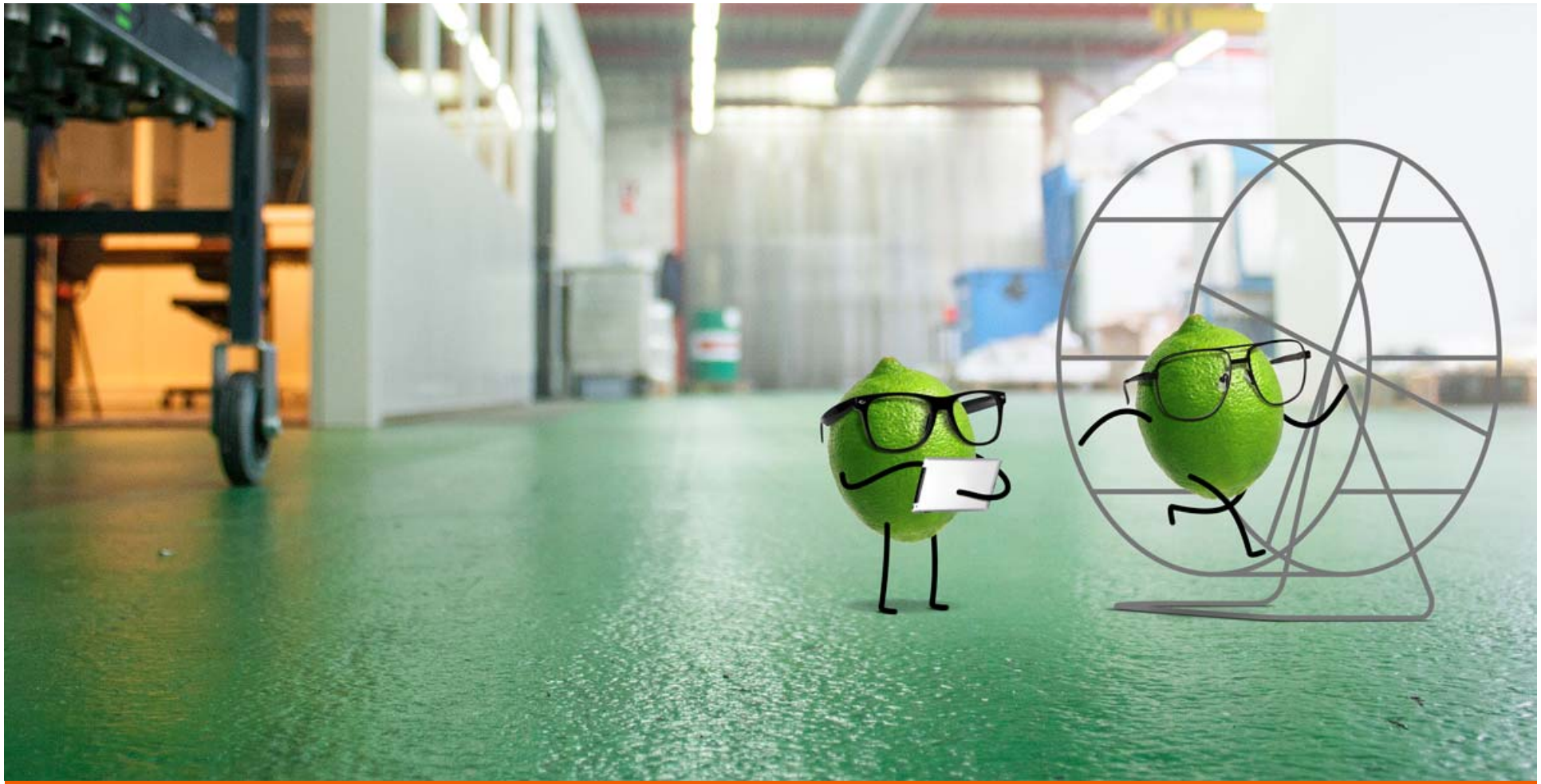
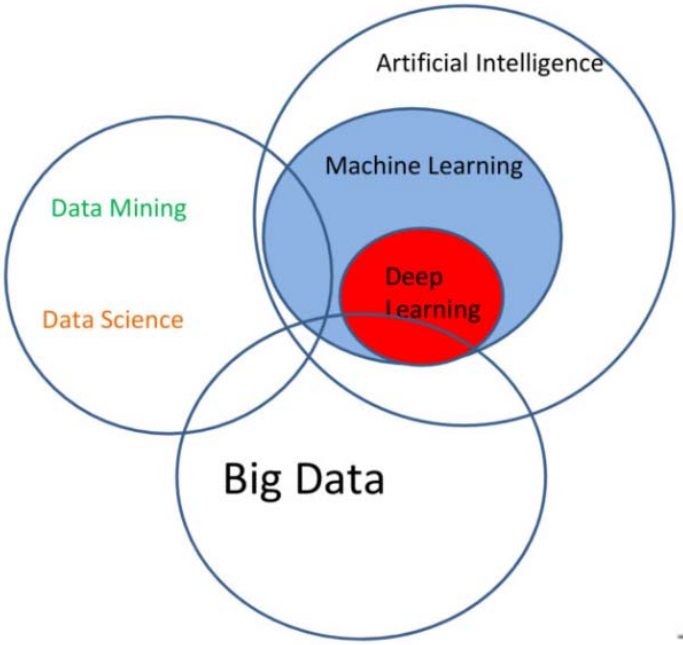
Data Science @ Sioux LIME

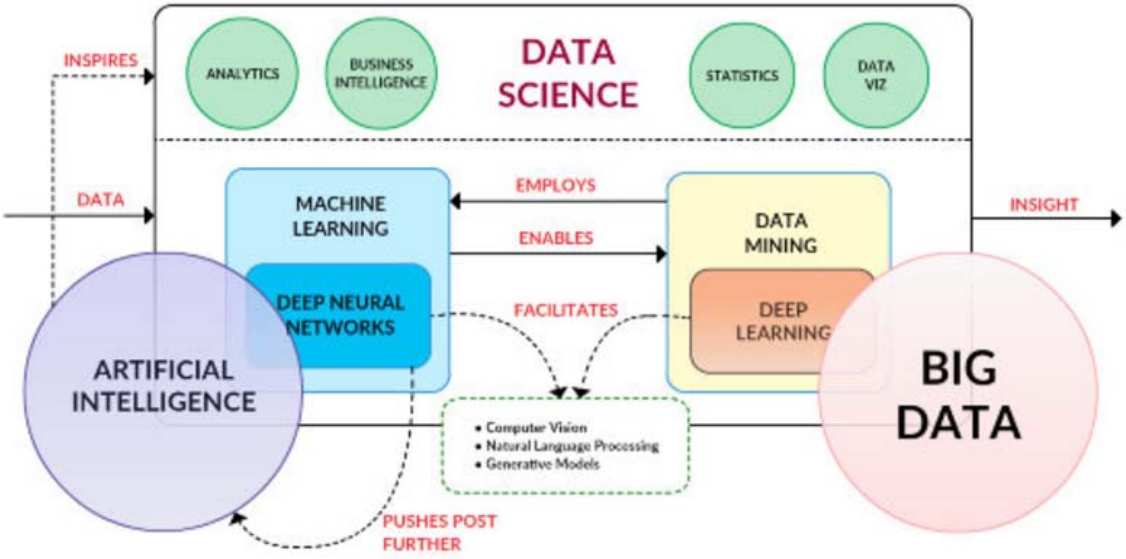
A peek at a recent project…

SIOUX LIME MATHWARE
SOURCE OF YOUR TECHNOLOGY



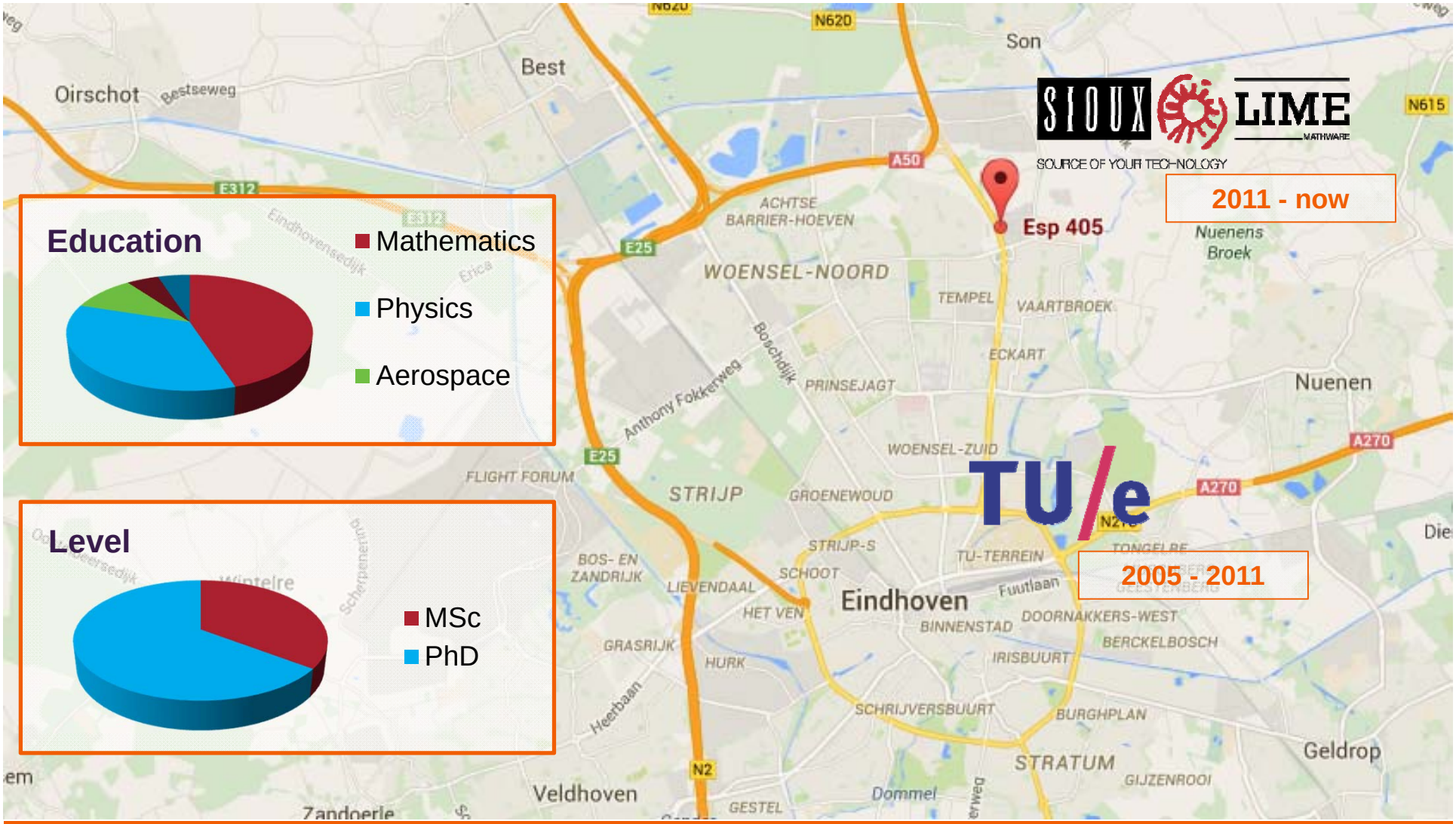
Big Data, Data Science, Machine Learning…





Source: http://www.kdnuggets.com/2016/03/data-science-puzzle-explained.html

Sioux LIME - Company profile

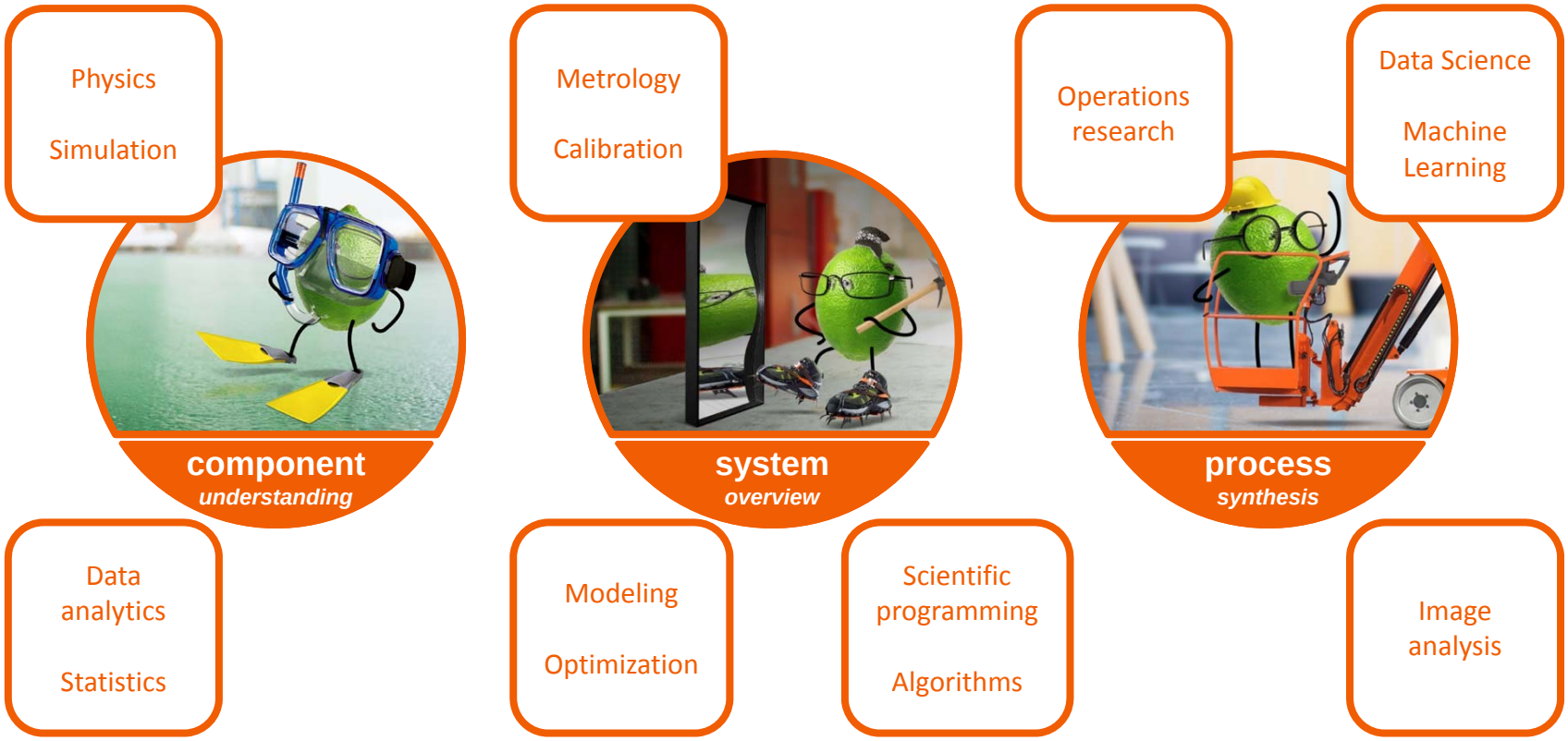


Mathware

Realize practical solutions for industry through scientific knowledge and expertise.

(Mathematical consulting)

Competences



Clients

component

system

process

TEIJIN

NXP

STIRLING CRYOGENICS

ASML

bpa blueprint automation flexolutions in packaging automation

VANDERLANDE

ACEA

Lloyd's Register

PHENOMWORLD

FEI

DYNOBEND

cyclomedia Smart Imagery Solutions

snappet Tabletonderwijs

Boston Scientific

ANTOFAGASTA MINERALS

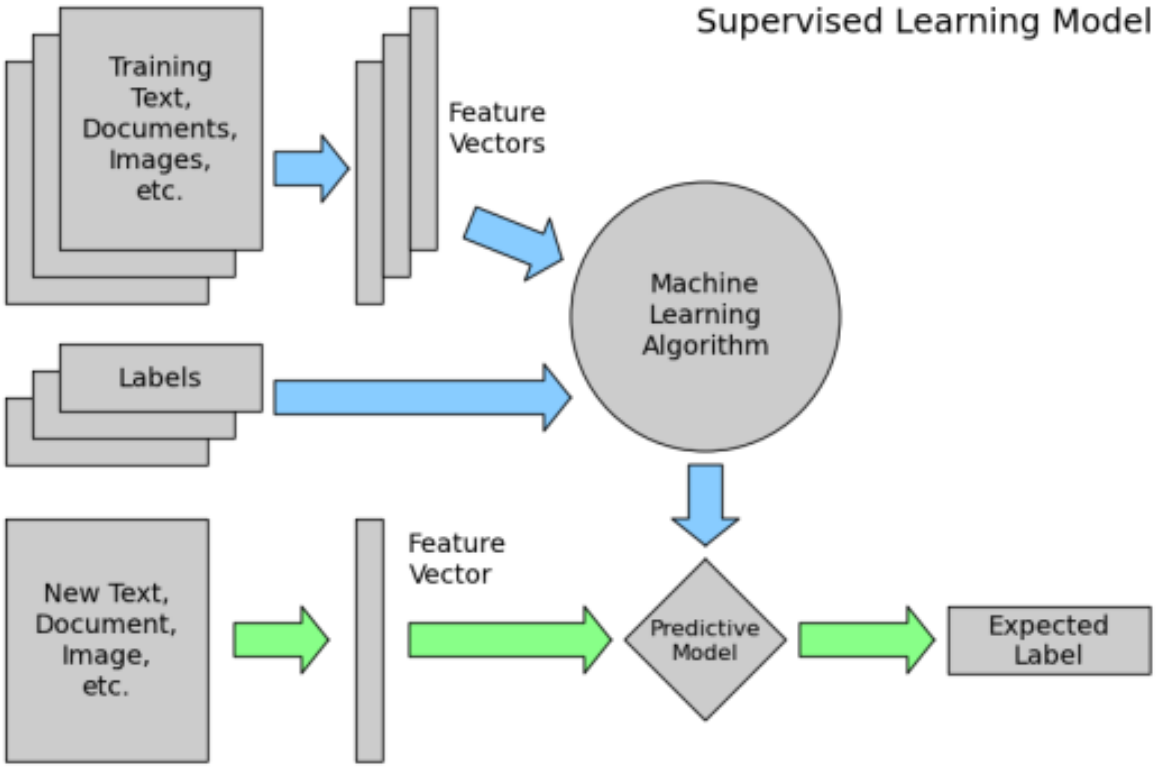
Presentation overview

- ✓ Short introduction of myself and Sioux LIME
- Case: Building a ‘virtual melon expert’ using Deep Learning
- ‘Technical walktrough’:
  - How to practically tackle a case like this?!
  - Methods and techniques used.
  - Technology and tools used.
- Q&A

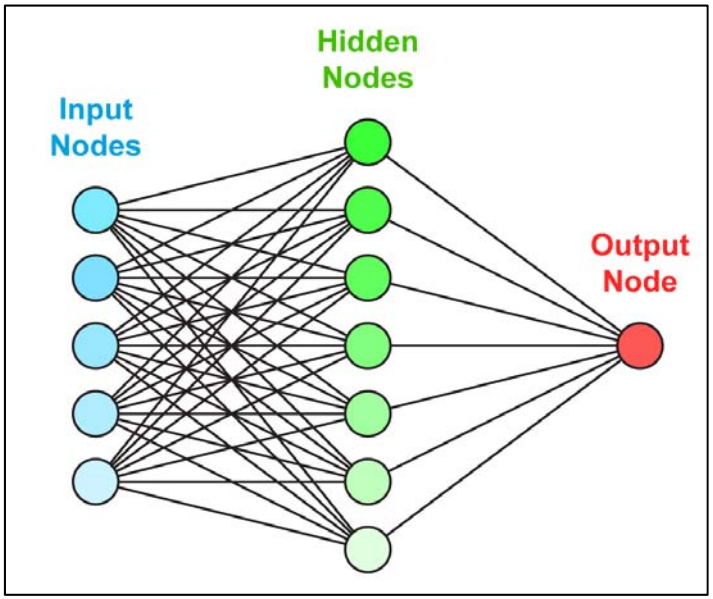
Case: Build a ‘melon expert’

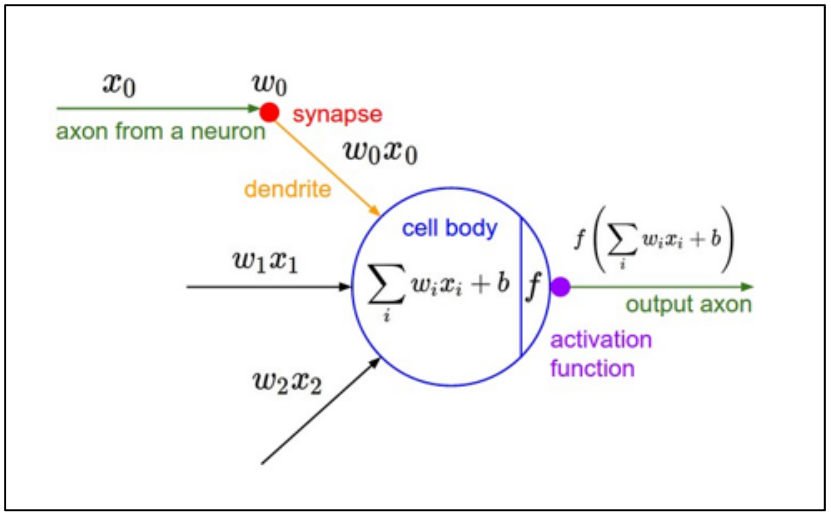
- Build a ‘virtual melon expert’ using Deep Learning technology.
- Using computer vision (only), assess melons on a set of criteria.
- Can we classify melon images according to their ‘net structure’?
- Short project, proof of concept.

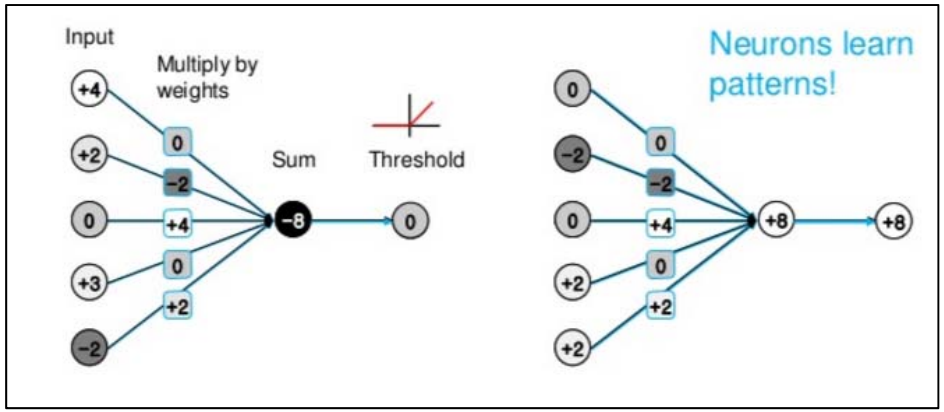
Classification – Supervised Learning

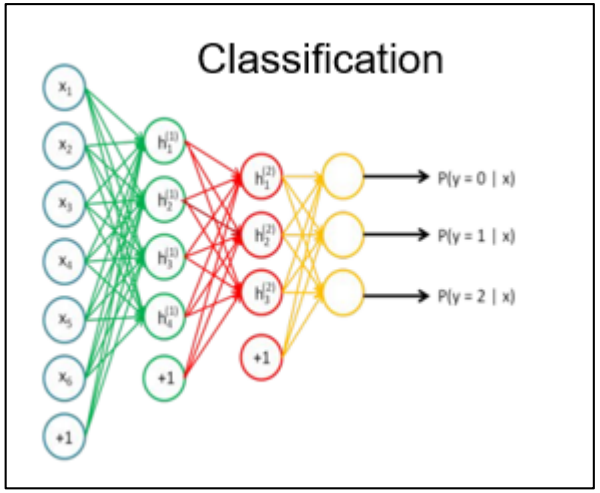


Neural Networks – key concepts

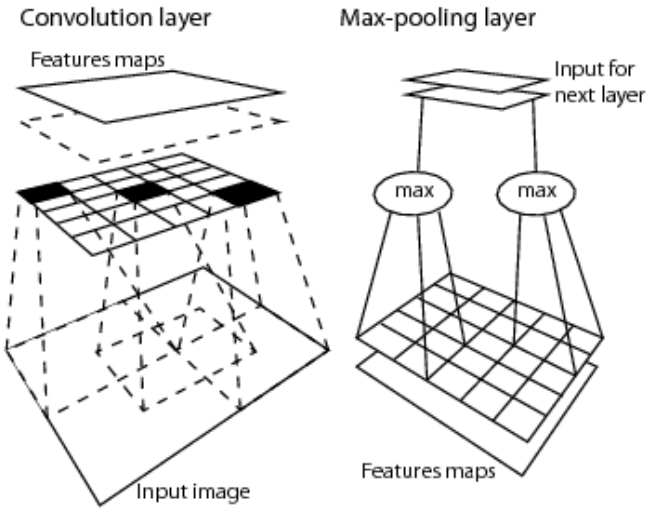


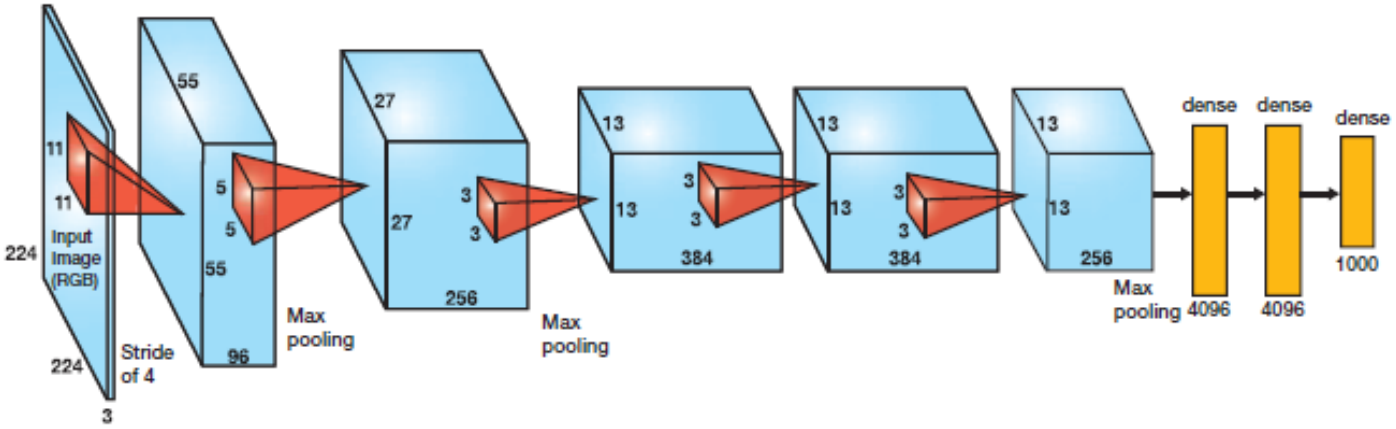






Convolutional Neural Networks (CNNs)



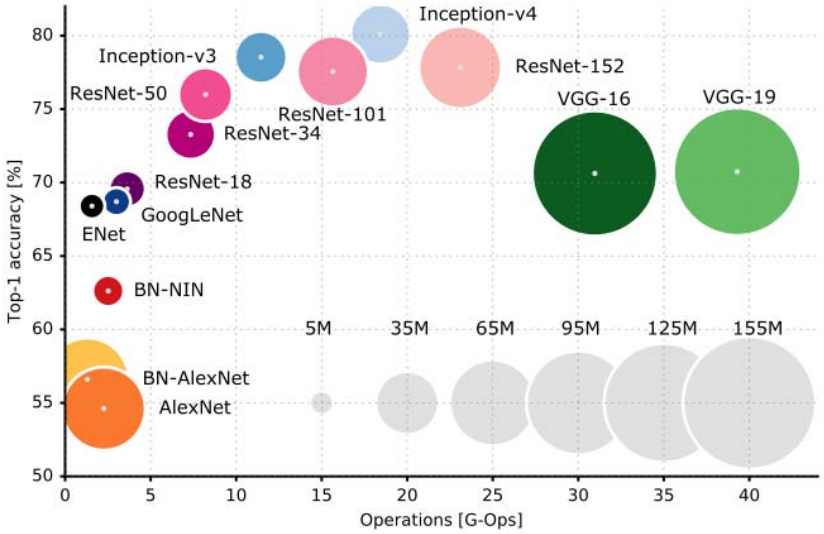


Deep Learning Software

Name	Interface(s)	Remarks
TensorFlow	Python, C++, Java	From Google
Caffe	Python, Matlab	Strong in computer vision
MXNet	Python, R, …	Choice of Amazon
Microsoft Cognitve Toolkit	Python, C++	
Torch	C, Lua	Coming from FB
Theano	Python	Used a lot in academia
Keras	Python	High-level framework
DeepLearningForJ	Java, Scala	Well-documented
….		

- We have used Caffe in this case.

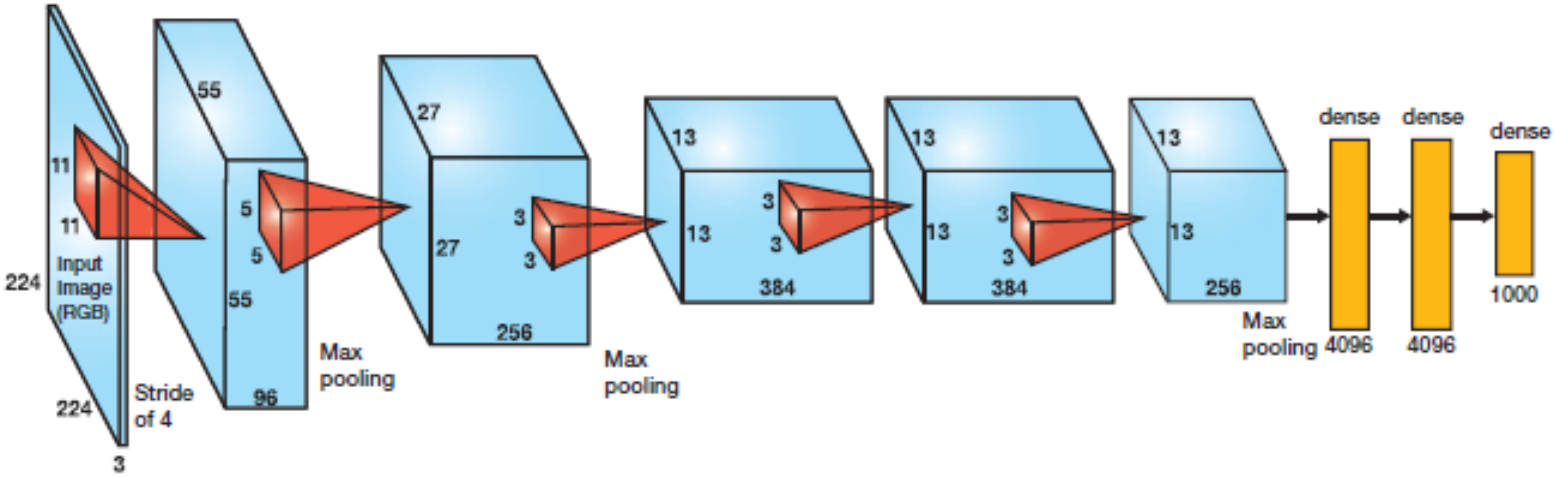
What network (architecture) to use?!



** Image taken from 'AN ANALYSIS OF DEEP NEURAL NETWORK MODELS FOR PRACTICAL APPLICATIONS' by Alfredo Canziani e.a.*

- Take an existing network architecture. 'Do not be a hero!'
- Specific 'project' constraints/requirements usually give guidance.
- We have used AlexNet (and also tried SqueezeNet (not in picture)).

AlexNet-architecture



- AlexNet was developed in 2012 to win the ImageNet challenge.
- It has 5 convolutional layers with 3 intermediate max pooling layers followed by 3 fully-connected layers.
- Number of parameters: around 60 million (!).

Setting up Caffe for GPU-based learning

- How do we actually use Caffe to train and test a model?
- You can install Caffe on a laptop with Ubuntu, OS X or even Windows...
- Ideally, we would like to be able to do GPU-based training.
- Convenient alternative: “in the cloud” at e.g. Amazon Web Services (EC-2):

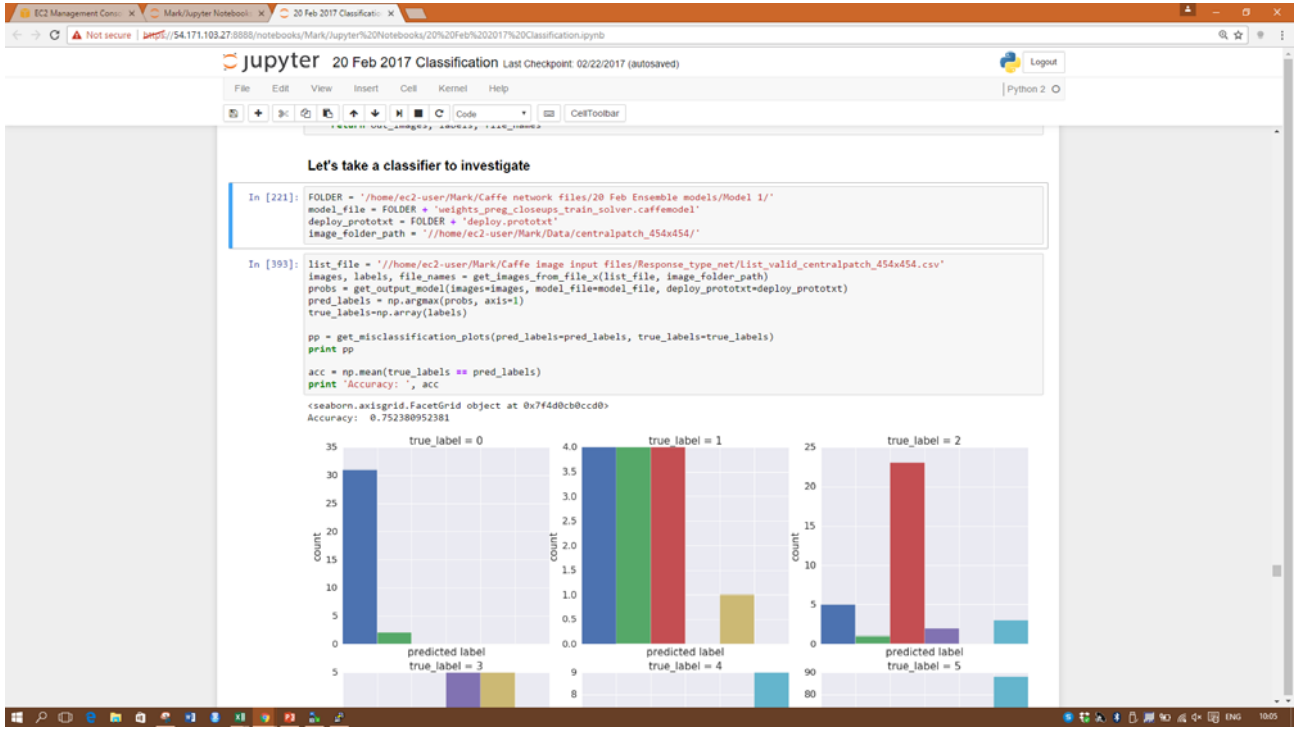
P2 Features

Powerful Performance

P2 instances provide up to 16 NVIDIA K80 GPUs, 64 vCPUs and 732 GiB of host memory, with a combined 192 GB of GPU memory, 40 thousand parallel processing cores, 70 teraflops of single precision floating point performance, and over 23 teraflops of double precision floating point performance. P2 instances also offer GPUDirect™ (peer-to-peer GPU communication) capabilities for up to 16 GPUs, so that multiple GPUs can work together within a single host.

- A p2.xlarge machine (single GPU) costs around 1 USD/h.
- Machine images with Caffe installed are available, so you can start straight away!

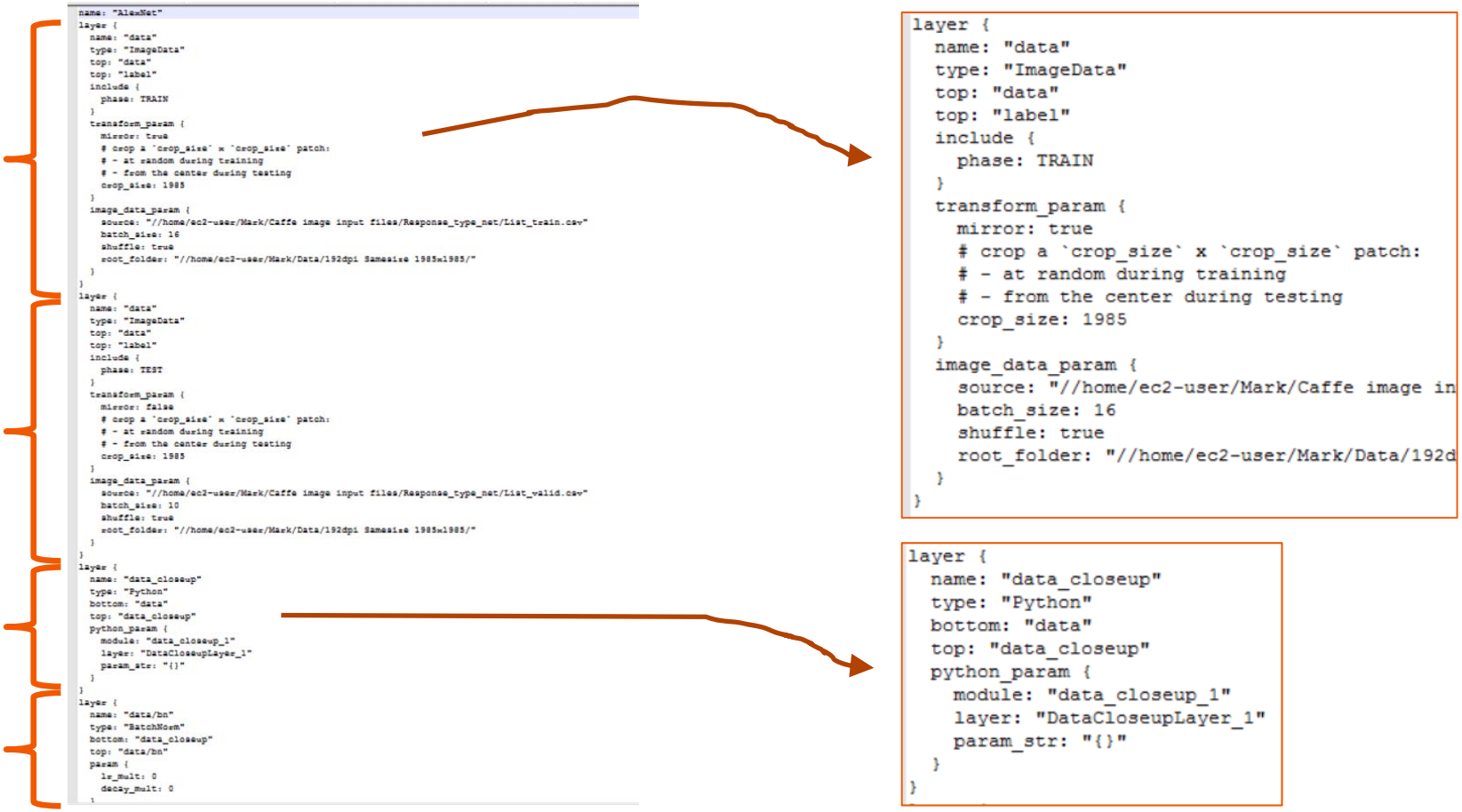
Running Caffe on AWS via a Jupyter Notebook



- A convenient way of working with Caffe (PyCaffe) is using a Jupyter Notebook.
  - Start a Jupyter Notebook server from your instance at AWS
  - Connect to this server through your local browser.
  - Open a notebook (located on your AWS instance).
  - Start coding and running notebook cells from your browser.

Defining a network architecture in Caffe

- In Caffe, a network architecture is defined in a '.prototxt'-file:



```
name: "AlexNet"
layer {
  name: "data"
  type: "ImageData"
  top: "data"
  top: "label"
  include {
    phase: TRAIN
  }
  transform_param {
    mirror: true
    # crop a `crop_size` x `crop_size` patch:
    # - at random during training
    # - from the center during testing
    crop_size: 1985
  }
  image_data_param {
    source: "//home/ec2-user/Mark/Caffe image input files/Response_type_net/List_train.csv"
    batch_size: 16
    shuffle: true
    root_folder: "//home/ec2-user/Mark/Data/192dpi Samesize 1985x1985/"
  }
}
layer {
  name: "data"
  type: "ImageData"
  top: "data"
  top: "label"
  include {
    phase: TEST
  }
  transform_param {
    mirror: false
    # crop a `crop_size` x `crop_size` patch:
    # - at random during training
    # - from the center during testing
    crop_size: 1985
  }
  image_data_param {
    source: "//home/ec2-user/Mark/Caffe image input files/Response_type_net/List_valid.csv"
    batch_size: 10
    shuffle: true
    root_folder: "//home/ec2-user/Mark/Data/192dpi Samesize 1985x1985/"
  }
}
layer {
  name: "data_closeup"
  type: "Python"
  bottom: "data"
  top: "data_closeup"
  python_param {
    module: "data_closeup_1"
    layer: "DataCloseupLayer_1"
    param_str: "{}"
  }
}
layer {
  name: "data/bn"
  type: "BatchNorm"
  bottom: "data_closeup"
  top: "data/bn"
  param {
    lr_mult: 0
    decay_mult: 0
  }
```

```
layer {
  name: "data"
  type: "ImageData"
  top: "data"
  top: "label"
  include {
    phase: TRAIN
  }
  transform_param {
    mirror: true
    # crop a `crop_size` x `crop_size` patch:
    # - at random during training
    # - from the center during testing
    crop_size: 1985
  }
  image_data_param {
    source: "//home/ec2-user/Mark/Caffe image in
    batch_size: 16
    shuffle: true
    root_folder: "//home/ec2-user/Mark/Data/192d
  }
}
```

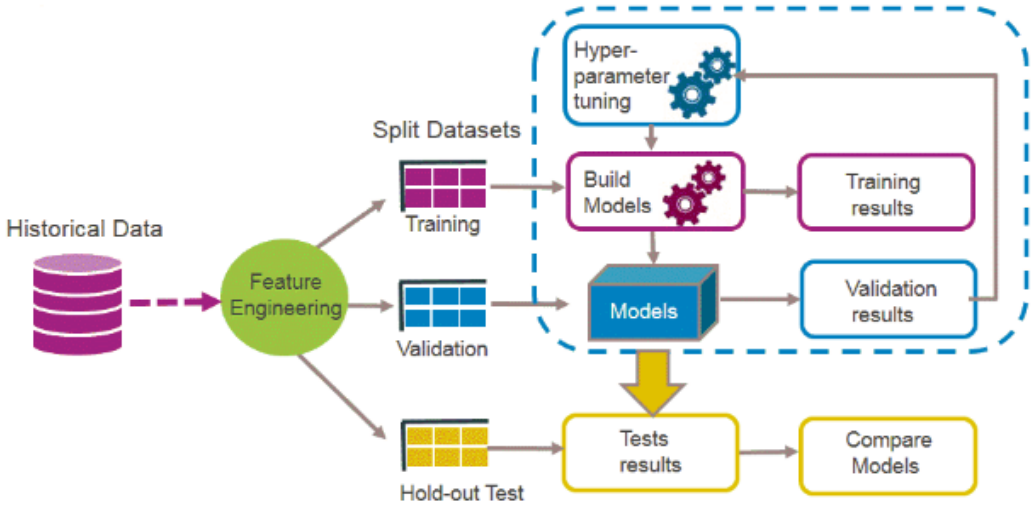
```
layer {
  name: "data_closeup"
  type: "Python"
  bottom: "data"
  top: "data_closeup"
  python_param {
    module: "data_closeup_1"
    layer: "DataCloseupLayer_1"
    param_str: "{}"
  }
}
```

Setting the solver parameters

```
# test_iter specifies how many forward passes the test should carry out.
# After every test_interval training iterations, test_iter * batch_size images are fetched for testing.
test_iter: 1
# Carry out testing every 1 training iterations.
test_interval: 1
base_lr: 0.01
display: 1
max_iter: 320000
lr_policy: "poly"
power: 1
momentum: 0.1
weight_decay: 0.5
snapshot: 5000
snapshot_prefix: "alexnet_cvgj"
random_seed: 0
net: "/home/ec2-user/Mark/Caffe network files/26 januari 2017 Learning rate 1/train.prototxt"
test_initialization: true
iter_size: 2
```

- We need to tell the solver a few things:
  - Learning rate (scheme), momentum.
  - The network we want to train/solve/optimize.
  - Regularization penalty (weight_decay).
  - ….

Preparing our data…

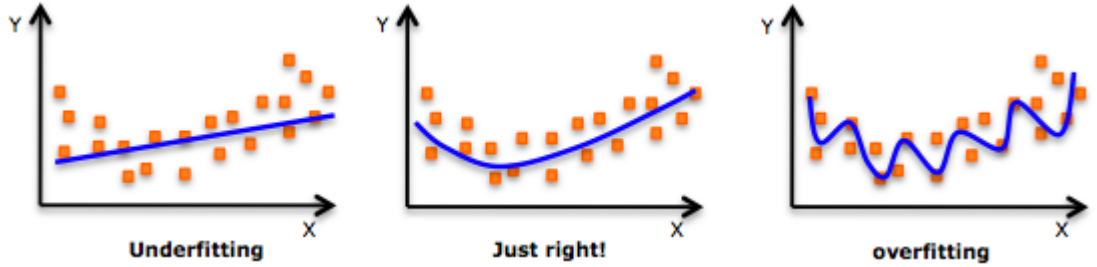


- Generated a train, validation and test set: 60%-20%-20%.

- Resize images to 227x227: AlexNet input dimensions.
- Actually, we have taken central 454x454 patches and downsized these.
- What about data augmentation? → I come back to that quickly.

How to avoid overfitting?
What to do when having limited data?

- In general, Supervised Learning is about the ‘bias-variance’ trade off.



- Underfitting is typically not a problem when using CNNs…but overfitting is…
- We will now look at some methods to avoid overfitting and improve the generalization capabilities of a model.

Transfer Learning

- In our case (and many other cases), data is limited.
- How to optimize 60 million parameters, based on ca. only 1000 images??
- Transfer Learning offers a potential solution for this problem.

- **Transfer learning**: take a ‘pre-trained’ model and use this.
- We have taken a pre-trained variant of the AlexNet-architecture.
- See e.g. the Caffe Model Zoo for many (pre-trained) networks.

- How to use a pre-trained model?
  - As a feature extractor (either from top or middle activations).
  - Retrain / fine-tune the weights of the final layer(s).

Data Augmentation (‘on the fly’)

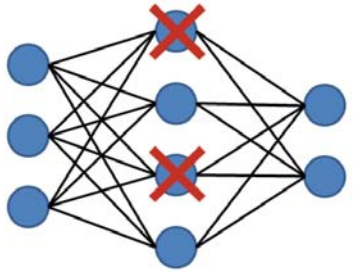
- Augmenting the data (images) is a means to avoid overfitting.
- Preferably, this data augmentation is done on-the-fly (and not off-line).
- This means that an input batch during training is augmented/transformed in real-time to a batch containing modified images.

- Possible ways to augment your data, based on original images:
  - Rotation: rotate an image around a random angle.
  - Translation: translate an image randomly in x and y direction.
  - Rescaling: apply a random scaling factor to the image.
  - Contrast/brightness: adjust contrast/brightness with a random amount.
  - Random crops: take a random ‘patch’ of the (larger) image.

- We implemented this in Caffe using a ‘custom Python Layer’.

Other techniques to improve generalization and/or prevent overfitting

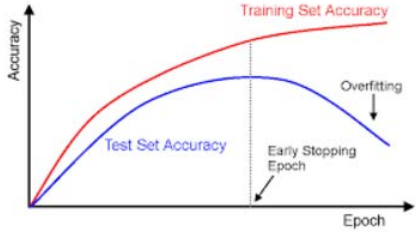
- Drop-out / Drop-connect:
  - Randomly remove neurons/connections from network while training.



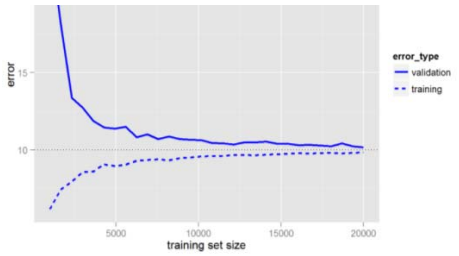
- Weights regularization
  - ‘Penalize parameters for being non-zero’.

$$\lambda \sum_{j=1}^{n} \theta_j^2]$$
Regularization

- Early stopping
  - E.g. via monitoring a validation error.

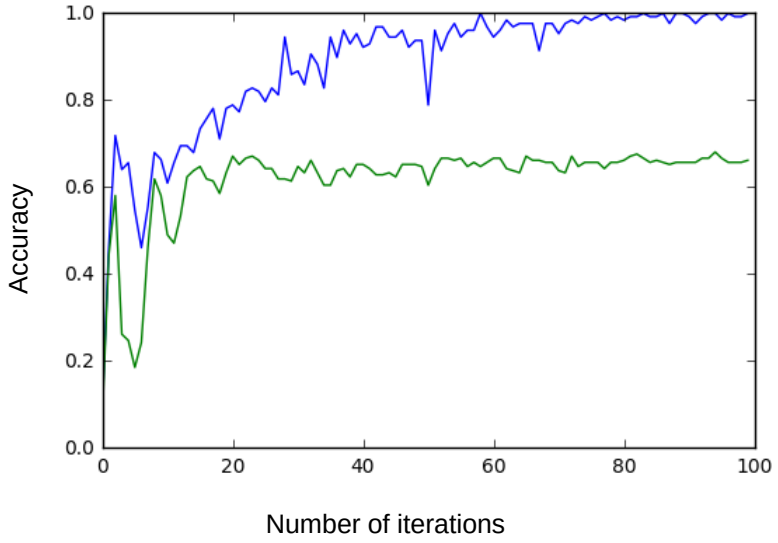


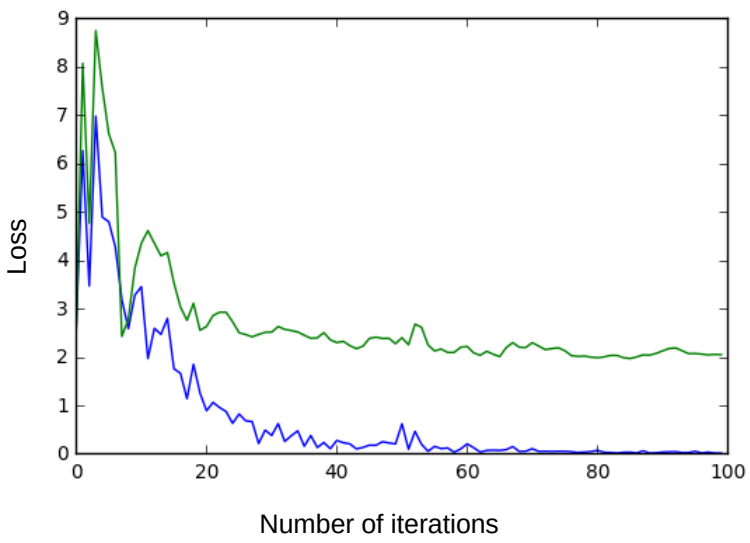
- Collect more data…
  - So-called ‘learning curve’ gives a clue…



Training our classifier

- Network architecture chosen; modified the last layer to have only 6 outputs.
- Configured the solver file with training (hyper)parameters.
- Loaded the pre-trained model weights / parameters.
- Prepared the data for input: central patch + resize.
- Added (“on the fly”) data augmentation.
- Develop some (basic) code to train a classifier and monitor (validation set) performance.
- Start training and ‘tune’ the hyper-parameters.





Questions?!