Tutorial: BEAT: An online web-platform for reproducible research

Anjos, El-Shafey, Abbet, Gaist, Tarsetti & Marcel
Contact: beat-devel@googlegroups.com

Face & Gesture 2015, May 8, 2015
Agenda

Motivation   Reproducibility
First contact  Search, experiment!
Going deeper   Core objects, design
More          Other Featurettes
The future    What comes up next - your input
Important info

- This is beta software
- It is the first time we give out a public tutorial
- Your feedback is important
- Reporting issues is even more important
- It’s about try/error, interact, contribute!
The Scientific Method

Graphical view

1. **The Scientific Method**
   - Observe natural phenomena
   - Formulate Hypothesis
   - Test hypothesis via rigorous Experiment
   - Establish Theory based on repeated validation of results

2. **The Actual Method**
   - Make up Theory based on what Funding Agency Manager wants to be true
   - Design minimum experiments that will prove show suggest Theory is true
   - Publish Paper: rename Theory a “Hypothesis” and pretend you used the Scientific Method
   - Defend Theory despite all evidence to the contrary
A Scalable Formulation of Probabilistic Linear Discriminant Analysis: Applied to Face Recognition

Laurent El Shafey, Chris McCool, Roy Wallace, and Sébastien Marcel

Appendix A
Mathematical Derivations

The goal of the following section is to provide more detailed proofs of the formulas given in the article for both training and computing the likelihood.

The following proofs make use of a formulation of the inverse of a block matrix that uses the Schur complement. The corresponding identity can be found in [1] (Equations 1.1 and 1.8).

\[
L \cdot M^{-1} N = \begin{bmatrix} R \hline -O^T \cdot \text{NR} \end{bmatrix} - \text{RMO}^{-1} \cdot \text{NRMO}^{-1}
\]

where we have substituted \( R = (L \cdot M)^{-1} N \).

\[
\text{where the} \ L \text{ block is distributed as the identity matrix (Equation C.7 of [2]), which states that:}
\]

\[
(L + \text{MON})^{-1} = L^{-1} + L^{-1} M (O_1^{-1} + L_1^{-1} M)^{-1} M L^{-1}
\]

A. Scalable training

The bottleneck of the training procedure is the expectation step (E-step) of the Expectation-Maximization algorithm. This E-step requires the computation of the first and second order moments of the latent variables.

1. Estimating the first order moment of the latent variables: The most computationally expensive part when estimating the latent variables is the inversion of the matrix \( \mathcal{P} \) (Equation 27). This matrix is block diagonal, the two blocks being \( \mathcal{P}_G \) (Equation 28) and a repetition of \( \mathcal{P}_G \) (Equation 29).

\[
\mathcal{P} = \begin{bmatrix} \mathcal{P}_G & \mathcal{P}_G \hline \mathcal{P}_G & \mathcal{P}_G \end{bmatrix}
\]

The inverse of \( \mathcal{P}_G \) is equal to the matrix \( \mathcal{G} \), defined by (30). This matrix is of constant size (\( \mathcal{G} = \mathcal{G} \)) irrespective of the number of training samples for the class. In addition, the inversion of \( \mathcal{P}_G \) can be further optimised using the block matrix inversion identity introduced at the beginning of this section, leading to:

\[
\mathcal{P}_G^{-1} = \begin{bmatrix} \mathcal{G} & \mathcal{G} \hline \mathcal{G} & \mathcal{G} \end{bmatrix}^{-1} = \begin{bmatrix} \mathcal{G}^{-1} & 0 \hline 0 & \mathcal{G}^{-1} \end{bmatrix}
\]

D. Performance of fusion systems

In our last experiment, we compare the four face verification systems when fused with ALL counter-measures using PLR fusion scheme. Firstly, we illustrate how fusion changes the score distribution for each of them separately in Figure 14. Then, in Figure 15 we compare which of the fused systems performs the best.

While Figure 10 shows that the spoofing attacks of Replay-Attack are in the optimal category when fed to the baseline face verification systems, Figure 14 illustrates that their effectiveness has vastly changed after fusion. The score distribution of the spoofing attacks is now mostly overlapping with the score distribution of the zero-effort impostors, allowing for better discriminability between the positive class and the two-negative classes. The results are reflecting this observation even when the threshold is obtained using the last scenario. SFAR has dropped to less than 0.5%

The comparison between the EPSC curves given in Figure 11(a) and Figure 15(a), confirms the above observations; while HTER increases rapidly with \( \omega \) and reaches up to 25% for some of the baseline systems, it increases very mildly and does not exceed 4.1% for the fused systems. The major augmentation of the robustness to spoofing of the systems after
Motivation

"I think you should be more explicit here in step two."

How many times?
Motivation

Crossed a publication and openly decided to **ignore it because it would be too hard to apply** those doubtful results on your research?
Worked day and night to incorporate some results on your own work but:

- There were untold parameters that needed adjustment and you couldn’t get hold of them?
- Realized the proposed algorithm worked only on the specific data shown at the original paper?
- Realized that something did not quite add up in the end?
Motivation

Had to take over the work from another colleague that left and had to start from scratch - months into programming to make things work again?
Would have liked to **replay to someone about your work**, but you couldn’t really remember all details when you first made it work? Or you **could not make it work** at all?
An article about computational science in a scientific publication is not the scholarship itself, it is merely advertising of the scholarship. The actual scholarship is the complete software development environment and the complete set of instructions which generated the figures.

D. Donoho, 2010

1http://biostatistics.oxfordjournals.org/content/11/3/385.long
Enter “Reproducible Research” (RR)²

One term that aggregates work comprising of:

- a **paper**, that describe your work in all relevant details
- **code** to reproduce all results
- **data** required to reproduce the results
- **instructions**, on how to apply the **code** on the **data** to replicate the results on the **paper**.

²http://reproducibleresearch.net
With respect to “Replication”³

- RR is the practice of presenting computational research such that members of a scientific community may easily reproduce and verify the results.
- Distinct from “scientific replication”
- Reproducibility i.e. RR verifies an experimental result with the same data and procedures
- Replication strengthens evidence about a scientific theory through different data and procedures

³http://jblevins.org/log/rep
Levels of Reproducibility

With respect to an independent researcher (reader):

0 Irreproducible
1 Cannot seem to reproduce
2 Reproducible, with extreme effort (> 1 month)
3 Reproducible, with considerable effort (> 1 week)
4 Easily reproducible (∼ 15 min.), but requires proprietary software (e.g. Matlab)
5 Easily reproducible (∼ 15 min.), only free software

---

Work Cycle

Need a program / Extend a program

Needs improvements

Now to testing

Extend work

Idea works
Pipeline

[Diagram of pipeline process]

1. Writer
   - Software/Algorithms
   - Protocols
   - Data

2. Data
   - Generates
   - Assets (Figures & Tables)

3. Assets (Figures & Tables)
   - Composes
   - Publications

4. Reader
   - Publications
   - Composes
   - Assets (Figures & Tables)
   - Generates
   - Data
   - Protocols
   - Software/Algorithms
Finally, writing and distributing code and data takes time...
Why?

Boost your research **impact (visibility)**:

- **Lower entrance barrier** to your publications
- The current number of RR papers is **rather small** - you have a clear chance to stand out today:
  - Only **10% of TIP** papers provide source code\(^5\).
- Statistically, your work is **more valuable** if it is RR:
  - **13 out of the top 15 most cited** articles in TPAMI or TIP provide (at least) source code
  - The average number of citations for papers that provide source-code in TIP is **7 fold** that of papers that don’t.

\(^5\) **Code Sharing is Associated with Research Impact in Image Processing**, Patrick Vandewalle, 2012
In practice

Organize yourself so you are *always* doing RR

Work as a team to:

- Organize **basic tools** so that you have a library that is documented and re-usable by all
- Organize **the data** so it is easy to replay analysis protocols by all
- Write and distribute **applications** that use your basic tools and data to **generate interesting results**.

Benefits:

- Research is always kept reproducible
- People help each other in case of problems
- New colleagues can start (nearly) immediately to produce high-quality results.
What can be improved?

- Downloading and storing **data** may be a privacy concern in many countries:
  - Need to work-out space for the growing number of samples
  - Not all databases are distributable (e.g. *forensic data*)

- **Software** management and installation can be hard
  - Software gets outdated: constant quality and integration
  - Plan for errors: re-distribution mechanism

- **Computing** can be limited
BEAT: A web platform for RR

- **Accessible**: no need to install extra software
- **Intuitive**: graphically connect blocks to run experiments
- **Social**: engagement gets you more processing power
- **Productive**: search the state-of-the-art by any filtering criteria
- **Data Privacy**:
  - No need to handle large-scale databases
  - Can run on un-distributable data (e.g. proprietary databases)
- **Assurance**:
  - fair (reproducible) evaluations of algorithms
  - online attestations for all produced results
- **Free**: build on open-source software and standards
EU-FP7 Project Proposal

Diagram showing the interconnect between back-end (computing cloud) and front-end with storage, databases, user results, uploaded algorithms, worker nodes, scheduler, ranking, uploader, downloads, documentation, and a graph titled BEAT :: My results with algorithm XYZ (Classifier).
Final Design

You (via internet)

Frontend

Web Server + RESTful API

Scheduler + Workers Nodes

Backend

Object Repository

Cache

Data

Execution of Experiments

commands <-> data
Visit a beta of BEAT

https://www.beat-eu.org/platform/
Let’s all register

This should allow you to contribute to the platform.

- Go to https://beat-eu.org/platform
- To register, click on the green button ”SIGN-UP”
- Fill-in all fields and check the ToS box
- Go to your e-mail box and confirm the registration
- Log-in with your password and go to your user page
- Browse publicly available objects
Experiment list

At the first contact as a logged-in user with the platform, you immediately have access to all experiments produced by you or shared with you.

Click any of them!
In the experiment display, you can access information about the experiment that was run:

- Results produced by its analysis, including plots and single numbers
- Parameters used by this experiment (single values, processing environments, execution time, work flow, etc)
- Each experiment produces an enormous amount of information that the platform tracks
Searching

Viewing a single experiment is not very useful. Let’s try to get an overview of results, for example, on “eigenface”.

• At the top search bar, type “eigenface”
• The platform will search any component with that string on the name
• If required, refine your search using the filters you have available - you can refine your search quite substantially using these selectors
• Notice you can only compare results produced with the same analysis algorithm
• After filtering, the table shows a few columns (these are the default for the analysis for these experiments), to display more, click on the “gear” icon. You can display the aggregated plots, for example.
Let’s search for the best results on MOBIO (protocol “male”) visible to you at the platform.

On the filter bar:

- choose database-name and then is mobio/1
- choose protocol-name and then is male
- choose analyzer and then is eerhter_postperf_iso.
Start contributing

We now want to run an experiment on our own, using the eigenface/AT&T setup as basis:

- Go to the user micro-site (2 icons left to your user icon)
- Search/click on the experiment tutorial/tutorial/eigenface/1/eigenfaces_5comp
- On the top-right, click on “Fork”
- Tune the experiment parameters on the right
- Choose a label
- Hit “Go!”
- Go back to the search (search for “eigenface” on the top-bar), compare your experiment with the others
- Add:
  - choose analyzer and then is eerhter_postperf_iso.

Do you do better or worse? Did you check your ROC? (gear icon)
Going deeper

The BEAT Platform provides an easy way to run pattern recognition experiments involving software components.

We hope that after some iterations, public material will allow for a large number of baselines and state-of-the-art results to be *easily accessible* and *reproducible*.

You can contribute in many levels:

- running experiments produced by others.
- **contributing** back to create your own unique experiments, shared to any number of parties you want.
BEAT is designed as an *opt-in* platform

- All your actions and results are kept private until you choose to change visibility, by clicking on the “Private” icon
- Once you (or an authorized peer) use an item, it cannot be modified (or erased) anymore
- So that an item is modifiable, it must not be *in use*
- If an item is not modifiable, it can be made modifiable again by removing all items pointing to it
Teams

It is possible for users to organize themselves using “Teams”.

Interact: Open the “Teams” tab, try to create a team and add users.

You can use “Teams” to share your contributions with a specific set of users.
Semantics: Toolchains (or Workflows)

A key concept in experimentation at the platform is the idea of *Toolchains*.

Toolchains are decomposable into blocks.
Toolchain Example: trivial Eigen-faces

Simple: no evaluation (test) set, threshold \textit{a posteriori}
Toolchain Example: trivial Eigen-faces

Translation as a BEAT toolchain

- All relations are explicit: I/O, work flow
- We do not define here which algorithms, datasets, analyzers, formats or other results to run and produce!
Toolchain Example: trivial Eigen-faces

Translation as a BEAT toolchain

- All relations are explicit: I/O, work flow
- We do not define here which algorithms, datasets, analyzers, formats or other results to run and produce!
Browsing

Interact: In your browser, try to click on the “Toolchains” tab:

- Use the small search bar under the tab to refine the search and look for “eigenface” (thanks to the “tutorial” user, it is available publicly);
- You can zoom in/out with a scroll button on your mouse or with an equivalent gesture on your laptop.
- You can move the canvas by middle-clicking and dragging (option-drag on a Mac, left-alt-drag on a PC)
- There are a few experiments that use this toolchain (click on any of them)
Most elements in a toolchain are *ordinary* blocks.
A look inside a block:

- **Inputs:** Each one accepts one data format
- **Configuration:**
  - Parameter #1
  - Parameter #2
  - ...
  - Parameter #N
- **Outputs:** Each one produces one data format

*Diagram of a block with input API, algorithm, output API, and storage with data.*
Blocks: Features

- Blocks can be equipped with **arbitrary** code
  - Potential to implement back-ends to support compiled code, any scripting language
  - We picked Python as our first implemented back-end
- Blocks can be executed on **arbitrary** architectures
- Blocks have inputs and outputs
- Data transmitted from block to block is formally defined \((Data\ Formats)\)
- Dataset blocks only have outputs, they define the possible data flows (loops)
- Analyzer blocks don’t output to any other block, they provide the results of your workflow
A block is composed of:

- an **unique name** (in the Toolchain)
- one or more **inputs**
- one or more **outputs**
- a synchronization channel (where the *for* loop is)

To perform an **experiment**, a **compatible algorithm** must be associated to each block.
Contribute: Like you did it for the experiment, let’s fork this toolchain and add a block for image pre-processing.

- Use the “Fork” button available on the right of the page.
- You’ll be re-directed to the editor page, eventually.
- Using your mouse, draw additional blocks for the “image” inputs, just after they have been emitted by the database.
- Save your work regularly.
A **dataformat** describes the structure of an atomic data unit: which fields it contains, and what their type is.

Interact: On your tabs, select “Data formats” and look around.

Example: A single floating-point value *(system/float/1)*

```
{
  "value": "float64"
}
```
Dataformats: Example

Example: Two integer fields (system/coordinates/1)

```json
{
    "x": "int32",
    "y": "int32"
}
```

In our Python backend, user code can assign values to an object of this type like this:

```python
obj.x = numpy.uint32(42)
obj.y = numpy.uint32(53)
```
Dataformats: Primitive types

The following types are available to declare new dataformats:

- Integers: `int8`, `int16`, `int32`, `int64`
- Unsigned integers: `uint8`, `uint16`, `uint32`, `uint64`
- Floating-point numbers: `float32`, `float64`
- Complex numbers: `complex64`, `complex128`
- `bool`
- `string`

Note: With a Python backend, those primitive types are implemented using their `numpy` counterparts.
A dataformat can declare whole objects as the type of its fields.

For instance, the coordinates of a rectangular region in an image could be represented by the following dataformat:

```json
{
    "coords": {
        "x": "int32",
        "y": "int32"
    },
    "size": {
        "width": "int32",
        "height": "int32"
    }
}
```

The fields can then be used in Python like this:

```python
data.coords.x = 10
data.coords.y = 20
data.size.width = 100
data.size.height = 100
```
Dataformats - Composition

It is possible to use an existing dataformat as the type of a field.

Example: `system/eye_positions/1`

```
{
    "right": "system/coordinates/1",
    "left": "system/coordinates/1"
}
```

The fields can then be used in Python as in the object example:

```python
data.left.x = 10
data.left.y = 50
data.right.x = data.left.x + 30
data.right.y = data.left.y
```
A field can also declare a multi-dimensional array of any other type.

For instance, consider the following example:

```json
{
   "field1": [10, "int32"],
   "field2": [10, 5, "bool"]
}
```

Here we declare that `field1` is a one-dimensional array of 10 integers (`int32`), and `field2` is a two-dimensional array of 10x5 booleans.

Note: In Python, those arrays are implemented using `numpy.ndarray`.
Dataformats - Array types (2)

It is also possible to declare an array without specifying the number of elements, by using a size of 0:

```json
{
    "field1": [0, "int32"],
    "field2": [0, 0, "bool"],
    "field3": [10, 0, "float32"]
}
```

Here `field1` is a one-dimensional array of integers (`int32`), `field2` is a two-dimensional array of booleans, and `field3` is a two-dimensional array of floating-point numbers (`float32`) where the first dimension is fixed to 10.
Dataformats - Array types (3)

Note that the following declaration isn’t valid (you can’t fix a dimension if the preceding one isn’t fixed too):

```json
{
    "error": [0, 10, "int32"]
}
```

When determining if that a data unit corresponds to a dataformat containing an array, the platform will check that:

- the number of dimensions is correct
- the size of each declared dimension that isn’t 0 is correct
- the type of each value in the array is correct
Dataformats: Extension (1)

When, by adding a few fields to an existing dataformat, another useful one can be created, it is possible to extend it.

For instance, from the `user/rectangular_area/1` dataformat:

```json
{
    "coords": {
        "x": "int32",
        "y": "int32"
    },
    "size": {
        "width": "int32",
        "height": "int32"
    }
}
```

we can create a new dataformat that describes a face:

```json
{
    "#extends": "user/rectangular_area/1",
    "eyes": "system/eye_positions/1"
}
```
A data unit of this dataformat can be used like that:

```plaintext
data.coords.x = 20
data.coords.y = 20
data.size.width = 100
data.size.height = 100
data.eyes.left.x = 40
data.eyes.left.y = 50
data.eyes.right.x = 100
data.eyes.right.y = 50
```
Dataformats: Your turn

Interact: Try to create yourself, a data format for a face bounding box

Tips:

1. A face bounding typically contains 4 integers (top-left coordinates, width and height)
2. Less is more: use other data formats by opening a second tab on your browser and checking what is available
3. Click on the formats to visualize them, read their documentation (if one was written)
4. Try to document your data format well, so you remember what it was for
Filling Blocks with Algorithms

• So that the block we added to our forked toolchain is useful, we must implement at least one algorithm that we can use it with

• Algorithms can be implemented to any number of complex tasks

• For this tutorial, we’ll use the *Python* programming language to examplify and create an algorithm for our image pre-processing block.

• We can classify algorithms by their inputs/outputs and their applicability to different types of blocks

• Let’s review some of the most common possibilities
Case study #1: “1-to-1”

The job of an *algorithm which fits this block* is to compute some features for each image received on its input.

For each **image** received on its input, the algorithm should generate one corresponding **feature** on its output.

**Examples**

Feature extractors, image pre-processors
N.B.: no loops!

```python
class Algorithm:

    def __init__(self):
        self.extractor = bob.ip.DCTFeatures(12, 12, 11, 11, 45)

    def process(self, inputs, outputs):

        # Convert the image to grayscale
        grayscale_image = bob.ip.rgb_to_gray(inputs['image'].data.pixels)

        # Convert the grayscale image to a floating-point representation
        float_image = grayscale_image.astype('float64') / 255.0

        # Compute the features
        features = numpy.vstack(self.extractor(float_image))

        # Generate a data unit on the output
        outputs['features'].write({'value': features})

        return True
```
Configurable algorithm

This algorithm implementation uses some magic numbers:

```python
class Algorithm:
    def __init__(self):
        self.extractor = bob.ip.DCTFeatures(12, 12, 11, 11, 45)
```

It is better to allow the creator of the experiment to choose those values.
A configurable algorithm must have the following structure:

```python
class Algorithm:

    def setup(self, parameters):
        # Here, you process user parameters and store
        return True

    def process(self, inputs, outputs):
        # Here, you use parameters to perform the task
        return True
```
class Algorithm:

    def setup(self, parameters):
        block_size = parameters.get('block-size', 12)
        block_overlap = parameters.get('block-overlap', 11)
        number_of_components = parameters.get('number-of-components', 45)

        self.extractor = bob.ip.DCTFeatures(
            block_size,
            block_size,
            block_size,
            block_overlap,
            block_overlap,
            number_of_components,
        )

        return True

    def process(self, self, inputs, outputs):
        # Same as before
Data unit generation

In the Python backend, write output data like this:

```python
outputs["coordinates"] .write({
    'y': numpy.int32(20),
    'x': numpy.int32(10)
})
```

The data written must match the structure defined for the type, in terms of keys names for each field **and** the value types. Unsafe casting is **not** allowed. In doubt, wrap the values with the “numpy” homologues of their supposed types.

```json
{
    "y": "int32",
    "x": "int32"
}
```

All operations must be **explicit**.
Case study #2: fancier “1-to-1”

The inputs of this block are **synchronized**: the platform will ensure that, at each execution of the algorithm, the data unit on the *image* input is related to the one on the *label* input.

For each synchronized pair of **data units** received on its inputs, the algorithm inside this block generates one corresponding **data unit** on its output.

**Nota Bene**
This behaviour is implemented by the database/protocol author. The platform only enforces it.

**Examples**
More complex feature extractors and image pre-processors which depend on input parameters (e.g. image quality).
Case study #2: Algorithm (2)

N.B.: no loops!

```python
import numpy

class Algorithm:
    def process(self, inputs, outputs):
        pixels = inputs['image'].data.pixels
        label = inputs['label'].data.text

        ...

        features = numpy.ndarray((10, 10), dtype=numpy.float64)
        features[0][0] = 4.0

        ...

        outputs['features'].write({
            'value': features
        })

    return True
```
Case study #3: Synchronized “N-to-1”

The inputs of this block are synchronized: the platform will ensure that, at each execution of the algorithm, the data unit on the *image* input is related to the one on the *label* input.

The algorithm inside this block generates one data unit on its output for each label received on its input. The algorithm does its computation on all the images corresponding to the current label before generating a data unit.

**Examples**

Object-class model training.
Case study #3: Synchronized “N-to-1”

N.B.: No loops, branch on isDataUnitDone()

class Algorithm:

    def __init__(self):
        self.images = []

    def process(self, inputs, outputs):

        self.images.append(inputs['image'].data.pixels)  # accumulates

        if inputs['label'].isDataUnitDone():
            label = inputs['label'].data.text
            ...
            outputs['features'].write({'value': features})
            self.images = []  # reset

    return True
Case study #4: **Unsynchronized inputs**

The inputs of the many blocks are connected to two different datasets: **they aren’t all synchronized together!**
Each dataset block is the origin of a synchronization channel, sharing its name.

All the outputs of a dataset block are synchronized together: the data units on one output are related to the ones on the other outputs.

Each connection is assigned to a synchronization channel.
Each *data unit* is assigned an *index* (or a *range of indexes*).

The platform uses this information to determine the relationships between *data units* on different *connections* that have the same synchronization channel.
The inputs of a block are **grouped** by their synchronization channel.

**Groups** aren’t synchronized together (not the same synchronization channel), but the inputs inside each group are still able to be synchronized.

Each block must select one group of its inputs as its main one.
Each time new data units are available on the main group of inputs of the block, the platform will invoke the process(...) method of its algorithm.

The algorithm is expected to take care of the other groups of inputs itself.
class Algorithm:

    def __init__(self):
        self.models = None
        self.score = 0.0

    def process(self, inputs, outputs):

        # At first execution, read all the models
        if self.models is None:
            self.models = {}
            group = inputs.groupOf('model')

            while group.hasMoreData():
                group.next()

                model_label = group['model_label'].data.text
                something = do_something_with(group['model'].data)

                self.models[model_label] = something

        ...

N.B.: loop only on unsynchronized inputs
def process(self, inputs, outputs):
...

# Process the current test image/label pair
image = inputs['image'].data.pixels
label = inputs['label'].data.text

self.score += ...

# After the very last test image, writes the score on the output
if not inputs['image'].hasMoreData():
    outputs['score'].write({
        'value': self.score
    })

return True
**Databases** provide special algorithm blocks that implement dataset blocks in Toolchains

Interact: Now click on the tab “Databases” and then select the “atnt” database, scroll down.
Each Database defines a number of protocols (how to use its data in a toolchain)

- Databases have a **name** and a **version**
- Each Protocol defines a number of Sets
- Each Protocol conforms to a Template
- Each Set defines a number of outputs, with a **precise type**
- So that a Set can be used in a Toolchain (inside a Dataset), there must be a **match** between the number of outputs in the said Set as well as it must be **compatible** the input of connected blocks
The web GUI is just a front end to a powerful computing grid

Backend

You (via internet)

Frontend

Web Server + RESTful API

Backend

Object Repository

Scheduler + Workers Nodes

Cache

Data

Execution of Experiments

commands <-> data
Queues

Our backend is organised into processing queues.

Queues:

- Have a name
- Have a number of *slots* associated to it
- Can be shared with people or groups
A *slot* is an abstract concept putting together:

- A number of cores in a host, with a base OS installed
- Memory (RAM)
- A base operating system
- Processing environments
Processing Environments

A processing *Environment* is a sandbox for your algorithms:

- A set of installed software that can be used on user algorithms and libraries for a given host/OS
- An API to the BEAT dataformat I/O structure
- A binary API to our Workers
Processing Strategy

Simple: 1 block = 1 slot (∼ one process)

- Each process is independent of each other
- The sandboxes only read and write data from a (rather big) disk cache
- This allows for an heterogenous processing farm
- This allows for **automatic** map-reduce (parallelization)
To sum-up

Because of this design:

- It is possible to control computing power and privileges of our users.
- You can use an heterogeneous farm of resources.
- You can automatically reduce processing time by applying parallelization.
- A central scheduler is responsible for assigning priorities and balancing the computing farm.
Initial configuration: 80 cores (12 Gb RAM/core), 10 Tb Cache, All nodes are virtualized
Create a parallelizable algorithm

We’ll create together, an algorithm that is parallelizable and you’ll use it on your new toolchain for experiments.

Interact: Click on the “Algorithms” tab, hit “New” (green button)
Example: Histogram Equalization

This is an histogram equalization algorithm that fulfills these criteria:

```
import numpy

class Algorithm:
    def __init__(self):
        self.nb_bins = 256
        self.normed = True

    def process(self, inputs, outputs):
        raw_image = inputs["raw_image"].data.value
        imhist, bins = numpy.histogram(raw_image.flatten(), self.nb_bins, normed=self.normed)
        cdf = imhist.cumsum()
        cdf = 255 * cdf / cdf[-1]
        output_image = numpy.interp(raw_image.flatten(), bins[:-1], cdf).reshape(raw_image.shape).astype(numpy.uint8)
        outputs["image"].write({
            'value': output_image
        })
        return True
```

Inputs / Outputs

Inputs: raw_image system/array_2d_uint8/1  Outputs: image system/array_2d_uint8/1
Interact: Go back to the experiment tab, click new and now select your toolchain.

Let’s, together, examine it details:

- Toolchain display
- Global parameter section
- Database configurator
- Default processing queue
- Local parameters and processing queue
- Parallelization
- Automated filtering of algorithms

Hit the “Go!” button.
Interact: Run a few experiments.

Let’s go back to the search:

- Use the top bar to search for results using your new toolchain
- Compare these results
- Save this search for future reference
A note on “modify-ability”

A key aspect in BEAT is “reproducibility”:

- If you use a component in any manner, you can no longer modify it
- Implications:
  - If you use an algorithm/dataformat/library in an experiment, you cannot delete/modify it any longer
  - If you fork an algorithm, you cannot modify the forked version
  - If you certify an experiment, everything it depends on cannot be modified any more
Locking hierarchy
- BEAT keeps track of data transmitted through all stages of the processing
- It caches the data in a large disk array (~10 Tb)
- The cache is invalidated automatically when things change:
  - Operating System or installed packages are updated
  - Toolchain changes
  - Database version changes
- One cached item is valid for a specific combination of all of the above
- If a cached item is available, it is used to speed-up processing.
An attestation mechanism is available in the platform.

- Allow 3rd. party verification of results obtained with a given configuration
- Allow for a scientific *review* process to take place in confidentiality
- N.B. An attestation *permanently* locks an experiment
Attestation: Message Flow

1. **Publication reader**
2. **Reviewer**
3. **BEAT User**
4. **BEAT Website**

**Setup of an experiment**

**Results, graphs, ...**

**Attestation request**

**Attestation number & publication template**

**Editing of the publication**

**Submission**

**Decision (Accepted/Rejected)**

**Attested results request**

**Attested results**

**Decision notification**

**Read the publication**

**Experiment request**

**All the details about the experiment**

**Change some parameters**

**Results, graphs, ...**

**Perform the experiment**

**Only if the publication is accepted**

**Allow anyone to access the experiment**

**Perform the new experiment**
You can also implement libraries, to apply DRY ("Don’t repeat yourself"):

- Libraries are like algorithms, except no input/output
- You can combine libraries with other libraries in order to build complex systems
- You can combine libraries with algorithms
- Follow the same modify-ability as for other objects
Planned Activities, Issues (1)

Priority scheduling:

- Currently, our scheduler just implements FIFO scheduling
- Improve to introduce priorities:
  - Wait time
  - Number of jobs
  - Forced priority
Planned Activities, Issues (2)

Reputation

- Gamify the platform
- The more users use from your stuff, more priority on processing
- Give opportunities for people with low computing power
- More to people that return more to the community
Plotting

- The plotting backend is quite advanced already
- Allow users to parameterize plotting
- Make this available through a simple interface on your micro-site
Planned Activities, Issues (4)

Search, report feature

- Introduce more extensive search criteria. E.g.:
  - Show me all results with CPU time $< X\text{minutes}$ for this particular block
  - Tabulate EER against a certain parameter value
- Export tables and figures (and its data) into re-usable text, images and tables for your publications ($\LaTeX$)
- Save reports for easy reference
Planned Activities, Issues (5)

Documentation & Public Release

- Our API documentation is already pretty solid
- Work on our main user guides
- Work on administration documentation
- **Public release** planned for begin of 2016
Remote Software Development Kit (SDK)

- Using our API, it is possible to interact remotely with the platform
- You download/upload objects, start experiments
- We plan to integrate all this into a virtual image with the required software so that you can create run your experiments locally, on a debugging environment
- N.B.: The raw data will not be available on the SDK (privacy requirement). You must procure it yourself.
Keeping in touch

We created a Google Groups forum for discussion. Join to:

- Keep up-to-date with the platform development and the Idiap prototype
- Request features and database inclusions
- Discuss issues and other problems you may be facing

Details:

- View posts: https://groups.google.com/d/forum/beat-devel
- Subscribe: beat-devel+subscribe@googlegroups.com
- Post: beat-devel@googlegroups.com