OpenBR – Open Source Biometric Recognition

Josh Klontz & Brendan Klare & Mark Burge

www.openbiometrics.org

January 9, 2013

Why Open Source?

Why Open Source?

Reproducible Research

Support a common set of file formats and tools for algorithm design, development, and evaluation.

Why Open Source?

Reproducible Research

Support a common set of file formats and tools for algorithm design, development, and evaluation.

Reduced Time to Market

Provide a well-engineered framework for deploying algorithms.

Why Open Source?

Reproducible Research
Support a common set of file formats and tools for algorithm design, development, and evaluation.

Reduced Time to Market
Provide a well-engineered framework for deploying algorithms.

Reduce Duplication
Provide state-of-the-art baseline algorithms for various modalities.

Why Open Source?

Reproducible Research
Support a common set of file formats and tools for algorithm design, development, and evaluation.

Reduced Time to Market
Provide a well-engineered framework for deploying algorithms.

Reduce Duplication
Provide state-of-the-art baseline algorithms for various modalities.

Improve Collaboration
Help foster a community where collaboration can take place at the source code level.

What's in it?

Off-the-shelf algorithms

- Face Recognition
- Gender Classification
- Age Estimation
- Commercial Wrappers

What's in it?

Off-the-shelf algorithms

- Face Recognition
- Gender Classification
- Age Estimation
- Commercial Wrappers

Tools for algorithm evaluation

- Standardized set of file formats
- Automatic plot generation
- Command line interface supporting common biometrics tasks

What's in it?

Off-the-shelf algorithms

- Face Recognition
- Gender Classification
- Age Estimation
- Commercial Wrappers

Tools for algorithm evaluation

- Standardized set of file formats
- Automatic plot generation
- Command line interface supporting common biometrics tasks

Software framework for algorithm development

- C++ plugin API for implementing new algorithms
- Grammar for image processing
- Automatic testing, packaging and deployment

Software Architecture

Qt

Cross-platform application and UI framework

OpenCV

Image processing library

Eigen

Linear algebra library

CMake

Cross-platform build system

Software Architecture

Qt
Cross-platform application and UI framework

OpenCV
Image processing library

Eigen
Linear algebra library

CMake
Cross-platform build system

br
Command line application for running algorithms and evaluating results.

C API
High-level interface for other programming languages.

C++ Plugin API
Core interface for using and developing algorithms.

Supported Platforms

Supported Platforms
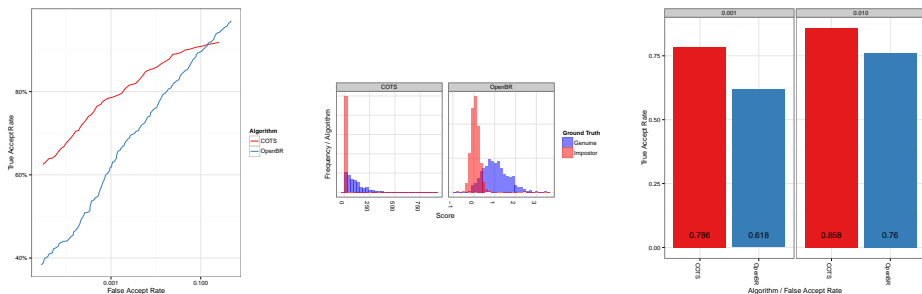
Supported Platforms

Algorithm Evaluation

Figure : OpenBR vs COTS face recognition on *MEDS* mugshot database.

Algorithm Evaluation
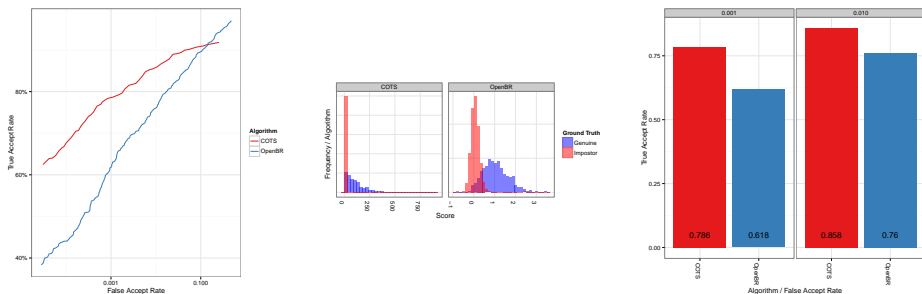

Figure : OpenBR vs COTS face recognition on *MEDS* mugshot database.

	OpenBR	COTS-A	COTS-B	COTS-C	COTS-D
TAR @ FAR = 0.01	0.77	0.93	<u>0.96</u>	0.86	0.80
Template Size (kB)	<u>0.75</u>	2.8	5.0	36	74
Enrollment Speed	<u>10</u>	N/A	N/A	1.3	1.2
Comparison Speed	3,800,000	N/A	110,000	19,000	2,000

Algorithm Example: Face Recognition

```
$ br -algorithm FaceRecognition -compare me.jpg you.jpg
```

Algorithm Example: Face Recognition

```
$ br -algorithm FaceRecognition -compare me.jpg you.jpg
```

FaceRecognition

FaceDetection!<FaceRegistration>!<FaceExtraction>+
<FaceEmbedding>+<FaceQuantization>:UCharL1

Algorithm Example: Face Recognition

```
$ br -algorithm FaceRecognition -compare me.jpg you.jpg
```

FaceRecognition

FaceDetection!<FaceRegistration>!<FaceExtraction>+
<FaceEmbedding>+<FaceQuantization>:UCharL1

FaceDetection

Open+Cvt(Gray)+Cascade(FrontalFace)

Algorithm Example: Face Recognition

```
$ br -algorithm FaceRecognition -compare me.jpg you.jpg
```

FaceRecognition

FaceDetection!<FaceRegistration>!<FaceExtraction>+
<FaceEmbedding>+<FaceQuantization>:UCharL1

FaceDetection

Open+Cvt(Gray)+Cascade(FrontalFace)

FaceRegistration

ASEFEyes+Affine(88,88,0.25,0.35)+FTE(DFFS)

Algorithm Example: Face Recognition

```
$ br -algorithm FaceRecognition -compare me.jpg you.jpg
```

FaceRecognition

FaceDetection!<FaceRegistration>!<FaceExtraction>+
<FaceEmbedding>+<FaceQuantization>:UCharL1

FaceDetection

Open+Cvt(Gray)+Cascade(FrontalFace)

FaceRegistration

ASEFEyes+Affine(88,88,0.25,0.35)+FTE(DFFS)

...

FaceEmbedding

Dup(12)+RndSubspace(0.05,1)+LDA(0.98)+Cat+PCA(768)

Live Coding

```
fill(161, 219, 114);
for (var x = 40; x < 150; x += 50) {
    rect(x, 33, 20, 10);
    rect(x, 45, 20, 15);
    rect(x, 62, 20, 25);            Draw a ▮ shape.
}
```

Live Coding

Live Coding

Inventing on Principle

http://www.youtube.com/watch?v=PUv66718DII
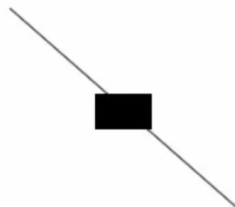
Welcome to the Parallel Jungle!

Figure : http://www.drdobbs.com/parallel/welcome-to-the-parallel-jungle/232400273

Figure : i7 3930k

Figure : GTX 680

The Economics

Figure : i7 3930k

$570.00

Figure : GTX 680

$568.50
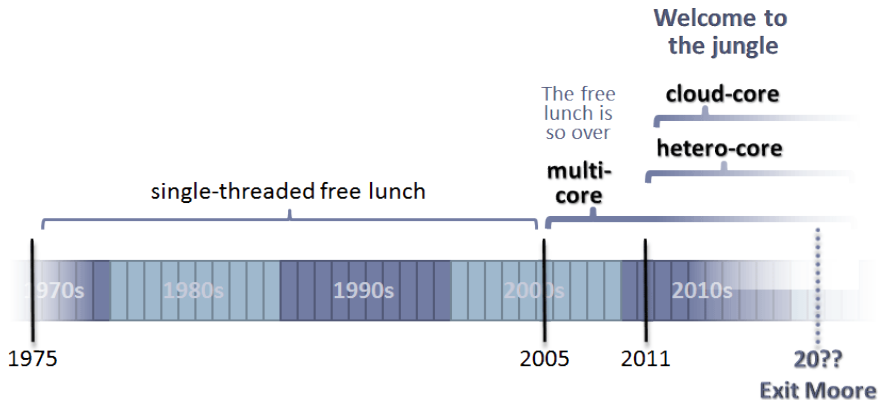
The Economics

Figure : i7 3930k

$570.00
76.8 GFLOPS

Figure : GTX 680

$568.50
1665 GFLOPS

The Economics

Figure : i7 3930k

$570.00
76.8 GFLOPS

Figure : GTX 680

$568.50
1665 GFLOPS

Gotcha: Memory Bandwidth

12.8 GFLOPS   **48.0 GFLOPS**

The Wish List

What we want

- Write once and run everywhere
- Automatically utilize all available hardware
- Run faster on future hardware

The Wish List

What we want
- Write once and run everywhere
- Automatically utilize all available hardware
- Run faster on future hardware

What we need
- Virtual machine or just-in-time compiler
- Express computations using induction variables (a.k.a. "kernels")

The Wish List

What we want

- Write once and run everywhere
- Automatically utilize all available hardware
- Run faster on future hardware

What we need

- Virtual machine or just-in-time compiler
- Express computations using induction variables (a.k.a. "kernels")

What we're proposing

- LLVM IR and JIT compiler
- Designing for OpenCL 2.0 standard
- C++ API for kernel construction

The Dream

Perfectly Composable Image Processing Primitives

A grammar for building algorithms from orthogonal primitive kernels with typeless semantics and optimized execution.

The Dream

Perfectly Composable Image Processing Primitives

A grammar for building algorithms from orthogonal primitive kernels with typeless semantics and optimized execution.

When we say...

```
Transform *lbpu2 = Transform::make("LBP(1)+U2");
```

The Dream

Perfectly Composable Image Processing Primitives

A grammar for building algorithms from orthogonal primitive kernels with typeless semantics and optimized execution.

When we say...

```
Transform *lbpu2 = Transform::make("LBP(1)+U2");
```

...we mean

Give me a pointer to a function that computes $LBP_{8,1}^{u2}$ on an image, minimizes main memory transactions by combining kernels, and is optimized for parallel execution on the hardware available.

The Dream

Perfectly Composable Image Processing Primitives

A grammar for building algorithms from orthogonal primitive kernels with typeless semantics and optimized execution.

When we say...

```
Transform *lbpu2 = Transform::make("LBP(1)+U2");
```

...we mean

Give me a pointer to a function that computes $LBP^{u2}_{8,1}$ on an image, minimizes main memory transactions by combining kernels, and is optimized for parallel execution on the hardware available.

Take-Home Message

Compilation = Source Code + Available Hardware + First Image

The End

Website

www.openbiometrics.org

Source

https://github.com/biometrics/openbr

E-mail

openbr-dev@googlegroups.com

Slides

www.openbiometerics.org/slides.pdf

Thank You!

Plugin Example: Local Binary Patterns

```
#include <openbr_plugin.h>
```

Plugin Example: Local Binary Patterns

```cpp
#include <openbr_plugin.h>
class LBP : public Transform {
```

Plugin Example: Local Binary Patterns

```
#include <openbr_plugin.h>
class LBP : public Transform {
  BR_PROPERTY(int, radius, 1)
```

Plugin Example: Local Binary Patterns

```cpp
#include <openbr_plugin.h>
class LBP : public Transform {
  BR_PROPERTY(int, radius, 1)
  void project(const Matrix &src, Matrix &dst) const {
```

Plugin Example: Local Binary Patterns

```
#include <openbr_plugin.h>
class LBP : public Transform {
 BR_PROPERTY(int, radius, 1)
 void project(const Matrix &src, Matrix &dst) const {
   for (int r=radius; r<src.rows-radius; r++)
     for (int c=radius; c<src.cols-radius; c++) {
       float cval = p[r*src.cols+c];
       dst(r, c) =
         (p[(r-radius)*src.cols+c-radius] >= cval ? 128 : 0) |
         (p[(r-radius)*src.cols+c] >= cval ? 64 : 0) |
         ...;
     }
```

Plugin Example: Local Binary Patterns

```cpp
#include <openbr_plugin.h>
class LBP : public Transform {
 BR_PROPERTY(int, radius, 1)
 void project(const Matrix &src, Matrix &dst) const {
   for (int r=radius; r<src.rows-radius; r++)
     for (int c=radius; c<src.cols-radius; c++) {
       float cval = p[r*src.cols+c];
       dst(r, c) =
         (p[(r-radius)*src.cols+c-radius] >= cval ? 128 : 0) |
         (p[(r-radius)*src.cols+c] >= cval ? 64 : 0) |
         ...;
     }
 }
};
```

Plugin Example: Local Binary Patterns

```
#include <openbr_plugin.h>
class LBP : public Transform {
  BR_PROPERTY(int, radius, 1)
  void project(const Matrix &src, Matrix &dst) const {
    for (int r=radius; r<src.rows-radius; r++)
      for (int c=radius; c<src.cols-radius; c++) {
        float cval = p[r*src.cols+c];
        dst(r, c) =
          (p[(r-radius)*src.cols+c-radius] >= cval ? 128 : 0) |
          (p[(r-radius)*src.cols+c] >= cval ? 64 : 0) |
          ...;
      }
  }
};
BR_REGISTER(Transform, LBP)
```

Plugin Example: Local Binary Patterns

```
#include <openbr_plugin.h>
class LBP : public Transform {
 BR_PROPERTY(int, radius, 1)
 void project(const Matrix &src, Matrix &dst) const {
   for (int r=radius; r<src.rows-radius; r++)
     for (int c=radius; c<src.cols-radius; c++) {
       float cval = p[r*src.cols+c];
       dst(r, c) =
         (p[(r-radius)*src.cols+c-radius] >= cval ? 128 : 0) |
         (p[(r-radius)*src.cols+c] >= cval ? 64 : 0) |
         ...;
     }
 }
};
BR_REGISTER(Transform, LBP)
...
Transform *lbp = Transform::make("LBP(1)");
```

Plugin Example: Local Binary Patterns

```
#include <openbr_plugin.h>
class LBP : public Transform {
  BR_PROPERTY(int, radius, 1)
  void project(const Matrix &src, Matrix &dst) const {
    for (int r=radius; r<src.rows-radius; r++)
      for (int c=radius; c<src.cols-radius; c++) {
        float cval = p[r*src.cols+c];
        dst(r, c) =
          (p[(r-radius)*src.cols+c-radius] >= cval ? 128 : 0) |
          (p[(r-radius)*src.cols+c] >= cval ? 64 : 0) |
          ...;
      }
  }
};
BR_REGISTER(Transform, LBP)
...
Transform *lbp = Transform::make("LBP(1)");
Transform *lbpu2 = Transform::make("LBP(1)+U2");
```