Develop a Face Recognition System Using OpenCV

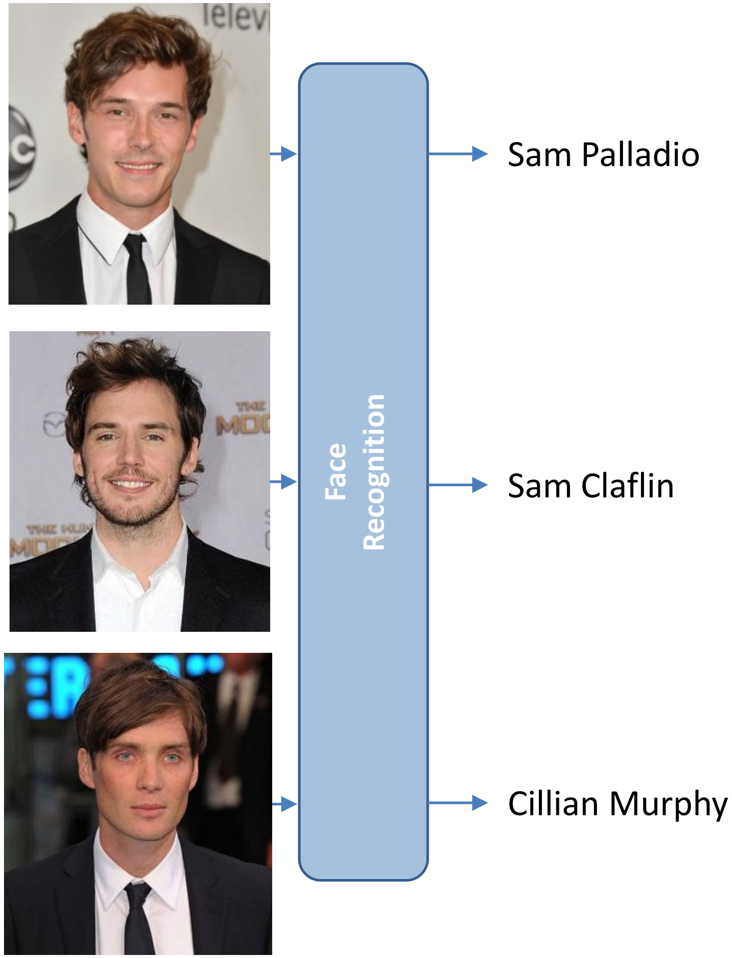
WU Jia

OpenCV China Team

Outline

- **Face recognition in brief**
- **Build a face recognition system**
  - Related APIs in OpenCV
  - Build the system step by step using OpenCV
  - Demo
- **Exercise**

Face recognition is to identify or verify a person from a digital image.



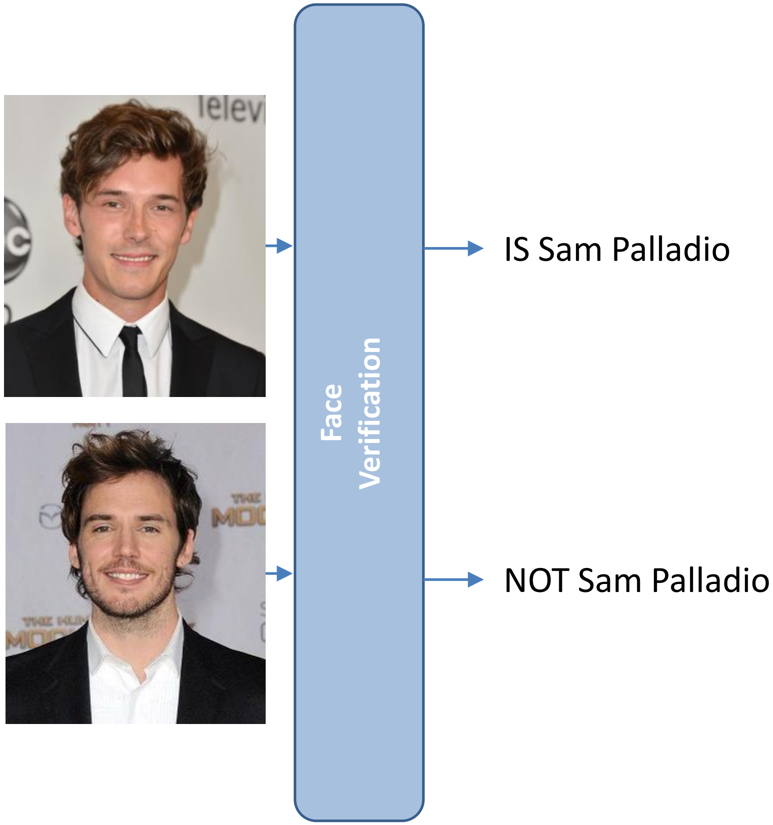
Face Recognition

Sam Palladio

Sam Claflin

Cillian Murphy

Who is this person, 1:N



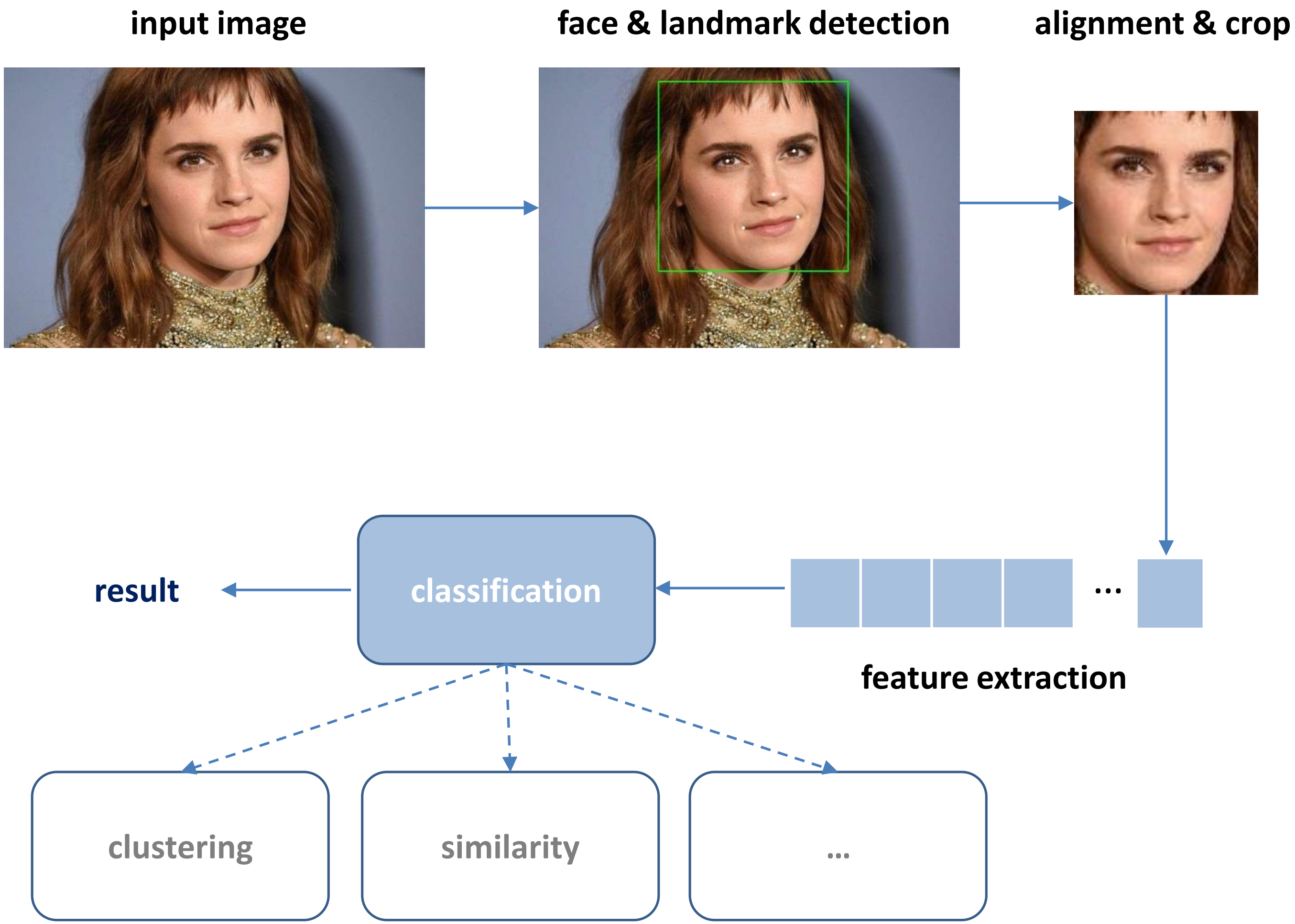
Face Verification

IS Sam Palladio

NOT Sam Palladio

Is this person X, 1:1

Face Recognition Workflow



input image

face & landmark detection

alignment & crop

feature extraction

classification

result

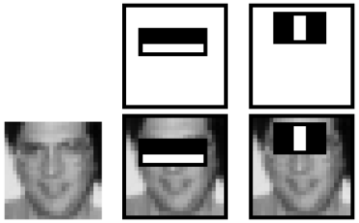
clustering

similarity

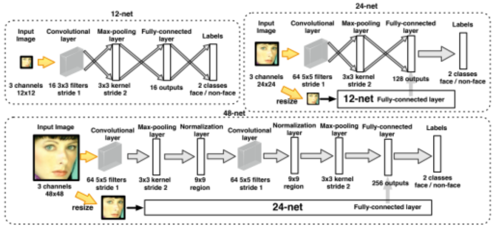
…

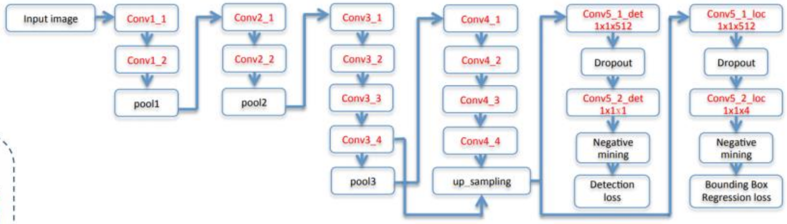
Face Detection Algorithms

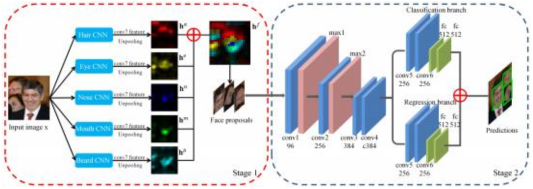
- Template Matching
- AdaBoost
  - VJ-cascade

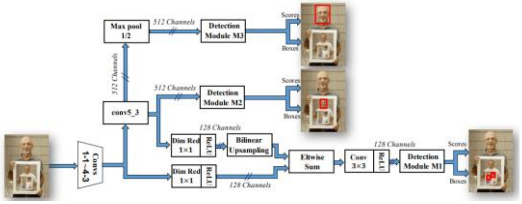


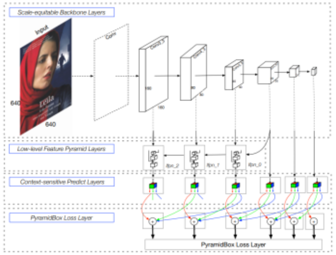
- DPM (deformable part model)
- Deep Learning
  - Cascade CNN
  - DenseBox
  - Faceness-Net
  - MTCNN
  - SSH
  - PyramidBox









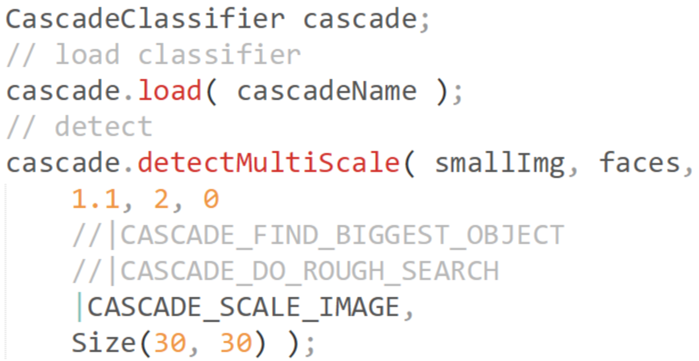

Scale-equitable Backbone Layers
Input
640
640
Low-level Feature Pyramid Layers
Context-sensitive Predict Layers
PyramidBox Loss Layer
PyramidBox Loss Layer

Face Detection API in OpenCV

- Traditional: cv::CascadeClassifier

  cv::CascadeClassifier::load()

  cv::CascadeClassifier::detectMultiScale()

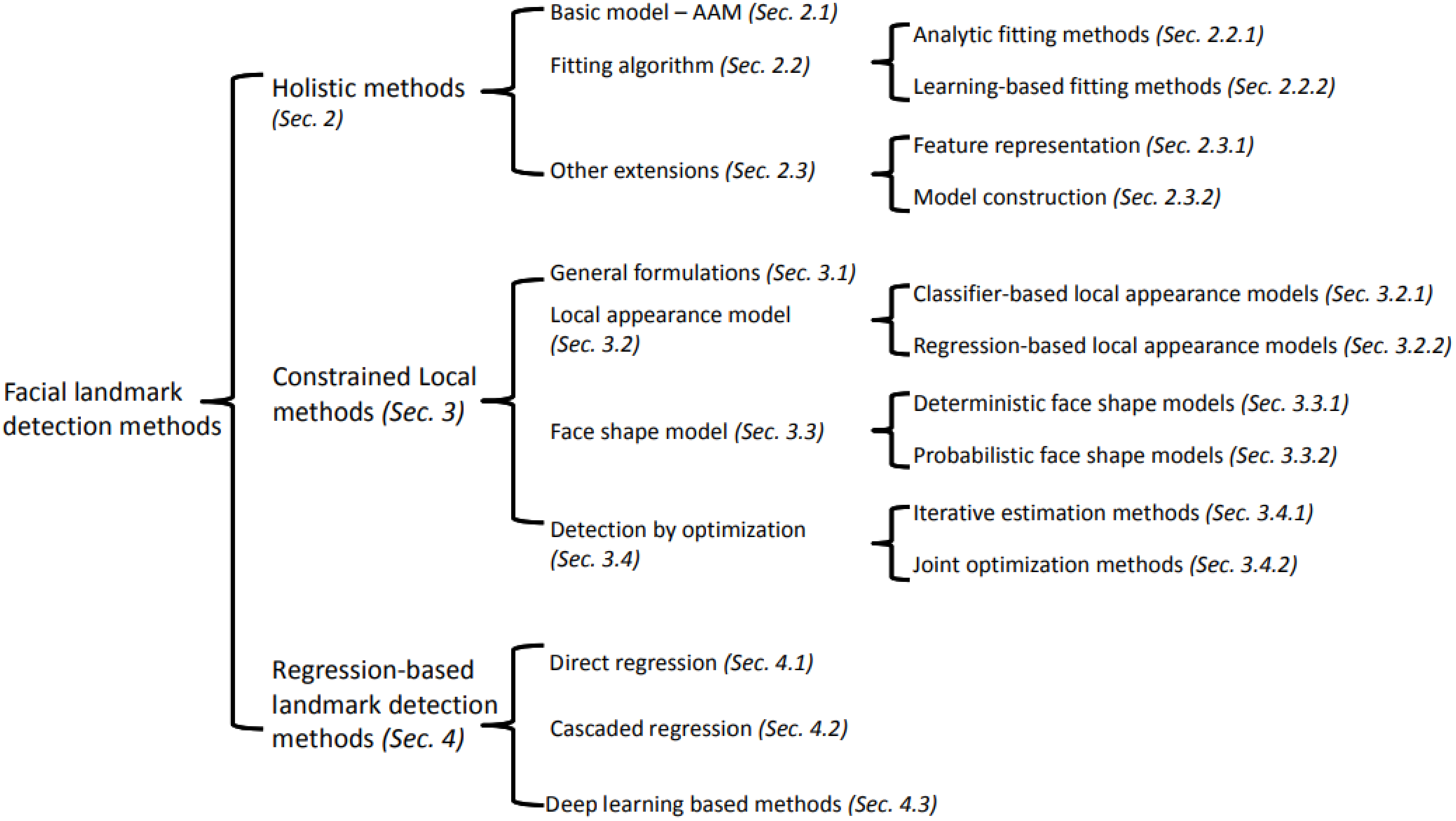


```
CascadeClassifier cascade;
// load classifier
cascade.load( cascadeName );
// detect
cascade.detectMultiScale( smallImg, faces,
    1.1, 2, 0
    //|CASCADE_FIND_BIGGEST_OBJECT
    //|CASCADE_DO_ROUGH_SEARCH
    |CASCADE_SCALE_IMAGE,
    Size(30, 30) );
```

https://docs.opencv.org/master/d4/d26/samples_2cpp_2facedetect_8cpp-example.html#_a2

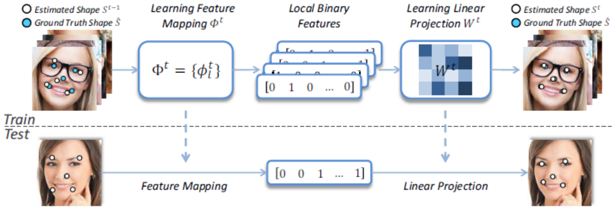
- Deep Learning: DNN module

Facial Landmark Detection Algorithms

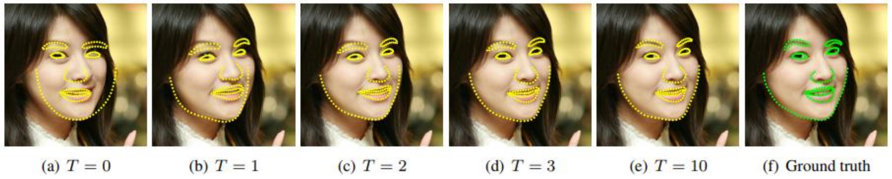


- Facial landmark detection methods
  - Holistic methods *(Sec. 2)*
    - Basic model – AAM *(Sec. 2.1)*
    - Fitting algorithm *(Sec. 2.2)*
      - Analytic fitting methods *(Sec. 2.2.1)*
      - Learning-based fitting methods *(Sec. 2.2.2)*
    - Other extensions *(Sec. 2.3)*
      - Feature representation *(Sec. 2.3.1)*
      - Model construction *(Sec. 2.3.2)*
  - Constrained Local methods *(Sec. 3)*
    - General formulations *(Sec. 3.1)*
    - Local appearance model *(Sec. 3.2)*
      - Classifier-based local appearance models *(Sec. 3.2.1)*
      - Regression-based local appearance models *(Sec. 3.2.2)*
    - Face shape model *(Sec. 3.3)*
      - Deterministic face shape models *(Sec. 3.3.1)*
      - Probabilistic face shape models *(Sec. 3.3.2)*
    - Detection by optimization *(Sec. 3.4)*
      - Iterative estimation methods *(Sec. 3.4.1)*
      - Joint optimization methods *(Sec. 3.4.2)*
  - Regression-based landmark detection methods *(Sec. 4)*
    - Direct regression *(Sec. 4.1)*
    - Cascaded regression *(Sec. 4.2)*
    - Deep learning based methods *(Sec. 4.3)*

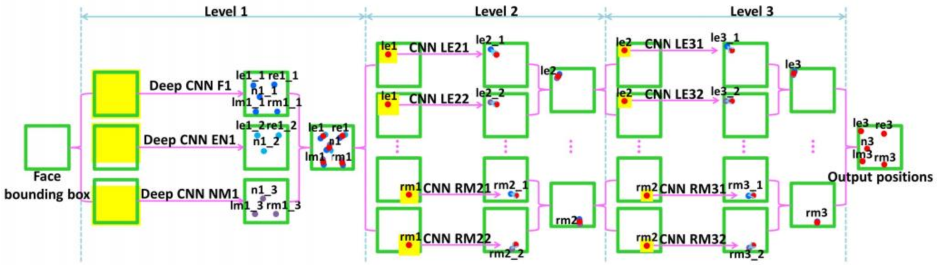
Yue Wu, and Qiang Ji, Facial Landmark Detection: a Literature Survey


Estimated Shape S^{t-1}
Ground Truth Shape $\hat{S}$
Learning Feature Mapping Φ^t
Local Binary Features
Learning Linear Projection W^t
Estimated Shape S^t
Ground Truth Shape $\hat{S}$
$\Phi^t = \{\phi_l^t\}$
[0 1 0 ... 0]
W^t
Train
Test
Feature Mapping
[0 0 1 ... 1]
Linear Projection

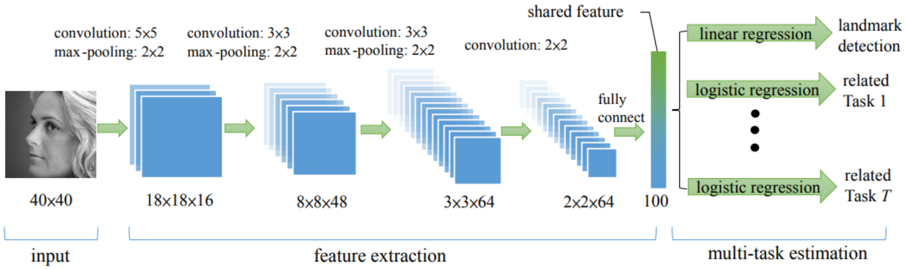
LBF


(a) $T = 0$ (b) $T = 1$ (c) $T = 2$ (d) $T = 3$ (e) $T = 10$ (f) Ground truth

ERT

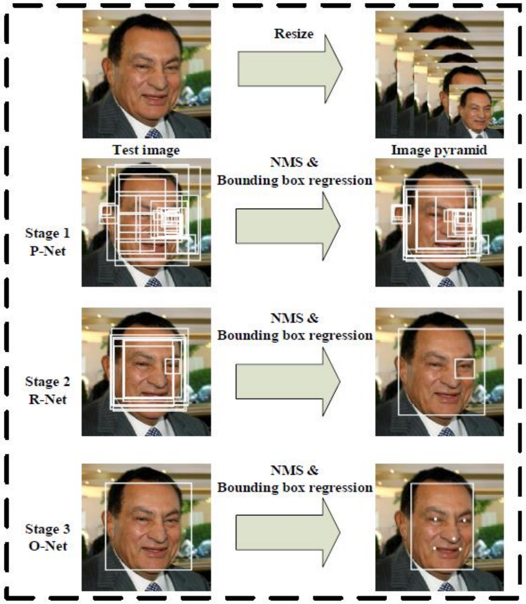


DCNN



TCDCN

MTCNN


Resize
Test image
Image pyramid
NMS & Bounding box regression
Stage 1 P-Net
NMS & Bounding box regression
Stage 2 R-Net
NMS & Bounding box regression
Stage 3 O-Net

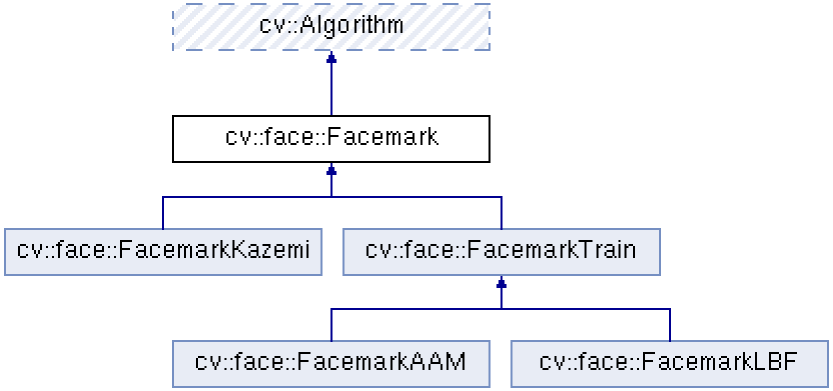
Facial Landmark Detection API in OpenCV

- Traditional: cv::face::Facemark in opencv_contrib

  Facemark::loadModel()

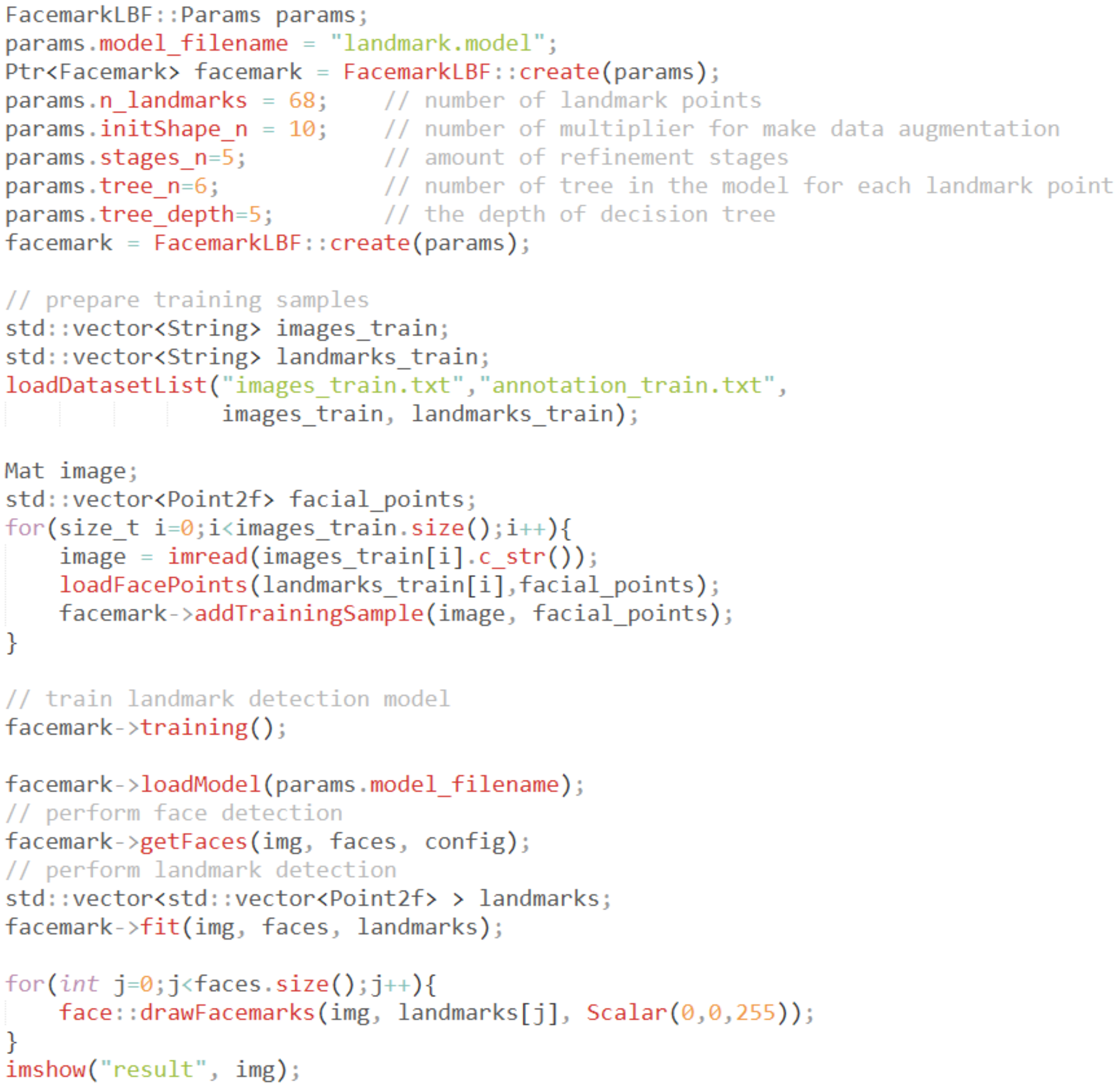
  Facemark::fit()

  Facemark::training()



https://docs.opencv.org/master/db/dd8/classcv_1_1face_1_1Facemark.html

- Deep Learning: DNN module



```
FacemarkLBF::Params params;
params.model_filename = "landmark.model";
Ptr<Facemark> facemark = FacemarkLBF::create(params);
params.n_landmarks = 68;     // number of landmark points
params.initShape_n = 10;     // number of multiplier for make data augmentation
params.stages_n=5;           // amount of refinement stages
params.tree_n=6;             // number of tree in the model for each landmark point
params.tree_depth=5;         // the depth of decision tree
facemark = FacemarkLBF::create(params);

// prepare training samples
std::vector<String> images_train;
std::vector<String> landmarks_train;
loadDatasetList("images_train.txt","annotation_train.txt",
                images_train, landmarks_train);

Mat image;
std::vector<Point2f> facial_points;
for(size_t i=0;i<images_train.size();i++){
    image = imread(images_train[i].c_str());
    loadFacePoints(landmarks_train[i],facial_points);
    facemark->addTrainingSample(image, facial_points);
}

// train landmark detection model
facemark->training();

facemark->loadModel(params.model_filename);
// perform face detection
facemark->getFaces(img, faces, config);
// perform landmark detection
std::vector<std::vector<Point2f> > landmarks;
facemark->fit(img, faces, landmarks);

for(int j=0;j<faces.size();j++){
    face::drawFacemarks(img, landmarks[j], Scalar(0,0,255));
}
imshow("result", img);
```

https://docs.opencv.org/master/d5/d47/tutorial_table_of_content_facemark.html
https://docs.opencv.org/master/de/d27/tutorial_table_of_content_face.html

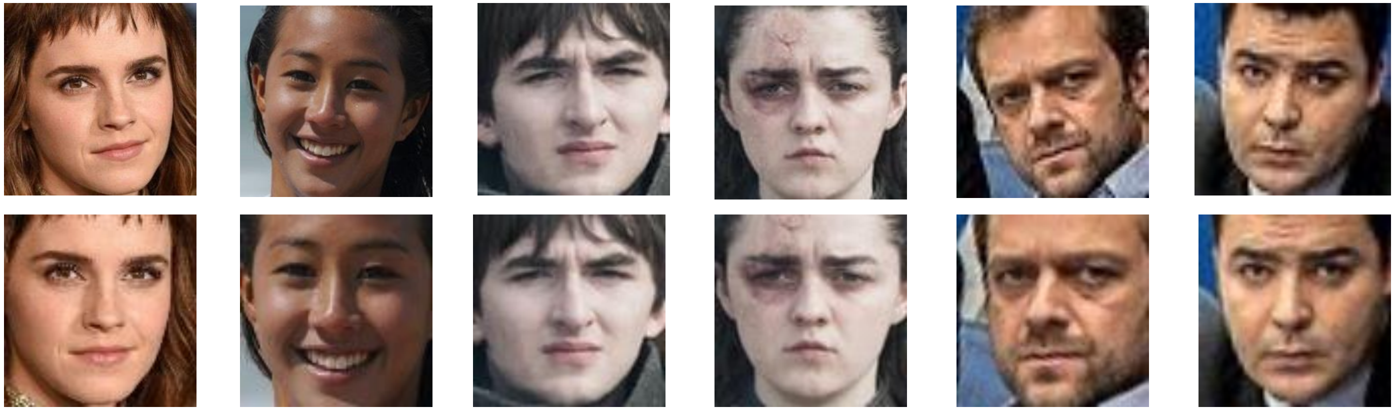
Face Alignment using OpenCV

```
estimateAffinePartial2D(InputArray from, InputArray to, OutputArray transform, ...)
```

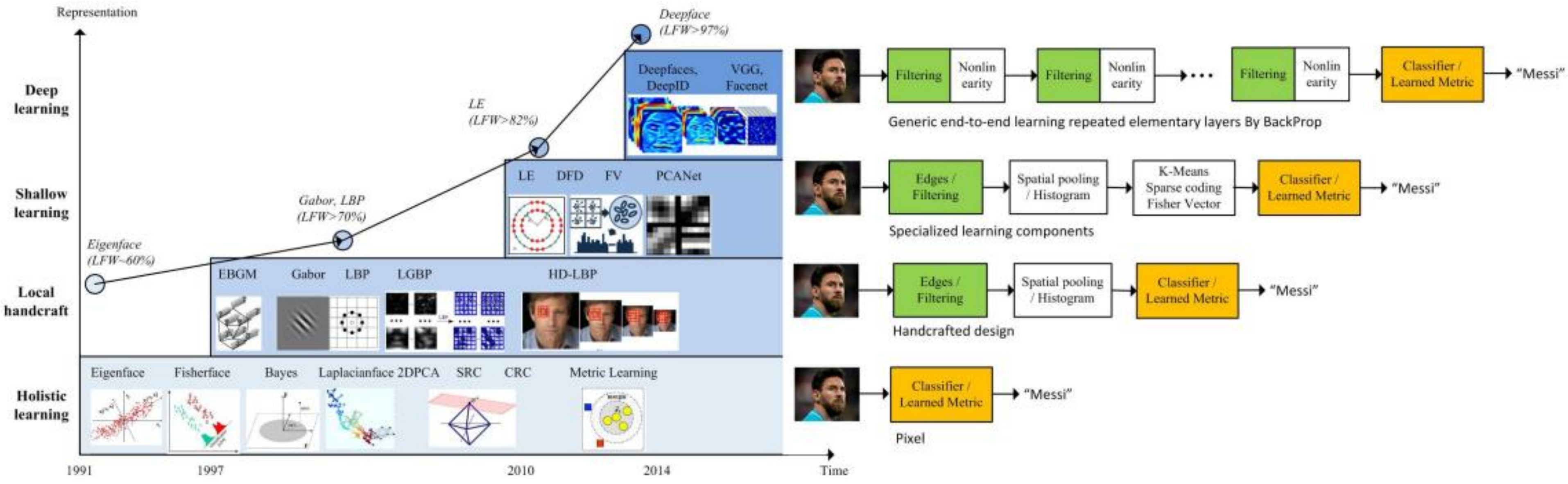
Compute an optimal limited affine transform with 4 degrees of freedom between two 2D point sets.

```
warpAffine(InputArray src, OutputArray dst, InputArray transform, Size dsize, ...)
```

Apply an affine transform to an image.



Feature Extraction Algorithms



Mei Wang, and Weihong Deng, Deep Face Recognition: A Survey

Face Recognition API in OpenCV

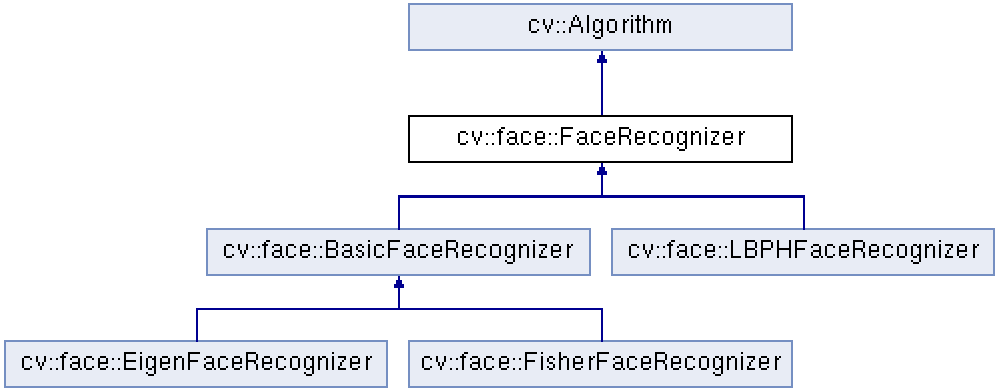
- Traditional: cv::face::FaceRecognizer

  FaceRecognizer::read()

  FaceRecognizer::predict()

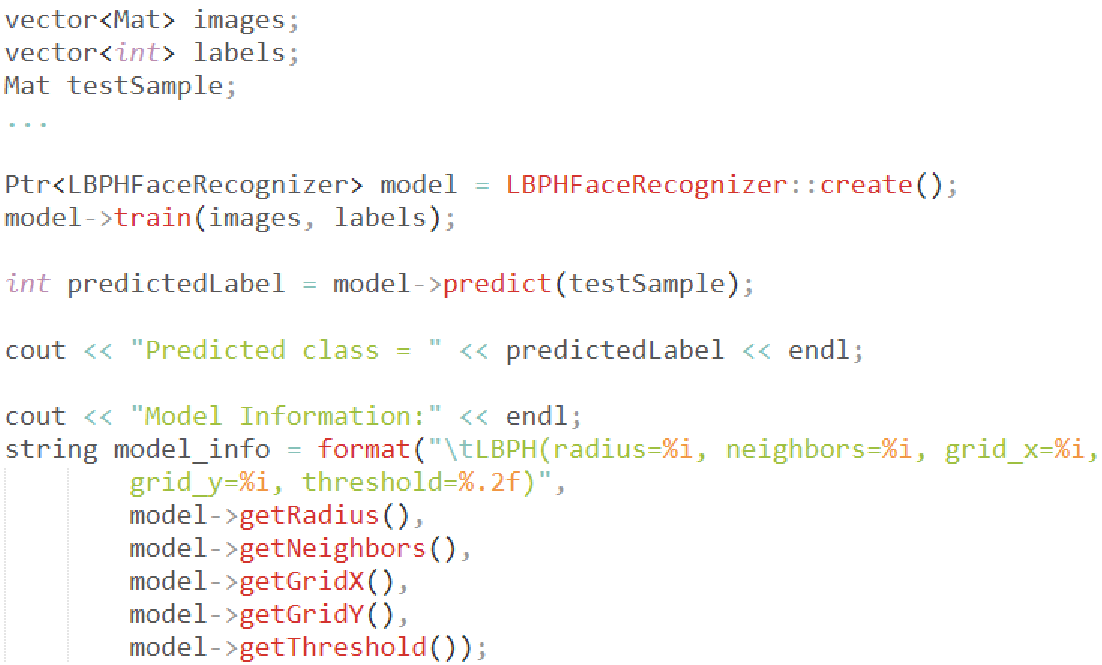
  FaceRecognizer::train()

  FaceRecognizer::write()


cv::Algorithm
cv::face::FaceRecognizer
cv::face::BasicFaceRecognizer
cv::face::LBPHFaceRecognizer
cv::face::EigenFaceRecognizer
cv::face::FisherFaceRecognizer

https://docs.opencv.org/master/dd/d65/classcv_1_1face_1_1FaceRecognizer.html

- Deep Learning: DNN module



```
vector<Mat> images;
vector<int> labels;
Mat testSample;
...

Ptr<LBPHFaceRecognizer> model = LBPHFaceRecognizer::create();
model->train(images, labels);

int predictedLabel = model->predict(testSample);

cout << "Predicted class = " << predictedLabel << endl;

cout << "Model Information:" << endl;
string model_info = format("\tLBPH(radius=%i, neighbors=%i, grid_x=%i,
        grid_y=%i, threshold=%.2f)",
        model->getRadius(),
        model->getNeighbors(),
        model->getGridX(),
        model->getGridY(),
        model->getThreshold());
```

Face Recognition with OpenCV:
https://docs.opencv.org/master/da/d60/tutorial_face_main.html

OpenCV DNN module

DNN module is implemented @opencv_contrib at v3.1.0 in Dec. 2015 and moved to main repo at v3.3.0 in Aug, 2017.

- Inference only
- Support different network formats: Caffe, TensorFlow, Darknet, Torch, ONNX compatible (PyTorch, Caffe2, MXNet, CNTK, …)
- Support hundreds of network
- Several backends available: CPU, GPU, VPU
- Easy-to-use API
- Low memory consumption (layers fusion, intermediate blobs reusing)
- Faster forward pass comparing to training frameworks (fusion, backends)

Easy-to-use:

```
Net net = readNet(model_name, model_config);
Mat blob = blobFromImage(img, ...);
net.setInput(blob);
Mat out = net.forward();
```

Fast:

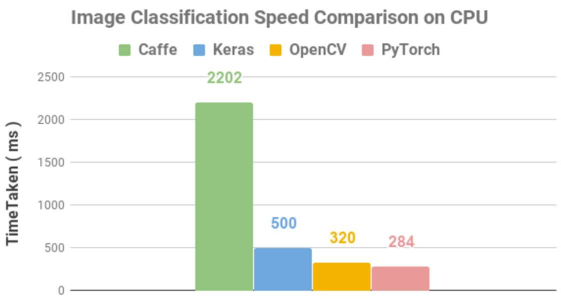
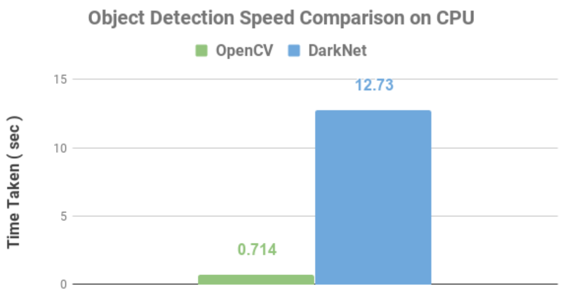
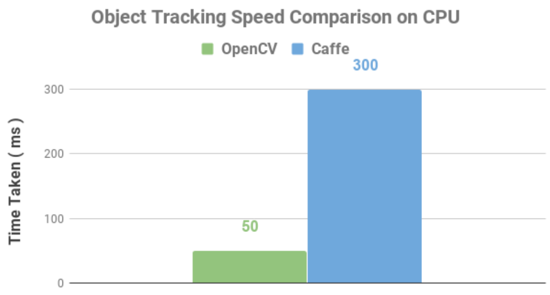


Image Classification Speed Comparison on CPU

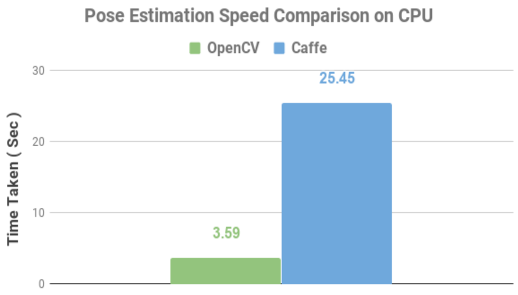
OpenCV is faster by 7X (Caffe) and 1.5X (Keras)


Object Detection Speed Comparison on CPU

OpenCV is 18X Faster


Object Tracking Speed Comparison on CPU

OpenCV is 6X Faster


Pose Estimation Speed Comparison on CPU

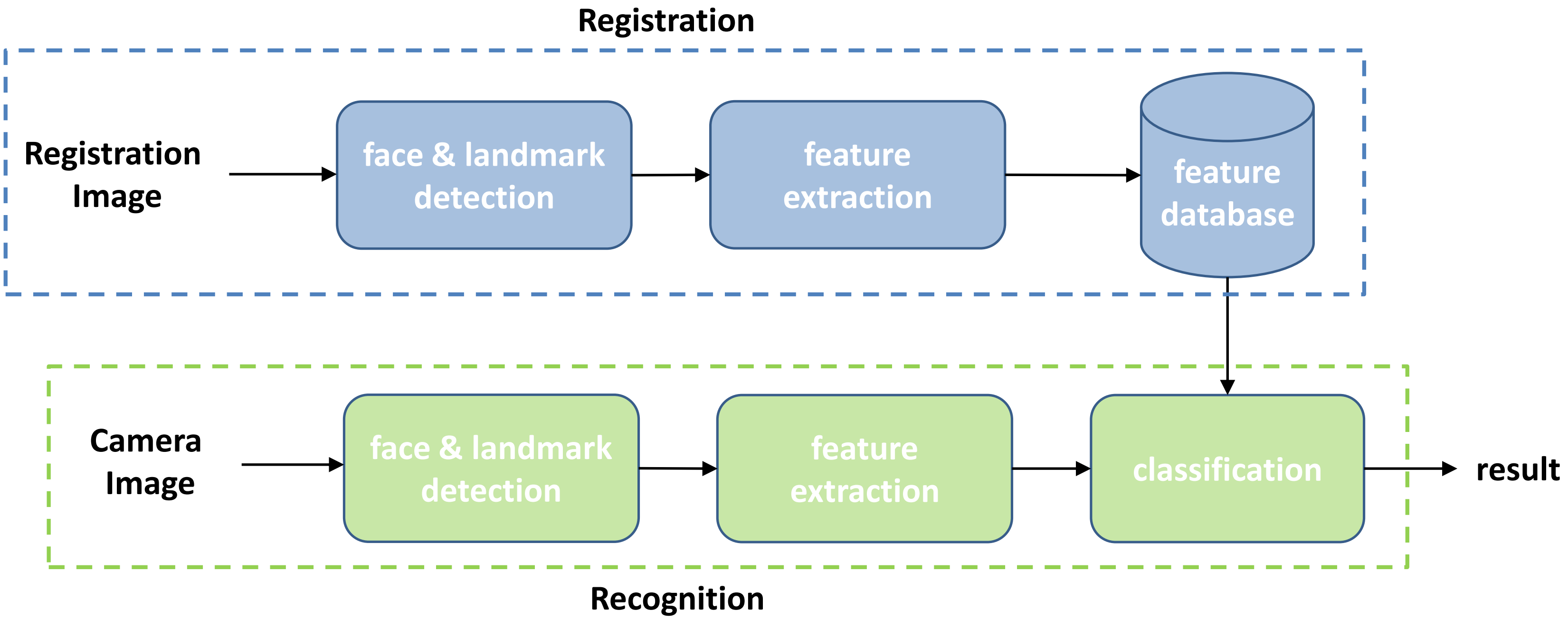
OpenCV is 7X Faster

https://www.learnopencv.com/cpu-performance-comparison-of-opencv-and-other-deep-learning-frameworks/

OpenCV DNN Key Dates

3.1.0 Dec, 2015	(GSoC) dnn module implementation @ opencv_contrib. Caffe and Torch frameworks.
3.2.0 Dec, 2016	(GSoC) TensorFlow importer. New nets: object detection (SSD), semantic segmentation
3.3.0 Aug, 2017	Substantial efficiency improvements, optional Halide backend (CPU/GPU), dnn moved from opencv_contrib to the main repo
3.3.1 Oct, 2017	OpenCL backend. Darknet importer
3.4.0 Dec, 2017	JavaScript bindings for dnn module. OpenCL backend speedup.
3.4.1 Feb, 2018	Intel's Inference Engine backend (CPU)
3.4.2 Jul, 2018	FP16 for OpenCL backend. GPU (FP32/FP16) and VPU (Myriad 2) for IE backend. Import of OpenVINO models (IR format). Custom layers support. YOLOv3 support.
4.0.0 Sep, 2018	ONNX models import, Vulkan backend support.
4.1.0 Apr, 2019	Myriad X support, better IE support (samples, layers), improved TensorFlow Object Detection API support.
4.1.1 July, 2019	3D convolution networks initial support
4.1.2 Oct, 2019	Introduces dnn::Model class and set of task-specific classes dnn::ClassificationModel, dnn::DetectionModel, dnn::SegmentationModel.
4.2.0 Dec, 2019	Integrated GSoC project with CUDA backend
4.3.0 Mar, 2020	Tengine backend for ARM CPU (collaboration between OpenCV China and Open AI Lab)

Face Recognition System Architecture



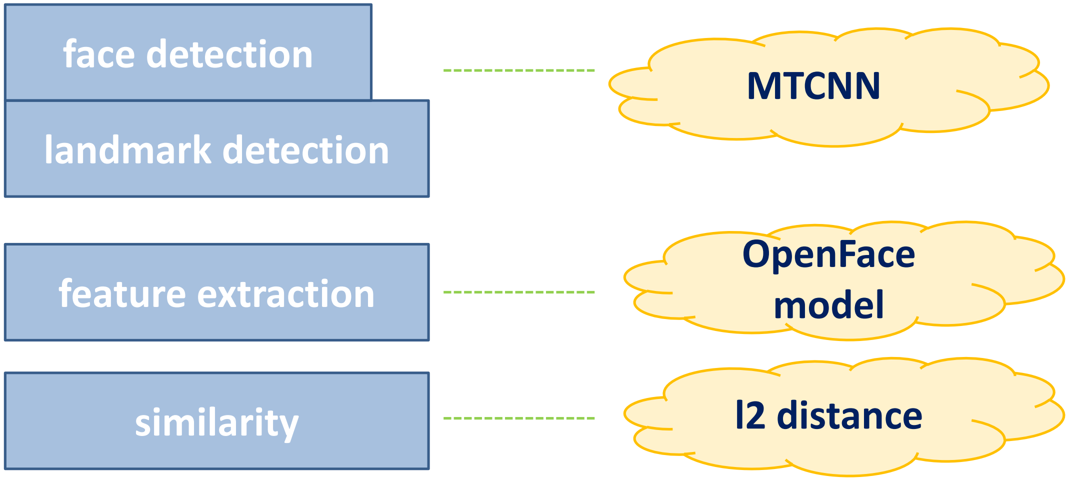
Build the system using OpenCV

devices

usb camera

arm dev board

algorithms



face detection

landmark detection

feature extraction

similarity

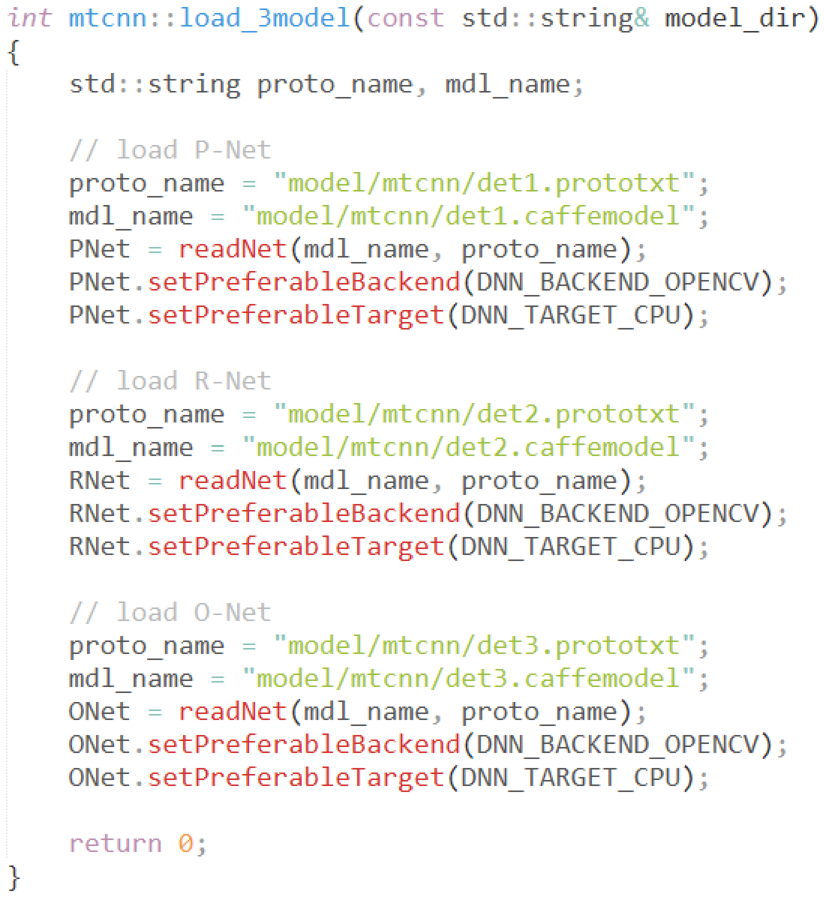
MTCNN

OpenFace model

l2 distance

Build the system

face & landmark detection

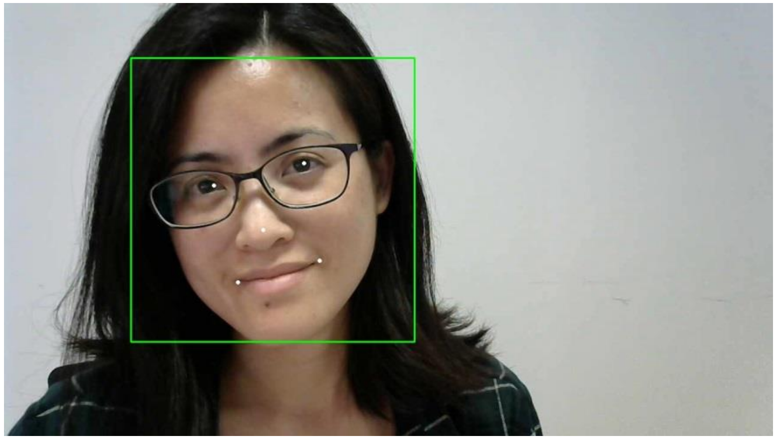


```
int mtcnn::load_3model(const std::string& model_dir)
{
    std::string proto_name, mdl_name;

    // load P-Net
    proto_name = "model/mtcnn/det1.prototxt";
    mdl_name = "model/mtcnn/det1.caffemodel";
    PNet = readNet(mdl_name, proto_name);
    PNet.setPreferableBackend(DNN_BACKEND_OPENCV);
    PNet.setPreferableTarget(DNN_TARGET_CPU);

    // load R-Net
    proto_name = "model/mtcnn/det2.prototxt";
    mdl_name = "model/mtcnn/det2.caffemodel";
    RNet = readNet(mdl_name, proto_name);
    RNet.setPreferableBackend(DNN_BACKEND_OPENCV);
    RNet.setPreferableTarget(DNN_TARGET_CPU);

    // load O-Net
    proto_name = "model/mtcnn/det3.prototxt";
    mdl_name = "model/mtcnn/det3.caffemodel";
    ONet = readNet(mdl_name, proto_name);
    ONet.setPreferableBackend(DNN_BACKEND_OPENCV);
    ONet.setPreferableTarget(DNN_TARGET_CPU);

    return 0;
}
```

```
Mat blob;
blobFromImage(img, blob, 1.0, Size(scale_w, scale_h), Scalar(), false, false);
PNet.setInput(blob);
// run model and get the outputs
std::vector<std::string> outNames;
outNames.push_back("conv4-2");
outNames.push_back("prob1");
std::vector<Mat> outs;
PNet.forward(outs, outNames);
Mat coords = outs[0];    // box regression
Mat probs = outs[1];     // scores
```

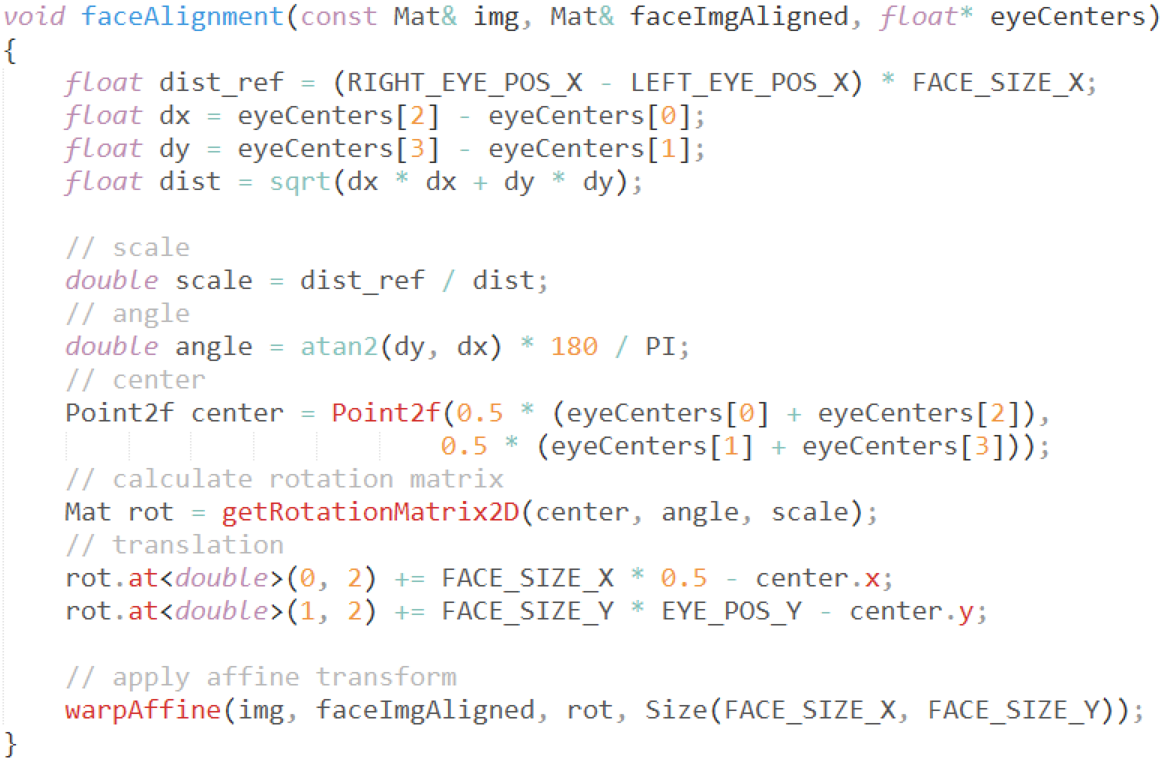
```
Mat blob;
blobFromImages(proposals, blob, 1.0, Size(), Scalar(), false, false);
RNet.setInput(blob);
// run model and get the outputs
std::vector<std::string> outNames;
outNames.push_back("conv5-2");
outNames.push_back("prob1");
std::vector<Mat> outs;
RNet.forward(outs, outNames);
Mat coords = outs[0];    // box regression
Mat probs = outs[1];     // scores
```

```
Mat blob;
blobFromImages(imgs, blob, 1.0, Size(), Scalar(), false, false);
ONet.setInput(blob);
// run model and get the outputs
std::vector<std::string> outNames;
outNames.push_back("conv6-2");
outNames.push_back("conv6-3");
outNames.push_back("prob1");
std::vector<Mat> outs;
ONet.forward(outs, outNames);
Mat coords = outs[0];        // box regression
Mat landmarks = outs[1];     // landmarks
Mat probs = outs[2];         // scores
```

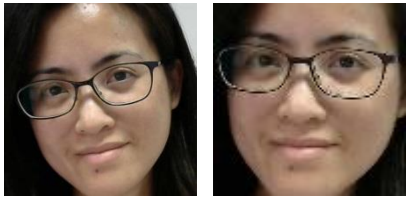


Build the system

face alignment

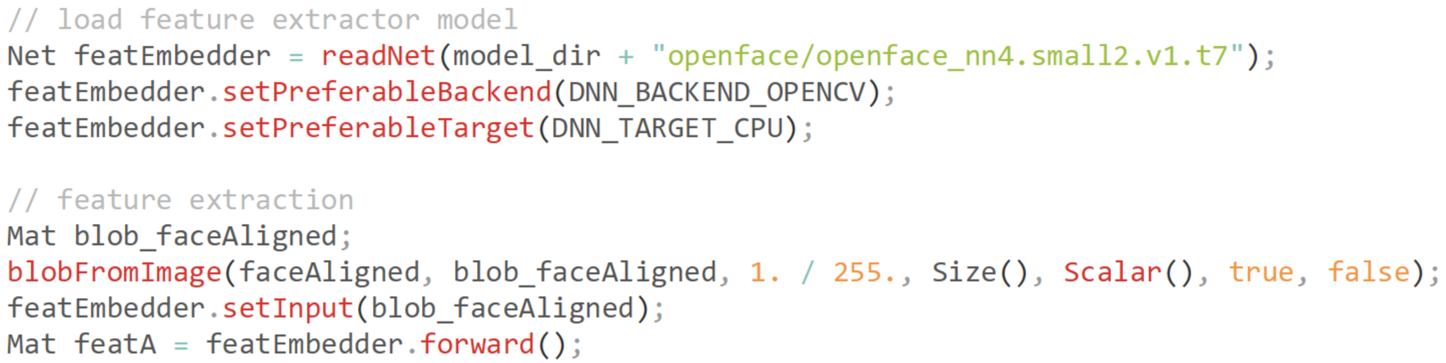


```
void faceAlignment(const Mat& img, Mat& faceImgAligned, float* eyeCenters)
{
    float dist_ref = (RIGHT_EYE_POS_X - LEFT_EYE_POS_X) * FACE_SIZE_X;
    float dx = eyeCenters[2] - eyeCenters[0];
    float dy = eyeCenters[3] - eyeCenters[1];
    float dist = sqrt(dx * dx + dy * dy);

    // scale
    double scale = dist_ref / dist;
    // angle
    double angle = atan2(dy, dx) * 180 / PI;
    // center
    Point2f center = Point2f(0.5 * (eyeCenters[0] + eyeCenters[2]),
                            0.5 * (eyeCenters[1] + eyeCenters[3]));
    // calculate rotation matrix
    Mat rot = getRotationMatrix2D(center, angle, scale);
    // translation
    rot.at<double>(0, 2) += FACE_SIZE_X * 0.5 - center.x;
    rot.at<double>(1, 2) += FACE_SIZE_Y * EYE_POS_Y - center.y;

    // apply affine transform
    warpAffine(img, faceImgAligned, rot, Size(FACE_SIZE_X, FACE_SIZE_Y));
}
```

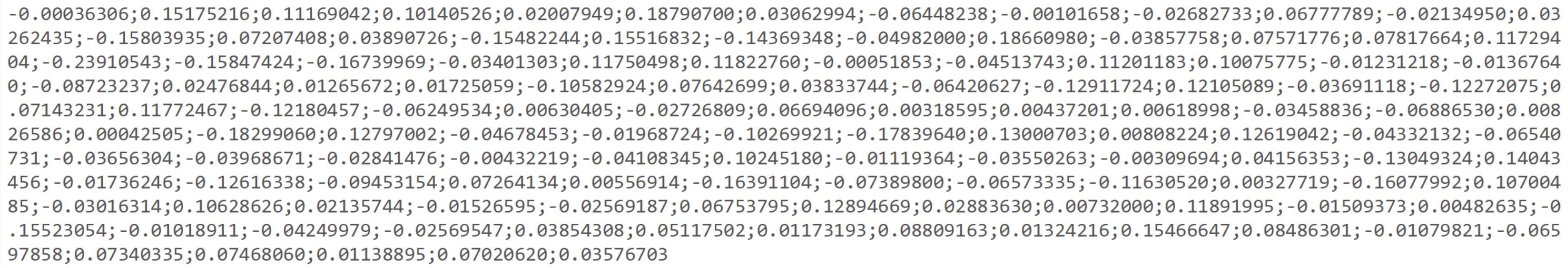


Build the system

feature extraction



```
// load feature extractor model
Net featEmbedder = readNet(model_dir + "openface/openface_nn4.small2.v1.t7");
featEmbedder.setPreferableBackend(DNN_BACKEND_OPENCV);
featEmbedder.setPreferableTarget(DNN_TARGET_CPU);

// feature extraction
Mat blob_faceAligned;
blobFromImage(faceAligned, blob_faceAligned, 1. / 255., Size(), Scalar(), true, false);
featEmbedder.setInput(blob_faceAligned);
Mat featA = featEmbedder.forward();
```



```
-0.00036306;0.15175216;0.11169042;0.10140526;0.02007949;0.18790700;0.03062994;-0.06448238;-0.00101658;-0.02682733;0.06777789;-0.02134950;0.03
262435;-0.15803935;0.07207408;0.03890726;-0.15482244;0.15516832;-0.14369348;-0.04982000;0.18660980;-0.03857758;0.07571776;0.07817664;0.117294
04;-0.23910543;-0.15847424;-0.16739969;-0.03401303;0.11750498;0.11822760;-0.00051853;-0.04513743;0.11201183;0.10075775;-0.01231218;-0.0136764
0;-0.08723237;0.02476844;0.01265672;0.01725059;-0.10582924;0.07642699;0.03833744;-0.06420627;-0.12911724;0.12105089;-0.03691118;-0.12272075;0
.07143231;0.11772467;-0.12180457;-0.06249534;0.00630405;-0.02726809;0.06694096;0.00318595;0.00437201;0.00618998;-0.03458836;-0.06886530;0.008
26586;0.00042505;-0.18299060;0.12797002;-0.04678453;-0.01968724;-0.10269921;-0.17839640;0.13000703;0.00808224;0.12619042;-0.04332132;-0.06540
731;-0.03656304;-0.03968671;-0.02841476;-0.00432219;-0.04108345;0.10245180;-0.01119364;-0.03550263;-0.00309694;0.04156353;-0.13049324;0.14043
456;-0.01736246;-0.12616338;-0.09453154;0.07264134;0.00556914;-0.16391104;-0.07389800;-0.06573335;-0.11630520;0.00327719;-0.16077992;0.107004
85;-0.03016314;0.10628626;0.02135744;-0.01526595;-0.02569187;0.06753795;0.12894669;0.02883630;0.00732000;0.11891995;-0.01509373;0.00482635;-0
.15523054;-0.01018911;-0.04249979;-0.02569547;0.03854308;0.05117502;0.01173193;0.08809163;0.01324216;0.15466647;0.08486301;-0.01079821;-0.065
97858;0.07340335;0.07468060;0.01138895;0.07020620;0.03576703
```

calculate similarity

```
cv::norm(featA - featB)
```

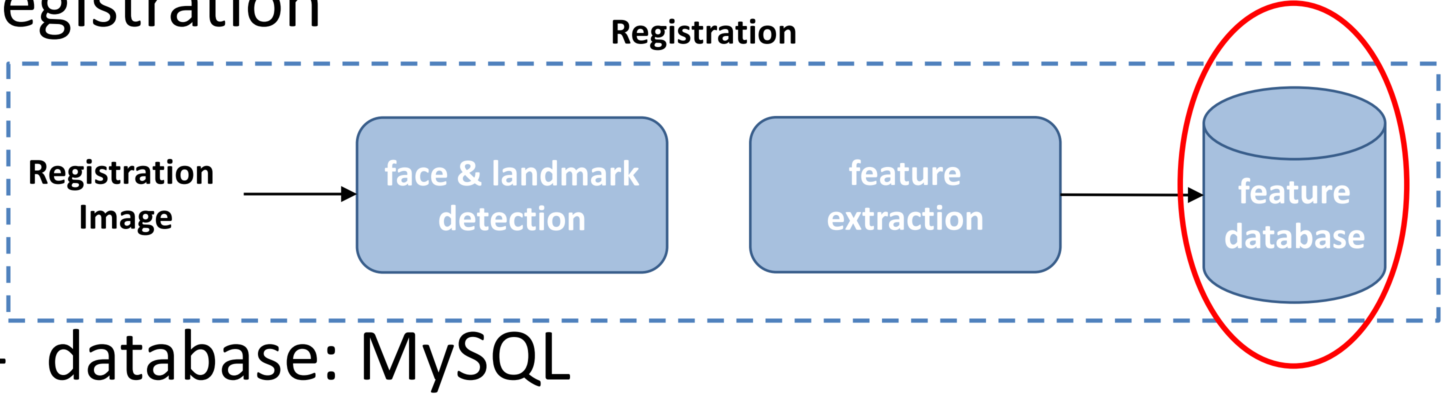
Demo1: on laptop

Demo2: on ARM dev board

Demo3: demo with registration on ARM board

Issues

- Training
  - traditional methods: train() in OpenCV
  - deep learning: using deep learning frameworks
- Registration



  - database: MySQL
- UIs
  - OpenCV with QT

Exercise

A. Build a complete face recognition system using OpenCV on ARM board, and submit a report in English about the system.

B. Build any other kind of biometric recognition system using OpenCV on ARM board, and submit a report in English about the system.

Note
Submit to: jia.wu@opencv.org.cn
Deadline: Jan. 15, 2020

Thank You !