

Facial Detection

Jianxing Lu
Wenjin Cao



Use Cases

- ◆ To Detect Faces And Offer Webcam Effects (e.g. to add a pair of glasses on user's face)
- ◆ To Analyze Facial Expressions (e.g. to detect smiles potentially)

Tools : OpenCV | JavaCV

- ◆ OpenCV (Open Source Computer Vision) is a library of programming functions mainly aimed at real-time computer vision.
The library has more than 2500 optimized algorithms which can be used to detect and recognize faces, identify objects, etc.
- ◆ JavaCV is a wrapper of OpenCV with easy-to-use methods. (Java interface to OpenCV and more.)

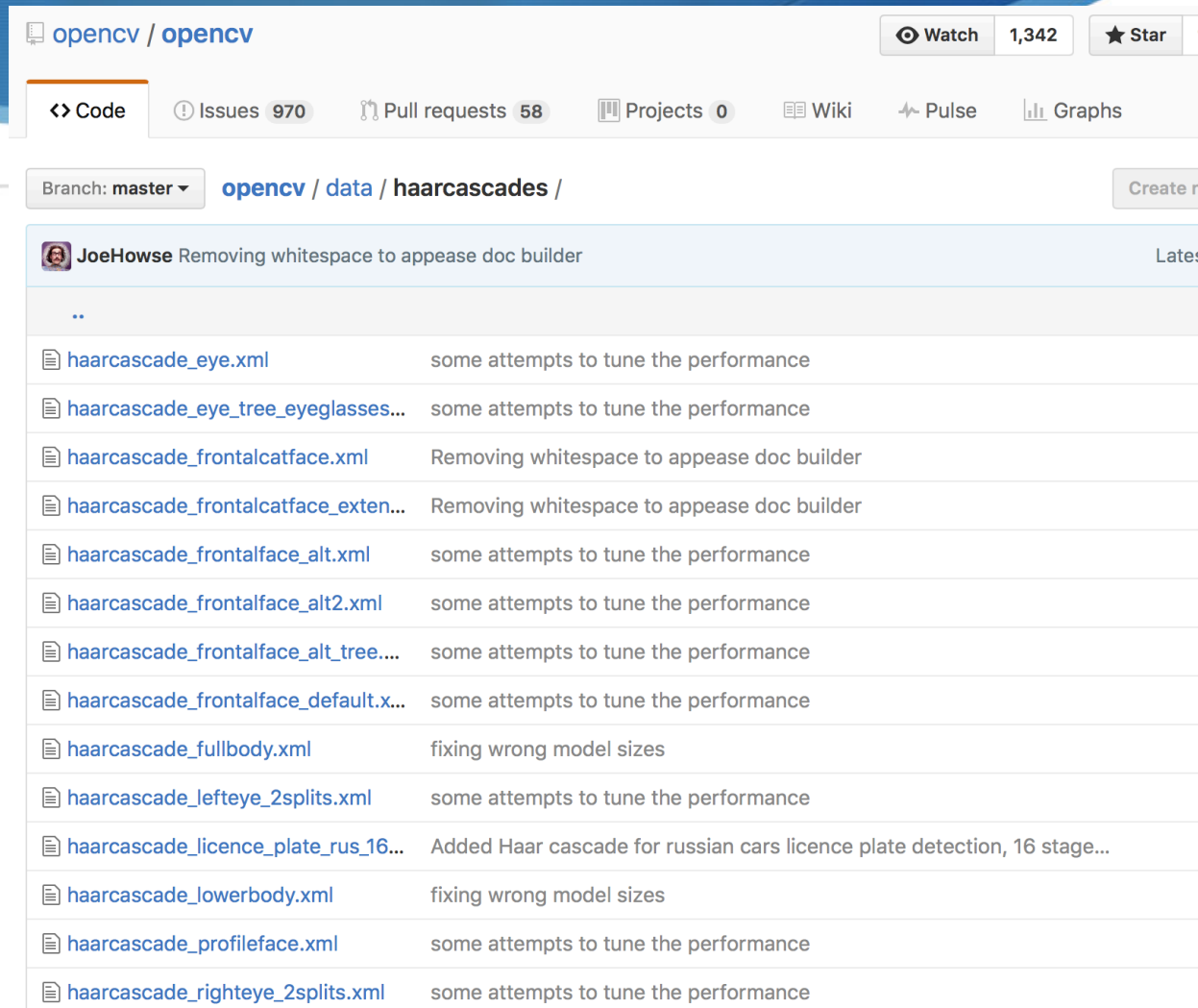
Tools : HAAR Classifiers

HAAR FEATURE-BASED CASCADE CLASSIFIERS

- Supported directly by OpenCV to **detect objects in images**
- A machine learning based approach with **pre-trained** classifiers
- “CASCADE” : Instead of applying all the features at once, group the features into different stages of classifiers and apply one-by-one. Only if the previous stage is passed, the next stage is applied and the process is continued.
(An area/object is considered to test positive (be identified) if all features in all stages of the definition return positive.)

Tools : HAAR Classifiers

- OpenCV has many pre-trained HAAR classifiers for frontal/profile face, eyes, etc.
- We got those XML files from Github:
- It's also possible to self-train a classifier, though we used the pre-trained ones for time's sake.



The screenshot displays the GitHub repository for OpenCV, specifically the `data/haarcascades` directory. The repository is maintained by `opencv` and has 1,342 watchers and 970 issues. The current branch is `master`. A commit by `JoeHowse` is shown, titled "Removing whitespace to appease doc builder". Below the commit, a list of XML files is displayed, each with a brief description of its purpose.

File Name	Description
<code>haarcascade_eye.xml</code>	some attempts to tune the performance
<code>haarcascade_eye_tree_eyeglasses...</code>	some attempts to tune the performance
<code>haarcascade_frontalcatface.xml</code>	Removing whitespace to appease doc builder
<code>haarcascade_frontalcatface_exten...</code>	Removing whitespace to appease doc builder
<code>haarcascade_frontalface_alt.xml</code>	some attempts to tune the performance
<code>haarcascade_frontalface_alt2.xml</code>	some attempts to tune the performance
<code>haarcascade_frontalface_alt_tree....</code>	some attempts to tune the performance
<code>haarcascade_frontalface_default.x...</code>	some attempts to tune the performance
<code>haarcascade_fullbody.xml</code>	fixing wrong model sizes
<code>haarcascade_lefteye_2splits.xml</code>	some attempts to tune the performance
<code>haarcascade_licence_plate_rus_16...</code>	Added Haar cascade for russian cars licence plate detection, 16 stage...
<code>haarcascade_lowerbody.xml</code>	fixing wrong model sizes
<code>haarcascade_profileface.xml</code>	some attempts to tune the performance
<code>haarcascade_righteye_2splits.xml</code>	some attempts to tune the performance

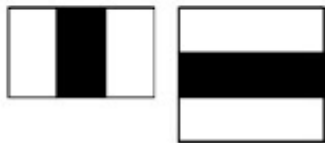
Tools : HAAR Classifiers

“CASCADE”

A ‘cascade’ is a series of ‘Haar-like features’ (digital image features used in object recognition) to form a classifier.



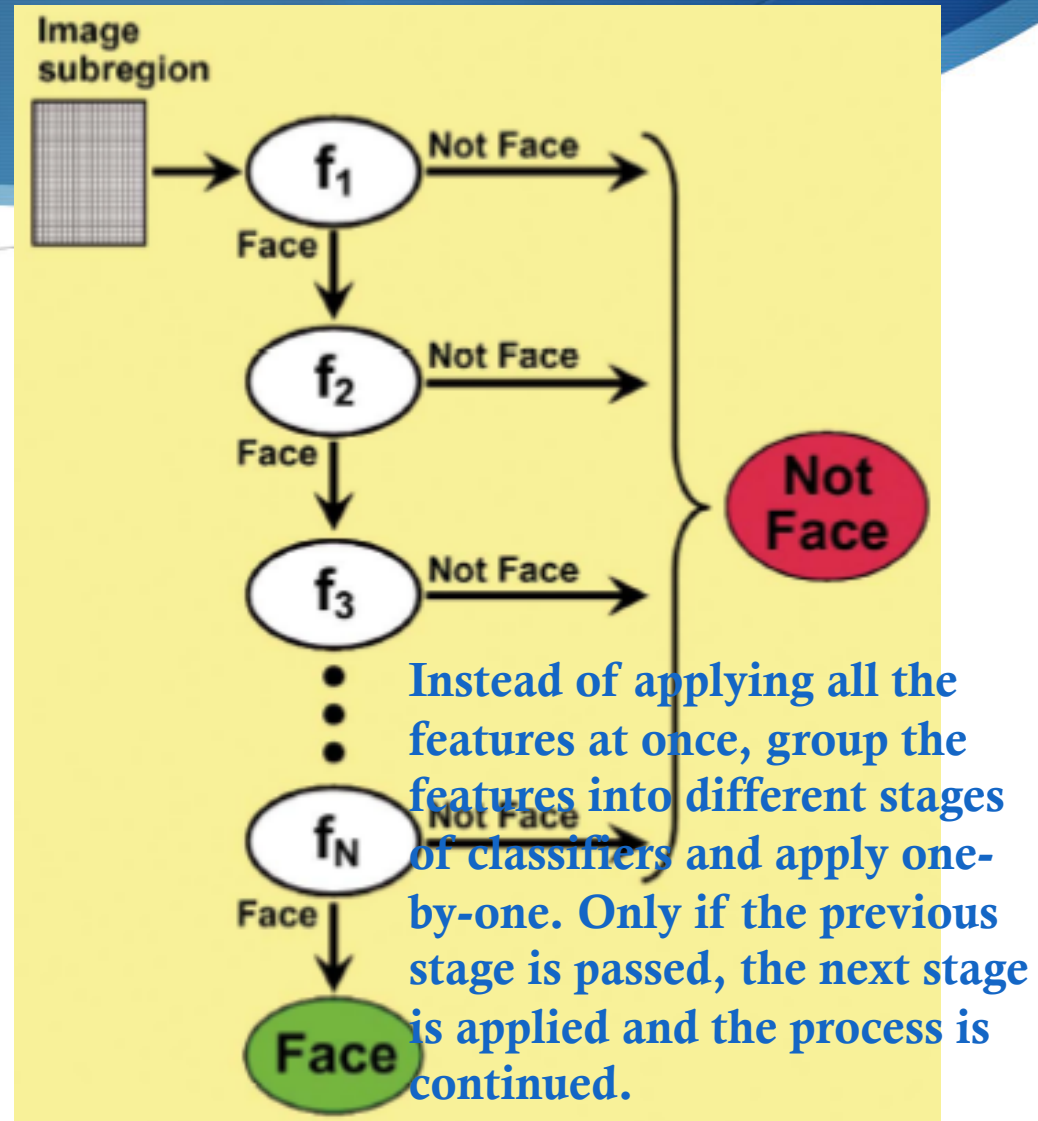
(a) Edge Features



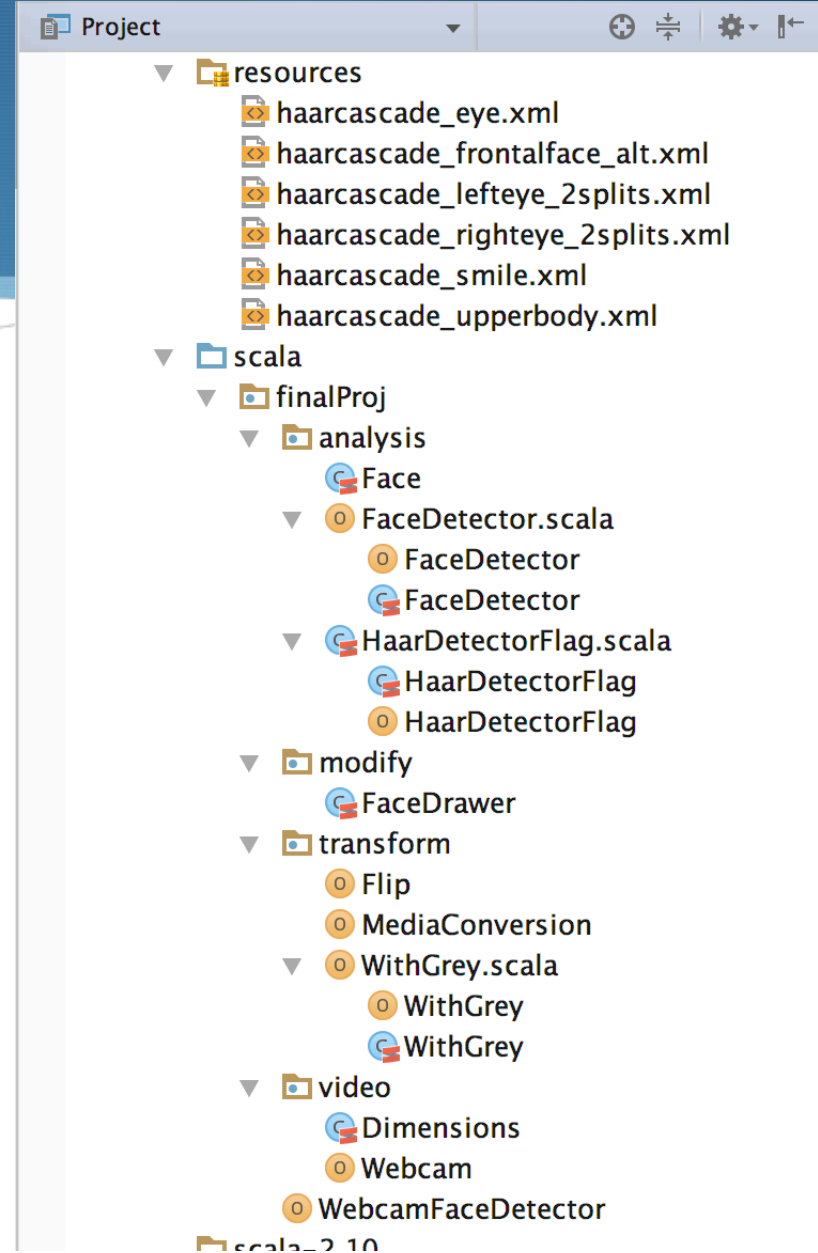
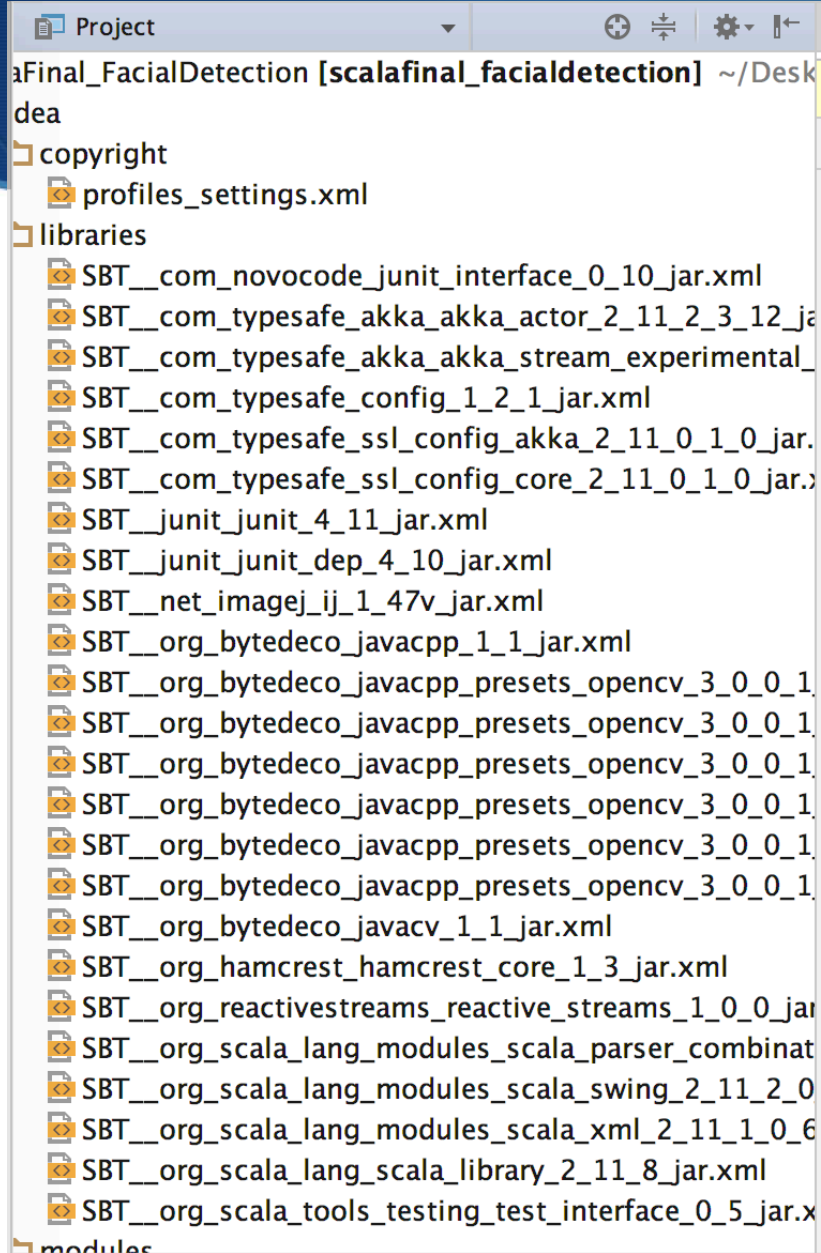
(b) Line Features



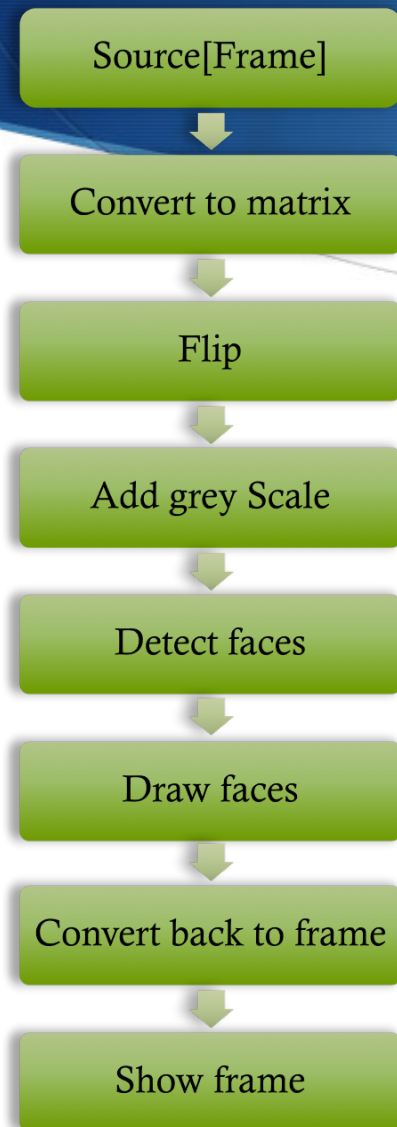
(c) Four-rectangle features



SBT-based Scala Project



Flow Chart



♦ **Conversion to grey scale :**

Many image analysis tools need to be run on greyscale images, both for simplicity and efficiency.

♦ **To detect faces :**

Instantiating a CascadeClassifier

Instantiating an instance of RectVector (a wrapper for a native vector of rectangles to denote where our objects are in the image matrix).

Pass the RectVector to the CascadeClassifier along with a greyscale image and some other options

♦ **A Swing app :**

1st frame: to choose between loading a custom Haar cascade classifier file or to load the default one that's packaged in resources folder

2nd frame: to shows our feed along with rectangles around detected objects in the CanvasFrame

Implementation of Flow Chart

```
val flow = webcamSource
    .map(MediaConversion.toMat) // most OpenCV manipulations require a Matrix
    .map(Flip.horizontal)
    .map(WithGrey.build)
    .map(faceDetector.detect)
    .map((faceDrawer.drawFaces _).tupled)
    .map(MediaConversion.toFrame) // convert back to a frame
    .map(canvas.showImage)
    .to(Sink.ignore)

flow.run()
```

AKKA for Webcam Frame

-- Reactive Stream

- akka.stream.scaladsl.**Source<Out,Mat>** -- Source[Frame, Unit]. A set of stream processing steps that has one open output(JavaCV Frame).
- akka.actor.Props -- To configure the creation of the actor
- akka.stream.actor.{ActorPublisher, ActorMaterializer}
 - To make the actor a stream publisher that keeps track of the subscription life cycle and requested elements.
 - Materialization turns a Source into a Reactive Streams Publisher
- akka.stream.scaladsl.Sink -- A Sink is a set of stream processing steps that has one open input and an attached output.

Implementation of AKKA

- ◆ Mainly used in Webcam class by generated a ActorSystem
- ◆ Utilized FrameGrabber to avoid bad things happens like video not be synchronized
- ◆ Designed a lazy WebcamFramePublisher to work with grabber

```

def source(
  deviceId: Int,
  dimensions: Dimensions,
  bitsPerPixel: Int = CV_8U,
  imageMode: ImageMode = ImageMode.COLOR
)(implicit system: ActorSystem): Source[Frame, Unit] = {
  //Create actor
  val props = Props(
    new WebcamFramePublisher(
      deviceId = deviceId,
      imageWidth = dimensions.width,
      imageHeight = dimensions.height,
      bitsPerPixel = bitsPerPixel,
      imageMode = imageMode
    )
  )

  val webcamActorRef = system.actorOf(props)
  //Keep Tracking
  val webcamActorPublisher = ActorPublisher[Frame](webcamActorRef)
  //Open Output
  Source.fromPublisher(webcamActorPublisher)
}

```

Demo Time

- 💧 Function 1: Face only detection
- 💧 Function 2: Eyes detection
- 💧 Function 3(main function): Facial detection of number of faces with left and right eye