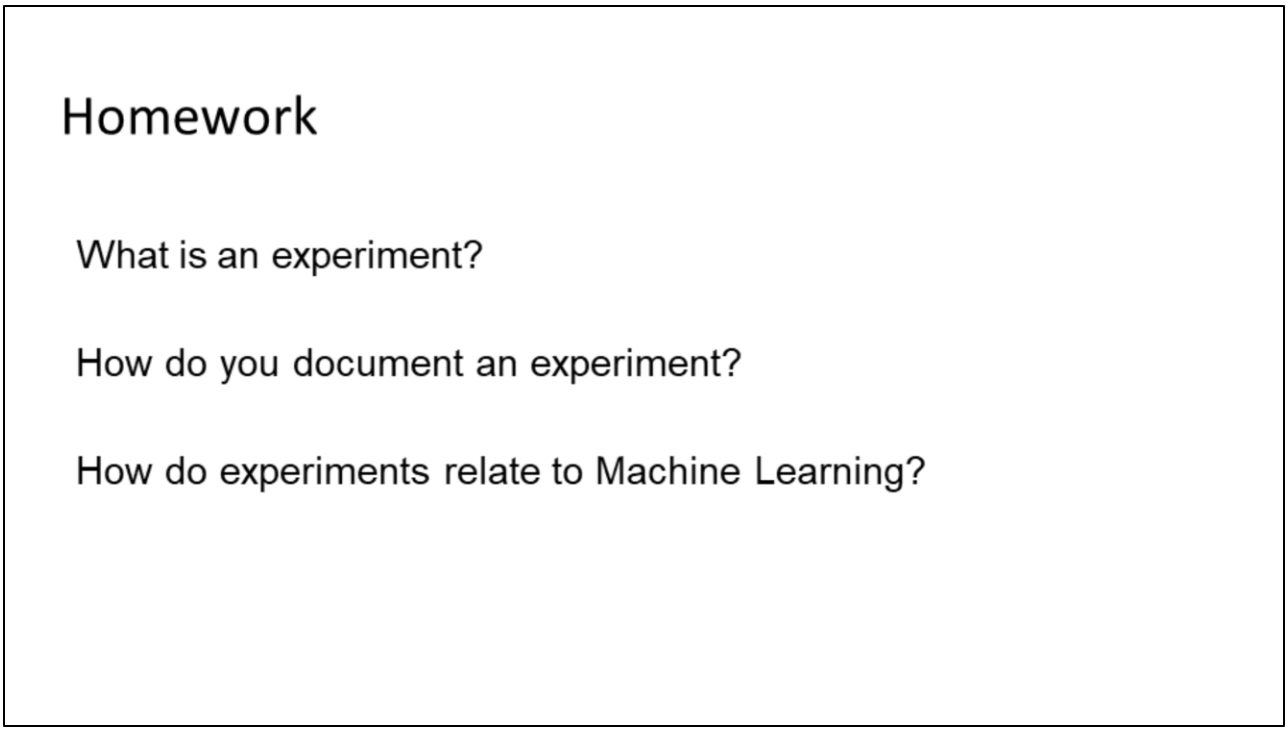
Complex Machine Learning Projects

Part 2: A tale of three experiments



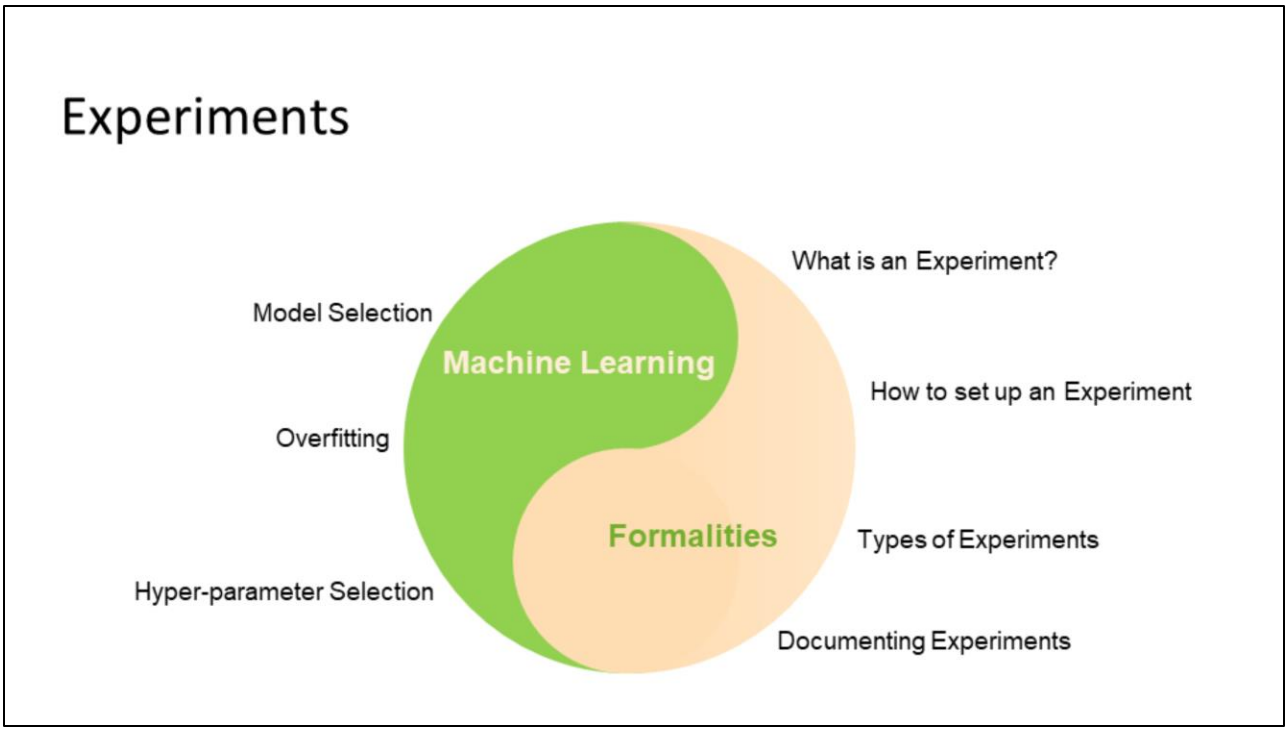
As part of your homework you were requested to research about experiments. What are your answers to the above questions?

My answers would be the following:

What is an experiment?
An experiment is a test you perform to gain information about the outcome of the test. You could do that in order to gain new knowledge, to test a hypothesis or to just demonstrate the outcome to someone else to convince them of something (for example during teaching). The test is usually controlled in the sense that you bring your test equipment / software into a certain state, then start a specific procedure and record and interpret the outcome of the procedure.

How do you document an experiment?
While documenting an experiment you should document the procedure you followed as well as the results you achieved. A well-documented experiment should enable a reader to understand how you performed the test. It should contain enough details that they can critique and repeat the experiment. The results of the experiment and your interpretation should be recorder and clearly separated. Sometimes you need to document these experiments in order to communicate their outcome, e.g., to argue for design decisions in a development team or to communicate project outcomes to clients. Describing them as rigorous experiments can help others follow and double-check your decisions.

How do experiments relate to Machine Learning?
In machine learning we often are not able to identify which design decisions are good until we try them out. This relates to activities like selecting the right model, testing training configurations and finding good hyper-parameters. Consciously thinking of these processes as an experiment can help to approach them in a more rigorous manner which produces more reliable results.

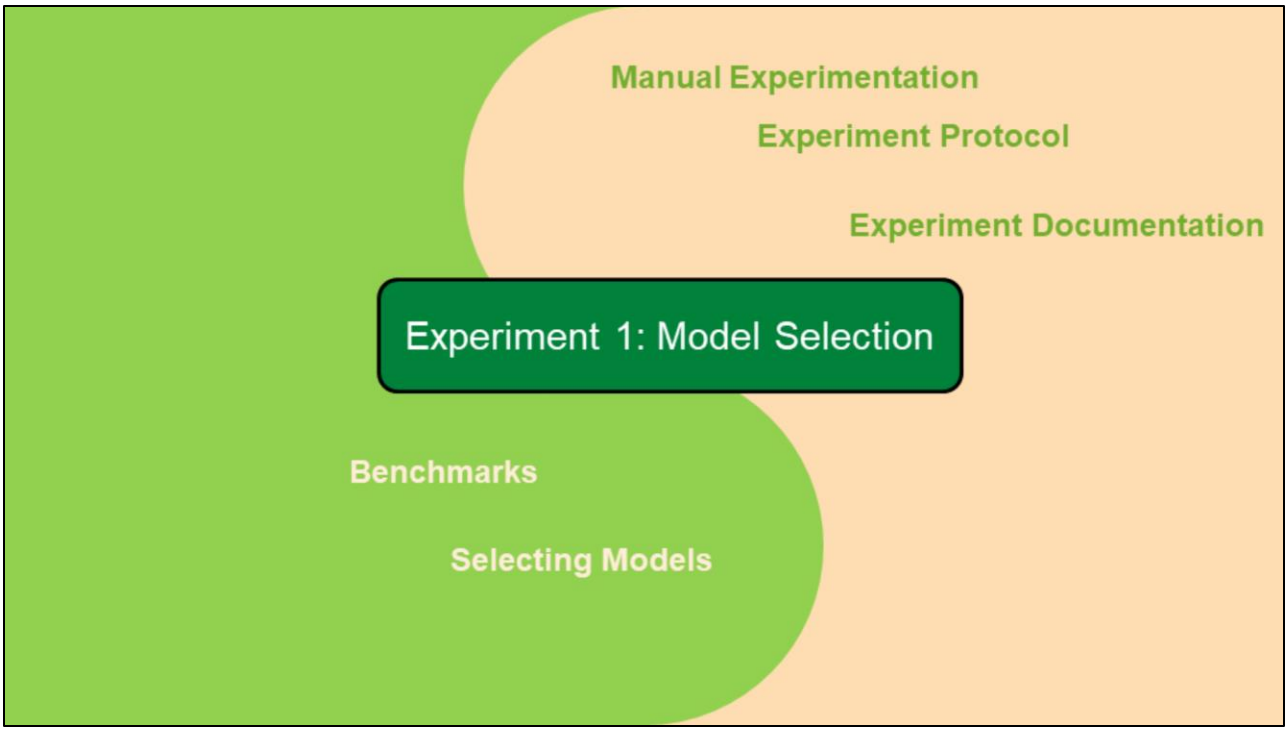


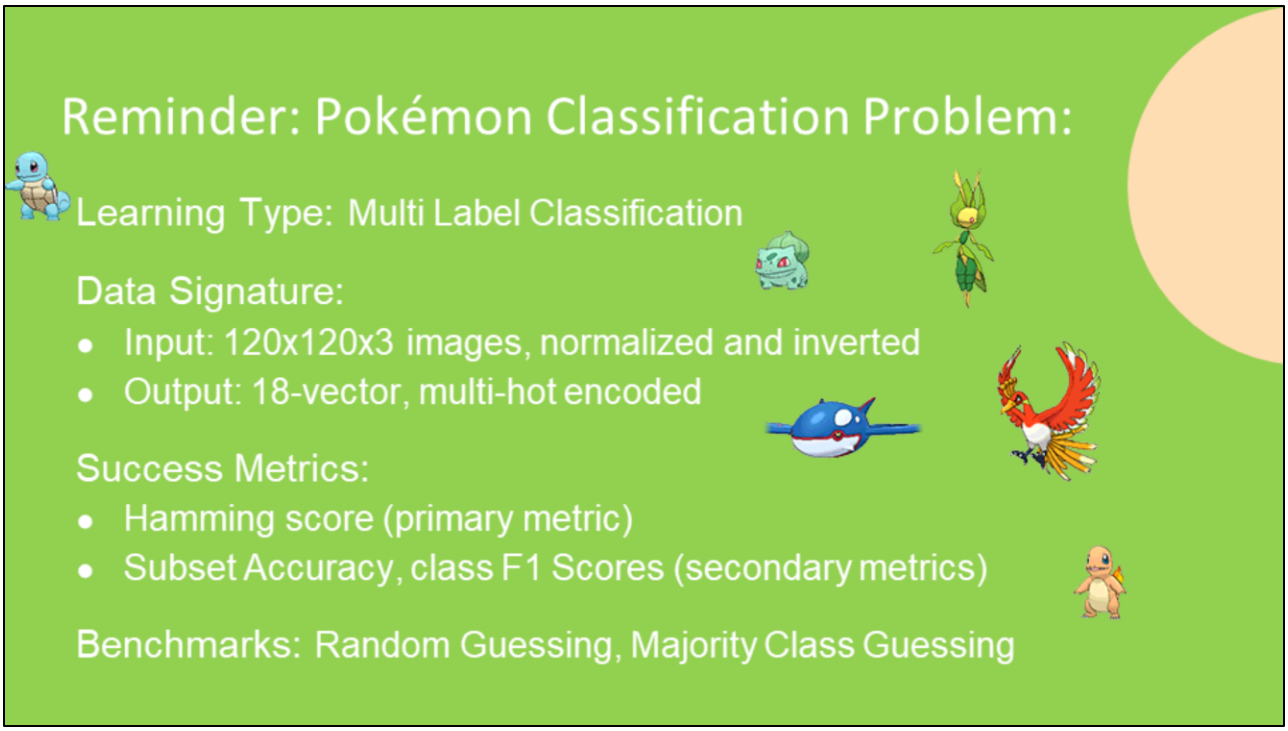
Today we will focus on experiments. As last time, we will talk about this topic from two different sides (although the sides differ a little bit).

On the **Machine Learning** side, we will talk about a few processes that likely require experiments. We will talk about Model Selection, the reduction of overfitting and Hyper-parameter selection.

On the **Formalities** side, we will talk about different types of experiments, how to set them up and how to document them.

We will do so by talking about three experiments we performed on our running example, the Pokémon classification. We will go through the respective notebooks and use the slides to talk about things on a more general level.


Manual Experimentation
Experiment Protocol
Experiment Documentation
Experiment 1: Model Selection
Benchmarks
Selecting Models



A quick reminder for our running example:

The learning type is a Multi-Label Classification. We have a classification problem where each Pokémon can have one or more types. The data signature of this problem contains images of dimension (120,120,3) as input and a vector of class predictions as output. There are 18 Pokémon types, so the output vector has a length of 18.

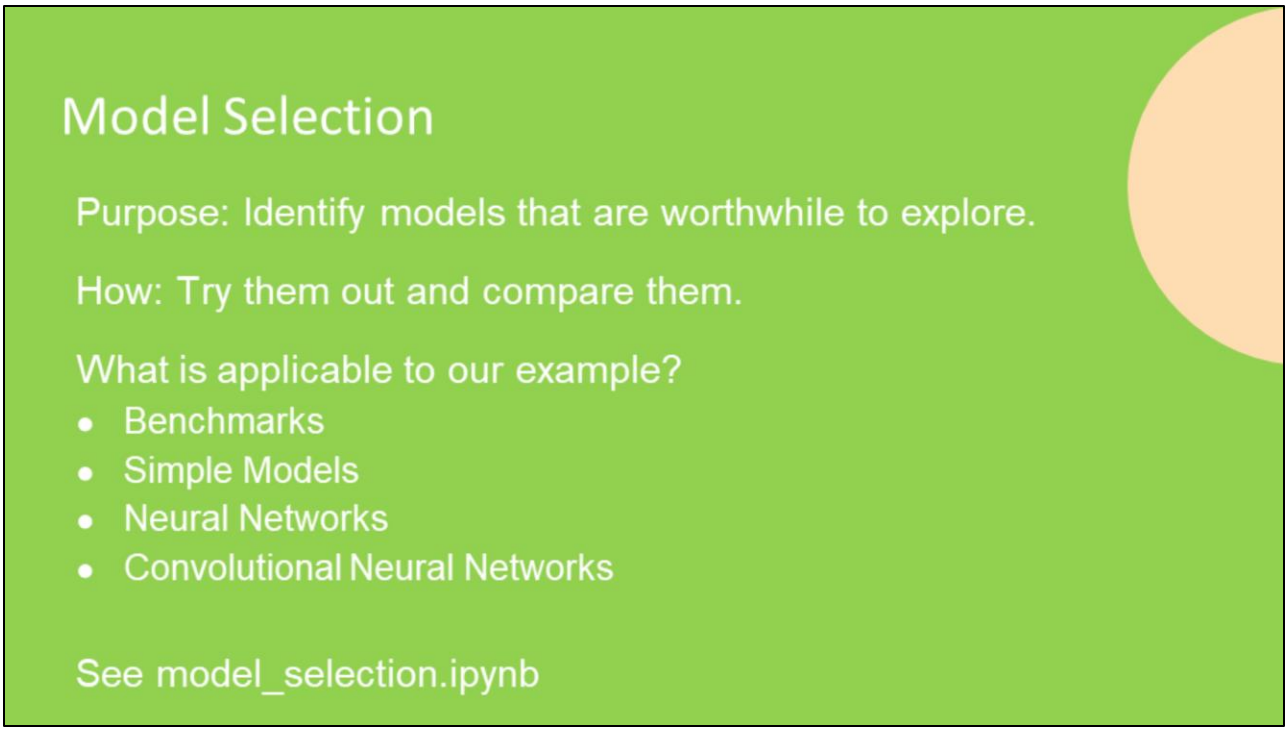
Several success metrics can be chosen for a multi-label classification:

- Subset Accuracy represents the percentage of Pokémon have been guessed correctly. This metric is fairly harsh – it does not reward parial or close guesses.
- The Hamming Score is a more lenient metric. It counts the relation between correct guesses and overall guesses and classes per sample.
- The F1 score is defined for binary classification. We can extend it to the multi-label classification problem by applying it to each individual label and then averaging.

We have selected the Hamming Score as our primary metric. We will use this during experiments to detect which models perform better. Subset Accuracy ad F1 scores will be checked when evaluating the final model. The F1 score will not be averaged. Instead, we will apply it to each class to get a feeling for how well that class is predicted.

There are two natural benchmarks that we want to beat: Guessing Randomly and

always guessing the class with the highest occurrence in our data. If we beat these two benchmarks, we can claim to have learned some meaningful patterns. However, being more meaningful than random with 18 classes is not a highly accurate result. We hope to beat these. scores by a good margin.
A very good result would be close to 1 in hamming score. Due to the small size of the data set we do not think this is realistic. We would be happy with a hamming score of between 0.5 and 0.8.



Model selection is the process of finding one or more suitable models that could solve your machine learning problem.

Your machine learning problem will already reduce the options of machine learning algorithms that can be applied to it. And an experienced Machine Learning engineer will likely be able to reduce the options further based on their experience and intuition. However, it still often becomes necessary to test out different options for models to find out which one performs well on your data.

Specifically, it is often a good idea to test out simple models, even if you suspect you will have to use some more complicated ones. Sometimes, simple models can surprise you or give you good benchmarks to compare the more complex ones against.

Speaking of benchmarks … you should think of simple benchmarks that your machine learning models absolutely have to beat. A usual example is guessing randomly. While the process of comparing models to each other can get you information on which models are better, you need to compare a benchmark in order to identify whether the results you achieved are promising at all.

In our running example, we have four categories of models that we can compare:

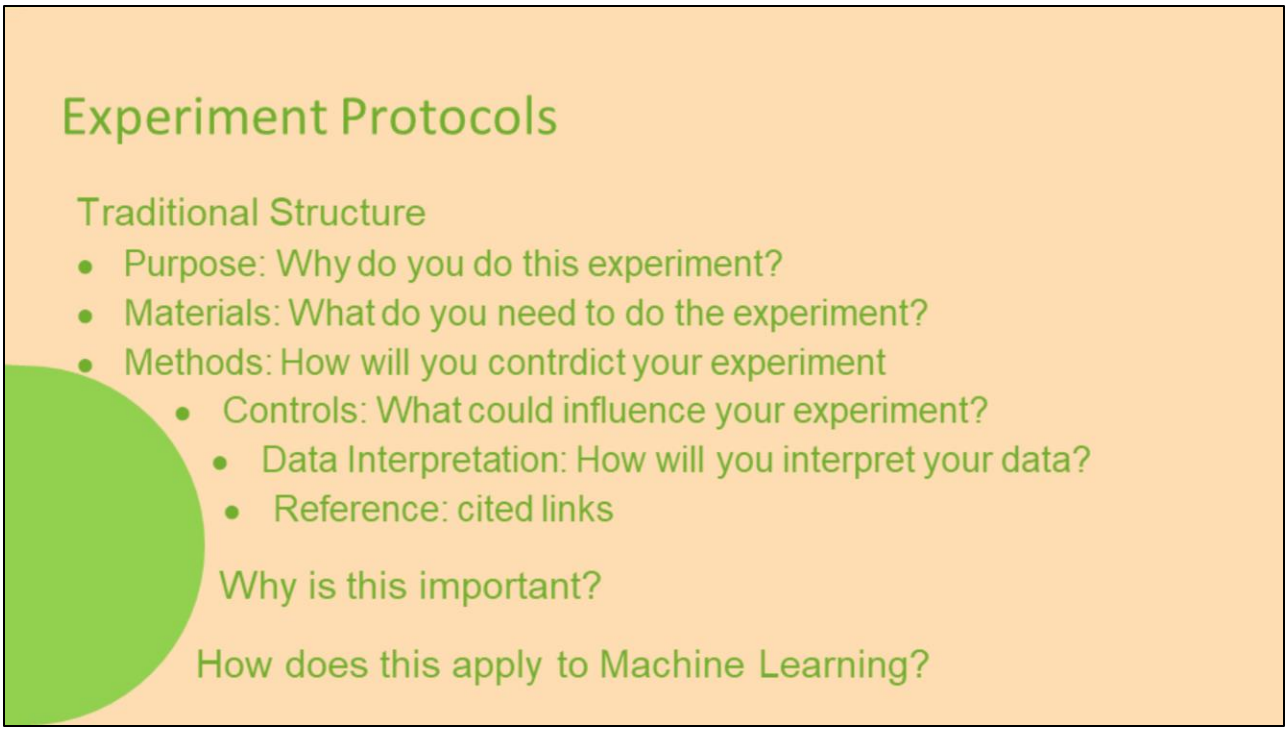
- Benchmarks: We will use random guessing and always guessing the majority class as benchmarks. A model is only viable if it beats both of these benchmarks by a

significant margin.
- Simple Models: These are the simpler models from the obvious machine learning libraries. We do not have any good intuition on which ones may be more promising, meaning we will simply try out all simple models that are applicable to our multi-label classification problem.
- Neural Networks: Artificial neural networks with only fully connected layers.
- Convolutional Neural Networks: Convolutional neural networks with convolutional layers, pooling and a fully connected part.

To see this experiment, you should work though *model_selection.ipynb*.

After performing this experiment we have concluded that Convolutional Neural Networks are the best model.

An experiences machine learning engineer usually has some good intuition on which types of model may be viab



Let's talk about how to plan experiments.

An experiment protocol is a plan for how to conduct an experiment that you define before you carry out the experiment. If you look up experiment protocols in science, you will likely fine a structure that is similar to this one:

1) Purpose: Decribes why you want to carry out the experiment and what insights you want to generate by it.
2) Materials: Describes what you need to carry out the experiment.
3) Methods: Describes how exactly you will conduct your experiment.
4) Controls: Describes which external factors could influence the outcome of your experiment and whether / how you can exclude them.
5) Data Interpretation: Describes how you plan to interpret the data produced by your experiment.
6) References: a list of references used in the above descriptions.

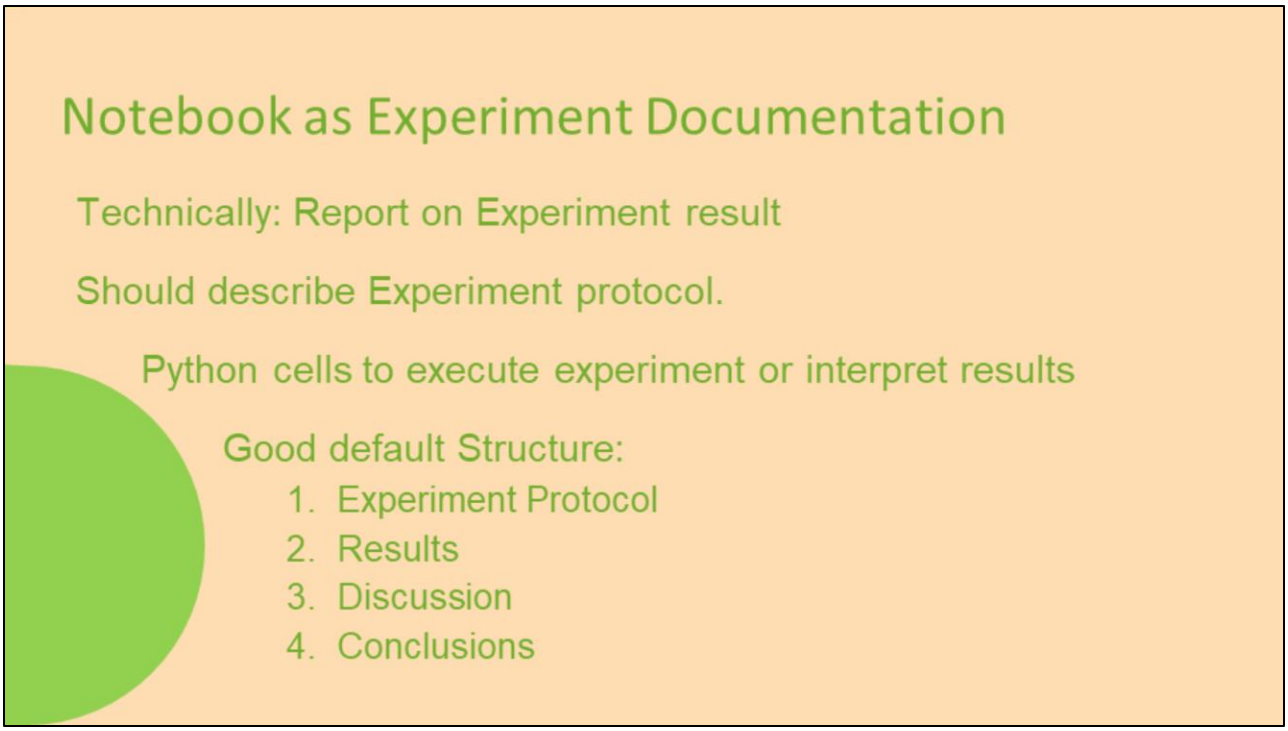
Why is it important to define such a protocol? It gives you guidance and makes sure that you actively think about how you will conduct your experiment before you do. Experiments can be expensive or take a long time. By thinking about methods and eventualities beforehand you can maximize the chances of the outcome being valuable.

This also applies to machine learning. You should think about all of the above aspects when designing an experiment. Some may be less relevant or less elaborate for

some experiments. That's fine.

Here are some thoughs on concerns that may be important in the context of Machine Learning:

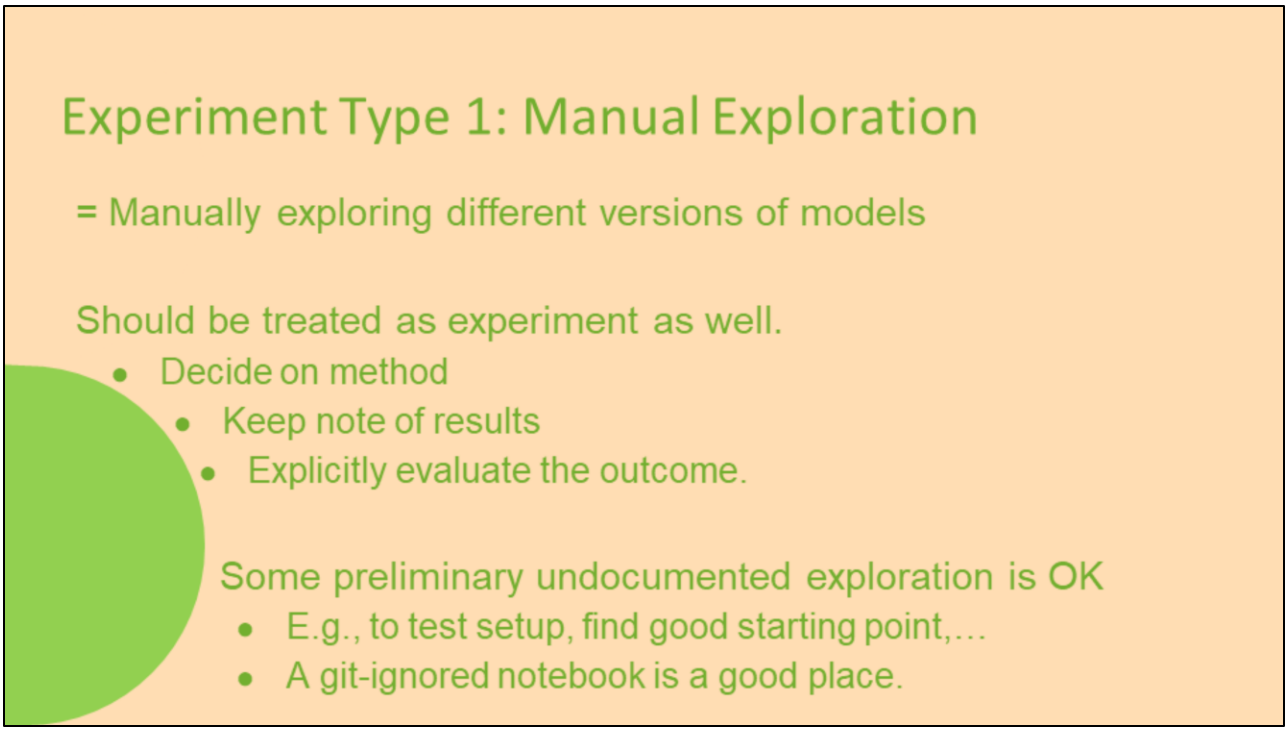
1) Purpose: The purpose of experiments in Machine Learning usually is to evaluate models or other parts of the learning process. Sometimes you may compare models, to find the best one, or the best configuration. Sometimes you may evaluate parts of your learning process, like which batch size or learning rate works best. And other times you may evaluate the outcome of the learning to determine how good your model actually is.
2) Materials: Since Machine Learning is a digital process, the required materials often come in the form of a computing platform that is powerful enough to support the learning that you want to do. If your experiments involves time measurements (e.g., if you compare the training duration of two models) then you should make sure to be very specific about which type of machine you run your experience on as the training time will have to be interpreted in context of the speed of the machine running the machine learning. The dataset could also be argued to be a digital material.
3) Methods: can differ a bit. They usually involve executing the learning process or running some other form of evaluation on a machine learning model. This is where Python Notebooks shine. They enable you to put the experiment code right into the notebook as executable cells. This way you can just show the reader how the experiment was implemented and have to do less description work. The methods can be more or less manual. We will talk about three variations of this in the following slides.
4) Data interpretation: This depends a lot on your experiment. If you want to test different candidates, it will likely consist in finding the best based on some metric. But more open-ended evaluations are also possible.
5) References: If you cite scientific publications they should be in a list of references. If you mainly point towards websites, you may be able to get away with hyperlinks in your markup cells.



A Python Notebook is an awesome environment for reporting on experiments. The extensive markup capabilities enable you to provide describing text while the Python cells enable you to provide executable code that one can use to repeat the experiment.

A good default structure is shown in the slide. It is OK to deviate from this structure if you have good reasons. But generally, you should produce a document that:

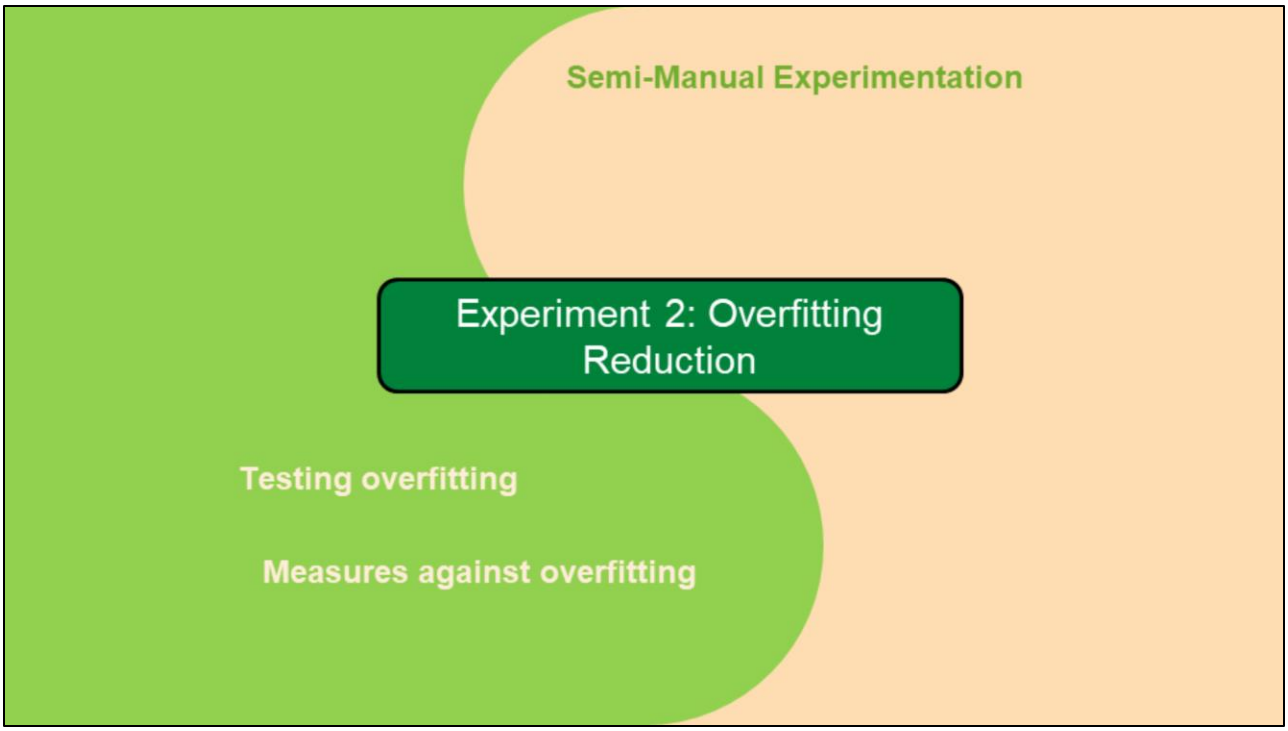
- let's the reader understand the purpose of your experiment
- Let's the reader understand which procedure you followed with sufficient details to repeat the experiment
- Describes and discusses the results with a clear distinction between the results and your discussion.
- Explicitly formulates the conclusions you draw from the experiment.

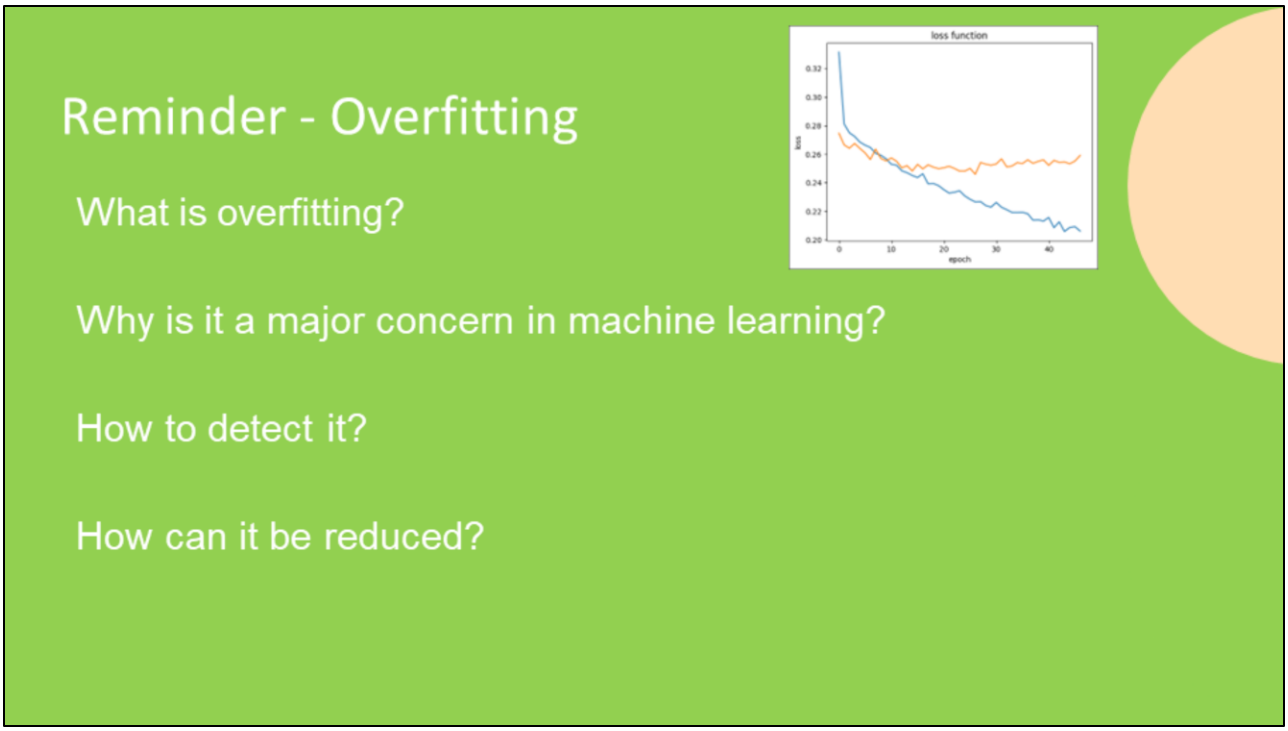


The experiment we performed for model selection is a very manual one. While we followed a procedure, the main method was basically “we let the researcher decide which models to try out”. This is sometimes fine. Especially, if we are talking about a pre-filtering experiment like the model selection experiment, where the actual best model does not matter as much as getting a good picture of the performance of different model categories that enables us to choose between them.

Nevertheless, you should still treat these types of manual explorations as an experiment, meaning you should plan them beforehand and decide on how you want to conduct them. And you should document your results and thoughts. If you don’t you may end up in a “what-the-hell was I even thinking” type of situation which makes it hard to argue for or double-check the conclusions you arrived at.

This doesn’t mean you need to treat all of your playing-around with models as an experiment. It is usually a good idea to do some sanity-checks to make sure that everything works in principal and integrates well before you do the more rigorous experiments. A good practice to have for those types of checks is to have a dedicated notebook that you add to gitignore, so it doesn’t accidentally land in the repository where it will likely confuse people.


Semi-Manual Experimentation
Experiment 2: Overfitting Reduction
Testing overfitting
Measures against overfitting



Our second experiment will center around overfitting.

If you look at this project, you should already have first experiences with Machine Learning, including overfitting. Let us remind ourselfes of this topic.

Overfitting happens when your machine learning model learns structure that is present in the training data but does not generalize to the test data. This usually means, it will achieve better scores on the training than on the test data. It can be recognized by comparing training and test scores. If you have an epoch-based model, like a neural network, it can often help to plot the score development over time. In the example image in the slide, the training and test score start to deviate around epic 10. At that point the model starts to learn relations that are not generalizing well to the test set.

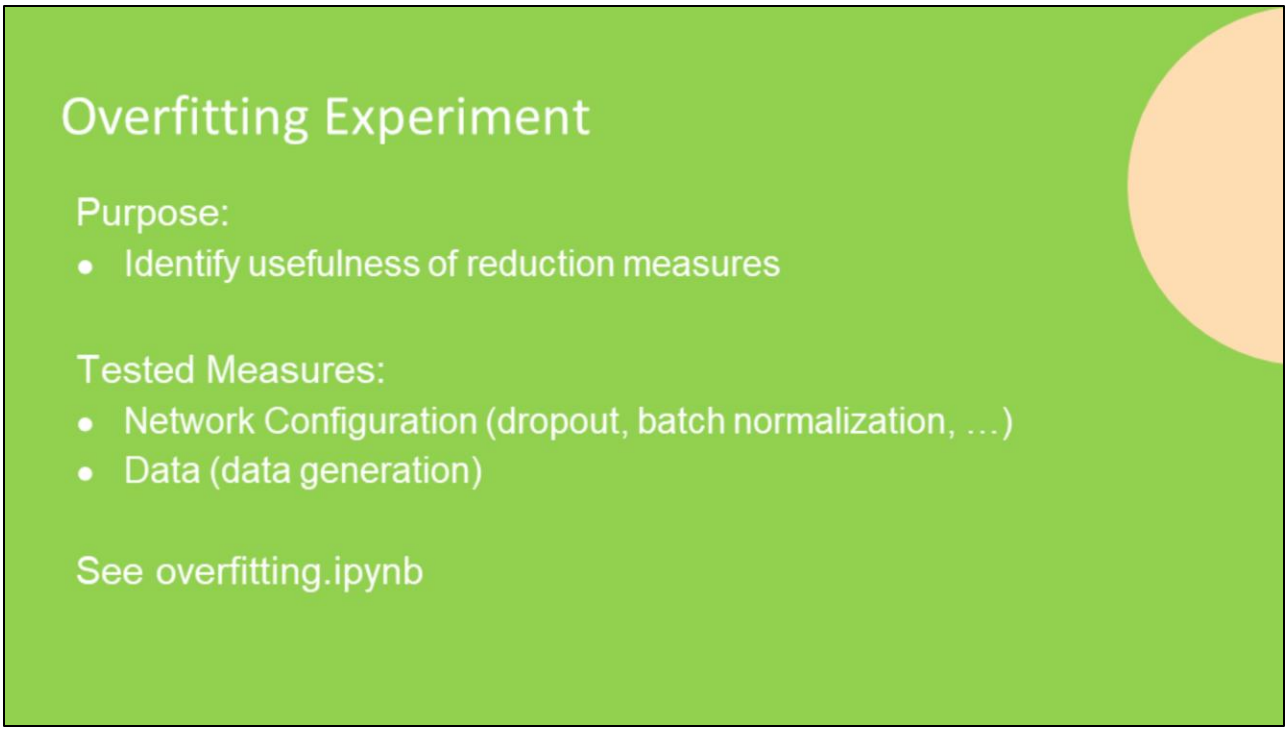
Overfitting can be a major concern in machine learning, since we are often more interested in the performance on the test than on the training set.

Reducing overfitting to zero is not always possible. However, there are certain things you can try:

- Reduce the complexit of the model: Overfitting often happens when the model is complex enough to learn arbitrary structures present in the noise of your training data. Such structures are often more complex than the actual learned relations. By getting your model to a point where it is complex enough to learn what is supposed

to learn but not complex enough to learn noise, you often can reduce overfitting.

- More data: Overfitting can also happen if your training data doesn't adequately cover the space of possible data. Simply put: if you don't give the model data with n > 1000, it cannot reliably learn how it is supposed to behave when predicting values for n > 1000. Adding more data may help this issue, although this is not always possible. If you can't acquire more data points directly, you may be able to get more by extracting it from other ata sets or by artificially embellishing your current data (e.g., flipping images) to introduce more variance.
- Training configurations: there are certain training configuration parameters, like dropout and regularization, that are intended for helping your model generalize. These are worth testing out.

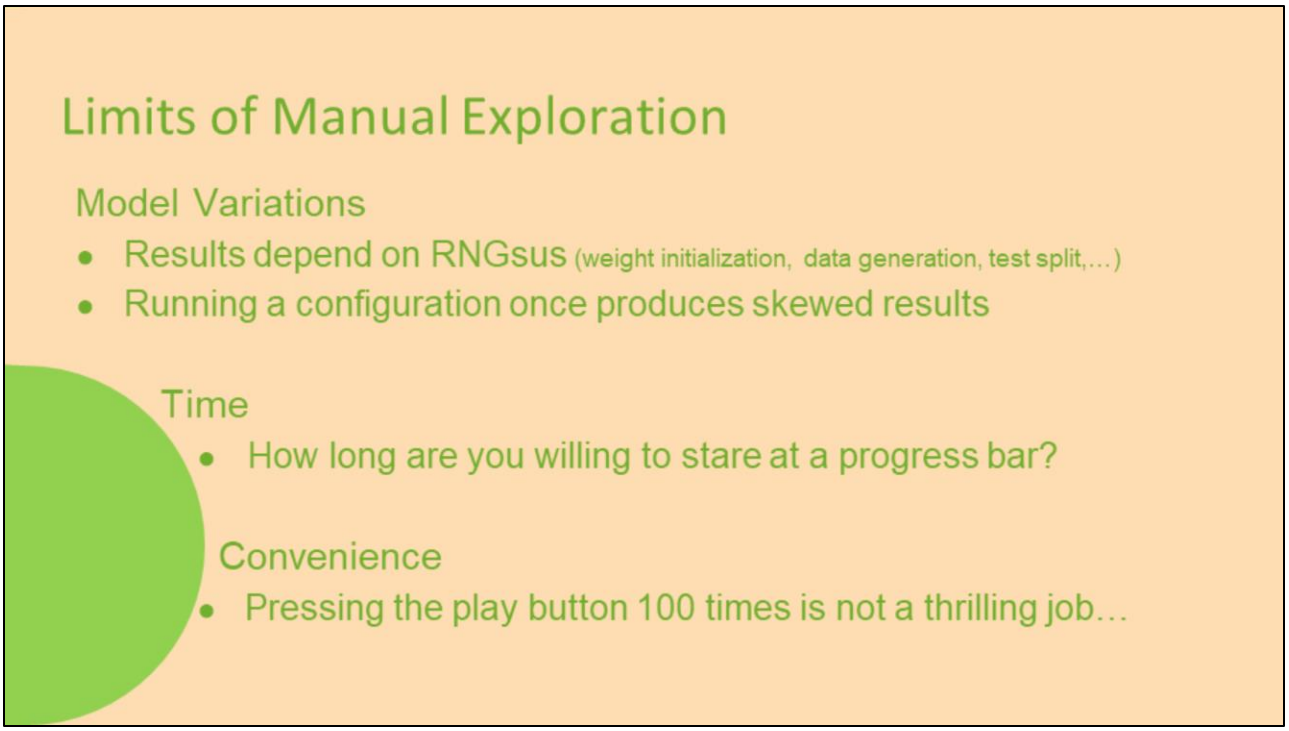


One observation from the model selection experiment was that overfitting seems to be an issue in our running example.
Given that we only have only one image per Pokemon and less than 1000 images overall, this is not a surprise and likely caused by having too little data.

However, it is worth it to spend some time on an experiment to see how much we can reduce overfitting. In this experiment we try to test different network configurations that can reduce overfitting and embellishing our data via data generation.

The experiment can be seen in the notebook *overfitting.ipynb*.

As a result we identified that data generators reduce overfitting slightly and that certain configuration parameters can be used. This gives us a good training configuration for future experiments.

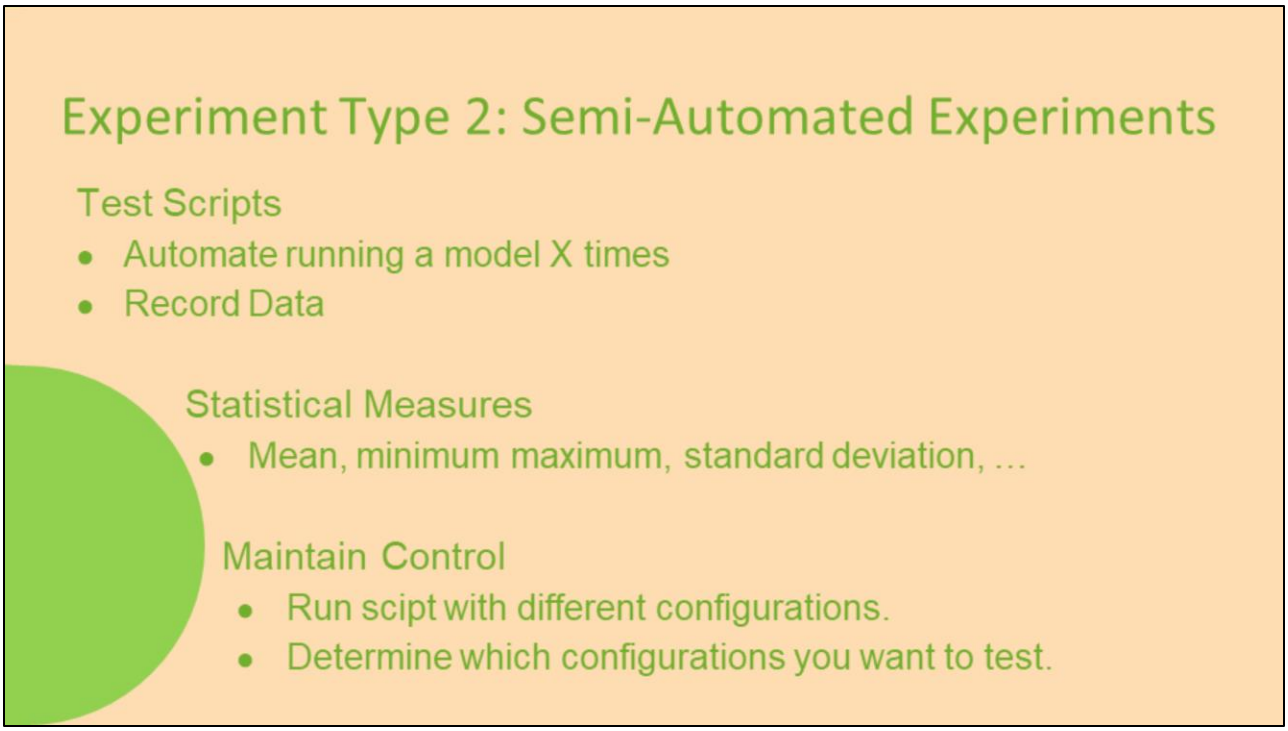


On the formal side, the overfitting experiment can be counted as a semi-manual experiment.

There are certain drawbacks to doing everything with manual exploration.

Model variation is the first. Machine learning models tend to be dependent on random number generation as there are several processed during the learning that are governed by random numbers. This includes the initialization of weights, the generation of data and the splitting of training and test sets. This means, if you start the learning from two different random states you will get two different results. They are likely somewhat similar but can vary. This means, if you do manual exploration, based on single test runs you run the danger of making decisions because one configuration seems better, even though it was just got lucky random number generation. Reducing the effect of randomness usually involves running experiments multiple times.

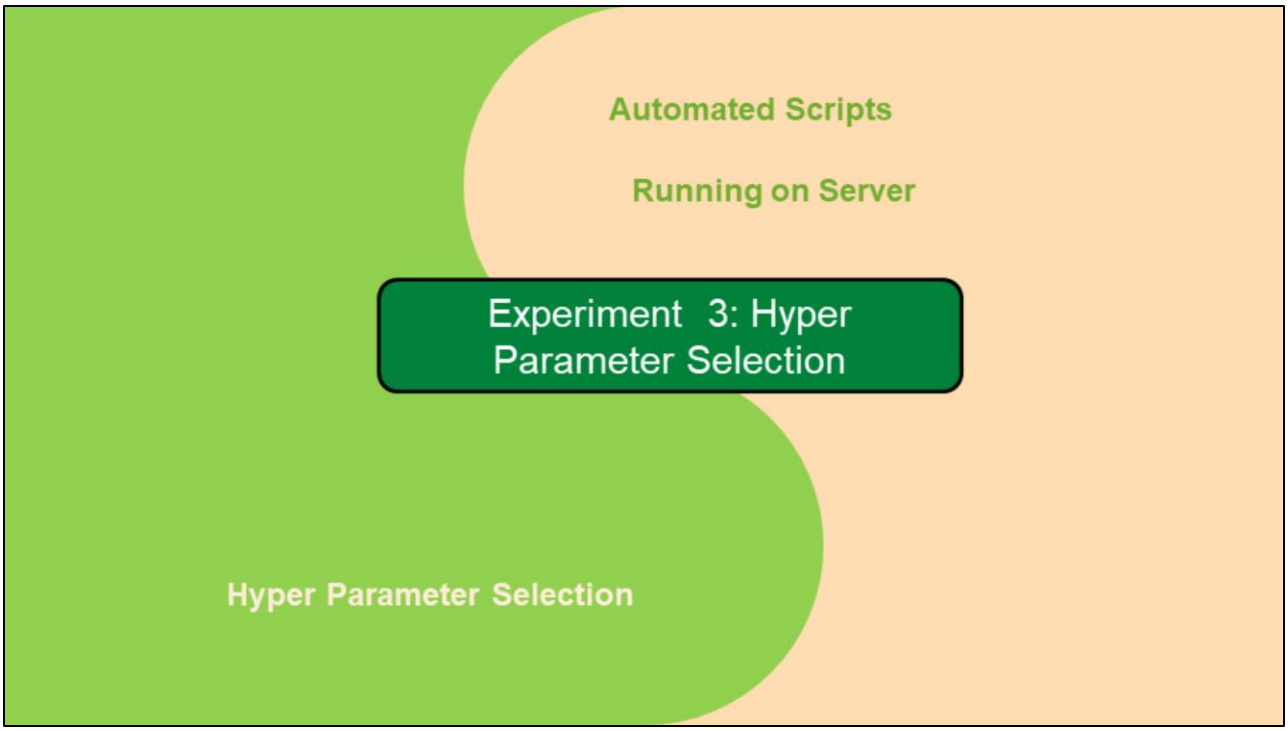
Time consuming learning is another thing that can mess with your ability to do random exploration. If each test run takes half an hour or even longer it becomes harder to keep your train of thought while experimenting, making it harder for you to guide the experiment process. This also translates to convenience. If you find that the main thing you're doing is to press play and write down results, then it may be time to think about automating that job.

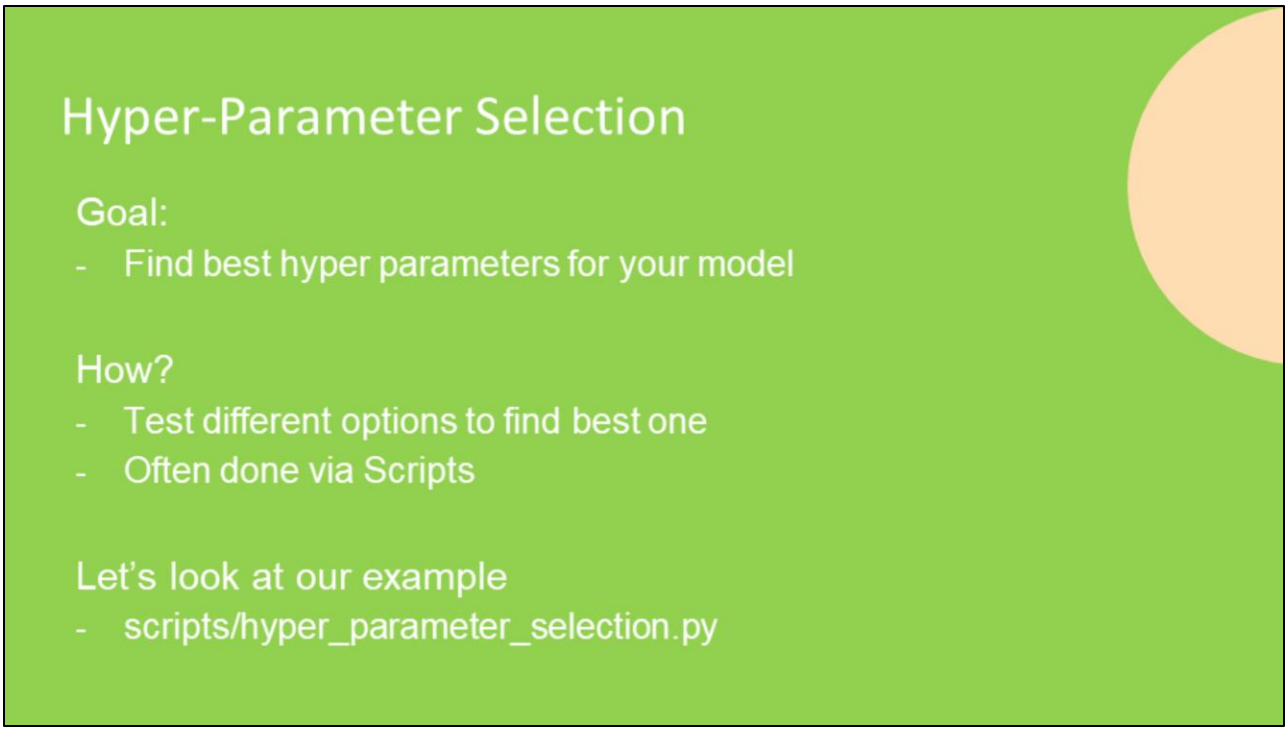


Our overfitting experiments can be seen as being semi-manual. We've defined the class MultiRunEvaluation which runs the same experiment ten times and records statistical measures for the hamming score. This means, we can automate the boring task of pressing play ten times to counteract the randomness.

The experiments still are somewhat manual as the different things tested are still configured and designed by a human. But some of the sub-experiments have the potential to be automated completely.

Semi-automated tests can be a good tool for a situation in which you want to run a more statistically sound analysis but still want to maintain the ability to make decisions while you are carrying out the experiment.


Automated Scripts
Running on Server
Experiment 3: Hyper Parameter Selection
Hyper Parameter Selection



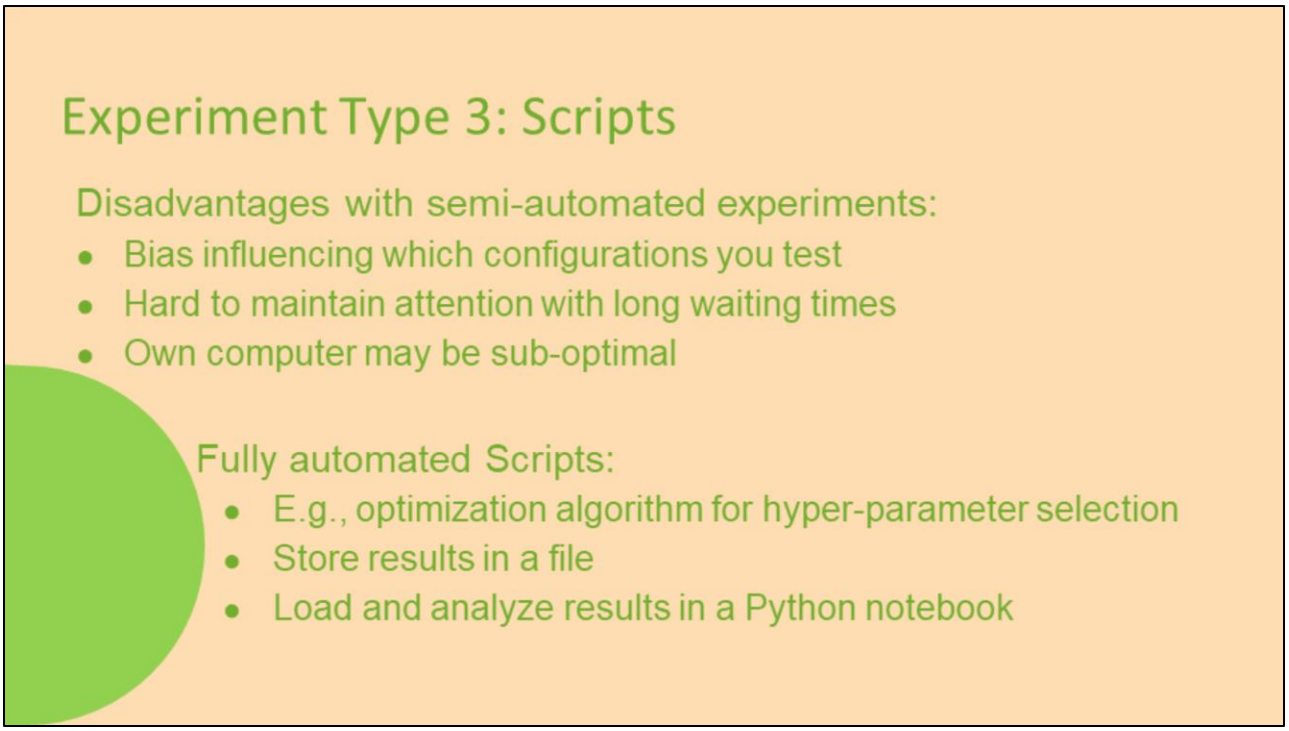
Our third experiment centers around Hyper-parameter selection.
In previous experiments we have already decided to use Convolutional Neural Networks and decided to do training with Image Generators and certain additional settings. What remains is to find out which network configuration performs best with this setup.

This is generally the goal with hyper-parameter selection. To find the hyper parameters for which the network performs / generalizes best.

In our running example, we will model the network structure as two hyper-parameters:
- CNN: How many convolutional layers the network has and how many patterns per layer.
- ANN: How many fully connected layers the network has and how many neurons per layer.

In this experiment the results matter a lot since we will use the best result as our final version of the model. This means we want it to be statistically sound, which means we will not only run the experiment 25 times for each configuration but also run each configuration on five different splits of the dataset. To find the best configuration we will employ grid search.

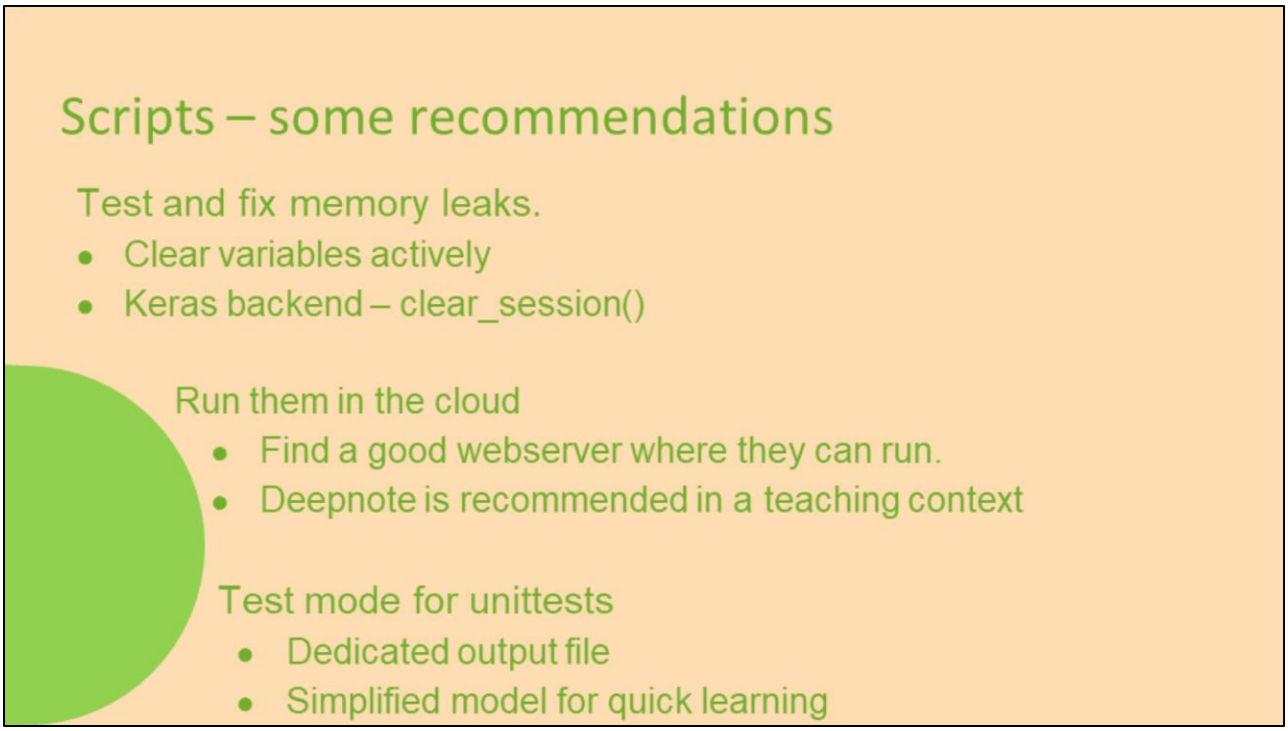
The experiment summary can be found in notebook *hyper_parameter_selection.ipynb*



Due to the big scope, our hyper parameter selection was implemented as a fully automatic script.

There are several disadvantages when dealing with semi-automatic experiments or manual experiments. The first one is potential bias or human errors which may lead you to focus on certain configuration because you perceive or expect them to be better. In addition, experiments with longer wait times tend to be harder to perform, especially if done on your own computer. If you have to wait multiple hours for each run, it is harder to keep track of what you already have tested / still need to test.

Fully automates scripts can be the solution for this. This means your manual selection process is usually substituted by an automated algorithm (in our example by the optimization algorithm Grid Search) that can be run without human supervision. This makes it possible to run the experiment as a longer script and document results in a file. The corresponding Python notebook usually then loads the results from the file and interprets them.



Some recommendations for if you implement scripts can be found here.

Once you run a machine learning experiment dozens or even hundred of times - especially if you try to get away with minimum server costs - , you may find memory leaks to become an issue. Just deleting your local variables can help but you should be aware that machine learning frameworks such as Keras maintain internal memory that can accumulate. They usually offer a way to reset this memory. In case of Keras this is the method *clear_session* from the Keras backend.
Generally, it is a good idea to track how much memory your Python program is using between experiments – especially while you're still debugging the script and searching for memory leaks. You can see an example for this in the Multi_Run_Evaluation where we first clean up memory by deleting variables and calling *clear_session* and then showing the currently used memory via *process.memory_info*. It is a good idea to trigger an active garbage collection before looking at memory (*gc.collect())* to assure your memory measurements are not influenced by when the garbage collector runs.

Computation heavy scripts are very often run in the cloud. There are plenty of web-servers and you should find the one that suits you. It is usually recommended to have at the very least 16 GB or ram for Machine Learning / Data Science projects. There are also servers with a dedicated GPU, should you need them. For our tests we used Deepnote. While deepnote is normally an online development environment, it also includes access to a server. The education rates are quite generous, so within an

education context – and if your machine learning project is of smaller scope – you may get away with using those.

Another thing that is strongly recommended is to have these scripts in your automated tests. Nothing is more annoying than running a script for two days just for it to crash because of an error during result storage. Keeping you scripts in unittests helps you find those issues early. Since these scripts can run quite long, you will probably have to put in some work to make them run at a reasonable amount of time during unit tests. We've found the following helpful here:

1) Using a smaller machine learning model and dataset for the unittests and configuring the project to only run a very small scope grid search with four states reduced the run time greatly.
2) Setting a dedicated test folder for unittests enabled us to run the scripts in test mode without overwriting any actual results

Homework

Homework:

Please try to find good hyper parameters

- Please use notebook manual_model_selection.ipynb
- Change the “Model Creation” cell to find a good model
- Post the cell content into this Google Form (link)

There will be a trophy for the best solution next week ☺

Questions?