# Introduction to Tensorflow in Scala

**Xavier Tordoir**

**Schiphol Developers Group**
**13 Feb 2020**

LUNATECH
SIMPLIFY YOUR IT

# Introduction to **Tensorflow in Scala**

Xavier Tordoir, Lunatech

LUNATECH
SIMPLIFY YOUR IT


**JVM development**

**Devops**

**ML & Big Data**

Scala · TensorFlow · play · Spark · kafka · amazon webservices · DC/OS · Java · akka · kubernetes · cassandra · ActiveMQ · Kotlin

LUNATECH

| 105 | 27 | Lots |
|---|---|---|
| Employees | Nationalities | Open source |


We welcome you to one of our offices




LUNATECH

# Outline

ML Concepts

Scala notebooks: Interactive Programming

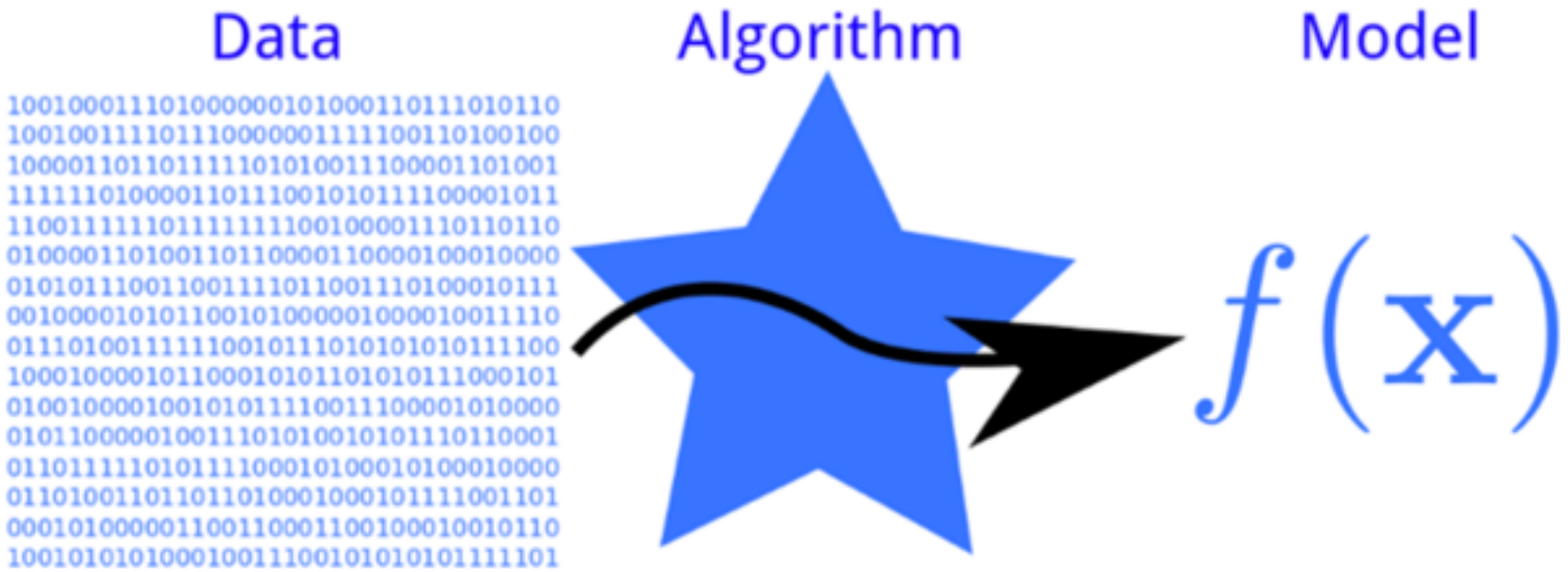
Tensorflow Framework, Overview with Scala Tensorflow

Inference service with Akka http

Managing ML artefacts

Managing development flow

# ML Introduction



# ML Introduction

**Data as a flat table**

```
type Feature = Double
type Label   = Double

val dataSet: Seq[ (Vector[Feature], Label) ]
```

| Surface | Land | Beds | Sidings |
|---|---|---|---|
| 110 | 896 | 2 | 4 |
| 120 | 435 | 3 | 2 |
| 150 | 210 | 4 | 3 |
| 170 | 713 | 4 | 4 |
| 80 | 231 | 4 | 4 |
| 90 | 238 | 3 | 4 |
| 130 | 118 | 2 | 3 |
| 146 | 695 | 4 | 4 |
| 155 | 644 | 4 | 4 |

| Price |
|---|
| 160 |
| 189 |
| 250 |
| 240 |
| 179 |
| 135 |
| 175 |
| 169 |
| 189 |

# ML Introduction

**A model is function a representing a facet of the data**

```
val model: Vector[Feature] => Label
```

| Surface | Land | Beds | Sidings |
|---|---|---|---|
| 110 | 896 | 2 | 4 |

=>

| Price |
|---|
| 160 |

# ML Introduction

## Learning a Model from Data

```
val train: Seq[ (Vector[Feature], Label)] =>  Vector[Feature] => Label
```

| Surface | Land | Beds | Sidings |
|---|---|---|---|
| 110 | 896 | 2 | 4 |
| 120 | 435 | 3 | 2 |
| 150 | 210 | 4 | 3 |
| 170 | 713 | 4 | 4 |
| 80 | 231 | 4 | 4 |
| 90 | 238 | 3 | 4 |
| 130 | 118 | 2 | 3 |
| 146 | 695 | 4 | 4 |
| 155 | 644 | 4 | 4 |

| Price |
|---|
| 160 |
| 189 |
| 250 |
| 240 |
| 179 |
| 135 |
| 175 |
| 169 |
| 189 |

=>

| Surface | Land | Beds | Sidings |
|---|---|---|---|
| 110 | 896 | 2 | 4 |

=>

| Price |
|---|
| 160 |

# ML Introduction

**Training by Minimizing Errors (Loss), e.g. sum of squared errors:**

```
val loss = dataSet.map{
    case (x, y) => y - model(x)
  }
  .map(Math.pow(_, 2))
  .reduce( _ + _ )
```

| Surface | Land | Beds | Sidings |
|---|---|---|---|
| 110 | 696 | 2 | 4 |
| 120 | 435 | 3 | 2 |
| 150 | 210 | 4 | 3 |
| 170 | 713 | 4 | 4 |
| 80 | 231 | 4 | 4 |
| 90 | 238 | 3 | 4 |
| 130 | 118 | 2 | 3 |
| 146 | 695 | 4 | 4 |
| 155 | 644 | 4 | 4 |

| Price |
|---|
| 160 |
| 189 |
| 250 |
| 240 |
| 179 |
| 135 |
| 175 |
| 169 |
| 189 |

| Price |
|---|
| 160 |
| 189 |
| 250 |
| 240 |
| 179 |
| 135 |
| 175 |
| 169 |
| 189 |

$$Loss = \sum_i (y_i - \hat{y}_i)^2$$

Missing pieces yet: How a model is built? What is ‘minimizing?

# ML Introduction

**Models as a vector of parameters**

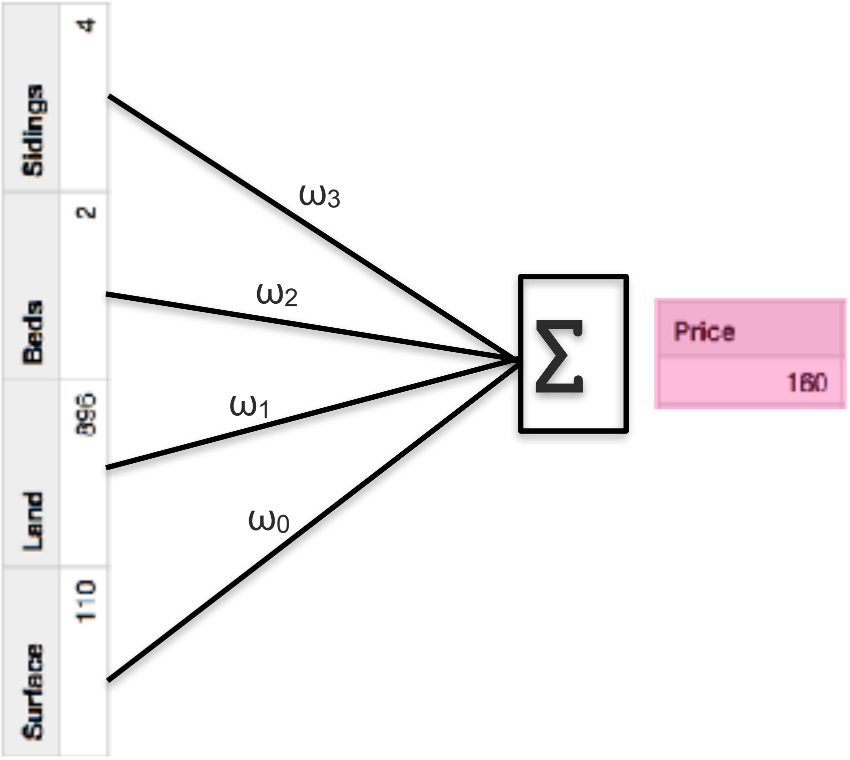
A model is a function, with some parameters, optimisation is finding the best parameters…

Example: A **Linear model** is a linear combination of features:

```
val params: Vector[Double]
val bias: Double

val model = x: Vector[Feature] =>
                    x.zip(params).map( case(wi, xi) => wi * xi )
                                 .reduce( _ + _ )
                      + bias
```

$$\hat{y} = \sum_i x_i \omega_i$$



# ML Introduction

**Optimisation algorithms**

Gradient based methods: How loss varies with each parameters ~ gradient ()

$$\Delta Loss \sim \Delta\omega_i$$

$$\omega_i^* = \omega_i - \gamma\frac{\Delta Loss}{\Delta\omega_i}$$

Loss and gradient are estimated on a subset of data (a batch) = stochastic gradient based methods

Iterations in batches and epochs (a full dataset pass)

# ML Introduction

**Metrics**

| Surface | Land | Beds | Sidings | Price |
|---|---|---|---|---|
| 110 | 896 | 2 | 4 | 160 |
| 120 | 435 | 3 | 2 | 189 |
| 150 | 210 | 4 | 3 | 250 |
| 170 | 713 | 4 | 4 | 240 |
| 80 | 231 | 4 | 4 | 179 |
| 90 | 238 | 3 | 4 | 135 |
| 130 | 118 | 2 | 3 | 175 |
| 146 | 695 | 4 | 4 | 169 |
| 155 | 644 | 4 | 4 | 189 |

After training: model evaluation

E.g.

Root Mean Squared Error in regression

Accuracy in classification (% correct binary prediction)

Metrics are used for model validation on test data not used in training

# ML Introduction

**Data** is multidimensional **Arrays** of **Floating** point values

**Models** are represented as **Arrays** of **Floating** point values and operators

**Training**, **Evaluating** and **Inference** on models are **operations** on these arrays

# Tensorflow: ML Framework

Computing paradigm for **Tensors** transformations

ML capabilities, Linear models and Neural networks

# Tensorflow: tensorflow_scala

Comprehensive tensorflow library in Scala:

https://github.com/eaplatanios/tensorflow_scala
http://platanios.org/tensorflow_scala/
https://brunk.io/deep-learning-in-scala-part-3-object-detection.html

JNI layer to interface the c library

Tensorflow Framework generated with protobuf

Scala API mimics lots of the python API

Reference scala API

# Tensorflow: Notebooks

**Data Science** implies:

- **knowledge** of the data, including its corner cases

- **Exploration** of how to guide the modelling, choosing the right methods for the data

- **Trials and errors**, no possibility to implement functional specs, only model validation

=> Need for **interactive programming**
=> **Notebooks**

Reference scala API

# Tensorflow: Scala Notebooks

https://jupyter.org/
- Python environment
- Kernels to support different languages:
  https://almond.sh/

*Zeppelin*
https://zeppelin.apache.org/

*Spark-notebook*
http://spark-notebook.io/

Scala notebooks

# Tensorflow: Jupyter Notebooks Environments

**Environments**

Local install (Linux with GPU or OSX on CPU)

- Virtualenv for python
- Jupyter
- coursier
- almond to wrap Ammonite

Google colaboratory (CPU, GPU, TPU)

- Preinstalled python/jupyter
- Console for Scala kernel

Custom Server & other providers

# Tensorflow: Overview of the core

Tensor as a Multidimensional array

Variables and Placeholders

Operators

Session and Graph

Abstraction from computing device and Linear Algebra Accelerator

Tensorboard

Linear Regression with Low Level API

# Tensorflow Scala Examples

**Very simple Linear Regression example**

00_tf_intro.ipynb

# Tensorflow: Tensors

Tensor <=> Multidimensional array

Rank = #dimensions

Shape = sizes for each dimension

Type = Type of data (Int, Double, String)

Static Tensors manipulation, read a csv to tensor

# Tensorflow: Dataset API

**Unified API over multiple Data sources**

Nested structure of Tensors (like a collection)

Iterators => basis for batching

Dataset API provides encoding and decoding of Text, Images, Protobuf (TFRecords)

Operations like batches, repeats, shuffle

Sharding for distributed training

Load a Tensor as a Dataset

# Tensorflow: Estimators API

**Encapsulates everything needed for training a Model:**

Model Signature:

- Input to define input types (like a placeholder shape)
- TrainInput to define prediction types (Again like a placeholder)

Model Function and Parameters:

- Build Neural network topologies with a simple API
- Trainable Parameters

Loss function, Optimizer, Metrics

Hooks for:

- Save models regularly (Graph structure and parameters) — checkpoint files
- Log metrics and Tensorboard

Define Model and Training

# Tensorflow Scala Examples

**Using Datasets and Estimators**

01_estimators_datasets.ipynb

# Linear Models to Deep Learning

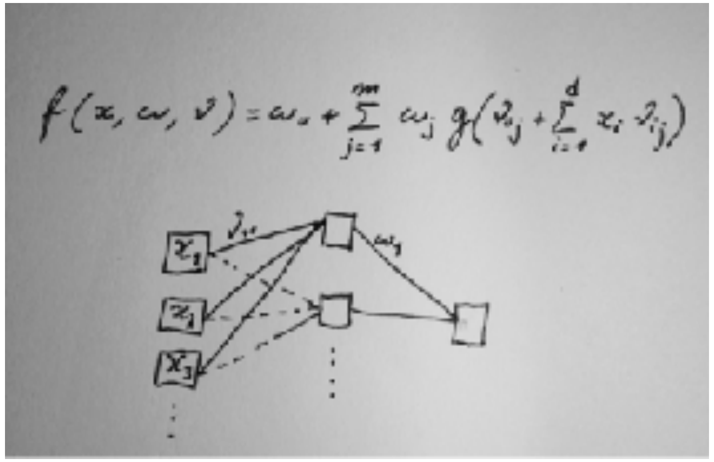
### Linear Model

$$\hat{y} = f(x, w) = \sum_{j=1}^{m} w_j \, g_j(x)$$

```
Linear[Float]("Layer_0", 1)
```

### Multi Layer Perceptron

```
Linear[Float]("Layer_0", 128)>> ReLU[Float]("Layer_0/Activation", 0.1f) >>
Linear[Float]("Layer_1", 64) >> ReLU[Float]("Layer_1/Activation", 0.1f) >>
Linear[Float]("Layer_2", 32) >> ReLU[Float]("Layer_2/Activation", 0.1f) >>
Linear[Float]("OutputLayer", 10)
```

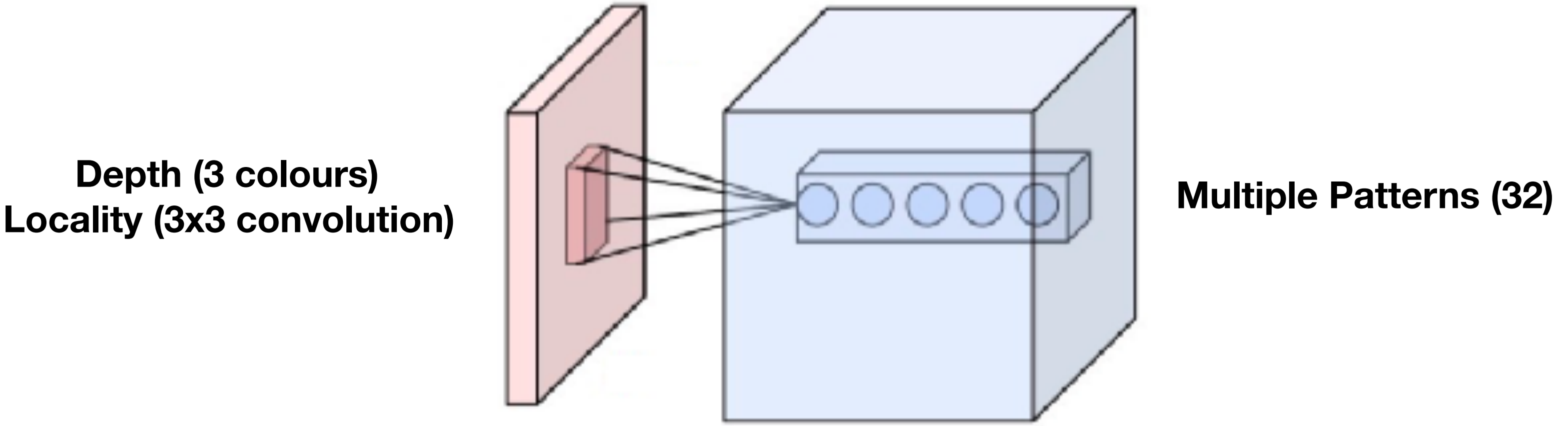



Deep is defined by accumulation of layers…and parameters

# Convolutional Neural Networks

```
tf.learn.Conv2D("Layer_1/Conv2D", Shape(3, 3, 3, 32), stride1 = 1, stride2 = 1, padding = ValidConvPadding) >>
```

## Convolution Layer

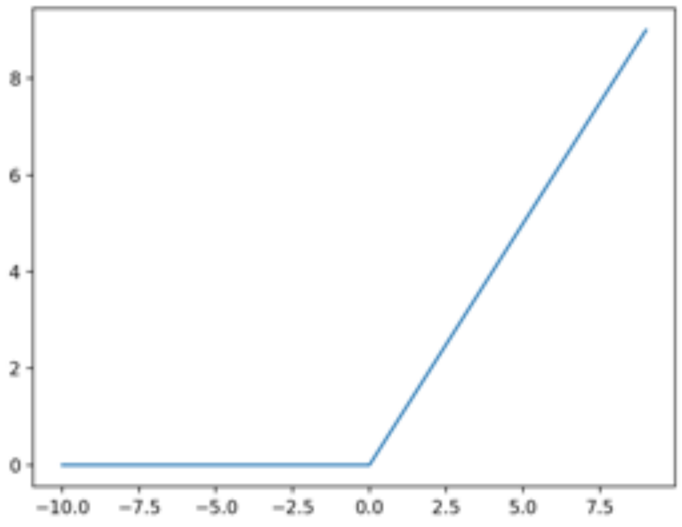



**Each Pattern is scanned against the image and scores are computed in the layer**
**Training learns the patterns that work well…**

# Convolutional Neural Networks, Activation

```
tf.learn.Conv2D("Layer_1/Conv2D", Shape(3, 3, 3, 32), stride1 = 1, stride2 = 1, padding = ValidConvPadding) >>
tf.learn.AddBias("Layer_1/Bias") >>
tf.learn.ReLU("Layer_1/ReLU", alpha = 0.1f) >>
```

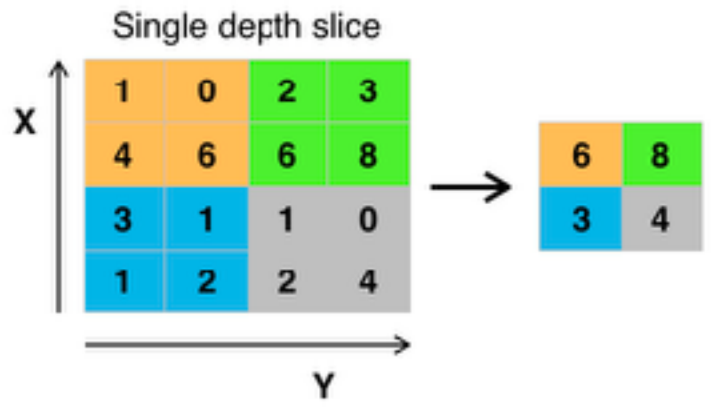
## Rectifying Layer…Breaking positive/negative symmetry



# Convolutional Neural Networks, subsampling

```
tf.learn.Conv2D("Layer_1/Conv2D", Shape(3, 3, 3, 32), stride1 = 1, stride2 = 1, padding = ValidConvPadding) >>
tf.learn.AddBias("Layer_1/Bias") >>
tf.learn.ReLU("Layer_1/ReLU", alpha = 0.1f) >>
tf.learn.MaxPool("Layer_1/MaxPool", windowSize = Seq(1, 2, 2, 1), stride1 = 2, stride2 = 2) >>
```

**MaxPooling: Scale down the 2-D size by selecting the best score per window pool**

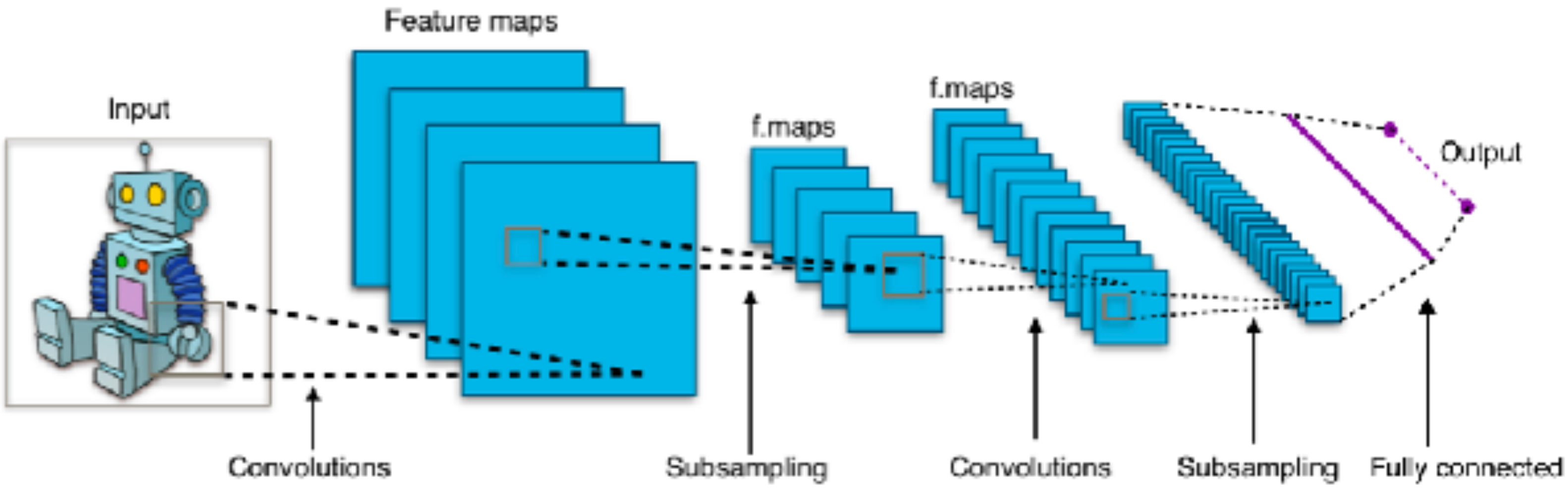



**This creates one hierarchy level of detected patterns with locality**

# Convolutional Neural Networks

```
tf.learn.Conv2D("Layer_1/Conv2D", Shape(3, 3, 3, 32), stride1 = 1, stride2 = 1, padding = ValidConvPadding) >>
tf.learn.AddBias("Layer_1/Bias") >>
tf.learn.ReLU("Layer_1/ReLU", alpha = 0.1f) >>
tf.learn.MaxPool("Layer_1/MaxPool", windowSize = Seq(1, 2, 2, 1), stride1 = 2, stride2 = 2) >>
tf.learn.Flatten("Layer_2/Flatten") >>
tf.learn.Linear("Layer_2/Linear", units = 512) >>
tf.learn.ReLU("Layer_2/ReLU", 0.01f) >>
tf.learn.Linear("OutputLayer/Linear", 2)
```

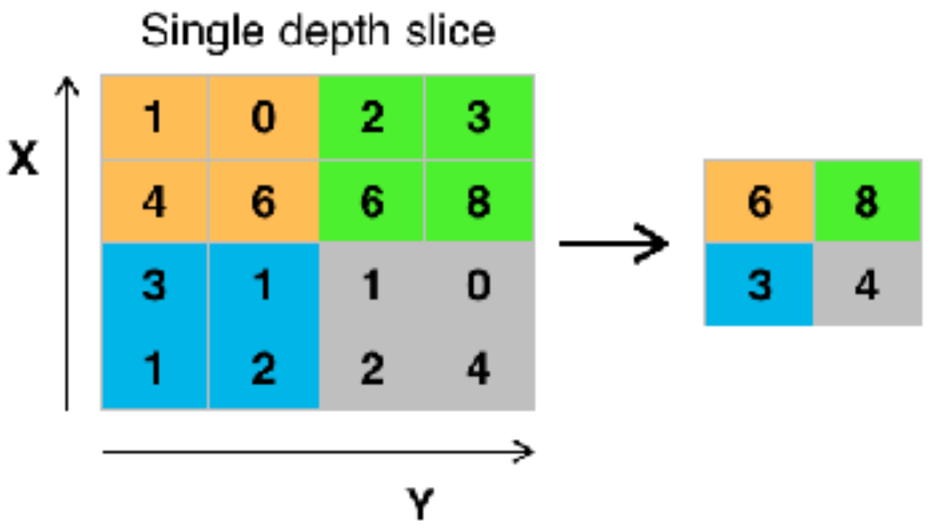
## And repeat…



# Linear Models to Deep Learning

### Convolutional Neural Network

```
tf.learn.Conv2D("Layer_1/Conv2D", Shape(3, 3, 3, 32), stride1 = 1, stride2 = 1, padding = ValidConvPadding) >>
tf.learn.AddBias("Layer_1/Bias") >>
tf.learn.ReLU("Layer_1/ReLU", alpha = 0.1f) >>
tf.learn.MaxPool("Layer_1/MaxPool", windowSize = Seq(1, 2, 2, 1), stride1 = 2, stride2 = 2) >>
tf.learn.Flatten("Layer_2/Flatten") >>
tf.learn.Linear("Layer_2/Linear", units = 512) >>
tf.learn.ReLU("Layer_2/ReLU", 0.01f) >>
tf.learn.Linear("OutputLayer/Linear", 2)
```

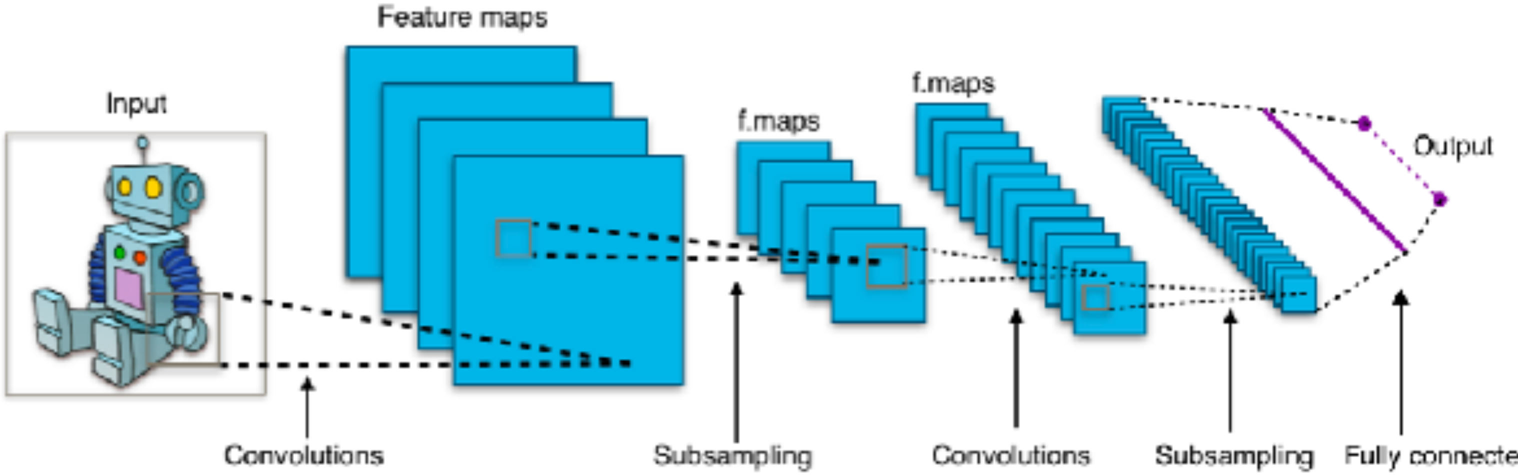



Deep is defined by accumulation of layers…and parameters

# Linear Models to Deep Learning

**Convolutional Neural Network**

```
tf.learn.Conv2D("Layer_1/Conv2D", Shape(3, 3, 3, 32), stride1 = 1, stride2 = 1, padding = ValidConvPadding) >>
tf.learn.AddBias("Layer_1/Bias") >>
tf.learn.ReLU("Layer_1/ReLU", alpha = 0.1f) >>
tf.learn.MaxPool("Layer_1/MaxPool", windowSize = Seq(1, 2, 2, 1), stride1 = 2, stride2 = 2) >>
tf.learn.Flatten("Layer_2/Flatten") >>
tf.learn.Linear("Layer_2/Linear", units = 512) >>
tf.learn.ReLU("Layer_2/ReLU", 0.01f) >>
tf.learn.Linear("OutputLayer/Linear", 2)
```




Deep is defined by accumulation of layers…and parameters

# Tensorflow: More models

**Resources for published models:**

https://github.com/tensorflow/models

https://tfhub.dev/

# Tensorflow: Takeaways

Data access API

Tensor representation and manipulation

Deeplearning Modelling API

Serializable Computing Graphs (Interoperable)

Access to Accelerated Computing Devices (gpu and tpu)

# Tensorflow: Serving from Scala Demo

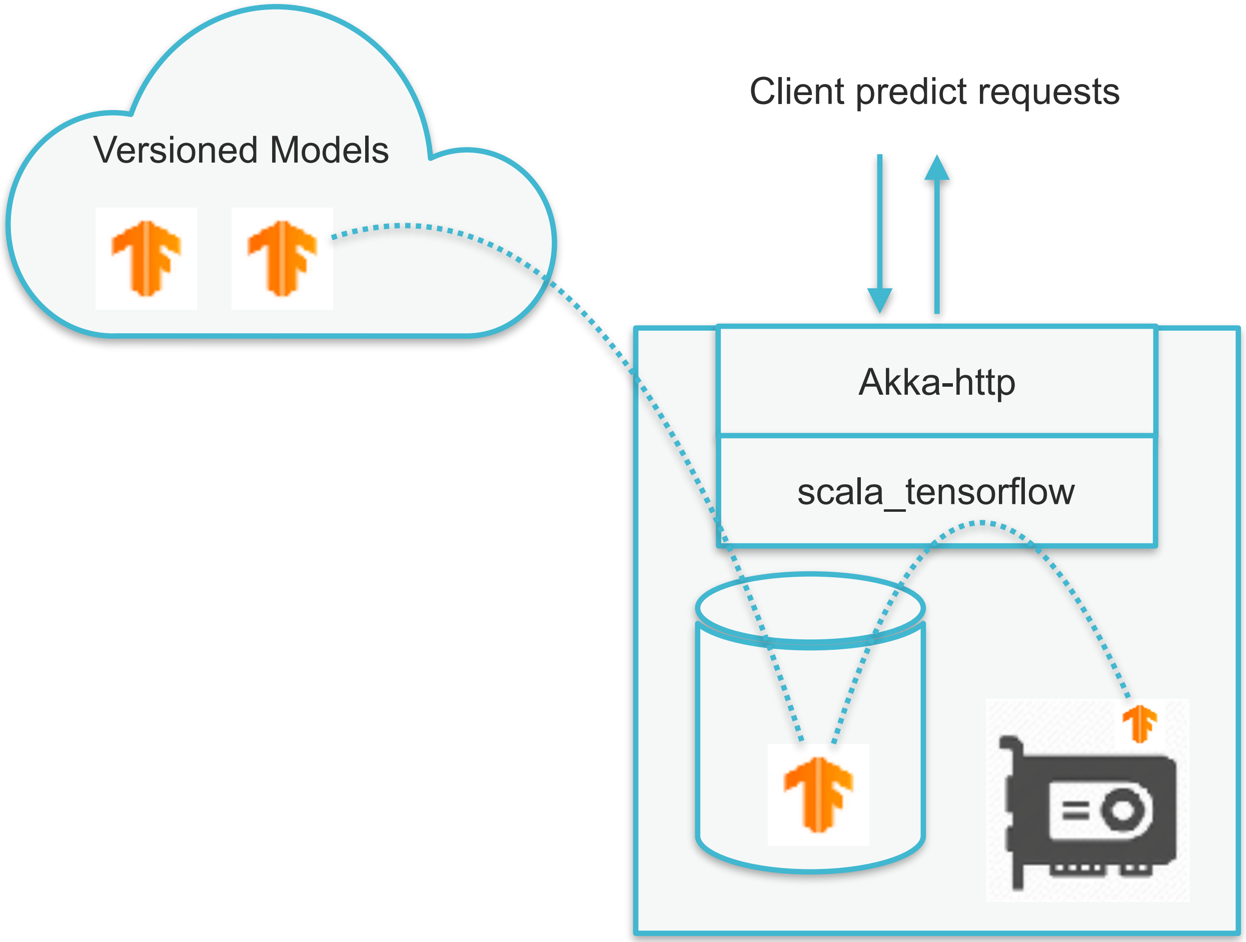
**Goals**

- Serve some common AI task with inherent complexity (e.g. object detection in images for anonymization)

- Leverage Scala strong concurrency and scheduling capabilities … ***akka-http***

- Consume ML models in with a degree of abstraction: A single use case should have the same methods signatures and specific models easily interchangeable … ***config-based model hub***

# Tensorflow: Serving from Scala Demo




**Goals**

- **Serve** common **AI** task (e.g. object detection in images for anonymization)

- Leverage Scala concurrency and scheduling capabilities … ***akka-http***

- Consume ML models: input signatures and interchangeable models … ***config-based model hub***

# Tensorflow: Serving from Scala Demo

**Start in a notebook to build the app skeleton**

02_akkahttp_tf.ipynb

03_tf_client.ipynb

# Tensorflow: Serving from Scala Demo

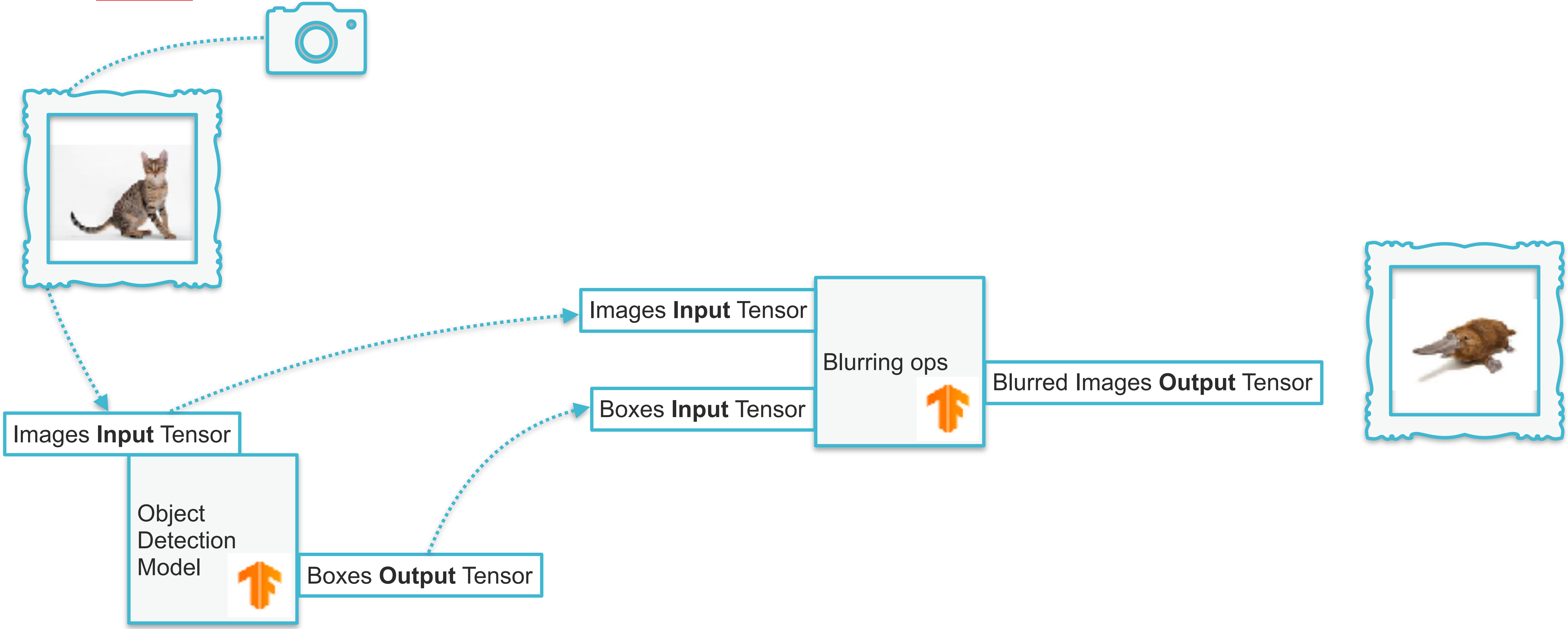
Move to sbt project

Abstract Model Service

Download from hub

Start service with different models

# Tensorflow: Anonymization use case



# Tensorflow: Unit Testing a Model

If the Tensorflow-side model signature is defined, how to unit test the code using it without relying on a large published model and some complex or unpredictable examples?

Just like database tables mocks, we need a mock of the model the can fit as test resources:

- Build a Tensorflow model with the required signature
- The Model should return predictable values for simple known cases
- And carry few computations

Example for object detection:

- return always the entire image and one defined quadrant
- class is always a pre-defined colour name
- score is the fraction of pixels with that exact colour

Such a model is small in size, and just requires a very small synthetic image to implement unit tests

# Tensorflow: Integration Tests

Integration must use a real model from a hub, to check that Data Science team models are compatible with the requirements.

It would be bad that the signature changes even slightly with little transparency…

# ML and Engineering: Team work

- Multiple environment and dependencies management is a burden to care about

- Favour Data Science artefacts as data, not code

- Define contracts: Model signatures, publishing protocols, Integration and Unit tests DS must comply with

- Leave room for experiments but value automation and maturity of models delivery

# Scala Tensorflow alternatives

scalapy

- https://github.com/shadaj/scalapy
- JEP and python environment
- Delicate threads management

Java API

- Official Tensorflow API on JVM

Tensorflow serving

- C++ compile / docker
- Manages models versions
- gRPC / REST

Actually…other engines (lite, js, python, rust, swift…)

# General conclusions

ML Engineering is progressing

- Data is available
- Computing is available
- Software Engineering and Deployment is mature

Tensorflow brings important features:

- Serializable and interoperable Models
- Model signatures can be defined
- Serving gRPC … the client doesn’t even need tensorflow
- Tensorflow Hub defines policies to publish models in a consistent manner
- Tensorflow Extended provides full pipelines management

Scala doesn’t need to be bitten by the snake when it comes to ML
Investing in getting lots done as Tensorflow operations is a valuable investment

# Merci !

xavier.tordoir@lunatech.nl
twitter.com/xtordoir

www.lunatech.com

LUNATECH
SIMPLIFY YOUR IT