How to Find out Your Data Science Progress or How much you have learned or you about Data Science

Entry level

Data handling

- [ ] Small datasets
- [ ] Simple preprocessing
- [ ] Image data
- [ ] Text data
- [ ] Audio and time-series data

Networks

- [ ] Classic machine learning
- [ ] Basic Neural Networks
- [ ] Convolutional Neural Networks
- [ ] Recurrent Neural Networks

General

- [ ] Data analysis
- [ ] Saving and loading models
- [ ] Working with metadata files
- [ ] Callbacks

Intermediate level

Data handling

- [ ] Generators
- [ ] Augmentations
- [ ] Large datasets
- [ ] Custom pipelines

Custom projects

- [ ] Custom audio / time-series project
- [ ] Custom image project
- [ ] Custom text project

Training

- [ ] Transfer learning
- [ ] Fine-tuning
- [ ] Custom callbacks
- [ ] Multi-GPU training
- [ ] Custom training loops
- [ ] Training in the cloud
- [ ] TPU training

General

- [ ] Problem thinking
- [ ] Generative networks
- [ ] Experiment tracking
- [ ] Hyperparameter search
- [ ] Custom layers
- [ ] Advanced architectures

Advanced level

General

- [ ] Huge datasets
- [ ] Model deployment
- [ ] Multi-worker training
- [ ] Reinforcement learning
- [ ] *Research*
- [ ] *Staying up-to-date*

Entry level

This is the place to start. It covers the fundamentals of your further journey.

Data handling

The intention behind this category is to focus on being able to handle the most common data types:

- Images
- Text
- Audio/Time-Series

In general, the datasets at this stage are quite small, and there is no (mental) overhead from handling larger-than-memory datasets. Good examples are the classic MNIST image dataset (PyTorch, TensorFlow), the IMDB reviews text dataset (PyTorch, TensorFlow), and small audio or time-series datasets. They are only a few hundred MBs at most and comfortably fit into the RAM.
Some of these datasets require minimal pre-processing (scaling images, shortening sentences), which is usually not more than a few lines of code. Since the number of examples is either very low or they are of small size only, you can easily do the processing during runtime (as opposed to running separate complex scripts in advance).

In summary, this category emphasizes handling small audio/time-series, image, and text datasets and applying simple operations to pre-process the data.

Networks

The intention behind this category is to transition from classic machine learning towards neural networks and knowing the common building blocks:

Classic machine learning techniques include Support Vector Machines, Linear Regression, and clustering algorithms. Even though the more sophisticated relatives, the neural networks, seem to dominate the recent research, they come in handy for small problems — or as a baseline. Knowing how to use them is also handy when it comes to analyzing the data.

When progressing towards deep learning, dense layers are a good start. I guess that they are used in nearly every (classification) model, at least to build the output layer.

The second commonly used network type is the Convolutional Neural Network, which uses convolution operation at its core. It’s hard to think about any successful research that has not used and benefited from this simple operation: You slide a kernel over your input and then calculate the inner product between the kernel with the patch that it covers. The convolution operation extracts a small set of features from every possible

location of the input data. It's no wonder that they are very successful in classifying images, where important characteristics are scattered all around.

The previous two network types are mainly used for static inputs, where we know the data's shape in advance. When you have data of varying shapes, take sentences as an example, you might look for more flexible approaches. That is where Recurrent Neural Networks come in: They can retain information over long periods, which enables connecting the "I" in "I, after getting up this morning and dressing, took a walk with my dog" and the "took a walk with my dog" to answer the question "Who took a walk with the dog?"

In summary, this category focuses on working with simple Dense, Convolutional, and Recurrent neural networks to classify image, text, and time-series data.

General
The intention behind this category is to learn the general handling of Data Science related tasks.

An important step is to get to know the data itself. This is not limited to image data or text data alone but also covers time-series and audio data and any further data type. The term exploratory data analysis (thanks to Andryas Waurzenczak for pointing this out) describes this step best: Using techniques to find patterns, outliers (data points that exceed a common range), substructures, label distributions, and also visualizing the data. This might incorporate PCA or dimensionality reduction techniques. The datasets that you work with at this point are usually already explored in detail (try a search for the dataset's name to find interesting characteristics), but learning this now will pay off once you proceed to custom datasets.
You will also learn how to load and save those models above so that you can re-use them later. What's also common is to store data about the data, the *metadata*, in separate files. Take a CSV file as an example: The first column stores the file paths to your data samples, and the second column stores the class for the samples. Being able to parse this is mandatory, but thanks to many libraries, that is a straightforward task.
When you begin training your basic networks on those small datasets, with those architectures above, you'll gradually uncover the helpfulness of callbacks. Callbacks are code that gets executed during training, and they implement many functionalities: Periodically saving your model to make it fail-safe, stopping the training, or changing parameters. The most common ones are already built-in with most libraries, and a short function call is all it takes to use them!

In summary, this category has you learn to handle the tasks around running neural networks on small datasets.

Intermediate level

In my opinion, this is where the great fun begins, and you'll reach it faster than you might expect! I found the shift to happen quite naturally: You look for a more efficient

way to process data — and before you realize it, you have written a custom pipeline for a large dataset.

Data handling

The intention behind this stage is to be able to handle larger or more complex datasets which might require special treatment in the form of augmentation and custom pre-processing pipelines.

At this stage, the datasets tend to become larger and might no longer fit into your RAM. Efficient pre-processing becomes more important, as you won't let your hardware idle around. Also, you might have to write your own generators when you handle samples of unequal shapes, fetch samples from a database, or do custom pre-processing.

Or you work with complex and unbalanced datasets, where the majority of your samples is of one class, and only a minority of samples is of the desired class. There are a few helpful techniques you'll learn: Augmenting the data to generate more samples of the minor class or downsampling the major class.

For both dataset types, custom pipelines become more important. When you quickly want to iterate settings, for example, cropping an image to 32x32 rather than 50x50, you rarely want to start long-running scripts. Incorporating such possibilities into your pipeline enables quick experimentation.

In summary, the datasets tend to become more complex (think unbalanced) and larger (think up to 30—40 GBs) and require more sophisticated handling.

Custom projects

This is the heart of the Intermediate level. The intention behind this is to work on custom projects and thereby learning and using many of the other categories' items.

The first level might have led you through training a network on the MNIST datasets. On this level, you'll naturally focus more on your own data and how to parse it. I have only listed the three major domains audio, images, and text, but there are many more.

By working on your own projects, you connect what you have learned earlier to solve your challenges.

Training

The intention behind this category is to learn more about training neural networks. This category is the largest category on the Intermediate level, which is due to the focus on more advanced topics.

A first step to customizing training is using transfer learning and fine-tuning. Before, you will usually have used the standard methods to train and test your models, but now you might require something more powerful. That's where it comes in handy to simply re-use models that other practitioners trained. You load their models and carefully train them on your own datasets.

In case you miss some feature, this is the point where you begin to write custom training loops and custom callbacks. Want to print some statistics every 20 batches? Write a

custom callback. Want to accumulate gradients over 20 batches? Write a custom training loop.

More complex loops might need more resources: multi-GPU training is coming! However, it's not simply adding a second, a third, or even more resources; you have to fetch the data fast enough. Keep in mind that multi-device training increases the complexity. Most of the background work is already done by PyTorch and TensorFlow, which handle this case with a few minor code changes only, but the front-end part is left to you. (But fret not, there are many guides out there to help you get this done!)

If you are like me and do not have access to multiple GPUs, then training on the cloud is a possible next step. All major cloud services provide you with the requested resources. At first, there will be some (mental) overhead while transitioning from local setups to cloud computing. After some attempts, this gets way easier. (When I started to run scripts on a Kubernetes cluster, things were very overwhelming: What is a Dockerfile? How to run jobs? How to use local resources? How to get data in and out? After experimenting around, I got used to it, and now it's a few simple commands to get my scripts running).

When you are already in the cloud, why not try TPU training? TPUs are Google's custom-made chips. And they are fast: A year ago, I worked with a team to classify a large text corpus, around 20,000 documents, and each document had about 10 pages of text. We ran our first experiments on a CPU only, and one epoch took 8 hours. In the next step, we improved pre-processing, caching, and used a single GPU: The time went down to 15 minutes, which was a huge leap. After reading TensorFlow's documentation and playing around with our code, I then managed to make TPU training possible, and one epoch went down to *15 seconds*. So, I encourage you to upgrade your pipeline and run your training on TPUs.
In summary, this category focuses on expanding the actual training parts towards more complex training loops and custom callbacks.

General
The intention behind this category is to learn more about what's possible beyond using the normal architectures.

Now that you are dealing with custom datasets, it becomes important to get a grasp on problem thinking. Let me explain it this way: Say you are working with an audio dataset. You have done your initial data analysis, and it turns out that your data is not only imbalanced, but the samples are of different lengths. Some are only three seconds long, others 20 seconds and more. Further, you want to classify only segments of the audio, not the whole clip as is. And lastly, this is for an industry project, so restrictions apply. Bringing all this together is what's required from you.

Luckily, you are not alone at this. Data analysis is ticked already, and pre-processing custom data is ticked too. It's the industrial part that gets the main attention here. Check what other folks have done (and published on GitHub), ask your people, and experiment with different techniques.

Say you want a network layer that normalizes your data following a custom scheme. Looking into your library's documentation, you don't find anything similar already implemented. Now is the time to implement such functionality yourself. As before, TensorFlow and PyTorch (which makes this an even simpler approach) provide some good resources to begin with.
For a project a while ago, I wanted to write a custom embedding layer. After finding no implementation, I decided to tackle this by writing a custom one. Now, that was a challenge! After many trial and error loops, I finally came up with a working TensorFlow layer. (Ironically, in the end, it did not prove better than the existing ones.)

Another large field in Deep Learning opens up now: Generative networks. Before, you mainly worked with classifying existing data, but nothing is holding you back from also generating it. That's exactly what generative networks do, and a large part of their recent success comes from one single paper: Generative Adversarial Networks. The researchers really came up with something clever: Instead of training one network, two networks are trained, and both get better over time. Examining their workings in detail is out of scope here, but Joseph and Baptiste Rocca did a terrific job explaining them here.
Besides only using a wider variety of models, you'll also want to track your experiments. This is helpful when you want to comprehend how your model's metrics are influenced by its parameters. Speaking of which, you might also start a parameter search, which aims to find the best set of parameter which optimizes a target metric.

As in the last part, you will definitely use more advanced models. Besides those generative networks, there are also huge language models. Have you ever heard of Transformers (not the movie)? I bet you have, and now comes the time to use them for your own projects. Or try getting access to GPT-3, and discuss AI with it.

In summary, this category expands beyond the normal models and explores further techniques around.

Advanced level

This is the last stage, though the boundaries to the previous stage are admittedly blurry. Congratulations on getting here! There is only one category for you to explore since the items build on your previous experience.

General

Even if you have come this far, there are still further things to explore.

The first item is *huge datasets*. While starting the training process is a routine task for you now, the focus here is on making training fast, despite huge datasets. With *huge* I mean datasets of several hundred GBs and more. Think ImageNet scale or The Pile scale. Tackling them requires thinking through your storage options, data fetching, and pre-processing pipelines. And not only the size increases, but also the complexity:

A multi-model dataset of a few hundred GBs, where each image is accompanied by a textual and auditive description? Now that requires some thinking.
When working with such enormous datasets, it might be required to run a multi-worker training setup. The GPUs are no longer installed on a single machine but distributed across multiple machines, the workers. Managing the distribution of your workload and your data becomes necessary. Thankfully, PyTorch and TensorFlow have some resources on this, too.
After you have spent hours setting up your models, why not deploy them? Use some libraries that let you build a nice GUI easily (such as streamlit.io) and deploy your models so that other practitioners can see your work live.
When you have also done that, get into Reinforcement Learning. Placing it in the advanced section might not be 100 percent correct (the basics can be grasped quite early), but it's mostly distinct from the fields you have worked on so far. As always, there are some resources available to get you started.
Now that you have gained some vast experience and knowledge, it's time to do research. Have you noticed anything on your journey that could be improved? That could be the first thing to work on. Or contribute to other research projects by providing code to their repositories. Collaborate with other enthusiasts and keep learning.

That might be the last thing that's left here: Staying up-to-date. Data Science is a fast-moving field, with many new exciting things coming up day after day. Keeping track of what matters is hard, so choose some newsletters (I read Andrew Ng and deeplearning.ai's The Batch) to get condensed information. To narrow down, you can also use Andrew Karpathy's Arxiv Sanity Preserver, which helps you filter papers that meet your interests.
In summary, this category focuses on exploring Reinforcement Learning as a completely new field, contributing to research, and staying up-to-date—the last two points are never truly ticked.

Where to go next?

You can find the checklist on Notion here and a PDF here. Make a copy, customize it, and then gradually tick it off.
Please leave any suggestions or remarks on GitHub.
If you are looking for specific resources:

- Most of the entry-level is covered by deeplearning.ai's TensorFlow Developer Professional Certificate (try the TF exam afterward to see what you have already learned!)
- Parts of the entry and intermediate and advanced levels are covered by Berkley's Full Stack Deep Learning course
- Some parts of the intermediate and advanced levels are covered by DeepMind's Advanced Deep Learning & Reinforcement Learning lecture

- Parts of the intermediate and advanced levels are covered by deeplearning.ai's TensorFlow: Data and Deployment Specialization and TensorFlow: Advanced Techniques Specialization

If you are looking for a more concrete list you can check Daniel Bourke's roadmap here.

Questions

… is missing.

Leave a comment and let me know. Keep in mind that this checklist is intended as a general overview and does not contain *working with NumPy/pandas/TensorFlow/…* or similar specific items.

Isn't there an overlap between some categories?

Yes, definitely. There are no distinct dividing lines between some items from different categories. For example, when you work with images, you might have stored them on disk, and a CSV file holds all the file paths and labels. You can thus tick two items in one go.

Similarly, there is sometimes an overlap between two items within the same category. This happens when they mostly go hand-in-hand but are still somewhat different.

Is there an order?

I originally tried to maintain order within each category (e.g., within *Training*), with the easiest task at the top and the harder tasks at the bottom. But there is not always a clear difficulty ranking between two items. Take *transfer learning* and *fine-tuning* as two examples: Both are tightly intertwined, and there's no clear order.

What's so special about TPUs to give them a separate place?

While it's thankfully easy to go to Colab and select a TPU, it's crucial to get the data ready first. My intention behind this point is not to simply having run a model with the help of TPUs, but to have made your pipelines very efficient so that data gets to the TPU on time. The focus is thus more on efficient pre-processing rather than on speeding up your computations.
And it's admittedly quite cool to run your code on TPUs.

The list is quite long. How can I complete it?

Two short answers:

— You don't have to; I am myself far away from this point. Use it as a mere guideline for your next endeavors.

— You don't have to complete one level or category before advancing to the next one.

And a longer answer:

Take one day per week, block all other things out.

On the other days, you can do your studies. Take a course on Wednesday, on Thursday, and Friday. Studying on the weekends is a bonus.

Then, on Monday, you repeat what you have learned over the last days, using the weekend to consolidate things.

And on your blocked Tuesday, you apply your new knowledge to your own projects.

These projects don't have to be big:

Learned something about efficient data processing? Set up a pipeline for your own data (and tick *custom pipelines*).
Learned how to classify images? Take some images of stuff in your room, and classify them (and tick *custom image project*).
Learned about a specific network architecture? Use a library of your choice and simply re-implement it (and tick *advanced architectures* or *convolutional neural networks* or both).
Use these short standalone projects to get your hands on a broad range of topics. The more small topics you work on, the easier you can get started with those large projects that seemed daunting at first—because they only consist of many small steps you have already taken.