Transparency in Complex Computational Systems

forthcoming, *Philosophy of Science*

Kathleen A. Creel
kac284@pitt.edu

Abstract:

Scientists depend on complex computational systems that are often ineliminably opaque, to the detriment of our ability to give scientific explanations and detect artifacts. Some philosophers have suggested treating opaque systems instrumentally, but computer scientists developing strategies for increasing transparency are correct in finding this unsatisfying. Instead, I propose an analysis of transparency as having three forms: transparency of the algorithm, the realization of the algorithm in code, and the way that code is run on particular hardware and data. This targets the transparency most useful for a task, avoiding instrumentalism by providing partial transparency when full transparency is impossible.

Transparency in Complex Computational Systems

1. Introduction

Scientists depend on complex computational systems to process their big data, but these systems are not always transparent. Physicists within the Large Hadron Collider's Compact Muon Solenoid working group are considering using deep learning algorithms to sort particle collision events and discard the uninteresting ones (Duarte et al. 2018). The new algorithms for doing so, while faster than the old, are complex enough that their decisions cannot be reconstructed in terms of why some events were interesting and thus saved, and why others were discarded (Roxlo and Reece 2018, 2). Some physicists, who want more transparent sorting procedures so they can check and understand the algorithms responsible for discarding all but 100 of the 600 million particle collision events detected each second, have called this change "a nightmare" (Castelvecchi 2015). They are also concerned by the limitations of proposed deep neural network classification algorithms for scientific explanations: "[u]nfortunately, as written such a classifier would probably be of limited usefulness in terms of providing insight into new physics" (Roxlo and Reece 2018, 2). Given that the scientists running computational projects want their systems to be more transparent, it is crucial to have a clearer analysis and definition of computational opacity. Such an analysis should provide the users of computational systems with tools to seek the kinds of epistemic goods they desire for their diverse ends, such as using the system for prediction, employing it to explain scientific phenomena, or improving and maintaining the computational system itself.

Existing analyses of transparency in science target complex climate models

because their complexity and opacity contribute to public doubt about the predictions of the models.[1] But computational opacity extends beyond climate models.[2] Whether using machine learning at the LHC, artificial neural networks for cancer detection, or visual recognition software to identify fossilized pollens, scientists have capitalized on advances in algorithms and computing power without addressing the epistemic problems that can accompany such advances (Tcheng et al. 2016; Esteva et al. 2017). They are now searching for ways to make opaque computation more transparent in order to better detect errors and provide scientific explanations.

This paper argues for the need for transparency in computational systems (§2) and claims that transparency comes in three forms. The first is functional transparency, or knowledge of the algorithmic functioning of the whole (§3a); the second is structural transparency, or knowledge of how the algorithm was realized in code (§3b); and the third is run transparency, or knowledge of the program as it was actually run in a particular instance, including the hardware and input data used (§3c). Identifying three forms of transparency, as compared with recent papers that each identify one, is necessary to explain recent successes in reducing opacity (§4). It also gives those attempting to increase transparency in future systems an analytical tool with which to identify the type of opacity to eliminate (§5).

---

[1] The analyses of transparency in (Humphreys 2004; 2009; Lenhard and Winsberg 2010) will be discussed in §3. For further epistemic problems caused by opaque models, see (Winsberg 2010; Lenhard 2006).

[2] In this paper, opacity and transparency are two sides of the same coin: opacity is a lack of transparency and vice versa.

2. Why Transparency?

Should we strive for transparency in complex computation? In this section I claim that we should because scientists, modelers, and the public all require transparency and because it facilitates scientific explanation and artifact detection. Lenhard and Winsberg, by contrast, claim that "it is impossible … by any method" to gain the form of transparency they identify in some complex climate models, so we ought not to seek it (Lenhard and Winsberg 2010, 254). Likewise Paul Humphreys has argued that when opacity is ineliminable, "we must abandon the insistence on epistemic transparency for computational science," focusing instead on improving models by using "trial-and-error procedures treating the connections between the computational template and its solutions as a black box" (Humphreys 2004, 150). The record of success of any computational system will certainly be important in evaluating it. However, as philosophers, we should not recommend abandoning the search for transparency, for two reasons. First, descriptively, transparency is important to all of the groups that create and use computational systems. Second, normatively, these groups are justified in caring about transparency.

Descriptively, the scientists who use the systems to investigate, the modelers and computer scientists who create the systems, and the non-scientist citizens who interact with or are affected by the systems all need transparency. As large-scale computation becomes vital to many scientific disciplines, scientists express dissatisfaction with the limited transparency it affords, especially when the new computational methods seem less transparent than previous methods for performing the same operation on smaller data

sets, as seen in the example above of the LHC.

Transparency in computational systems also matters to computer scientists. The machine learning community debates the topics of transparency, opacity, and interpretability, with over 20,000 papers on arXiv and multiple yearly workshops such as Fairness, Accountability, and Transparency (ACM FAT*). While attempts to increase transparency burgeon, analysis of what would be required to achieve transparency remains implicit or only briefly mentioned in most machine learning papers. The disagreement in implicit or explicit definitions of transparency makes it more difficult for researchers to progress (Sørmo, Cassens, and Aamodt 2005).[3]

Lack of transparency in computational systems also concerns the public. Non-specialists frequently decline to use statistical and computational tools that function as "black boxes," such as the many systems designed to help doctors diagnose patients and evaluate their risk of disease (Miotto et al. 2016). These systems use decades of patient records, including biographical and medical information, to learn to diagnose current conditions and predict the probability of future ones. Although neural networks like Weng et. al. (2017) successfully deliver diagnoses or predictions, the teams often can give no reasons for their neural networks' decisions. Lack of explanation for the diagnosis given is cited as a cause of doctors' lack of trust in and therefore failure to use

[3] Philosophers who consider the term "transparency" to be misleadingly suggestive of an overly high standard for success, such as Sabina Leonelli, still consider questions similar to the ones discussed in this paper, about good epistemic practices for data, algorithm, and information disclosure in contexts of science and public policy, albeit under different concept terms (Leonelli 2016; Leonelli, Rappert, and Davies 2017).

devices that assist medical diagnoses even when these devices have been shown to decrease error (Ribeiro, Singh, and Guestrin 2016).

As complex computational systems enter the public sphere, citizens and lawmakers need explanations of outputs that these systems are too opaque to give. For example, the European Union's General Data Protection Regulation, which went into effect May 25, 2018, includes a "right to explanation" clause which allows any EU citizen affected by "automated individual decision-making, including profiling" to request and receive an explanation for the basis of that decision (Goodman and Flaxman 2017). In order to comply with similar regulations, there must be a standard of transparency for the tools being used (Datta, Tschantz, and Datta 2015, 106). In sum, one reason for philosophers to pursue an analysis of transparency is that research scientists, computer scientists, and the public all value transparency and could benefit from a better analysis of the concept.

Not only do interested parties want transparency, they are right to want it. Transparency is necessary to illuminate the relationship between the explanans and the explanandum on some leading accounts of scientific explanation. Mechanistic explanations, for example, require transparent access to the relationships between constituent parts. By its nature, opacity makes it difficult to identify "entities and activities organized such that they are productive of regular changes from start or set-up to finish or termination conditions" (Machamer, Darden, and Craver 2000, 3). The same holds true for the identification of "interactions between parts … characterized by direct, invariant, change-relating generalizations" (Glennan 2002, S344). Although many types of mechanistic explanation exist, all require the identification of parts and the causal

relationships among parts. Thus if identifying mechanisms is required for good causal explanations, some degree of transparency will be also. As illustrated by this example, access to only the observable inputs and outputs of a completely opaque black-box system is not a sufficient basis for explanation on some leading accounts of scientific explanation.

An additional advantage of transparency is that it makes artifacts easier to detect. In an engineered system like a computer, an artifact is a problem in functioning, a result or behavior that the designers did not expect or want. Since often there is no non-computational procedure to verify the results of a computation or model, transparency provides an approach to check the system for such unwelcome behavior. Strategies to increase transparency of computation for the sake of troubleshooting have been deployed since at least the 1960s, when computer scientists used visualization and dynamically generated diagrams to make clearer the inner workings of programs and to debug them (Ananny and Crawford 2016, 5). Gaining one or more of the three transparencies below increases the ability to detect artifacts by granting access to the process by which the result was produced.

Providing transparency to scientists, modelers, and citizens requires overcoming the barriers to transparency identified by Humphreys, Winsberg, and others. It is both premature and unnecessary to "abandon the insistence on epistemic transparency for computational science" (Humphreys 2004, 150). Instead, we need an analysis of transparency that captures more ways that systems can and should be transparent. In the next section, I argue that a tripartite analysis of transparency overcomes the barriers identified by Humphries, Lenhard, and Winsberg.

3. The Three Varieties of Transparency

My analysis focuses on the opacity that is endemic to complex computational systems and the kinds of transparency that can ameliorate it. I argue that there are three forms of transparency corresponding to different granular scales of a computational system: algorithmic transparency, structural transparency, and run transparency. Although an intervention to increase one form of transparency will sometimes increase another, there is not a necessary connection between the forms. There are many systems in which only one form of transparency is present or achievable, and the systems developed by computer scientists to enhance transparency discussed in §4 each address only one of these three types. In this section, I will first distinguish the kinds of transparency I am interested in from other common uses of the word in science and politics; then describe the three kinds of transparency; and finally relate the three to one another.

Opacity can occur when a government or company conceals information or when scientists do not reveal their methodology or code in sufficient detail (Fink 2017). It can also occur when information is shared with users who cannot understand it, as can occur when scientists collaborate across disciplines or use off-the-shelf software packages. This paper will not address these opacities. It will focus only on opacity that vexes skilled and knowledgeable creators and users of computational systems.

Although the information needed to produce output is present in complex computational systems, it is not always in the form of reasons that a human, no matter how skilled, can understand as an explanation. For machine learning, this opacity "stems

from the mismatch between mathematical optimization in high-dimensionality [which is] characteristic of machine learning and the demands of human-scale reasoning and styles of semantic interpretation" (Burrell 2016, 1–2). The next three subsections will describe three forms of transparency that alleviate opacity in computational systems. These are the transparency of the system's algorithm, the way that algorithm is represented in code, and the way that code is instantiated and run on particular hardware and data.[4]

3a. Functional Transparency: Algorithmic Knowledge

The first form of transparency is functional transparency, or knowledge of the algorithmic functioning of the whole. When a computational system is functionally transparent, it is possible to know the high-level, logical rules according to which the system will transform a given input into an output. In other words, to have functional transparency is to be able to know which algorithm the program instantiates. Since "functional" can mean many things, it is worth clarifying that my functional transparency is not an analysis of a system in terms of the relations of its constituent parts; I will call that structural transparency and discuss it in the next section. Structural transparency is

[4] My three forms of transparency draw on the concepts in classical computation from which David Marr developed his three levels of understanding in information processing systems. My first and third forms of transparency are similar to his second level, "representation and algorithm," and third level, "hardware implementation," respectively. Marr's levels are meant to aid understanding of perceptual information processing in vision. They do not address transparency or scientific computation. For more on Marr's levels, see (Marr 2010, 24–27).

knowing how the program instantiates a particular algorithm. Functional transparency, conversely, is knowing which algorithm the program instantiates.

An algorithm is an abstract mathematical object. A computer program is a particular instantiation of an algorithm (Knuth 1977, 63). Because an algorithm can be multiply realized in code, knowing the algorithm does not entail knowing the parts of a program or the relations between its parts. Programs that successfully carry out the same algorithm can be composed of different arrangements of parts, especially if they are written in different types of programing languages, whether procedural, functional, object-oriented, or assembly.

A program generated by machine learning may fail to be functionally transparent if its algorithm was developed autonomously, without being programmed by a person, as in genetic programming.[5] Genetic programming loosely mimics the process of mutation and selection present in populations of natural organisms. The process begins with a large pool of "chromosomes," or strings comprising small possible subsets of the program, which are aggregated into programs and run on the task set by the human programmer. After each competition, the "fittest" programs are evolved in two ways: parts of their "chromosomes" are randomly swapped with those of other fit programs and the chromosomes themselves are randomly mutated. In each evolution phase the learning algorithm modifies the operations of the programs themselves, significantly changing the code of each program as it improves. Since such generated algorithms are often complex, long, and unintuitive, it can be difficult for humans to understand the final algorithm. We can know the class of algorithms into which it falls without knowing the algorithm itself.

[5] For a philosophical discussion of genetic programming, see (Clark 2013, 98).

In cases without machine learning, a program is typically written by humans in order to instantiate a particular algorithm. Even when human programmers are competent, however, programs may still be functionally opaque. One illustration of this occurrence is when the program involves substantial kludging, as discussed by Lenhard and Winsberg (2010) in climate models.

Climate models are often built from many sub-models such as models of atmospheric circulation, temperature over time, cloud albedo, ocean circulation, behaviors of sea and land ice, carbon-capturing properties of vegetation, and more. Creating a climate model that unites these models and their heterogeneous sources of data involves a process of accretion: new subcomponents are added and the model adjusted to incorporate them.

Over time, this accretion of data and sub-modules requires increasing amounts of "kludging." A kludge is a software fix made locally that is theoretically unmotivated: the addition of new bits of code that make the model work but not for any principled reason. Kludging creates a "piece of program or machinery which works up to a point but is very complex, unprincipled in its design, ill-understood, hard to prove complete or sound and therefore having unknown limitations, and hard to maintain or extend'' (Lenhard and Winsberg 2010, 257).

Since kludges lack theoretical motivation, they are harder to understand as part of a model and are not related to the model's algorithmic functioning. One cannot say about a kludge, for example, that it improves the accuracy of a cloud behavior module within the larger climate model by better modeling raindrop nucleation site dynamics, or that it instantiates a linear regression model (Khain, Rosenfeld, and Pokrovsky 2005). The

kludge exists only to make what was broken functional. It is the patch on the rubber tire.

In my taxonomy, kludging impedes functional transparency. A climate model might appear to have a tidy, high-level algorithm and be functionally transparent, but its kludges introduce disorder which, though essential for its functioning, may alter the nature of its algorithm. Lenhard and Winsberg think that by eliminating this kind of opacity they will gain the ability to detect artifacts, which they see as the purpose of transparency. They are correct that kludging leads to opacity and artifacts, and "should be expected to continue" to do so – opaqueness is not always eliminable (2010, 258). But as I argue in §4, kludging cannot fully capture the problem of opacity, nor is functional transparency always the right kind of transparency to enable detection of artifacts.[6]

3b. Structural Transparency: Process Knowledge

The second form of transparency is structural transparency, or knowledge of how an algorithm is realized in code. As mentioned in §2a, the same algorithm can be multiply realized in code: written in different programming languages, or written in ways

[6] For machine learning in particular, the scope of application of functional transparency requires clarification. There are (at least) two algorithms in a machine learning system: the algorithm that guides the learning process, and the algorithm, model, or decision procedure that is learned. Lipton (2016) describes transparency of the former as algorithmic transparency, but both versions are extant and sought in the computer science literature. Functional transparency therefore refers to both, but I encourage users of this taxonomy to add further classifications within each of the three types when doing so can help precisify the form of transparency delivered by a particular system.

different enough to change the operation of the code. It is possible to know the algorithm that the code realizes but not know how the code realizes it, i.e. to have functional but not structural transparency. To have structural transparency is not just to be able to read the code; it is to understand how the code as written brings about the result of the program.

One path towards structural transparency is through modular decomposition, part of Paul Humpreys' suggested definition of transparency. On his view, complex climate models are structurally transparent when one can decompose them into modular steps, "each of which is methodologically acceptable both individually and in combination with others" (Humphreys 2004, 160).[7] This form of transparency relies, first, on being able to decompose the process of the model and second, on being able to understand the dynamic flow from step to step.

However, there are two problems with the details of Humphreys's account. First, he describes opacity as occurring when models are "too fast for humans to follow in detail," or "so fast and so complex that no human or group of humans can in practice reproduce or understand the process" (Humphreys 2004, 150; 2009, 619). Speed, however, is not the operative concept. Neither Humphreys nor others in the discussion of

[7] In later work, Humphreys relaxes the definition of opacity to "a process is epistemically opaque relative to a cognitive agent X at time t just in case X does not know at t all of the epistemically relevant elements of the process" (Humphreys 2009, 618). Although his 2009 definition appears to require only the static identification of relevant elements, Humphreys's descriptions of stepping through a process between input and output and identifying steps of a procedure remains the same in the explanation of the definition. Therefore, I will take the two definitions to be equivalent.

opacity argue that transparency must occur in real time, and they ought not. Even the simplest calculation on a modern computer occurs "too fast for humans to follow in detail," but it can be understood immediately after its completion by examining the code and the statements left behind in the logs of that computational run.

Humphreys might agree that real-time tracing is not at the heart of transparency. He suggests later that transparency can be achieved by process decomposition, a step-by-step tracing of "details of the process between input and output" (2004, 150). He writes, "If we think in terms of such a process and imagine that its stepwise computation was slowed down to the point where, in principle, a human could examine each step in the process, the computationally irreducible process would become epistemically transparent" (ibid.). But realizing Humphreys' counterfactual would not necessarily yield structural transparency. In principle, we could print each of the thousands of base-cases solved in a computer-aided proof and check the code-as-run line by line. However, a human manually examining each step in the process could, for some long or complex programs, require longer than a human lifetime. Although Humphreys is correct that each individual step could be transparent, and that "in principle" the whole would thus be transparent, idealizing away from the duration of time in this way means that any program of a sufficient complexity is in practice opaque to humans and likely to remain so. The size and complexity of the program make the checking infeasible, not the initial speed of its run.

The gap between principle and practice typifies the larger problems with Humphrey's process decomposition as a generalizable analysis of computational opacity. In a bare sense, a computer program will always be process decomposable, since the path

of execution can always be traced. Most code written in object-oriented programming languages comes pre-decomposed into modules: the functional units of the program, called "functions" or "methods," each of which performs a specific task (eg, matrix multiplication), guarantees a type of output (a matrix) if given a correct input (two or more matrices), and has an inheritance relationship to other functions (matrix multiplication "inherits" some of its functionality from multiplication; it could be extended for use in sub-types of matrix multiplication). Knowing the modular steps at the function level is often possible.

It may nevertheless be the case that "most steps in the process are not open to direct inspection and verification," because the "steps" are higher-level steps (ibid., 147). The difficulty in determining which emergent properties, complexity effects, or unexpected errors the code might possess cannot always be resolved by being able to trace the code line-by-line or function-by-function. Only at a higher level does structural opacity emerge. These are "computationally irreducible processes, processes that are of the kind best described, in David Marr's well-known classification, by Type 2 theories … in which a problem is solved by the simultaneous action of a considerable number of processes whose interaction is its own simplest description" (ibid., 148–9). Thus, a second source of opacity would remain even if hand checking were possible: the program would still not be fully transparent because we could not understand how all the steps interact to bring about the algorithm or to what extent each individual step contributed to the final result.

A form of transparency similar to Humphreys's process decomposition evades the first part of this critique by instituting a reasonability criterion on the time given to

stepping through the code. Zachary Lipton says that a model is simulatable if "a human [can] take input data together with the parameters of the model and in *reasonable* time step through every calculation required to produce a prediction" (Lipton 2016, 5).[8] This improves on Humphreys's definition in two ways: first, by stipulating that it must be done in reasonable time and second by stipulating that the unit of step-wise evaluation is the calculation, thereby abstracting away the necessity of line-by-line evaluation.

The problem with both Lipton and Humphrey's definitions is that successfully stepping through a process linking *one* input with *one* output is not enough to call the whole process or model, respectively, transparent. There are many cases in which understanding the sub-components of a whole model might be prohibitively complex while a few of the model's input-output paths remain easily understandable. While knowing these paths would improve our epistemic standing, this would be a weaker form of transparency, one in which the transparency of the whole system would depend on the luck of the simplest path. Would a model count as transparent for which only one path could be traced, such as the failure case that immediately produced an error message? Separating the transparency of individual paths or individual predictions from the transparency of the model or algorithm as a whole, as some analyses do, is reasonable, but one cannot stand in for the other (Ribeiro, Singh, and Guestrin 2016).

Instead, Humphreys and Lipton might require that a human be able to step

[8] Lipton glosses this form of transparency as being "able to contemplate the entire model at once," but as I have suggested in §3a, this is a meaningfully different form of transparency.

through *every possible* path of the program.[9] Although it would more appropriately model the transparency of the system as a whole, this analysis still relies on the sum of all the tokens to stand in for the type. The ability to wander down every possible path of the code, or to know that you have done so, requires having my form of structural transparency. It requires knowledge of the code as it instantiates the algorithm, sometimes represented with a code map or architecture diagram. This model could be a simple box and line code map; it could be a Unified Modeling Language (UML) software architecture diagram complete with structure diagrams, behavior diagrams, and interaction diagrams. Such a model visually represents the structure and behavior of a system by decomposing it into its parts and by representing the ways in which the interactions of those parts support the algorithm.

Without such knowledge of the sub-components and their relations, it would be difficult to successfully trace the path between every possible input and its output. Since there are an infinite number of possible inputs, the possibility of predicting the behavior of the system, and especially of identifying the sources of unexpected behavior in the code, requires a model of the system and how it will treat inputs.[10] Therefore the more complete form of analysis is the knowledge of relations between the subcomponents, not

---

[9] Lipton may already intend to require this; his phrasing is ambiguous between the two possibilities.

[10] I say "possibility" to suggest that in some cases, it will not be possible to predict the output of the system without completing its computation, such as in irreducible computations, or at all, as suggested by the undecidability of the halting problem.

the sum of the ability to traverse all possible paths.[11]

A further question concerns the "level" at which the algorithm is instantiated in code. The purpose of higher level programming languages is to hide the machine code that implements each command behind a language that is easier to read and write. Thus the algorithm is almost always implemented in multiple languages. The differences between these two languages can itself lead to opacity, for predictable reasons: the implementation of the programming language has been intentionally concealed in order to provide a seamless experience for the programmer.

However, because of this opacity the programmer can make false assumptions about how the language will interpret commands. For example, Python has built-in functions for integration (Scipy Community 2017). These functions accept many data types in addition to integers, but may not treat them in the way that the programmer expects unless she researches the specifications of the functions in the Python documentation or writes her own integration functions. The silent error that occurs when an integration function accepts data and returns a number but in fact is not performing the calculation as the programmer expects can be difficult to find when tracing the code the

[11] Structural transparency also incorporates a second aspect of Lipton's characterization of transparency, which he calls "decomposability": that "each part of the model – each input, parameter, and calculation – admits an intuitive explanation" (Lipton 2016, 5). Again, Lipton's characterization is somewhat ambiguous as to token and type. It also does not specify what kind of explanation suffices – a literature unto itself in both philosophy of science and explainable AI. I consider structural transparency to have captured the best features of both of his characterizations.

programmer has written. The error exists in the clash between the programmer's expectations and the integration function written by the developers of the programming language.

Silent errors can persist even in popular scientific software. For example, scientists working on the Dark Energy Survey implemented the same correlation function model using two standard cosmology software packages, CLASS and CAMB, one written in C, the other implemented in Fortran but controllable with a Python interface.[12] By comparing the results produced by the two software packages, they "uncovered actual coding errors; to reach the final level of agreement, further iterations required validation of numerical implementation details, such as integration accuracy" (Krause et al. 2017, 7). Before scientists compared CLASS and CAMB, the cosmology community had been unaware of the integration errors.

The structural opacity of a large cosmology or climate simulation is an accidental function of size and accretion; a similar but smaller program would be easier to understand. Nevertheless, such opacity can be difficult to reduce. A code base of sufficient complexity, such as a climate model or a cosmology simulation which itself relies on a large software package like CLASS or CAMB, may be difficult to map accurately in its entirety due to the complexity of interactions between its parts and its pre-written software functions.

Conversely, when the functional units of the program are tiny, simple, and

[12] For general discussion see (Krause et al. 2017, 6). For more on the implementation of CLASS see (Lesgourgues and Tram 2017); for CAMB see (Lewis 2017; Lewis and Challinor 2017).

numerous, as are the neurons of a deep neural network, a subcomponent map would prove insufficient. To see why, imagine a simple neural network: a perceptron (Rosenblatt 1958). A perceptron consists of binary input "neurons" connected to an output neuron which functions as a classifier. The original perceptron returns one if a weighted sum of its inputs is over a certain threshold, and zero otherwise; others have nonlinear activation functions and can use adjustable real numbers as their activation values. Perceptrons can be stacked to form a larger neural network or used for simple logical operations such as AND or OR.[13]

Because perceptrons are simple, eschewing modern feedback techniques, we have formal guarantees concerning their performance and learning capacities (Harman and Kulkarni 2011). However, such results rarely apply to hybrid deep neural nets of the kind used in most scientific computing and industry applications. Contemporary deep neural networks, while stemming from their simpler ancestors, differ in at least four important ways, as summarized by Buckner (2019): they contain many more layers, increasing their efficiency; they are composed of heterogeneous processing units with different activation functions; they are sparsely connected; and they use more techniques to avoid overfitting. These added complexities increase both performance and opacity.

With access to the input data and to the current weights of the network, it would be possible to predict or trace all of the thousands of neurons' responses to a new image through the layers of the network and thus to predict final classification of the image. In this sense, at least for smaller neural networks, it is possible to know how the algorithm is

[13] I rely here on the original description of a perceptron, but note that "multilayer perceptron" is sometimes used now to describe a deep, fully-connected neural network.

instantiated in code.

However, in all but the smallest networks it would be difficult to predict the outcome without tracing each step or to understand the behavior of the network, especially if the network includes feedback loops. More importantly, without further analysis it would be unclear to the observer *why* this neural net successfully classified an image and to what extent each of the neurons contributed to the result, or why different neural nets might have different patterns of classification. In this sense, although we know how the learning algorithm works and what formal guarantees (if any) we have about its performance, we do not know how the learned "algorithm" brings about the classification result. Thus we lack functional transparency.

3c. Run Transparency: System Knowledge

The third form is run transparency, or knowledge of the program as it was run in a particular instance, including the hardware and input data used. While functional transparency can often be analyzed by surveying the text of the code, run transparency requires analysis of one particular run on an individual machine using actual data. Achieving such transparency will allow the detection of artifacts caused by interaction effects between the program and hardware; between the program and unexpected input data; and between the program and its implementation in a particular programming language that is then translated into the machine code that actually runs. I will illustrate run opacity using two problems obscured by it: one in hardware and one in the input data.

One hardware problem that run transparency illuminates is the corruption of sensitive detector equipment by cosmic rays. When balloon-borne telescopes are

launched to measure cosmic microwave background radiation, powerful cosmic rays flip the telescopes' bits and corrupt their data. Therefore, balloon experiments use special "space-qualified" hardware: circuits and logic gates that are less susceptible to bit flipping and can detect and repair the flips that do occur (Dobbs et al. 2009, 14). Changing the system level implementation and hardware components protects the system from errors that would be difficult to detect with access only to the algorithmic or structural levels. Knowledge about the hardware and the state of the data stored in a particular run is necessary in order to pinpoint the bit-flipping errors generated by cosmic rays.

Second, features of the data for which the programmers did not account can also interact with the program to create artifacts. Although the problem may be located in the data, in an opaque system it is more difficult to detect. Increasing run transparency can reveal previously undetected problems in the interactions between input data and the software.

For example, consider the use of algorithmically generated "risk scores" to predict the probability of recidivism in bond assignments and sentencing decisions. These risk scores often rely on data that are biased due to their method of collection, as when they use the number of prior arrests to determine probability of re-offending (Kirchner 2016). In a country in which probability of arrest and re-arrest given the same crime(s) is racially skewed, this initial data introduces a racial bias into sentencing (Chouldechova 2017). This is a classic garbage-in, garbage-out data problem, not unique to opaque algorithms, but one difficult for defendants to identify due to lack of transparency. It also cannot be identified using functional transparency or structural transparency, in other

words by looking at the algorithm or the code itself. Knowing the way the data was collected, its distributional features, and the way it is used by the software is required.

3d. Relationship of the Three Forms of Transparency

The three forms of transparency – functional transparency, structural transparency, and run transparency – each individually improve our knowledge. Having all three provides more transparent understanding than one alone; together, they divide types of knowledge of a computational system. Although other divisions are possible, such as subdivisions within these three types or orthogonal divisions of the logical space, I will in the next section motivate the value of this division.

All three types of transparency are dissociable: each can be exhibited without requiring any of the others. This independence conflicts with existing philosophical accounts, according to which possession of one type of transparency necessarily affords another (Humphreys 2004; Lenhard and Winsberg 2010). On Humphreys' view, possessing structural transparency provides high-level algorithmic, functional transparency. Complex climate models become transparent for Humphreys when one can decompose them into modular steps, "each of which is methodologically acceptable both individually and in combination with others" (2004, 160). This is a partial form of structural transparency.

Once attained, according to Humphreys, structural transparency will provide a high level understanding or an "explicit algorithm," a functional transparency, in my account (2004, 149). However, the argument in §2 regarding the ease and shallowness of obtaining a line-by-line or function-by-function decomposition of any program shows

that the process Humphreys describes will not always lead to functional transparency or to understanding the algorithm. Humphreys does not get two transparencies for the price of one.

The other analysis of transparency in the climate modeling literature comes from Lenhard and Winsberg. Lenhard and Winsberg's ability to localize success or failure to a specific subcomponent of the program is a version of my structural transparency. For them, models are transparent when one can identify the extent to which each of the sub-models is contributing to the success of the model. One increases transparency by "the process of teasing apart the sources of success and failure of a simulation ... we would say that one has such understanding precisely when one is able to identify the extent to which each of the sub-models of a global model is contributing to its various successes and failures" (2010, 258). This criterion for transparency differs from Humphreys' because it is not necessary to know how the sub-components relate to one another, as Humphreys requires. Transparency in Lenhard and Winsberg's view does not require a full trace of the process.

Although helpful, this ability to localize success or failure will neither entail full transparency nor provide a useful kind of transparency in all circumstances. For an example of why localizing success and failure will not lead to Lenhard and Winsberg's desired result of transparency, consider a common problem with models in the social sciences, here illustrated with a model from political science. Ray Fair's (1978) classic model of democratic functioning suggests that a strong economy so outweighs other relevant causal factors that if the economy is strong, the party in power will be reelected. The subsequent literature discussing this phenomenon argues that while the correlation is

fairly robust, it cannot be considered a satisfying theory until the explanations for and processes underlying the correlation are made clear. Without those causal connections, the model remains opaque. The economy-reelection correlation localizes the prediction's success or failure to one particular component of a complex model of factors that affect elections, as Lenhard and Winsberg would desire. However, it merely presents a predictive correlation useful for localizing success or failure but useless for a high-level algorithmic understanding of the effect.

I have shown that transparency can be divided into three types and that these three types dissociate: having one type does not guarantee having any other. Because of this dissociability, each type of transparency can be achieved independently. The question, then, is how such transparency can be achieved. This is the topic of the next section.

4. Solutions for Transparency

Reducing the opacity of complex computation is a burgeoning field of research in computer science. However, the methods researchers use to increase transparency are different than the ones Humphreys or Lenhard and Winsberg would recommend. In this section, I will discuss two existing methods for increasing transparency and show how the three-form analysis of transparency illuminates these cases. The major conclusions are: (1) that all three forms of transparency are required to fully reduce opacity and (2) that focusing only on a single form of transparency nevertheless can suffice for a particular goal.

4a. Post-hoc Explanation and LIME

The first method for increasing transparency in an existing computational system is to create an algorithm that generates post-hoc decision explanations. One such algorithm, LIME, was created by Ribeiro et. al. (2016) to reduce the opacity of existing machine learning classifiers. LIME, or Local Interpretable Model-agnostic Explanations, aims to explain the predictions of a classifier by fitting a linear model to the pattern of its prediction given the input data.

Although it can be used in any domain, I will describe the functioning of LIME using an example of its application to medical diagnosis. Existing medical diagnostic machine learning systems make diagnoses and treatment recommendations based on information available in patients' records and case notes from the patients' most recent visits (Caruana et al. 2015). Using this information and correlations between medical information and correct diagnoses, the system delivers a diagnosis, such as "flu" or "food poisoning." However, because the diagnostic systems do not give an explanation or reason for the diagnosis, doctors often deem them untrustworthy and avoid them. Applying LIME to such a system gives doctors a rationale for the existing system's diagnoses.

Because of the nature of the algorithms used, however, the diagnostic systems did not already contain those reasons in human-understandable form. In fact, the diagnostic systems made diagnoses by weighing hundreds or thousands of micro-factors expressed in artificial neurons. LIME attempts to query the decision space, or set of outcomes possible given an input, of the existing program from the outside. It reconstructs the "decisions of any model in a local region near a particular point, by learning a separate

sparse linear model to explain the decisions of the first" model (Lipton 2016). In other words, LIME attempts to decrease opacity by providing a local, post-hoc, high-level explanation. The explanation provided is in a tidy, human-understandable list of three or four most important factors influencing the decision, such as "sneeze, headache, no fatigue => flu" (Ribeiro, Singh, and Guestrin 2016).

While locally descriptive of the model, LIME's list does not demystify the overall decision space used to make the flu diagnosis. LIME offers a high-level explanation for the decision-making, a type of functional transparency. Although it does not provide the whole algorithm, it provides the parts of the algorithm used to make the particular decision. LIME does not improve structural or run transparency. It provides neither the structural components used to implement the decision-making process nor an accurate low-level depiction of the system's functioning.

Nevertheless, such outside in, post-hoc explanation generation increases a type of functional transparency: it provides access to the decision space most relevant to the token classification. In the language of functional transparency, it thus succeeds in explaining the functioning of the algorithm on that particular decision, albeit in coarse-grain, human-interpretable terms. For example, in a text classification task, LIME might select certain words from the input text as most relevant to the classification. Since it increases the usefulness of the classification while providing "faithful" access to features relevant to the decision made by the algorithm, namely words from the input text, it offers a form of transparency.

In only a very loose sense does LIME provides the form of transparency that Humphreys describes. There is a minimal high level explanation of the patient's

complaint, the diagnosis as backed by symptoms, which appears to the doctor to have been generated by a weighting of the "symptoms" that are given as reasons for the diagnosis, thus providing an explanation of the process. The decision space is decomposed into components comprising major symptoms or facts about the patient, such as "runny nose" and "under age 12," that generated the diagnosis. And this description of the model is in some sense accurate; it is based on a sparse linear model that fits the local decision space. But the decision-making process implied by LIME's list of symptoms is not the way that the original system decides. At no point in the original machine learning system's decision making does it build a sparse linear model that provides discrete symptoms. The linear model used by LIME accurately characterizes the relevant space for that decision, but it would not be of much use in detecting problems with the functioning of the original system. Nor does it open the "steps in the process" to "direct inspection and verification" (Humphreys 2004, 148). Humphreys' analysis thus does not explain how LIME could improve transparency.

Humphreys might respond that reporting the steps and structure of the original system would be a better way to increase transparency than LIME is. However, a step-by-step process taken from the original system would be useless to the doctors. It would not provide an explanation for the diagnosis in the same way that the linear model does. While the step-by-step process of functioning would be useful to the makers and maintainers of the original system in detecting some kinds of artifacts, LIME would be useful in detecting others. Since it brings to light the logic of a particular decision, it can detect patterns and curiosities in the correlations the system has learned ("why are toe injuries an important symptom in predicting myopia?"). Therefore, this system does not

fulfill Humphreys' hope that process transparency would lead to functional transparency and be sufficient for artifact detection.

Likewise, LIME would not increase Lenhard and Winsberg's type of transparency because it does not localize success and failure. Although some artifacts might be unearthed if they appeared as improbable symptoms in the system cluster shown by the linear model, LIME does not illuminate the processes that lead those symptoms to be the ones chosen as relevant.

Nevertheless, LIME does increase transparency. It provides faithful access to high-level features of the decision making process that are useful in aiding artifact detection and explanation. Recognizing multiple forms of transparency, each of which is singly capable of providing relevant information, is necessary to explain the usefulness of systems such as LIME.

4b. Feature Reconstruction and Google DeepDream

The second example comes from Google DeepDream, a visualization tool connected to large and sophisticated neural nets trained to detect images. Given millions of pictures, it can learn to recognize dumbbells, Dalmatians, daffodils, and diamonds. However, as with many neural nets, the criteria by which DeepDream's neural net recognizes dogs and the reasons it identifies some pictures as pictures of dogs are unclear. In order to understand these reasons, Google researchers reversed the algorithm (Dosovitskiy and Brox 2016). For each label that DeepDream recognized, they iteratively fed the system a white noise image and asked it to determine which of two random modifications of that image were closer to its understanding of "dumbbell." This

common post-hoc strategy is known as visualization. Using visualization, the researchers made explicit what the program looks for when it looks for dumbbells (Simonyan, Vedaldi, and Zisserman 2014). The fragmented, surrealist pictures produced by visualization contained many dumbbells with different sized weights, but also forearms and biceps hoisting the weights (Mordvintsev, Olah, and Tyka 2015).

Google researchers have used reversal to detect and eliminate artifacts. The fact that the reversed images of dumbbells come with partial images of arms means that arm images are useful in raising the probability that this is an image of a dumbbell and therefore making a successful classification. However, perhaps because of implicit essentialism about categories, the Google researchers interpreted the presence of arms as an artifact: "[T]he network failed to completely distill the essence of a dumbbell. Maybe it's never been shown a dumbbell without an arm holding it. Visualization can help us correct these kinds of training mishaps" (Mordvintsev, Olah, and Tyka 2015). The choice to eliminate the arms shows a commitment to the correctness of the researchers' prior concept of the essence of dumbbell. Despite acknowledging elsewhere that the labels derived are cluster concepts and despite the status of dumbbells as an object created by humans for a purpose, the researchers are unwilling to accept lifting biceps as part of the functional concept of dumbbell.

Their decision illustrates the tension between the kinds of explanations, solutions, and concepts ordinarily generated by machine learning and the human requirements for what counts as a good exemplar of each of these. Recall that LIME's linear model translated between the complex decision function generated by an opaque classifier and the kind of justification that humans find satisfying as a justification for a classification.

In the case of medical diagnosis, this was a short, discrete list of symptoms. In order to produce its own human-satisfying explanation, DeepMind uses visualization in two ways: to transcribe the complex series of features for which the neurons looked for each concept into a human-understandable series of images and also to produce images as guides to circumscribing the scope of the cluster concepts. The detection of artifacts is a normative decision based in part on what kinds of explanations satisfy humans.

Part of the ineliminable opacity of complex computational systems comes from the fact that the factors that influence machine classifications or predictions are of different kinds from those described as reason-giving by humans. Even when information that is sufficient for machine classifications or predictions is available, its difference in kind and scale means that it explains little to us. This difference in kind leaves human scientists and researchers to create post-hoc explanations that will satisfy humans either by adding a further layer of abstraction to our understanding by reverse-engineering a model's local decisions, as LIME does, or by attempting to make understandable the computer's own decision-making criteria, as Google DeepDream does with visualization. The former wraps an ersatz explanation around the true functioning of the program, satisfying one of the criteria by obscuring the others. The latter, while it allows for artifact detection of a kind, obscures the algorithmic process by which the images are made.

Thus visualization gives us a loose version of what Lenhard and Winsberg desire as a result of transparency without employing their mechanism for achieving it. By supporting a high-level understanding of which features the algorithm is using to decide whether an image should be classified with a label, visualization delivers the capacity to

detect artifacts. However, it does not localize the detected artifact to a particular sub-model. Each of the artificial neurons that (metaphorically) fire differentially when presented with the image of the dumbbell is looking for slightly different sub-components or aspects of the definition of dumbbell. Although we can know which neurons are firing when presented with an arm, a dumbbell, and an arm with a dumbbell, teasing apart which "subcomponent" should be modified to fix the artifact is difficult due to the fuzzy modularity of neuron groups. Therefore, researchers typically approach problems like this by re-training the learning system, perhaps by stocking the training corpus with arm-less dumbbells, rather than altering the existing program.

Visualization does not increase transparency in a way that Lenhard and Winsberg's definition would suggest, nor does it give us Humphreys' step-by-step tracing of the process. Nevertheless, it allows insight into the decision making of the classifier and the elimination of artifacts. In my taxonomy, it does this by increasing run transparency. Visualization provides knowledge of features of the initial training data and subsequent trained algorithm that would otherwise be difficult to access merely from the output. Without a prior theory of the case, it is unlikely that Google engineers would have discovered the reliance on arms to identify dumbbells from the pattern of successful identification alone. Visualization provided insight into features of the training data set, namely the preponderance of biceps in dumbbell pictures, which influenced the outcome in ways the team deemed deleterious. Thus visualization provided the transparent access to previously unknown features of the training data that allowed the researchers to eliminate an artifact. It increased run transparency.

5. Conclusion

The pessimism about transparency expressed by philosophers interested in modeling is understandable. Complex computation will retain aspects of ineliminable opacity. However, we need not, and ought not, give up transparency. The types of transparency for which Humphreys, Lenhard, and Winsberg hope rely too heavily on aspects of computation that are often either difficult to salvage, like transitory variables, or insufficiently explanatory, like neurons. They conclude, therefore, that we must abandon transparency as an epistemic aim for complex systems. However, widespread efforts in computer science to increase transparency and interpretability confirm that some researchers are not content with black boxes. New methods are proliferating in both commercial and scientific machine learning. To return to the Large Hadron Collider, post-hoc visualization strategies similar to those used by the DeepDream researchers have been used to describe the criteria of otherwise opaque event classification algorithms (Roxlo and Reece 2018). In order to explain why these new methods can ameliorate opacity, we must expand our definition of what counts as transparency. In this broader analysis, gaining any of the three transparencies – functional, structural, or run – can improve our epistemic position by targeting a precise type of opacity relevant to particular tasks or goals.

The tripartite analysis presented in this paper allows the identification of the relevant form of transparency to target for a given epistemic need, and thereby makes it more tractable to improve one form of opacity without having to tackle them all. My analysis provides the epistemic grounding necessary to support further normative work. The issue of transparency in opaque computation systems opens questions for further

exploration, such as the status of opaque systems with respect to justice in civil society or trust in science. Differentiating these three forms of opacity and identifying tools by which targeted transparency improvements can be made lets us address these questions with greater clarity.

Acknowledgements:

I am grateful for helpful comments from and discussions with Holly Andersen, Robert Batterman, Nora Mills Boyd, Liam Kofi Bright, Mazviita Chirimuuta, Roger Creel, Javier Duarte, Mahi Hardalupas, Paul Humphreys, Benjamin Jantzen, Sabina Leonelli, Johannes Lenhard, Jake Levinson, Edouard Machery, Sandra Mitchell, Elinor Nichols, Kathleen Nichols, Aaron Novick, Olivia Ordoñez, William Penn, Rebecca Traber, Porter Williams, Eric Winsberg, and two anonymous reviewers. Thanks also to generous audiences at *Philosophical Perspectives on Data-Intensive Science* in Hannover; *Models and Simulations 8* in Columbia, SC; the *Machine Learning Workshop* in Irvine, CA; and *Science and Art of Simulation IV* in Stuttgart.

References

Ananny, Mike, and Kate Crawford. 2016. "Seeing without Knowing: Limitations of the Transparency Ideal and Its Application to Algorithmic Accountability." *New Media & Society*, December, 1461444816676645. https://doi.org/10.1177/1461444816676645.

Buckner, Cameron. 2019. "Deep Learning: A Philosophical Introduction." *Philosophy Compass* 14 (10): e12625. https://doi.org/10.1111/phc3.12625.

Burrell, Jenna. 2016. "How the Machine 'Thinks': Understanding Opacity in Machine Learning Algorithms." *Big Data & Society* 3 (1). https://doaj.org.

Caruana, Rich, Yin Lou, Johannes Gehrke, Paul Koch, Marc Sturm, and Noemie Elhadad. 2015. "Intelligible Models for HealthCare: Predicting Pneumonia Risk and Hospital 30-Day Readmission." In *Proceedings of the 21th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, 1721–1730. KDD '15. New York, NY, USA: ACM. https://doi.org/10.1145/2783258.2788613.

Castelvecchi, Davide. 2015. "Artificial Intelligence Called in to Tackle LHC Data Deluge." *Nature News* 528 (7580): 18–19. https://doi.org/10.1038/528018a.

Chouldechova, Alexandra. 2017. "Fair Prediction with Disparate Impact: A Study of Bias in Recidivism Prediction Instruments." *Big Data* 5 (2): 153–63. https://doi.org/10.1089/big.2016.0047.

Clark, Andy. 2013. *Mindware*. Oxford University Press.

Datta, Amit, Michael Carl Tschantz, and Anupam Datta. 2015. "Automated Experiments on Ad Privacy Settings." *Proceedings on Privacy Enhancing Technologies* 2015 (1): 92. https://doi.org/10.1515/popets-2015-0007.

Dobbs, Matt, Mark Halpern, Kent D. Irwin, Adrian T. Lee, J.A.B. Mates, and Benjamin Mazin. 2009. "Multiplexed Readout of CMB Polarimeters." *Journal of Physics: Conference Series* 155 (1): 012004.

Dosovitskiy, A., and T. Brox. 2016. "Inverting Visual Representations with Convolutional Networks." In *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 4829–37.

Duarte, Javier, Song Han, Philip Harris, Sergo Jindariani, Edward Kreinar, Benjamin Kreis, Jennifer Ngadiuba, et al. 2018. "Fast Inference of Deep Neural Networks in FPGAs for Particle Physics." *Journal of Instrumentation* 13 (07): P07027.

Esteva, Andre, Brett Kuprel, Roberto A. Novoa, Justin Ko, Susan M. Swetter, Helen M. Blau, and Sebastian Thrun. 2017. "Dermatologist-Level Classification of Skin Cancer with Deep Neural Networks." *Nature* 542 (7639): 115–18. https://doi.org/10.1038/nature21056.

Fair, Ray C. 1978. "The Effect of Economic Events on Votes for President." *The Review of Economics and Statistics* 60 (2): 159–73.

Fink, Katherine. 2017. "Opening the Government's Black Boxes: Freedom of Information and Algorithmic Accountability." *Information, Communication & Society* 0 (0): 1–19. https://doi.org/10.1080/1369118X.2017.1330418.

Glennan, Stuart. 2002. "Rethinking Mechanistic Explanation." *Philosophy of Science* 69 (S3): S342–53. https://doi.org/10.1086/341857.

Goodman, Bryce, and Seth Flaxman. 2017. "EU Regulations on Algorithmic Decision-Making and a 'Right to Explanation.'" *AI Magazine*, Fall 2017.

Harman, Gilbert, and Sanjeev Kulkarni. 2011. *An Elementary Introduction to Statistical Learning Theory*. New York: John Wiley & Sons.

Humphreys, Paul. 2004. *Extending Ourselves: Computational Science, Empiricism, and Scientific Method*. Oxford University Press.

———. 2009. "The Philosophical Novelty of Computer Simulation Methods." *Synthese* 169 (3): 615–26.

Khain, A., D. Rosenfeld, and A. Pokrovsky. 2005. "Aerosol Impact on the Dynamics and Microphysics of Deep Convective Clouds." *Quarterly Journal of the Royal Meteorological Society* 131 (611): 2639–63. https://doi.org/10.1256/qj.04.62.

Kirchner, Julia Angwin, Surya Mattu, Jeff Larson, Lauren. 2016. "Machine Bias: There's Software Used Across the Country to Predict Future Criminals. And It's Biased Against Blacks." ProPublica. May 23, 2016. https://www.propublica.org/article/machine-bias-risk-assessments-in-criminal-sentencing.

Knuth, Donald E. 1977. "Algorithms." *Scientific American* 236 (4): 63–81.

Krause, E., T. F. Eifler, J. Zuntz, O. Friedrich, M. A. Troxel, S. Dodelson, J. Blazek, et al. 2017. "Dark Energy Survey Year 1 Results: Multi-Probe Methodology and Simulated Likelihood Analyses." *ArXiv:1706.09359 [Astro-Ph]*, June. http://arxiv.org/abs/1706.09359.

Lenhard, Johannes. 2006. "Surprised by a Nanowire: Simulation, Control, and Understanding." *Philosophy of Science* 73 (5): 605–16. https://doi.org/10.1086/518330.

Lenhard, Johannes, and Eric Winsberg. 2010. "Holism, Entrenchment, and the Future of Climate Model Pluralism." *Studies in History and Philosophy of Modern Physics* 41 (3).

Leonelli, Sabina. 2016. "Locating Ethics in Data Science: Responsibility and Accountability in Global and Distributed Knowledge Production Systems." *Phil. Trans. R. Soc. A* 374 (2083): 20160122.

Leonelli, Sabina, Brian Rappert, and Gail Davies. 2017. *Data Shadows: Knowledge, Openness, and Absence*. SAGE Publications Sage CA: Los Angeles, CA.

Lesgourgues, Julian, and Thomas Tram. 2017. "CLASS: The Cosmic Linear Anisotropy Solving System." Documentation. CLASS: The Cosmic Linear Anisotropy Solving System. March 25, 2017. class-code.net.

Lewis, Antony. 2017. "CAMB Python." Documentation. CAMB Python — Code for Anisotropies in the Microwave Background (CAMB) 0.1.5.3 Documentation. June 8, 2017. http://camb.readthedocs.io/en/latest/.

Lewis, Antony, and Anthony Challinor. 2017. "Code for Anisotropies in the Microwave Background." Documentation. CAMB.Info. January 2017. http://camb.info/.

Lipton, Zachary C. 2016. "The Mythos of Model Interpretability." In *Proceedings of the 2016 ICML Workshop on Human Interpretability in Machine Learning*. New York, NY. http://arxiv.org/abs/1606.03490.

Machamer, Peter, Lindley Darden, and Carl F. Craver. 2000. "Thinking about Mechanisms." *Philosophy of Science* 67 (1): 1–25. https://doi.org/10.1086/392759.

Marr, David. 2010. *Vision: A Computational Investigation into the Human Representation and Processing of Visual Information*. Cambridge, MA and London, England: The MIT Press.

Miotto, Riccardo, Li Li, Brian A. Kidd, and Joel T. Dudley. 2016. "Deep Patient: An Unsupervised Representation to Predict the Future of Patients from the Electronic Health Records." *Scientific Reports* 6 (May): 26094.

Mordvintsev, Alexander, Christopher Olah, and Mike Tyka. 2015. "Inceptionism: Going Deeper into Neural Networks." *Research Blog* (blog). June 17, 2015. https://research.googleblog.com/2015/06/inceptionism-going-deeper-into-neural.html.

Ribeiro, Marco Tulio, Sameer Singh, and Carlos Guestrin. 2016. "'Why Should I Trust You?': Explaining the Predictions of Any Classifier." In *Proceedings of the 22Nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, 1135–1144. KDD '16. New York, NY, USA: ACM. https://doi.org/10.1145/2939672.2939778.

Rosenblatt, Frank. 1958. "The Perceptron: A Probabilistic Model for Information Storage and Organization in the Brain." *Psychological Review* 65 (6): 386.

Roxlo, Thomas, and Matthew Reece. 2018. "Opening the Black Box of Neural Nets: Case Studies in Stop/Top Discrimination." *ArXiv Preprint ArXiv:1804.09278*.

Scipy Community. 2017. "Integration (Scipy.Integrate)." Documentation. SciPy v0.19.1 Reference Guide. June 21, 2017. https://docs.scipy.org/doc/scipy/reference/tutorial/integrate.html.

Simonyan, Karen, Andrea Vedaldi, and Andrew Zisserman. 2014. "Deep inside Convolutional Networks: Visualising Image Classification Models and Saliency Maps." In *International Conference on Learning Representations (ICLR)*. Banff, Canada.

Sørmo, Frode, Jörg Cassens, and Agnar Aamodt. 2005. "Explanation in Case-Based Reasoning–Perspectives and Goals." *Artificial Intelligence Review* 24 (2): 109–43. https://doi.org/10.1007/s10462-005-4607-7.

Tcheng, David K., Ashwin K. Nayak, Charless C. Fowlkes, and Surangi W. Punyasena. 2016. "Visual Recognition Software for Binary Classification and Its Application to Spruce Pollen Identification." *PLOS ONE* 11 (2): e0148879. https://doi.org/10.1371/journal.pone.0148879.

Wagenknecht, Susann. 2014. "Opaque And Translucent Epistemic Dependence In Collaborative Scientific Practice." *Episteme* 11 (4): 475–92. https://doi.org/10.1017/epi.2014.25.

Weng, Stephen F., Jenna Reps, Joe Kai, Jonathan M. Garibaldi, and Nadeem Qureshi. 2017. "Can Machine-Learning Improve Cardiovascular Risk Prediction Using Routine Clinical Data?" *PLOS ONE* 12 (4): e0174944. https://doi.org/10.1371/journal.pone.0174944.

Winsberg, Eric B. 2010. *Science in the Age of Computer Simulation*. The University of Chicago Press.