embedded VISION SUMMIT 2018

Developing Computer Vision Algorithms for Networked Cameras

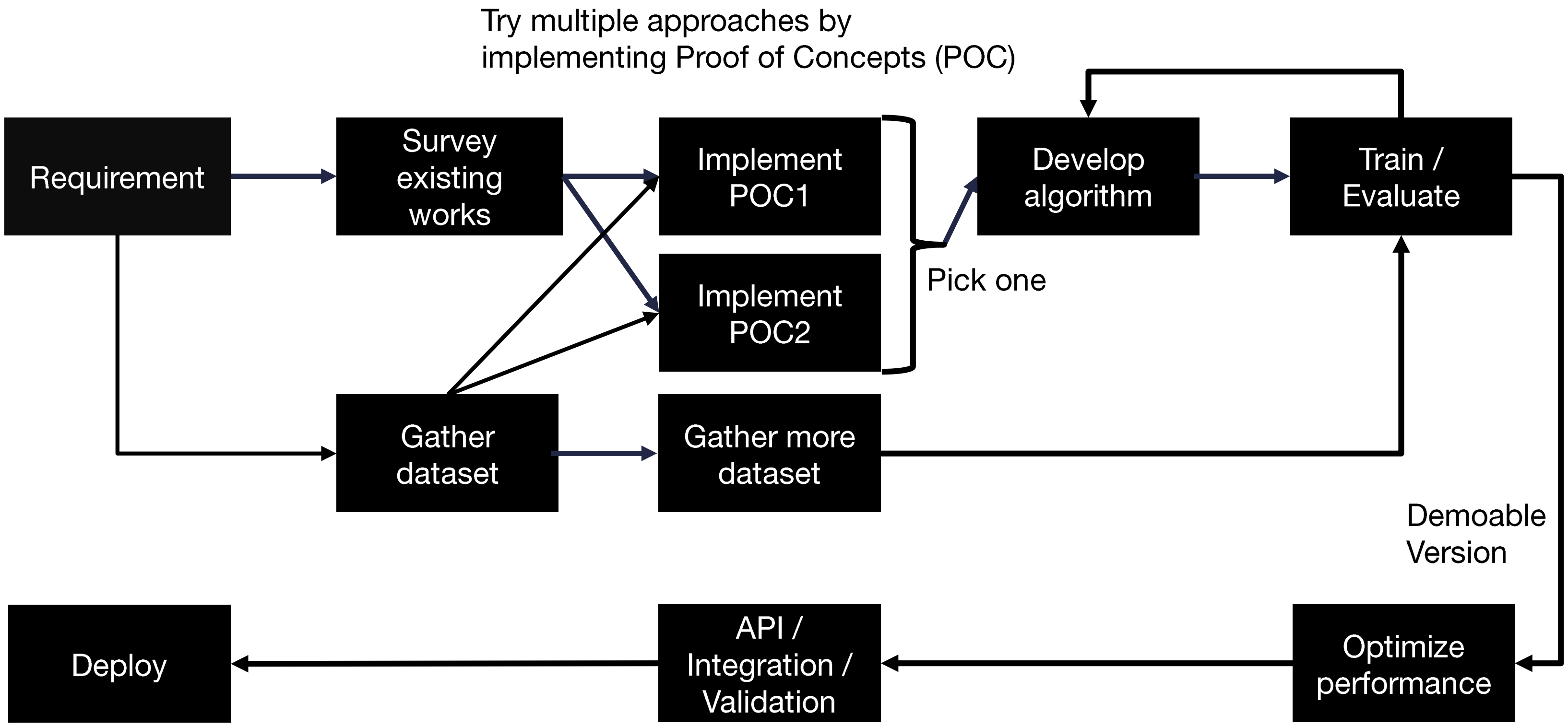
Dukhwan Kim, Software Architect

5/22/2018

Outline

- Algorithm development overall flow
- High-level algorithm design
- Data acquisition
- Hardware optimization
- Continuous update
- Conclusion

Algorithm Development Overall Flow



Challenges

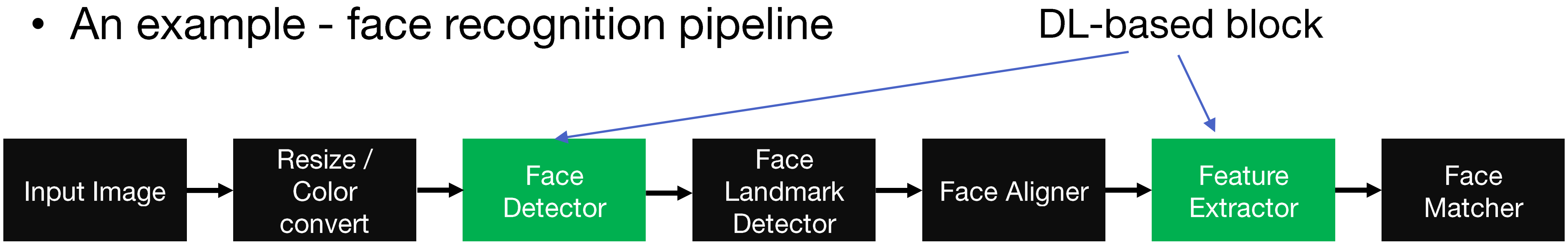
- Too many design choices from high-level algorithm design stage
- Dataset acquisition is not trivial
- Hardware optimization – How to scale?
- One-time deployment is not the end of the work

High-level Algorithm Design

- Probably the first question - Traditional CV vs. DL-based?
- Traditional CV may be still good for
  - Certain type of algorithms such as Object Tracking, Camera Tampering Detection and Change Detection
  - Edge devices with limited hardware resource
- However, pure DL-based algorithms are rare in real applications → Mostly hybrid (Traditional CV + DL)
  - Even for a simple detection algorithm, we need a tracker
  - Preprocessing (e.g., color conversion, resize) and post-processing (Non-maximum suppression)

High-level Algorithm Design

- An example - face recognition pipeline



- Tradeoff 1 - Face Detector can be done using Traditional CV – ***if we know faces are mostly frontal and of good quality***
- Tradeoff 2 - Face Landmark Detector + Face aligner can be done with DL – ***if we can afford to have bigger computing power***
- In real applications, it's typical to have hybrid pipeline

High-level Algorithm Design Process

- Start with known/public algorithms
  - Inventing whole new topology or architecture is not practical (e.g., Face recognition - DeepFace, FaceNet)
- Try multiple approaches (preferably in parallel – to reduce training time)
- Pick the most promising one
  - != The most accurate one
  - == The one best satisfies the ***requirement***
- Then refine / tweak existing algorithms as needed and add a few ideas to see if that helps
  - e.g., replace feature extraction block

Example of Good Requirement

Item	Requirement
Feature	Recognizing blacklisted people in the public area (indoor and outdoor)
Expected output	The algorithm shall provide list of people who are in the blacklist from incoming video at realtime
Input image	Max 1080p, RGB, 15 ~ 60 fps
Max num of persons	Maximum 6 persons per frame
Camera configuration	0 ~ 45 degree; distance from camera to face is up to 10 m; Face rotation coverage : roll +/- 20, yaw +/- 30, pitch +/- 45
Accuracy	99% verification with LFW dataset; 80% identification with MegaFace dataset (from 1 M faces)
Database	Up to 1 million people; number of images per person : 1 ~ 20
Performance	Min 30 fps throughput with <15% CPU residency; Latency less than 30 ms

Could also be added: power consumption, hardware platform, OS, API, etc.

Expected input and output

Assumption on camera configuration

Measurable accuracy with datasets and test protocol

Performance under a specific circumstance

Challenge in Data Acquisition

- Face detection / recognition
  - Public datasets are pretty good because of long history
    - But it's different story whether you can use them due to privacy and license
  - Privacy laws prohibit gathering dataset from public space
  - We mostly rely on private datasets
    - including images captured under controlled environment for corner cases

Challenge in Data Acquisition

- And challenges from the real world
  - Images from our cameras have blurred faces. Can your algorithm work in the situation?
  - Does your algorithm work on IR cameras?
  - Our cameras are mounted at very high locations (camera angle can be up to 90 degrees). Can your face detector / recognizer work?
  - Can we use passport photos for registration?

- We need to gather additional dataset in order to be able to say “Yes”

Plan for Data Acquisition

- Ways to gather datasets
  - Take photos/videos of yourself, your family and colleagues
  - Use public dataset (free / purchase) if legitimate
  - Work with 3rd party data vendor
  - Work with customers
  - Use data augmentation or transfer learning
- 60:20:20 theory
  - 60% comes from common source (public or 3rd party)
  - 20% comes from user-scenario specific source (e.g., testbed)
  - Still remains 20% that only can be obtained in real situation

Challenge in Hardware Optimization

- Challenge 1 – Hardware optimization is not scalable
  - New platform / chipset will require new work
  - Multiplied by number of available hardware
    - CPU, GPU, FPGA, DSP and DL accelerator
    - Need to apply different optimization strategy per hardware
  - What if the algorithm changes even slightly
- It is essential that chip makers provide good frameworks / toolsets
- A good strategy is to develop algorithms based on the frameworks / toolsets provided by chip makers from the beginning

Challenge in Hardware Optimization

Hardware	Challenges	Tools
CPU	• Challenge is to reduce overall residency and power consumption • Compilers are usually very good, but adding directives (as in OpenMP) helps for SIMD/Multi-threaded applications	Traditional compilers, Halide
GPU	• Challenge is to increase hardware utilization by having good parallelization strategy, memory access pattern, etc. • Using precompiled libraries is a good starting point	OpenCL, OpenVX, CUDA, Halide
FPGA	• Generally need more experience and knowledge on hardware for efficient programming • Not efficient to use for programs with many logics and branches	OpenCL, HDL
DSP + Accelerator	• Highly dependent on compiler/tool support from the vendor • May be very good at running some functions, but may not be good at running others	OpenCL, OpenVX, Dedicated compilers

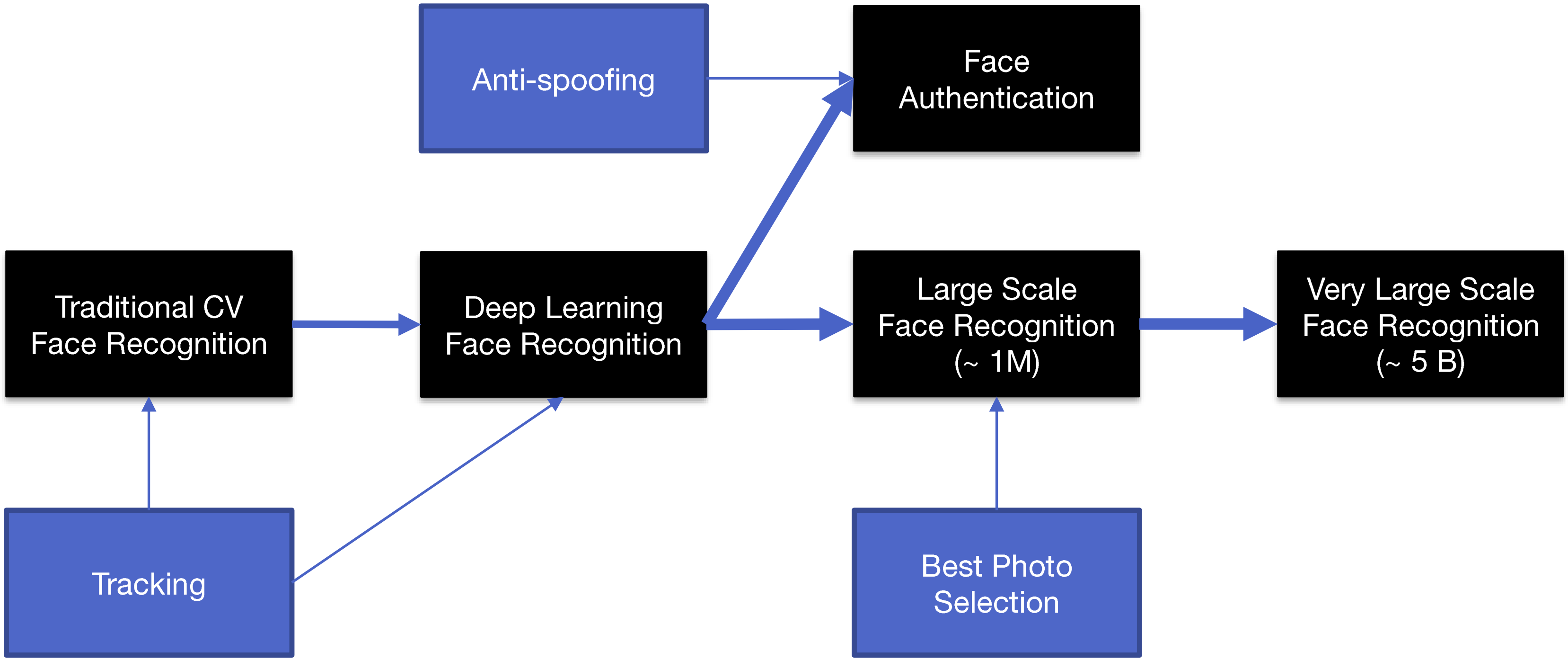
Challenge in Hardware Optimization

- Challenge 2 – Most algorithms are hybrid (Traditional CV + DL)
  - Traditional CV could be bottleneck if not well-optimized and balanced
- If the framework / toolset can handle Traditional CV kernels well, that’s great
- If not, apply traditional ways of optimization, for example
  - use performance analysis tools and optimize hotspots first
  - reduce data transfer between hardware blocks
  - carefully design high-level software architecture
- Or consider developing middleware that provides abstraction

Why Continuous Update

- Once installed, cameras tend to stay for a long time (3 ~ 5 years), but
  - Customers want to add new features
  - Algorithm / model updates are inevitable
  - For example:
    - Can you reduce the feature size of face recognition? Another vendor has very small size!
    - Can you add anti-spoofing feature?
    - Database size can grow up to 10 M persons. Is it ok?

Evolution of Face Recognition



Summary & Some Advices

- Developing algorithms (data engineering, new network topology) is just one thing and only the beginning
- First, clearly understand what you want to build
- Data acquisition is the most challenging part, so plan ahead
- Work closely with hardware
- For productization, continuous updating is also important
- Employ some level of realism – Deep Learning made lots of things possible, but it's not a silver bullet (yet)

Resources

- Face recognition
  - DeepFace https://research.fb.com/publications/deepface-closing-the-gap-to-human-level-performance-in-face-verification/
  - FaceNet https://arxiv.org/abs/1503.03832

- OpenVINO™ (formerly Intel® Computer Vision SDK) https://software.intel.com/en-us/openvino-toolkit
- Please check out Intel booth for demos and more information!