Optimization for Convolution Neural Network on GPU

Contents

The goal of the lecture

What we will talk about

Covolutional Neural Network is a one of Deep Learning Networks.

CNN is widely used in image classification, obeject dection, segmentation
which are categories of Computer Vision.

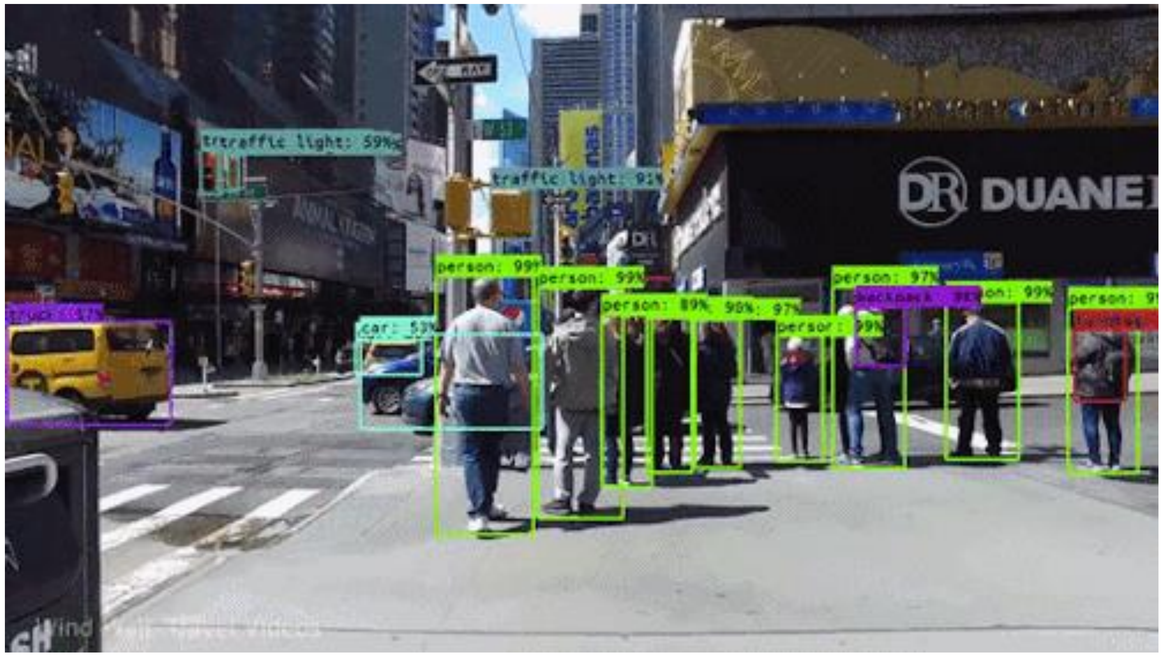
There are already many good CNN models out there.

However, most of them are slow and heavy in pracical and commercial usages.

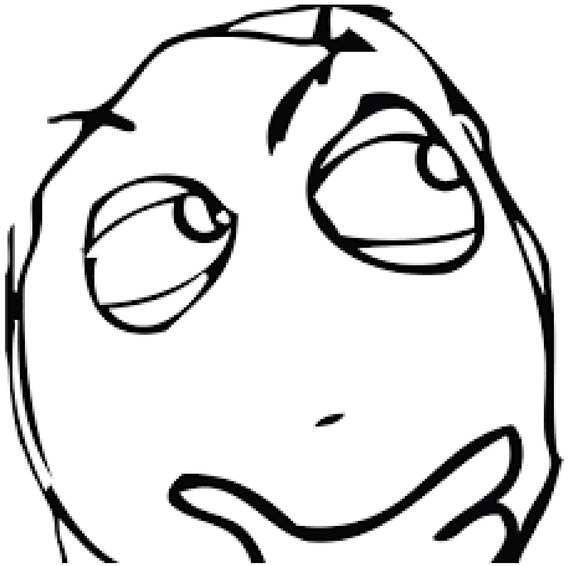
So, we are going to know the way to opitimize.

Convolutional Neural Network

CNN make computer see the world.



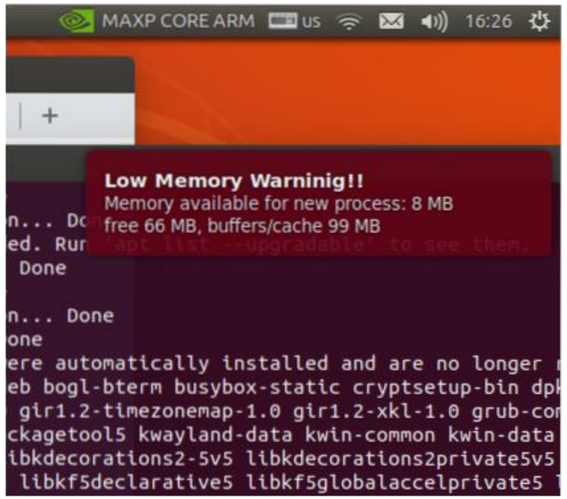
https://towardsdatascience.com/everything-you-ever-wanted-to-know-about-computer-vision-heres-a-look-why-it-s-so-awesome-e8a58dfb641e

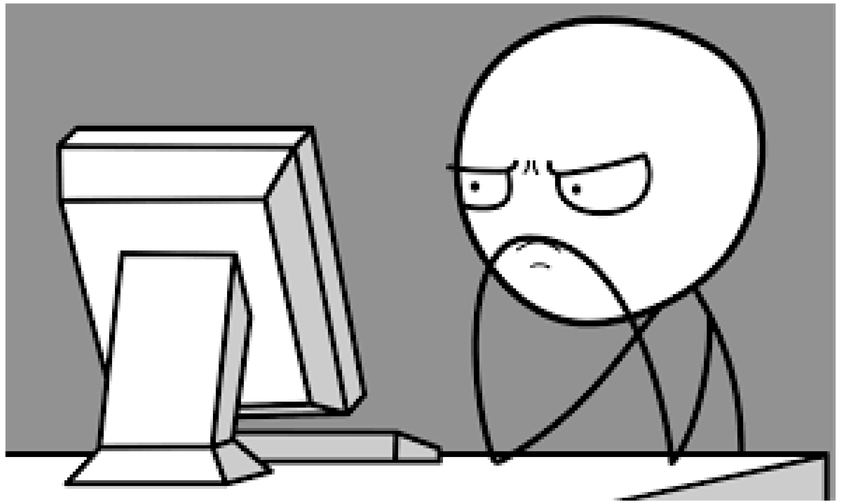


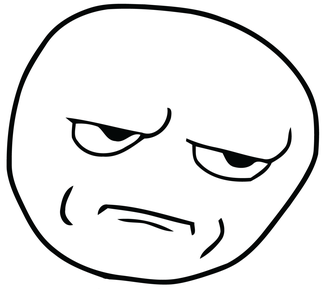
Well... this would be very useful.

Need for optimization of CNN

In reality, you feel like something is wrong...







It is too slow and uses too much memory.
I need fast and not heavy model.
How can I do?

Before knowing optimization, we should have basic understanding of CNN.
Let' s do it now.

What is CNN?

Architecture of CNN

The Hidden layers/Feature extraction part

convolution and **pooling** operations

features are detected

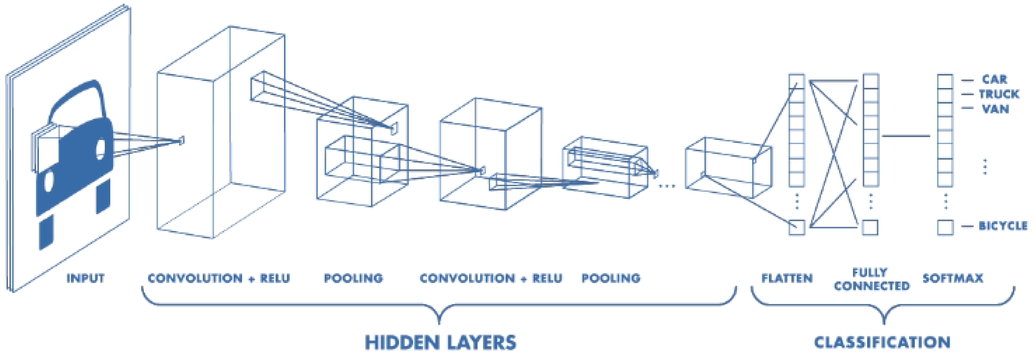
a picture of car,
its window, two wheels, and two headlights are **detected**

The Classification part

fully connected layers

these layers are **classifier**

this part assign a **probability** for object



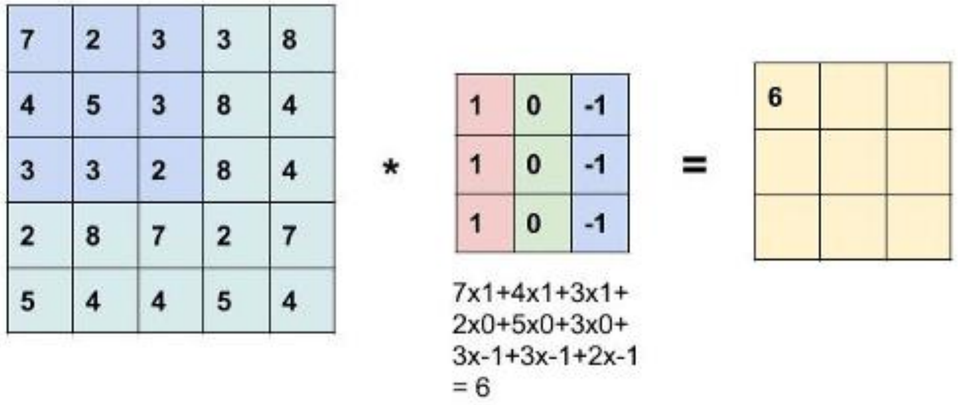
https://www.freecodecamp.org/news/an-intuitive-guide-to-convolutional-neural-networks-260c2de0a050/

https://www.mathworks.com/videos/introduction-to-deep-learning-what-are-convolutional-neural-networks--1489512765771.html

Convolution?

What is convolution?

The term convolution refers to the mathematical **combination** of two functions to produce a third function. **It merges two sets of information**. In the case of a CNN, the convolution is performed on the input data with the use of **a kernel or filter** to then produce a **feature map**.



https://medium.com/datadriveninvestor/convolutional-neural-networks-3b241a5da51e

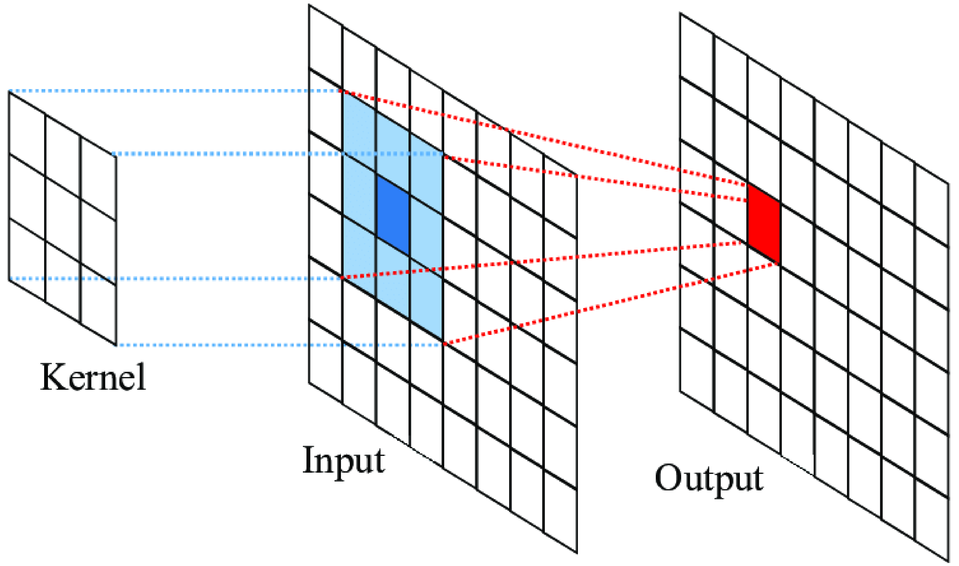
Kernel is a matrix or tensor!
This is what you remember.

Why convolution?

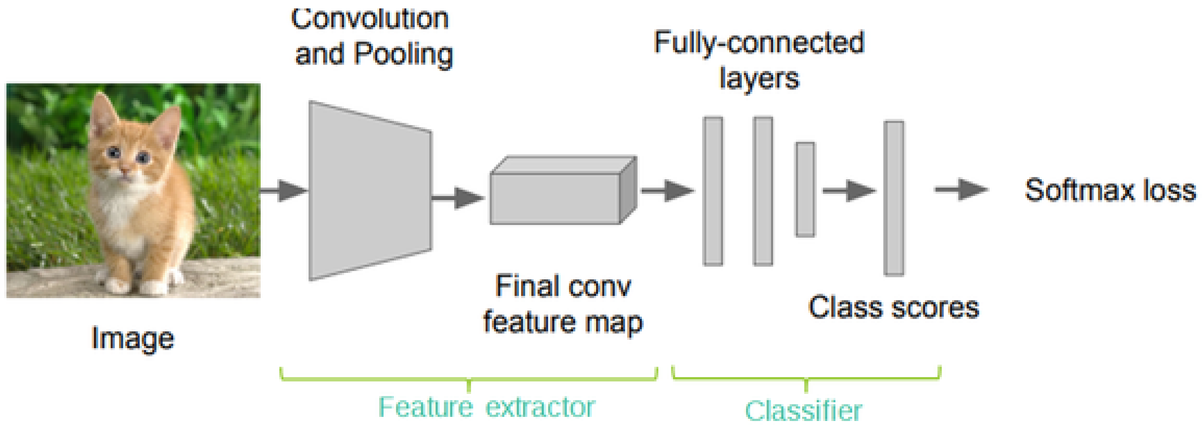
Feature extraction of image

Convolution is an efficient way of **feature extraction**, skilled in reducing data dimension and producing a less redundant data set, also called as a **feature** map.

Kernel works as a **feature** identifier, filtering out where the **feature** exists in the original image.



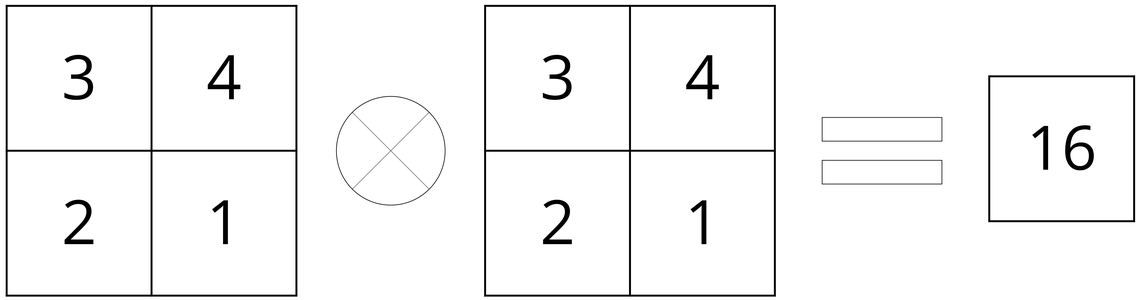
https://www.researchgate.net/publication/317685652_A_Novel_Kernel-Based_RegularizationTechnique_for_PET_Image_Reconstruction/link/5b72ae99a6fdcc87df79858c/download



https://www.quora.com/How-can-I-use-CNN-for-feature-extraction-of-images

Convolution Basic

Convolution is to **multiply and sum**. This is **all** you need know in this course.



3 * 1 + 4 * 2 + 2 * 1 + 1 * 3 = 16

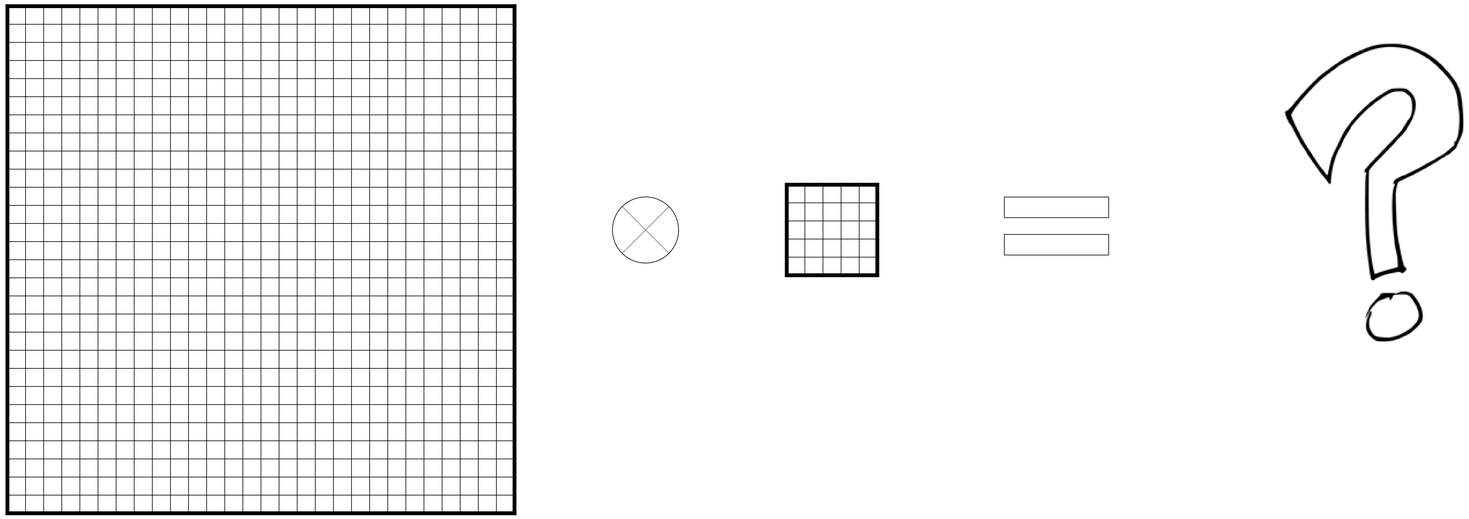
```
int sum = 0;
for (int i = 0; i < 2; i++){
    for (int j = 0; j < 2; j++){
        sum += A[i][j] * B[i][j];
    }
}
printf("%d\n", sum);
```

Shape after Convolution

$$oH = H - kH + 1$$
$$oW = W - kW + 1$$

if H = 28, W = 28, kH = 5, kW = 5

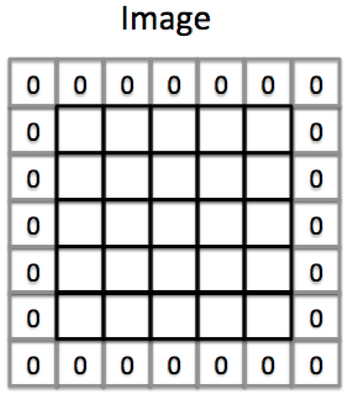
oH, oW ?



Padding

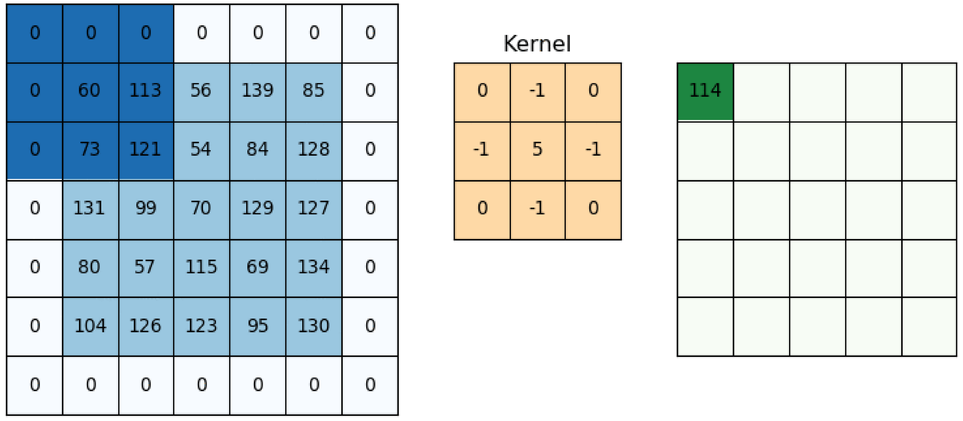
What is padding?

Padding is to **add extra pixels** outside the image. And **zero padding** means every pixel value that you add is zero.



Why need padding?

The kernel moves across the image, scanning and converting the data into a **smaller**, or sometimes larger.
In order to assist the kernel with processing the image, padding is added to the image to allow for more space for the kernel to **maintain** the size. This allows for **more accurate** analysis of images.



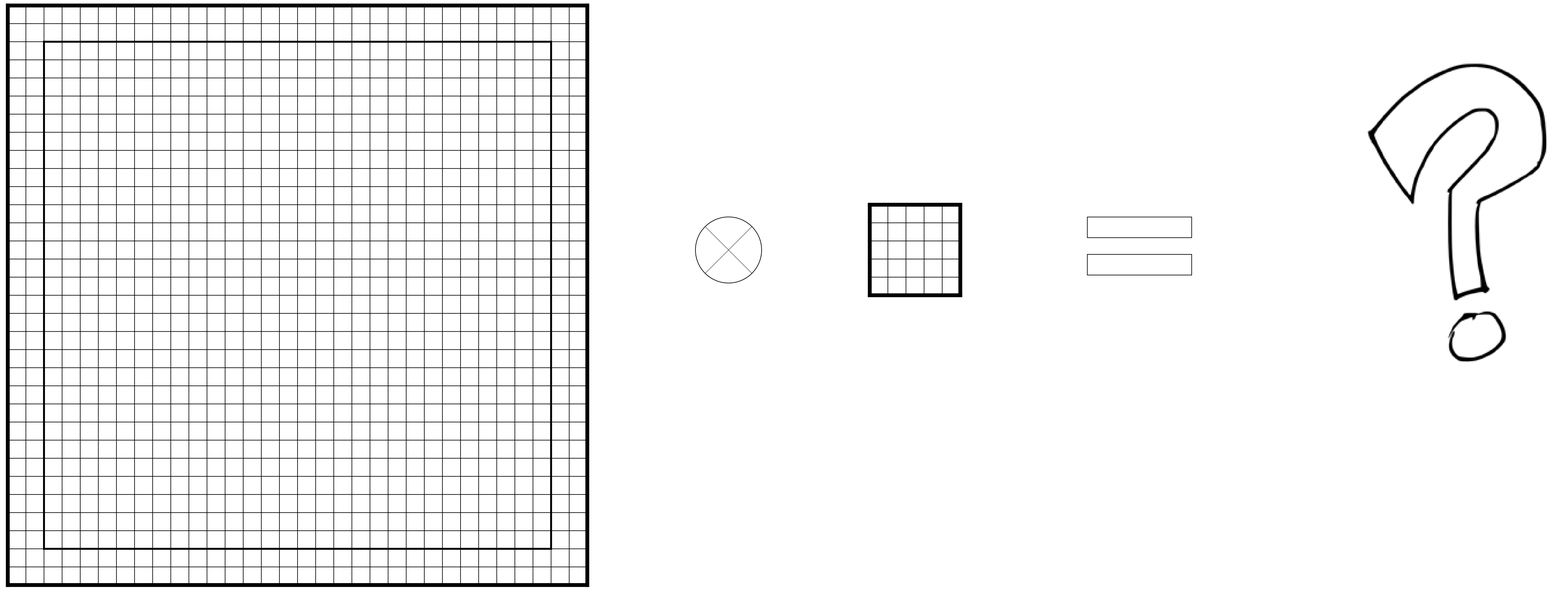
https://www.pyimagesearch.com/2018/12/31/keras-conv2d-and-convolutional-layers/

Shape with padding

oH = (H - 2 * PH -kH) + 1
oW = (W + 2 * pW - kW) + 1

if H = 28, W = 28, kH = 5, kW = 5, pH = 2, pW = 2

oH, oW ?



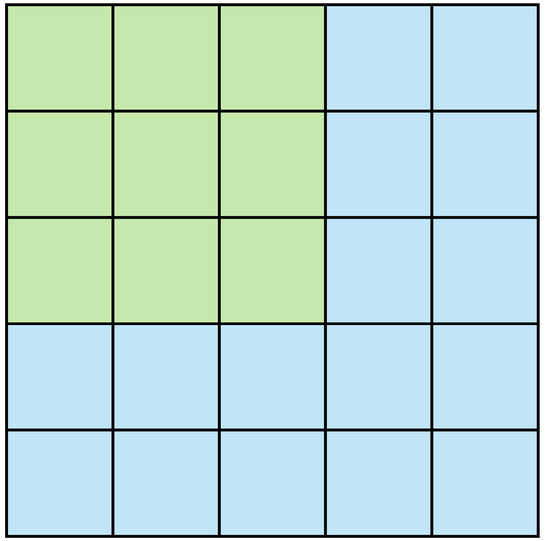
Stride

What is stride?

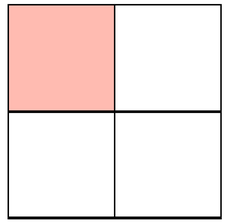
Stride denotes **how many steps** we are moving in convolution.

Why need stride?

Stride can **control** size of output as padding.
This effect is similar to **pooling**.



Stride 2



Feature Map

https://stackoverflow.com/questions/57384181/how-to-control-the-overlap-of-a-1d-convolution-model

Shape with stride

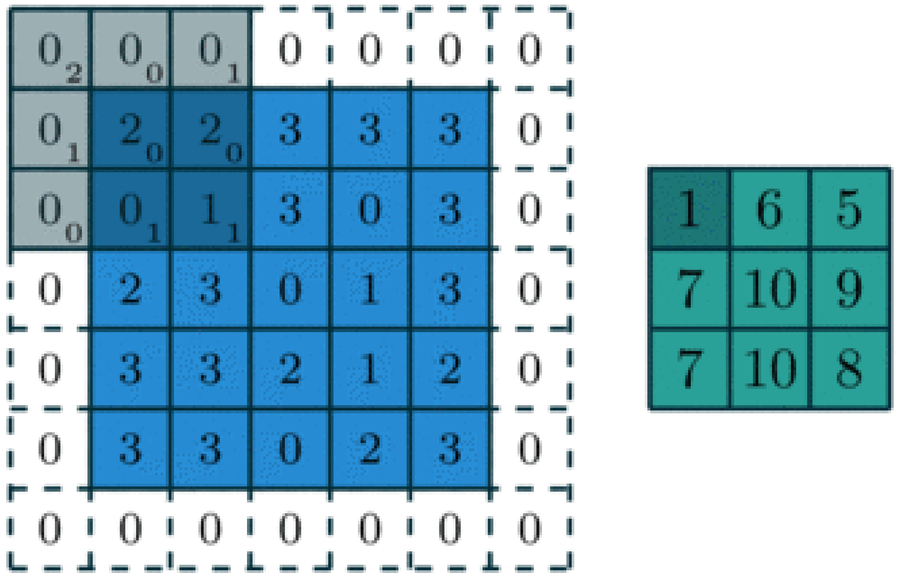
$$oH = (H - kH) / sH + 1$$
$$oW = (W - kW) / sW + 1$$

if H = 28, W = 28, kH = 5, kW = 5, sH = 2, sW = 2

oH, oW ?

Padding and Stride

You can apply both padding and stride to image.



http://deeplearning.net/software/theano/tutorial/conv_arithmetic.html

Shape with padding and stride

$$oH = (H + 2 * pH - kH) / sH + 1$$
$$oW = (W + 2 * pW - kW) / sW + 1$$

if H = 28, W = 28, kH = 5, kW = 5, pH = 2, pW = 2, sH = 2, sW = 2

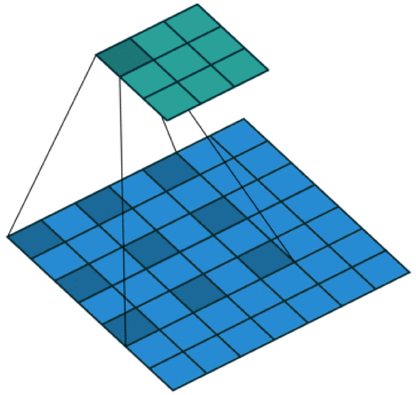
oH, oW ?

Tensorflow

$$oH = ceil((H + 2 * pH - kH) / sH) + 1$$
$$oW = ceil((W + 2 * pW - kW) / sW) + 1$$

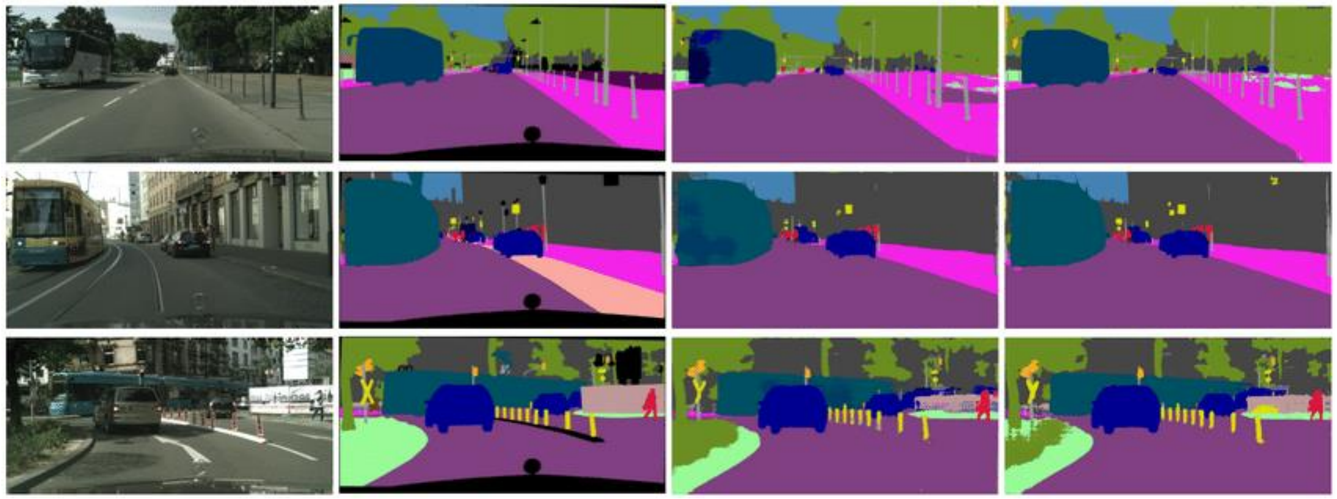
Dilation

Dilation is a convolution applied to input with defined **gaps**.
Dilation can be thought as a **combination** of convolution and maxpooling.
It **reduces** loss of a spatial dimension and numbers of weight.
Dilated convolution is widely used in **segmentation**.



dilation = 2

https://commons.wikimedia.org/wiki/File:Convolution_arithmetic_-_Dilation.gif

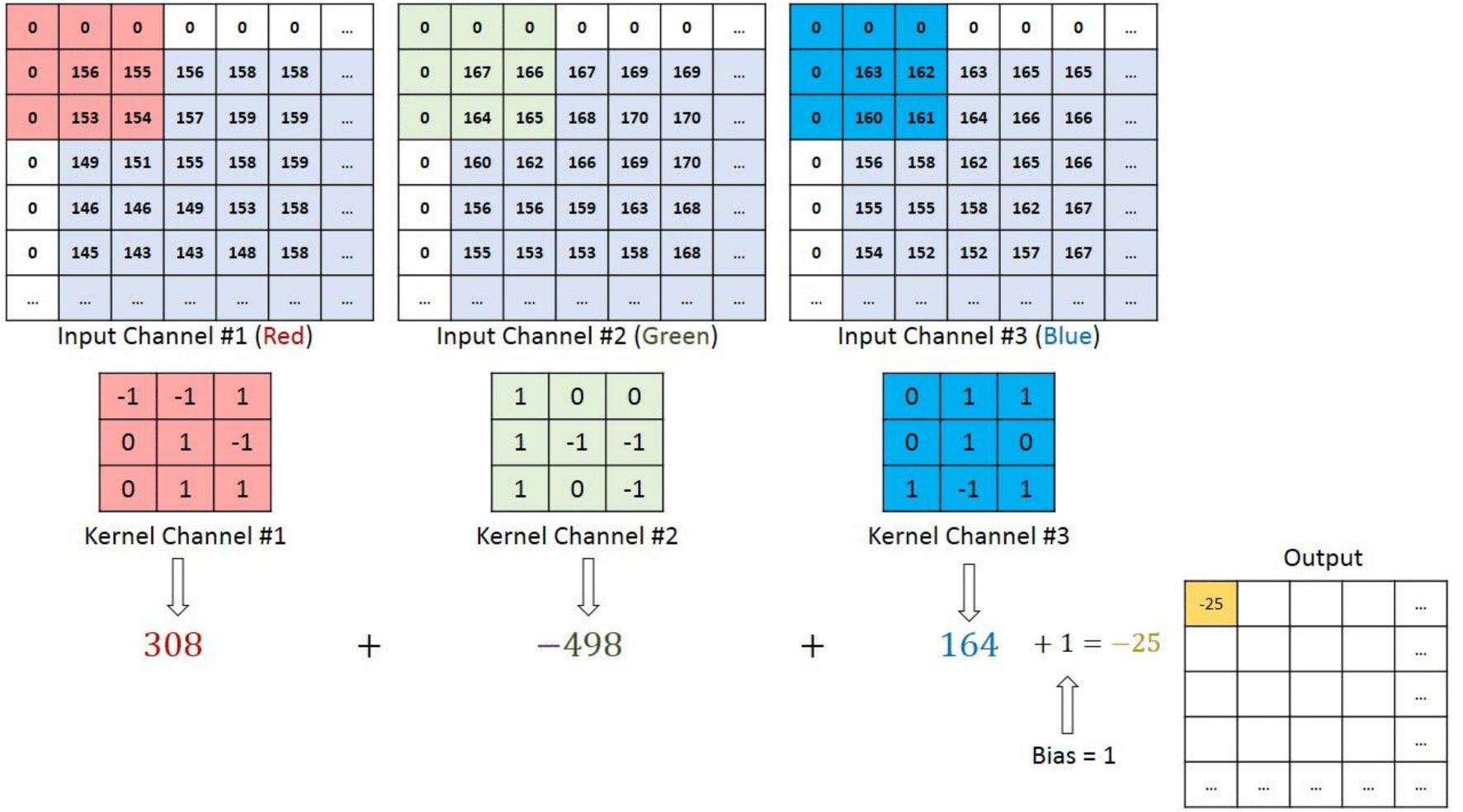


https://www.researchgate.net/figure/Effect-of-Hybrid-Dilated-Convolution-HDC-on-the-Cityscapes-validation-set-From-left-to_fig3_314115448

Semantic Segmentation

Multi channel convolution

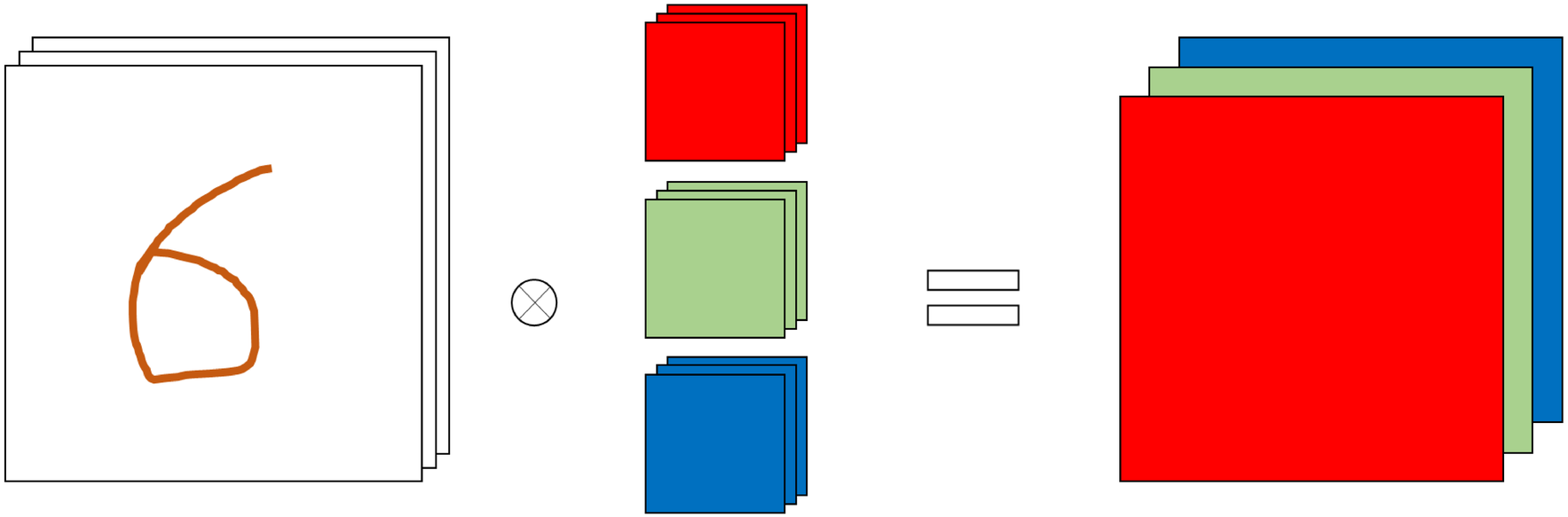
What would happen convoloution on multi channel?



https://www.researchgate.net/post/How_will_channels_RGB_effect_convolutional_neural_network

Multi channel convolution

When C(in_channel) = 3, K(out_channel) =3



(1, 3, 28, 28) (3, 3, 3, 3) (1, 3, 26, 26)

Most of images in real have three channels(RGB format).

API of Convolution

How to use API in PyTorch and TensorFlow.

PyTorch

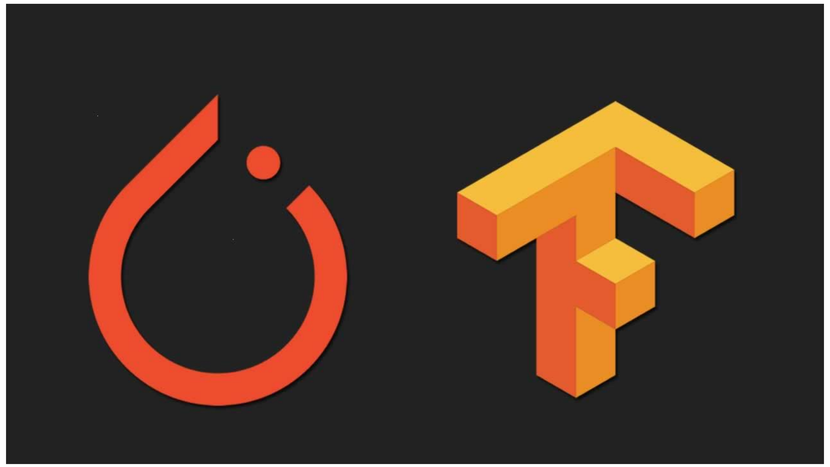
torch.nn.Conv2d(in_channels, out_channels, kernel_size, stride, **padding = (0, 0)**, dilation, bias = True)

padding = **(0, 0)** -> **no** padding
specify padding size after **calculating** for **same** shape

TensorFlow

tensorflow.keras.layers.Conv2D(filters, kernel_size, strides, **padding="valid"**, dilation_rate, use_bias=True)

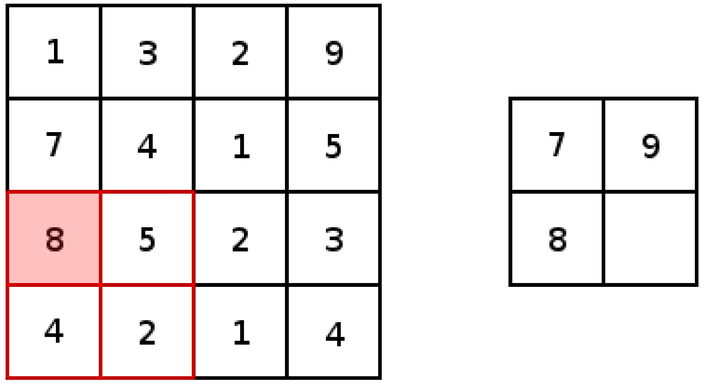
padding = "**valid**" -> **no** padding
padding = "**same**" -> pad **automatically** input for **same** shape



Maxpooling

What is maxpooling?

Maxpooling is a **pooling** operation that selects the **maximum** element from the region of the feature map covered by the filter.



https://www.oreilly.com/radar/visualizing-convolutional-neural-networks/

Why need maxpooling?

The output after **maxpooling** layer would be a feature map containing the **most prominent** features of the previous feature map.

Average pooling, Global Average pooling

Average pooling -> average of the window
Global Average pooling -> average of the whole feature map
These poolings **extract features** and **reduce the size** of feature map.

Maxipooling is similar to convotion.
For loop, fine the maximum instead of sum

Batch Normalization

What is batch normalization?

The **batch normalization** is to **normalize** the activations of each layer (transforming the inputs to be **mean 0** and **variance 1**).

Why need batch normalization?

The basic idea behind batch normalization is to **limit covariance shift**. This allows each layer to learn on a **more stable** distribution of inputs, and would thus **accelerate** the training of the network.

Input: Values of x over a mini-batch: $\mathcal{B} = \{x_{1...m}\}$;
Parameters to be learned: γ, β
Output: $\{y_i = \mathrm{BN}_{\gamma,\beta}(x_i)\}$

$$\mu_{\mathcal{B}} \leftarrow \frac{1}{m}\sum_{i=1}^{m} x_i \qquad \text{// mini-batch mean}$$

$$\sigma_{\mathcal{B}}^2 \leftarrow \frac{1}{m}\sum_{i=1}^{m}(x_i - \mu_{\mathcal{B}})^2 \qquad \text{// mini-batch variance}$$

$$\widehat{x}_i \leftarrow \frac{x_i - \mu_{\mathcal{B}}}{\sqrt{\sigma_{\mathcal{B}}^2 + \epsilon}} \qquad \text{// normalize}$$

$$y_i \leftarrow \gamma\widehat{x}_i + \beta \equiv \mathrm{BN}_{\gamma,\beta}(x_i) \qquad \text{// scale and shift}$$

Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift https://arxiv.org/abs/1502.03167

$$y = \gamma \frac{x' - m}{\sqrt{\sigma^2 + \varepsilon}} + \beta$$

Fusion of Convolution and Batch Normalization

In fact, we can **combine** convolution and batch normalization.
We call this **fusion** of convolution and batch normalization.

$$x' = wx + b \quad \boxplus \quad y = \gamma \frac{x' - m}{\sqrt{\sigma^2 + \varepsilon}} + \beta$$

$$\Rightarrow \quad \mathbf{y = \frac{\gamma w}{\sqrt{\sigma^2 + \varepsilon}} x + \frac{\gamma(b - m)}{\sqrt{\sigma^2 + \varepsilon}} + \beta}$$

This is a **fusion** of convolution and batch normalization.

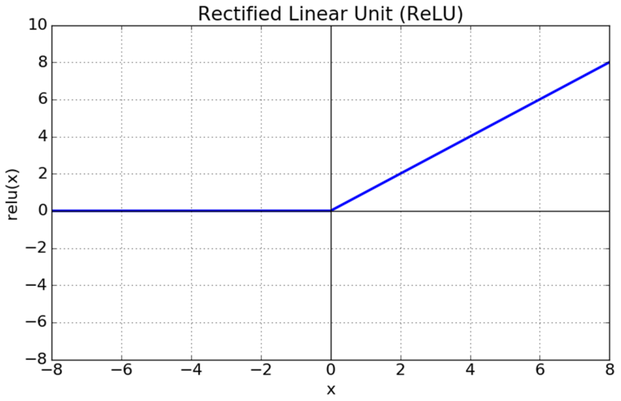
This makes **two** operations to **one** operation.

This makes model **fast** and **lite**.

Relu

What is relu?

Relu returns **zero** if it receives any **negative** input, but for any **positive** value it returns that **value back**.



f(x)=max(0,x)

https://www.oreilly.com/radar/visualizing-convolutional-neural-networks/

Why need relu?

Relu is a **nonlinear** function. If **non-linear** activation functions weren't used, the network would be a **large linear** classifier, and could be simplified by simply multiplying the weight matrices together. So **nonlinear** activation functions are important because the function you are trying to learn is usually **nonlinear**.

$$y1 = a1 * x1 + b1$$

$$y2 = a2 * y1 + b2 = a2 * (a1 * x1 + b1)$$

$$y3 = a3 * y2 + b3 = a3 * (a2 * a1 * x1 + a2 * b1) + a3$$

...

$$\boldsymbol{y = a' * x + b'}$$

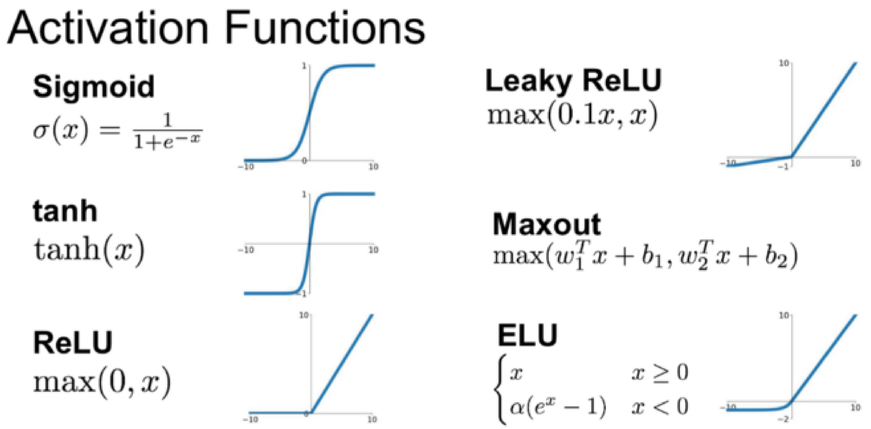
Relu is one of activation functions.

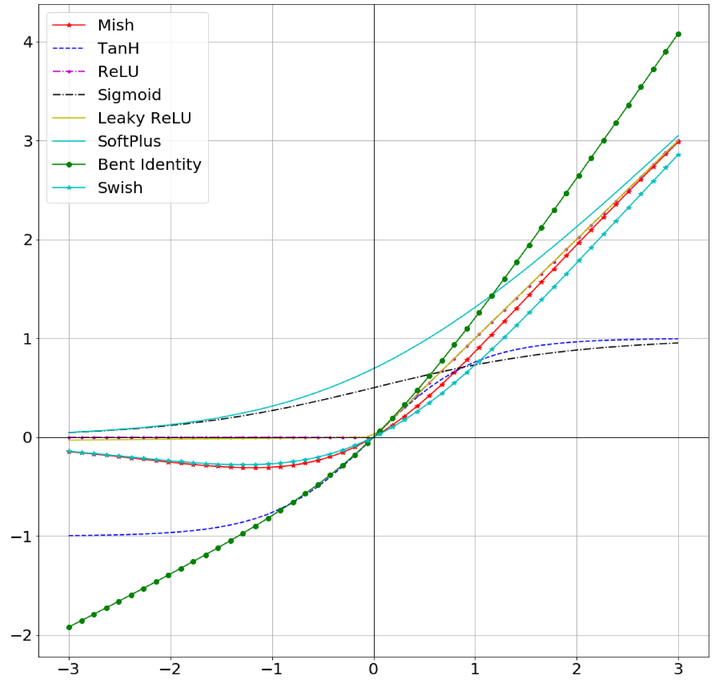
The **point** of activation function is that it makes **non-linearity**.
Activation function should be **differential**.
It makes **back-progation** possible.

Activation Functions

Good activation Fuction?

1. Non-Linearity -> **training**
2. Differential -> **back-propagation**
3. Simple computation -> **cost**





Researches of activation function continue.

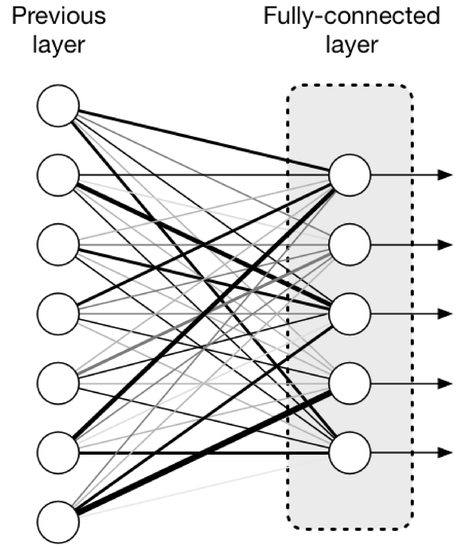
Fully Connected layer(Dense layer)

What is fully connected layer?

Fully Connected layers in a neural networks are those layers where **all** the inputs from one layer are **connected** to **every** activation unit of the next layer.

What is objective of fully connected layer?

The **objective** of a fully connected layer is to take the results of the convolution/pooling process and use them to **classify** the image into a label.



https://medium.com/@tecokids.monastir/fully-connected-layer-with-dynamic-input-shape-70c869ae71af

torch.nn.Linear(in_features, out_features)

tensorflow.keras.layers.Dense(units)

In the deep learning framework, you use **flatten** layer before dense layer. But, we won't use **flatten** layer in this course. **You will know the reason later**.

Softmax

What is softmax?

Softmax is one of **activation functions**. This outputs a vector that represents the **probabilities**(not exact) of a list potential outcomes.

$$\text{Softmax}(x_i) = \frac{\exp(x_i)}{\sum_j \exp(x_j)}$$

What does do softmax?

Softmax assigns **probabilities** to each class in a **multi-class** problem. Those **probabilities** must add up to **1.0**. This constraint helps training **converge more quickly** than it otherwise would.

softmax ≠ probability

Probabilities

1	0.1
2	0.2
5	0.5
2	0.2
0	0

Softmax

e^1	2.72	0.016
e^2	7.39	0.044
e^5	148.41	0.89
e^2	7.39	0.044
e^0	1	0.006

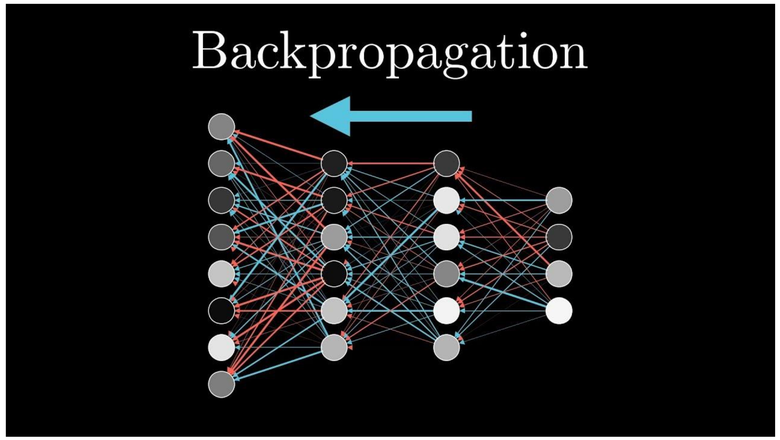
Back-Propagation

What is back-propagation?

Back-propagation is just a way of **propagating the total loss back** into the neural network to **know how much of the loss** every node is responsible for, and **updating the weights** in such a way that **minimizes the loss** by giving the nodes with higher error rates lower weights and vice versa.

Back-propagation updates the **weights** and you get the **last weights** after train was **finished**.
As a result, **it makes the model run**.
But, this is out of our course.

Model



https://i.ytimg.com/vi/Ilg3gGewQ5U/maxresdefault.jpg

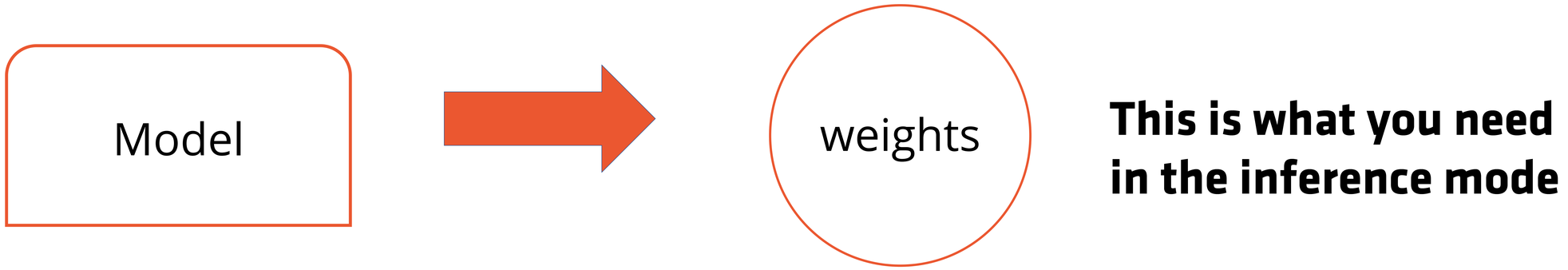
Weights

After training, you get weights from the model.

There are a lot of **weights** and other things **not needed** from each layer in the model file.
When the model is doing **inference**, you need only **weights**.

If you use other things, too.
It causes a **slowdown** and **memory leaks**.

If you can use only **what is essential**, model will be **fast** and **light**.



The inspiration for optimization

In the **training** mode, weights can be **updated** by back-propagation.

In the **inference** mode, weights are **constant**.

This means that you can make the model **fast and light**.

Summary

What have we talked about so far?

CNN and why need optimization

Basic understanding of CNN(convolution, maxpooling, batch-normalization...)

Inspiration for optimization(fusion, remove values not need)

We will talk about the way to optimize the CNN model.

How to optimize

First, you collect what is important in the model file.
Then only use them in the time of inference.

What is important is weight of each layer.

Second, you write efficient codes of GPU.

In order to that, you need to know about GPU.

Why optimization is difficult?

First, there are differences over Deep Learning Framwork.

They have their own formats, rules and structures.

Second, you should have understanding of GPU.

It is not familiar to write a code for GPU because you are unfamiliar with GPU.

Therefore, you need to learn about them.

Differencs between PyTorch and Tensorflow

PyTorch	TensorFlow
- (N, C, H, W)	- (N, H, W, C)
- channel first	- channel last
- padding : (pH, pW)	- padding : "same", "valid"
- n * C * H * W + c * H * W + h * W + w	- n * H * W * C + h * W * C + w * C + c

This difference makes you crazy in practice.

The shapes of weight, output and many things are different.

MNIST

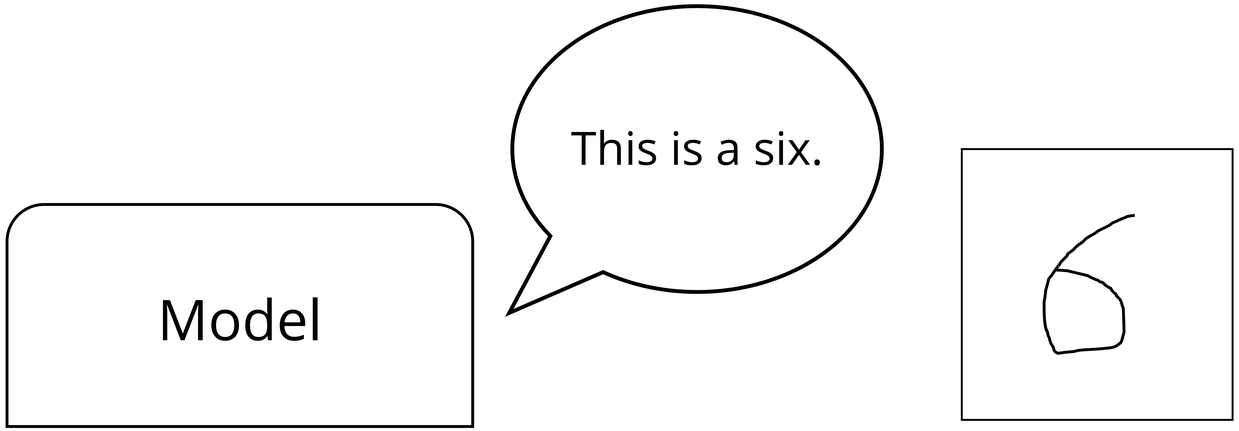
What is MNIST?

MNIST is a **dataset** of small square **28 × 28 pixel grayscale** images of **handwritten** single digits between 0 and 9.

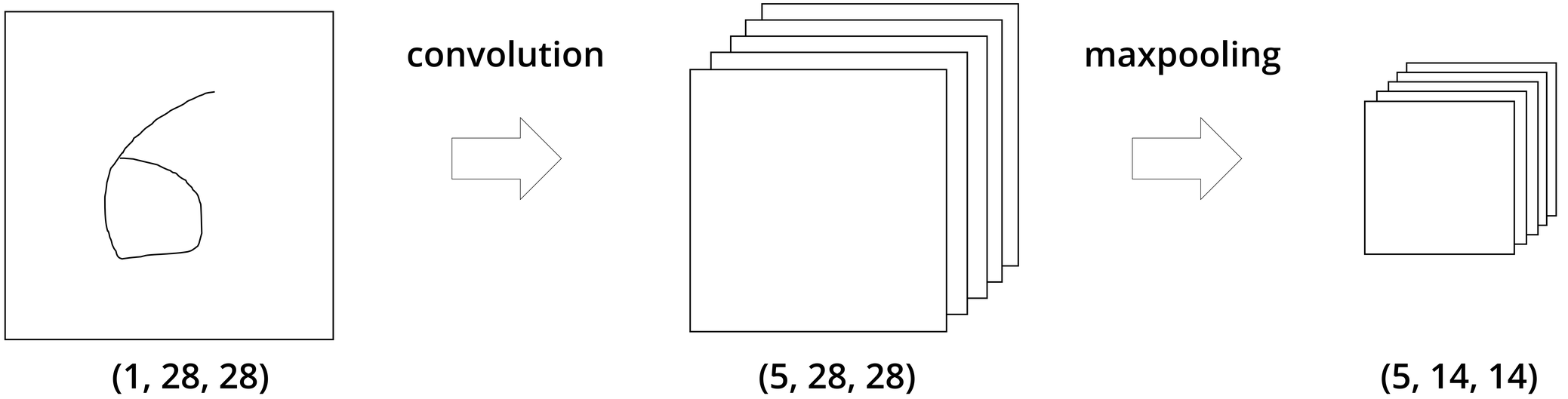
http://yann.lecun.com/exdb/mnist/

We will make CNN model for MNIST.

The **model** will **predict** a **label** of each image.



CNN model for MNIST



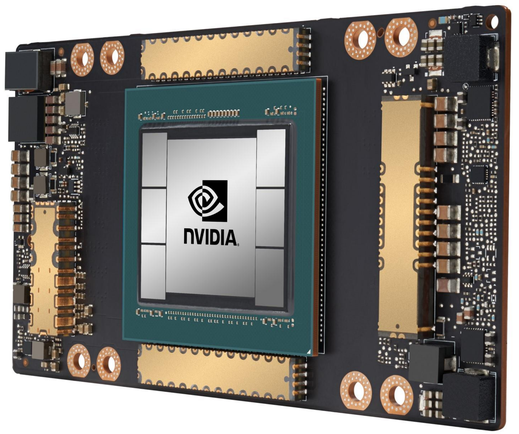
GPU

What is GPU?

GPU is a **graphics processing unit**. It is a processor that is specially-designed to handle intensive graphics rendering tasks originally.

Why do we use GPU in Deep Learning?

GPU can process multiple computations **simultaneously**.
They have a **large number of cores**, which allows for better computation of multiple **parallel** processes.
GPU makes **fast C**onvolutional **N**eural **N**etwork especially.

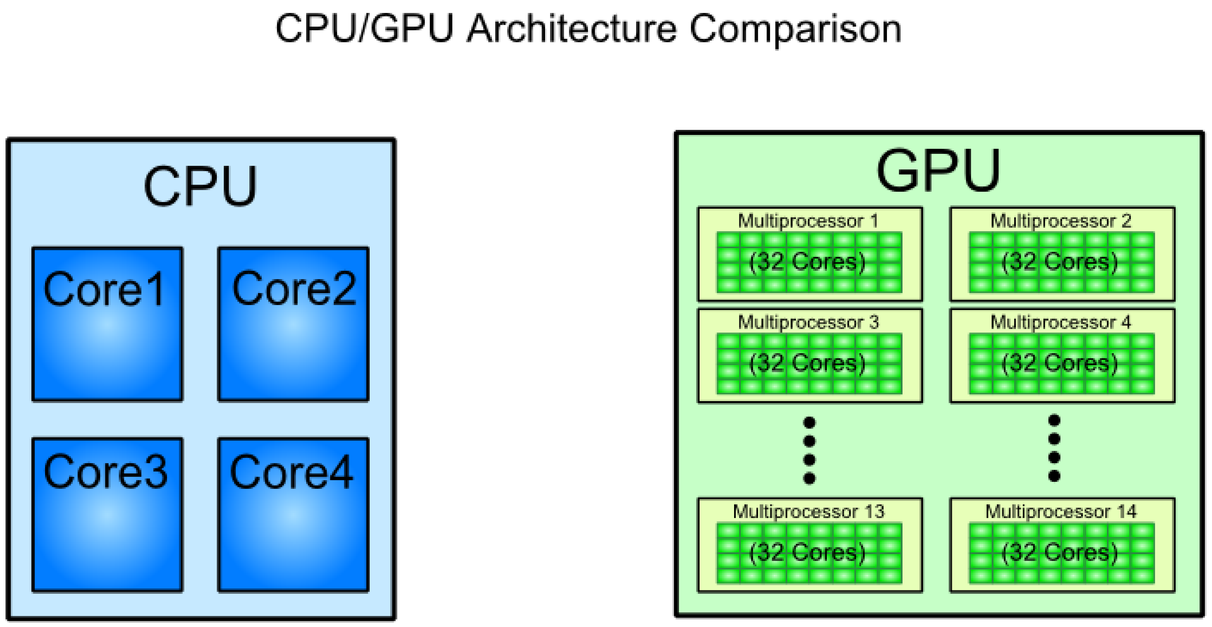


NVIDIA A100

Differences of CPU and GPU

CPU is composed of **a few** cores with lots of cache memory that can handle **a few threads** at a time

GPU is composed of **hundreds** of cores that can handle **thousands of threads** simultaneously.



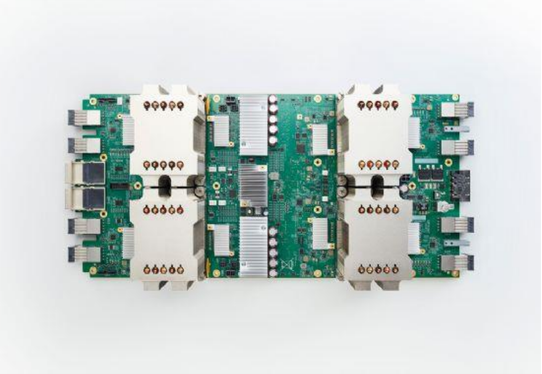
Advantages of GPU

So, why GPU?

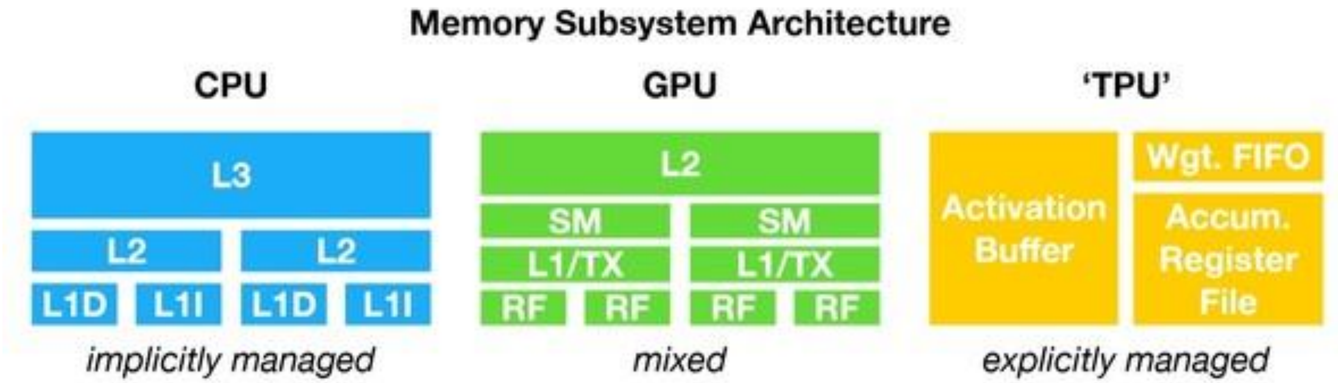
GPUs do massively **parallel** operations such as convolution of all values in a region of memory **quickly** than **CPU** due to **lots of cores**. This is why **GPU** is used in **CNN**.

TPU?

TPUs are **specialized** to process neural network simulations. They **only do this**. So they are better at **artificial intelligence**. This is very advanced and used in **AI** related super computers - servers.

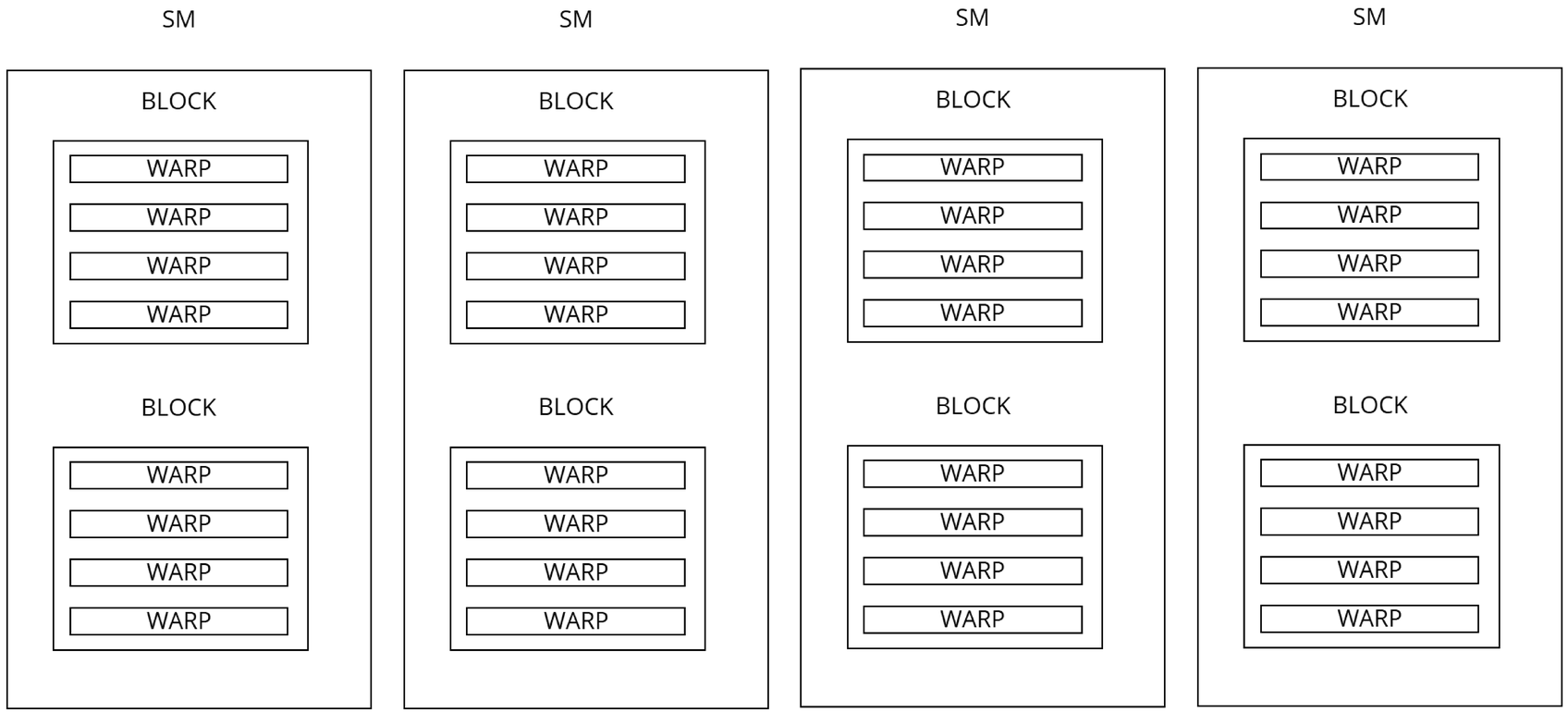


Google Cloud TPU



How is it possible?

Cores of GPU execute in the same time.



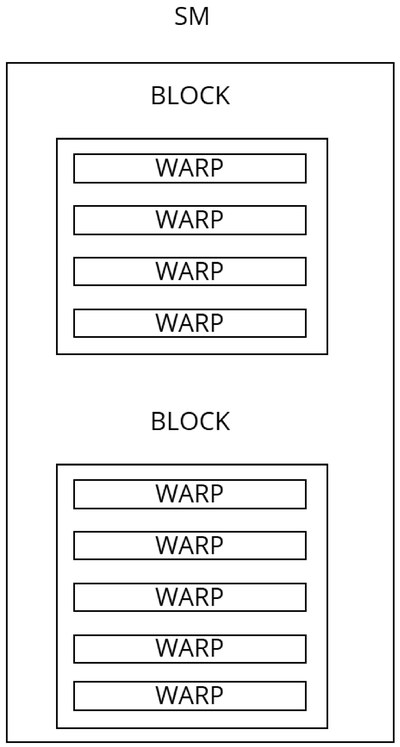
About GPU

SM(Streaming Multiprocessor) : Part of the GPU that runs CUDA **kernel**s.

WARP : The basic unit of execution. **32** THEADs.

THREAD : a basic element of the data to be processed.

BLOCK : it consist of WARPs. It's size can be different.



WARP consists of 32 THREADs

BLOCK consists of WARPs

SM consists of BLOCKs

BLOCK’s size can be different

CUDA

What is CUDA?

CUDA (**C**ompute **U**nified **D**evice **A**rchitecture) is a **parallel** computing platform and application programming interface(API) model created by **NVIDIA**.
You think it as programming **language** for **GPU**.

Standard C Code

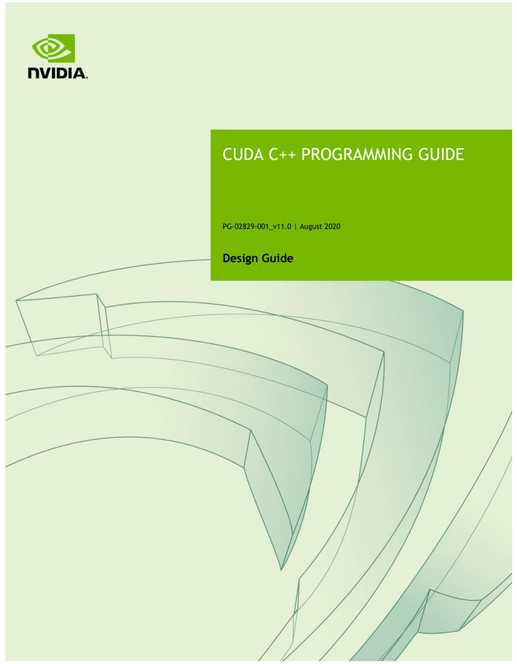
```
void saxpy(int n, float a,
           float *x, float *y)
{
  for (int i = 0; i < n; ++i)
    y[i] = a*x[i] + y[i];
}

int N = 1<<20;


// Perform SAXPY on 1M elements
saxpy(N, 2.0, x, y);
```

C with CUDA extensions

```
__global__
void saxpy(int n, float a,
           float *x, float *y)
{
  int i = blockIdx.x*blockDim.x + threadIdx.x;
  if (i < n) y[i] = a*x[i] + y[i];
}

int N = 1<<20;
cudaMemcpy(x, d_x, N, cudaMemcpyHostToDevice);
cudaMemcpy(y, d_y, N, cudaMemcpyHostToDevice);

// Perform SAXPY on 1M elements
saxpy<<<4096,256>>>(N, 2.0, x, y);

cudaMemcpy(d_y, y, N, cudaMemcpyDeviceToHost);
```

https://blogs.nvidia.com/blog/2012/09/10/what-is-cuda-2/

Example of CUDA

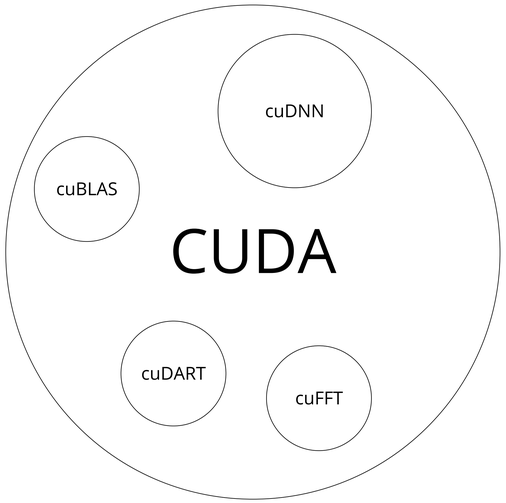


https://docs.nvidia.com/cuda/pdf/CUDA_C_Programming_Guide.pdf

cuDNN

What is cuDNN?

cuDNN is one of **D**eep **N**eural **N**etwork libraries based on **CUDA**.
The deep learning framework like **tensorflow**, **pytorch** uses **cuDNN**.




```
2020-08-20 13:43:32.203533: I tensorflow/stream_executor/platform/default/dso_loader.cc:44] Successfully opened dynamic library nvcuda.dll
2020-08-20 13:43:32.247549: I tensorflow/core/common_runtime/gpu/gpu_device.cc:1618] Found device 0 with properties:
name: GeForce RTX 2080 Ti major: 7 minor: 5 memoryClockRate(GHz): 1.545
pciBusID: 0000:01:00.0
2020-08-20 13:43:32.247746: I tensorflow/stream_executor/platform/default/dso_loader.cc:44] Successfully opened dynamic library cudart64_100.dll
2020-08-20 13:43:32.281948: I tensorflow/stream_executor/platform/default/dso_loader.cc:44] Successfully opened dynamic library cublas64_100.dll
2020-08-20 13:43:32.314147: I tensorflow/stream_executor/platform/default/dso_loader.cc:44] Successfully opened dynamic library cufft64_100.dll
2020-08-20 13:43:32.332191: I tensorflow/stream_executor/platform/default/dso_loader.cc:44] Successfully opened dynamic library curand64_100.dll
2020-08-20 13:43:32.384317: I tensorflow/stream_executor/platform/default/dso_loader.cc:44] Successfully opened dynamic library cusolver64_100.dll
2020-08-20 13:43:32.413627: I tensorflow/stream_executor/platform/default/dso_loader.cc:44] Successfully opened dynamic library cusparse64_100.dll
2020-08-20 13:43:32.497180: I tensorflow/stream_executor/platform/default/dso_loader.cc:44] Successfully opened dynamic library cudnn64_7.dll
2020-08-20 13:43:32.498616: I tensorflow/core/common_runtime/gpu/gpu_device.cc:1746] Adding visible gpu devices: 0
2020-08-20 13:43:32.500385: I tensorflow/core/platform/cpu_feature_guard.cc:142] Your CPU supports instructions that this TensorFlow binary was not compiled to use: AVX2
2020-08-20 13:43:32.503618: I tensorflow/core/common_runtime/gpu/gpu_device.cc:1618] Found device 0 with properties:
name: GeForce RTX 2080 Ti major: 7 minor: 5 memoryClockRate(GHz): 1.545
pciBusID: 0000:01:00.0
2020-08-20 13:43:32.503818: I tensorflow/stream_executor/platform/default/dso_loader.cc:44] Successfully opened dynamic library cudart64_100.dll
2020-08-20 13:43:32.503948: I tensorflow/stream_executor/platform/default/dso_loader.cc:44] Successfully opened dynamic library cublas64_100.dll
2020-08-20 13:43:32.504074: I tensorflow/stream_executor/platform/default/dso_loader.cc:44] Successfully opened dynamic library cufft64_100.dll
2020-08-20 13:43:32.504214: I tensorflow/stream_executor/platform/default/dso_loader.cc:44] Successfully opened dynamic library curand64_100.dll
2020-08-20 13:43:32.504347: I tensorflow/stream_executor/platform/default/dso_loader.cc:44] Successfully opened dynamic library cusolver64_100.dll
2020-08-20 13:43:32.504478: I tensorflow/stream_executor/platform/default/dso_loader.cc:44] Successfully opened dynamic library cusparse64_100.dll
2020-08-20 13:43:32.504607: I tensorflow/stream_executor/platform/default/dso_loader.cc:44] Successfully opened dynamic library cudnn64_7.dll
2020-08-20 13:43:32.505080: I tensorflow/core/common_runtime/gpu/gpu_device.cc:1746] Adding visible gpu devices: 0
2020-08-20 13:43:34.216284: I tensorflow/core/common_runtime/gpu/gpu_device.cc:1159] Device interconnect StreamExecutor with strength 1 edge matrix:
2020-08-20 13:43:34.216420: I tensorflow/core/common_runtime/gpu/gpu_device.cc:1165]      0
2020-08-20 13:43:34.216498: I tensorflow/core/common_runtime/gpu/gpu_device.cc:1178] 0:   N
2020-08-20 13:43:34.218271: I tensorflow/core/common_runtime/gpu/gpu_device.cc:1304] Created TensorFlow device (/job:localhost/replica:0/task:0/device:GPU:0 with 8685 MB memory)
Epoch 1/3
2020-08-20 13:43:34.736606: I tensorflow/stream_executor/platform/default/dso_loader.cc:44] Successfully opened dynamic library cublas64_100.dll
2020-08-20 13:43:35.232558: I tensorflow/stream_executor/platform/default/dso_loader.cc:44] Successfully opened dynamic library cudnn64_7.dll
2020-08-20 13:43:36.940103: W tensorflow/stream_executor/cuda/redzone_allocator.cc:312] Internal: Invoking ptxas not supported on Windows
Relying on driver to perform ptx compilation. This message will be only logged once.
```

Tensorflow opens many libraries when using GPU.

Libraries based on CUDA.

cuBLAS -CUDA Basic Linear Algebra Subroutines library

cuDART – CUDA Runtime library

cuFFT – CUDA Fast Fourier Transform library

cuRAND – CUDA Random Number Generation library

cuSOLVER – CUDA based collection of dense and sparse direct solvers

cuSPARSE – CUDA Sparse Matrix library

How to use GPU

There are **two** ways in a big way.

The one is a way to use **existing** framework. (**TensoFlow**, **PyTorch**...)

In the second way, you **make** code **yourself** by using **CUDA**.

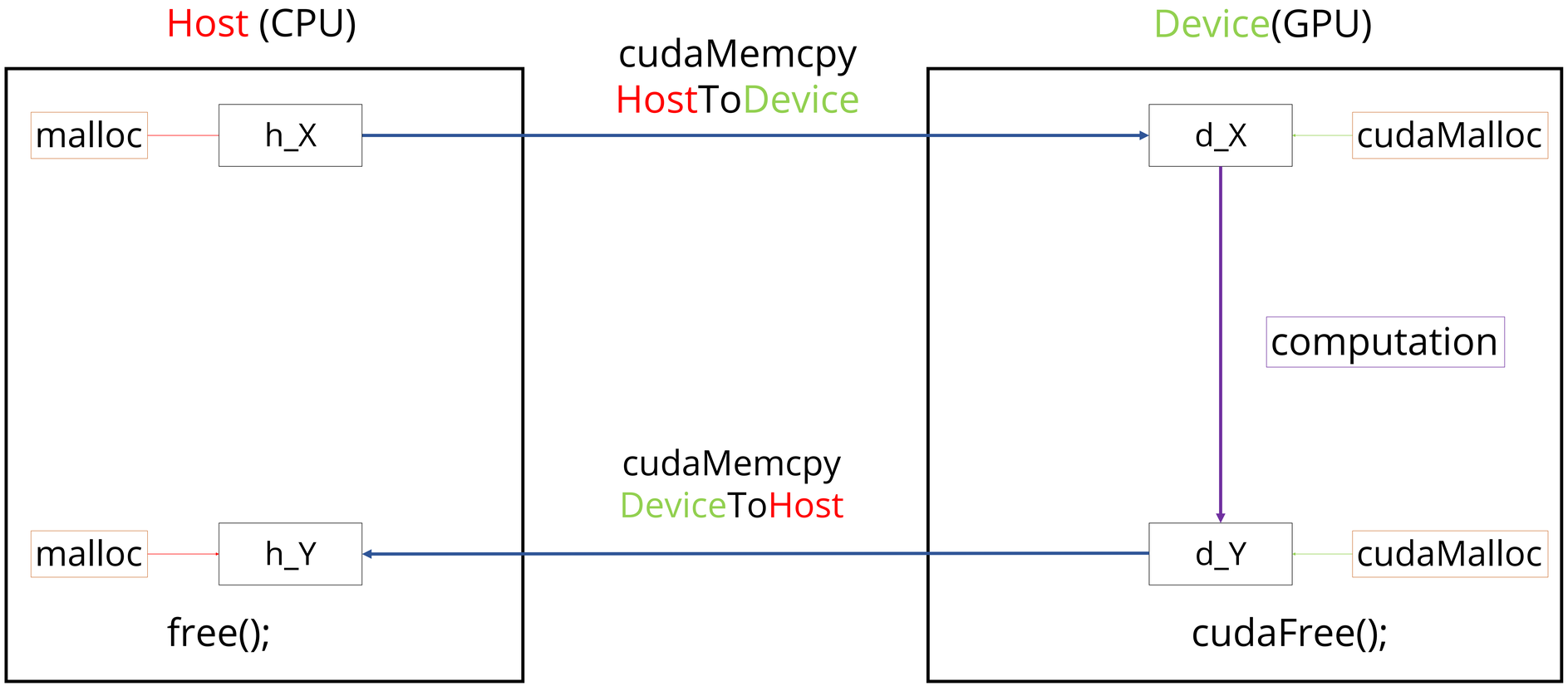
In this course, we will learn very **briefly** about the **second** way.

Before studying **CUDA**, we will get familiar to convolution programming in **CPU** using **C/C++**.
After that, we are going to learn how to do **parallel** programming in **GPU** by **CUDA**.

CUDA provides many extensions to programming language including **C/C++**, **Python**, **Java**, **Ruby**, etc.
We will use **CUDA C/C++.**

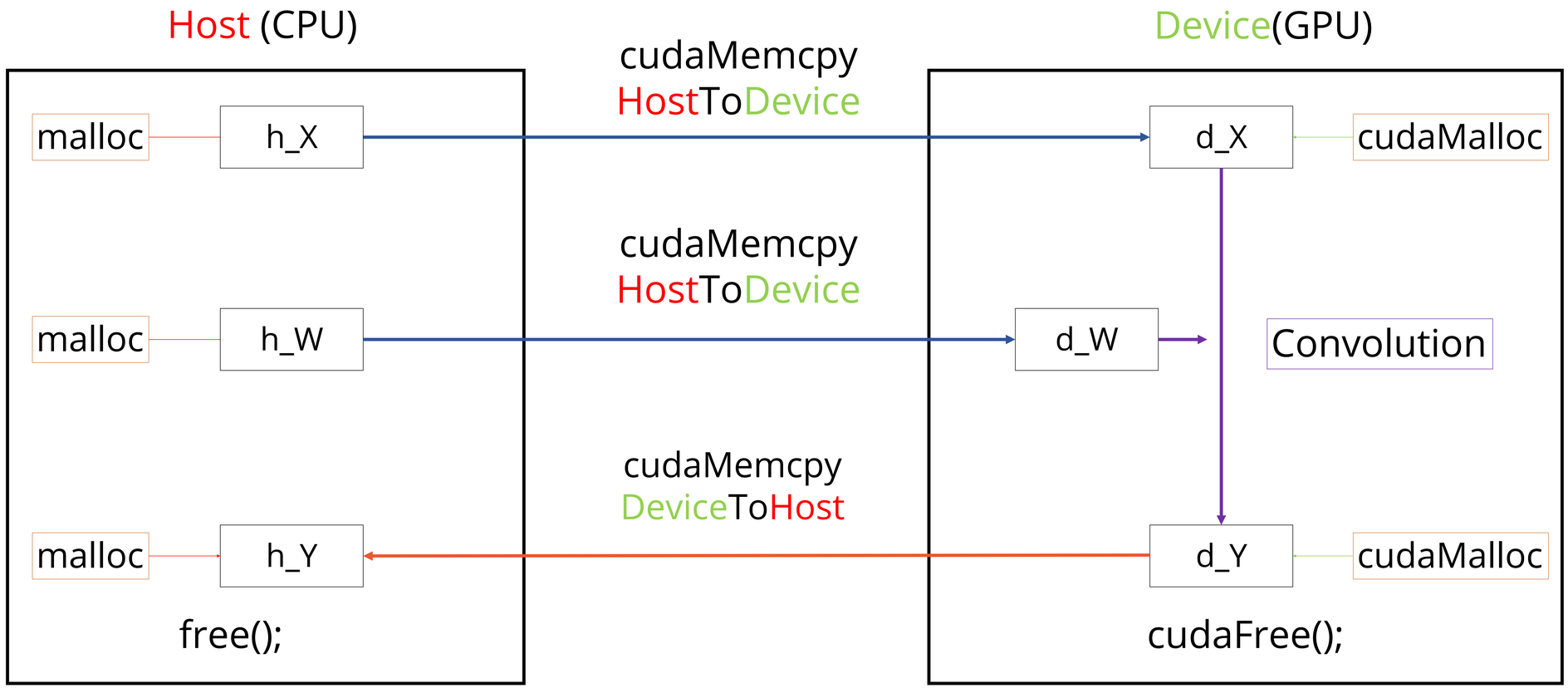
Computation on GPU

1. allocate in CPU
2. allocate in GPU
2. copy data to GPU
3. compute on GPU
4. copy data to CPU
5. free and cudaFree



Convolution on GPU

In convolution, we need data and weight files.



cuDNN Convolution

In cuDNN, there are many arguments to set.

```
cudnnConvolutionForward(cudnnHandle_t handle,
                        const void *alpha,
                        const cudnnTensorDescriptor_t xDesc,
                        const void *x,
                        const cudnnFilterDescriptor_t wDesc,
                        const void *w,
                        const cudnnConvolutionDescriptor_t convDesc,
                        cudnnConvolutionFwdAlgo_t algo,
                        void *workSpace,
                        size_t workSpaceSizeInBytes,
                        const void *beta,
                        const cudnnTensorDescriptor_t yDesc,
                        void *y);
```

Before convolution, you have to

set Handle
set TensorDescriptor
set FilterDescriptor
set ConvolutionDescriptor
set ConvolutionFwdAlgo
set workspace
set alpha, beta

You should tell cuDNN everything.

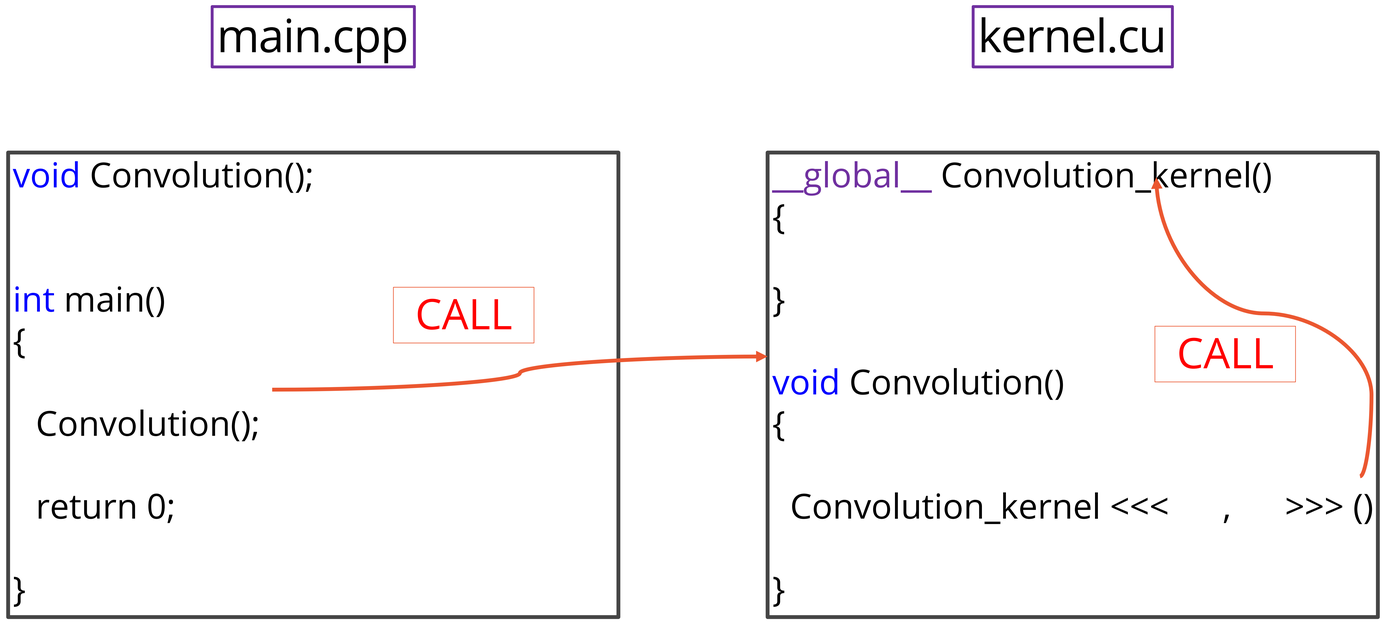
This is a code for convolution by cuDNN.

Why is it so complicated?

```
cudnnHandle_t HANDLE;
cudnnStatus_t error_create_HANDLE = cudnnCreate(&HANDLE);
if (error_create_HANDLE != CUDNN_STATUS_SUCCESS) {
printf("cudnnCreate error!\n");
exit(-1);
}

cudnnTensorDescriptor_t imageDesc;
cudnnStatus_t error_create_imageDesc = cudnnCreateTensorDescriptor(&imageDesc);
if (error_create_imageDesc != CUDNN_STATUS_SUCCESS) {
printf("cudnnCreateTensorDescriptor error!\n");
exit(-1);
}

cudnnStatus_t error_set_imageDesc = cudnnSetTensor4dDescriptor(
imageDesc,
dataformat,
datatype,
N,
C,
H,
W);
if (error_set_imageDesc != CUDNN_STATUS_SUCCESS) {
printf("cudnnSetTensor4dDescriptor error!\n");
exit(-1);
}

cudnnFilterDescriptor_t kernelDesc;
cudnnStatus_t error_create_kernelDesc = cudnnCreateFilterDescriptor(&kernelDesc);
if (error_create_kernelDesc != CUDNN_STATUS_SUCCESS) {
printf("cudnnCreateFilterDescriptor error!\n");
exit(-1);
}

cudnnStatus_t error_set_kernelDesc = cudnnSetFilter4dDescriptor(
kernelDesc,
datatype,
dataformat,
K,
C,
kH,
kW);
if (error_set_kernelDesc != CUDNN_STATUS_SUCCESS) {
printf("cudnnSetFilter4dDescriptor error!\n");
exit(-1);
}

cudnnConvolutionDescriptor_t convDesc;
cudnnStatus_t error_create_convDesc = cudnnCreateConvolutionDescriptor(&convDesc);
if (error_create_convDesc != CUDNN_STATUS_SUCCESS) {
printf("cudnnCreateConvolutionDescriptor error!\n");
exit(-1);
}

cudnnStatus_t error_set_convDesc = cudnnSetConvolution2dDescriptor(
convDesc,
pH,
pW,
sH,
sW,
dH,
dW,
conv_mode,
datatype);
if (error_set_convDesc != CUDNN_STATUS_SUCCESS) {
printf("cudnnSetConvolution2dDescriptor error!\n");
exit(-1);
}

cudnnTensorDescriptor_t conv_imageDesc;
cudnnStatus_t error_create_conv_imageDesc = cudnnCreateTensorDescriptor(&conv_imageDesc);
if (error_create_conv_imageDesc != CUDNN_STATUS_SUCCESS) {
printf("cudnnCreateTensorDescriptor error!\n");
exit(-1);
}

cudnnStatus_t error_set_conv_imageDesc = cudnnSetTensor4dDescriptor(
conv_imageDesc,
dataformat,
datatype,
N,
K,
P,
Q);
if (error_set_conv_imageDesc != CUDNN_STATUS_SUCCESS) {
printf("cudnnSetTensor4dDescriptor error!\n");
exit(-1);
}

cudnnConvolutionFwdAlgo_t conv_algo;
cudnnConvolutionFwdPreference_t preference = CUDNN_CONVOLUTION_FWD_PREFER_FASTEST;
size_t memoryLimitInBytes = 0;

cudnnStatus_t error_cudnnGetConvolutionForwardAlgorithm = cudnnGetConvolutionForwardAlgorithm(
HANDLE,
imageDesc,
kernelDesc,
convDesc,
conv_imageDesc,
preference,
memoryLimitInBytes,
&conv_algo);
if (error_cudnnGetConvolutionForwardAlgorithm != CUDNN_STATUS_SUCCESS) {
printf("cudnnGetConvolutionForwardAlgorithm error!\n");
exit(-1);
}

size_t sizeInBytes;
cudnnStatus_t error_cudnnGetConvolutionForwardWorkspaceSize = cudnnGetConvolutionForwardWorkspaceSize(
HANDLE,
imageDesc,
kernelDesc,
convDesc,
conv_imageDesc,
conv_algo,
&sizeInBytes);
if (error_cudnnGetConvolutionForwardWorkspaceSize != CUDNN_STATUS_SUCCESS) {
printf("cudnnGetConvolutionForwardWorkspaceSize error!\n");
exit(-1);
}

void *d_workspace_conv_fusion;
cudaMalloc((void **)&d_workspace_conv_fusion, sizeInBytes);

const float alpha_conv = 1.0f;
const float beta_conv = 0.0f;

void *d_conv_fusion;
cudaMalloc((void **)&d_conv_fusion, N * K * P * Q * sizeof(float));

cudnnStatus_t err_conv = cudnnConvolutionForward(
HANDLE,
&alpha_conv,
imageDesc,
d_image,
kernelDesc,
d_weight,
convDesc,
conv_algo,
d_workspace_conv_fusion,
sizeInBytes,
&beta_conv,
conv_imageDesc,
d_conv_fusion);
if (err_conv != CUDNN_STATUS_SUCCESS) {
printf("cudnnConvolutionForward error!\n");
exit(-1);
}
```

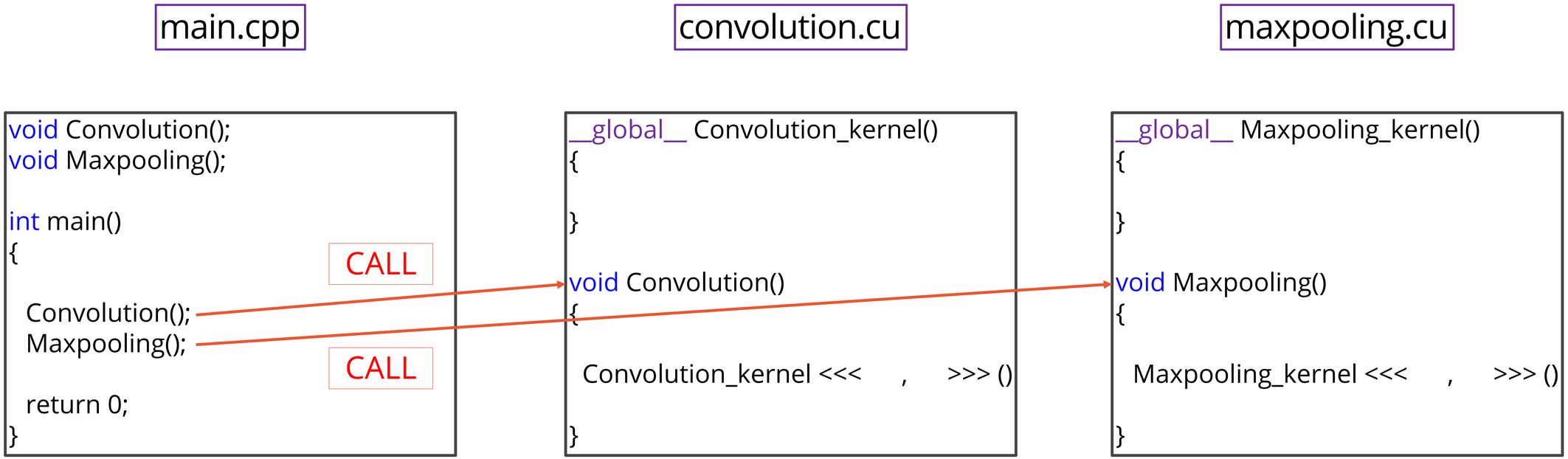
Composition of CUDA files

The main.cpp calls kernel.cu.
Host function(Convolution) calls device function(Convolution_kernel).



How to make CUDA file

more than 2 .cu files are available.



Kernel arguments

Say GPU how many threads per block and number of blocks are needed.

convolution.cu

```
__global__ Convolution_kernel()
{

}

void Convolution()
{

  Convolution_kernel <<<      ,      >>> ()

}
```

```
int THREADS_PER_BLOCK;
int TOTAL_SIZE;
int NUMBER_OF_BLOCKS = (TOTAL_SIZE + THREADS_PER_BLOCK - 1) / THREADS_PER_BLOCK;

convolution<<< NUMBER_OF_BLOCKS, THREADS_PER_BLOCK >>> ( );
```

Comparison of CPU and GPU code

GPU computes **simultaneously**

```
for (int n = 0; n < N; n++) {
 for (int k = 0; k < K; k++) {
  for (int p = 0; p < P; p++) {
   for (int q = 0; q < Q; q++) {
    float sum = 0.0f;
    for (int c = 0; c < C; c++) {
     for (int kh = 0; kh < kH; kh++) {
      int input_h_index = p * sH + kh - pT;
      if (input_h_index >= 0 && input_h_index < H) {
       for (int kw = 0; kw < kW; kw++) {
        int input_w_index = q * sW + kw - pL;
        if (input_w_index >= 0 && input_w_index < W) {
         int input_index = n * C * H * W + c * H * W + input_h_index * W +
input_w_index;
         int weight_index = k * C * kH * kW + c * kH * kW + kh * kW + kw;
         sum += weight[weight_index] * input[input_index];
        }
       }
      }
     }
    }
    int output_index = n * K * P * Q + k * P * Q + p * Q + q;
    sum += bias[k];
    output[output_index] = sum;
   }
  }
 }
}
```

CPU code

```
int tid = blockIdx.x * blockDim.x + threadIdx.x;
float sum = 0.0f;
for (int c_idx = 0; c_idx < C; c_idx++) {
 for (int kh_idx = 0; kh_idx < kH; kh_idx++) {
  int h_idx = p_idx * sH + kh_idx - pH;
  if (h_idx >= 0 && h_idx < H) {
   for (int kw_idx = 0; kw_idx < kW; kw_idx++) {
    int w_idx = q_idx * sW + kw_idx - pW;
    if (w_idx >= 0 && w_idx < W) {
     int input_index = n_idx * C * H * W + c_idx * H * W + h_idx * W + w_idx;
     int weight_index = k_idx * C * kH * kW + c_idx * kH * kW + kh_idx * kW +
kw_idx;
     sum += input[input_index] * weight[weight_index];
    }
   }
  }
 }
}
sum += bias[k_idx];
output[tid] = sum;
```

GPU code

Summary

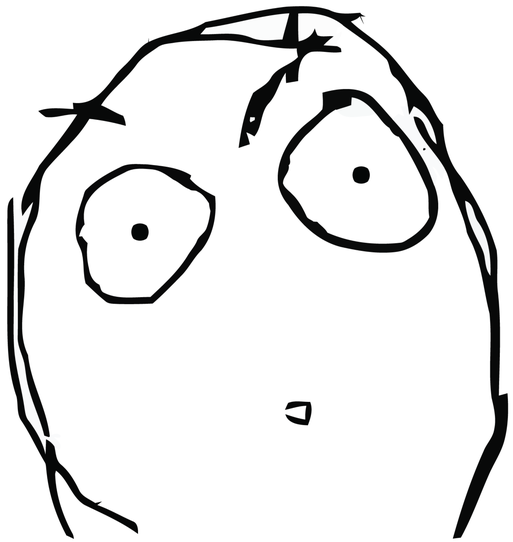
What have we learned about GPU?

We get used to **how to** make CUDA code!
This makes model **fast** and **light**.

Let's do CUDA.

The things you nedd to learn more

If you want to more optimize GPU, you should learn....



Unified Memory

Shared Memory

Asynchrounous operation

Mutil Threads

Multi Devices

Multi-dimensional indexing

Device Mangement

Occupancy

Reduction

Seung Hwan Baek