Robust Real-Time Face Detection

From an academic paper to a custom built ASIC accelerator in 2x45 minutes

LiTH Linköping
February 24th 2020

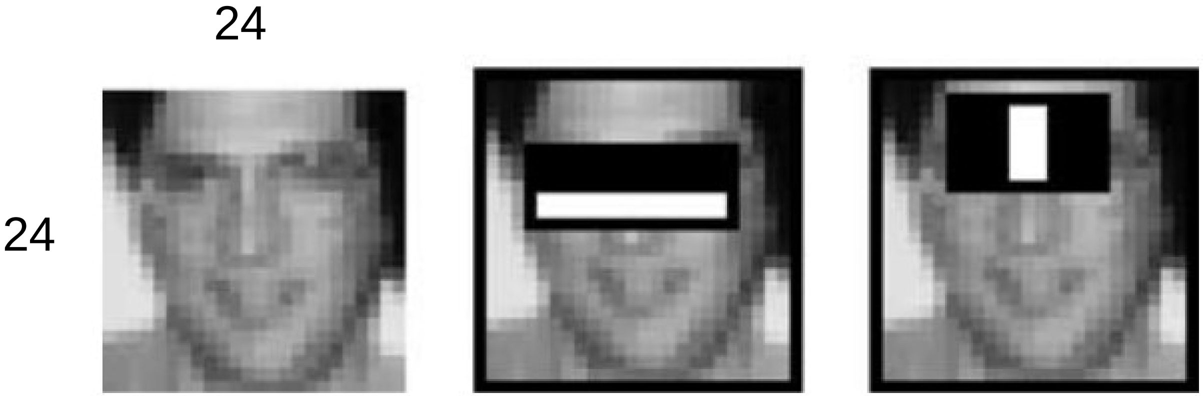
Anders Lloyd
Axis Communications

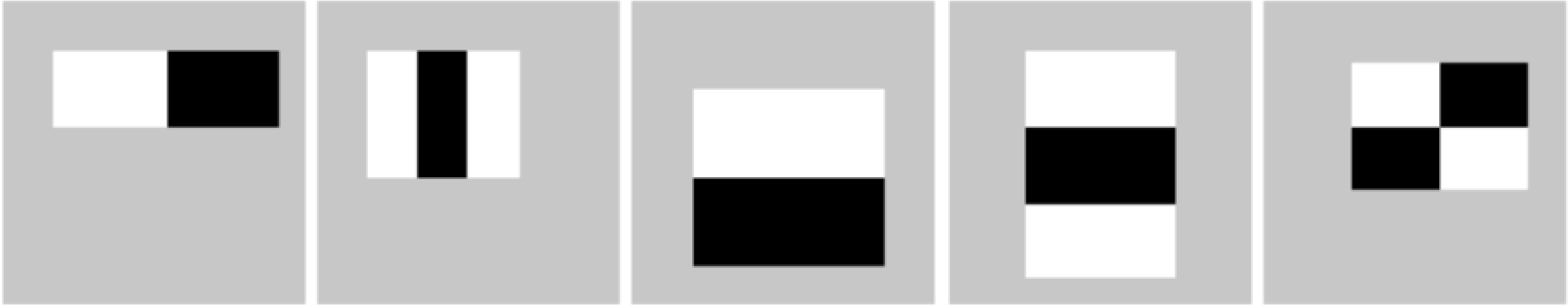
Hardware for Machine Learning
PhD Course Jan-Mar 2020

In 2001 Paul Viola and Michael Jones presented the (first) real time face detector.

“Operating on 384 by 288 pixel images, faces are detected at 15 frames per second on a conventional 700 MHz Intel Pentium III.“

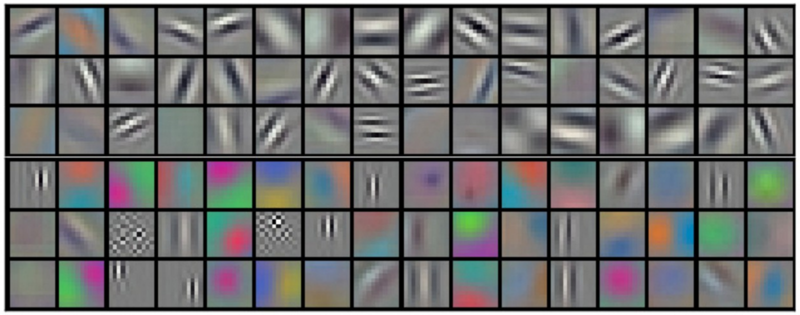
Quotes from: http://www.vision.caltech.edu/html-files/EE148-2005-Spring/pprs/viola04ijcv.pdf





A **limited set** of basic (symmetric) feature types.

“For this system there is also a second critical motivation for features: the feature-based system operates much faster than a pixel-based system.”



Compare with Gabor filter

There are two general problems to solve:

- How can we calculate these features (difference between sum of pixels) really fast?
- How do we handle large faces (>24x24) in our 384x288 input image?

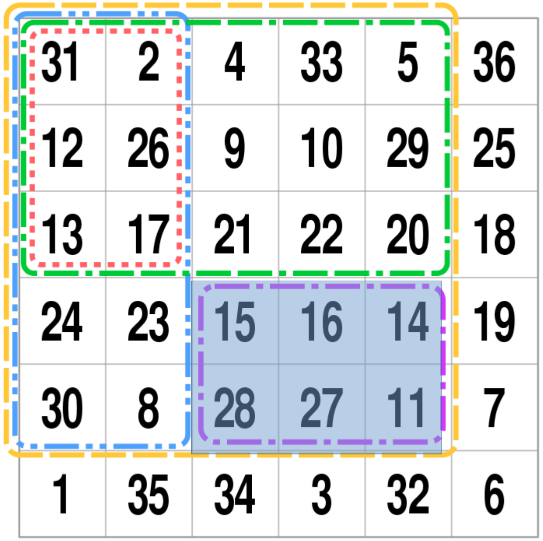
They found one solution to both problems

“In order to compute these features very rapidly at many scales we introduce the ***integral image*** *representation for images”*

“Once computed, any one of these Haar-like features ***can be computed at any scale*** *or location in constant time.”*

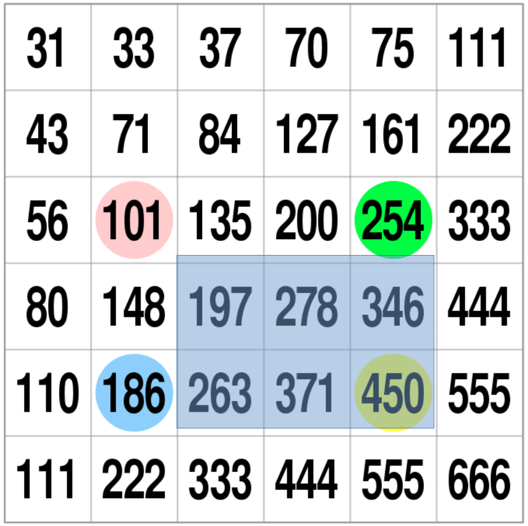
Solution to feature calculation

1.



31	2	4	33	5	36
12	26	9	10	29	25
13	17	21	22	20	18
24	23	15	16	14	19
30	8	28	27	11	7
1	35	34	3	32	6

2.

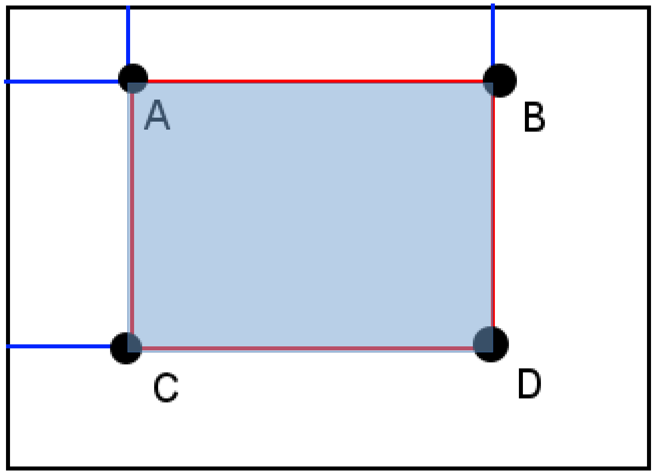


31	33	37	70	75	111
43	71	84	127	161	222
56	101	135	200	254	333
80	148	197	278	346	444
110	186	263	371	450	555
111	222	333	444	555	666

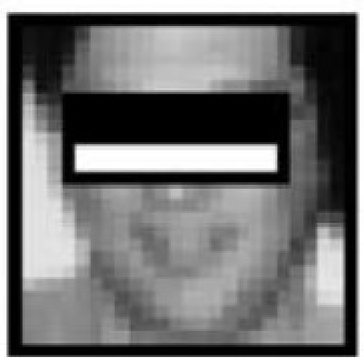
15 + 16 + 14 + 28 + 27 + 11 =
101 + 450 – 254 – 186 = 111

Only four
values needed

Works on any scale!

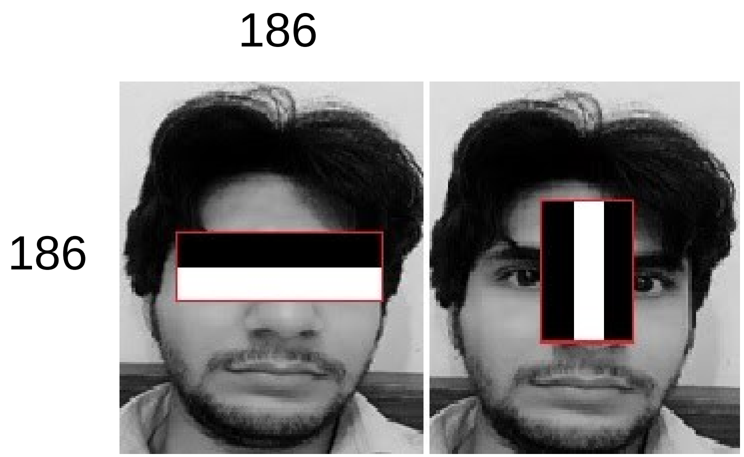


Sum = D - B - C + A



Only six values needed!

Solution to feature scaling



We can compute the results from the features at any scale, without scaling the original image.

“Like most face detection systems, our detector scans the input at many scales; starting at the base scale in which faces are detected at a size of 24×24 pixels, a 384 by 288 pixel image is scanned at 12 scales each a factor of 1.25 larger than the last. The conventional approach is to compute a pyramid of 12 images, each 1.25 times smaller than the previous image. A fixed scale detector is then scanned across each of these images. ***Computation of the pyramid, while straightforward, requires significant time.*** *Implemented efficiently on conventional hardware (using bi-linear interpolation to scale each level of the pyramid) it takes around .05 seconds to compute a 12 level pyramid of this size (on an Intel PIII 700 MHz processor).”*

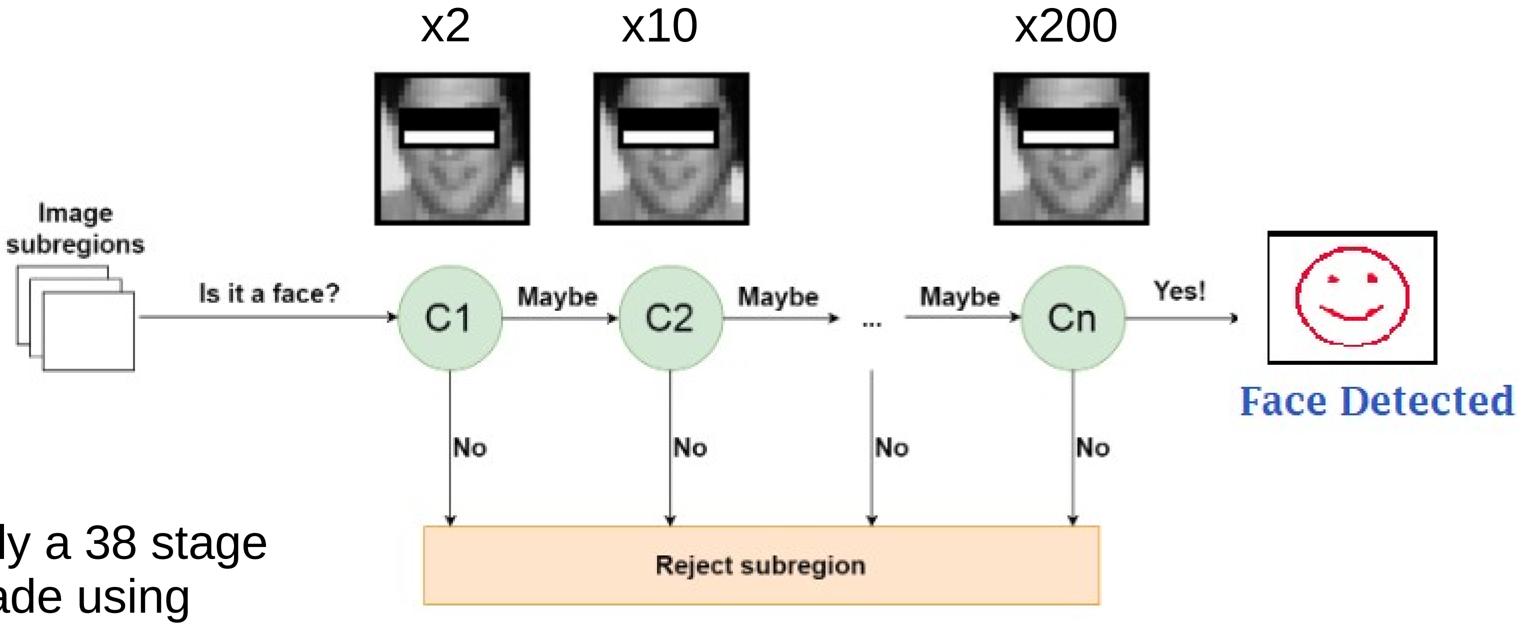
They scale the features instead if the image!

“The second contribution of this paper is a simple and efficient classifier that is built by selecting a small number of important features from a huge library of potential features using AdaBoost”

“The face training set consisted of 4916 hand labeled faces scaled and aligned to a base resolution of 24 by 24 pixels.”

How to select the features is part of the training process, which we won't talk about today.

“The third major contribution of this paper is a method for combining ***successively more complex classifiers*** *in a cascade structure which dramatically increases the speed of the detector by focusing attention on promising regions of the image.”*



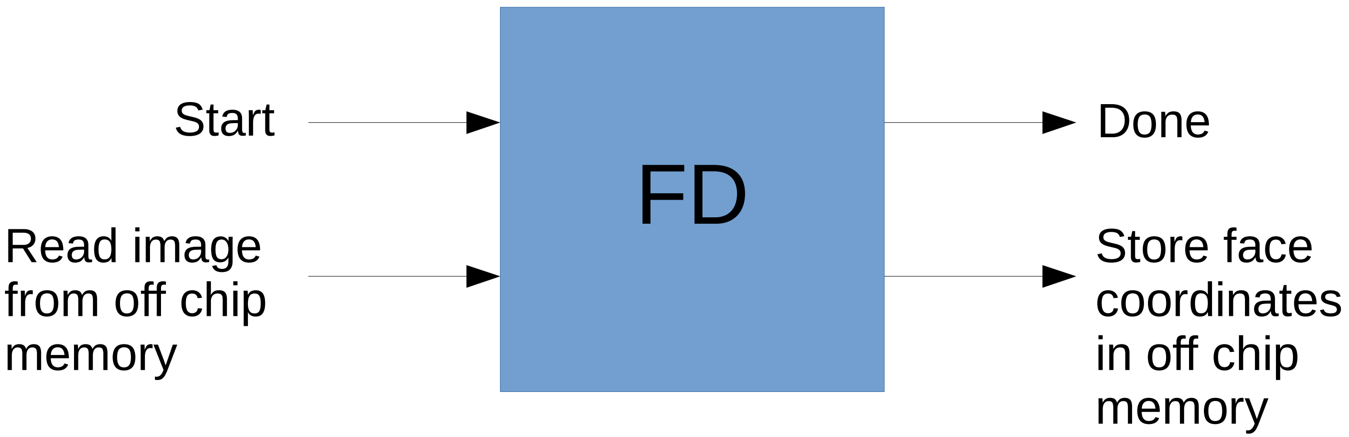
Initially a 38 stage cascade using 6060 features.

To summarize:

- Focus is on **speed** → real time applications
- **Integral image** → utilized the to speed up calculation of basic features
- A classifier with **few features** → easy to train, limited amount of data needed
- **No image scaling** → scale the features instead (to detect large faces)
- The **cascade** → discard none face areas as soon as possible (less compute → better performance)

Ordered requirements were:

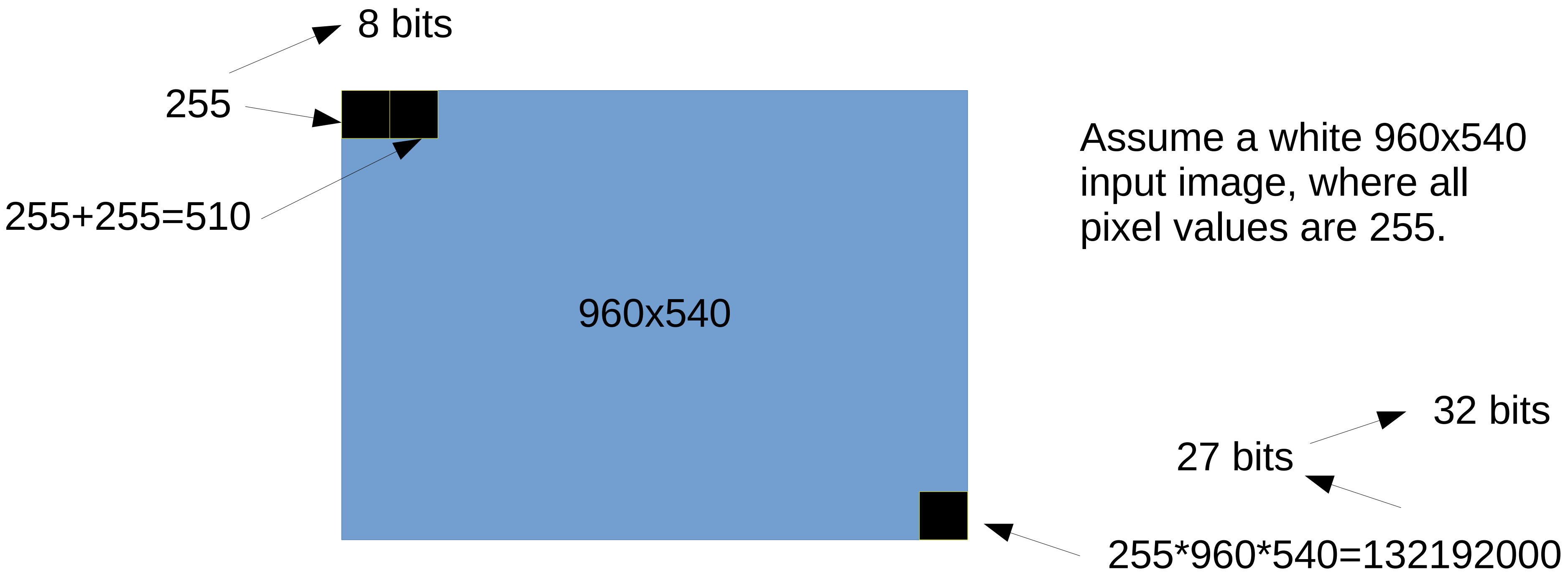
- Implement (something similar to) the VJ-algorithm
- Image in → face coordinates out
- Input image size 1080p/2 (960x540) pixels
- Detect faces down to 20x20 pixels
- “No limit” in number of faces detected
- Speed >20 FPS


Start
Read image from off chip memory
FD
Done
Store face coordinates in off chip memory

Chip-team requirements/constraints:

- Your module (FD) is only a small part of the SoC → resources are shared
- External memory bandwidth is shared → minimize your usage
- External memory is DDR → don't use random access, always try to use full bursts
- Save area → area is money (and static power)
- Minimize (dynamic) power → simplify algorithm and use minimal data types
- RTL correctness → everything must be verified using a reference model flow
- Predefined clocks → only a few to pick from 100 and/or 200 MHz
- All work; design, implementation and verification must finish in time → use a known work methodology

Integral image (/IM) observations (1/2)

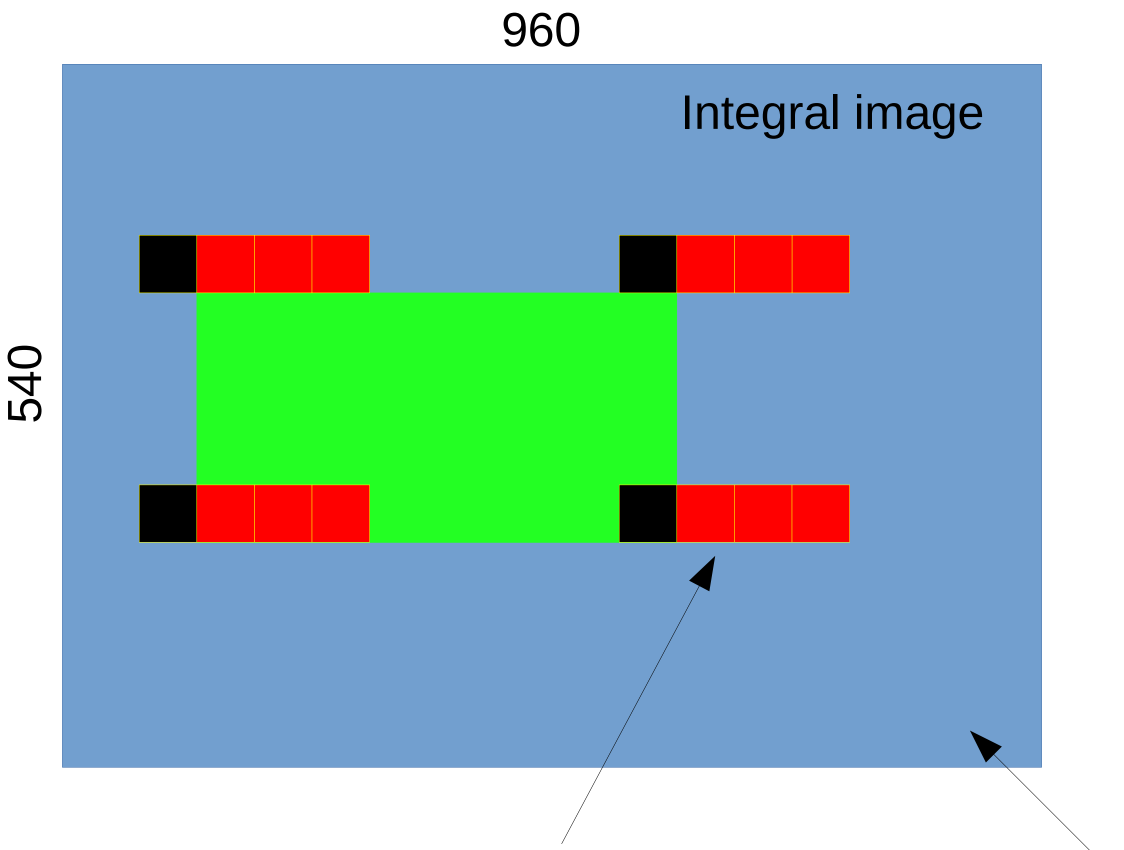


Assume a white 960x540 input image, where all pixel values are 255.

Our 8 bit gray scale image will be represented by a 32 bit integral image.

The “size” of the image will increase 4 times!

Integral image (/IM) observations (2/2)



960

540

Integral image

Memory read burst

32*960*540/8 ~= 2MBytes

⬛ Data we need

🟥 Read but not needed

Random memory accesses without a cache will be very bad. We can't cache a whole integral image.

Memory bandwidth will be wasted → not good and not allowed!

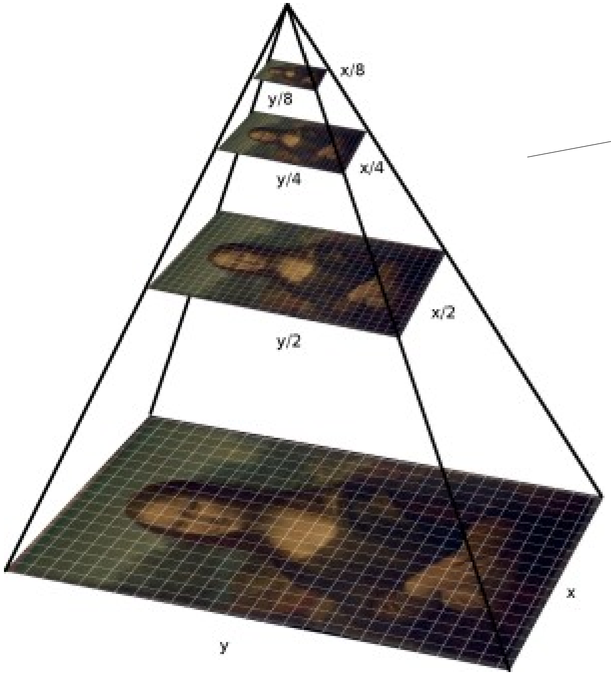
Integral image conclusions:

- Most of the invention was based on the use of an integral image.
- But now we concluded that this isn't such a good idea (small random DDR memory reads) from a system perspective.
- What to do, give up?
- Or can we “keep it” and at the same time avoid it?

Lets go back and look at one of the motivators for the integral image...

A pyramid of images at different scales

*“Computation of the **pyramid**,while straightforward, **requires significant time**. Implemented efficiently on conventional hardware (using bi-linear interpolation to scale each level of the pyramid) it takes around .05 seconds to compute a 12 level pyramid of this size (on an Intel PIII 700 MHz processor).”*



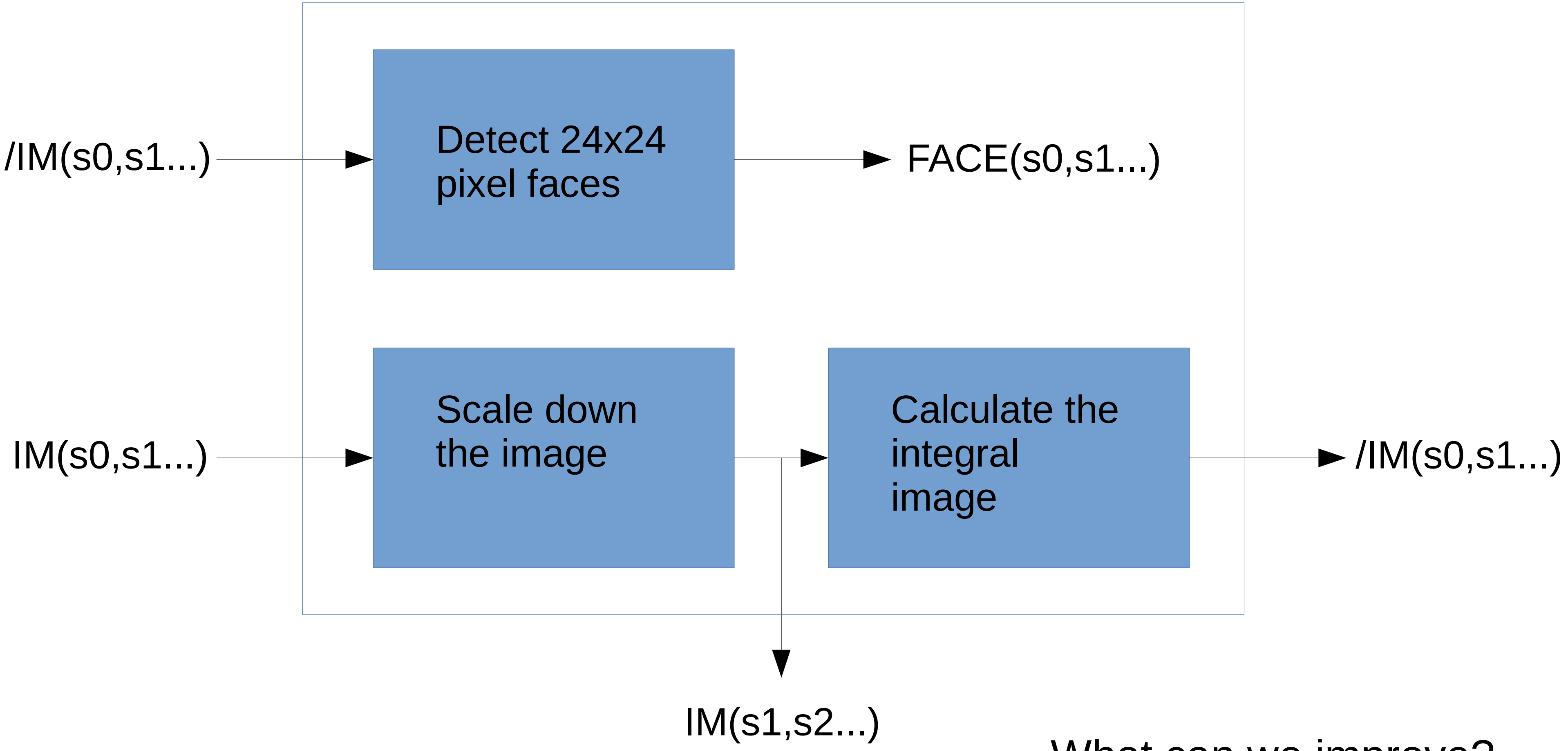
Convert these to integral images and then scan for fixed size 24x24 pixel faces.

All data can be read linearly from DDR memory using full bursts.

Can we fix this scaling performance bottleneck?

What do you think of this solution?

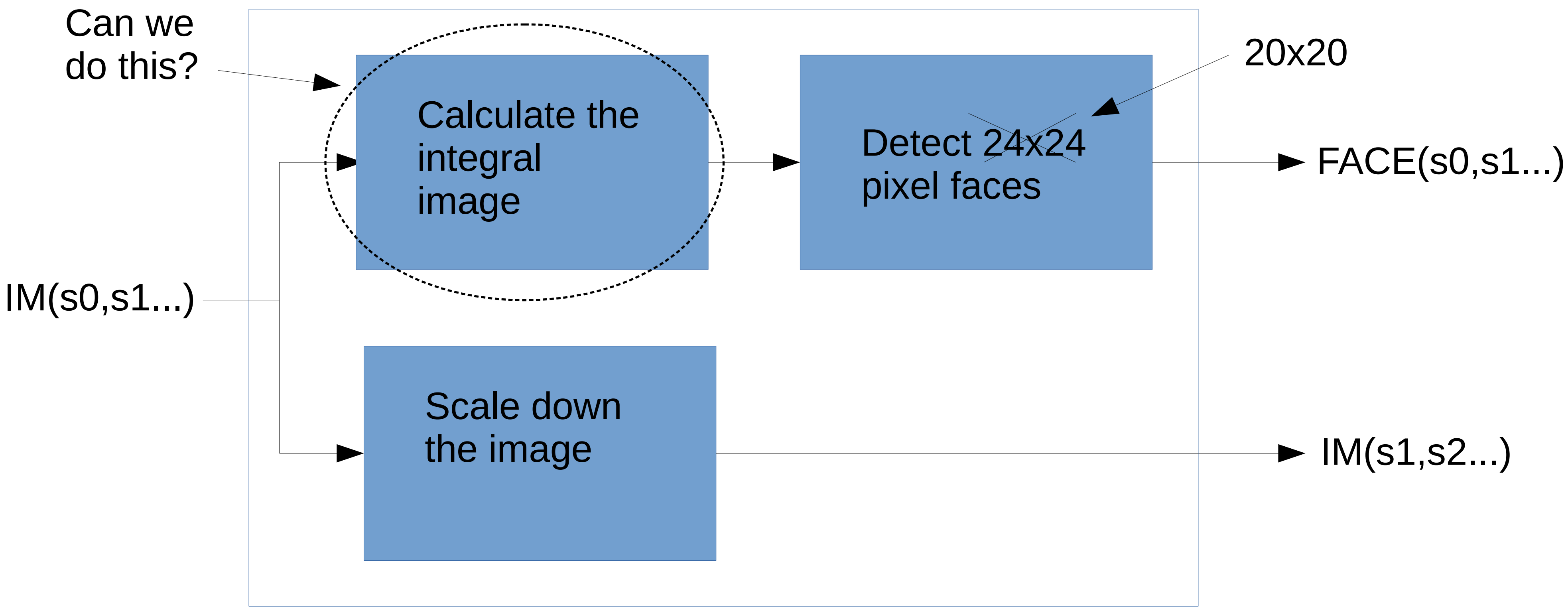
Detecting and scaling runs in parallel and we only write and read linearly in memory.



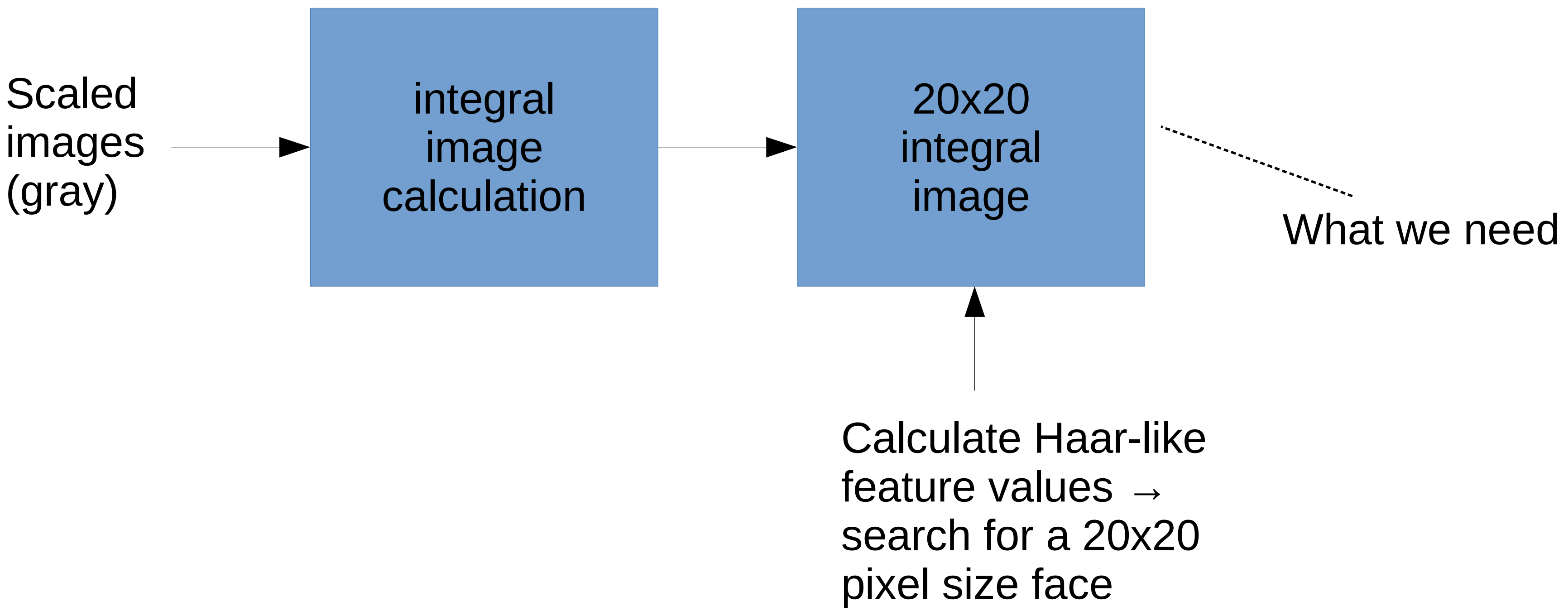
What can we improve?

This would be a very nice solution

Detecting and scaling runs in parallel and we only write and read linearly in memory. The integral images are “gone” and we only need to read 8 bit data instead of 32 bit integral images.



This is what we will aim for. Now it is time to think about implementation optimizations!



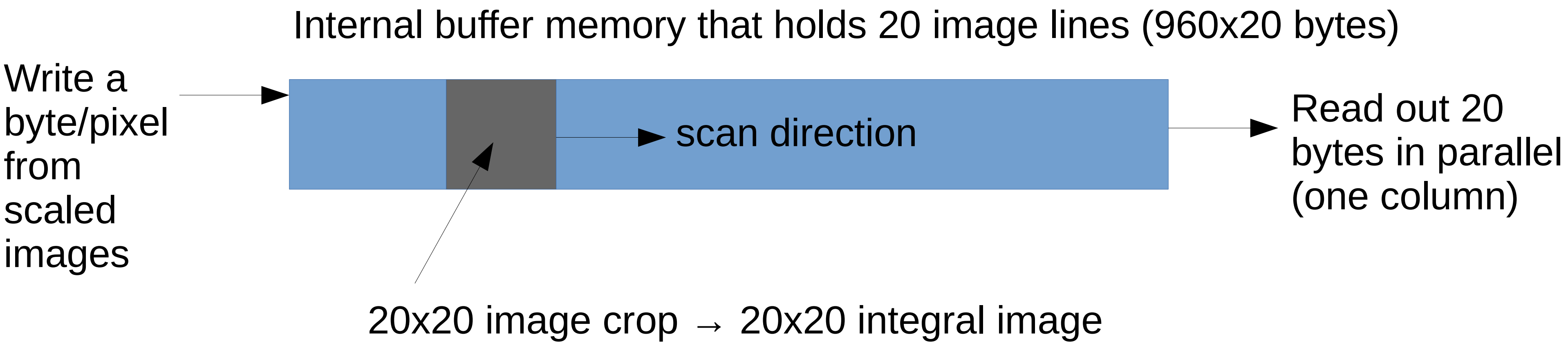
Scaled images (gray)

integral image calculation

20x20 integral image

What we need

Calculate Haar-like feature values → search for a 20x20 pixel size face



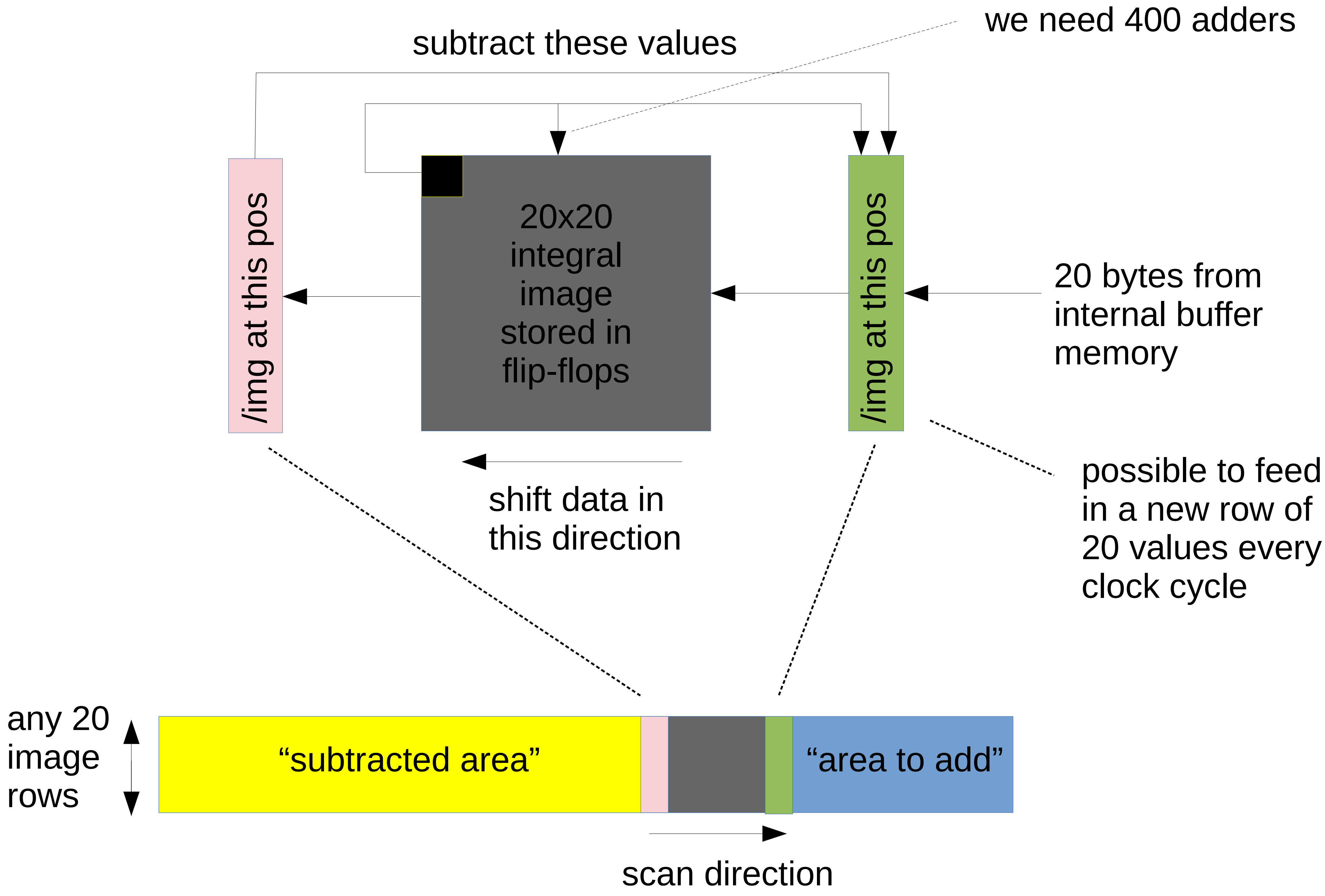
Internal buffer memory that holds 20 image lines (960x20 bytes)

Write a byte/pixel from scaled images

scan direction

Read out 20 bytes in parallel (one column)

20x20 image crop → 20x20 integral image



we need 400 adders

subtract these values

/img at this pos

20x20 integral image stored in flip-flops

/img at this pos

20 bytes from internal buffer memory

shift data in this direction

possible to feed in a new row of 20 values every clock cycle

any 20 image rows

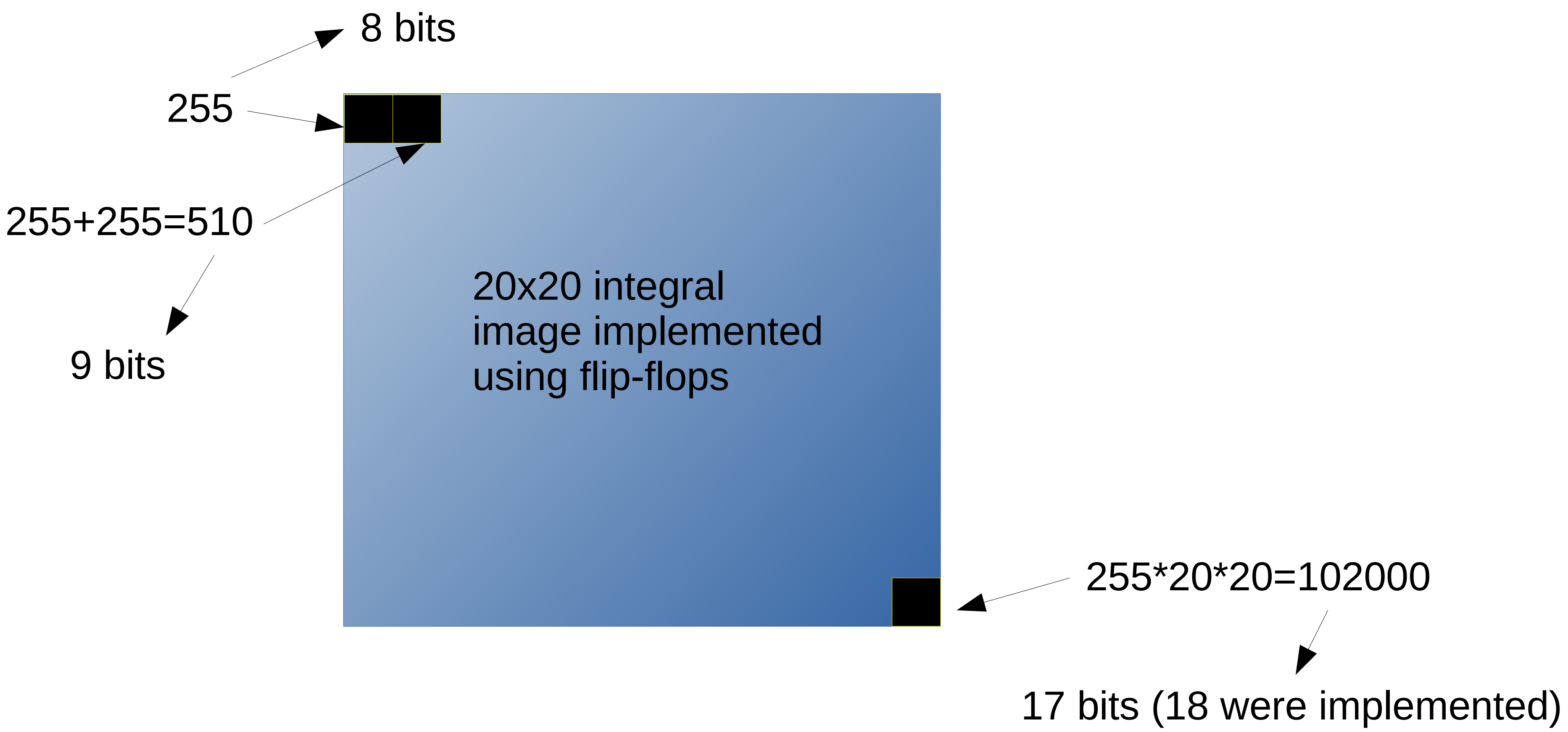
“subtracted area”

“area to add”

scan direction

The “pixel values” of the integral image will never grow!

A normalized (optimal) integral image is calculated for each position



8 bits

255

255+255=510

9 bits

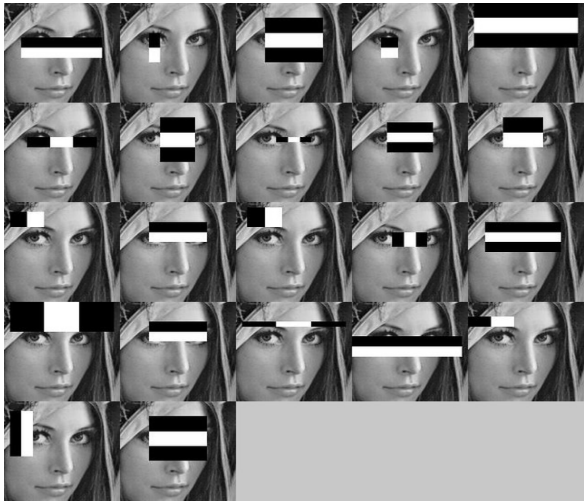
20x20 integral image implemented using flip-flops

255*20*20=102000

17 bits (18 were implemented)

We can use different data types (# of bits) for each coordinate → this will save area and power!

3 features in the first stage

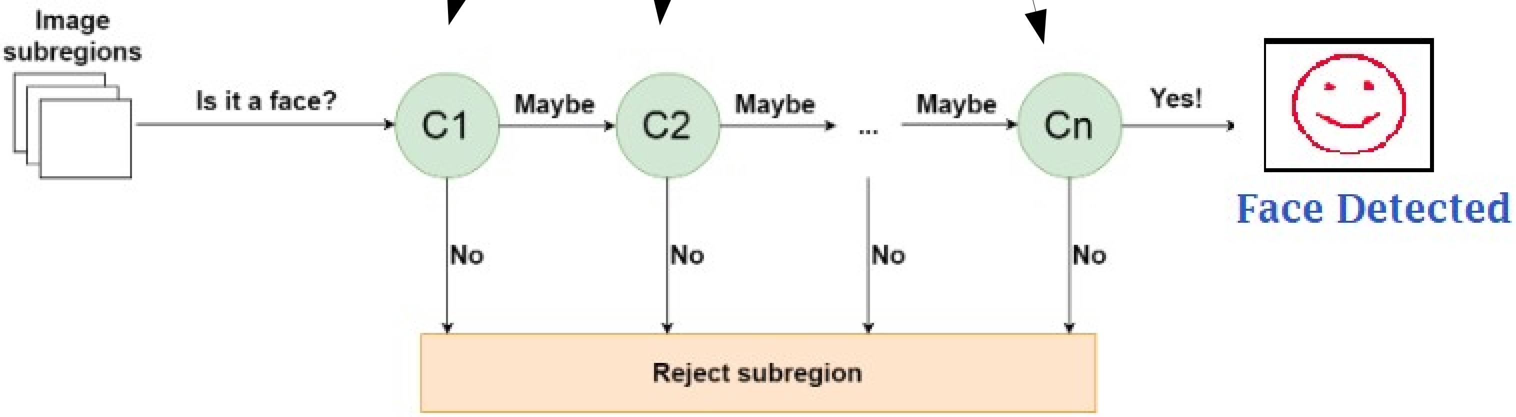


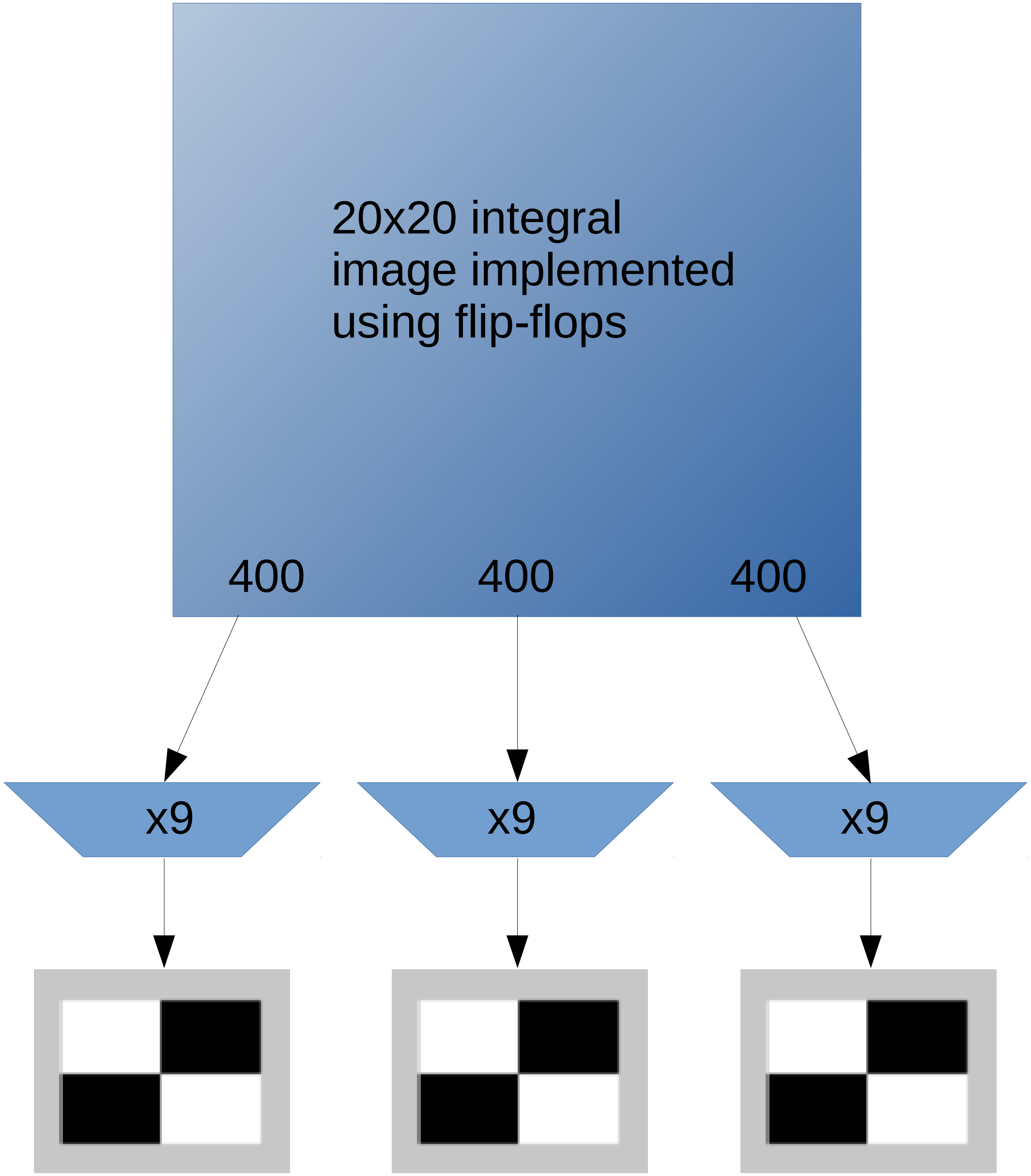
It’s time for some parallel computing→ lets do 3 features in one clock cycle!

x3

x10

x200





- 3 features in parallel
- 9 values to read per feature
- 18 bit values in /IM
- 400 values to select from

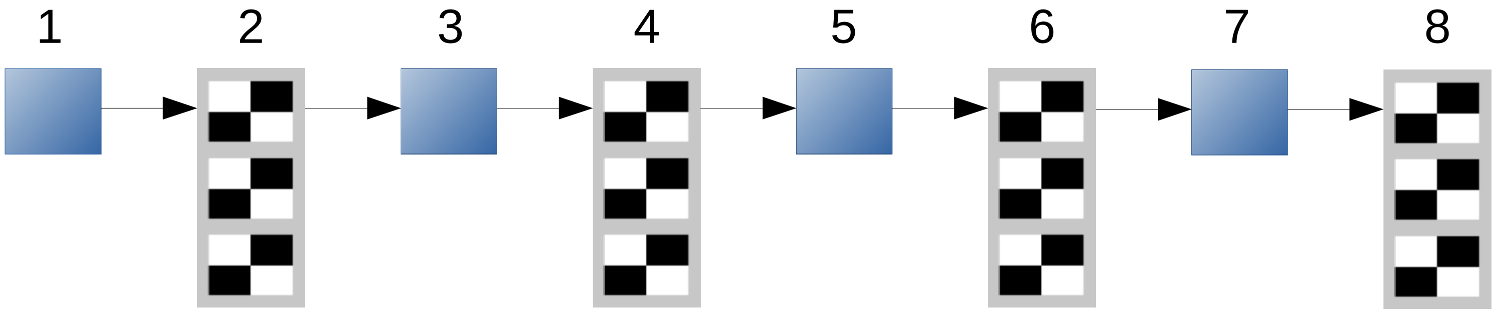
→ 27 multiplexes with 400 inputs selecting 18 bit values

Great performance but a lot of wires!

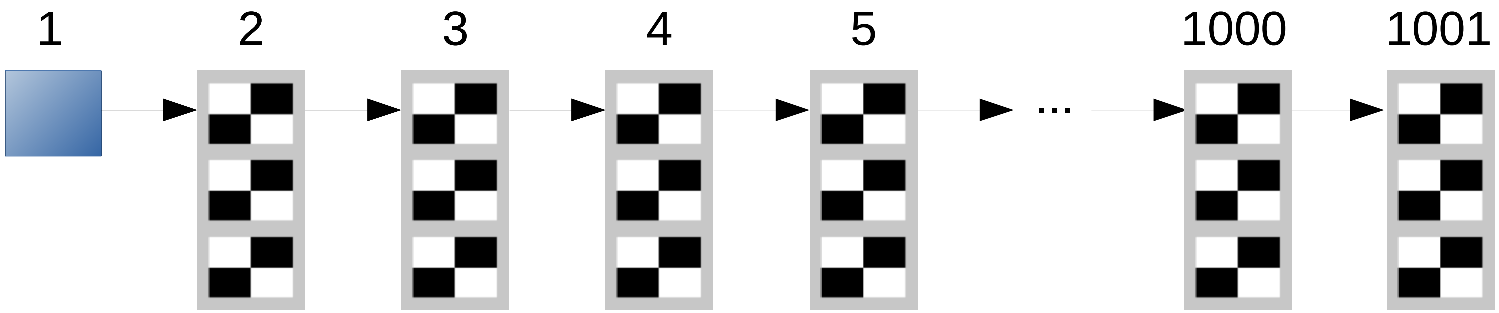
↓

We can expect layout utilization to be low.

Our implementation will have the following performance



Best case, 1 sub-image every second clock cycle → no face

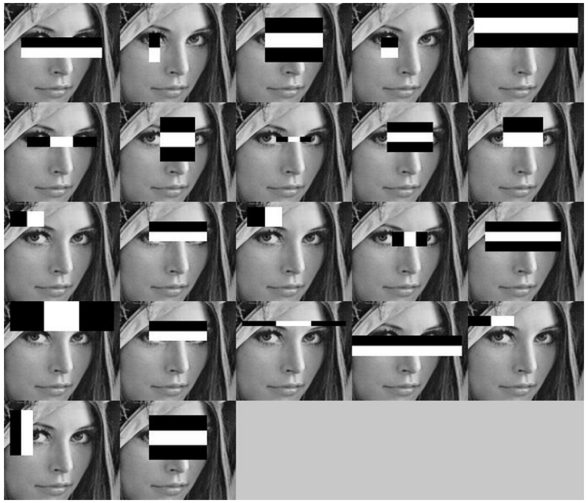


Worst case, 1 sub-image approx. every 1000 clock cycle → face found

(if ~3000 features needed to identify a face)

Average performance will depend on number of face-like structures in the image.

About the feature database



The feature data base needs to be stored in local on-chip RAM where we can fetch parameters for three features at the same time.

All features were trained off line. We had to limit the feature database (cascade depth and feature count) to be able to store it on chip.

We invented our own compressed format to describe and store the features.

Pros and cons list

	Software/CPU	Hardware/ASIC
Performance (speed)	+	+++
Performance (accuracy)	+	-
Power	--	++
Bandwidth	-	+
Area	-	+
Flexibility	++	--
Implementation effort	+++	---
Cost	+	-
Sum	8+ 4-	7+ 7-

Conclusion: A custom hardware implementation is possible and gives a significant performance boost, but this algorithm is probably better suited to be run on a standard CPU.

Future of AI acceleration

My personal thoughts on what to spend research resources on

LiTH Linköping
February 24th 2020

Anders Lloyd
Axis Communications

Hardware for Machine Learning
PhD Course Jan-Mar 2020

Convolutional Neural Networks

CNNs are dominating today. They are relatively easy to understand and we can train both weights and network architectures. Base data type has come down to 8 bits and acceleration has moved from CPU/GPU to convolution engines. This is all good but...

The base operation (convolution) requires a lot of parallel multipliers and multipliers are probably one of the **worst power consumers on silicon**. So...

For CNN:s we should aim for simplified implementations of our multipliers. One solution to explore is all research already done in the filed of **approximate computing** with various multiplier implementations. Small compute errors can easily be handled in the training process.

A more **long term solution** for “CNNs” compute problem would be to get rid of the convolutions and use **other operations** to find correlations between data points. New inventions here could also help reduce the number of weights needed.

In all honesty, research on CNNs is like research on diesel engines for cars. I would strongly recommend you **spend your efforts on other topics**. The industry (and so many other graduate students) can take care of things from now on.

Spiking Neural Networks

*“Spiking neural networks (SNNs) are artificial neural networks that **more closely mimic natural neural networks**.”*

*“A spiking neural network **considers temporal information**. The idea is that not all neurons are activated in every iteration of propagation (as is the case in a typical multilayer perceptron network), but only when its membrane potential reaches a certain value. When a neuron is activated, it produces a signal that is passed to connected neurons, raising or lowering their membrane potential.”*

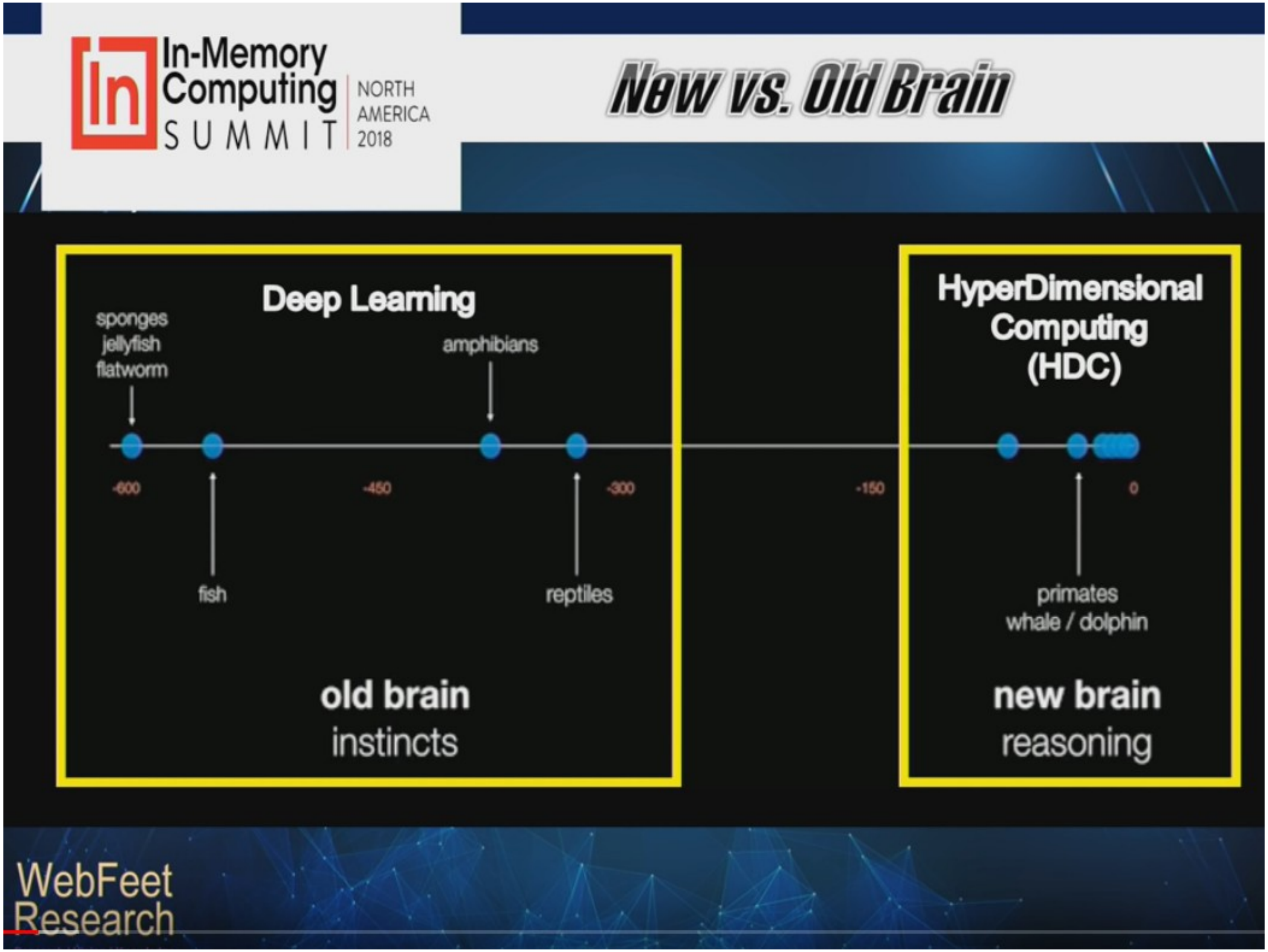
*“The SNN approach uses a **binary output** (signal/no signal) instead of the continuous output of traditional ANNs.”*

These are definitely promising for the future. Good hardware acceleration will be key to their success. The **base operation is addition** and not multiplication, and the add operations are event driven and only executed when needed (compared to CNNs where you just compute without thinking…).

Research on SNN acceleration **will not be wasted**.

Hyper Dimensional Computing and In-Memory Compute

From: https://www.youtube.com/watch?v=AkrseddrZpA



In-Memory Computing SUMMIT NORTH AMERICA 2018

New vs. Old Brain

Deep Learning

sponges
jellyfish
flatworm

amphibians

-600 -450 -300 -150 0

fish

reptiles

old brain
instincts

HyperDimensional Computing (HDC)

primates
whale / dolphin

new brain
reasoning

WebFeet Research

Hyper Dimensional Computing and In-Memory Compute

From: https://web.stanford.edu/class/ee380/Abstracts/171025-slides.pdf

WHAT IS HIGH-DIMENSIONAL (HD) COMPUTING?

It is a system of computing that operates on high-dimensional **vectors**.

- The algorithms are based on operations on vectors

Traditional computing operates on **bits** and **numbers**.

- Built-in circuits for arithmetic and for Boolean logic

Watch this presentation:

Stanford Seminar - Computing with High-Dimensional Vectors (**Pentti Kanerva**)

Link: https://www.youtube.com/watch?v=zUCoxhExe0o

I think HD computing could be the next big thing. Research is ongoing but there are a lot of things to explore and improve in this area. This is for sure **my number one candidate** for research efforts.

Thank you

Questions?