5 October 2026

Annotated Bibliography on Generalization and Reliability in Machine Learning

Key

Introduction, yellow

Aims and Research Methods, green

Scope, blue

Usefulness, purple

Limitations, red

Conclusions, orange

Reflection, teal

Domingos, Pedro. "A Few Useful Things to Know About Machine Learning." *Communications of the ACM*, vol. 55, no. 10, Oct. 2012, pp. 78-87. *ACM Digital Library*, doi.org/10.1145/2347736.2347755.

Introduction. Domingos, a computer science professor at the University of Washington, wrote this article to pass on twelve lessons that working machine learning researchers know and textbooks tend to leave out. It reads more like advice from a senior teammate than a research paper. Nothing in it is a new result. That is the point.

Aims and Research Methods. The goal is to give readers who already know how a model gets trained the judgment that usually takes years of mistakes to build. The article opens with a simple frame. Every learning method is some mix of a way to represent the model, a way to score it, and a way to search for the best one. The rest covers where projects go wrong, including overfitting and the way theoretical guarantees get misread. There are no experiments.

Domingos draws on published work and the folk knowledge of the field, then explains each lesson with small examples.

Scope. The article focuses on classification, the most common kind of machine learning, where a program learns to sort examples into categories from labeled examples. The variables are the choices a builder controls, such as which features to use, how much data to collect, what type of model to pick, and how to test it. Those are the choices that decide whether a project works, so it makes sense that the author picked them.

Usefulness. I wish I had read this before I started writing vision code. Most of what I learned came from tutorials that show how to train a model, and almost none of them say what to do when the score looks good and the model is still wrong. The lesson that hit hardest was about test data leaking into training. I have changed a setting, checked the score, and changed it again more times than I can count, so my test set stopped being a test a long time ago. That was uncomfortable to read, but I'd rather learn it from a paper than at a competition.

Limitations. The article is an opinion piece backed by experience and a reading of the literature, so most of the advice isn't supported by an experiment run for the article. It was also published in 2012, before deep networks took over image recognition. The lesson that feature engineering matters most reads differently now that the networks I use learn their own features. The short format limits it too. Some lessons get only a few paragraphs, and the one on high-dimensional data deserves a lot more.

Conclusions. My next classifier will start with the simplest model I can find, and it will only get more complicated if the simple one clearly fails. I'm also going to put the test set in a folder I don't open until the end. This paper left me with a question that a later source in this

bibliography takes on directly. Domingos treats overfitting as the central problem, but I still don't understand why a huge network that could memorize its training set often does not.

Reflection. Until now I thought of studying machine learning as collecting techniques, one architecture after another. This article suggests that techniques are the easy part and judgment is the hard part, and I want my ISM work to show some of that judgment. If my final project is only a model with a good score, I will have done what the tutorials taught me. I'd like it to include evidence that I know when not to trust the score.

Henderson, Peter, et al. "Deep Reinforcement Learning That Matters." *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 32, no. 1, 2018, pp. 3207-14. doi.org/10.1609/aaai.v32i1.11694.

Introduction. This article asks whether the progress reported in deep reinforcement learning, where a program learns by trial and error in a simulated environment, can be trusted. Henderson and five coauthors found that the same algorithm can look excellent or ordinary depending on details that papers rarely mention. It works as a warning and as a short list of habits for reporting results honestly.

Aims and Research Methods. The authors wanted to measure how much of a reported result comes from the method and how much comes from luck and setup. They ran well-known algorithms, including TRPO and DDPG, on simulated movement tasks like HalfCheetah and Hopper. Then they varied things that papers often leave out, such as network size, activation function, reward scale, random seed, and the code base that implemented the algorithm. They also compared groups of five runs that differed only in seed. The last section reviews how results get reported and recommends significance tests and full disclosure of settings.

Scope. The study covers continuous control, where an agent learns to move a simulated body, and it looks at a handful of popular algorithms. The variables are the algorithm, the settings, the seed, and the implementation, and the measure is average return across runs. The authors chose these because they are decisions every researcher makes and the ones most likely to go unreported, and that is where unfair comparisons can hide.

Usefulness. This was the most humbling paper on my list. I had assumed that if a result is published, anyone who wants to repeat it just has to follow the steps. The authors show that two groups of five runs of the same code can look like two different algorithms. That changed what I think a single training run means. I run a model once, see a number, and treat it as the answer. It was also a little discouraging, because every number I've shown my team is softer than it looked.

Limitations. The tests cover a small set of simulated tasks and algorithms from 2017, so the specific numbers shouldn't be treated as current. The guidelines are general, so each team still has to decide for itself how many runs are enough. A simulator is also not a real robot, and real hardware adds noise this paper never measures. I don't work in reinforcement learning myself, so I can't check the technical details, only whether the argument holds up for the kind of models I train.

Conclusions. From now on I'll train each model with several seeds and report the average and the spread instead of one number. If the spread is as large as the improvement I was excited about, I should stop calling it an improvement. My new question is how many runs are enough, because every extra run costs training time that a small team doesn't have much of. I also want to see if the spread is bigger for a model that can memorize its data, which connects to the last paper in this bibliography.

Reflection. This paper made me realize I have been acting like a hobbyist about results. I tried things until one worked and then reported that one. A researcher would ask whether it would work again. I want my ISM presentation to say how often my results hold, even if the honest answer is less impressive. It's the kind of claim a mentor can check, and a small claim that survives is worth more to me than a big one that doesn't.

Hendrycks, Dan, and Thomas Dietterich. "Benchmarking Neural Network Robustness to Common Corruptions and Perturbations." *International Conference on Learning Representations*, 2019. *arXiv*, arxiv.org/abs/1903.12261.

Introduction. Hendrycks and Dietterich built a test for something ordinary accuracy scores skip, which is what happens to an image classifier when the pictures get worse. They took the standard ImageNet validation photos and damaged them in fifteen ways, including blur, noise, fog, and compression. Then they measured how much accuracy dropped for a range of popular models.

Aims and Research Methods. To make robustness as easy to measure as accuracy, the authors built two benchmarks. ImageNet-C applies fifteen kinds of damage at five levels of severity and scores a model by its error compared with AlexNet, a measure called mean corruption error. ImageNet-P checks whether predictions stay steady when an image changes only slightly, like consecutive frames in a video. The models were trained on ordinary clean photos and tested on the damaged ones. The authors also tried a few possible fixes, such as bigger networks and multiscale designs.

Scope. This benchmark is about photo classification with the 1,000 ImageNet categories. The damage falls into four families, which are noise, blur, weather, and digital effects. The

authors picked everyday problems like fog, motion blur, and JPEG compression instead of the specially crafted attacks that other robustness papers study, because they wanted to test what a camera would actually run into.

Usefulness. Of everything on this list, this is the closest to my actual problem. A robot camera shakes, and the lights in a competition hall are not the lights in our classroom. Motion blur, brightness, noise, and compression are all on the authors' list, so I finally have names for failures I have only been guessing at. The finding that surprised me is that models got much more accurate on clean images between AlexNet and ResNet, but the amount they lost when images were damaged barely changed. I had assumed a better model would just handle bad images better.

Limitations. The damage is computer generated, so it only stands in for real conditions. The authors keep four extra corruptions aside so that fixes can be checked on damage a model hasn't already seen. The benchmark is also about classification, while my robot has to find an object inside a frame, so I can't assume the same numbers carry over. The paper is from 2019, and training methods built for robustness have improved since then.

Conclusions. The first thing I'll build is a small damaged test set for my own model, using the same kinds of changes. I'll start with blur, brightness, and compression at a few levels and plot how fast accuracy falls. If I try to fix it with augmentation, I need to test on a kind of damage the model wasn't trained on, which is the same contamination problem Domingos describes. My new question is whether a model that handles blur also handles low light, or whether each kind needs its own fix.

Reflection. I used to think of my vision work as pushing one number up on a fixed data set. This paper changes the target to how much a model loses when conditions get worse, and

that's a better description of a competition field. If I'm going to say my project is useful, I should be able to say how it does in the worst light in the building. That is a more honest question for my ISM year than asking for one more percentage point.

Sculley, D., et al. "Hidden Technical Debt in Machine Learning Systems." *Advances in Neural Information Processing Systems*, vol. 28, 2015, pp. 2503-11. *NeurIPS Proceedings*, proceedings.neurips.cc/paper/2015/file/86df7dcfd896fcaf2674f757a2463eba-Paper.pdf.

Introduction. Sculley and nine other Google researchers argue that building a machine learning system is fast and cheap, but keeping one running is not. They borrow technical debt from software engineering, where a shortcut saves time now and charges interest later, and show that machine learning creates debt of its own. The article is more about the software around a model than the model itself.

Aims and Research Methods. The point of the article is to warn teams about maintenance costs that don't show up in a tutorial or a test score. This is an experience paper, not an experiment. The authors go through problems they have seen in real systems and sort them into groups, including boundary erosion, data dependencies, feedback loops, glue code, and configuration. The article ends with a list of questions a team can use to estimate how much debt it already has.

Scope. The authors write about large, long-running production systems, mostly drawn from their own experience at Google, where a prediction model sits in the middle of a lot of other code. What they look at is the parts around the model, such as the data it depends on, the

code that connects it, its settings, and the other systems that quietly use its output. They picked those because that is where they say the cost builds up.

Usefulness. The picture that stuck with me shows the model as a small box in the middle of a much larger diagram of data collection, configuration, monitoring, and serving. The authors say a mature system may end up being at most 5 percent machine learning code and at least 95 percent glue code. I'd never thought of glue as its own category. Mine is a pile of scripts that resize images, rename files, and feed a model, and I have always treated that as the boring work around the real work. I'm putting their rule that changing anything changes everything on a sticky note over my desk.

Limitations. This paper has no measurements. Every claim rests on the authors' experience, so there's no way to tell how common each kind of debt really is. The systems described are Google scale, and a team of a few students has fewer places for debt to hide, so some of the advice would be overkill for me. It's also better at naming problems than at solving them, and it's from 2015, so tools for tracking data and models have gotten better since.

Conclusions. I want to draw my own vision pipeline from the camera to the decision and list every piece of data and every setting it depends on. The first question I plan to answer is one the authors close with, how quickly a new team member could be brought up to speed. Whoever inherits my code after I graduate is the person I should be writing it for. My open question is whether a small project like mine can get most of the benefit from a few checks and skip the rest.

Reflection. Most of what I've learned about machine learning is how to make a model, and almost none of it is about keeping one alive. This paper is a reminder that a model I can't reproduce or hand off isn't really finished. I'd like my ISM project to be something another

person could pick up and run without me in the room. I think that takes planning from the start and not cleanup at the end.

Zhang, Chiyuan, et al. "Understanding Deep Learning Requires Rethinking Generalization." *International Conference on Learning Representations*, 2017. *arXiv*, arxiv.org/abs/1611.03530.

Introduction. Zhang and four coauthors ran a simple but unsettling test on neural networks. They trained image classifiers on pictures whose labels had been replaced with random ones, so there was nothing real to learn, and the networks fit them anyway. The paper uses that result to argue that the usual explanations for why neural networks generalize are not enough.

Aims and Research Methods. The question behind the experiments is what separates a network that learns from one that memorizes. The authors trained standard image networks on CIFAR-10 and ImageNet and changed the data in several ways, including random labels, partly random labels, and pictures replaced by pure noise. They tracked training error and test error each time and repeated the runs with weight decay, dropout, and data augmentation turned on. A short theorem then shows that a two-layer network with enough parameters can fit any labeling of a sample.

Scope. The experiments use two image data sets, CIFAR-10, with ten categories of small pictures, and ImageNet, with 1,000, and several network types including Inception, AlexNet-style networks, and plain multilayer ones. These were the standard benchmarks of the time, and the authors needed cases where everyone agreed the networks generalize well so that the contrast with memorizing would be hard to argue with.

Usefulness. This one made me uneasy, in a good way. I had a picture in my head where a network is forced to find patterns because it is too small to remember everything. The results say that picture is wrong, since the network can remember everything and still do well on real labels. The result that stays with me is test error climbing to 90 percent on CIFAR-10 while training error stays at zero. It's the clearest picture of overfitting I have seen, and it means my training accuracy tells me almost nothing, and Domingos says the same thing but this paper shows it.

Limitations. The paper is much better at showing what does not explain generalization than at saying what does. The authors treat this as an open problem for the field, and the theorem only shows that a network could fit any labels, not why training on real labels ends up somewhere good. The experiments use 2016-era image sets, and the ImageNet networks are far bigger than anything I could train. A robotics team's data set is probably much smaller than the 50,000 training images in CIFAR-10, so I can't tell whether the same lessons apply at my size.

Conclusions. I'm going to run the same kind of test on my own model. If I shuffle the labels in my training set and the model still reaches a low training loss, then it can memorize my data and I should only trust the test results. It is a cheap check that I could run in an afternoon. The question I can't answer yet is why networks generalize at all if they can memorize, and I think it's a good one to bring to my mentor.

Reflection. This is the paper that made machine learning feel like a field with open questions instead of a set of recipes. I chose this topic because I like making things work, and this paper is about something that works that nobody can fully explain. That used to bother me. Now I think it's part of the reason to study it, and I want what I present for ISM to be honest about what I still don't understand.