7 Tips for Dealing With Small Data

Because more often than not, that's what you're gonna get.

Daniel Rothmann Jun 25, 2019 · 7 min read ★

We often hear that ***Big Data*** is the key to building successful machine learning projects.

> ***This is a major problem:*** *Many organizations won't have the data you need.*

How can we prototype and validate machine learning ideas without the most essential raw material? How can we efficiently obtain and create value with data when resources are sparse?

At my workplace, we produce a lot of functional prototypes for our clients. Because of this, I often need to make ***Small Data*** go a long way. In this article, I'll share 7 tips to improve your results when prototyping with small datasets.

1: Realize that your model won't generalize that well.

This should be the first order of business. You're building a model whose knowledge is based on a **tiny fraction of the universe**, and that should be the only place or situation where it can be expected to work well.

If you're building a computer vision prototype based on a selection of indoor photos, don't expect it to work well outdoors. If you have a language model based on chatroom banter, don't expect it to work for a fantasy novel.

Make sure this is understood by your manager or your client. This way, everyone can align on a realistic expectation of the results that your model should deliver. It also creates an opportunity to come up with useful new KPIs to quantify model performance both inside and outside the prototype scope.

2: Build good data infrastructure.

In many situations, the client won't have the data you need, and public data won't be an option. If part of your prototype requires collecting and labeling new data, make sure that your infrastructure to do this creates as little friction as possible.

You'll want to **make sure data labeling is very easy** so it's approachable for non-techies as well. We have started using ***Prodigy***, which I think is a good tool: Both accessible and extendable. Depending on the size of the project, you might also want to **set up an**

automated data ingest which can take in new data and automatically feed it to the labeling system.

> *If getting new data into your system is quick and easy,* ***you will get more data.***

3: Do some data augmentation.

You can often extend your dataset by augmenting the data that you have. It's about making slight changes to the data that should not significantly change the model output. For example, an image of a cat is still an image of a cat if it's rotated 40 degrees.

In most cases, **augmentation techniques allow you to produce many more "semi-unique" data points for training your model**. You could, as a start, try adding small amounts of Gaussian noise to your data.

For computer vision, there are lots of neat ways to augment your images. I have had positive experiences with the ***Albumentations*** library which does many useful image transformations while keeping your labels unharmed.

Photo credit: Albumentations on Github

Another augmentation technique that many have found useful is ***Mixup***. This technique literally takes two input images, mixes them together and combines their labels.

Photo credit: Cecilia Summers & Michael J. Dinneen

When augmenting other input data types, some consideration needs to be given to which transformations would change a label and which wouldn't.

4: Generate some synthetic data.

If you have exhausted your options for augmenting real data, you could start thinking about creating some fake data. Generating synthetic data can also be a great way to cover some edge cases that your real dataset does not.

An an example, many reinforcement learning systems for robotics (like OpenAI's ***Dactyl***) are trained in simulated 3D environments before they are deployed to real robots. For image recognition systems, you can similarly build 3D scenes that could provide you with thousands of new data points.

15 simulated instances of **Dactyl** training in parallel.

There are many approaches to creating synthetic data. At Kanda, we are developing a turntable-based solution to create data for object detection. If you have a very high data requirement, you could consider using ***Generative Adverserial Networks*** to create synthetic data. Be aware that ***GANs*** are notorious for being hard to train, so make sure that it'll be worth it first.

NVIDIAs **GauGAN** in action!

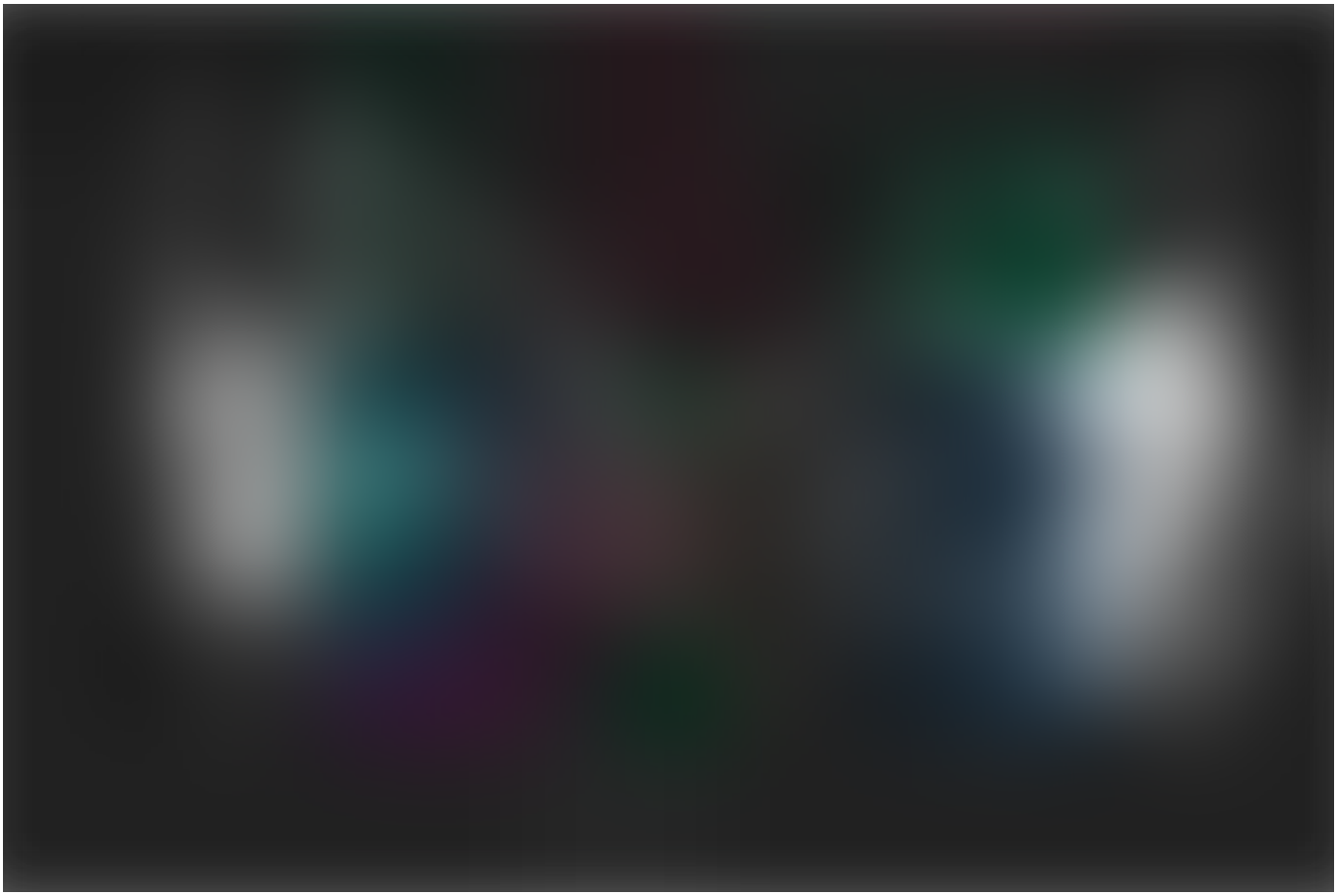
Sometimes you can combine approaches: Apple had a very clever way of **using a GAN to process images of 3D modeled faces to look more photo-realistic**. Awesome technique to extend your dataset, if you have the time.

5. Beware of lucky splits.

When training machine learning models, it is quite common to randomly split the dataset into **train and test sets** according to some ratio. Usually, this is fine. But when working with small datasets, there is a high risk of noise due to the low volume of training examples.

In this case, **you may accidentally get a lucky split**: A particular dataset split where your model will perform and generalize really well to the test set. In reality though, this might just be because the test set (by coincidence) contained no difficult examples.

In this scenario, ***k-fold Cross-Validation*** is a better choice. Essentially, you split the dataset into ***k*** "folds" and train a new model for each ***k*** where one fold is used for the test set and the rest is used for training. This controls that the test performance you see is not simply due to a lucky (or unlucky) split.



6. Use transfer learning.

If you're working with a somewhat standardized data format like text, images, video or sound, you can leverage all the previous work others have put into these domains with transfer learning. It's like standing on the shoulders of giants.

When you do transfer learning, you take models that others have built (usually, *"others"* being Google, Facebook or a major university) and **fine-tune them to suit your particular needs**.

Transfer learning works because most tasks to do with language, images or sound share many common characteristics. For computer vision, it could be detecting certain types of shapes, colors or patterns, for example.

Recently, I worked on an object detection prototype for a client with a high accuracy requirement. I was able to speed up development enormously by fine-tuning a ***MobileNet*** **Single Shot Detector** which had been trained on Google's ***Open Images v4*** dataset (~9 million labeled images!). After a day of training, I was able to produce a fairly robust object detection model with a **test mAP of 0.85 using ~1500 labeled images**.

7. Try an ensemble of "weak learners".

Sometimes, you just have to face the fact that you do not have enough data to do anything fancy. Luckily, there are many traditional machine learning algorithms you can fall back to which are less sensitive to the size of your dataset.

> *Algorithms like the* ***Support Vector Machine*** *are a good choice when the dataset is small and the dimensionality of the data points is high.*

Unfortunately, these algorithms are not always as accurate as state-of-the-art methods. This is why they can be called "weak learners", at least in comparison to highly

parameterized neural networks.

One way to improve the performance is to combine several of these “weak learners” (this could be an array of Support Vector Machines or Decision Trees) so that they “work together” to produce a prediction. **This is what Ensemble Learning is all about.**

Thanks for reading! If you found this article useful, **come find me at LinkedIn** where I regularly post more stuff like this 🤓

Do you have any tips for dealing with Small Data? Please feel free to share!

Machine Learning Deep Learning Neural Networks Data Science Towards Data Science