Siamese neural networks for a generalized, quantitative comparison of complex model outputs

Colin G. Cess¹ and Stacey D. Finley¹,²,³*

¹Department of Biomedical Engineering, University of Southern California, Los Angeles, CA
²Department of Quantitative and Computational Biology, University of Southern California, Los Angeles, CA
³Mork Family Department of Chemical Engineering and Materials Science, University of Southern California, Los Angeles, CA

ABSTRACT
Computational models are quantitative representations of systems. By analyzing and comparing the outputs of such models, it is possible to gain a better understanding of the system itself. Though, as the complexity of model outputs increases, it becomes increasingly difficult to compare simulations to each other. While it is straightforward to only compare a few specific model outputs across multiple simulations, it is more informative to be able to compare model simulations as a whole. However, it is difficult to holistically compare model simulations in an unbiased manner. To address these limitations, we use Siamese neural networks to compare model simulations as a single value, with the neural networks capturing the relationships between the model outputs. We provide an approach to training Siamese networks on model simulations and display how the trained networks can then be used to provide a holistic comparison of model outputs. This approach can be applied to a wide range of model types, providing a quantitative method of analyzing the complex outputs of computational models.

1 INTRODUCTION
As the modern century marches forward, computing power continues to increase. At the same time, advances in experimental techniques enable one to capture enormous amounts of data¹–⁵. By taking advantage of these powerful computing resources and large experimental datasets, scientists can construct larger and more complex computational models of various systems⁶–¹³. These models allow us to interrogate a system in extensive detail, predicting how the system responds to various perturbations. In some cases, models enable us to examine pieces of the system that cannot yet be experimentally measured. In addition, models permit us to perform large-scale simulations to produce in silico data that would either take too much time or be too costly to generate experimentally.

To understand the system being modeled, these model simulations must be properly analyzed. Many methods currently exist for analyzing computational models and in silico data. Some methods, such as sensitivity analyses or parameter sweeps, are designed specifically to examine a model¹⁴–²¹. Other methods, such as partial least squares and clustering, are typically used to analyze data²²–²⁵ but can easily be used to analyze models as well²⁶. However, a current drawback of most model analysis methods is that they generally examine only a piece of the system, rather than the whole set of simulated data²⁷–³¹. For example, performing a sensitivity analysis requires one to explicitly define the output of interest, and the output selected influences the analysis itself. Consider a system that oscillates with time. For one variable, we can easily calculate outputs such as the amplitude and frequency of peaks. But as the number of variables increases, to get a complete view of the system, we would have to account for how the amplitudes and frequencies for each variable relate to those of every other variable. To be complete, we must also consider the possibility that a variable may not oscillate at all parameter values being simulated. Additionally, we would have to scale the different comparisons to eliminate potentially unjust influence based on output values. This could potentially introduce bias from the researcher, who would have to manually determine these comparison metrics and could easily fail to account for every possible comparison. This example highlights limitations in obtaining a holistic and unbiased comparison of model outputs.

To address these limitations, in this study, we present a generalized, model-agnostic method for comparing complex model outputs as a single, continuous value by using Siamese neural networks. Siamese networks are a pair of identical neural networks that project a pair of inputs to individual, low-dimensional points. By taking the distance between these points, the similarity between the inputs can be determined. Siamese networks have been used to detect signature forgeries, to interpret single-cell data, for face verification, and as a measure of continuous disease progression. Additionally, as a form of dimensionality reduction, Siamese networks have been shown to work much better than principal component analysis for separating images of different classes.

Here, we use Siamese networks to compare the outputs of complex computational models. By projecting outputs to low-dimensional points and taking the distance between points, we can determine how different two simulations are. The distance between the points provides a single, holistic value that accounts for the complex relationships between model output features that would be difficult to otherwise calculate manually. We display our approach on several example models that have very different types of outputs to show its broad applicability.
2 RESULTS

We first clarify some terminology to eliminate potential confusion throughout the paper. The term “model” is used to refer to any sort of computational model, such as systems of differential equations, that can produce a “model output”, any simulation or calculation performed by a model. These two terms, model and model output, specifically refer to the computational models that are being analyzed, and not to the neural networks that are used to perform the analysis. The term “Siamese network” is used to refer to the method of training neural networks and the use of those networks to compute the distances between pairs of inputs.

Here, we describe our approach and provide three examples where we apply the approach to models that produce very different outputs. Our goal with this study is to display the approach’s utility and applicability to a wide variety of model types, and how it can potentially be used to examine each of these models. We briefly discuss each model and the analysis results so that the analyses can be understood in the context of the model and how they can be applied to similar models. However, we avoid discussing the implications of the results on the modeled systems, as our aim is to display how this same approach can be applied to disparate models, and not to draw novel insights about the systems. The three model types that we examine are a metabolic flux model (one-dimensional vector output), an ordinary differential equation model (time-series output), and an agent-based model (spatial output). We provide two tests for each model. The first is the same for each (shifting a single parameter value) to display how we can use the output of the Siamese network to perform the same analyses on very different models.

We start with an overview of Siamese networks along with our training method. We then discuss each model individually, providing an overview on how the approach is applied and showing the example analyses. The full description of our approach is detailed in the Methods.

2.1 Siamese networks: Overview and training

Our comparison approach makes use of Siamese networks to reduce the dimensionality of model outputs and then calculate the distance between the projected points to determine how different they are from each other. A schematic of this is displayed in Figure 1. To be trained, Siamese networks require examples of several inputs and simply the knowledge that two inputs are either similar or dissimilar. We produce training datasets by performing Monte Carlo simulations for each model, randomly sampling model parameters. Each simulation is treated as a unique input that is dissimilar from other simulations. Once we have this training dataset, we train the Siamese networks using triplet loss, where an anchor input is compared to a positive (similar) and a negative (dissimilar) input. We create the positive inputs by adding Gaussian noise to the anchor. Triplet loss aims to minimize the distance between the anchor and positive, while maximizing the distance between the anchor and negative.

2.2 Example 1: Flux model

The first example uses a model of metabolic reactions for E. coli, containing 2,477 reactions organized into 44 subsystems. Here, the model parameters are the upper and lower flux bounds for each reaction. These bounds are fixed, and
the model estimates the reaction fluxes within the bounds needed to optimize a specified objective function. Positive fluxes represent flow in the forward direction, while negative fluxes are flow in the reverse direction. The output for this model is a one-dimensional vector of the flux through each reaction for the optimized state. Because the output is a simple vector, we use a basic feedforward neural network to analyze this model.

The first test we perform on this model is shifting the lower bound for uptake of uridine diphosphate glucose (a nutrient for the organism) from 0 to -1,000 (in the model, negative uptake corresponds to flow into the cell) and compute the distance of the optimized flux state to that of when the lower bound is 0. This allows us to see how the overall metabolic state changes as the cell becomes able to take up more of this nutrient. We find that there is a large increase in the distance from the base state once the organism becomes able to uptake this nutrient (Figure 2), meaning that this nutrient causes a distinct change in the metabolic pathways utilized from the base state. This difference continues to increase as E. coli can take up more of the nutrient, until it finally levels off, with increased potential uptake not changing the metabolic state.

The second test we perform, which is very specific to this type of model, is a series of knockouts. Here, we set the upper and lower bounds for a specific reaction equal to zero before solving the model for fluxes through the rest of the reactions (Figure 3). This means the organism cannot utilize that particular reaction, and it has to direct flux through alternative pathways. Because of the large number of reactions in this model, it would be difficult to compare the effect of each individual knockout on the estimated flux through each metabolic reaction. Instead, we examine the average change for knockouts in each subsystem. For each subsystem, we knockout each reaction individually, compute the distance to the base state, and then average the distances for each reaction in the subsystem. This allows us to determine which subsystems contribute the most to the base state and thus could be subject to further analysis.

### 2.3 Example 2: Ordinary differential equation model

The second example is a Lotka-Volterra model, which is comprised of ordinary differential equations. This model has been used in many fields, such as ecology, chemistry, and economics. We chose this model because of its broad usage and because, depending on parameter values, it can reach a steady state or produce oscillations. Specifically, we use a
four-species model, setting the base parameters to a set listed in Vano et al., who examined chaotic behavior in this system\textsuperscript{51}. We increase three of the parameter values from zero to 0.01 (shown in equation (3) of Vano et al.), which maintains the oscillatory behavior, but allows us to sample these parameters above and below the base value of 0.01 when producing the training dataset. The output of this model is a vector of values at each time point for each species. We organize the model outputs as a two-dimensional matrix, \textit{timesteps} \times \textit{species}. The Siamese network we use for this type of model is a one-dimensional convolutional neural network, which has been used to classify time-series data\textsuperscript{52,53}.

As in the first model example, the first test that we perform is shifting the value of a parameter; here, from two-fold below to two-fold above the base value. We calculate the distance between the output from each parameter value and the base output (Figure 4). As we would expect, the further the parameter value is shifted from the base value, the further the overall output is. Overlays show the simulations at the lowest parameter value, base parameter value, and maximum parameter value in order to illustrate how the model outputs differ. This clearly demonstrates that even a simple model can produce complex outputs that are difficult to directly compare when having to manually calculate their differences.

Our second test for this model is a local sensitivity analysis (Figure 5), which is used to determine how sensitive a model output is to changes in a parameter. A local sensitivity analysis changes one parameter at a time from a base value (here, increasing and decreasing by 10\%) and records the corresponding change in a specified output. We consider two cases: (1) using a Siamese network to holistically compare the model outputs and (2) specifying a single model output. With the Siamese network, we calculate the distance between the output from the perturbed parameter and the base output to get an overall comparison between the two. For the specified output, we use the average change of all four final values of the differential equations. This output is only capturing one moment in time, and fails to account for temporal behaviors, such as the oscillations that this model is able to produce. One common usage for a sensitivity analysis is to determine which parameters most impact the output, and thus are most important for parameter estimation. With this in mind, we averaged the sensitivity results for increasing and decreasing each parameter value for each case (Siamese network and specified output) and ranked the parameters by the most sensitive. Because the Siamese network outputs a distance in low-dimensional space, while the specified output is a percent change from the base, we normalize the sensitivities to between 0 and 1 so that we can compare them qualitatively. Comparing the two cases, we find that some parameters (3, 11, 16) have similar rankings, whereas other parameters (6, 14, 19) have very different rankings. Because the specified output only looks at one point in time, and because the model oscillates, it could over- or under-estimate a parameter’s importance.

### 2.4 Example 3: Spatial agent-based model

The final example is an agent-based model (ABM) of tumor-immune interactions\textsuperscript{54}. ABMs are used in many disparate fields to explore how interactions between individuals yield complex, emergent behaviors at the population level\textsuperscript{55–65}. This ABM is an on-lattice model consisting of three cell types (cancer cells, T cells, and macrophages) that can take on a total of six states. Our output is the final spatial layout, which we format as a three-dimensional matrix. The first two dimensions are the lattice dimensions of the model, and the third dimension is for each cell state. Because this is a similar format to an image, with the number of cell states being akin to color-channels, we used a two-dimensional convolutional neural network\textsuperscript{66–69}. 

![Figure 4](image1.png)

**Figure 4.** Shifting the value of a single parameter over a two-fold range (x-axis) and calculating the distance from the center parameter (y-axis).

![Figure 5](image2.png)

**Figure 5.** Local sensitivity analysis for the second test model. Compares the most sensitive parameters when using a Siamese network as the output (top) to using the final value of each variable as the output (bottom). Values are normalized to the maximum for each sensitivity output.
As with the previous examples, we first performed a single parameter shift. We varied the tumor cell proliferation rate across 21 different values (Figure 6), with the base simulation being the middle value. Because this is a stochastic model, we performed each simulation with ten replicates. When taking the distance between simulations for two different parameter sets, we compared each replicate to each other replicate and then took the average distance between them. What is immediately noticeable is that the distance for the base simulation is not zero. This is due to the stochasticity of the model, producing simulations that are not exactly the same as each other, meaning that the projected points for the base simulation are also slightly different.

The final test we performed was clustering the Monte Carlo simulations that produced the training dataset (Figure 7). We use the distance calculated by the Siamese network to group the simulations via hierarchical clustering. From these clusters, we examined the distributions for model parameters that produced the simulations (Figure 7, top two rows). We looked at different numbers of clusters, with three being the largest number that maintained clear separation in the parameter distributions. We see that very clear differences exist in the distributions for the parameters, allowing us to identify how different model end-states are similar to each other based on model parameter values. We can also look at selected specific model outputs (Figure 7, bottom row), which gives insight into some of the systemic characteristics of each cluster. For example, we see that, in the green cluster, low numbers of cancer cells are clustered with high numbers of T cells, which kill cancer cells. Additionally, these two outputs are clustered with a low macrophage recruitment rate. This result makes sense as, in this model, macrophages suppress T cell activity. This is an example on how our approach can be used to examine how overall model output is linked to model parameters and specific outputs.

3 DISCUSSION

With this study, we display how Siamese neural networks can be used to compare complex model simulations as a single quantitative value. We show how the same method can be applied to three very different types of models, only needing to change the neural network structure to accommodate the format of the model output. By using a neural network to reduce the model output to a low-dimensional point and taking the distance between two points, we can obtain a holistic view of how different two simulations are from each other without the need to manually calculate complex comparison metrics.

To display the utility of our approach, we provided a total of six tests performed across three example models, showing

![Figure 6](image6.png)

Figure 6. Shifting the value of one parameter (x-axis), comparing the distance calculated by a Siamese network to the middle parameter value (y-axis).

![Figure 7](image7.png)

Figure 7. Clustering Monte Carlo simulations for the third test model. Distributions four sampled parameters are shown for each of the three clusters in the white plots (top two rows) and for two selected model outputs in the gray plots (bottom row).
how it allows us to compare model simulations in an unbiased manner. The first test for each model was similar, where we shift the value of a single parameter across a range of values. This simple test displays how model outputs change overall as a parameter is varied, allowing for the determination of areas of parameter space where the model output changes more drastically, or the identification of potential regions where bifurcated behavior is exhibited. For the first example model, we also displayed how our approach can be applied to gene knockout simulations. Often, these simulations are used to identify genetic perturbations that optimize a specific biological process. Our approach can extend these analyses to determine how different the new metabolic states are from the original or from other knockout states. Our second example model is a time series differential equation model used in a variety of fields. We show how our approach can be used to aid in sensitivity analyses, which can then identify important parameters for parameter fitting, model reduction, and model expansion. Lastly, we used our approach to cluster simulations from an agent-based model. From these clusters, we were able to examine how different parameter values tend to create different simulation end-states, along with how specific model outputs are linked to the different end-states. In the context of the tumor model that we examined, our approach would allow us to better predict how different tumor properties yield different types of tumors.

We only displayed simple examples of analyses so that our approach could be easily understood; however, it can be extended to more complex analyses. For example, the sensitivity analysis that we showed was a simple, local analysis. However, this only captures a small region of parameter space, whereas global sensitivity analyses yield a better exploration of the model. Instead of comparing to a base parameter set, each dimension of the projected points can be treated as an output for a global sensitivity analysis, providing a more exhaustive description of how sensitive the model as a whole is to each parameter. This approach could also be used to aid in uncertainty and robustness analyses. By accounting for model behavior as a whole instead of only focusing on a small part of the output space, our approach provides a better characteristic of the system. Another potential use is for optimizing a perturbation to a system that would produce a desired change in an output of interest while minimizing the overall change in the system.

The main limitation with our approach is that it can be computationally expensive. Smaller models can generate large numbers of training samples quickly; however, as model complexity increases, the time required to perform Monte Carlo simulations increases greatly. Additionally, it takes time to determine optimal neural network structure, especially when more complex neural networks need to be used, based on the format of model outputs. However, once a neural network is trained, it can be used for many different analyses.

Additionally, because neural networks are black-box methods, it is difficult to interpret how model outputs are being projected into low-dimensional space. Our approach simply determines how different two simulations are, not why they are different. However, as we show with the third example, combining this approach with specific model outputs helps yield further insight into the system.

Despite these limitations, we present an approach to compare model outputs via Siamese networks. This work addresses the limitations of most commonly used model analysis methods; namely, a narrowly focused and potentially biased comparison. We show how these networks can be trained on a set of model-generated data and how the trained networks can be applied to analyze a range of model outputs. We demonstrate that this approach provides an additional way to interrogate models beyond current methods.

4 METHODS

The approach we detail here uses Siamese neural networks to compare the outputs of a computational model as a single, continuous value. In the following sections, we describe: (1) an overview of Siamese networks, (2) generation of training data, (3) neural network construction for each example model, (4) neural network training, (5) implementation of the Siamese network to analyze model outputs. The neural network construction subsection is the only subsection that is model specific, meaning that considerations must be taken based on the type of model being analyzed and how its outputs are formatted.

Our study focuses on how Siamese neural networks can be applied to analyze the outputs of computational models and what considerations in neural network structure and data-processing need to be accounted for, based on the model. Therefore, we only discuss the neural network details that are directly relevant to our approach.

Overview of Siamese neural networks. Neural networks are used to predict specific outputs based on inputs, in a “black-box” approach. A very common use of neural networks is classification, which typically requires large amounts of data and all of the classes to be known at the time of training. However, there are instances when there is insufficient data available and not all classes are present in the training data. Siamese neural networks, on the other hand, simply compute how similar two inputs are, instead of directly determining which class an input belongs to. The final neuron layer of the neural network is treated as a multi-dimensional point, thus the neural network is a form of non-linear dimensionality reduction. There are multiple ways that the points created from a set of inputs can be compared. In this study, we use Euclidean distance. For a complete description of Siamese networks, we point the reader to Chicco 2021.
Generation of training data. To train the neural networks, we create a dataset of model simulations as our training data. We generate the training data in the same way for each model, by performing Monte Carlo simulations randomly sampling model parameters. This produces a wide range of possible outputs, so that the Siamese network is trained on many areas of the input parameter space. We perform 10,000 simulations for the first and second example models, and 1,000 for the third, with 10 replicates (as that model is stochastic), for a total of 10,000 simulations. Sampling ranges were chosen based on the model and are the maximum ranges that we would examine for each parameter.

When training the Siamese network (detailed in a following section), inputs are compared that are either similar or dissimilar to each other. Generally, this is applied to classification, with similar inputs being from the same class and dissimilar inputs being from different classes. Here, we have no separate classes to draw training inputs from. Instead, we treat each simulation as an individual and create similar inputs by adding Gaussian noise to the simulations.

Neural network construction. We construct and implement our neural networks in Python using TensorFlow. Due to the nature of the outputs for each of our example models, we need to use different neural network structures for each model. We chose example models that have unique output types in order to display how different neural network structures can be used with this one approach. We acknowledge that we are not performing an exhaustive examination of every possible model output type.

In general, the type of neural network that is used is influenced by the format of the computational model output. In this study, we assume that a model always produces simulations of the same size and shape. For example, a time-series simulation is always of the same duration and time-step. If a model outputs a list of unordered values (meaning that changing the order of the outputs does not change their meaning), then a simple feedforward architecture would suffice for the neural network. However, if the model outputs must be ordered in a specific way to understand them (such as temporal or spatial outputs), then more complex neural network structures must be used. In most cases, a convolutional neural network can be successfully applied. One-dimensional convolutional networks have been successful in classifying time-series data, while two-dimensional convolutional networks are regularly used for image (spatial) data. With sufficient computing power, a three-dimensional convolutional network can be used for spatiotemporal outputs.

Neural network training. The loss function that we use for Siamese network training is triplet loss. This takes in three training inputs at a time: an anchor (A), a positive (P) sample (similar to A), and a negative (N) sample (dissimilar to A). Equation 1 displays how these inputs are compared.

\[
\mathcal{L}(A, P, N) = \max(D(A, P) - D(A, N) + \text{margin}, 0)
\]

In brief, loss is calculated by taking the distance (D) between the neural network output vectors for A and P, and comparing it to the distance between the outputs for A and N. This loss function outputs a loss value when the D(A,N) is not greater than D(A,P) by at least the margin. Here, we use Euclidean distance as our distance function.

To train the Siamese network for each model, we randomly generate triplets. Triplets are formed from two randomly selected simulations (A and N), and P is generated by adding Gaussian noise to A. We then produce additional triplets as a means of validating the model. Simulations that were used to form validation triplets were not included in the set of simulations that produced the training triplets.

To determine how many simulations to use for training, and how many triplets to form, we compare the final validation loss for different amounts of training simulations and training triplets, using 1,000 validation triplets. When using a small number of training triplets, we find that there are minor differences between the different numbers of training samples, however, as the number of training triplets increases, validation loss decreases and there is little difference with the number of training simulations used (Figure S1). Based on this, we decided to use 90% of the simulations for training and formed 10,000 triplets.

Different scaling methods can be considered for the model outputs, based on the interest of the analysis. For the flux model (example model 1), each output can take on different positive or negative values. We divide each output by the maximum absolute output, so the absolute maximum for each simulation is 1. For the differential equation model (example model 2), which only produces positive values, we scale each time-series to have a maximum of 1. The output from the agent-based model (example model 3) is already between 0 and 1, so we do not scale it.

Implementation of Siamese networks. After training the Siamese network, its main use is to reduce the original model output, from either a high-dimensional data-point or a series of points with complex spatial or temporal relationships, to a single, low-dimension point. From this projected point, we can find the Euclidean distance from another projected point (representing another model output) in order to determine how different the two outputs are. An advantage with this approach is that there is no need to manually calculate complex comparison metrics. We note that the distance between two projected points can be influenced by the neural network structure and training, and thus has little meaning by itself. Instead, comparisons between multiple outputs are needed to give it meaning. For example, for outputs O₁, O₂, and O₃, the distance between O₁ and O₂ alone provides little information, however if that distance is smaller than the distance between
O₁ and O₃, we conclude that O₁ is more similar to O₂ than it is to O₃. Besides taking the distance between two model outputs, we can use the Siamese network to project many model outputs into low-dimensional space to easily perform cluster analyses (see application to example model 3).

In practice, we use Siamese networks in ensembles of 50 and average the distances that they calculate. In this study, for consistency, all of the neural networks have a final output layer consisting of two neurons with a linear activation function, however the number of neurons in the final layer is a hyperparameter that could be considered when determining neural network structure. Neural network hyperparameters (such as the number of neurons in the hidden layers) were set to minimize validation loss. An example of training each Siamese network for different neural network structures is shown in Figure S2.

ACKNOWLEDGEMENTS

The authors thank members of the Finley research group for critical comments and suggestions.

REFERENCES

1. Cremin, C. J., Dash, S. & Huang, X. Big data: Historic advances and emerging trends in biomedical research. *Current Research in Biotechnology* (2022).
2. Dash, S., Shakyawar, S. K., Sharma, M. & Kaushik, S. Big data in healthcare: management, analysis and future prospects. *Journal of Big Data* 6, 1–25 (2019).
3. Risteviski, B. & Chen, M. Big data analytics in medicine and healthcare. *Journal of integrative bioinformatics* 15, (2018).
4. Garcia-Milian, R., Hersey, D., Vukmirovic, M. & Dupriot, F. Data challenges for biomedical researchers in the age of omics. *PeerJ* 6, e5553 (2018).
5. Stephens, Z. D. et al. Big data: astronomical or gemocical? *PLoS biology* 13, e1002195 (2015).
6. Metzcar, J., Wang, Y., Heiland, R. & Macklin, P. A review of cell-based computational modeling in cancer biology. *JCO clinical cancer informatics* 2, 1–13 (2019).
7. Mehta, C. H., Narayan, R. & Nayak, U. Y. Computational modeling for formulation design. *Drug Discovery Today* 24, 781–788 (2019).
8. Guest, O. & Martin, A. E. How computational modeling can force theory building in psychological science. *Perspectives on Psychological Science* 16, 789–802 (2021).
9. Jenner, A. L., Aogo, R. A., Davis, C. L., Smith, A. M. & Craig, M. Leveraging computational modeling to understand infectious diseases. *Current Pathobiology Reports* 8, 149–161 (2020).
10. Graiciar, L. et al. Towards operando computational modeling in heterogeneous catalysis. *Chemical Society Reviews* 47, 8307–8348 (2018).
11. Li, J. et al. A review of computational modeling techniques for wet waste valorization. Research trends and future perspectives. *Journal of Cleaner Production* 133025 (2022).
12. Grammatikopoulos, P., Sowwan, M. & Kioseoglou, J. Computational modeling of nanoparticle coalescence. *Advanced Theory and Simulations* 2, 1900013 (2019).
13. Jafarzadeh, S., Chen, Z. & Bobaru, F. Computational modeling of pitting corrosion. *Corrosion reviews* 37, 419–439 (2019).
14. Derjany, P., Namilaee, S. & Srinivasan, A. Parameter Space Exploration in Pedestrian Queue Design to Mitigate Infectious Disease Spread. *Journal of the Indian Institute of Science* 101, 329–338 (2021).
15. Koo, H. et al. Position paper: Sensitivity analysis of spatially distributed environmental models—a pragmatic framework for the exploration of uncertainty sources. *Environmental modelling & software* 134, 104857 (2020).
16. Marrel, A. & Chabridon, V. Statistical developments for target and conditional sensitivity analysis: application on safety studies for nuclear reactor. *Reliability Engineering & System Safety* 214, 107711 (2021).
17. Pang, Z., O’Neill, Z., Li, Y. & Niu, F. The role of sensitivity analysis in the building performance analysis: A critical review. *Energy and Buildings* 209, 109859 (2020).
18. Qian, G. & Mahdi, A. Sensitivity analysis methods in the biomedical sciences. *Mathematical biosciences* 323, 108306 (2020).
19. Sadeghi Lahjani, M., Gayatri, R., Islam, T., Srinivasan, A. & Namilaee, S. Architecture-aware modeling of pedestrian dynamics. *Journal of the Indian Institute of Science* 101, 341–356 (2021).
20. Saltelli, A. et al. Why so many published sensitivity analyses are false: A systematic review of sensitivity analysis practices. *Environmental modelling & software* 114, 29–39 (2019).
21. Wagen, T. & Pianosi, F. What has Global Sensitivity Analysis ever done for us? A systematic review to support scientific advancement and to inform policy-making in earth system modelling. *Earth-science reviews* 194, 1–18 (2020).
22. Bayonne, E., Marin-Garcia, J. A. & Alfalla-Luque, R. Partial least squares (PLS) in Operations Management research: Insights from a systematic literature review. *Journal of Industrial Engineering and Management* 13, 565–597 (2020).
23. Khatri, P., Gupta, K. K. & Gupta, R. K. A review of partial least squares modeling (PLS) for water quality analysis. *Modeling Earth Systems and Environment* 7, 705–714 (2021).
24. Lee, L. C., L iong, C.-Y. & Jemain, A. A. Partial least squares-discriminant analysis (PLS-DA) for classification of high-dimensional (HD) data: a review of contemporary practice strategies and knowledge gaps. *Analyst* 143, 3526–3539 (2018).
25. Purwanto, A. & Sudargini, Y. Partial least squares structural equation modeling (PLS-SEM) analysis for social and management research: a literature review. *Journal of Industrial Engineering & Management Research* 2, 114–123 (2021).
26. Cess, C. G. & Finley, S. D. Data-driven analysis of a mechanistic model of CAR T cell signaling predicts effects of cell-to-cell heterogeneity. *Journal of Theoretical Biology* 489, 110125 (2020).
27. Campbell, K., McCoy, M. D. & Williams, B. J. Sensitivity analysis when model outputs are functions. *Reliability Engineering & System Safety* 91, 1468–1472 (2006).
28. Lenhart, T., Eckhardt, K., Fohrer, N. & Frede, H.-G. Comparison of two different approaches of sensitivity analysis. *Physics and Chemistry of the Earth, Parts A/B/C* 27, 645–654 (2002).
29. Li, D. & Finley, S. D. Mechanistic insights into the heterogeneous response to anti-VEGF treatment in tumors. *Computational and Systems Oncology* 1, e1013 (2021).
30. Mortlock, R. D., Georgia, S. K. & Finley, S. D. Dynamic regulation of JAK-STAT signaling through the prolactin receptor predicted by computational modeling. *Cellular and molecular bioengineering* 14, 15–30 (2021).
31. Song, M. & Finley, S. D. Mechanistic characterization of endothelial sprouting mediated by pro-angiogenic signaling. *Microcirculation* 29, e12744 (2022).
32. Chicco, D. Siamese neural networks: An overview. *Artificial Neural Networks* 73–94 (2021).
33. Ilina, O., Zyadinov, V., Klennon, N. & Tereshonok, M. A Survey on Symmetrical Neural Network Architectures and Applications. *Symmetry* 14, 1391 (2022).
34. Bromley, J., Guyon, I., LeCun, Y., Säckinger, E. & Shah, R. Signature verification using a siamese time delay neural network. *Advances in neural information processing systems* 6, (1993).
35. Szubert, B., Cole, J. E., Monaco, C. & Drozdov, I. Structure-preserving visualisation of high dimensional single-cell datasets. *Scientific reports* 9, 1–10 (2019).
36. Chopra, S., Hadsell, R. & LeCun, Y. Learning a similarity metric discriminatively, with application to face verification. In 2005 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR’05) vol. 1 539–546 (IEEE, 2005).
37. Li, M. D. et al. Siamese neural networks for continuous disease severity evaluation and change detection in medical imaging. *NPJ digital medicine* 3, 1–9 (2020).
38. Zheng, L., Duffner, S., Idrissi, K., Garcia, C. & Baskurt, A. Siamese multi-layer perceptrons for dimensionality reduction and face identification. *Multimedia Tools and Applications* 75, 5055–5073 (2016).
39. Chen, T., Kornblith, S., Norouzi, M. & Hinton, G. A simple framework for contrastive learning of visual representations. In International conference on machine learning 1597–1607 (PMLR, 2020).
40. Hoffer, E. & Alon, N. Deep metric learning using triplet network. In International workshop on similarity-based pattern recognition 84–92 (Springer, 2015).
41. Schröff, F., Kalenichenko, D. & Philbin, J. Facenet: A unified embedding for face recognition and clustering. In Proceedings of the IEEE conference on computer vision and pattern recognition 815–823 (2015).
42. Wang, J. et al. Learning fine-grained image similarity with deep ranking. In Proceedings of the IEEE conference on computer vision and pattern recognition 1386–1393 (2014).
43. Archer, C. T. et al. The genome sequence of E. coli W (ATCC 9637): comparative genome analysis and an improved genome-scale reconstruction of E. coli. BMC genomics 12, 1–20 (2011).
44. Orth, J. D., Thiele, I. & Palsson, B. B. What is flux balance analysis? Nature biotechnology 28, 245–248 (2010).
45. Choon, Y. W. et al. Gene knockout identification using an extension of bees hill flux balance analysis. BioMed research international 2015, 1 (2015).
46. Chu, P. S. et al. Identifying a gene knockout strategy using a hybrid of the bat algorithm and flux balance analysis to enhance the production of succinate and lactate in Escherichia coli. Biotechnology and Bioprocess Engineering 20, 349–357 (2015).
47. Saleh, A. H. M. et al. Gene knockout identification for metabolite production improvement using a hybrid of genetic ant colony optimization and flux balance analysis. Biotechnology and bioprocess engineering 20, 685–693 (2015).
48. Gandolfo, G. Giuseppe palomba and the lotka-volterra equations. Rendicont Lincol 19, 347–357 (2008).
49. Gavina, M. K. A. et al. Multi-species coexistence in Lotka-Volterra competitive systems with crowding effects. Scientific reports 8, 1–8 (2018).
50. Hering, R. H. Oscillations in Lotka-Volterra systems of chemical reactions. Journal of mathematical chemistry 5, 197–202 (1990).
51. Vano, J., Wildenberg, J., Anderson, M., Noei, J. & Sprott, J. Chaos in low-dimensional Lotka-Volterra models of competition. Nonlinearity 19, 2391 (2006).
52. Kuang, D. A 1d convolutional network for leaf and time series classification. arXiv preprint arXiv:1907.00069 (2019).
53. Tang, W. et al. Omni-Scale CNNs: a simple and effective kernel size configuration for time series classification. In International Conference on Learning Representations (2021).
54. Cess, C. G. & Finley, S. D. Multi-scale modeling of macrophage—T cell interactions within the tumor microenvironment. PLoS Computational biology 16, e1006519 (2020).
55. Axelrod, R. Agent-based modeling as a bridge between disciplines. Handbook of computational economics 2, 1565–1584 (2006).
56. Bankes, S. C. Agent-based modeling: A revolution? Proceedings of the National Academy of Sciences 99, 7199–7200 (2002).
57. DeAngelis, D. L. & Diaz, S. G. Decision-making in agent-based modeling: A current review and future prospects. Frontiers in Ecology and Evolution 6, 237 (2019).
58. Glen, C. M., Kemp, M. L. & Voit, E. O. Agent-based modeling of morphogenetic systems: Advantages and challenges. PLoS computational biology 15, e1006577 (2019).
59. Khodabandelu, A. & Park, J. Agent-based modeling and simulation in construction. Automation in Construction 131, 103882 (2021).
60. Kremmydas, D., Athanasiadis, I. N. & Rozakis, S. A review of agent based modeling for agricultural policy evaluation. Agricultural systems 164, 95–106 (2018).
61. Lindkvist, E. et al. Navigating complexities: agent-based modeling to support research, governance, and management in small-scale fisheries. Frontiers in Marine Science 8, 733 (2020).
62. Macy, M. W. & Willer, R. From factors to actors: Computational sociology and agent-based modeling. Annual review of sociology 143–166 (2002).
63. Smith, E. R. & Conrey, F. R. Agent-based modeling: A new approach for theory building in social psychology. Personality and social psychology review 11, 87–104 (2007).
64. Stiele, D. et al. Agent-based modeling and simulation in architecture. Automation in Construction 141, 104426 (2022).
65. Tracy, M., Cerdá, M. & Keyes, K. M. Agent-based modeling in public health: current applications and future directions. Annual review of public health 39, 77 (2018).
66. Dhillon, A. & Verma, G. K. Convolutional neural network: a review of models, methodologies and applications to object detection. Progress in Artificial Intelligence 9, 85–112 (2020).
67. Li, Z., Liu, F., Yang, W., Peng, S. & Zhou, J. A survey of convolutional neural networks: analysis, applications, and prospects. IEEE transactions on neural networks and learning systems (2021).
68. Lindsay, G. W. Convolutional neural networks as a model of the visual system: Past, present, and future. Journal of cognitive neuroscience 33, 2017–2031 (2021).
69. Yamashita, R., Nishio, M., Do, R. K. G. & Togashi, K. Convolutional neural networks: an overview and application in radiology. Insights into imaging 9, 611–629 (2018).
70. Sreedhar Kumar, S., Madheswaran, M., Vinutha, B., Manjunatha Singh, H. & Charan, K. A brief survey of unsupervised agglomerative hierarchical clustering schemes. Int J Eng Technol (IJET) 8, 29–37 (2019).
71. Zhou, Y., Lu, Z., Xiao, S. & Yun, W. Distance correlation-based method for global sensitivity analysis of models with dependent inputs. Structural and Multidisciplinary Optimization 60, 1189–1207 (2019).
72. Dong, M. et al. Uncertainty and global sensitivity analysis of levelized cost of energy in wind power generation. Energy Conversion and Management 229, 113781 (2021).
73. Hafer, M., Koepfl, H., Hasler, M. & Wagner, A. “Glocal”robustness analysis and model discrimination for circadian oscillators. PLoS computational biology 5, e1000534 (2009).
74. Harter, H., Singh, M. M., Schneider-Marin, P., Lang, W. & Geyer, P. Uncertainty analysis of life cycle energy assessment in early stages of design. Energy and Buildings 208, 109635 (2020).
75. Tian, W. et al. An analysis of uncertainty analysis in building energy assessment. Renewable and Sustainable Energy Reviews 93, 285–301 (2018).
76. Weisberg, M. Robustness analysis. Philosophy of science 73, 730–742 (2006).
77. Abiodun, O. I. et al. State-of-the-art in artificial neural network applications: A survey. Helijon 4, e00938 (2018).
78. Abiodun, O. I. et al. Comprehensive review of artificial neural network applications to pattern recognition. IEEE Access 7, 158862–158864 (2019).
79. Alzubaidi, L. et al. Review of deep learning: Concepts, CNN architectures, challenges, applications, future directions. Journal of Big Data 8, 1–74 (2021).
80. Samek, W., Montavon, G., Lapuschkin, S., Anders, C. J. & Müller, K.-R. Explaining deep neural networks and beyond: A review of methods and applications. Proceedings of the IEEE 109, 247–278 (2021).
81. Baldominos, A., Saez, Y. & Isasi, P. A survey of handwritten character recognition with mnist and ennist. Applied Sciences 9, 3169 (2019).
82. Emment-Streib, F., Yang, Z., Feng, H., Tripathi, S. & Dehmer, M. An introductory review of deep learning for prediction models with big data. Frontiers in Artificial Intelligence 4, 2 (2020).
83. Pedamonti, D. Comparison of non-linear activation functions for deep neural networks on MNIST classification task. arXiv preprint arXiv:1804.02763 (2018).
84. Saravanan, K. & Sasithra, S. Review on classification based on artificial neural networks. Int J Ambient Syst Appl 2, 11–18 (2014).
85. Abadi, M. et al. Tensorflow: Large-scale machine learning on heterogeneous distributed systems. arXiv preprint arXiv:1603.04467 (2016).
86. Diba, A. et al. Temporal 3d convnets: New architecture and transfer learning for video classification. arXiv preprint arXiv:1711.08200 (2017).
87. Jegede, K., Agrawal, R., Yao, Y. & Fletcher, C. W. Morph: Flexible acceleration for 3d cnn-based video understanding. in 2018 51st Annual IEEE/ACM International Symposium on Microarchitecture (MICRO) 953–964 (IEEE, 2018).
Figure S1. Training (left) and validation (right) losses for different fractions of the total dataset used for training the Siamese networks (legend) for the agent-based model (top), flux model (middle), and differential equation model (bottom).
Siamese neural networks for comparing model outputs

Figure S2. Training examples for the three models, looking at four different numbers in the fully-connected layers for each (red [1024 -> 256], green [1024 -> 1024], blue [4096 -> 1024], black [4096 -> 4096]).