Deep Learning In OpenCV

Wu Zhiwen

zhiwen.wu@intel.com

Agenda

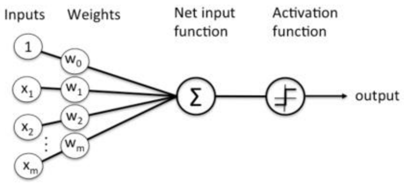
- Background information
- OpenCV DNN module
- OpenCL acceleration
- Vulkan backend
- Sample

What is OpenCV?

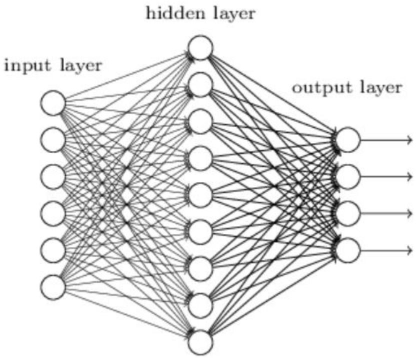
- Open Source Compute Vision (OpenCV) library
- 2500+ Optimized algorithms for compute vision and machine learning
- C/C++/Python compatible and cross platform
- 20000+ forks in GitHub
- OpenCV 4.0 is on the way
  - Switch to C++ 11
  - No longer binary-compatibility
  - Better performance on CPU (AVX2)
  - Compact footprint
  - Big revision of DNN module

Key concepts of Deep Neural Networks (DNN)

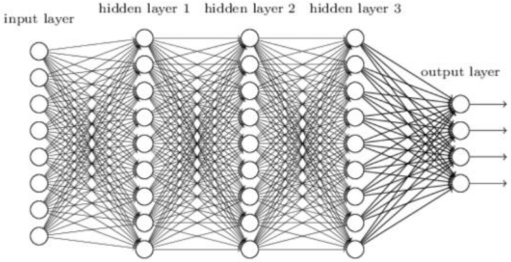
- Node/Layer/Network/Deep Neural Networks



Node/Neuron/Perceptron



Layer



Deep Neural Network

Key concepts of Deep Neural Networks (DNN)

- Training

step1: initialize weigths

step2: set input data (e.g. a image) and compute the output

step3: compare the output and the ground truth and calculate the error

step4: modify the weights and go to step 2 until the error is small enough

Complicated?
Deep Learning Frameworks will do that for you

TensorFlow Caffe torch

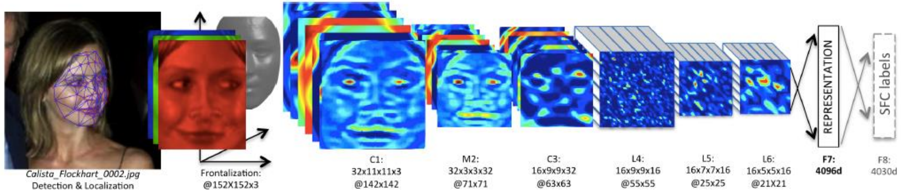
Key concepts of Deep Neural Networks (DNN)

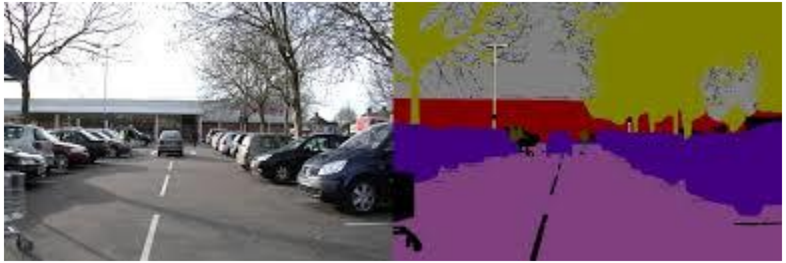
- Inference/forward/predict

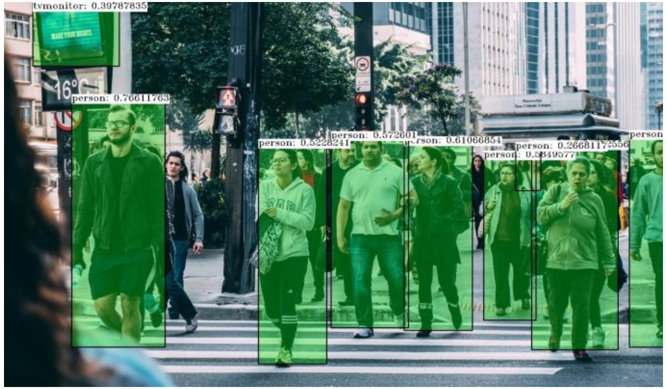
  You have a trained model, i.e. weights and other parameters.

  Set input data and use Deep Learning Framework to compute the output.

- Use case







OpenCV DNN module

- Included in main OpenCV repo since version 3.3
- Inference only
- Compatible to many popular Deep Learning frameworks

TensorFlow

Caffe

torch

Darknet

OpenCV DNN module

Why we need a new wheel of DNN in OpenCV?

- Lightness

  - inference only can simply the code, speed up the installation and compilation process

- Convenience

  - build-in implementation, minimum external dependency

  - easy to add deep networks support to your existed OpenCV project

- Universality

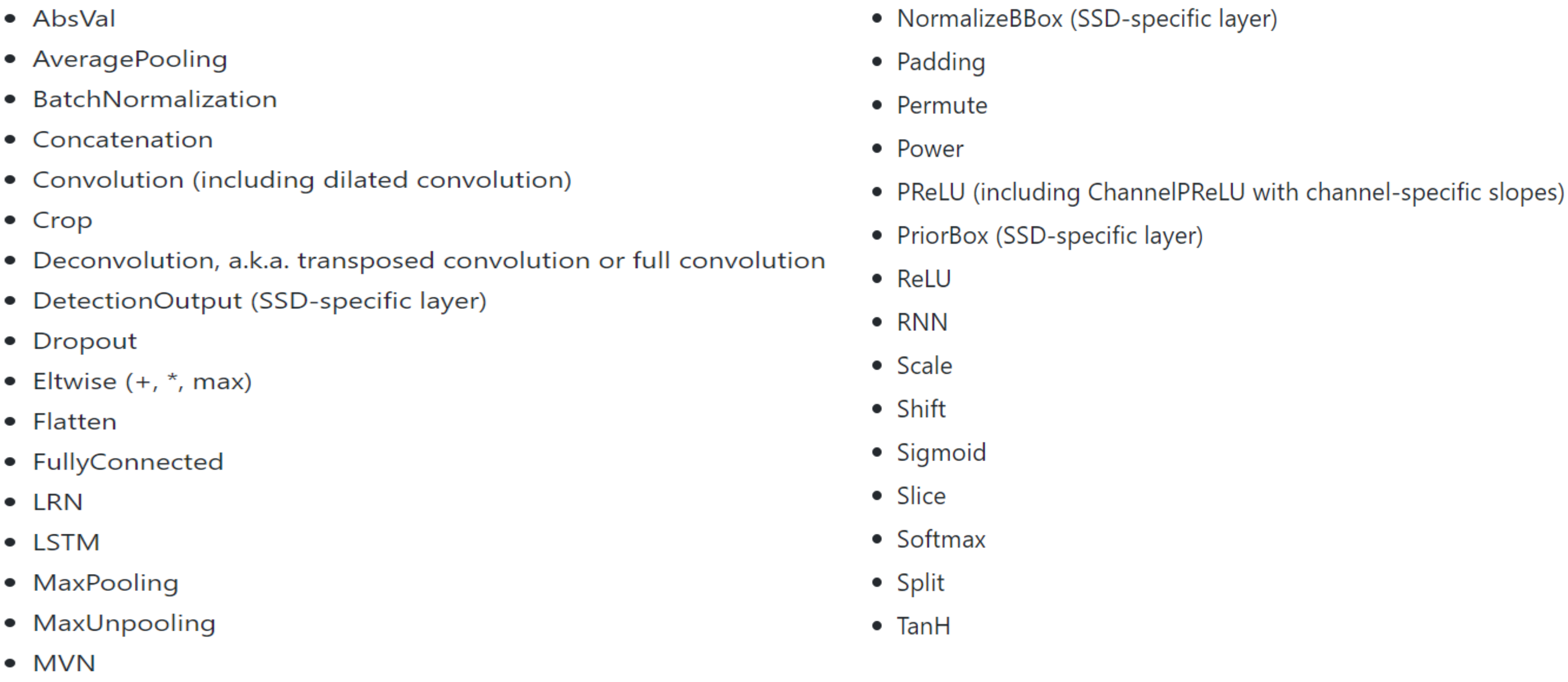
  - Unified interface to manipulate net models

  - Support multiple target device and OS

    Device: CPU, GPU, VPU OS: Linux, Windows, Android, MacOS

OpenCV DNN module

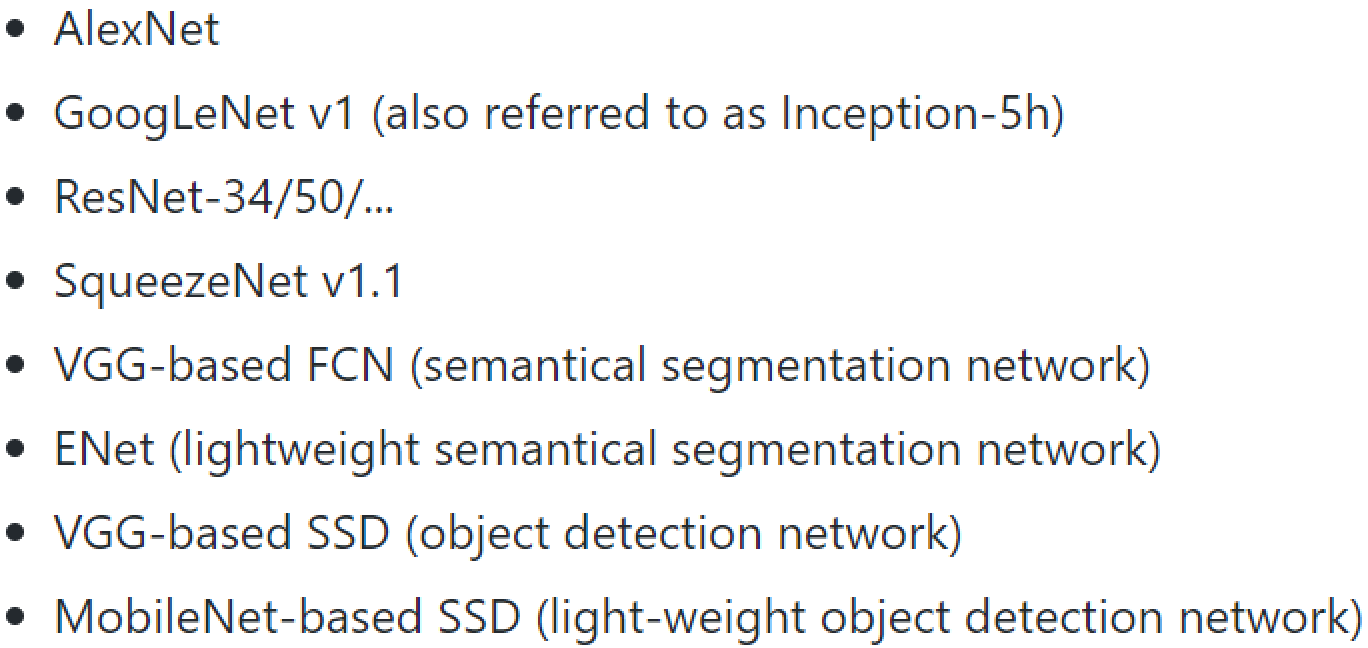
- Support ~40 layer types



- AbsVal
- AveragePooling
- BatchNormalization
- Concatenation
- Convolution (including dilated convolution)
- Crop
- Deconvolution, a.k.a. transposed convolution or full convolution
- DetectionOutput (SSD-specific layer)
- Dropout
- Eltwise (+, *, max)
- Flatten
- FullyConnected
- LRN
- LSTM
- MaxPooling
- MaxUnpooling
- MVN
- NormalizeBBox (SSD-specific layer)
- Padding
- Permute
- Power
- PReLU (including ChannelPReLU with channel-specific slopes)
- PriorBox (SSD-specific layer)
- ReLU
- RNN
- Scale
- Shift
- Sigmoid
- Slice
- Softmax
- Split
- TanH

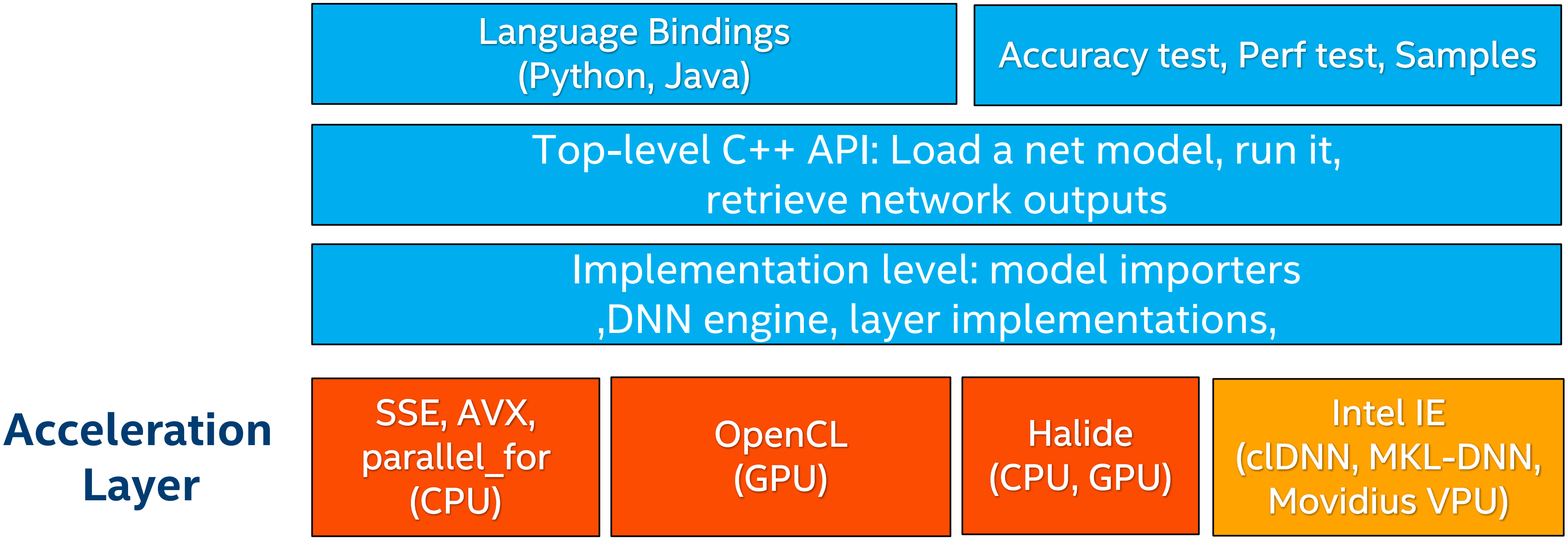
OpenCV DNN module

- Network well tested



- AlexNet
- GoogLeNet v1 (also referred to as Inception-5h)
- ResNet-34/50/...
- SqueezeNet v1.1
- VGG-based FCN (semantical segmentation network)
- ENet (lightweight semantical segmentation network)
- VGG-based SSD (object detection network)
- MobileNet-based SSD (light-weight object detection network)

Architecture of DNN module



Language Bindings
(Python, Java)

Accuracy test, Perf test, Samples

Top-level C++ API: Load a net model, run it, retrieve network outputs

Implementation level: model importers ,DNN engine, layer implementations,

Acceleration Layer

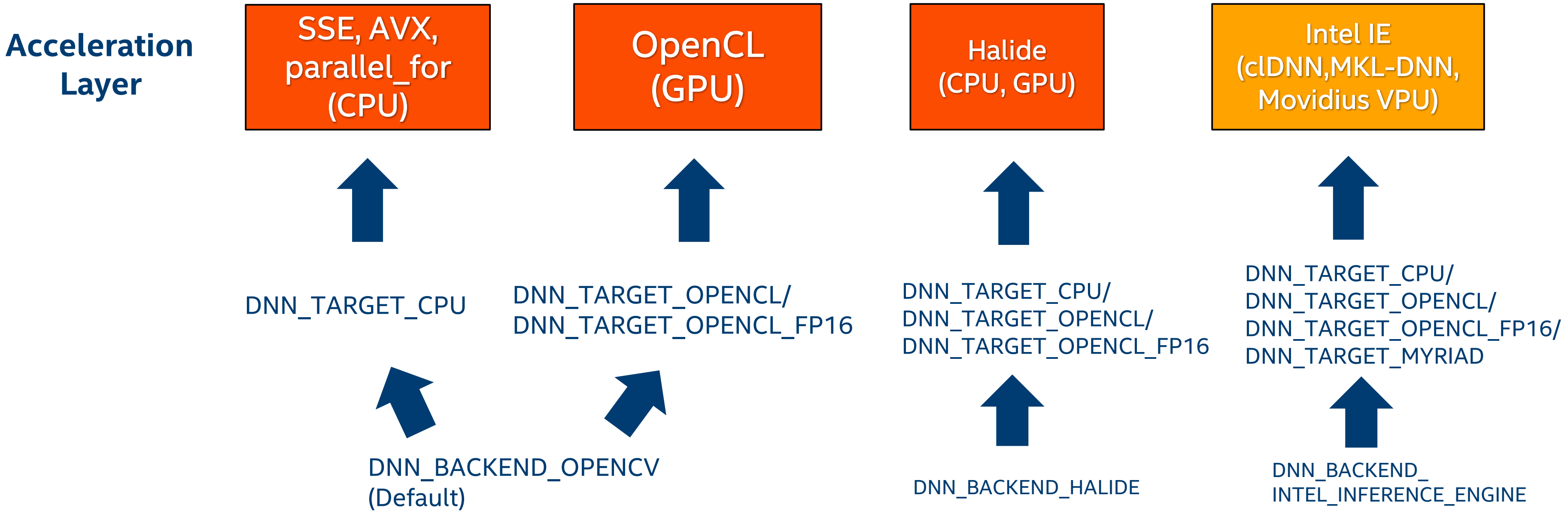
SSE, AVX, parallel_for (CPU)

OpenCL (GPU)

Halide (CPU, GPU)

Intel IE (clDNN, MKL-DNN, Movidius VPU)

Backend and target



Acceleration Layer

SSE, AVX, parallel_for (CPU)

OpenCL (GPU)

Halide (CPU, GPU)

Intel IE (clDNN,MKL-DNN, Movidius VPU)

DNN_TARGET_CPU

DNN_TARGET_OPENCL/
DNN_TARGET_OPENCL_FP16

DNN_TARGET_CPU/
DNN_TARGET_OPENCL/
DNN_TARGET_OPENCL_FP16

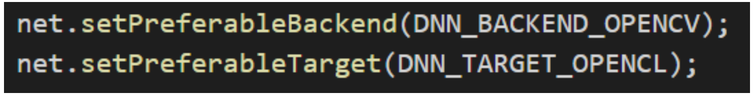
DNN_TARGET_CPU/
DNN_TARGET_OPENCL/
DNN_TARGET_OPENCL_FP16/
DNN_TARGET_MYRIAD

DNN_BACKEND_OPENCV
(Default)

DNN_BACKEND_HALIDE

DNN_BACKEND_
INTEL_INFERENCE_ENGINE

E.g. use the build-in GPU acceleration



```
net.setPreferableBackend(DNN_BACKEND_OPENCV);
net.setPreferableTarget(DNN_TARGET_OPENCL);
```

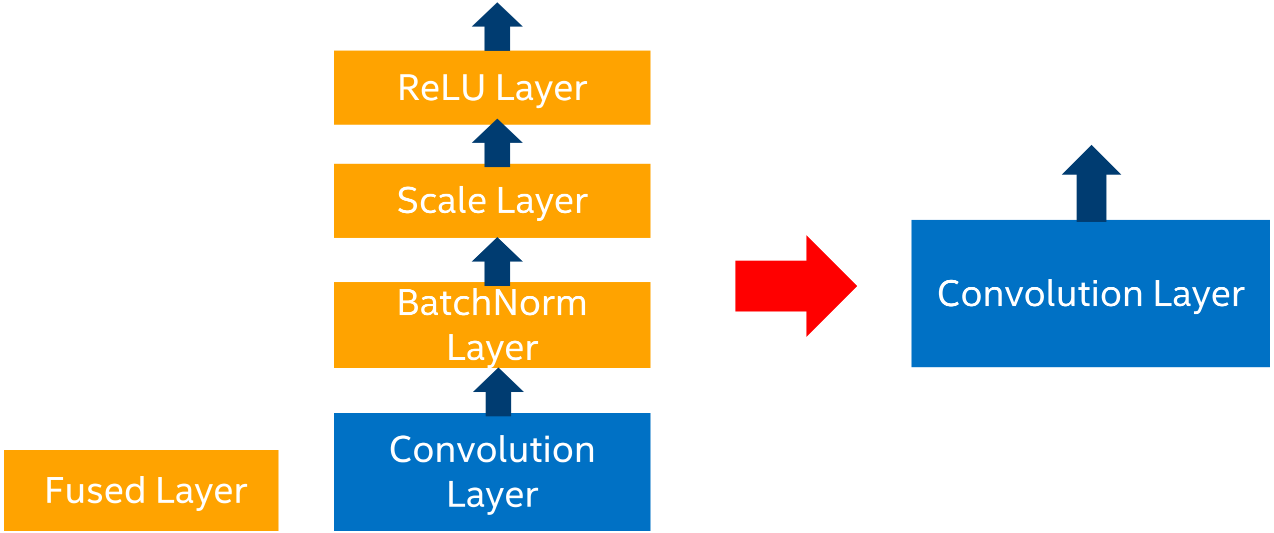
Network optimizations

- DNN module implemented its own framework internally, these optimizations are not tied to any specific Deep Learning Frameworks.

- Benefit all the net models no matter what their original framework is.

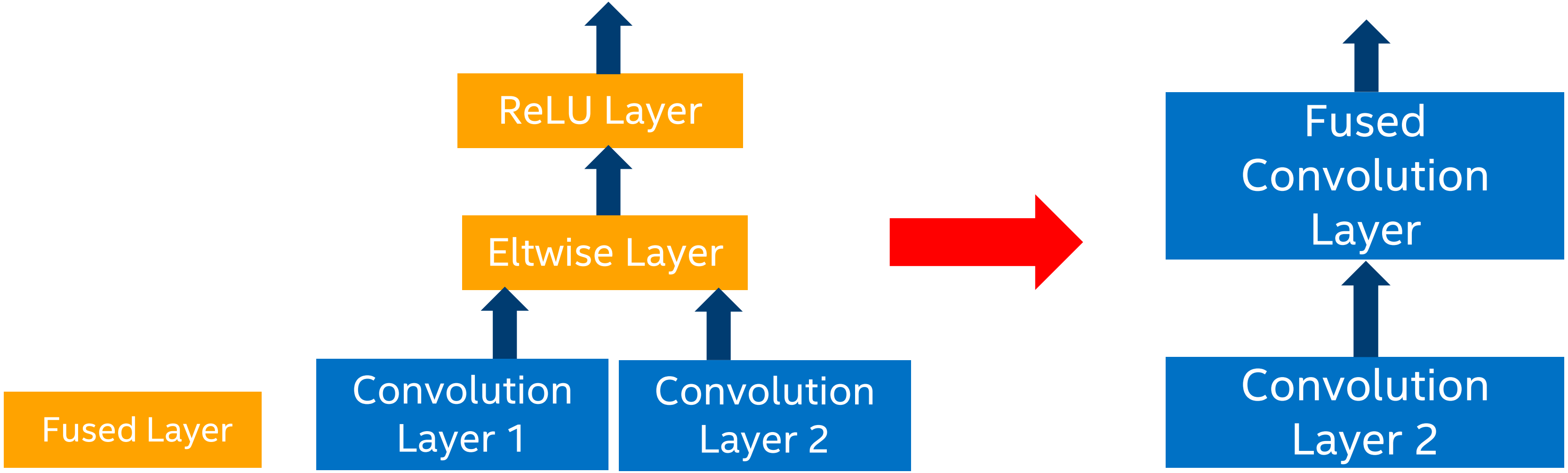
Layer Fusion

DNN module analysis network structure and, if possible, merge some layers into another one.



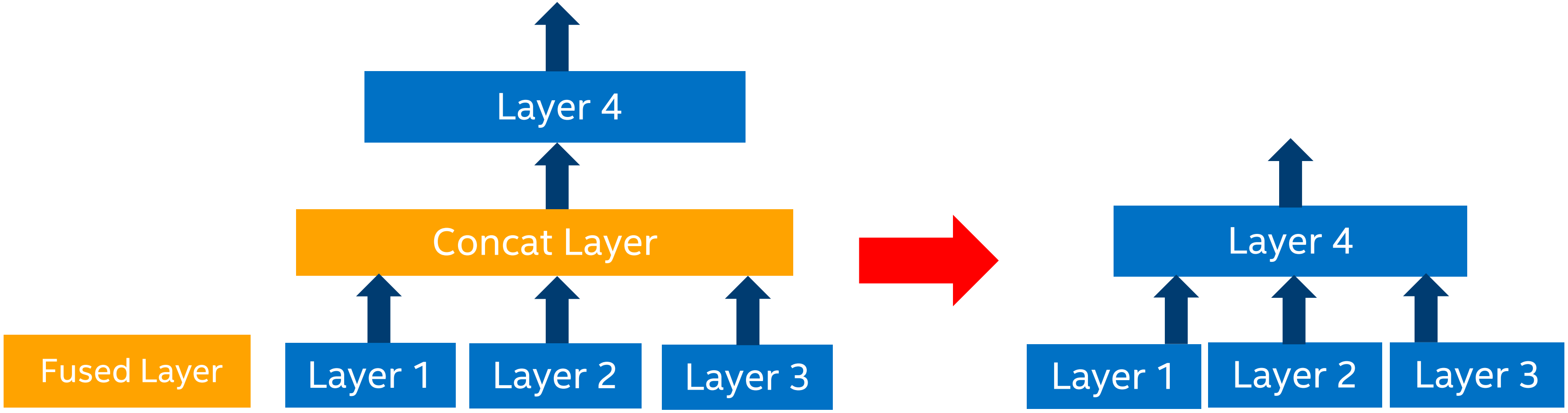
structure in ResNet50

Layer Fusion



structure in ResNet50

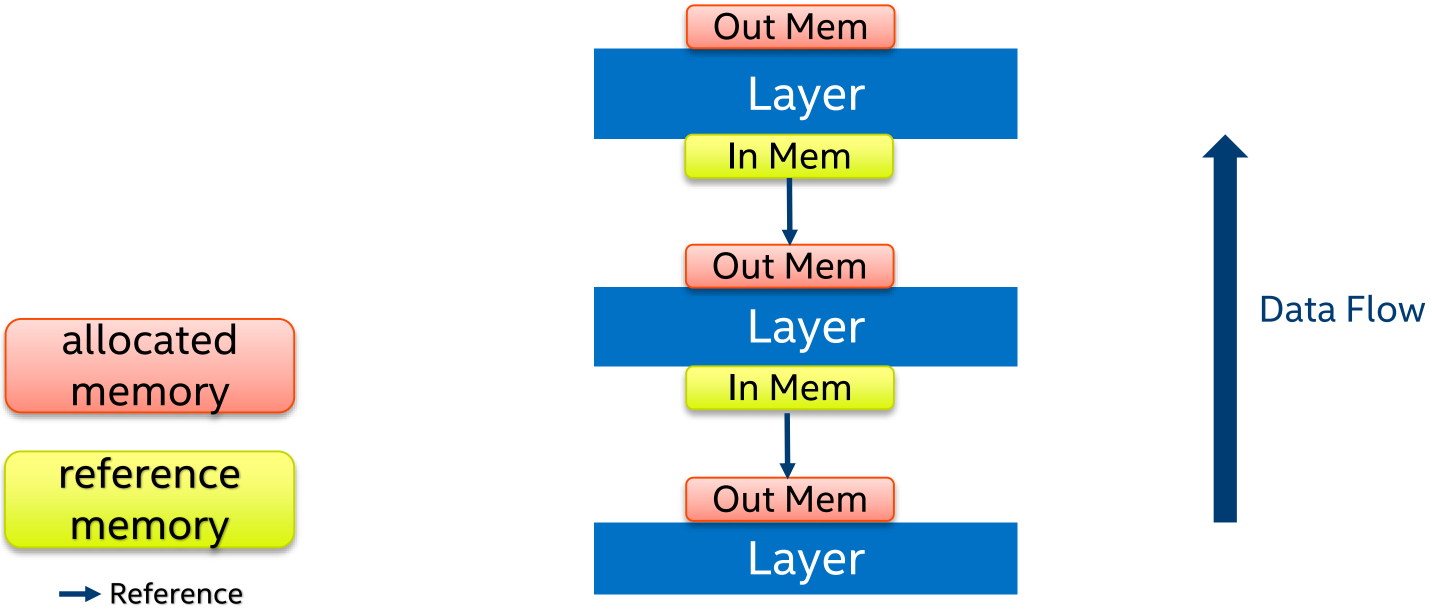
Layer Fusion



structure in SSD

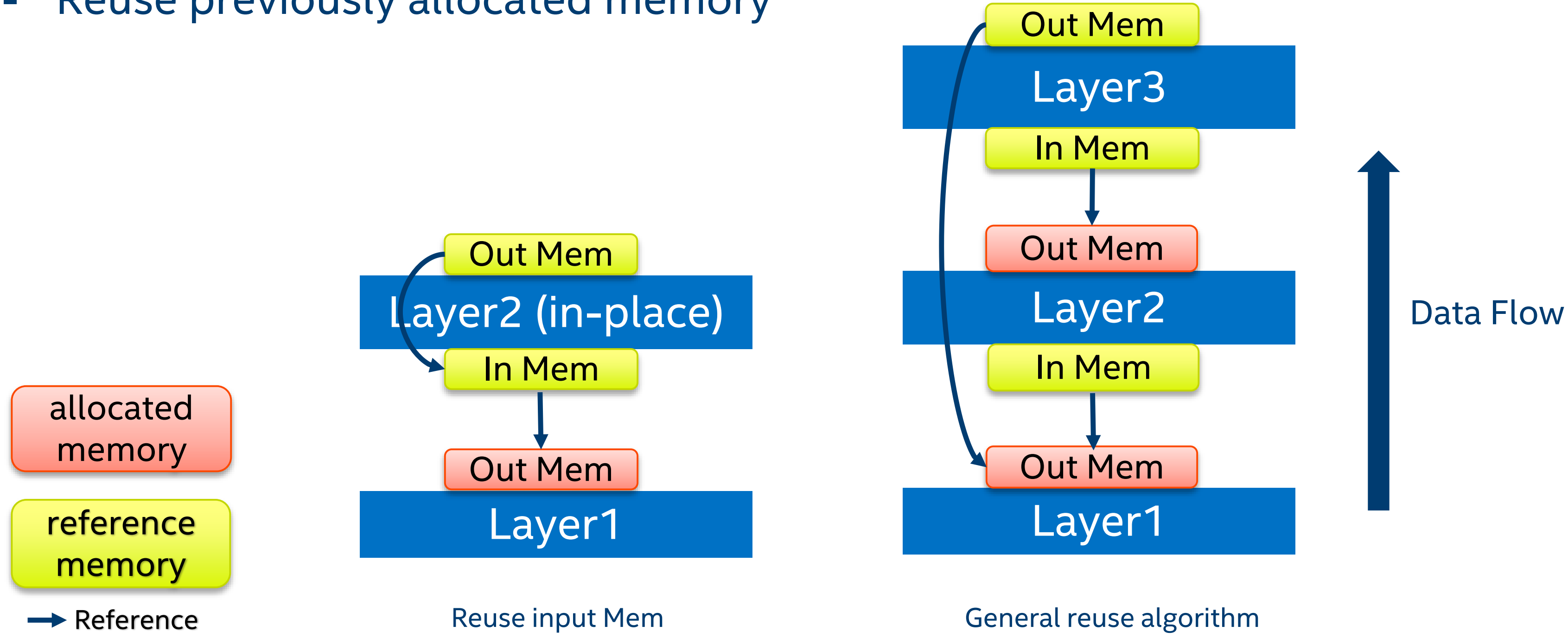
Memory reuse

- memory usage without reuse



Memory reuse

- Reuse input memory
- Reuse previously allocated memory

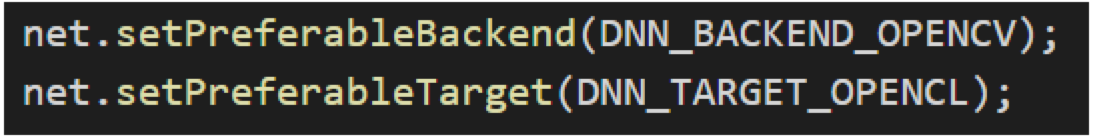


Reuse input Mem

General reuse algorithm

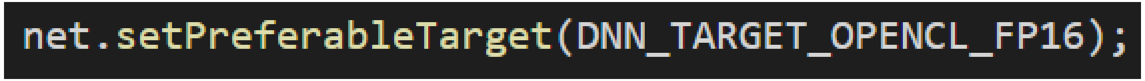
OpenCL acceleration

- Enable OpenCL acceleration



```
net.setPreferableBackend(DNN_BACKEND_OPENCV);
net.setPreferableTarget(DNN_TARGET_OPENCL);
```

Choose FP16:



```
net.setPreferableTarget(DNN_TARGET_OPENCL_FP16);
```

- No external dependency except OpenCL runtime
- Support FP 32 and FP16 data format

OpenCL acceleration

- Highly optimized convolution kernels
  - auto-tuning to find the best kernel configurations for a specific deployment environment
  - A set of pre-tuned kernel configurations built in the library
  - Tuning your own convolution kernel

    If you want to get the best performance for your specific deployment,

    try to run auto-tuning instead of using the default configurations.
  - How to enable auto-tuning?

    "export OPENCV_OCL4DNN_CONFIG_PATH=/path/to/config/dir"

    If you enable auto-tuning, the first time running a net model will be a little bit long.

    Next time, DNN module will use the cached configs directly and no need tuning again.

OpenCL acceleration

- For better performance on Intel GPU, use Neo driver if possible

  - Neo is the open-source OpenCL driver for Intel GPU

  - Supported Platforms

    Intel Core Processors with Gen8 graphics devices (formerly Broadwell) - OpenCL 2.1

    Intel Core Processors with Gen9 graphics devices (formerly Skylake, Kaby Lake, Coffee Lake) - OpenCL 2.1

    Intel Atom Processors with Gen9 graphics devices (formerly Apollo Lake, Gemini Lake) - OpenCL 1.2

  - Use the version as new as possible

    new version always has better performance

OpenCL acceleration

- Performance Data (in milliseconds):

Model	DNN, C++	DNN, OpenCL
AlexNet	19.32	11.83
GoogLeNet	23.08	8.20
ResNet-50	53.26	15.74
SqueezeNet V1.1	5.94	2.60
Inception-5h	24.30	9.27
Enet @ 512*256	68.26	17.26
OpenFace(nn4.small2)	17.47	4.02
MobileNet-SSD @ 300*300 20 classes Caffe	30.89	8.71
MobileNet-SSD v2@ 300*300 90 classes, TensorFlow	47.57	15.40

Configuration:
OS: Linux 4.16.0 x86_64 (Ubuntu 16.04)
Compiler: c++ 5.4.0
OpenCV: 3.4.3-308-g761c269
CPU: Intel(R) Core(TM) i7-6770HQ CPU@2.60GHz x8
GPU: Intel® Iris™ Pro Graphics 580 (Skylake GT4e)

For more performance data, see:
https://github.com/opencv/opencv/wiki/DNN-Efficiency

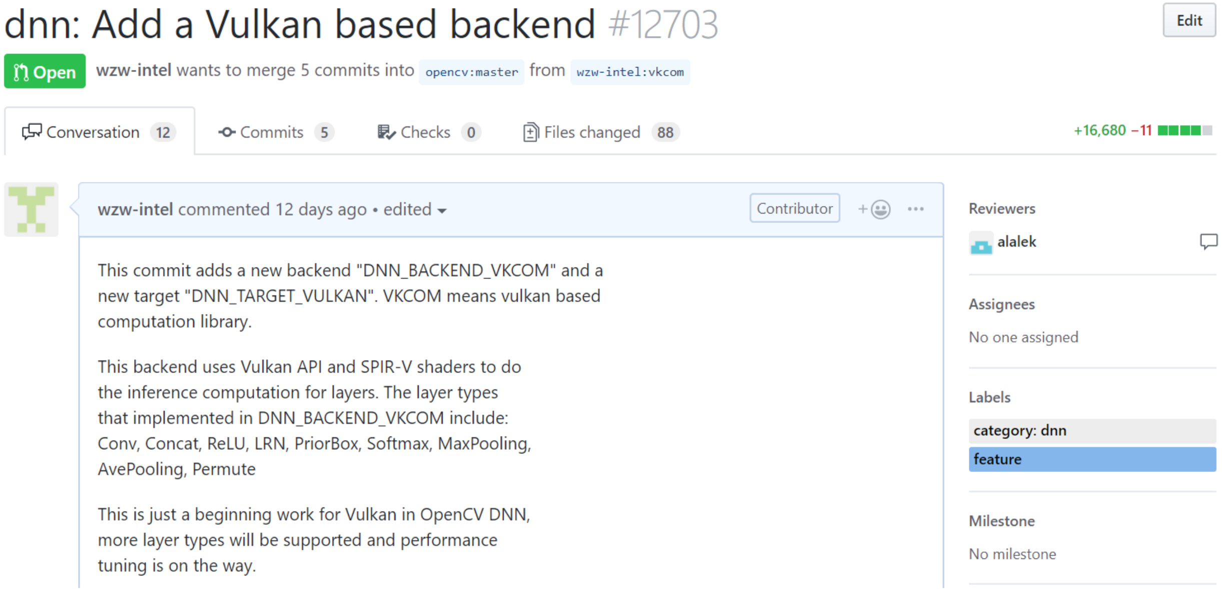
Vulkan backend

- Vulkan is the next generation Graphics and Compute API from Khronos, the same cross-industry group that maintains OpenGL
- Extend the usage of GPU acceleration for DNN module
- Use compute shader to implement layer computation

Vulkan backend

- A PR for Vulkan backend is in review

  https://github.com/opencv/opencv/pull/12703



dnn: Add a Vulkan based backend #12703

wzw-intel wants to merge 5 commits into `opencv:master` from `wzw-intel:vkcom`

wzw-intel commented 12 days ago • edited

This commit adds a new backend "DNN_BACKEND_VKCOM" and a new target "DNN_TARGET_VULKAN". VKCOM means vulkan based computation library.

This backend uses Vulkan API and SPIR-V shaders to do the inference computation for layers. The layer types that implemented in DNN_BACKEND_VKCOM include: Conv, Concat, ReLU, LRN, PriorBox, Softmax, MaxPooling, AvePooling, Permute

This is just a beginning work for Vulkan in OpenCV DNN, more layer types will be supported and performance tuning is on the way.

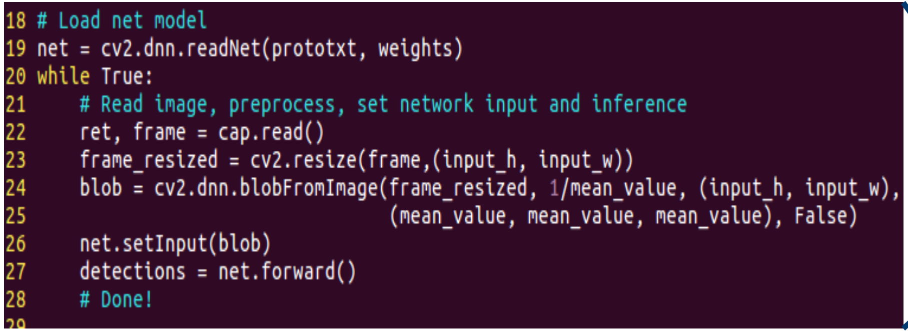
Reviewers: alalek

Assignees: No one assigned

Labels: category: dnn, feature

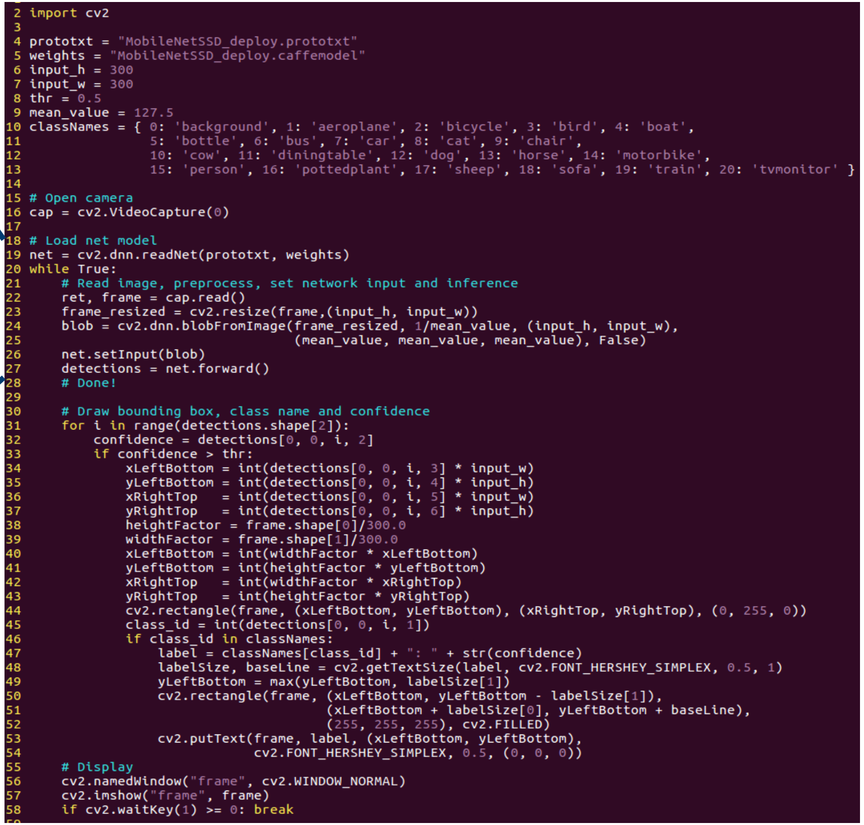
Milestone: No milestone

Sample: real-time objection detection with MobileNet-SSD



```python
# Load net model
net = cv2.dnn.readNet(prototxt, weights)
while True:
    # Read image, preprocess, set network input and inference
    ret, frame = cap.read()
    frame_resized = cv2.resize(frame,(input_h, input_w))
    blob = cv2.dnn.blobFromImage(frame_resized, 1/mean_value, (input_h, input_w),
                                 (mean_value, mean_value, mean_value), False)
    net.setInput(blob)
    detections = net.forward()
    # Done!
```

More samples at:
https://github.com/opencv/opencv/tree/master/samples/dnn



```python
import cv2

prototxt = "MobileNetSSD_deploy.prototxt"
weights = "MobileNetSSD_deploy.caffemodel"
input_h = 300
input_w = 300
thr = 0.5
mean_value = 127.5
classNames = { 0: 'background', 1: 'aeroplane', 2: 'bicycle', 3: 'bird', 4: 'boat',
               5: 'bottle', 6: 'bus', 7: 'car', 8: 'cat', 9: 'chair',
               10: 'cow', 11: 'diningtable', 12: 'dog', 13: 'horse', 14: 'motorbike',
               15: 'person', 16: 'pottedplant', 17: 'sheep', 18: 'sofa', 19: 'train', 20: 'tvmonitor' }

# Open camera
cap = cv2.VideoCapture(0)

# Load net model
net = cv2.dnn.readNet(prototxt, weights)
while True:
    # Read image, preprocess, set network input and inference
    ret, frame = cap.read()
    frame_resized = cv2.resize(frame,(input_h, input_w))
    blob = cv2.dnn.blobFromImage(frame_resized, 1/mean_value, (input_h, input_w),
                                 (mean_value, mean_value, mean_value), False)
    net.setInput(blob)
    detections = net.forward()
    # Done!

    # Draw bounding box, class name and confidence
    for i in range(detections.shape[2]):
        confidence = detections[0, 0, i, 2]
        if confidence > thr:
            xLeftBottom = int(detections[0, 0, i, 3] * input_w)
            yLeftBottom = int(detections[0, 0, i, 4] * input_h)
            xRightTop   = int(detections[0, 0, i, 5] * input_w)
            yRightTop   = int(detections[0, 0, i, 6] * input_h)
            heightFactor = frame.shape[0]/300.0
            widthFactor = frame.shape[1]/300.0
            xLeftBottom = int(widthFactor * xLeftBottom)
            yLeftBottom = int(heightFactor * yLeftBottom)
            xRightTop   = int(widthFactor * xRightTop)
            yRightTop   = int(heightFactor * yRightTop)
            cv2.rectangle(frame, (xLeftBottom, yLeftBottom), (xRightTop, yRightTop), (0, 255, 0))
            class_id = int(detections[0, 0, i, 1])
            if class_id in classNames:
                label = classNames[class_id] + ": " + str(confidence)
                labelSize, baseLine = cv2.getTextSize(label, cv2.FONT_HERSHEY_SIMPLEX, 0.5, 1)
                yLeftBottom = max(yLeftBottom, labelSize[1])
                cv2.rectangle(frame, (xLeftBottom, yLeftBottom - labelSize[1]),
                                     (xLeftBottom + labelSize[0], yLeftBottom + baseLine),
                                     (255, 255, 255), cv2.FILLED)
                cv2.putText(frame, label, (xLeftBottom, yLeftBottom),
                            cv2.FONT_HERSHEY_SIMPLEX, 0.5, (0, 0, 0))
    # Display
    cv2.namedWindow("frame", cv2.WINDOW_NORMAL)
    cv2.imshow("frame", frame)
    if cv2.waitKey(1) >= 0: break
```

Q & A

intel®

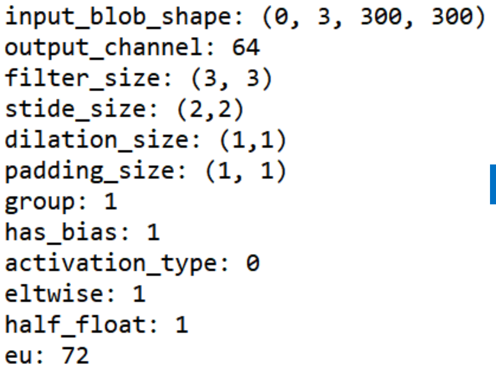
experience
what's inside™

Backups

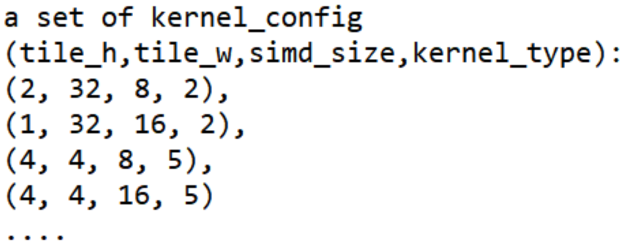
OpenCL acceleration

- Auto-tuning

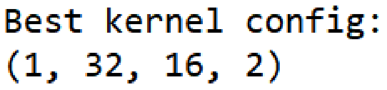
  - For each convolution “key”, generate a set of kernel configurations
  - Compile kernel for each kernel configuration, run kernel, get running time
  - Choose the best kernel configuration and store it on disk or memory



```
input_blob_shape: (0, 3, 300, 300)
output_channel: 64
filter_size: (3, 3)
stide_size: (2,2)
dilation_size: (1,1)
padding_size: (1, 1)
group: 1
has_bias: 1
activation_type: 0
eltwise: 1
half_float: 1
eu: 72
```



```
a set of kernel_config
(tile_h,tile_w,simd_size,kernel_type):
(2, 32, 8, 2),
(1, 32, 16, 2),
(4, 4, 8, 5),
(4, 4, 16, 5)
....
```



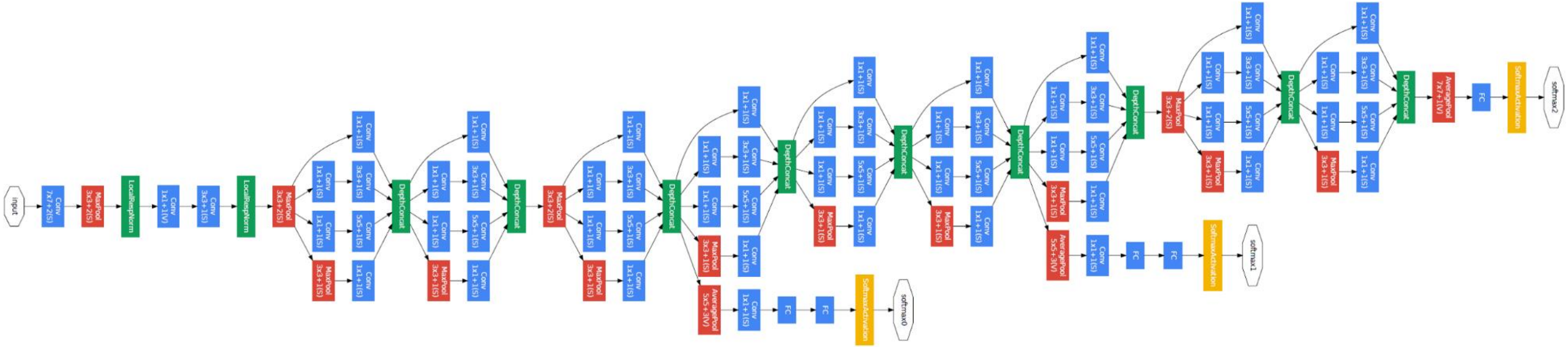
```
Best kernel config:
(1, 32, 16, 2)
```

A convolution “key” is a combination of all convolution parameters and GPU’s execution unit number.

A kernel_config is a combination of tile size, simd size and kernel type

Key concepts of Deep Neural Networks (DNN)

- A sample : GoogLeNet-V1



21 convolution layers + FC layer