine) we only have a unique object, a rotating and moving spatial pattern.

\textbf{The physical problem}

The physical problem of interest in this paper is a two-dimensional, spatial pattern formed an hour after a pollutant has been released into the atmosphere and deposited on the surface. The release location ($x, y$) of the pollutant is assumed to occur near the surface and anywhere in a two-dimensional domain of $5000m \times 5000m$. The pollutant is blown in a direction controlled by the large-scale atmospheric inflow winds expressed as a wind speed ($w_s$), which varies from $0.5$ to $15m/s$, and wind direction ($w_d$), which can be anywhere in the interval $[0, 360)\,^\circ$ degrees following standard mathematical convention. For training purposes, however, we use $w_u = w_s \cos w_d$ and $w_v = w_s \sin w_d$. We assume that the spatial patterns were collected by an imaging device that records the magnitude of the logarithm of deposition as a red, green, blue (RGB) color image with channels containing integer values ranging from 0 to 255. The goal is to predict a deposition image given its associated release location and wind velocity (four scalar quantities). In other words, we are interested in the following mapping: $[s_x, s_y, w_u, w_v] \rightarrow [\text{height} \times \text{width} \times \text{RGB channel}]$.

\textbf{The data}

Given large-scale winds as an inflow boundary condition, the computational fluid dynamics code Aeolus\textsuperscript{20} uses millions of grid cells to simulate fluid flow and material transport in complex, three-dimensional environments at high resolution, accounting for turbulence from structures and obstacles, and predicting deposition on the ground and other surfaces. For demonstration purposes, megapixel deposition images were obtained by processing the output of three-dimensional Aeolus simulations, which were run using a resolution of $(x,y,z) = 1000 \times 1000 \times 104$ cells, each cell representing $5m \times 5m \times 5m$. Even though Aeolus
simulates deposition on surfaces in three dimensions, we only consider deposition at ground level. To obtain a two-dimensional representation of the data, we gathered the deposition from ground level. Due to elevation changes, any given ground level cell may be at different heights, usually varying from 10m to 15m, that is between the second and the fourth cell in the z direction. The entire dataset, which was created by running Aeolus multiple times, contains 12,000 deposition images varying the four scalar quantities associated with each image (release location and inflow winds). The data images are stored as \([\text{number of images, height, width, RGB channels}] = [12000, 1000, 1000, 3]\). Each megapixel image shows the spatial deposition pattern of a unique release scenario in Aeolus. As previously noted, RGB pixel colors are associated with the logarithm of the deposition values. The rainbow colormap is used to create the RGB images for training and testing the autoencoder. Figure 2 shows the mapping between the RGB values, colormap, and deposition values. Note that the actual deposition values are not considered in this paper.

![Figure 2. Colormap used to create the images. The figure shows the corresponding values of red, green and blue at each point throughout the colorbar and the corresponding magnitude of deposition.](image)

The deep-learning model
The architecture of the deep learning model consists of three parts: (i) the autoencoder; (ii) the bottleneck model; and (iii) the corrector model. The autoencoder is used to reduce the number of spatial elements in the original megapixel image which allows us to train a fully-connected bottleneck model linking geographic and meteorological scalar input quantities to the encoded space. Although accurate, the bottleneck model predictions are biased near the source (red color), which we correct using a denoiser model. Once this is achieved, the decoder is used to recover an estimate of a megapixel image, completing the loop. Figure 3 shows a schematic of the full deep-learning prediction model.

Without an autoencoder, training a fully connected model with a single hidden layer and three million neurons (rule of thumb used is that the neurons needed are \(2/3 \times \text{input size} + \text{output size}\)) to connect four scalars to a megapixel image would involve roughly a trillion hyperparameters and a petabyte of storage for training on batches of 128 images (note that even if we use a single image per batch, we still need tents of terabytes), which is intractable for today’s computer power. Furthermore, even in the hypothetical case that we are able to build this model, the prediction times associated would be impractical.

Results
In this section, we show qualitative results and assess the model performance quantitatively using two, conceptually different, metrics. Figure 4 shows, in each row, predictions for a randomly selected test case never seen by the model. The figure shows a total of six cases. In the first column, the four scalar inputs are depicted by the green cross symbolizing the location of the source release and a wind vector arrow that shows the speed (arrow length) and direction (arrow direction) of the winds. The second column shows the latent space associated with each case predicted by the bottleneck model. The third column shows the corrected latent space. The fourth column shows the overall model predictions obtained from the decoder. Finally, the fifth column shows the ground truth megapixel images, that is, the desired model output.

Figure 5 shows the pixel-level Pearson correlation between truth and predicted images for 300 test cases. Figure 5(a) shows the correlations between the autoencoder predictions and the corresponding true images, while Figure 5(b) shows the correlation between the overall model predictions and the corresponding true images. The figures show that the performance is good throughout the spatial domain, with almost all the pixels having correlations higher than 0.9, except near the lower-left and upper-right corners where they are smaller. The mean correlation value is larger than 0.8 in both figures. Note that the
Figure 3. Schematic of the overall spatial prediction model, which combines the bottleneck model, corrector, and decoder.

Figure 4. Images corresponding to different stages of the spatial prediction process for six randomly selected test cases shown in each row. The first column shows the scalar inputs (source location, wind speed and wind direction), the second column shows the predicted latent space using the bottleneck model, the third column shows the corrected latent space, the fourth column shows the final prediction using the decoder, and finally, the fifth column shows the ground truth. The colors of the pixels in the two rightmost columns denote the logarithm of deposition.
correlations in Figure 5(b) are lower than those in Figure 5(a) because the latter only includes autoencoder errors while the former includes the errors associated with the bottleneck model, the corrector, and the decoder.

(a) Correlations between the true images and the autoencoder predictions for each pixel. The mean correlation value is 0.86. The minimum correlation value in a pixel is 0.18 and the maximum 0.98.

(b) Correlations between the true images and the overall model predictions for each pixel. The mean correlation value is 0.82. The minimum correlation value in a pixel is 0.20 and the maximum 0.95.

Figure 5. Pixel-level correlation values between the ground truth and predicted images for 300 test cases. The correlation shown per pixel is the average of the three channels.

Figure 6 shows histograms of two performance metrics applied to 300 test pairs of predicted and ground truth images: (i) normalized root mean squared error (NRMSE) is a metric commonly used for image prediction applications, and (ii) figure of merit in space (FMS), which considering only non-white pixels, is a metric used in to assess the amount of overlap between a pair of spatial patterns.

Eq. (1) shows how the NRMSE is calculated for a pair of predicted and true images (case $i$) where $\| \cdot \|_2$ denotes the $l_2$-norm. A perfect prediction would have an NRMSE of 0%. Figure 6(a) shows the NRMSE histogram considering the 300 cases set aside for testing purposes. The figure includes the mean NRMSE (orange continuous line) and median NRMSE (green dashed line), which in this case overlap and are both around 8%. We also added the mean NRMSE performance of a baseline model (red dashed line). The baseline model, which is not a trained model, is built so it takes the source location and wind velocity as inputs and returns random patterns. In this case, the baseline NRMSE error is 20% and is a reference point for evaluating the performance of the deep learning model. Note that the NRMSE takes into account white pixels, that is, pixels where no deposition exists. The images in our dataset are approximately 85% white pixels, hence even if the prediction and the true pattern do not overlap, they still will have around 70% of matching pixels. This explains why the mean NRMSE for the baseline model is only 20%.

\[
NRMSE_i = 100 \times \frac{\| \text{pixels(true image)}_i - \text{pixels(predicted image)}_i \|_2}{\| \text{pixels(true image)}_i \|_2}
\]

Eq. (2) shows how the FMS\textsuperscript{22} is calculated for a pair of images (case $i$) where $\cap$ and $\cup$ are the set notation of intersection and union, respectively. The FMS captures the fractional overlap of two spatial patterns and a perfect prediction results in an FMS of 100% \textsuperscript{1}. Figure 6(b) shows the FMS histogram considering the 300 test cases available. The figure includes the mean FMS (orange continuous line) around 71% and median FMS (green dashed line) around 78%. We also added the mean FMS performance of the random baseline model (red dashed line). The mean FMS value for the baseline model is 8%. The cases that skew the results and give a long tail of poor FMS values are the ones where the pattern lies near the border (i.e. it occupies only a few pixels). The performance is substantially improved if only cases where the release location is at least 100m from the border (the overall domain size is 5,000m $\times$ 5,000m) and where the wind direction points inside of the domain are considered.

\[
FMS_i = 100 \times \frac{\text{true pattern}_i \cap \text{predicted pattern}_i}{\text{true pattern}_i \cup \text{predicted pattern}_i}
\]

\textsuperscript{1}The pattern referred to in Eq. (2) is created as follows: (1) convert color images into grayscale by taking the average of each of the image three pixels values, (2) set the values larger than 240 (white pixels) to "NaN" and (3) convert values smaller or equal to 240 (gray and black pixels) to zero (black). Then, in Eq. (2) these converted images (spatial patterns) are used to compute the intersection divided by the union of the zero values (black pixels). Note that NaNs are ignored during this computation.
A full comparison of autoencoders with traditional orthogonal function decomposition is out of scope for this paper. However, we computed performance metrics between a PCA-based model and the autoencoder-based model presented in this work. We arrived at the following conclusions: For our dataset (i) PCA requires five orders of magnitude more memory to store the model than the decoder (15 Gb versus 171 Kb). (ii) The PCA latent space (principal components) is harder to predict using a simple bottleneck model, which is possibly due to the linear assumptions of PCA. (iii) Tensorflow-Keras makes it easier to train a model in batches so very little memory in parallel is needed for training. Conversely, to train PCA we needed to increase the available memory by a factor of six. (iv) The FMS of the PCA-based reconstruction without a bottleneck regression is an outstanding 99%. By comparison, the FMS for only the autoencoder is 93%. However the NRMSE has a mean of 13% (versus 7% for the autoencoder). These results indicate that only the overlap of the spatial patterns, but not the pixel colors, is well predicted compared with the autoencoder-based model.

Discussion

In this paper, we leveraged the spatial reduction that convolutional neural network-based autoencoders can achieve. We created an autoencoder (≈ 50,000 hyperparameters) that reduces the dimension of a color image representing a two-dimensional deposition spatial pattern by 99.98%. This enabled the use of a fully-connected bottleneck model linking the source location and wind conditions to the reduced space achieving high accuracy. Then the decoder part of the autoencoder was used to recover the original resolution completing the loop. In other words, the color images, originally 1000 × 1000 pixels with red, green and blue channels, were reduced to 0.02% of their size (grayscale images of 25 × 25 pixels). This large reduction enables the training of a regression model that connects four scalars (source location and wind velocity boundary conditions) with the reduced space. Images of the original size are recovered using the decoder. The accuracy of the model is 92% on the test dataset (data never seen by the model during training).

Although more complex versions of autoencoders are available (such as variational autoencoders and adversarial autoencoders) the present problem was accurately represented by a traditional autoencoder. To test these other architectures, we compared the performance between traditional autoencoders and the variational and adversarial counterparts for the spatial pattern dataset but using images with a resolution 100 times smaller (color images of 100 × 100 pixels). The comparison showed that, in both cases, the FMS improved while the NRMSE degraded. We found two main drawbacks of using these more complex architectures: (1) The more sophisticated models resulted in more than 100 million hyperparameters (this is 200 times more hyperparameters for a problem with images with a resolution 100 smaller!) and, once built, they needed 2Gb of storage space compared with the 31Kb that the traditional autoencoder needed for images of the same resolution. (2) The latent space was harder to interpret if compared with the “reduced-resolution” grayscale spatial pattern found by traditional autoencoders (see Figures 3 and 4), and hence a more sophisticated bottleneck model is needed to link the scalars with the latent space.

Although this work, as proof of concept, is based on images of patterns obtained from physics-based models, in future work we plan to use real-world images for training. The ultimate goal is the exploitation of satellite and/or real-world images to train predictive models of spatial patterns. Beforehand, pre-build models trained with images from physics models, like the one built in this paper, and/or trained on images obtained experimentally, such as smoke releases under controlled laboratory conditions, can be fine-tuned to learn further from satellite images to build a more realistic target model using transfer learning techniques.
Methods

The autoencoder architecture

We trained a CNN-based autoencoder (via Tensorflow-Keras). This autoencoder can be thought of as an identity operation that has two parts, an encoder, and a decoder. While the encoder takes the original image and reduces it to a latent space dimension, the decoder takes the reduced space vector and recovers the original image. In this case, the original RGB color image of size $1000 \times 1000$ pixels is reduced to a grayscale image of $25 \times 25$ pixels via the encoder. Note that the size of the reduced image is less than 0.02% of the original size. The original images can be recovered using the decoder. The autoencoder architecture was inspired by the tutorial\textsuperscript{25} by Keras creator François Collet. Figure 7 shows schematically the autoencoder architecture. The pixels values of the three color channels were normalized to the interval $[0,1]$ before training. The data was divided into 10,530 training cases (used for training), 1170 validation cases (used to evaluate the loss at the end of each epoch), and 300 test cases (data never seen by the model). The model was trained using single precision data.

![Diagram of autoencoder architecture](image)

**Figure 7.** Schematic of the autoencoder architecture. The encoder receives the original image (shape $[1000,1000,3]$) and it reduces its size to less than 0.02% ($[25,25,1]$). The decoder is able to recover the original image from the reduced space quite accurately.

The encoder is built using six two-dimensional convolutional layers and five two-dimensional max-pooling layers interspersed among the convolutional layers (11 layers total). All layers have padding with zeros evenly. All the convolutional layers have a rectilinear linear unit (ReLU) as the activation function. The first convolutional layer of the encoder has 32 kernels with a height and width of seven. The second has 16 kernels with a height and width of five. The third has eight kernels with a height and width of three. The fourth has four kernels with a height and width of three. The fifth has three kernels with a height and width of three. Finally, the sixth and last convolutional layer of the encoder has one kernel with a height and width of three. The first, second, and third max-pooling layers have a pool size of two, while the fourth has a pool size of five and finally, the fifth layer has a pool size of one. The output of the encoder has dimensions of $[25,25,1]$. Note that this is a factor of 4,800 reduction, that is, we have reduced our original image to a 0.02% of its size. The decoder is built using seven two-dimensional convolutional layers and six two-dimensional upsampling layers interspersed among the convolutional layers (13 layers total). All layers have padding with zeros evenly. All the convolutional layers have a ReLU as the activation function except the last layer whose activation function is the sigmoid function. The first convolutional layer of the decoder has one kernel with a height and width of three. The second has three kernels with a height and width of three. The third has four kernels with a height and width of three. The fourth has eight kernels with a height and width of three. The fifth has 16 kernels with a height and width of five. The sixth has 32 kernels with a height and width of seven. Finally, the seventh and last convolutional layer of the decoder has three kernels with a height and width of seven. The first upsampling layer has an upsampling factor of one. The second upsampling layer has an upsampling factor of five. The third, fourth, and fifth upsampling layers have an upsampling factor of two, and finally, the sixth upsampling layer has a pool size of one. The output of the decoder has dimensions of $[1000,1000,3]$. The sigmoid function used in the decoder’s last layer forces the autoencoder to produce output values between 0 and 1 which is desired because we normalized the three channels values to the interval $[0,1]$. The original pixel values can be recovered by multiplying by 255. The autoencoder model has a total of 52,634 hyperparameters. Most of the details stated above are summarized in Table 1.

For the first-order gradient-based optimization we used the Adam algorithm\textsuperscript{26} and the binary cross-entropy as loss function. In the studied case, being a regression problem, it would have been more natural to use the mean squared error as a loss function.
This loss, indeed, results in a more accurate autoencoder with a more sophisticated latent space. The reason this loss was not used here is that we are looking for the best overall accuracy (bottleneck model + corrector model + autoencoder) and we found that this more complex latent space was not well represented by the bottleneck model leading to overall poor results. The bottleneck model is an independent fully connected neural network (trained via scikit-learn) that maps the scalar input vector ([4, 1]) to the reduced space ([25, 25, 1]) and it is described in the next section.

The usefulness of the autoencoder is that it can be divided into an encoder and a decoder, which can be used separately. In this case, we used the encoder as a compression tool where the input is a [1000, 1000, 3] image and the output is [25, 25, 1]. The bottleneck layer output is a good single feature map that resembles the original color image in a much-reduced dimension. The decoder is then used by the full prediction model to recover the high-resolution images.

The bottleneck architecture
The goal is to link the geographic and meteorological scalar input quantities to the reduced encoded space through the bottleneck model and the second step is to correct the bias using the corrector model. Both are described in the following subsections.

The bottleneck model
The bottleneck dimension, 25 × 25 × 1, is now small enough to train a simple model linking the wind conditions and release location (four scalar quantities) with the reduced space. For this task, we have used the neural network MLPRegressor model from the Python package scikit-learn version 0.24.2. The optimizer used is Adam and the activation is a sigmoid function. We used default options except for the adaptive learning rate, the L2 penalty alpha set to 10 (128 times larger than the input image dimension). This latent space extracts 128 features from the original image and recovers the average bottleneck model score is 95% on the 300 test cases (data never seen before by the model).

The corrector model
Even if the bottleneck model described in the previous subsection had a great performance overall, it did not perform as desired on high deposition areas (red areas) hence the need of improving the latent space prediction with a corrector. The corrector model is a second CNN-based autoencoder model (via Tensorflow-Keras), inspired by denoising or imputing autoencoders. Instead of reducing dimensions, as the autoencoder presented before, this autoencoder is used as a "denoiser" with a larger bottleneck dimension (also known as overcomplete autoencoder). The corrector model takes as input the bottleneck model output (shape [25, 25, 1]) and returns a corrected grayscale latent space of shape [25, 25, 1]). The corrector bottleneck dimension is 25 × 25 × 128 (128 times larger than the input image dimension). This latent space extracts 128 features from the original image and recovers a corrected image. The corrector model was trained using the bottleneck model outputs as the input training data, and the encoder outputs as output training data. The pixel values were normalized to the interval [0,1] before training. The data was divided into 10,530 training cases (used for training), 1170 validation cases (used to evaluate the loss at the end of each epoch), and 300 test cases (data never seen by the model). The corrector encoder is built using two two-dimensional convolutional layers. All layers have padding with zeros evenly. All the convolutional layers have a ReLU as the activation function. The first convolutional layer of the corrector encoder has 64 kernels with a height and width of seven. The second has 128 kernels with a height and width of seven. The output of the corrector encoder has a shape of [25, 25, 128]. The corrector decoder is built using three two-dimensional convolutional layers. All layers have padding with zeros evenly. All the convolutional layers have a ReLU as the activation function. The first convolutional layer of the corrector decoder has 64 kernels with a height and width of seven. The second has 32 kernels with a height and width of seven. The second has 32 kernels with a height and width of seven. Finally, the third has one kernel with a height and width of seven. The output of the corrector decoder has a shape of [25, 25, 1]. Most of the details stated above are summarized in Table 1. The original pixel values can be recovered by multiplying by 255. For the first-order gradient-based optimization we used the Adam algorithm and the mean squared error as loss function. The corrector model has a total of 908,161 hyperparameters. The average bottleneck model R² score is 93% on the 300 test cases (data never seen before by the model). Even if the corrector model R² score is slightly poorer (if compared with the R² = 95% resulting of using only the bottleneck model without the corrector, see the previous subsection), the overall accuracy of the full prediction model was improved resulting in a mean NRMSE decrease of 2% and a mean FMS increase of 5%.

The full prediction model
The full prediction model consists of aligning three different independent models. First the bottleneck model connecting four scalar quantities (x, y, wₜ, wₐ) to the grayscale latent space of shape [25, 25, 1]; second, correcting the grayscale latent space of shape [25, 25, 1] through the corrector model obtaining a corrected grayscale latent space of shape [25, 25, 1]; and third connecting the estimated corrected latent space to the high-resolution prediction of the deposition spatial pattern color image of shape [1000, 1000, 3], through the decoder. Figure 3 shows a schematic representation of the process described.
Model interpretability

Figure 8 shows the mean Pearson correlation between each of the four inputs of the bottleneck model and each of the $25 \times 25$ pixels of the latent space. The figure was generated using the training data (10,530 images). The test data was not used for this calculation because in this case, we are interpreting the trained model and not evaluating its performance. As expected for the variable $x$ the correlation changes substantially along the horizontal axis and there is roughly no change along the vertical axis. Conversely, for $y$, the correlation variation is strong through the vertical axis while almost nonexistent through the horizontal axis. We have the same behavior for $w_u$ (the horizontal component of the wind velocity) and $w_v$ (the vertical component of the wind velocity).

Figure 8. Correlation between the two-dimensional latent space of shape $[25, 25, 1]$ and the inputs of interest: source location $(x, y)$, wind velocity $(w_u, w_v)$ on the training data (10,530 images).

Data availability

The data that support the findings of this study are available from the corresponding authors upon request.

Code availability

The deep learning architecture, as well as the trained model and statistical analysis are based on already published work and it has been detailed in the document for the reader reproduction. The Tensorflow version used was 2.5.0 and the Keras version was 2.2.4.

References

1. Franks, P. J. Spatial patterns in dense algal blooms. *Limnol. Oceanogr.* **42**, 1297–1305 (1997).
2. Hugenholtz, C. H., Levin, N., Barchyn, T. E. & Baddock, M. C. Remote sensing and spatial analysis of aeolian sand dunes: A review and outlook. *Earth-science reviews* **111**, 319–334 (2012).
3. McMillan, W. W. *et al.* Daily global maps of carbon monoxide from NASA’s Atmospheric Infrared Sounder. *Geophys. Res. Lett.* **32**, DOI: 10.1029/2004GL021821 (2005). eprint: https://onlinelibrary.wiley.com/doi/pdf/10.1029/2004GL021821.
4. Poulidis, A. P. *et al.* Meteorological Controls on Local and Regional Volcanic Ash Dispersal. *Sci Rep* **8**, 6873, DOI: 10.1038/s41598-018-24651-1 (2018).
5. Faivre, N., Jin, Y., Goulden, M. L. & Randerson, J. T. Controls on the spatial pattern of wildfire ignitions in southern California. *Int. J. Wildland Fire* **23**, 799–811 (2014).

6. Lorenz, E. N. *Empirical orthogonal functions and statistical weather prediction*, vol. 1 (Massachusetts Institute of Technology, Department of Meteorology Cambridge, 1956).

7. Higdon, D., Gattiker, J., Williams, B. & Rightley, M. Computer Model Calibration Using High-Dimensional Output. *J. Am. Stat. Assoc.* **103**, 570–583, DOI: 10.1198/016214507000000888 (2008).

8. Francom, D., Sansó, B., Bulaevskaya, V., Lucas, D. & Simpson, M. Inferring Atmospheric Release Characteristics in a Large Computer Experiment Using Bayesian Adaptive Splines. *J. Am. Stat. Assoc.* **114**, 1450–1465, DOI: 10.1080/01621459.2018.1562933 (2019).

9. Held, D., Thrun, S. & Savarese, S. Learning to track at 100 fps with deep regression networks. In *European conference on computer vision*, 749–765 (Springer, 2016).

10. He, K., Gkioxari, G., Dollár, P. & Girshick, R. Mask r-cnn. In *Proceedings of the IEEE international conference on computer vision*, 2961–2969 (2017).

11. Redmon, J. & Farhadi, A. Yolov3: An incremental improvement. *arXiv preprint arXiv:1804.02767* (2018).

12. Karras, T., Laine, S. & Aila, T. A style-based generator architecture for generative adversarial networks. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, 4401–4410 (2019).

13. Sadeghi, M. et al. PERSIANN-CNN: Precipitation Estimation from Remotely Sensed Information Using Artificial Neural Networks–Convolutional Neural Networks. *J. Hydrometeorol.* **20**, 2273–2289, DOI: 10.1175/JHM-D-19-0110.1 (2019).

14. Rumelhart, D. E., Hinton, G. E. & Williams, R. J. Learning representations by back-propagating errors. *nature* **323**, 533–536 (1986).

15. Plaut, E. From principal subspaces to principal components with linear autoencoders. *arXiv preprint arXiv:1804.10253* (2018).

16. Sharma, S., Sharma, S. & Athaiya, A. Activation functions in neural networks. *Towards data science* **6**, 310–316 (2017).

17. Anirudh, R., Thiagarajan, J. J., Bremer, P.-T. & Spears, B. K. Improved surrogates in inertial confinement fusion with manifold and cycle consistencies. *Proc. Natl. Acad. Sci.* **117**, 9741–9746 (2020).

18. Kong, Q. et al. Deep convolutional autoencoders as generic feature extractors in seismological applications. *arXiv preprint arXiv:2110.11802* (2021).

19. Deng, L. The MNIST database of handwritten digit images for machine learning research. *IEEE Signal Process. Mag.* **29**, 141–142 (2012).

20. Gowardhan, A. A. et al. Large Eddy Simulations of Turbulent and Buoyant Flows in Urban and Complex Terrain Areas Using the Aeolus Model. *Atmosphere* **12**, 1107, DOI: 10.3390/atmos12091107 (2021). Number: 9 Publisher: Multidisciplinary Digital Publishing Institute.

21. Krishnan, S. How to determine the number of layers and neurons in the hidden layer? Date 2022-01-28. https://medium.com/geekculture/introduction-to-neural-network-2f8b8221fbd3.

22. Mosca, S., Graziani, G., Klug, W., Bellasio, R. & Bianconi, R. A statistical methodology for the evaluation of long-range dispersion models: an application to the ETEX exercise. *Atmospheric Environ.* **32**, 4307–4324, DOI: 10.1016/S1352-2310(98)00179-4 (1998).

23. Kingma, D. P. & Welling, M. Auto-encoding variational bayes. *arXiv preprint arXiv:1312.6114* (2013).

24. Goodfellow, I. et al. Generative adversarial nets. *Adv. neural information processing systems* **27** (2014).

25. Collet, F. Building Autoencoders in Keras. Date 2021-08-30. https://blog.keras.io/building-autoencoders-in-keras.html.

26. Kingma, D. P. & Ba, J. Adam: A Method for Stochastic Optimization. *arXiv:1412.6980 [cs]* (2017).

27. Vincent, P., Larochelle, H., Bengio, Y. & Manzagol, P.-A. Extracting and composing robust features with denoising autoencoders. In *Proceedings of the 25th international conference on Machine learning*, 1096–1103 (2008).

28. Costa, A. F., Santos, M. S., Soares, J. P. & Abreu, P. H. Missing data imputation via denoising autoencoders: the untold story. In *International symposium on intelligent data analysis*, 87–98 (Springer, 2018).
Acknowledgments

This work was performed under the auspices of the United States Department of Energy (DOE) by Lawrence Livermore National Laboratory under contract DE-AC52-07NA27344 and public release number LLNL-JRNL-831189. This research was funded by the National Nuclear Security Administration, Defense Nuclear Nonproliferation Research and Development (NNSA DNN R&D). The authors acknowledge important interdisciplinary collaboration with scientists and engineers from LANL, LLNL, MSTS, PNNL, and SNL.

The authors would like to thank Gemma J. Anderson, Akshay Gowardhan, and Nipun Gunawardena for their valuable guidance and Lee G. Glascoe and Stephen C. Myers for supporting this publication.

Author contributions statement

M.G.F.G. : Writing – original draft, Visualization, Formal analysis, Investigation, Conceptualization. D.D.L : Supervision, Project administration, Conceptualization, Methodology, Writing – review & editing. Q.K. : Supervision, Methodology, Writing – review & editing