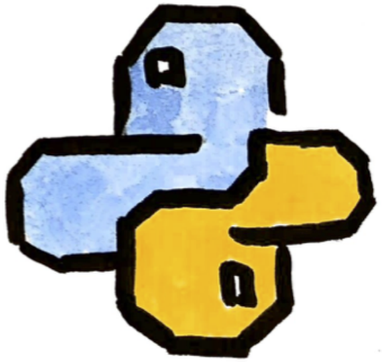
END 2 END DATA SCIENCE

Mark Keinhörster &
Dr. Shirin Glander

Convolutional
Neural
Nets

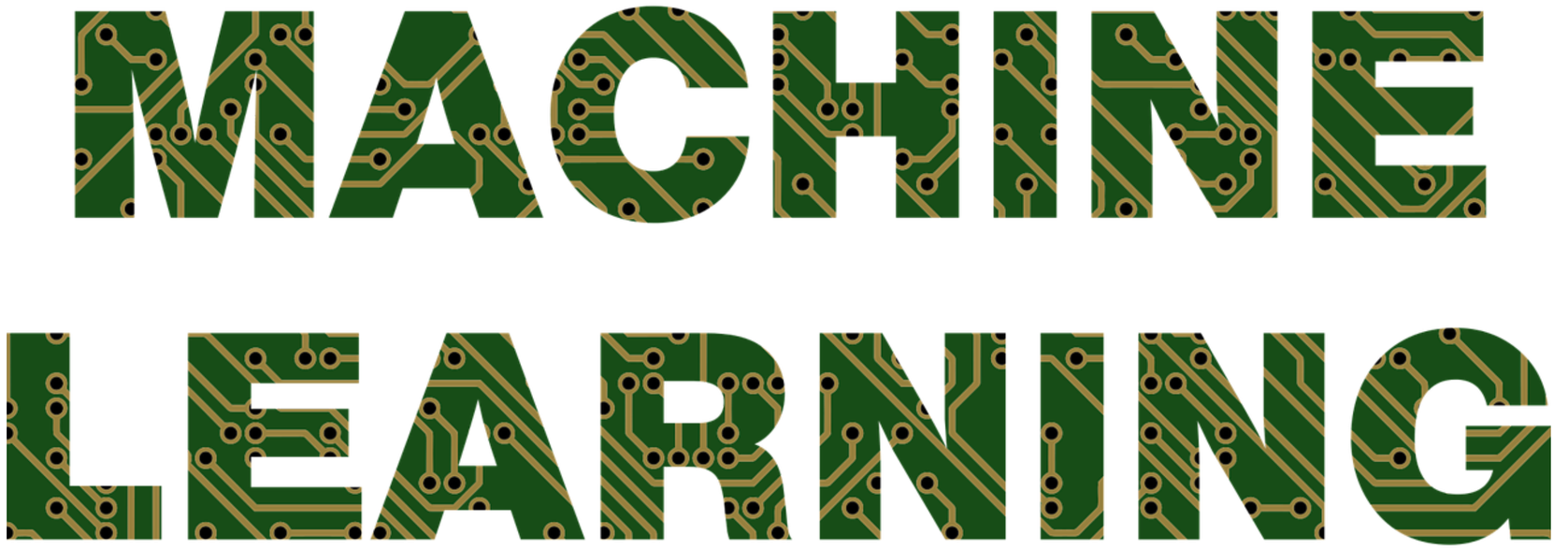


Computer
Vision

About this Workshop

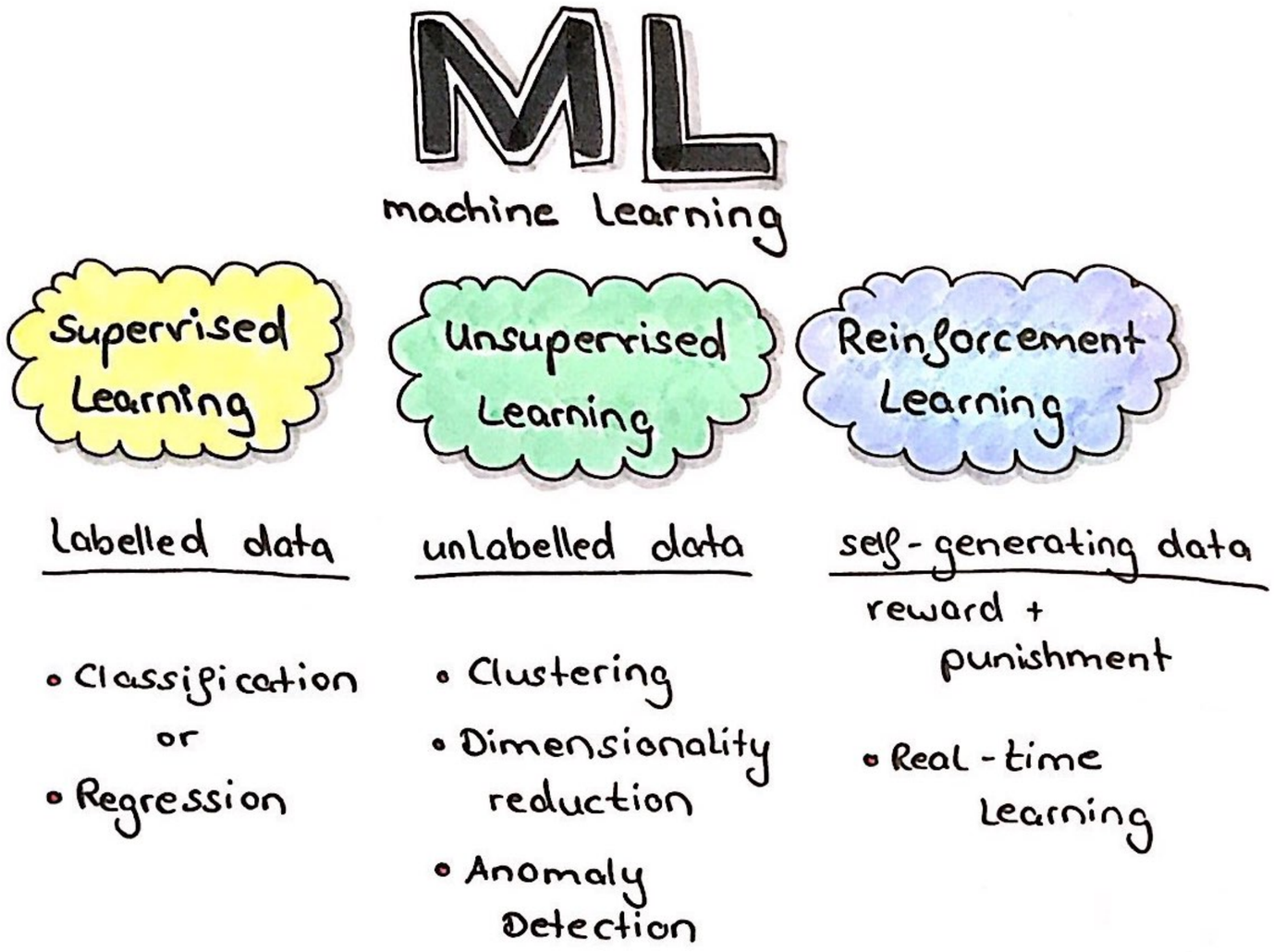
- Learn what neural nets are and what the big deal is with deep learning
- Interactively build a model that differentiates between fruits on images
- Get a glimpse of production-readiness
- Learn about Luigi pipelines and their main components
- Write your production ready pipeline
- Get an overview of luigis modules
- Get an overview about DVC and how to implement pipelines
- Use TensorFlow Serving to deploy your model

About us

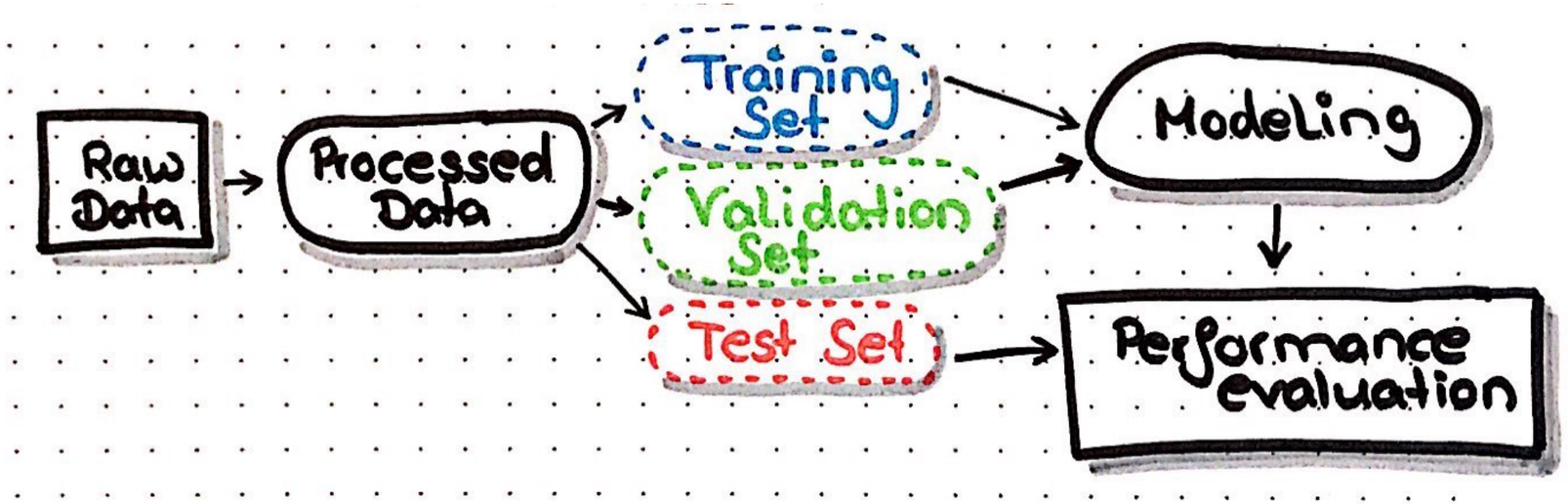


https://pixabay.com/de/a-ich-ai-anatomie-2729781/

What is Machine Learning (ML)

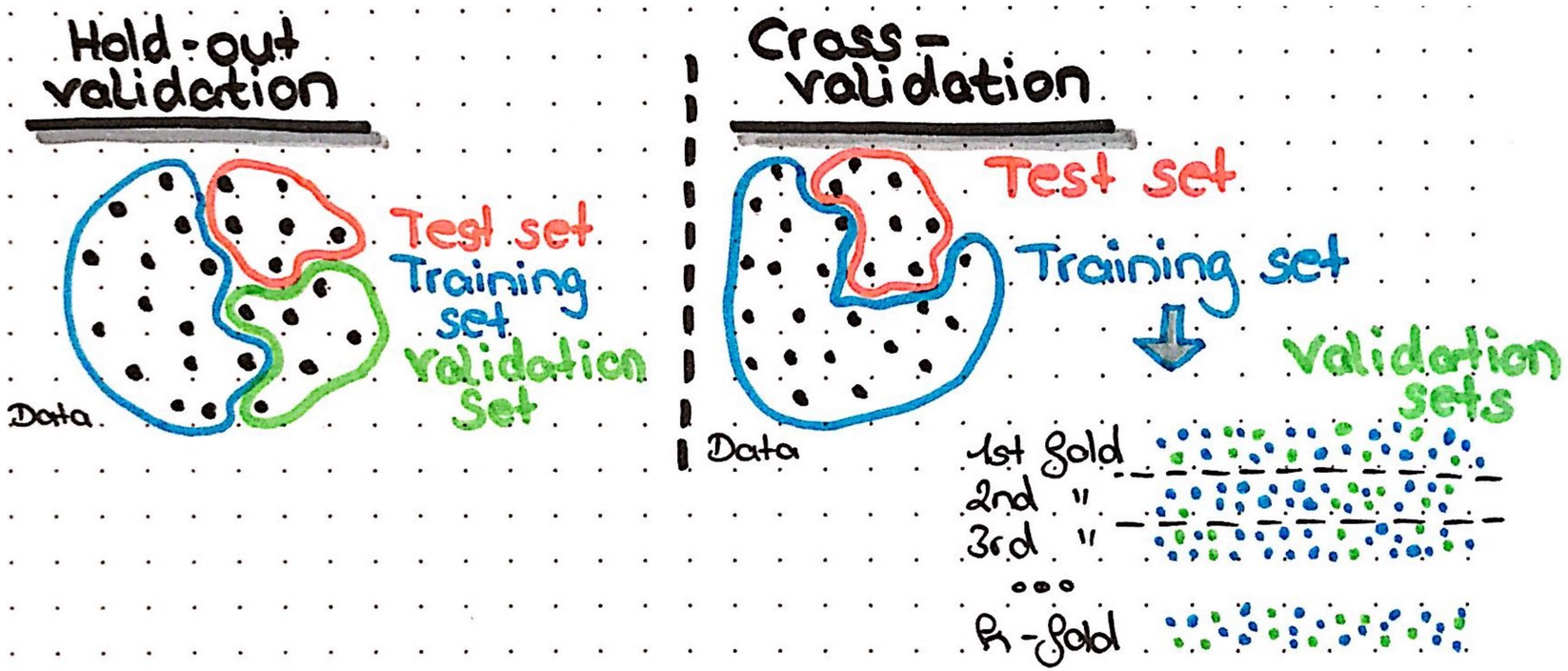


A typical ML Workflow

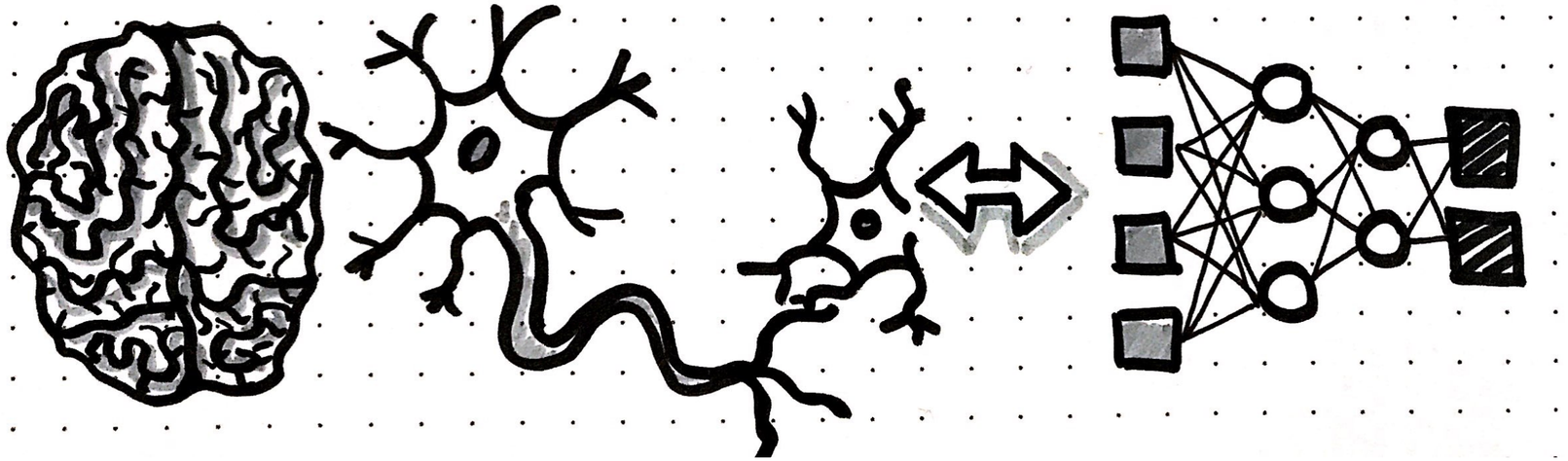


Why use validation and test data?

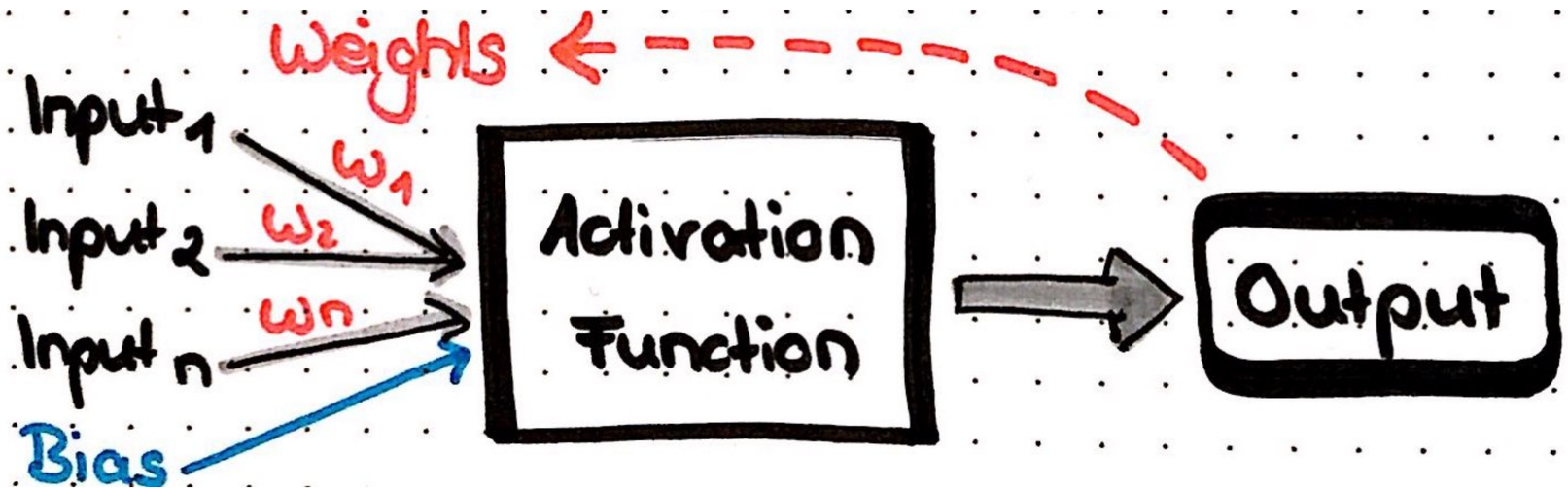
Validation methods



What are neural nets?

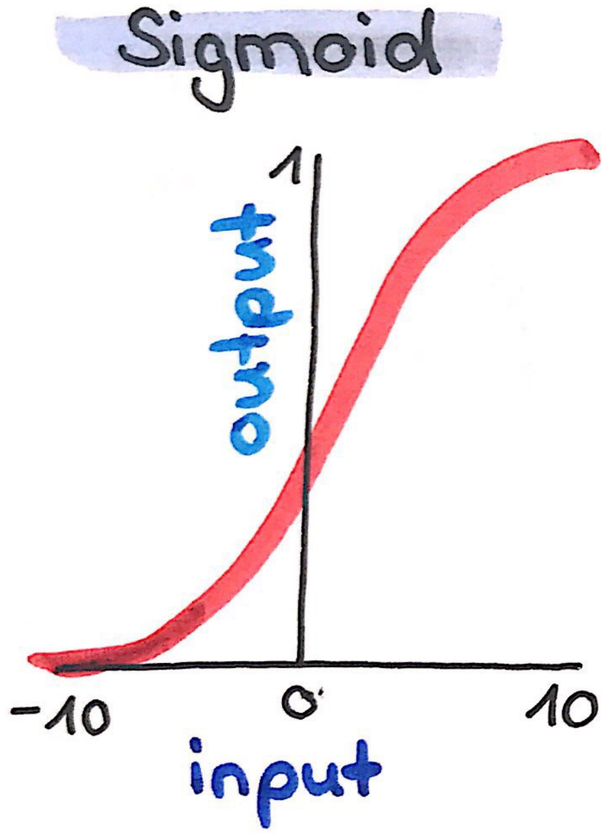


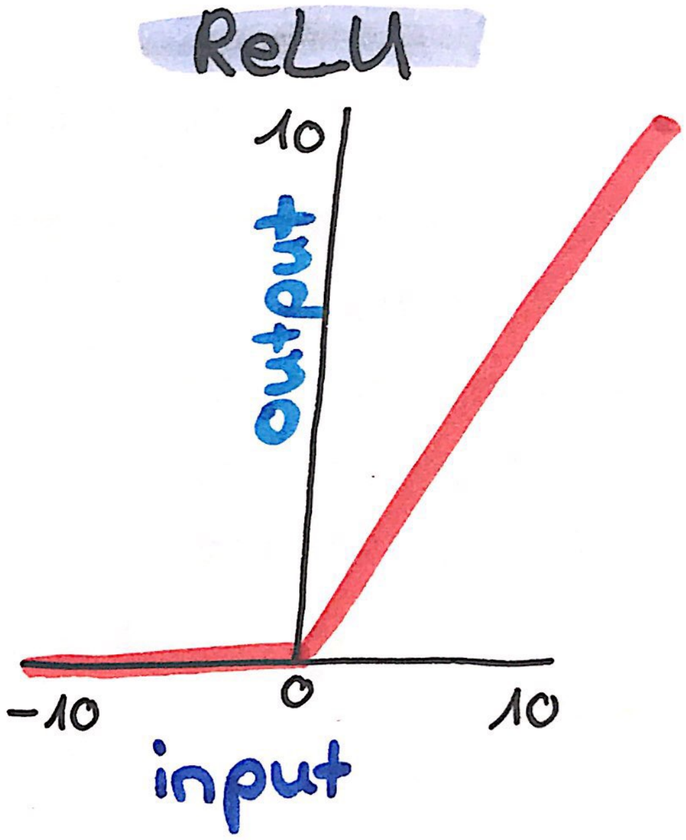
Perceptrons



Activation functions normalise input

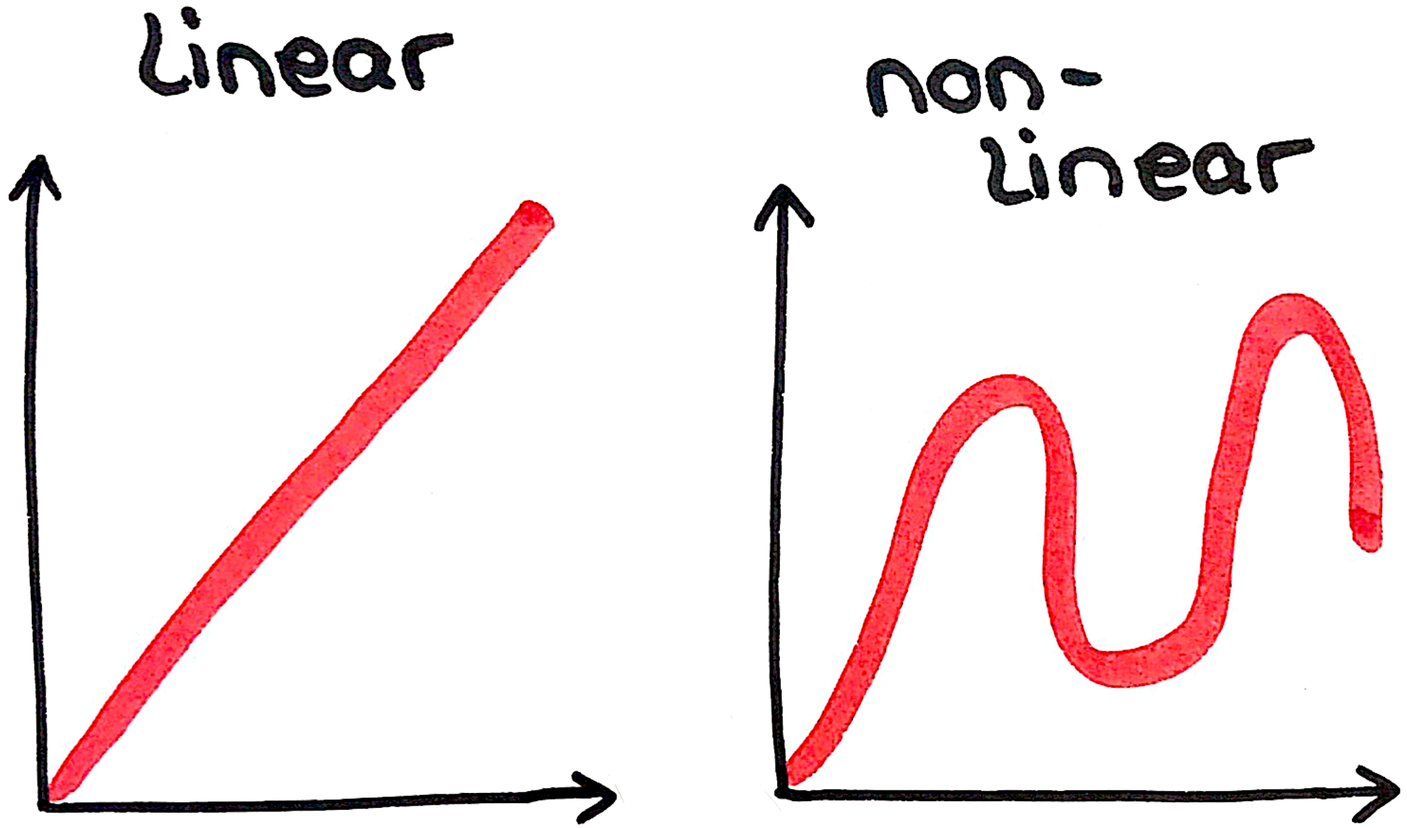
- every problem can be described with a mathematical function



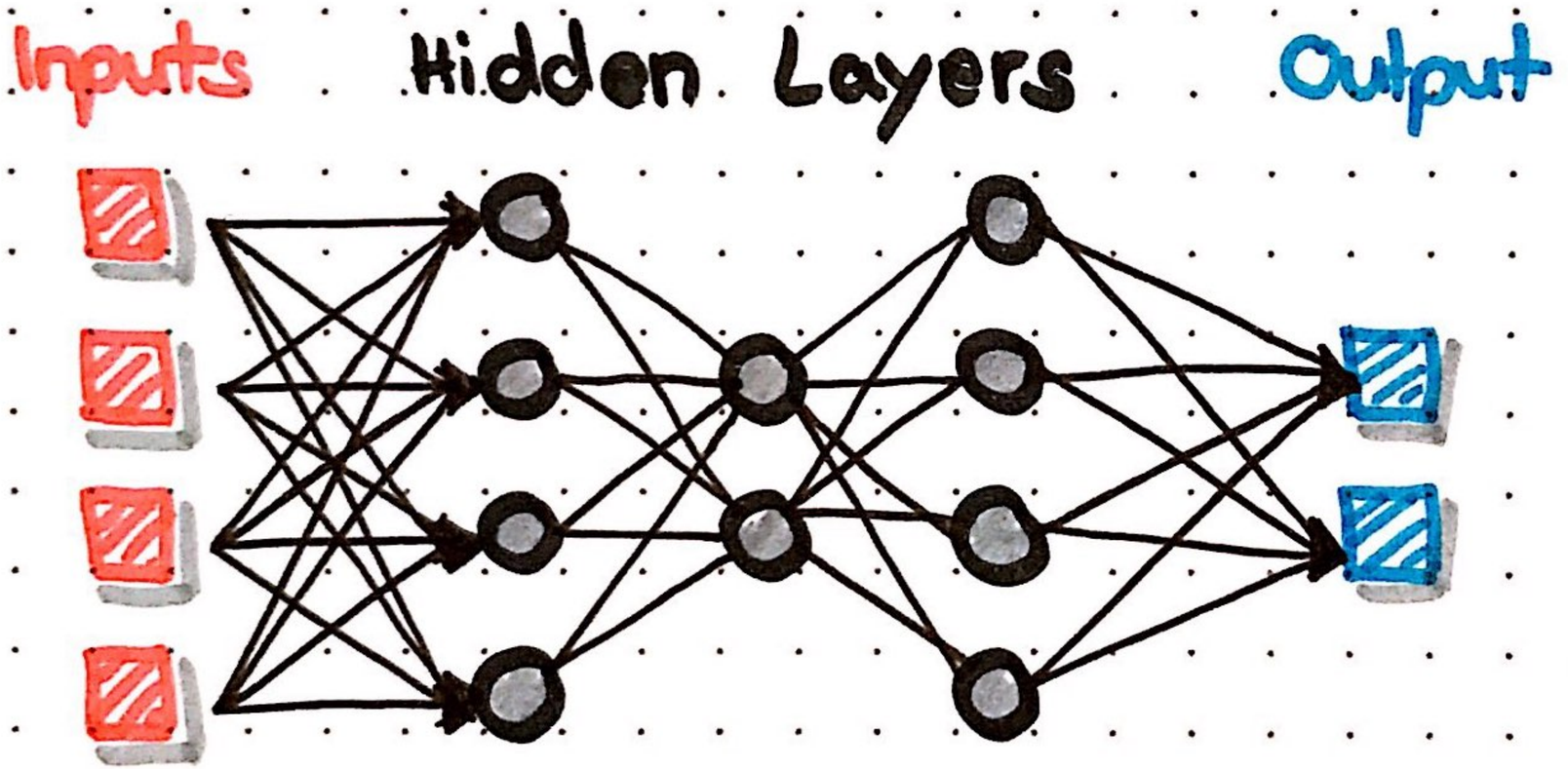


Activation functions make non-linearity possible

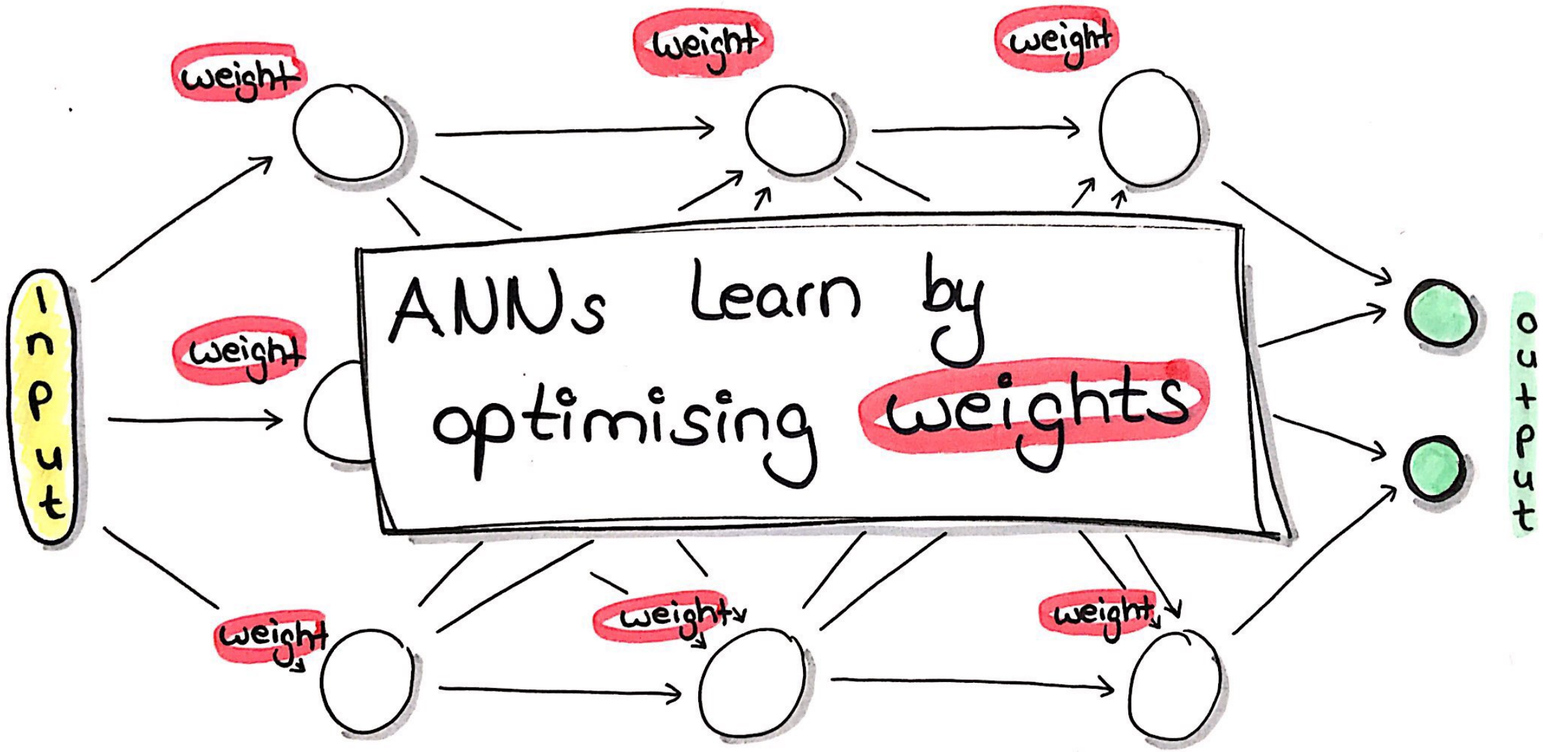
- non-linear activation functions allow us to approximate ANY mathematical formula with neural nets



Multi-Layer Perceptrons

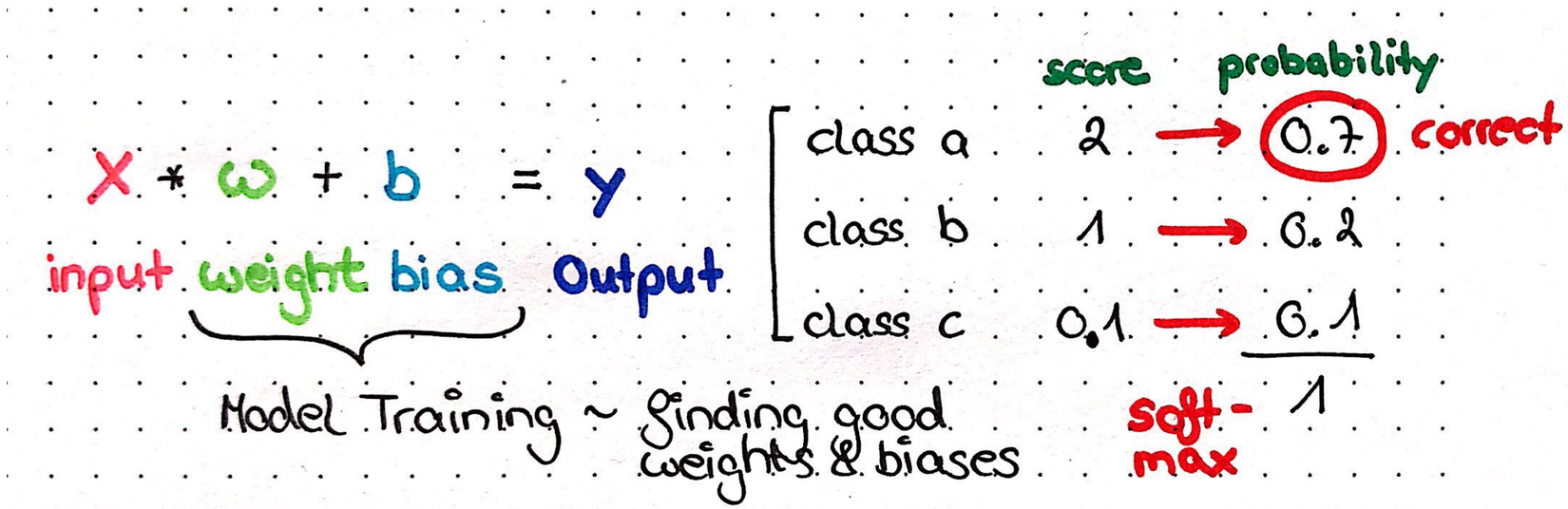


How does a neural net learn?



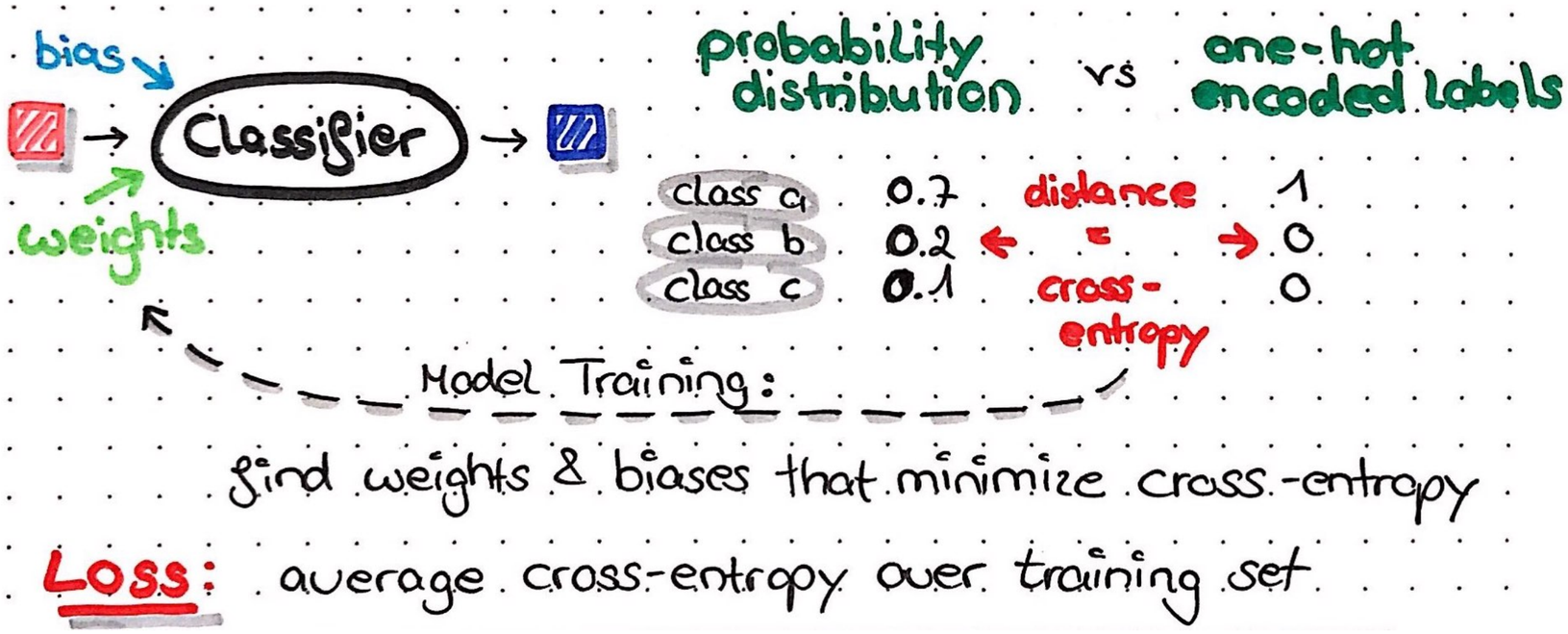
How does a neural net learn?

The Softmax function



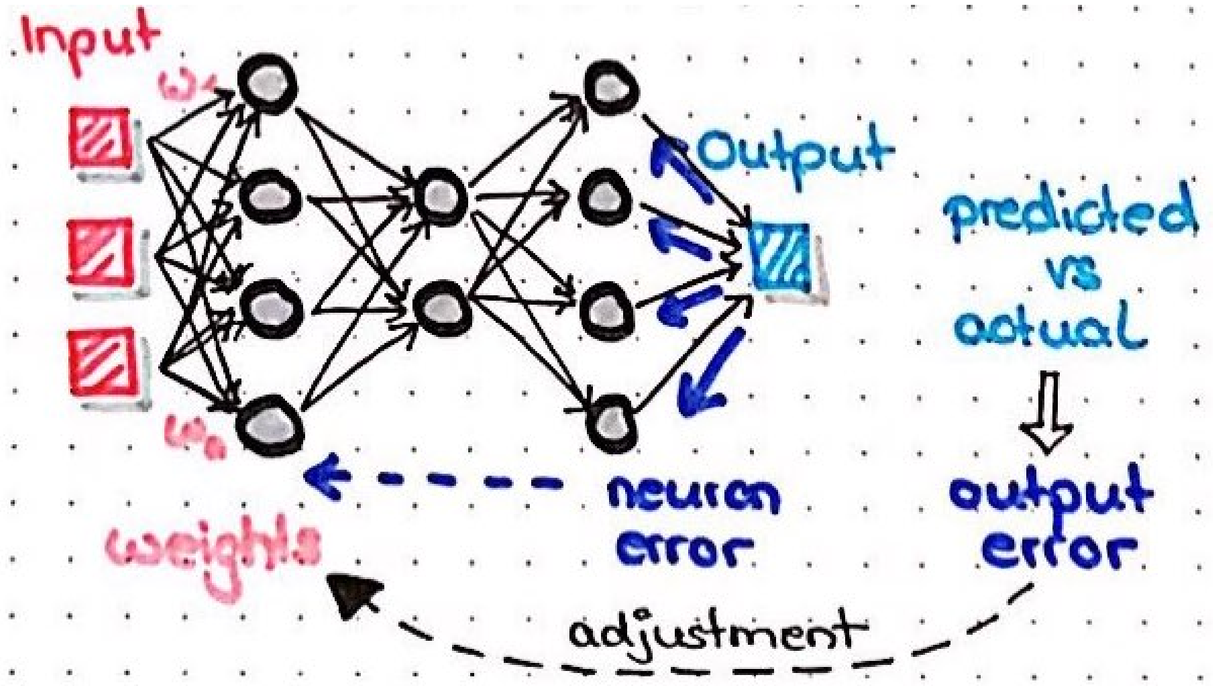
How does a neural net learn?

Cross-entropy



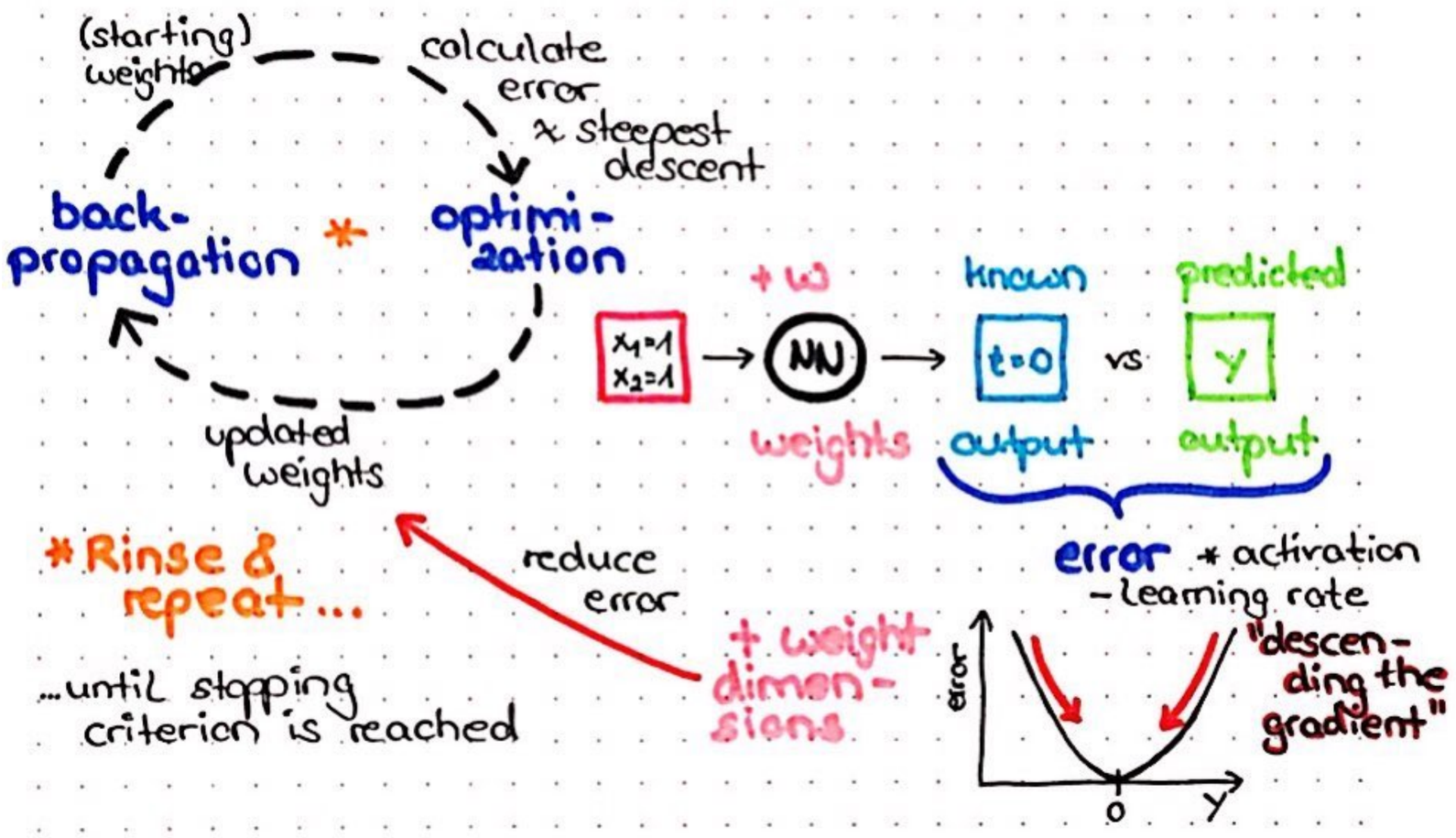
How does a neural net learn?

Backpropagation

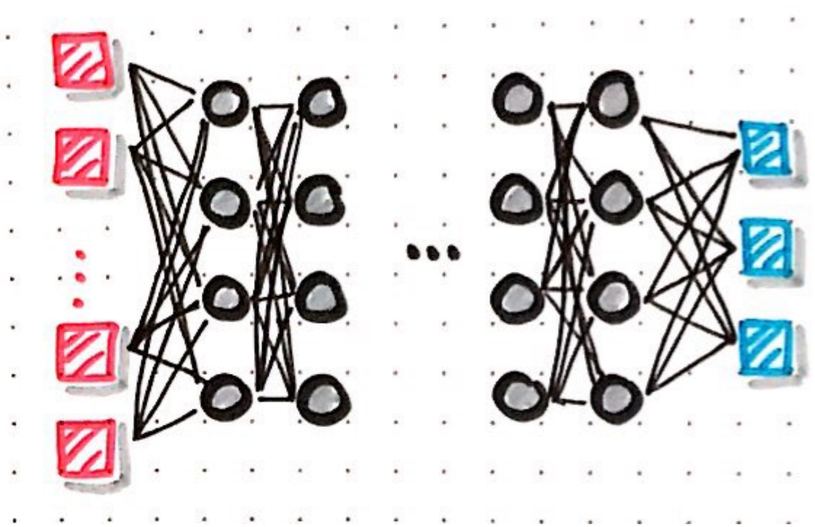


How does a neural net learn?

Gradient descent optimization



Deep Learning

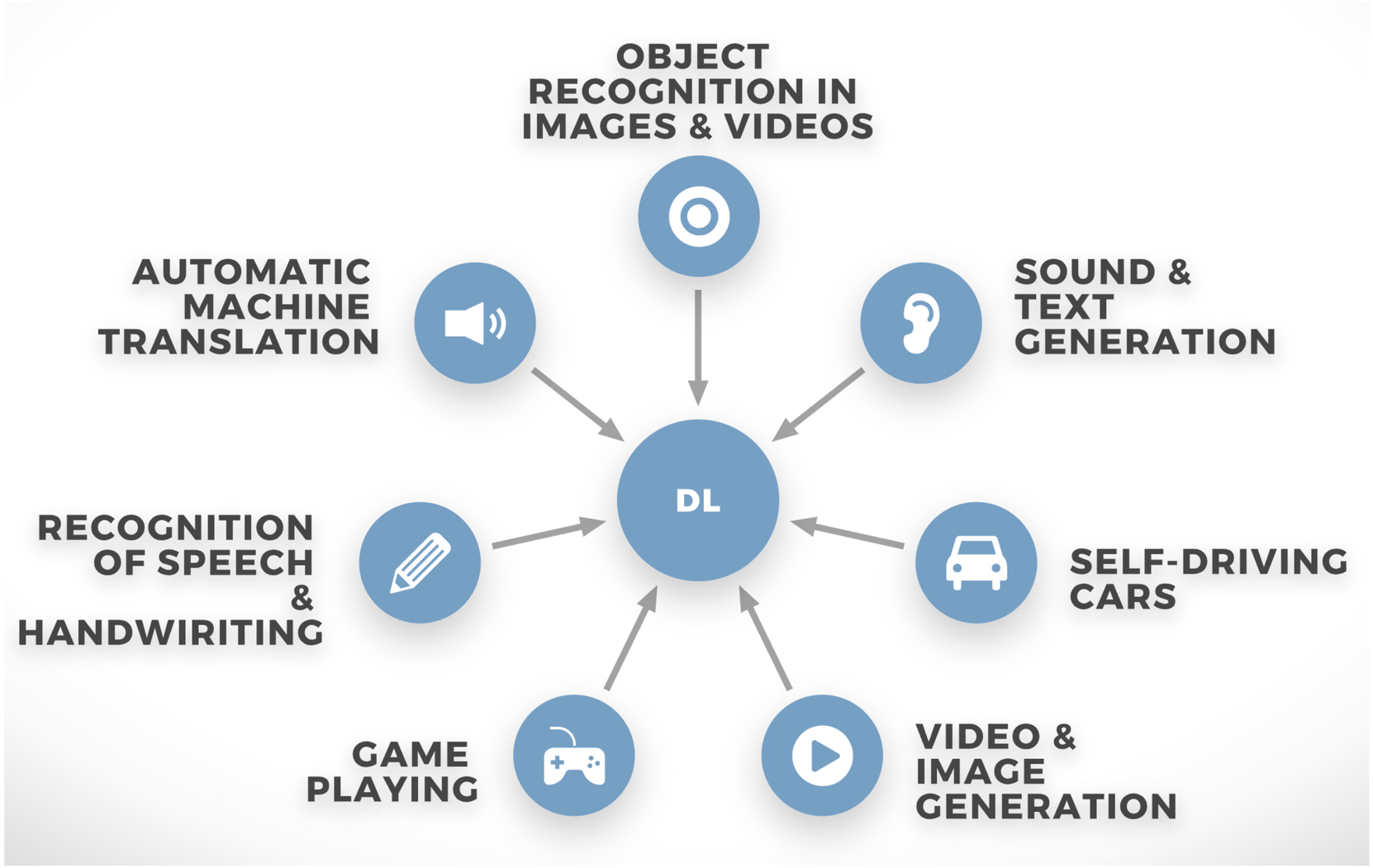


- supervised, semi-supervised or unsupervised

- NLP, speech recognition
- image recognition
- object classification
- recommender systems

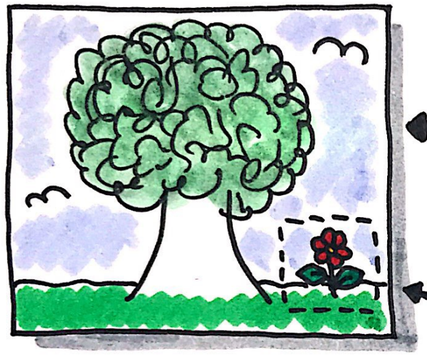
etc.

Deep Learning in the wild



How does a computer learn to "see"?

Convolutional
Neural
Nets



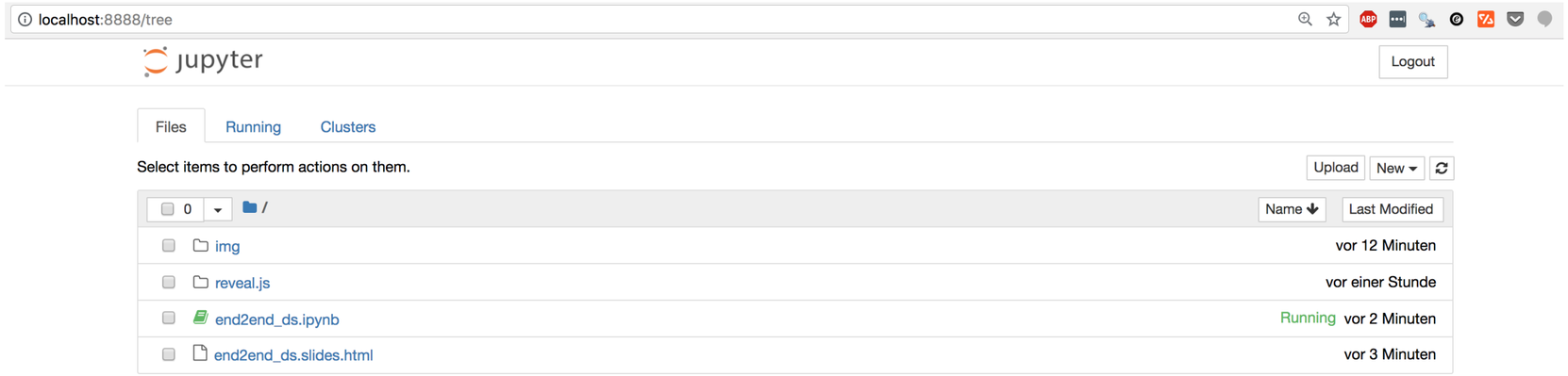
- image classification
  class = tree
- object detection
  flower

Computer
Vision

How does a computer learn to ”see”?

Jupyter Lab and Python

- i(nteractive) Python
- browser based notebook
- code + markdown + output



Jupyter tips'n'tricks

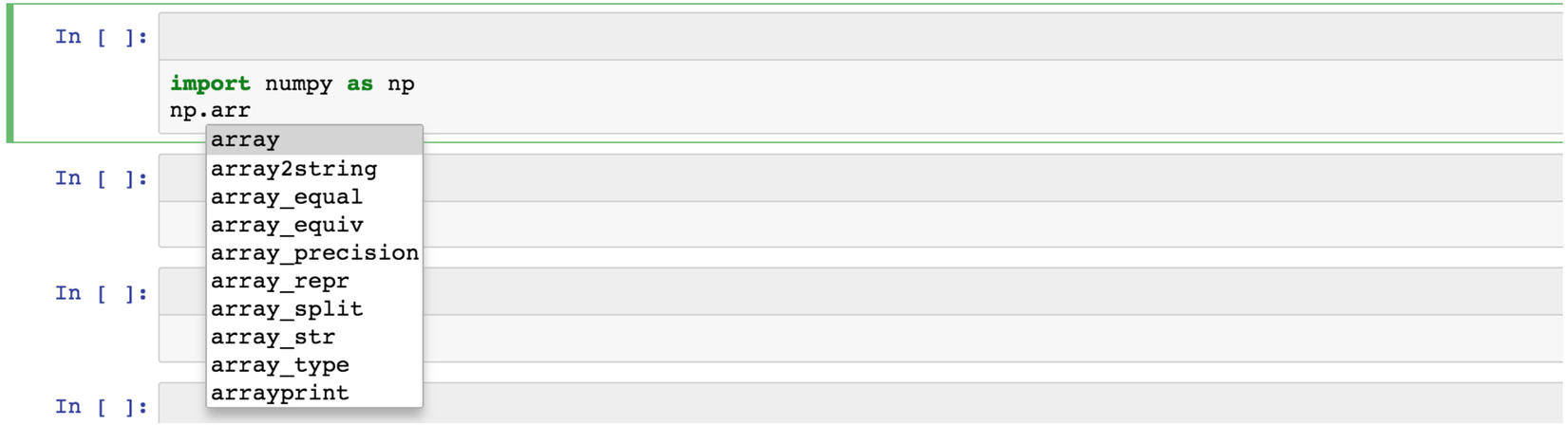
- ?: search documentation

```
In [5]: len?
```

- ??: look at source code

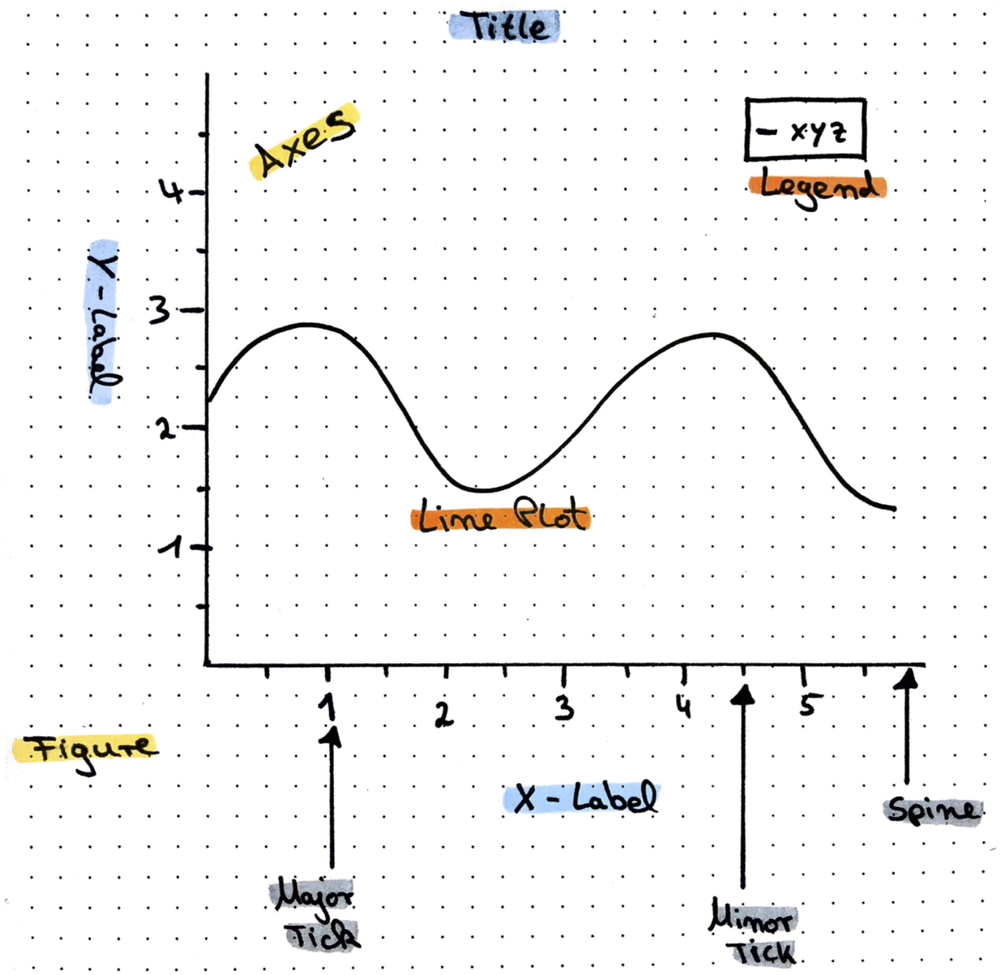
```
In [6]: len??
```

- tab: autocomplete
- shift + tab: look at function call



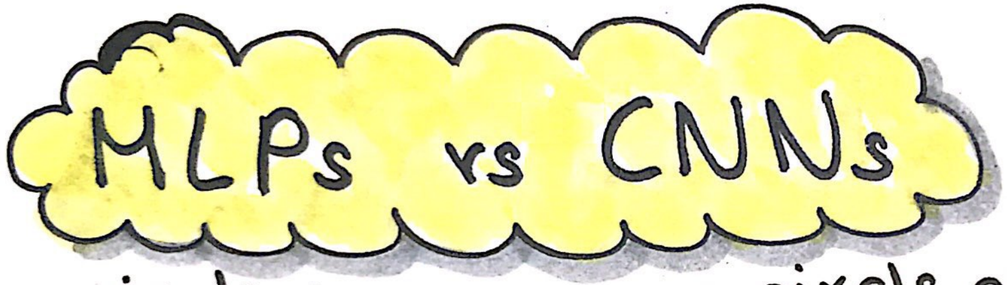
Visualization with Matplotlib

- Matplotlib is a Python 2D plotting library
- Can be used in Python scripts, Python/IPython shells, Jupyter notebooks



Matplotlib

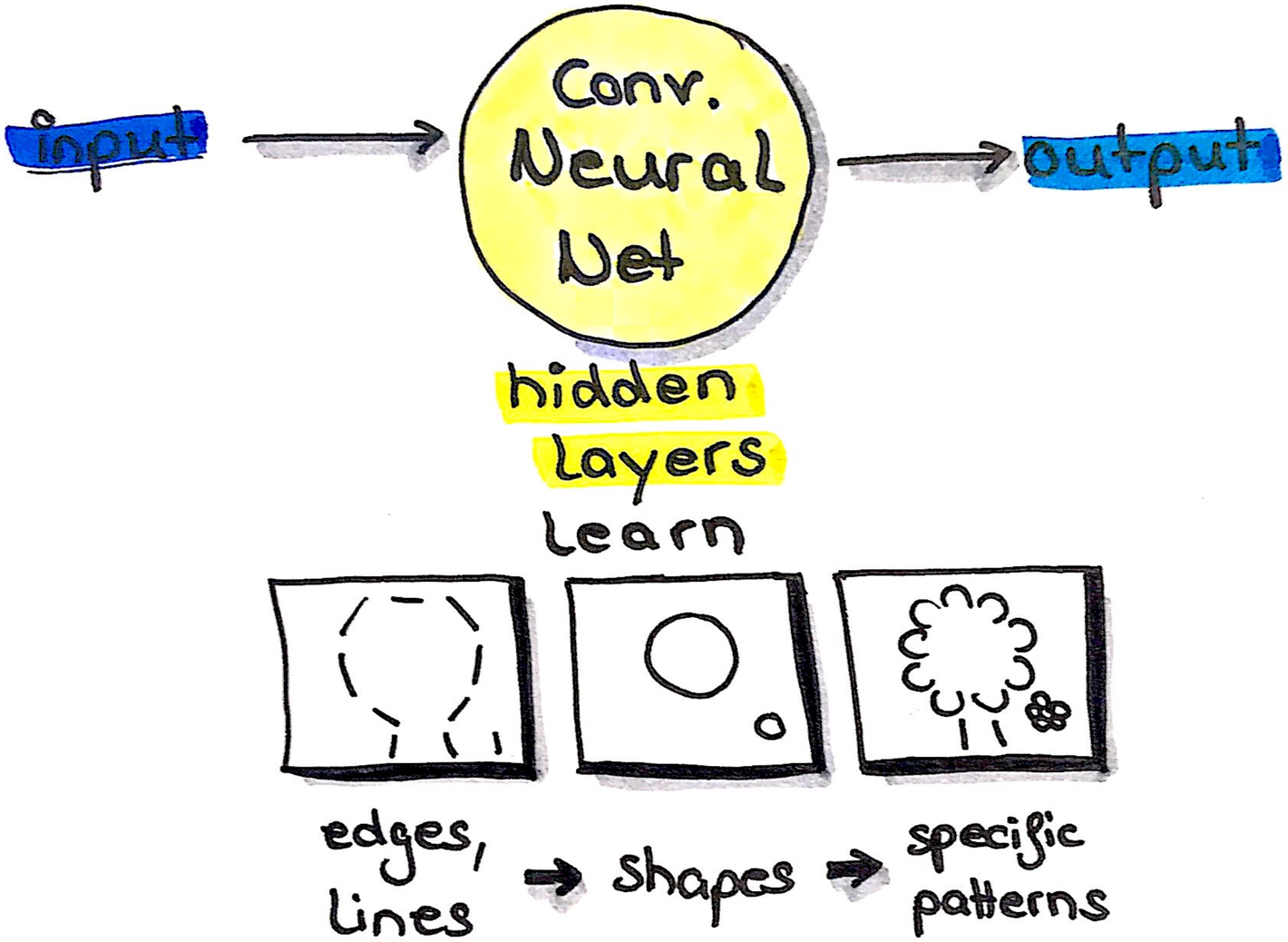
Convolutional Neural Nets



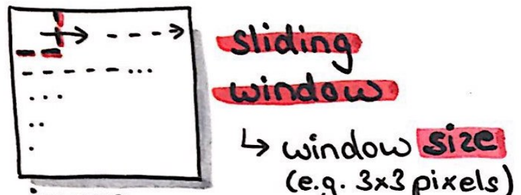
- pixels are considered independent
- computationally faster
- Learned : weights

- pixels are considered as groups of connected information (context)
- analysed as chunks (= windows)
- Learned : filters

ConvNets


input
Conv.
Neural
Net
output
hidden
layers
learn
edges,
lines
shapes
specific
patterns

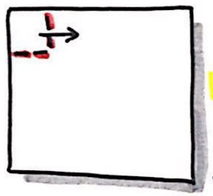
ConvNets



sliding window

↳ window **size** (e.g. 3x3 pixels)

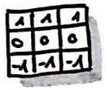
image



apply **filters** to every sliding window

e.g.

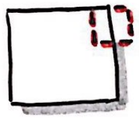
vertical lines



horizontal lines

- **Filters**
  detect shapes & patterns in window chunks
- multiple filters are combined
- filters are learned

padding



- fake pixels are created at the edge of images to incorporate border pixels

stride

- how much overlap to have in sliding window

Convolutional Layers

apply filters

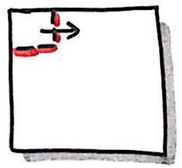
output

=

feature maps

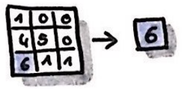
stacks of

feature maps



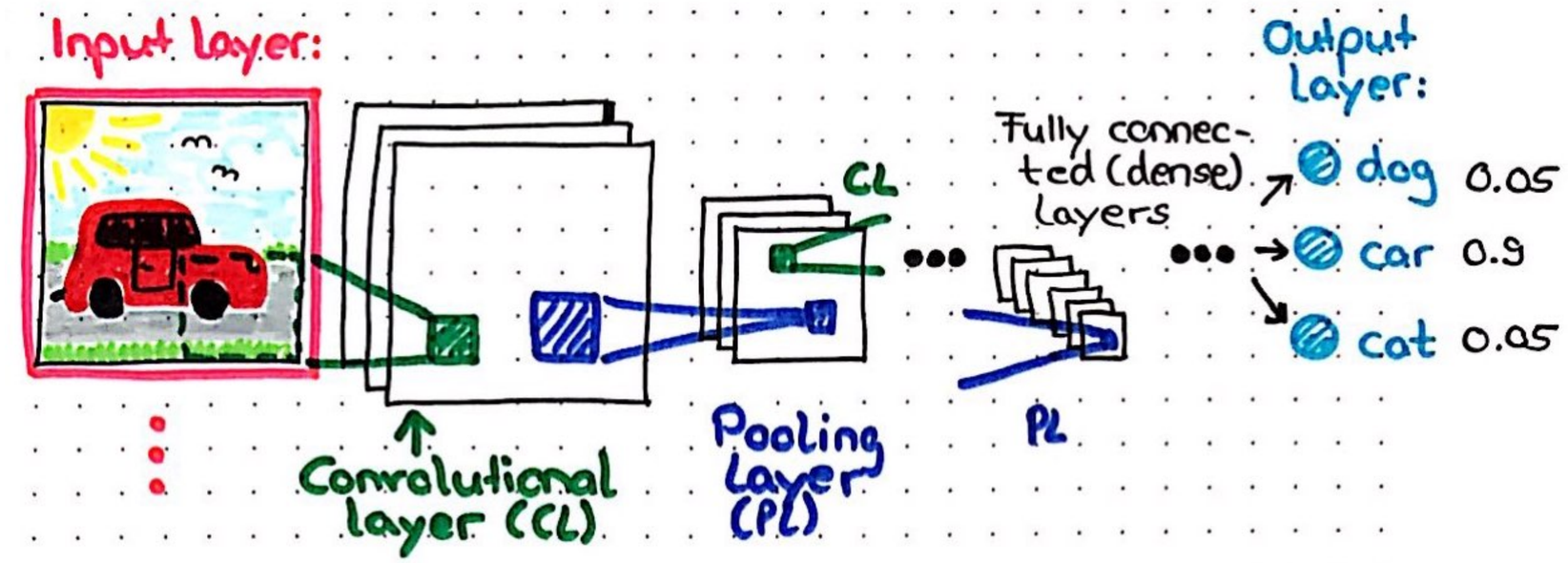
Pooling Layers

e.g. max pooling

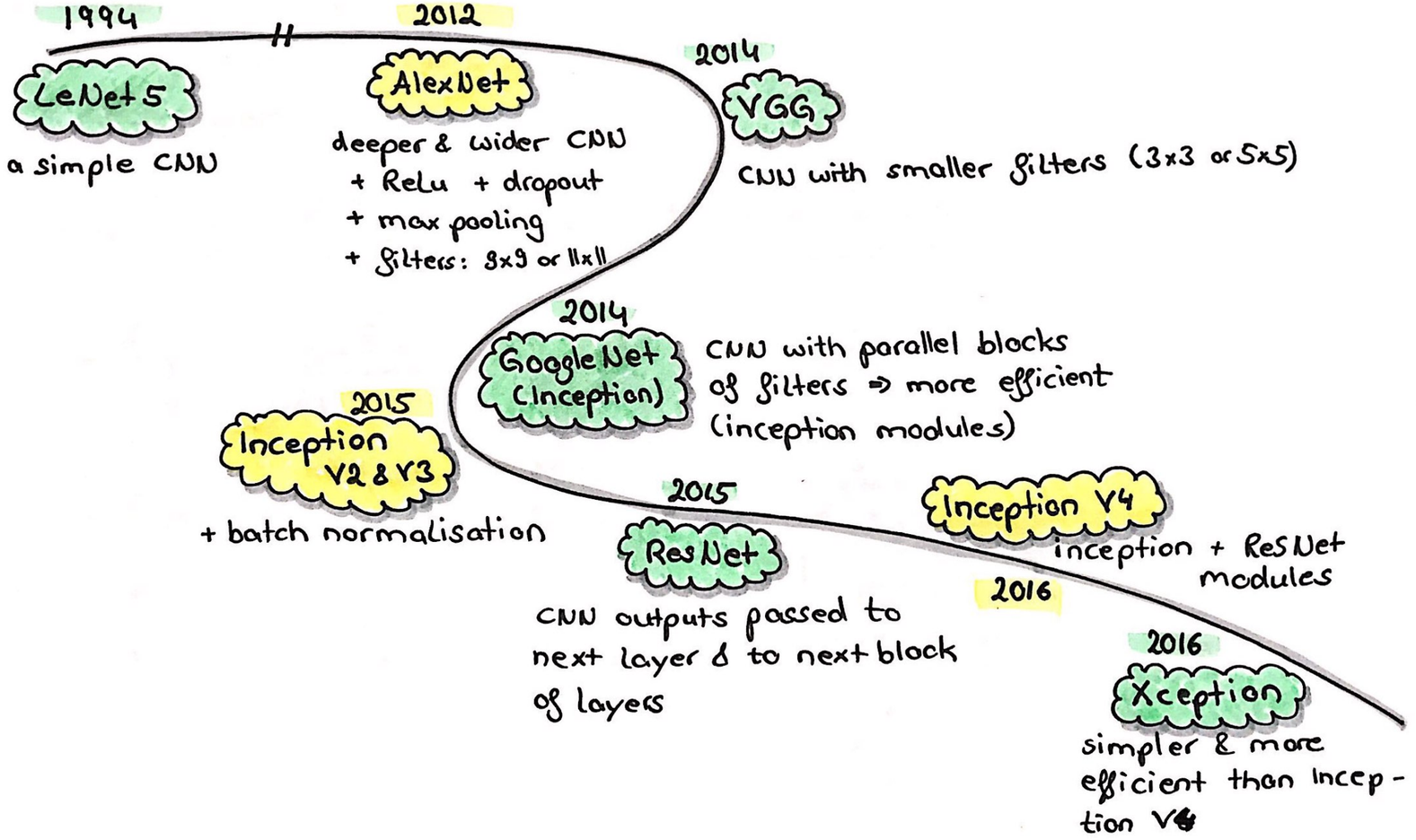


- reduces compute time
- boils information down

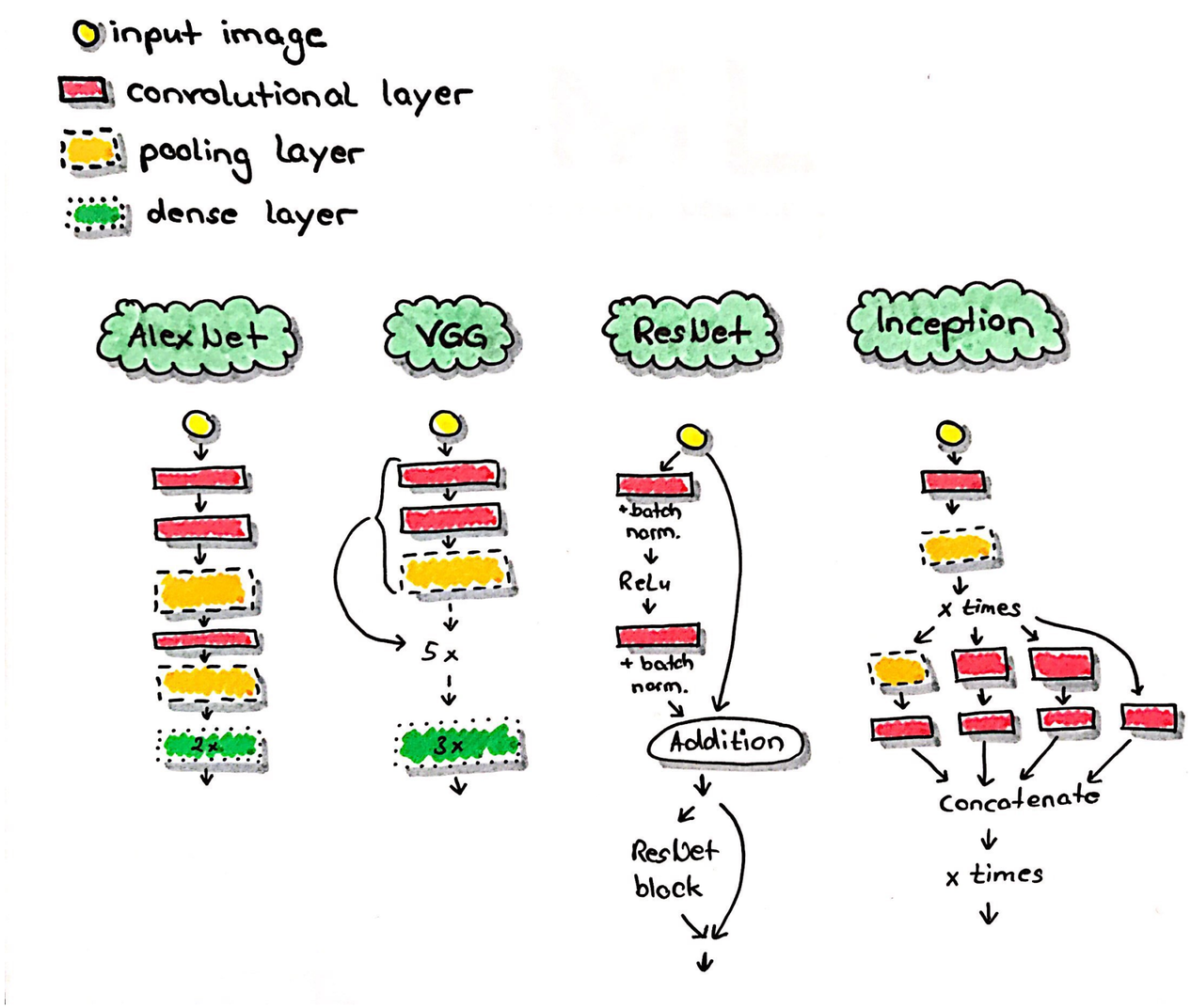
ConvNets


Input layer:
Convolutional layer (CL)
CL
Pooling layer (PL)
PL
Fully connected (dense) layers
Output layer:
dog 0.05
car 0.9
cat 0.05

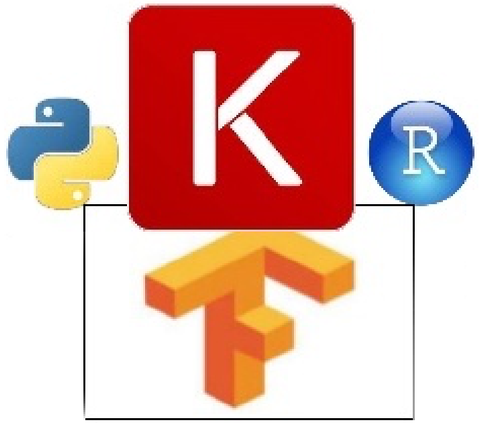
Evolution of neural nets for image recognition



CNN architectures

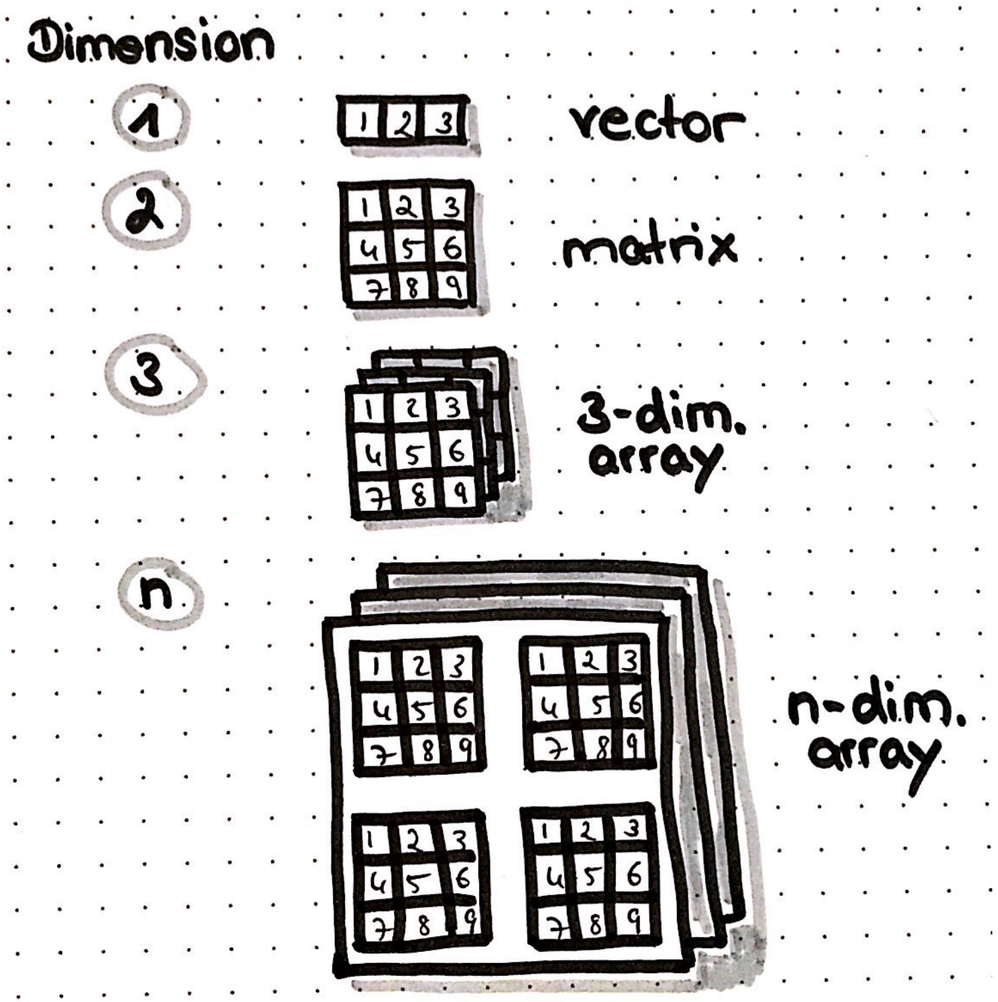


Introduction to TensorFlow



What are tensors?

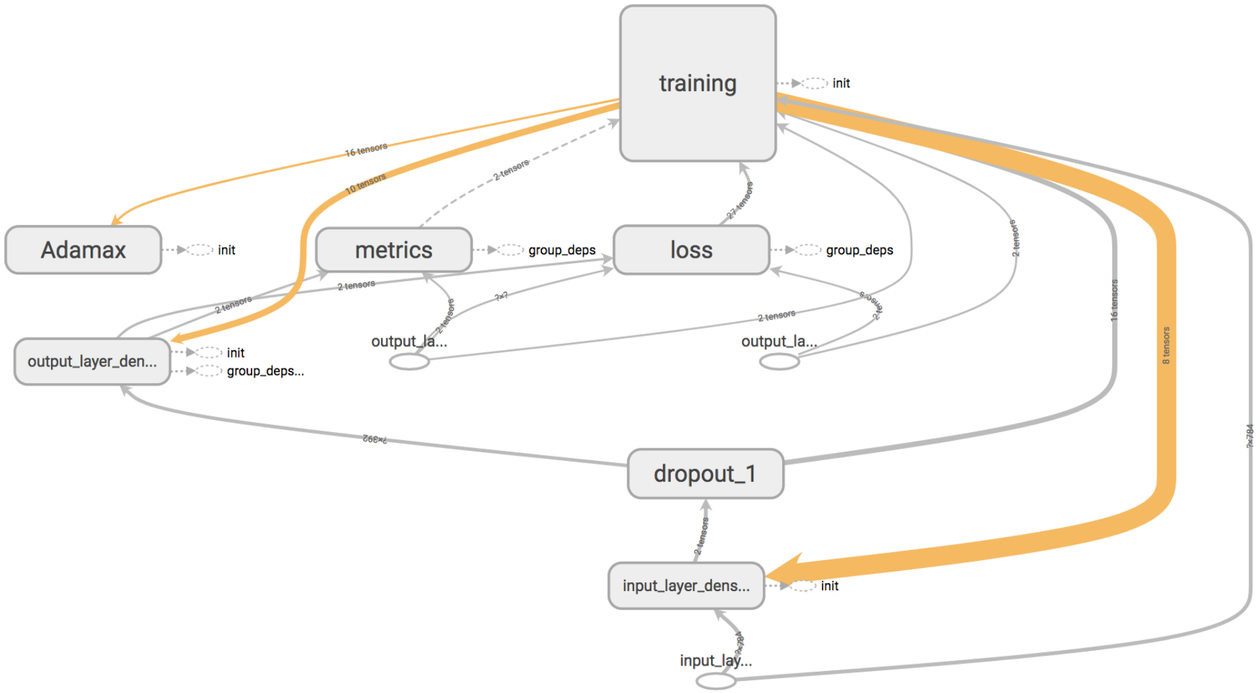
Tensors = multidimensional arrays



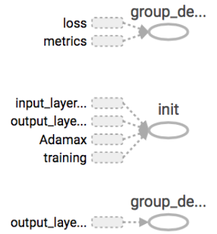
Tensor "Flow"

Graphs in TensorBoard

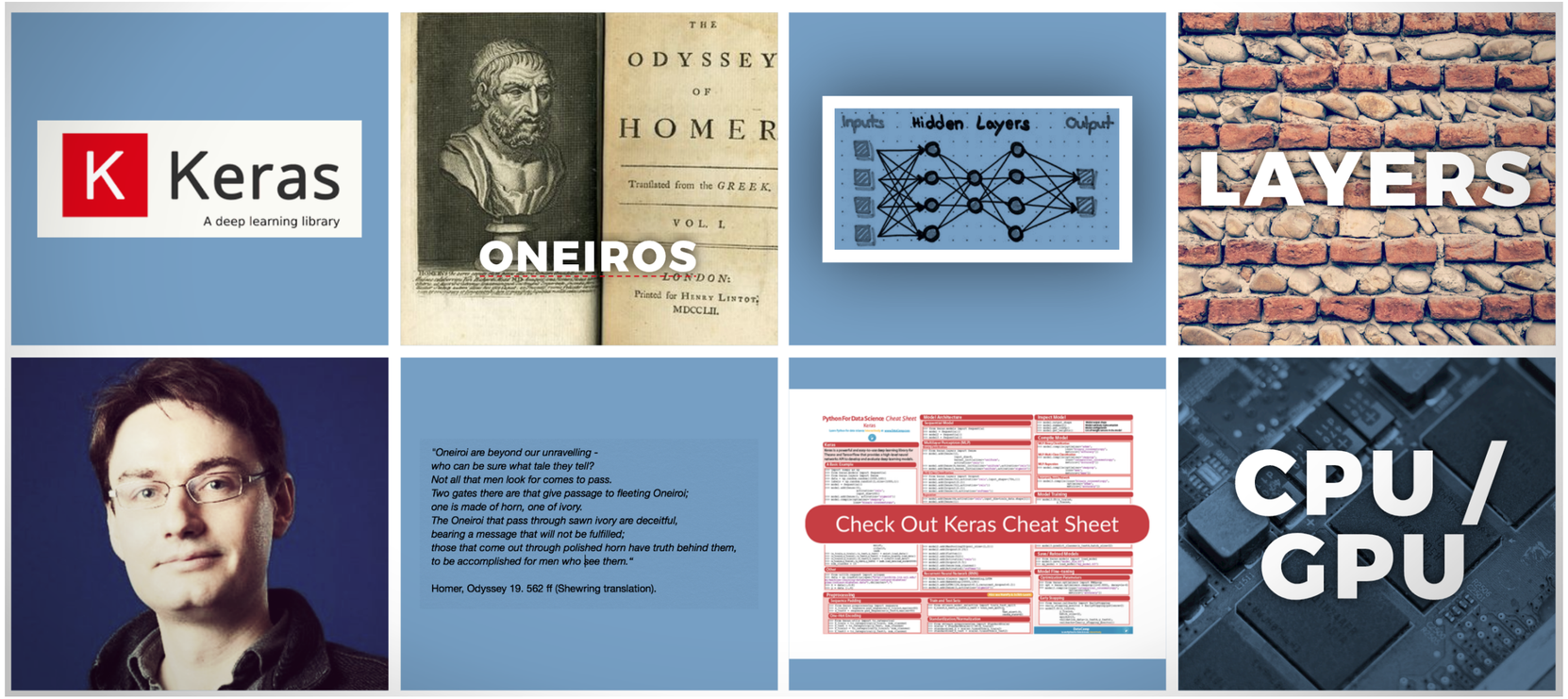
Main Graph



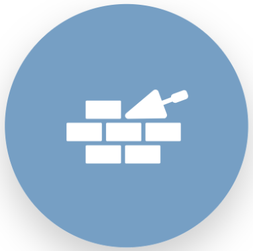
Auxiliary Nodes



Keras High-Level API for TensorFlow



Keras APIs



SEQUENTIAL MODELS

simple

suitable for most cases

linear order of layers

only one direction from input to output

FUNCTIONAL API

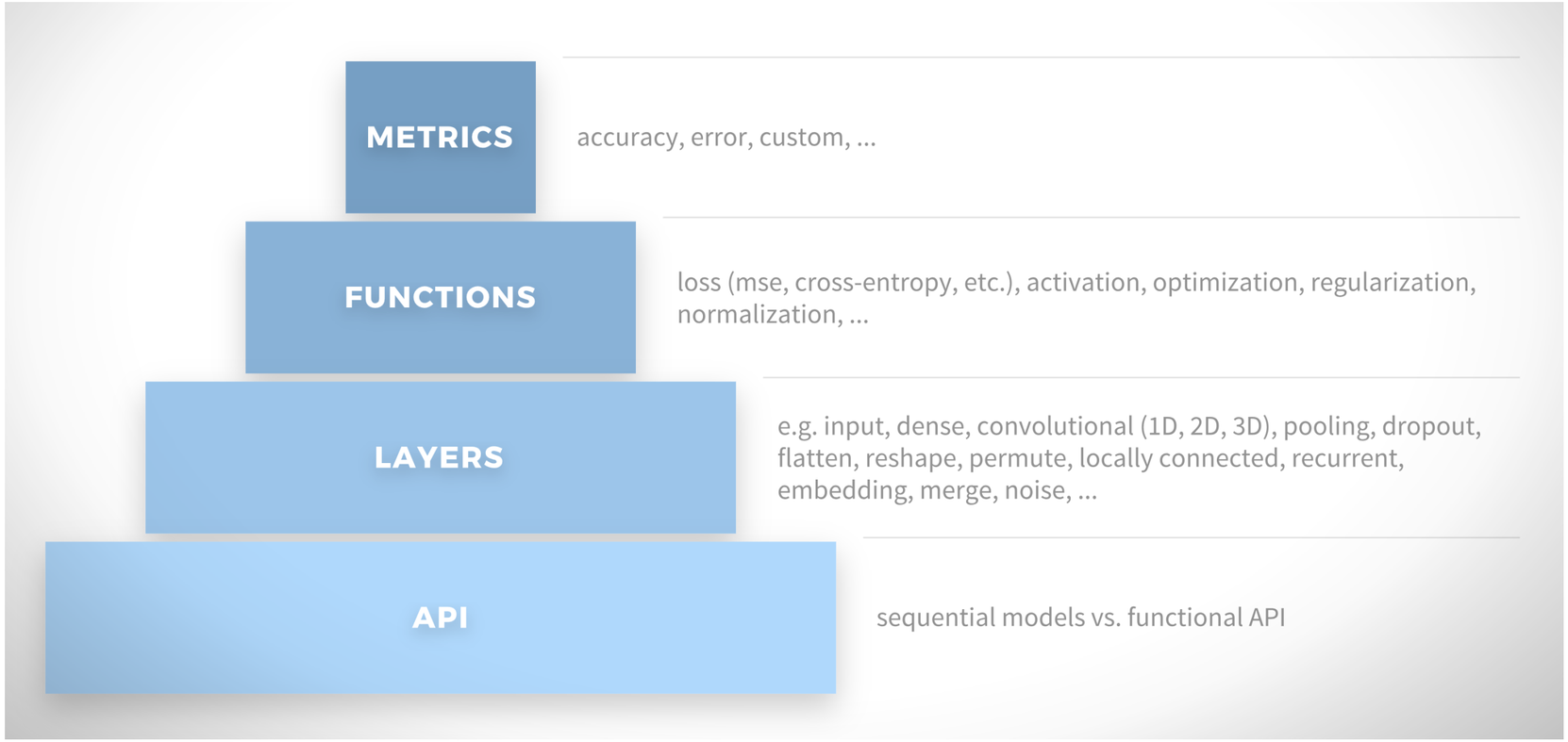
more complex

suitable for complex models

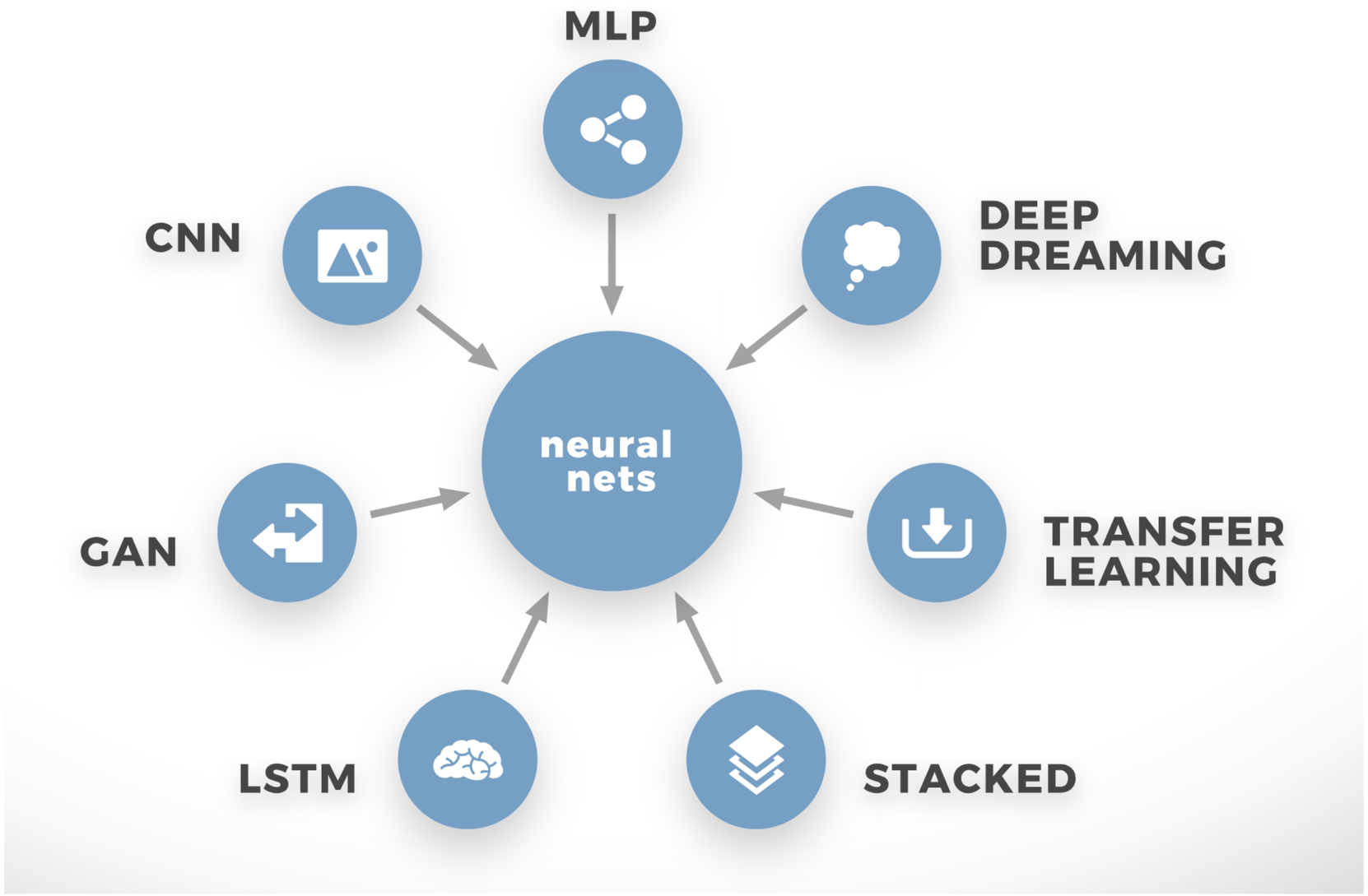
can have multiple in- or outputs

layers can be non-sequential, e.g. LSTM

Keras layers

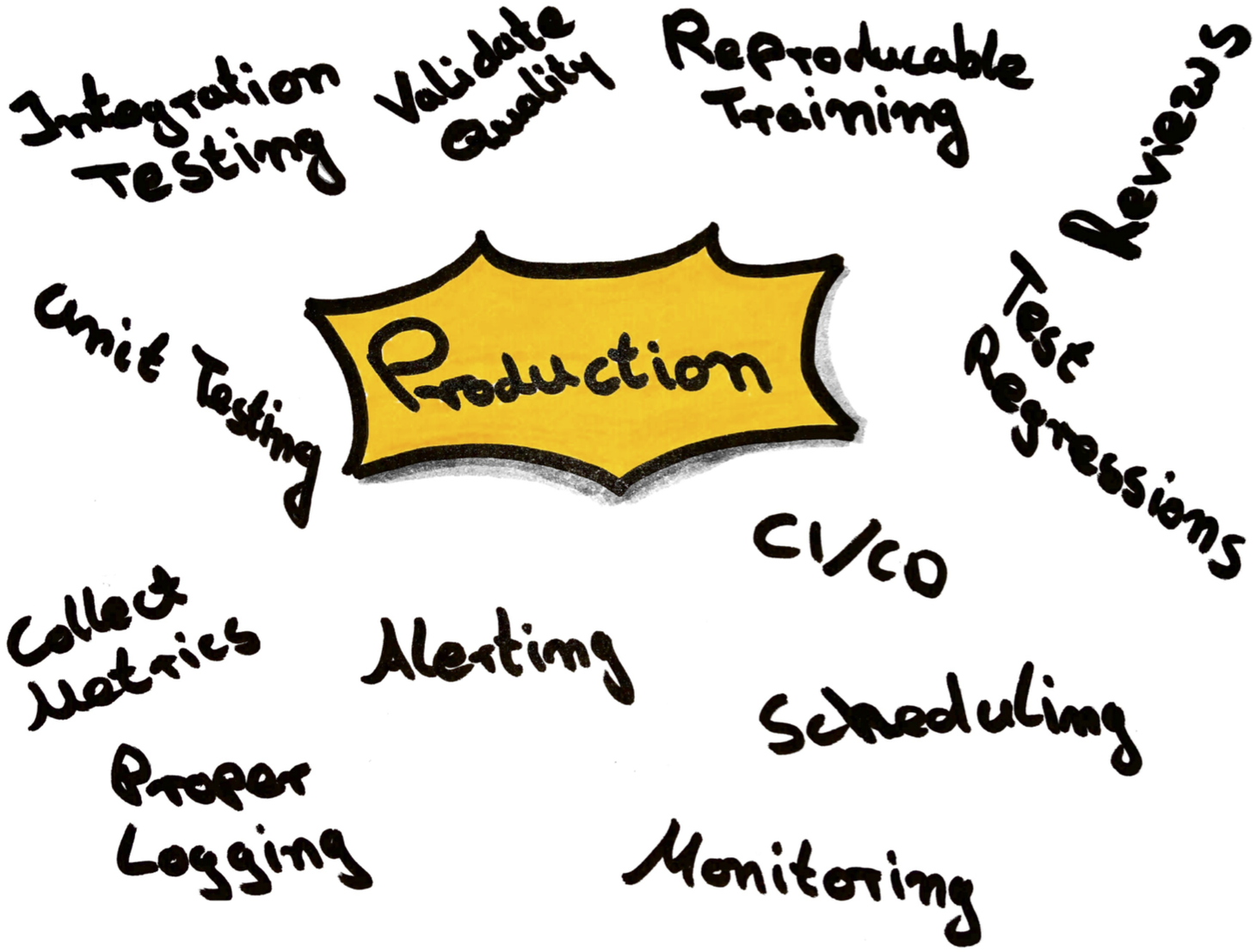


Endless possibilities



It works, now DEPLOY it!

Production ready



A bit about Luigi

Luigi helps to stitch long running tasks together into pipelines

It contains a wide toolbox of task templates (e.g. Hive, Pig, Spark, Python)

How to compose workflows?

A workflow consists of Targets, Tasks and Parameters

Targets correspond to a file or a database entry or some other kind of checkpoint

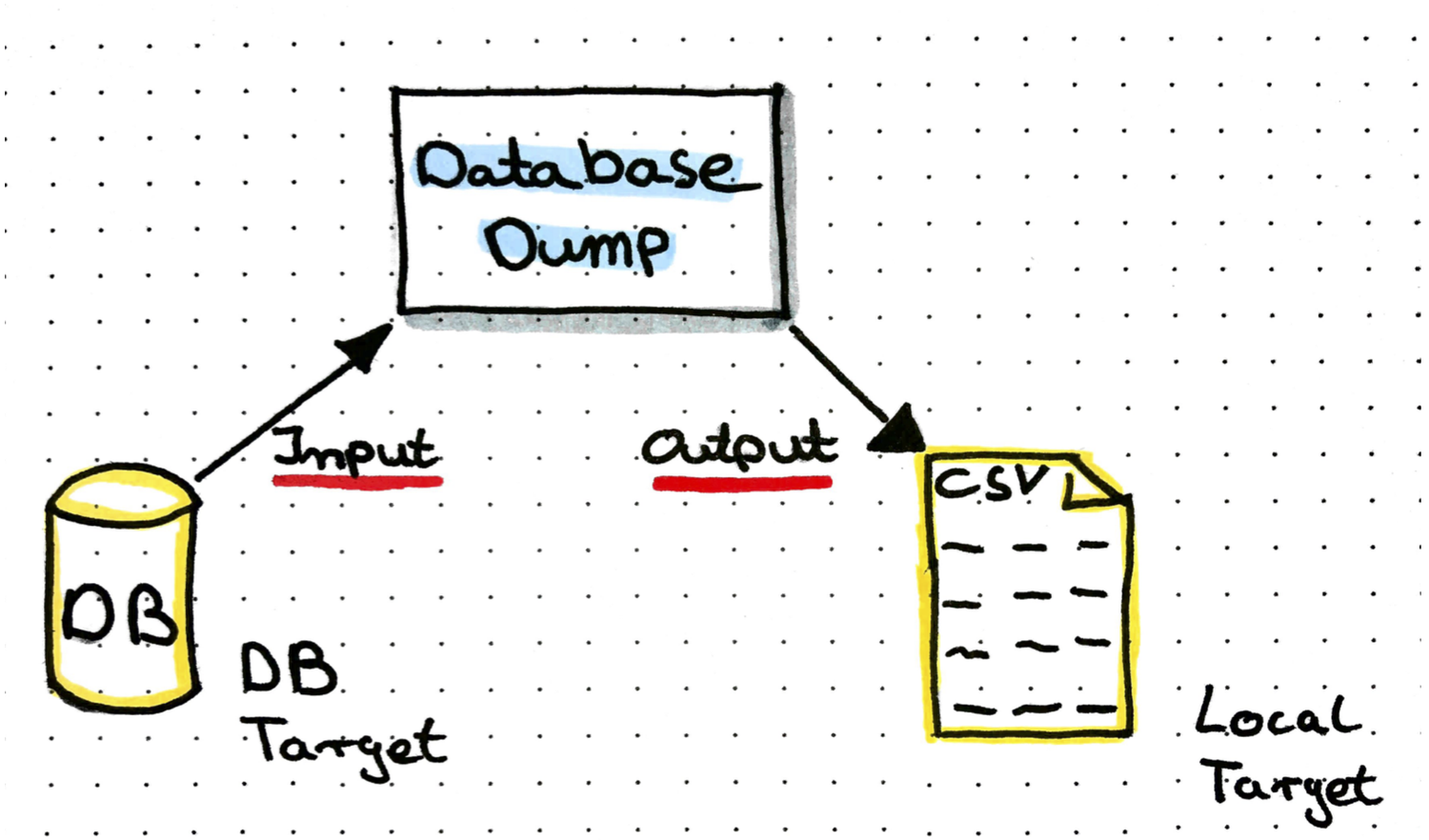
Tasks consume Targets of other tasks, run a computation, then output a target

Parameters take care of task parameterization

Targets

- Files on disk or database entries
- Checkpoints that prevent tasks from multiple executions
- A lot of implementations already exist in the Luigi framework
- LocalTarget (File), RemoteTarget (SSH), HDFSTarget, MySqlTarget, ...

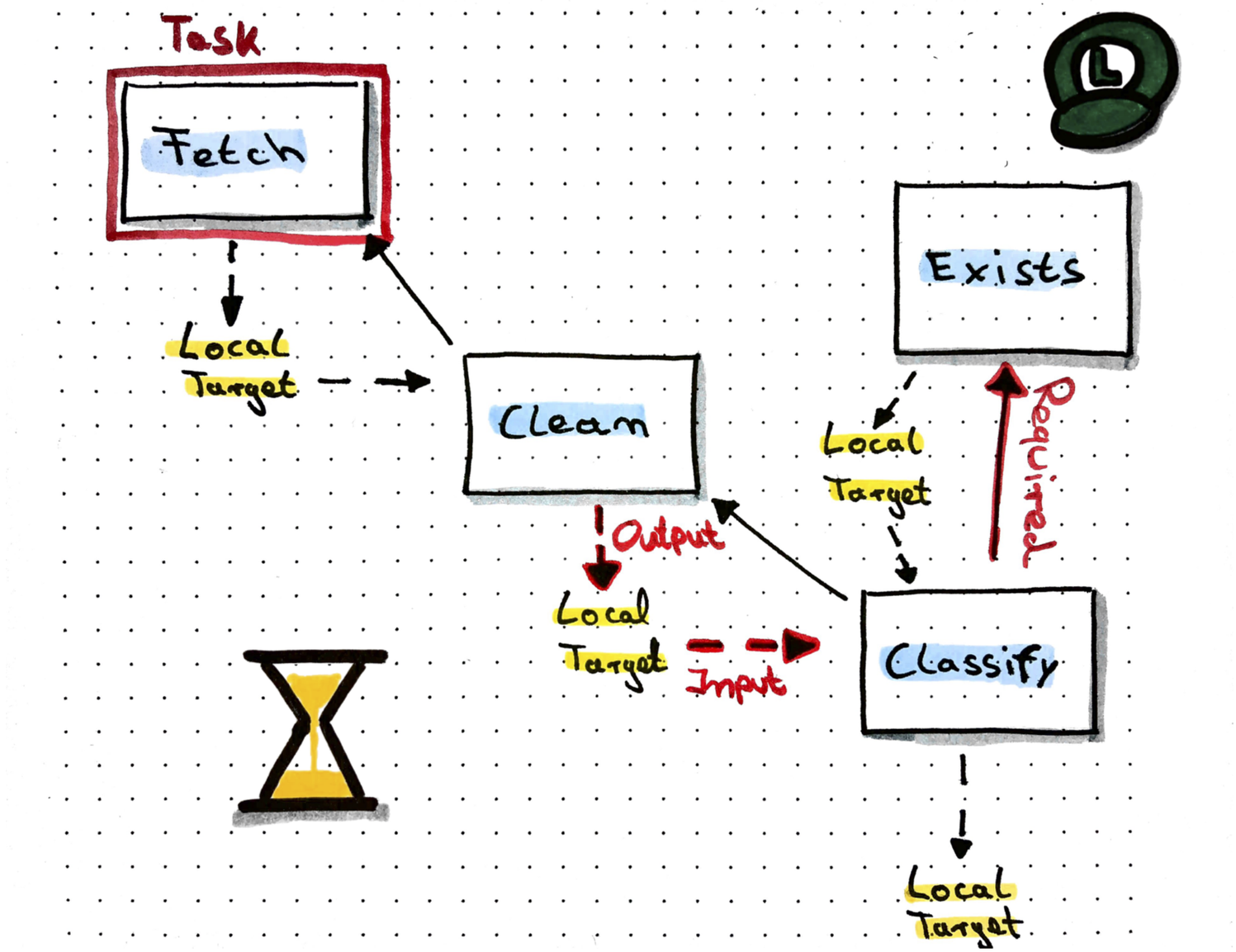
Targets



Tasks

- Implement the actual processing
- Consume targets, process data and save results in new target
- Respect dependencies to other tasks
- Implemented via Python-Classes

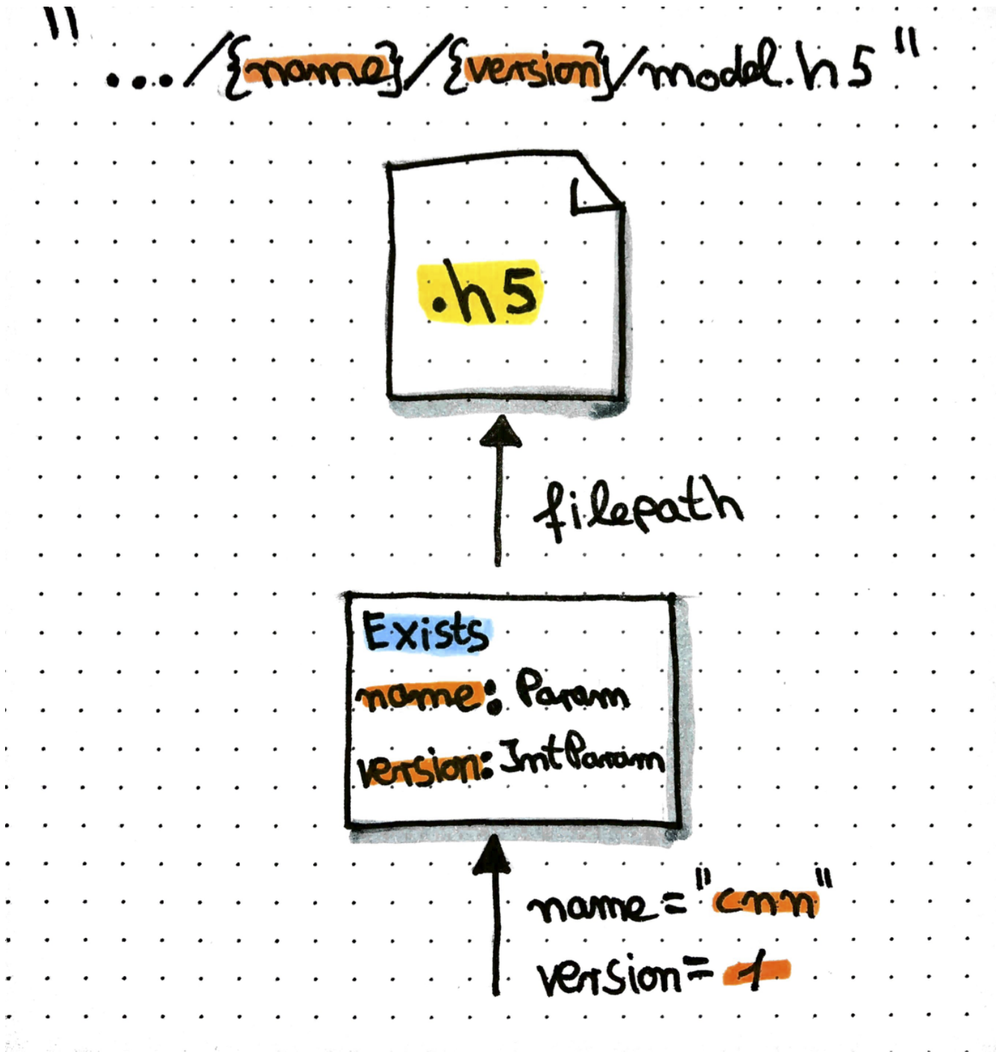
Tasks

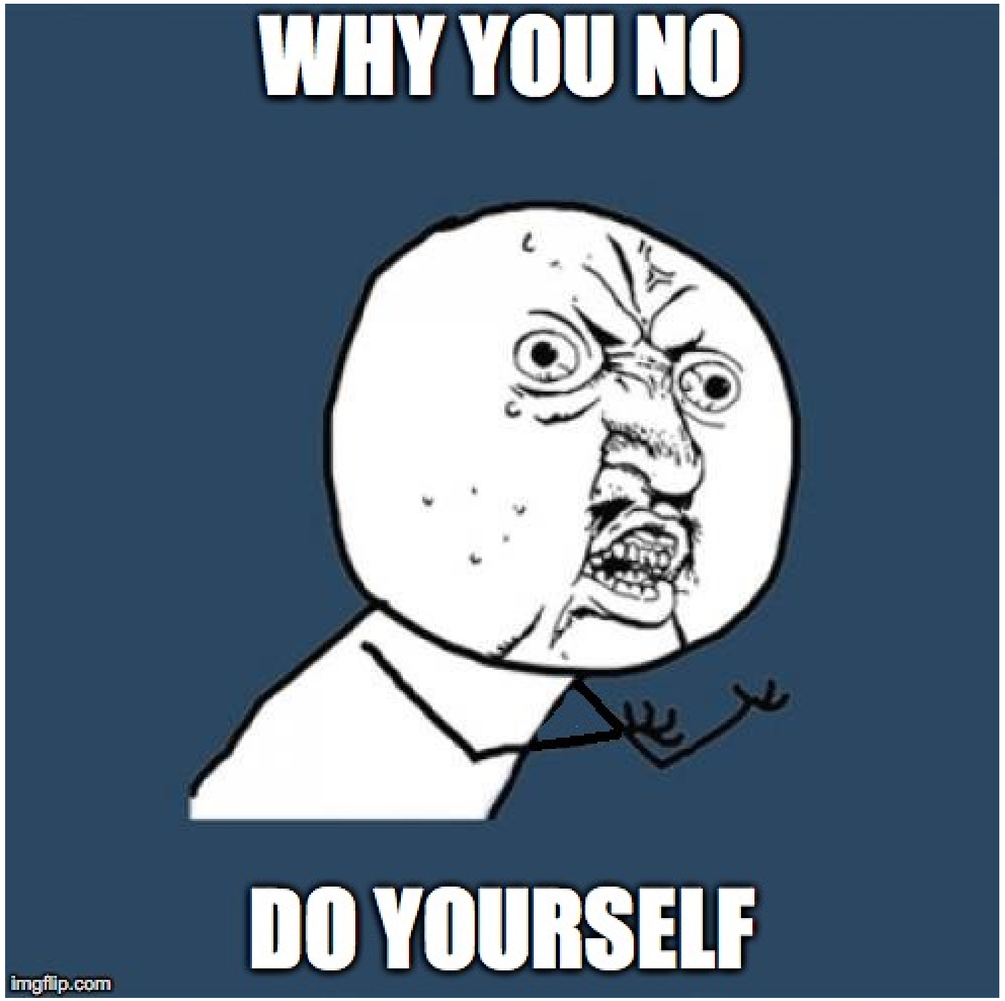

Task
Fetch
Local Target
Clean
Output
Local Target
Input
Exists
Local Target
Required
Classify
Local Target

Parameters

- like constructur parameters
- Luigi takes care of parameter validation
- Again, a lot of implementations already exist in the Luigi framework
- IntParameter, BoolParameter, DateParameter, etc...

Parameters

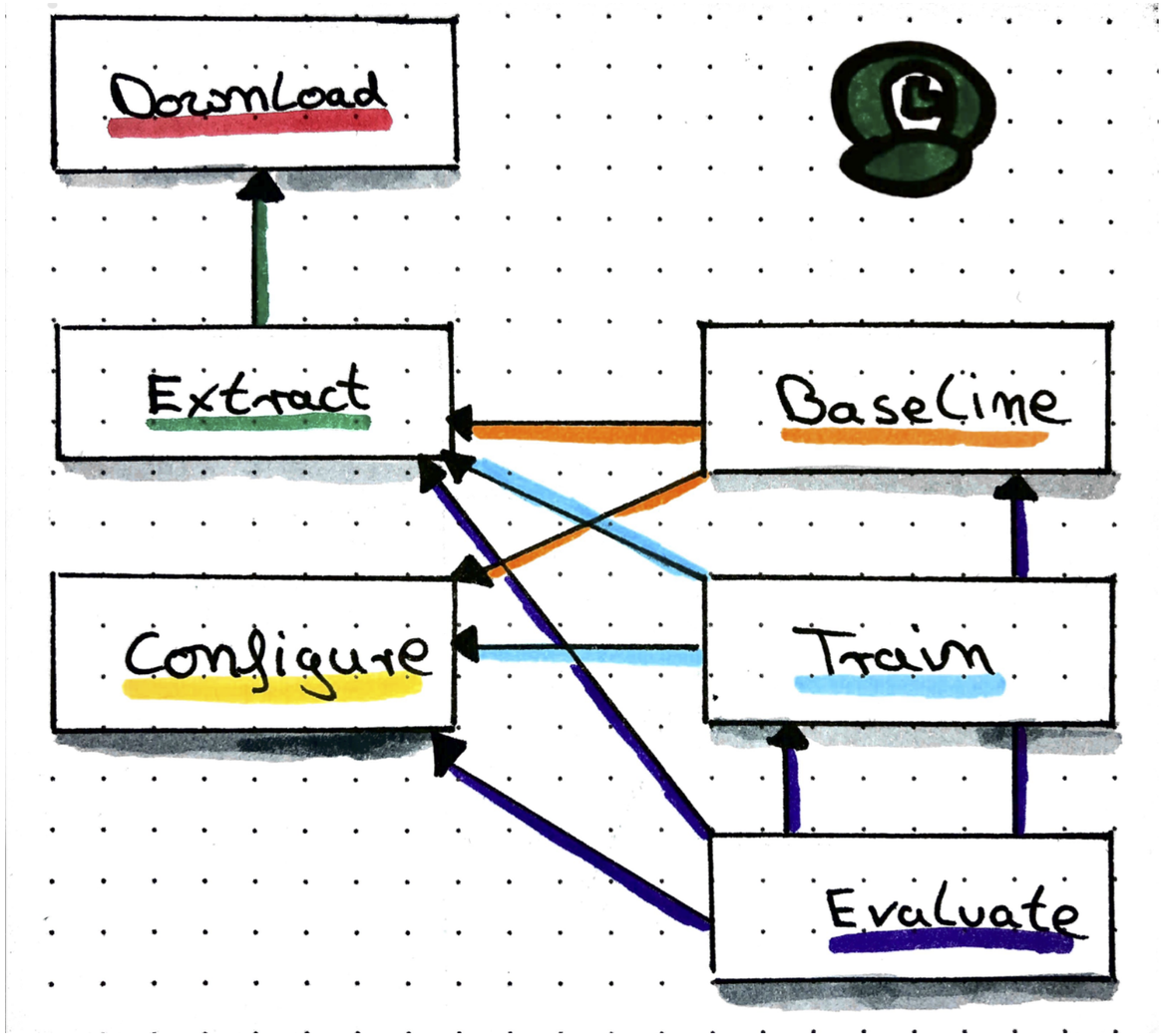

".../{name}/{version}/model.h5"
.h5
filepath
Exists
name: Param
version: IntParam
name="cnn"
version=1

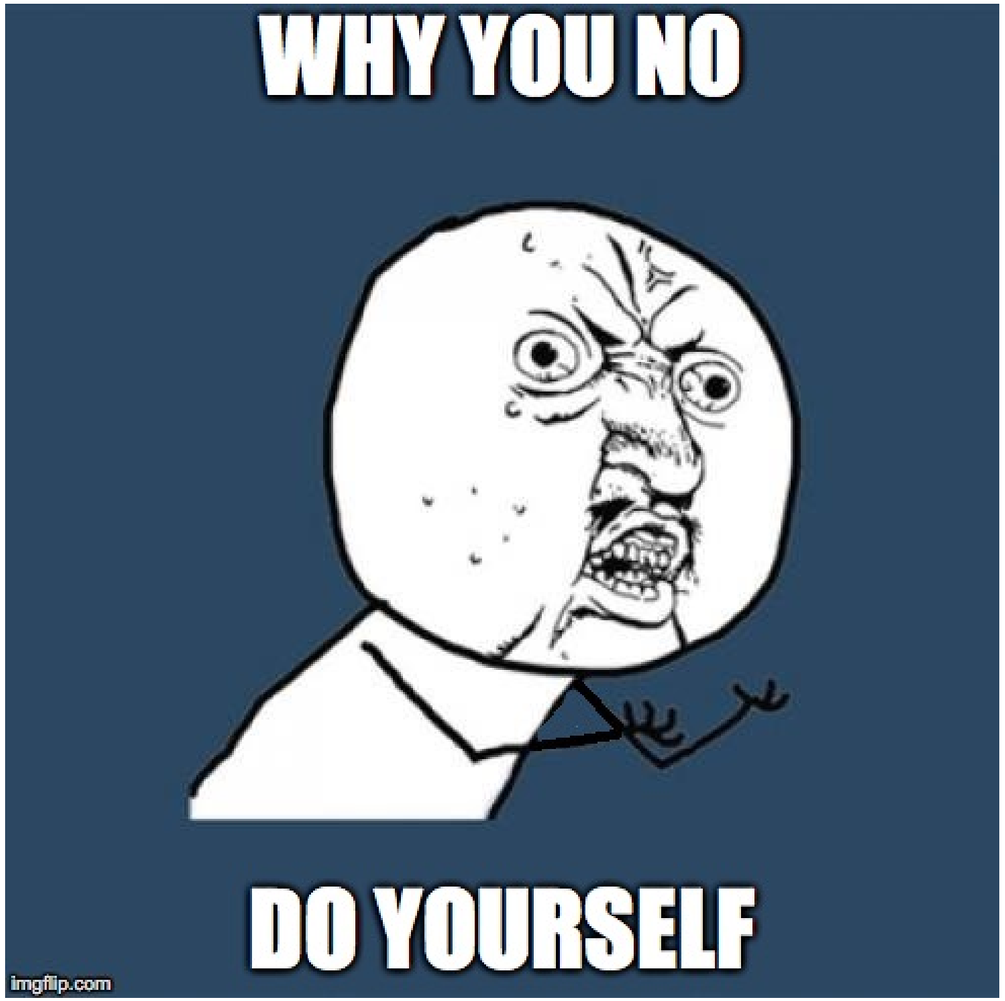

WHY YOU NO
DO YOURSELF
imgflip.com

How would a production-ready Workflow look like?

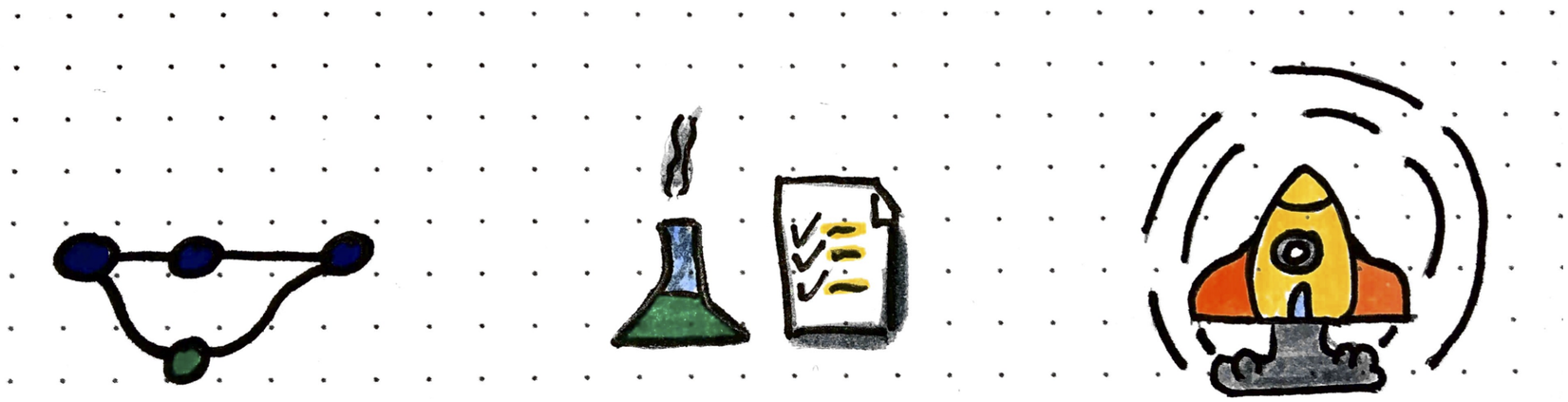
- Download the dataset
- Extract the data
- Create a preprocessing configuration
- Run the baseline validation
- Train the model
- Evaluate the model

Workflow

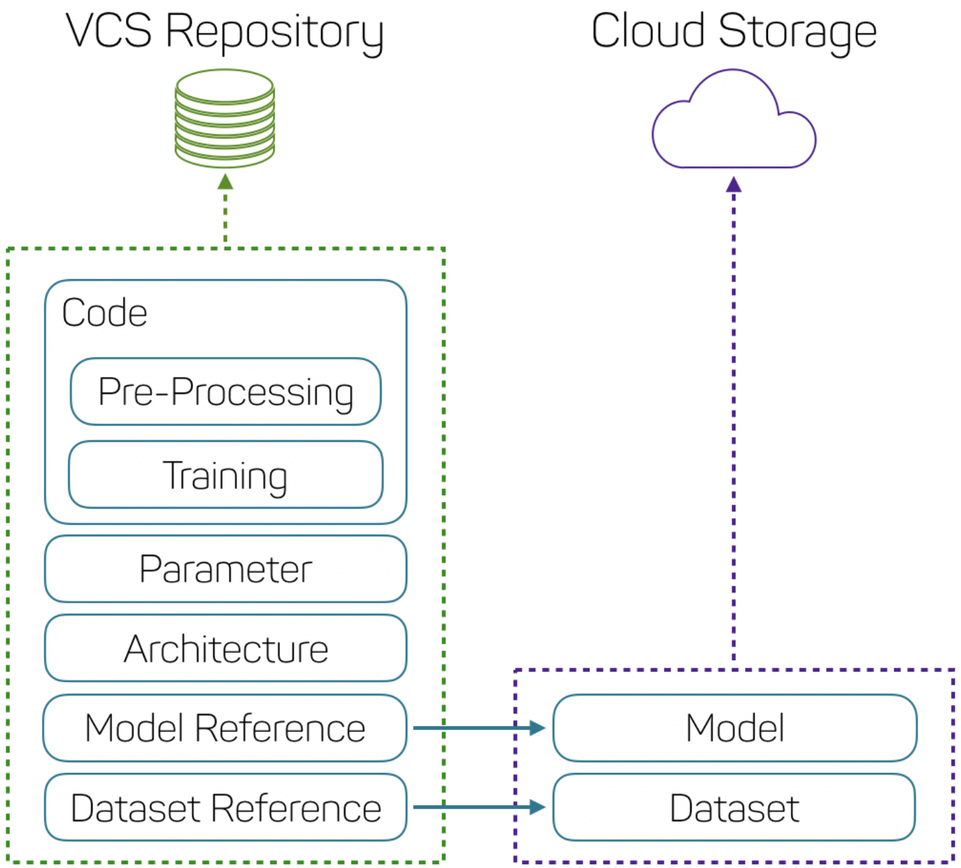



WHY YOU NO
DO YOURSELF
imgflip.com

DVC - Data Version Control

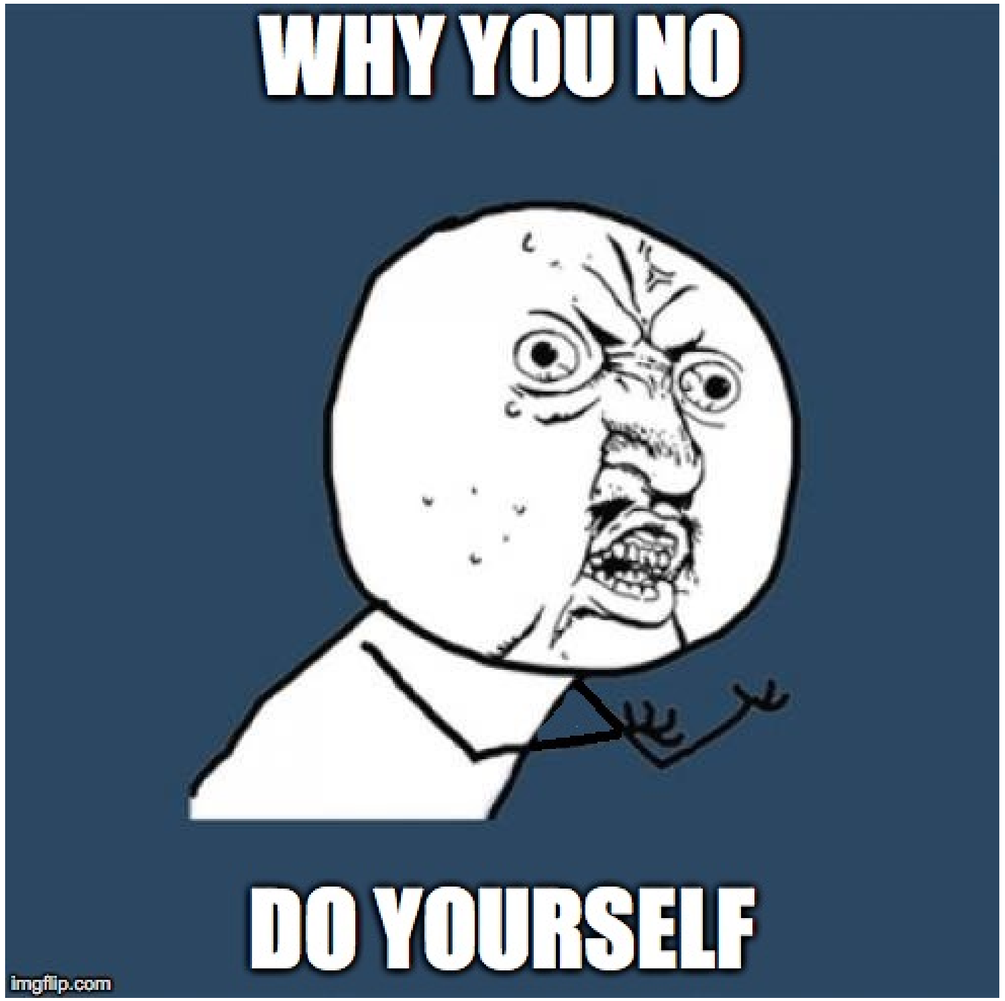


Artifacts and Storage

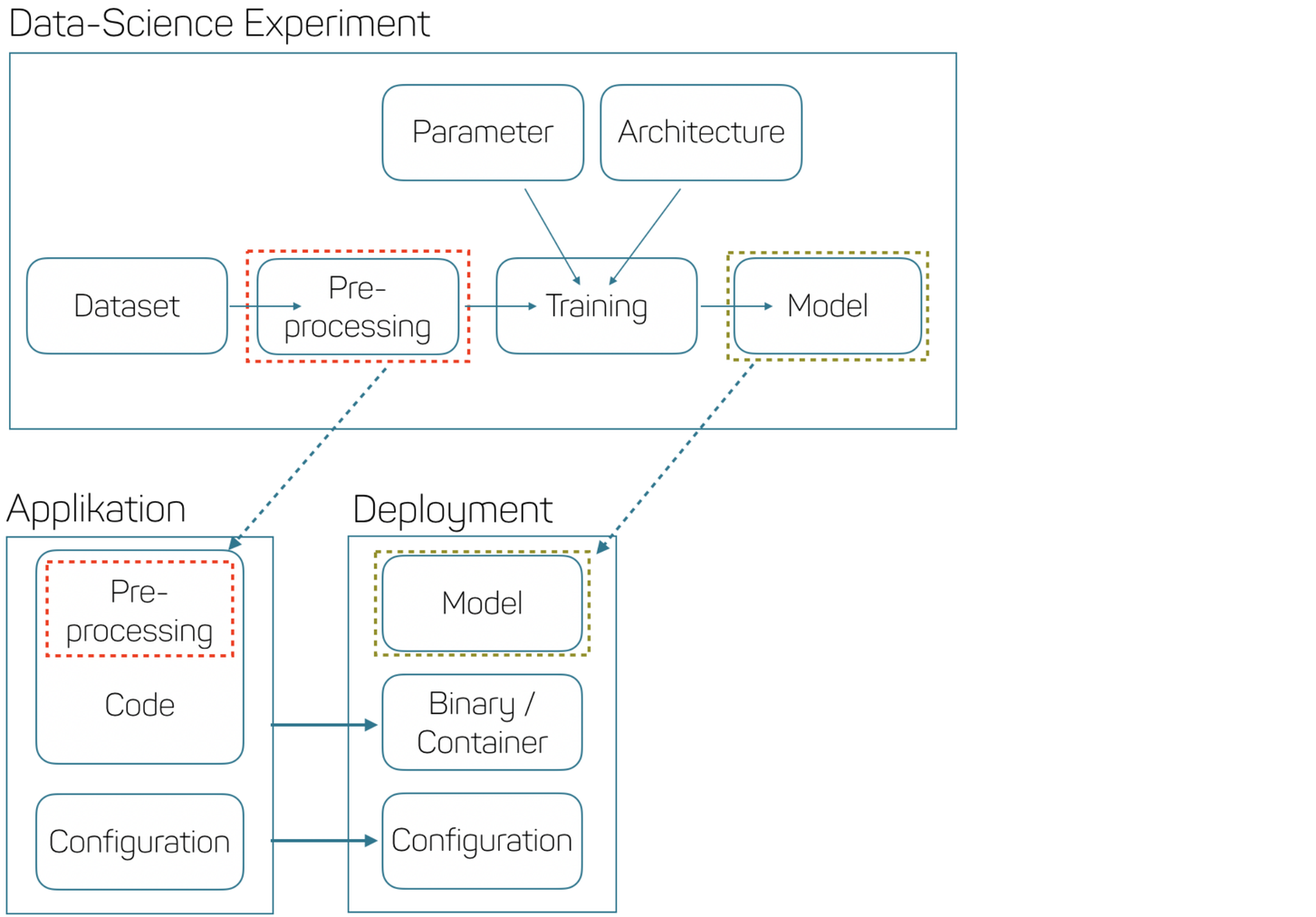


Using DVC

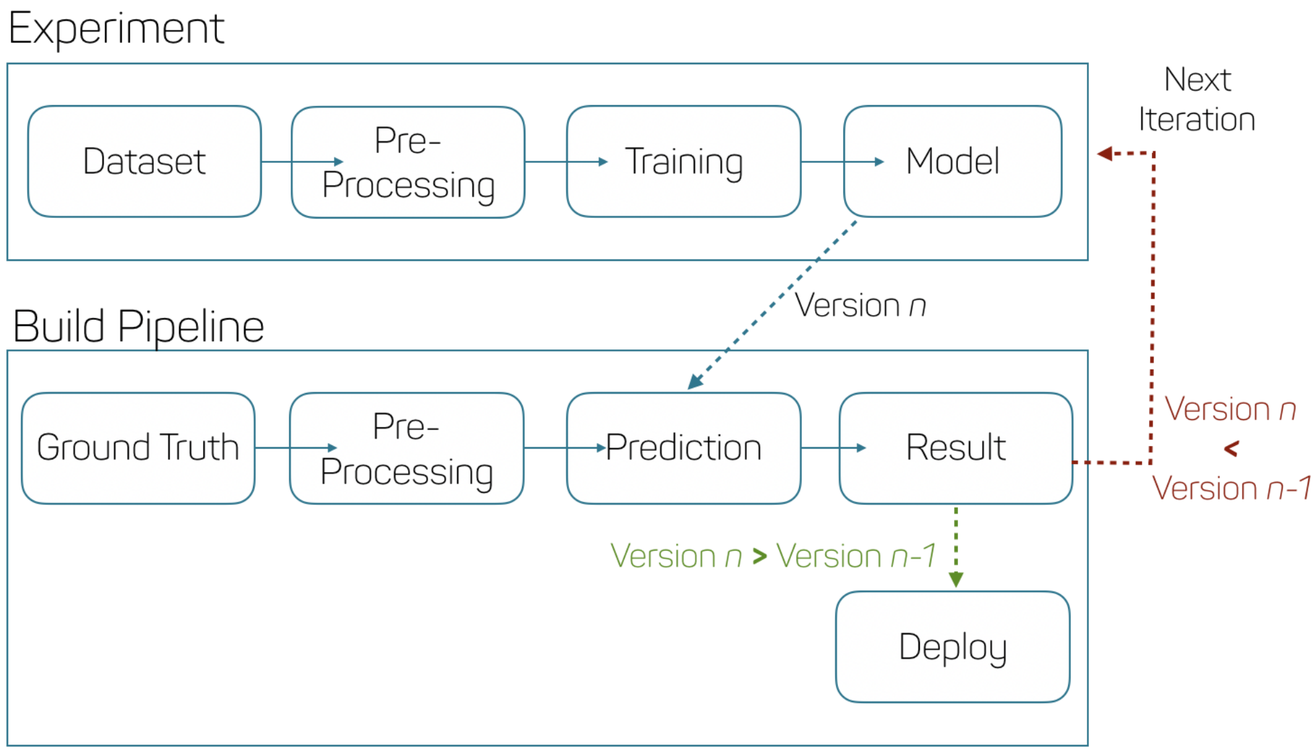
- DVC feels like Git
- Basic commands like: dvc init, dvc checkout, dvc add, dvc remove ...
- Lightweight pipelines: dvc run, dependencies wiht -d, outputs with -o, -O
- Management commands: dvc metrics, dvc pipeline


WHY YOU NO
DO YOURSELF
imgflip.com

Data Engineering - Development vs Experiments

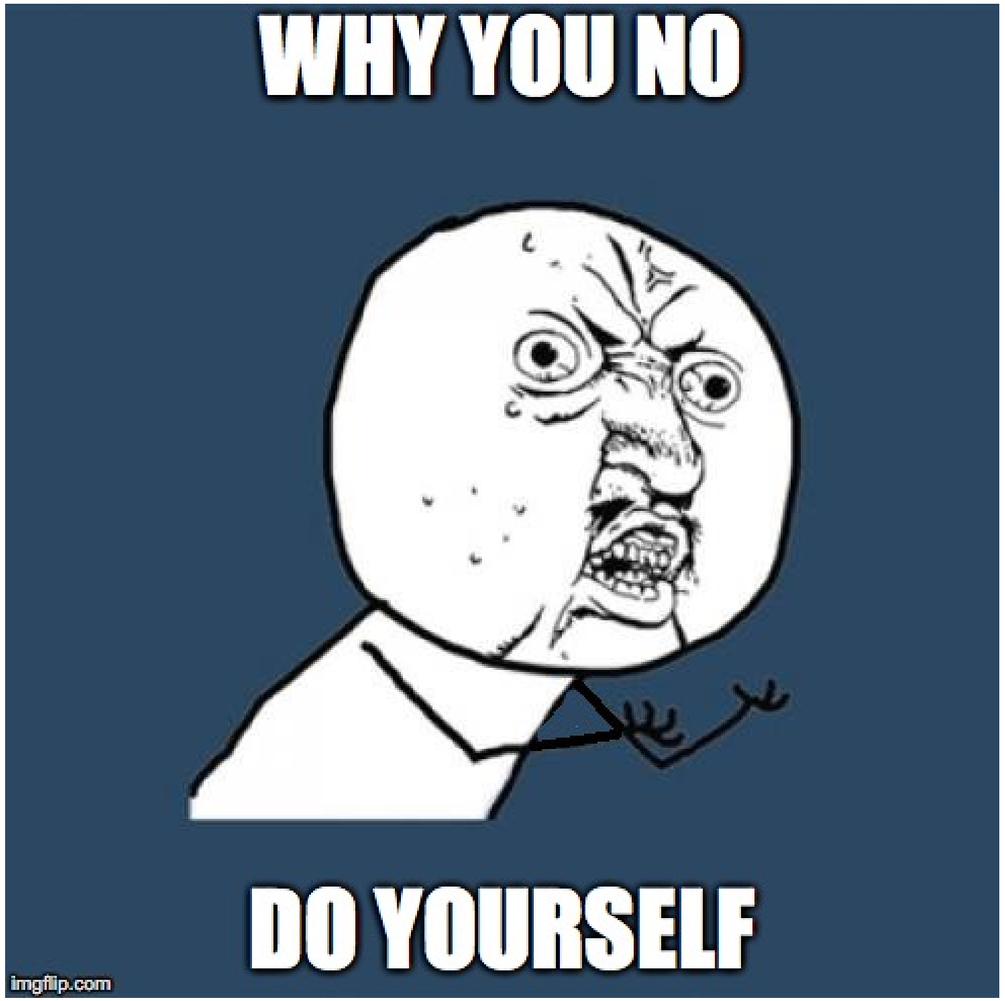


CI/CD for Experiments

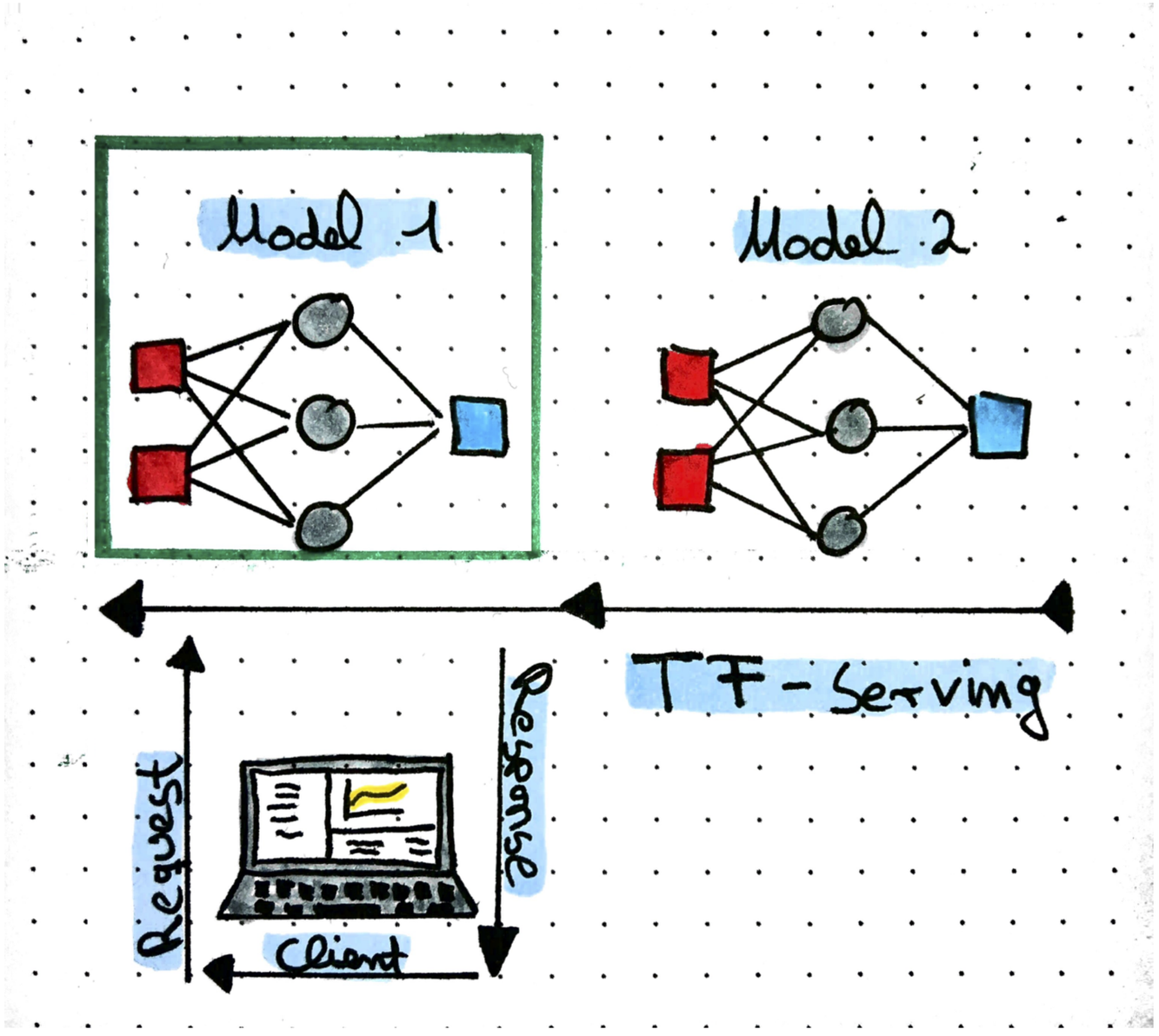


Jenkins

- Jenkins is an automation server
- Used for CI and CD
- Helps us to automate builds and deployments


WHY YOU NO
DO YOURSELF
imgflip.com

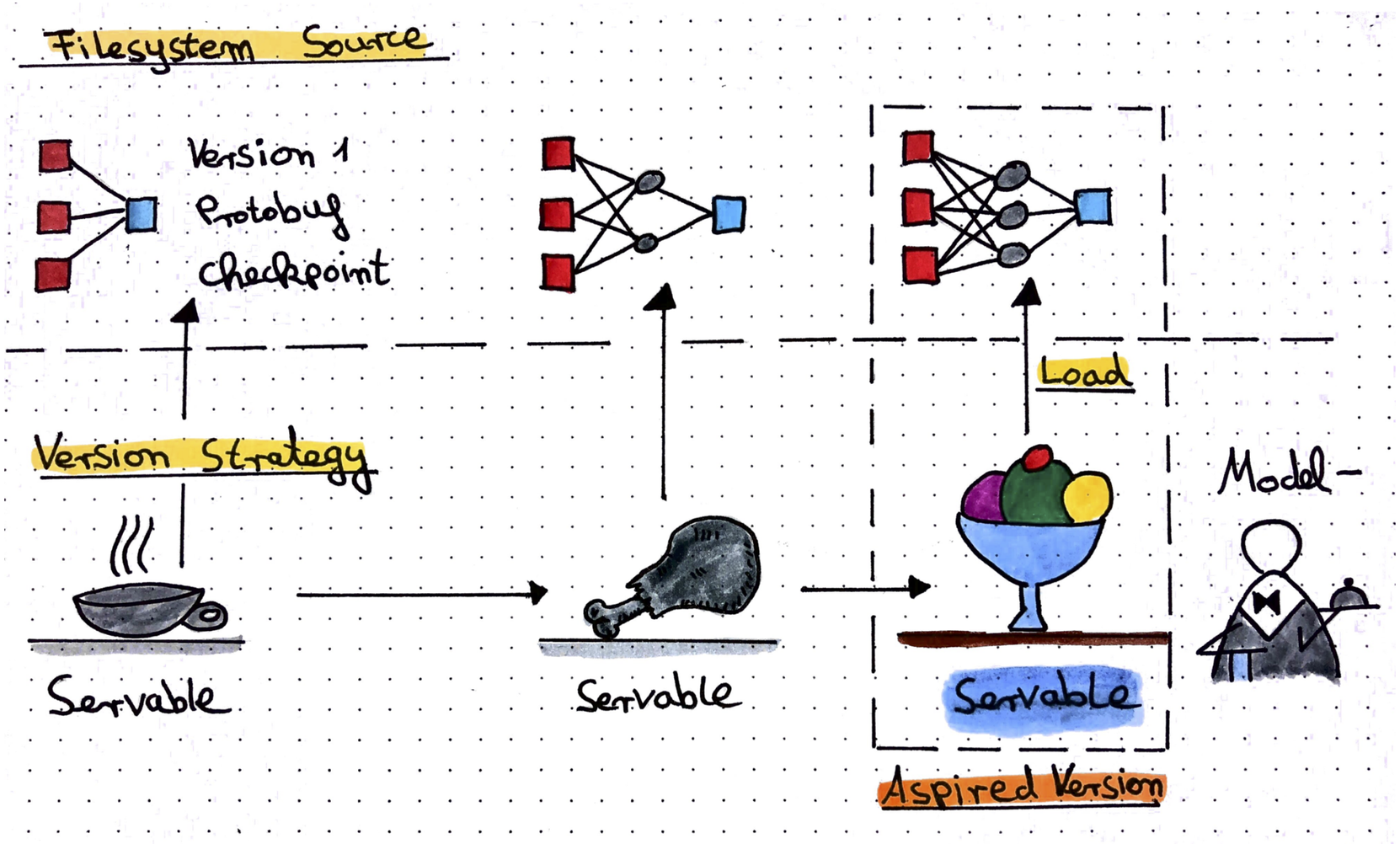
TensorFlow Serving



TensorFlow Serving

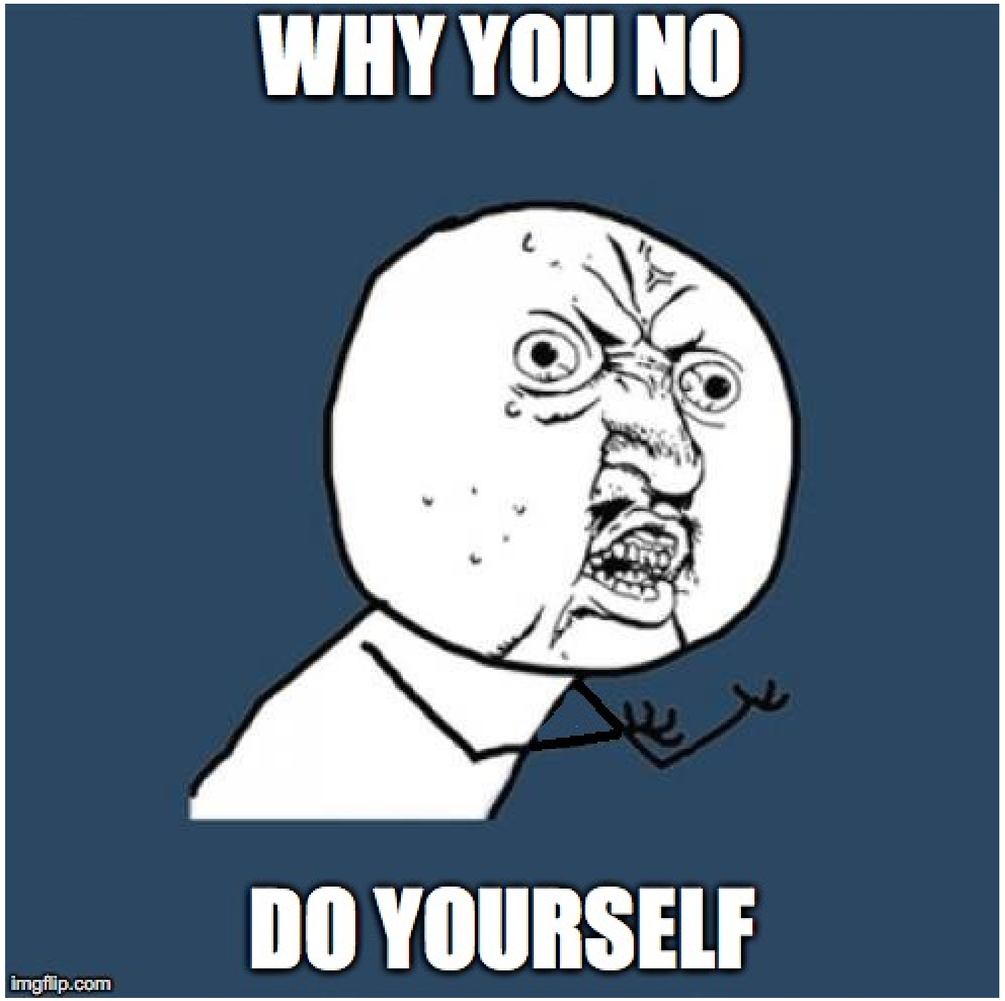
- ModelServer as Server-runtime for ML models
- Can natively handle TensorFlow graphs
- Highly flexible
- Designed for production use

TensorFlow Serving in Detail



TensorFlow Serving Components

- Protobuf file for graph, checkpoint files for weights (if not frozen)
- Version strategy defines how many models are loaded into memory
- Clients request inferences via gRPC or REST (yes REST!)
- TF-Serving consists of pluggable components, namely sources, loaders and version strategies


WHY YOU NO
DO YOURSELF
imgflip.com

Integrate pre-processing into model

- The pre-processing and the model are tightly coupled
- Datagenerator can be pickled to make training reproducable
- But how can the client be sure to do the "right" thing?

Possible ways:

- The client just "knows"
- Wrap service around model and expose an API
- Built the pre-processing right into the model

In [ ]:

```
def pre_process(x):
    resized = tf.image.resize_images(x, size=[100, 100])
    max_color = tf.constant(255, dtype=tf.float32)
    rescaled = tf.divide(resized, max_color)
    return rescaled

additional = Lambda(pre_process, output_shape=(100, 100, 3), name="preprocessing")
```

In [ ]:

```
prod_model = Sequential()
prod_model.add(InputLayer(input_shape=(None, None, 3)))
prod_model.add(additional)
for layer in model.layers:
    prod_model.add(layer)
```