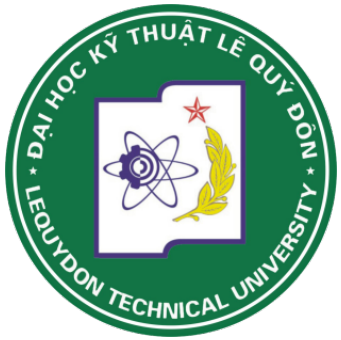
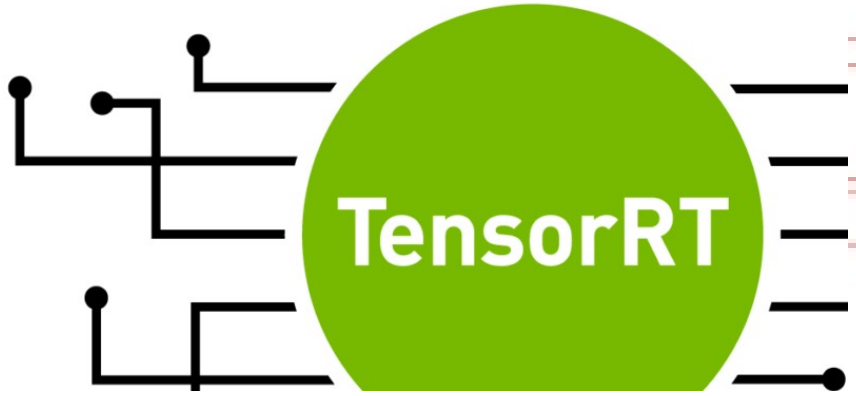
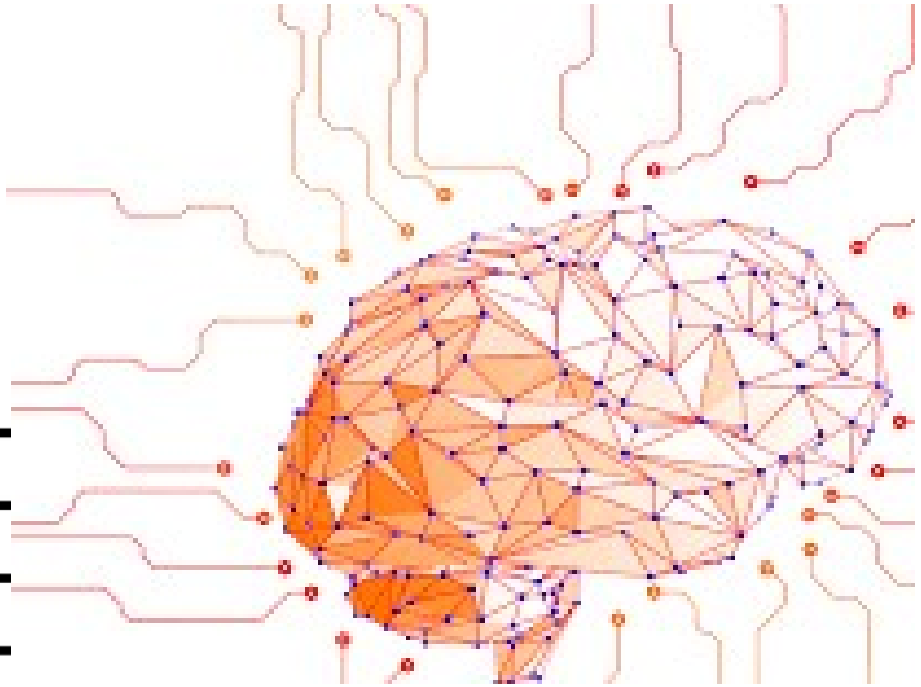




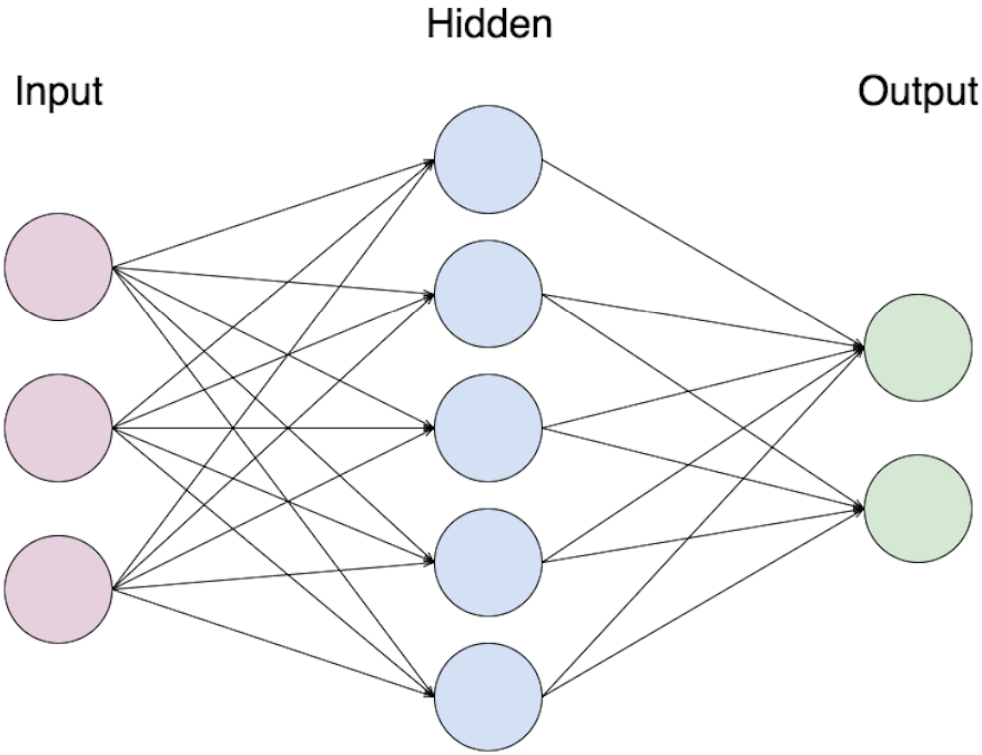


Lý thuyết và thực nghiệm

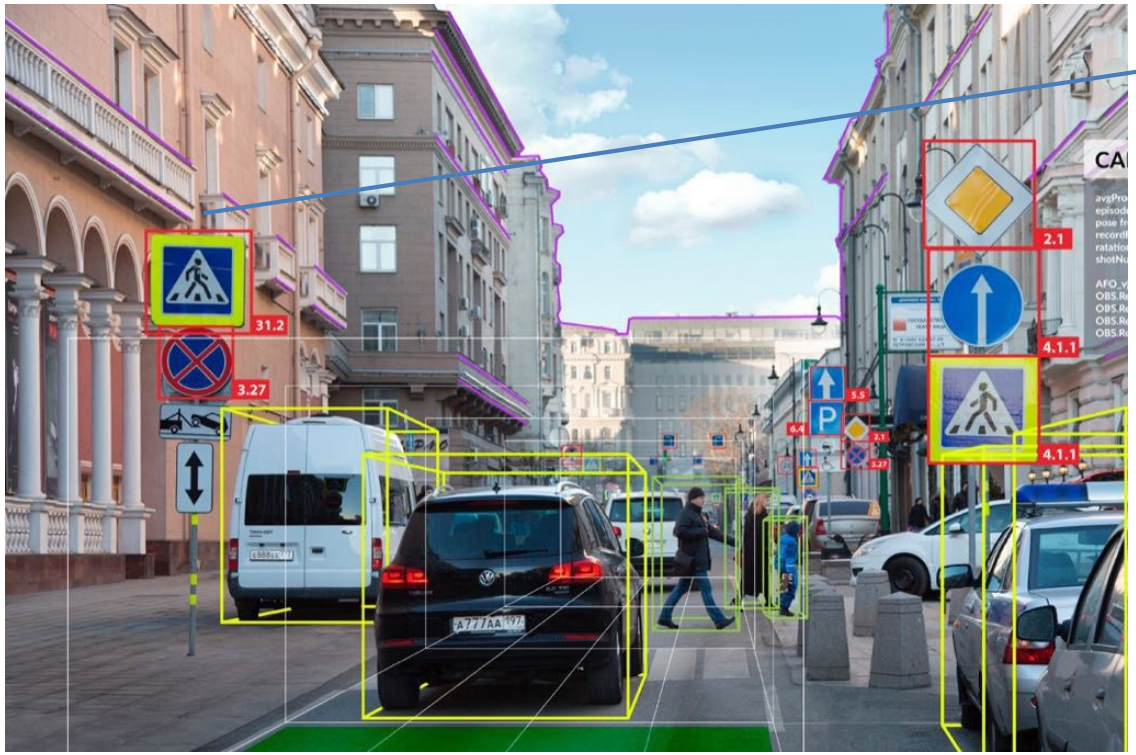
Nội dung

- **1. Mạng Neural và ứng dụng trong thị giác máy tính**
- **2. Trích chọn đặc trưng và nhân chập**
- **3. Mạng Neural nhân chập và các mô hình học sâu**
- **4. TensorRT**
- **5. Cơ chế tăng tốc của TensorRT**
- **6. Cách thức sử dụng TensorRT phù hợp**
- **7. Demo**

Mạng Neural

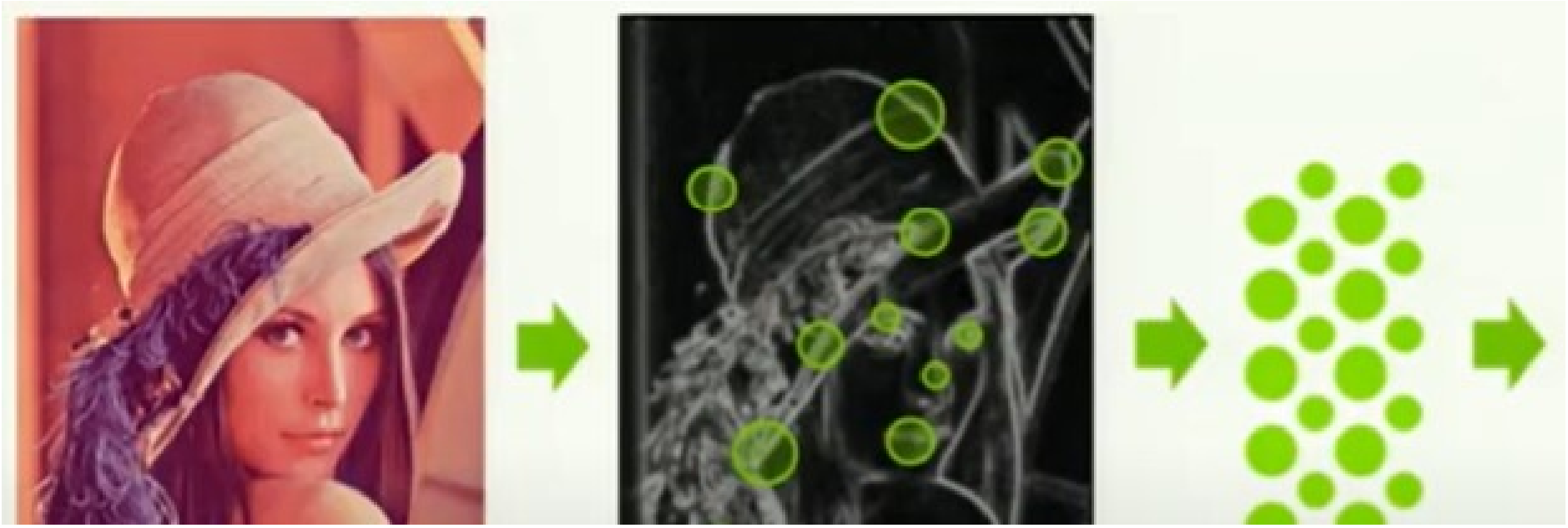


Nhận dạng đối tượng có đơn giản?



Biển báo người đi bộ

Máy nhận dạng thông qua đặc trưng

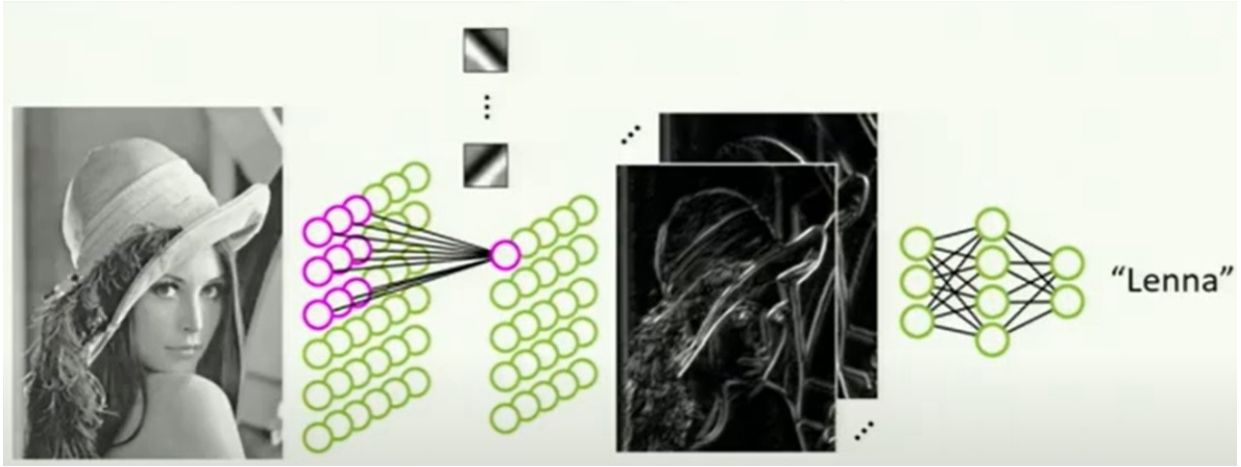


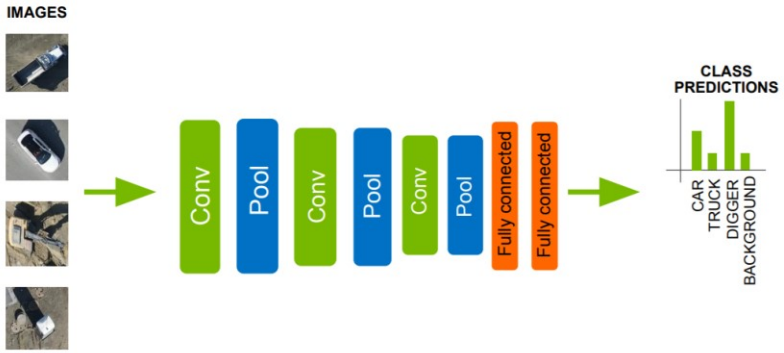
Phép nhân chập

$$y[n] = x[n] * h[n] = \sum_{k=-\infty}^{\infty} x[k]h[n-k]$$

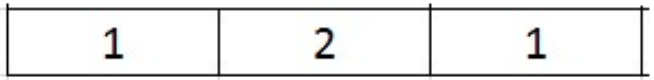
Mạng Neural nhân chập

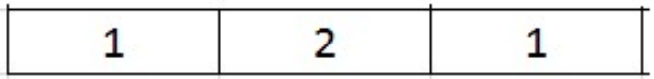
- Phép nhân chập tuyến tính-> tổng hợp dựng các lớp cho mạng Neural
- Các đặc trưng được phát hiện qua quá trình huấn luyện





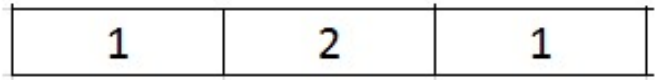
Phép nhân chập và cửa sổ trượt

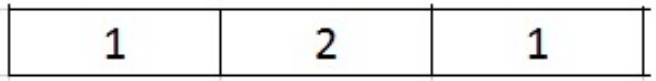


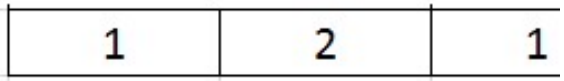


$$\mathcal{F}\{f * g\} = \mathcal{F}\{f\} \cdot \mathcal{F}\{g\}$$

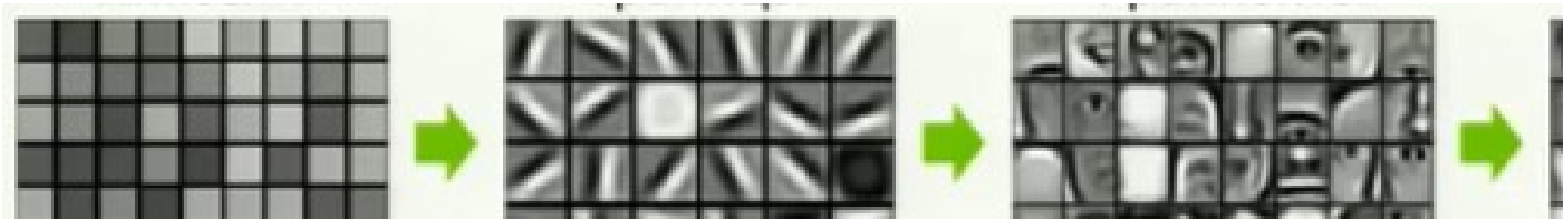
$$f * g = \mathcal{F}^{-1}\{\mathcal{F}\{f\} \cdot \mathcal{F}$$



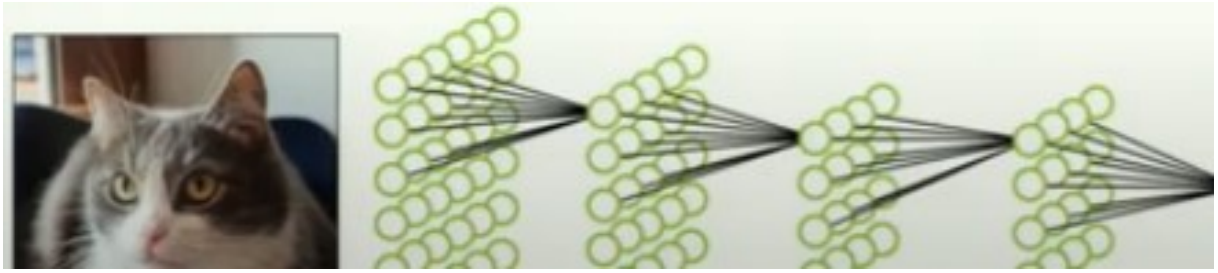




Tổ hợp các đặc trưng



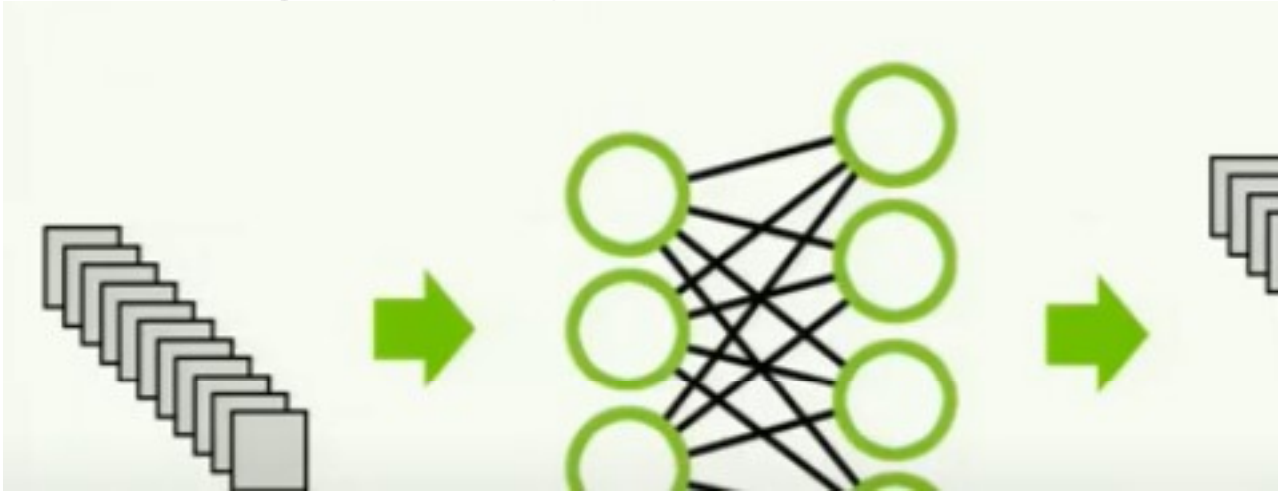
Điểm ảnh Phân tích theo cạnh Phân vùng Đối tượng



Lớp tích chập

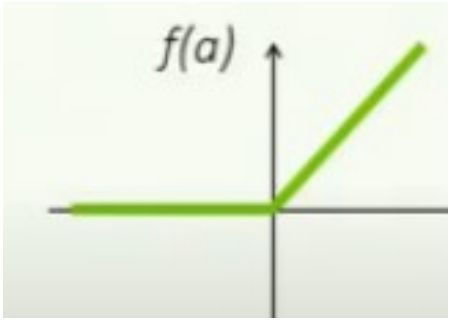
Giảm chiều dữ liệu - Pooling

Tầng Fully Connected

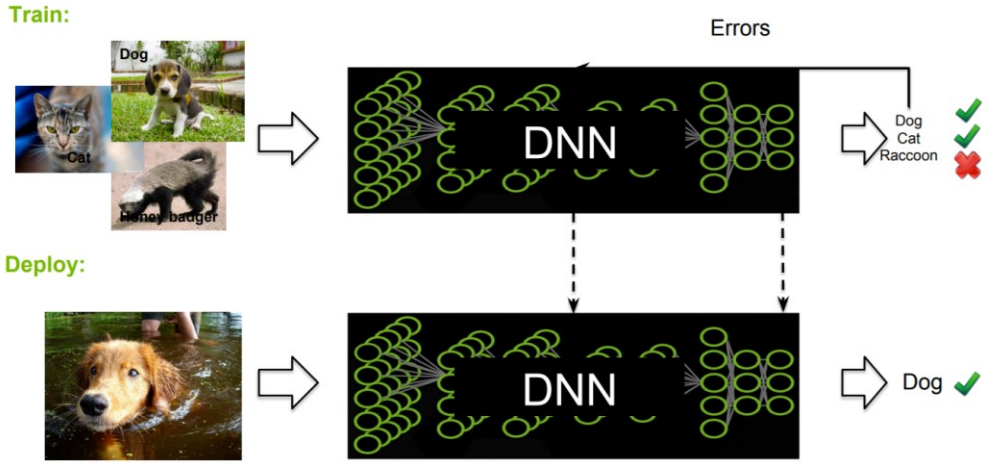


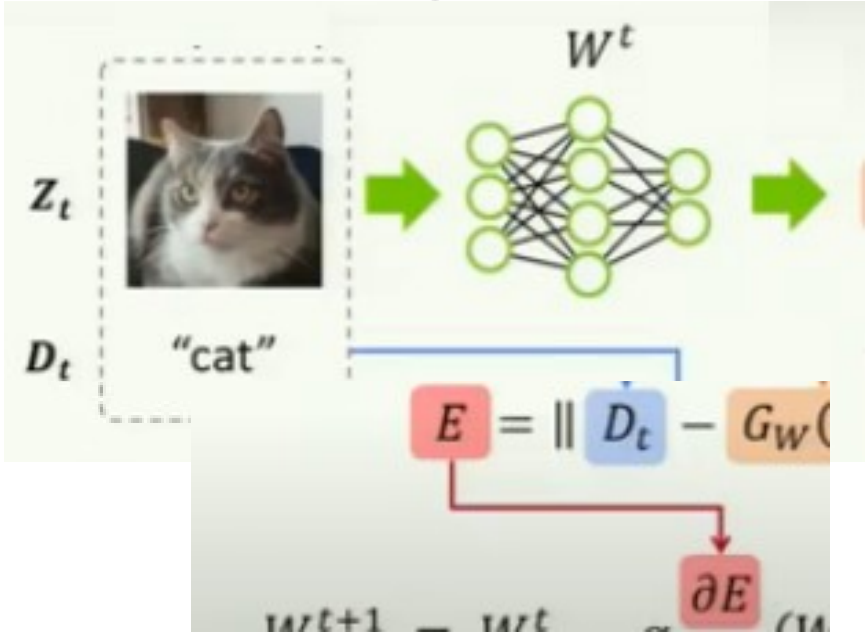
Hàm kích hoạt- Activation function

- ReLU
- Heaviside
- Logistic
- Một số kỹ thuật kết hợp: Dropout, Batch normalization

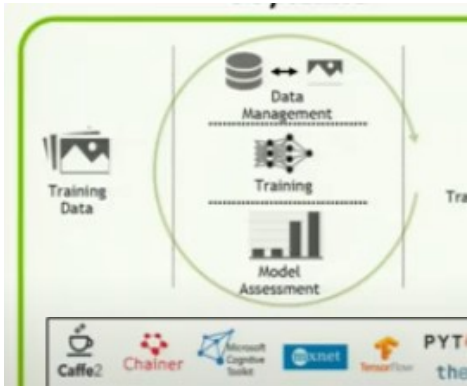


Huấn luyện và sử dụng

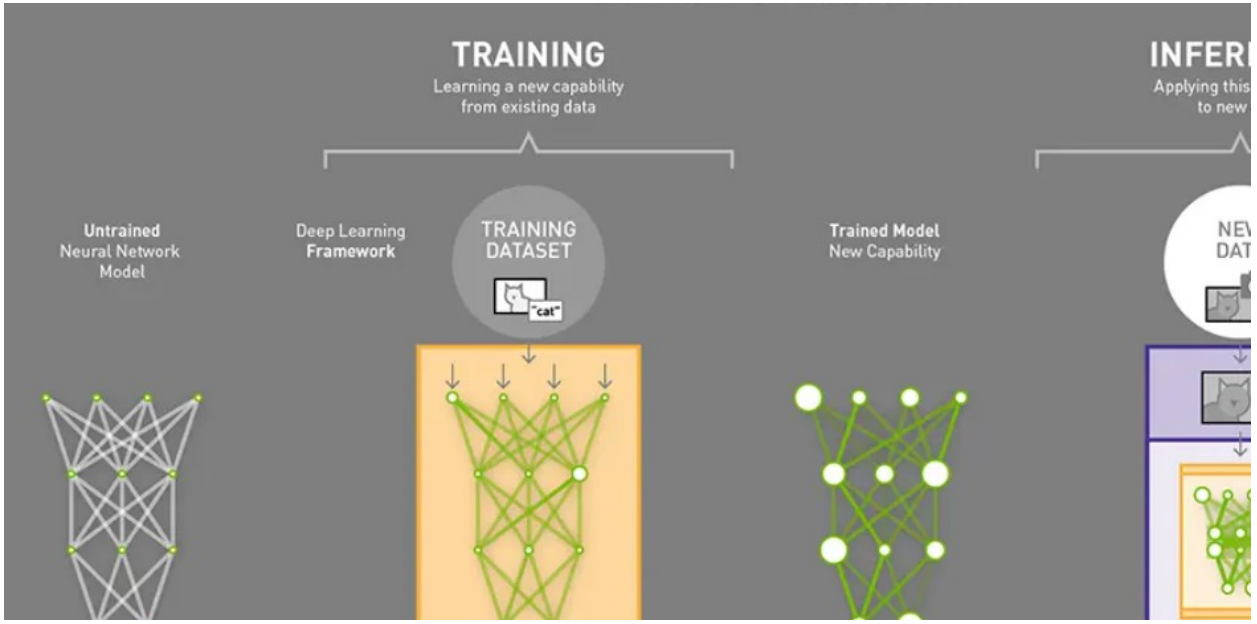




Inference

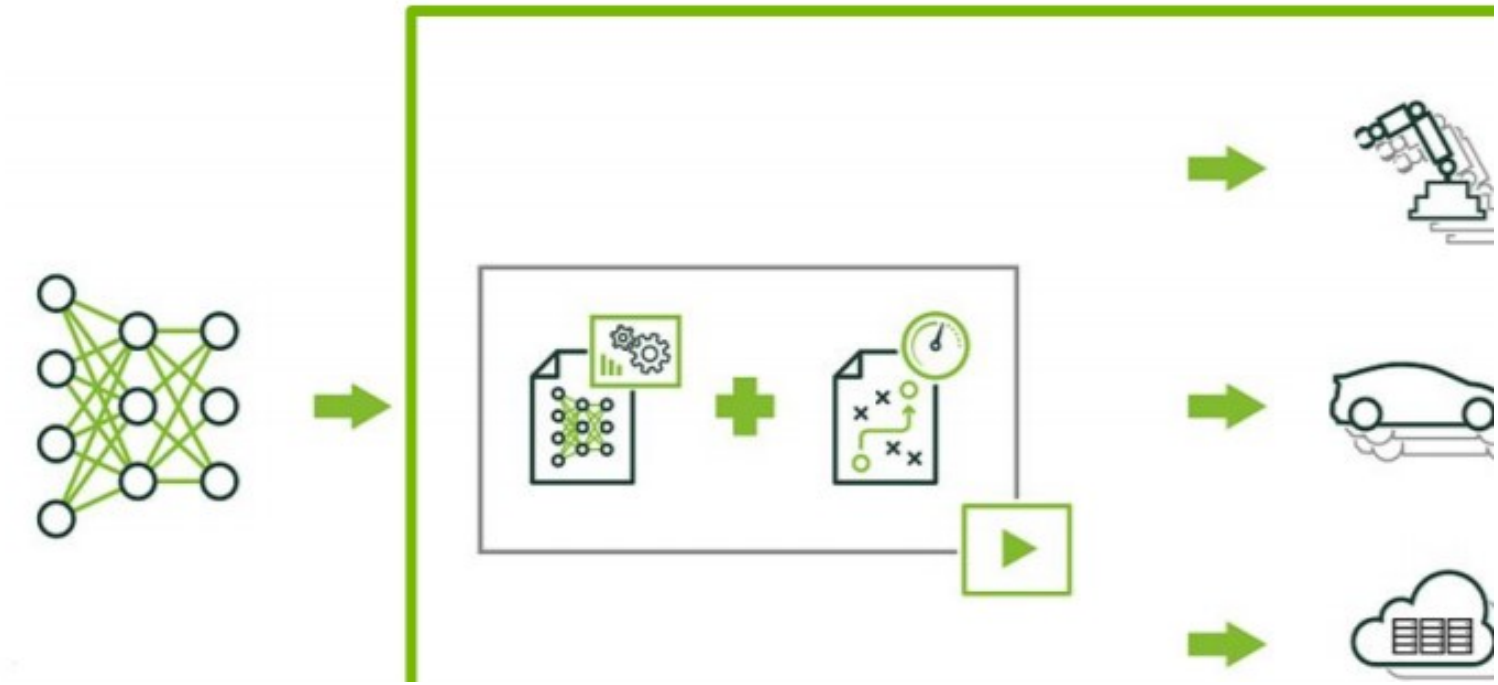


Thư viện cuDNN chứa các toán tử cơ bản của Deep Learning

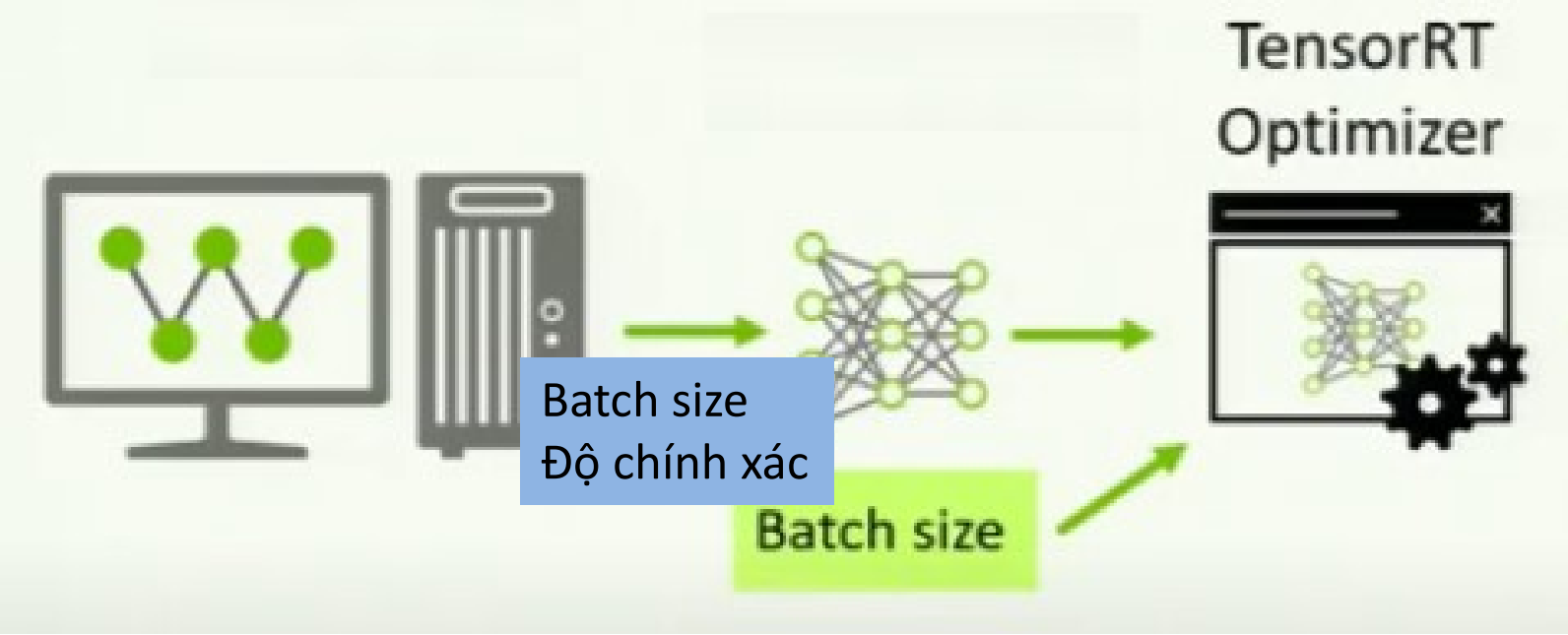


"The NVIDIA CUDA® Deep Neural Network library (cuDNN) is a GPU-accelerated library of primitives for deep neural networks. cuDNN provides highly tuned implementations for standard routines such as forward and backward convolution, pooling, normalization, and activation layers."

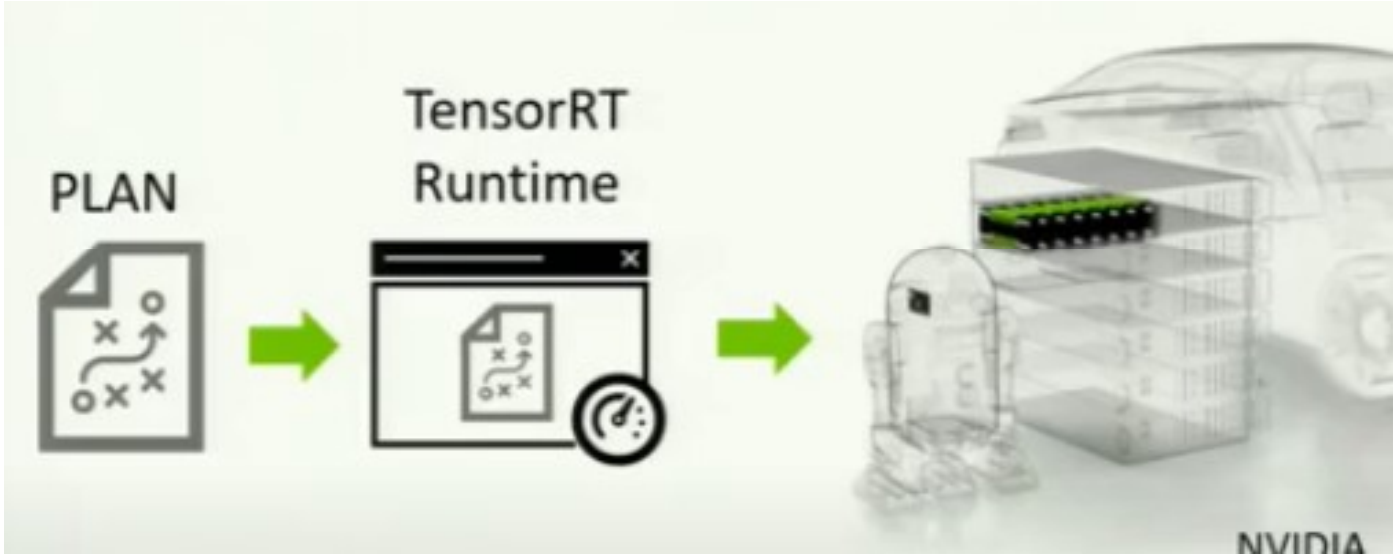
TensorRT là gì



Bước 1 - Optimization



Bước 2 - Inference

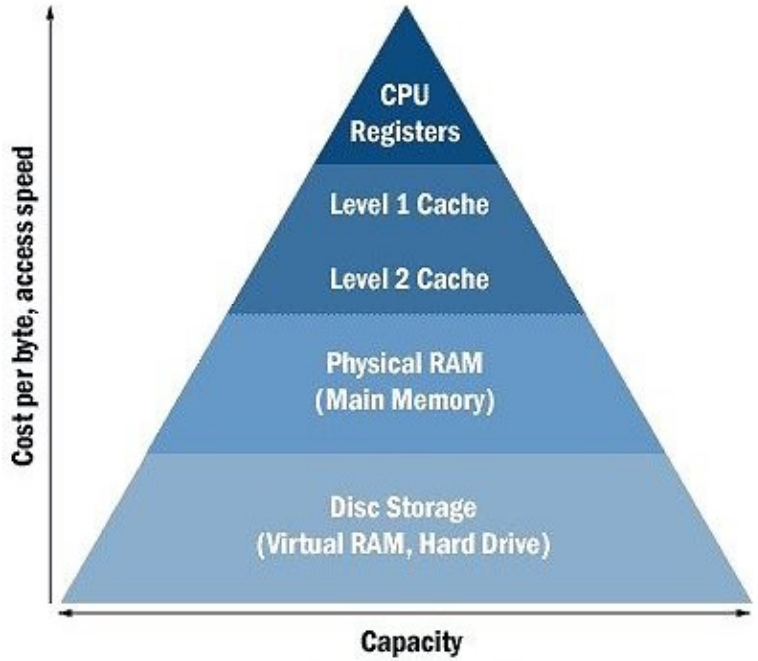


Import model

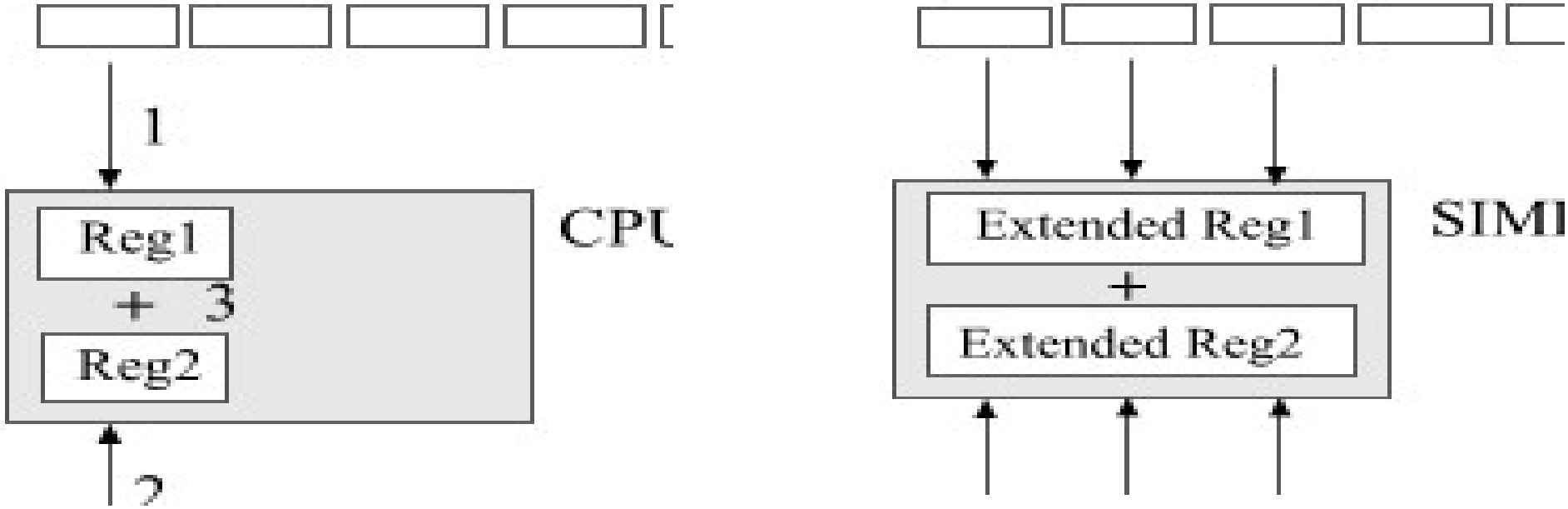
Các mô hình tính toán song song

- SIMD/Vector parallelism
- **Multi-core/multi-thread parallelism**
- Distributed computing

Vấn đề Truy xuất bộ nhớ

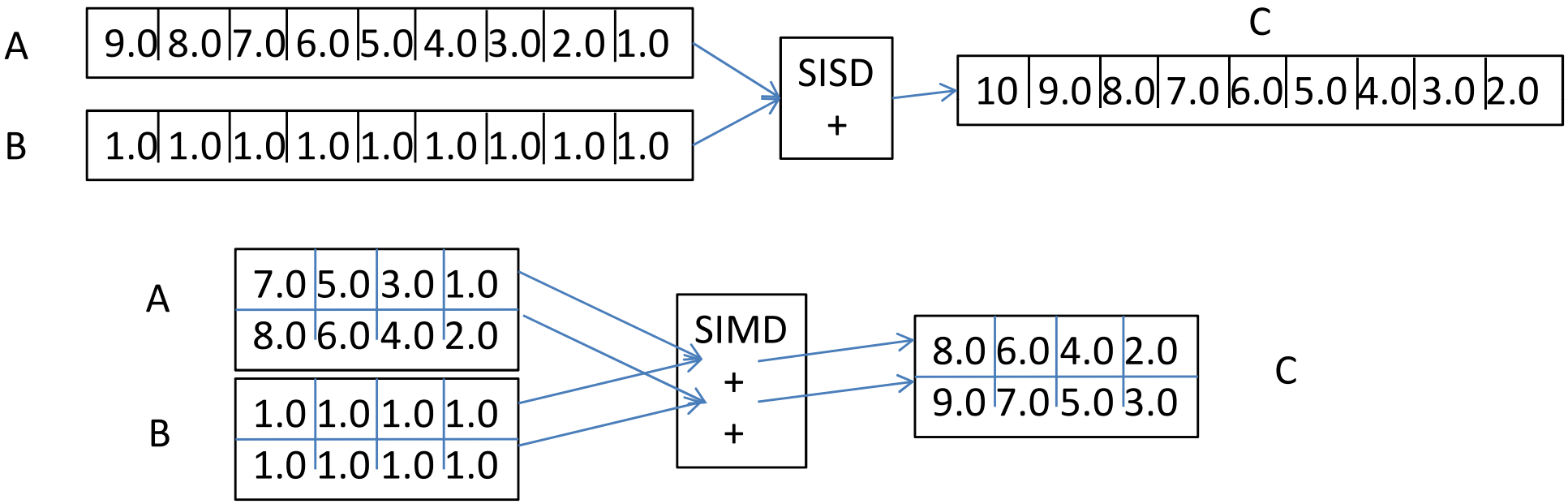


SIMD -**Single instruction, multiple data**



Khác biệt SISD và SIMD

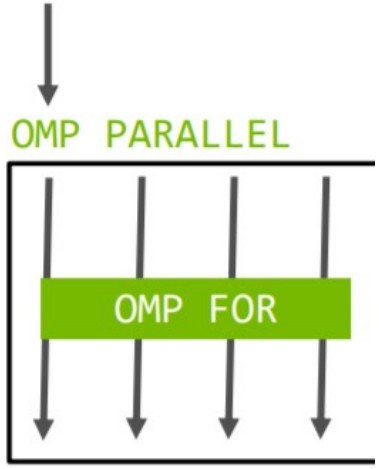
- C = A + B
  - For (i=0;i<n; i++) c[i] = a[i] + b[i]



SIMD/SSE code

```
float x[k]; float y[k]; // vectors of length k
__m128 X, Y; // 128-bit values
__m128 acc = _mm_setzero_ps(); // set to (0, 0, 0, 0)
float inner_prod, temp[4];
for(i = 0; i < k - 4; i += 4) {
X = _mm_load_ps(&x[i]); // load chunk of 4 floats
Y = _mm_load_ps(y + i); // alternate way, pointer arithmetic
acc = _mm_add_ps(acc, _mm_mul_ps(X, Y));
}
_mm_store_ps(&temp[0], acc); // store acc into an array of floats
inner_prod = temp[0] + temp[1] + temp[2] + temp[3];
// add the remaining values
for(; i < k; i++)
inner_prod += x[i] * y[i];
```

OpenMP



- error = 0.0;
- #pragma omp parallel for reduction(max:error)
- for(int j = 1; j < n-1; j++) {

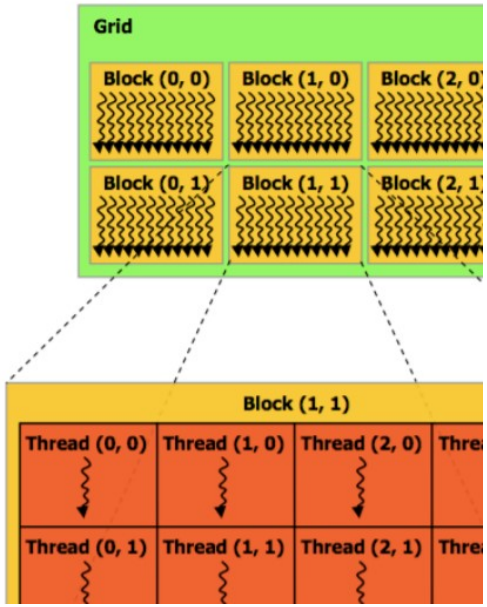
```
  for( int i = 1; i < m-1; i++ ) {
    Anew[j][i] = 0.25 * ( A[j][i+1] + A[j][i-1] + A[j-1][i] + A[j+1][i]);
     error = fmax( error, fabs(Anew[j][i] - A[j][i])); } }
```

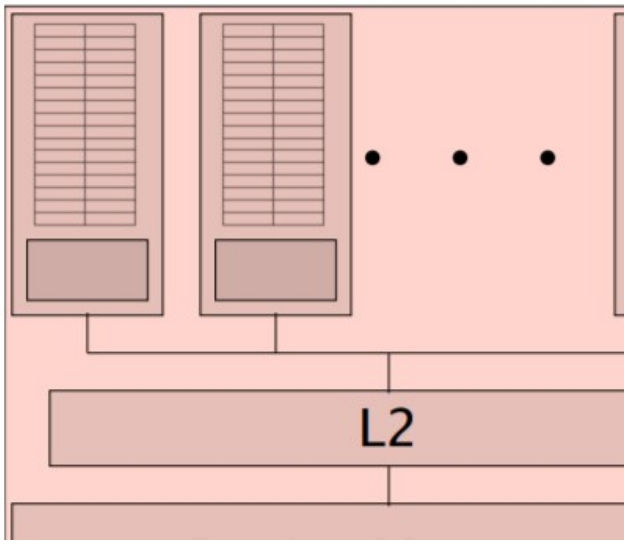
Tính toán trên GPU

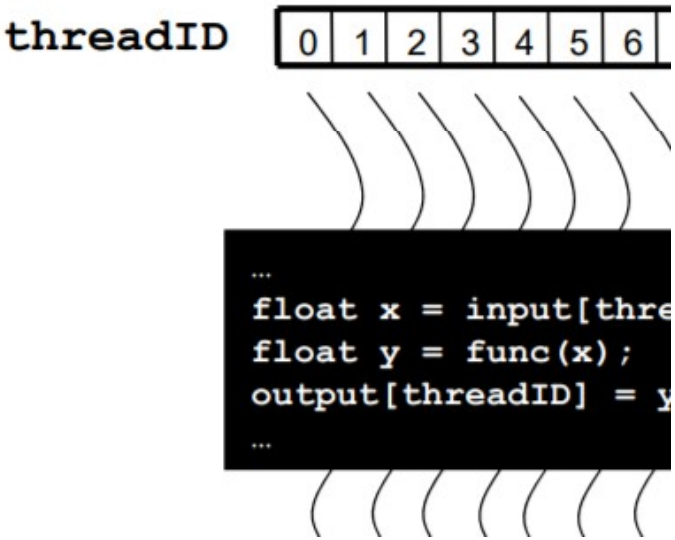
```
// Kernel definition
__global__ void VecAdd(float* A, float* B,
{
    int i = threadIdx.x;
    C[i] = A[i] + B[i];
}

int main()
{
    ...
    // Kernel invocation with N threads
```

CUDA

- Compute Unified Device Architecture



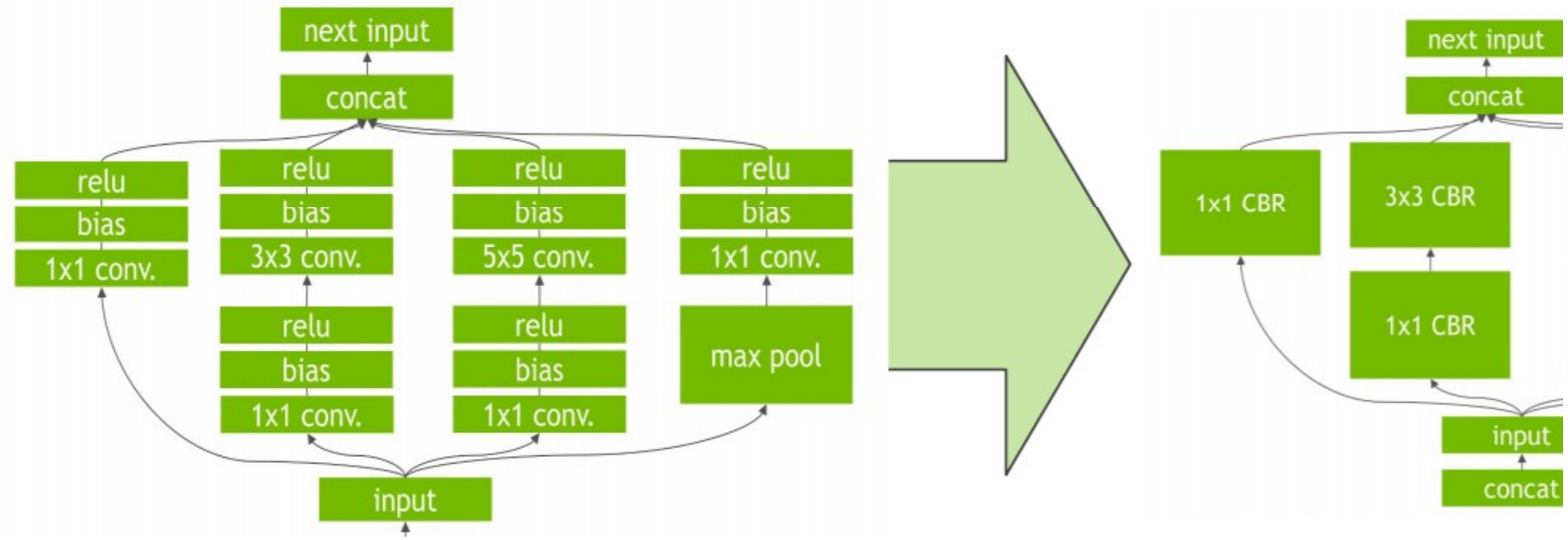




TensorRT Optimizer hỗ trợ những đối tượng tính toán nào?

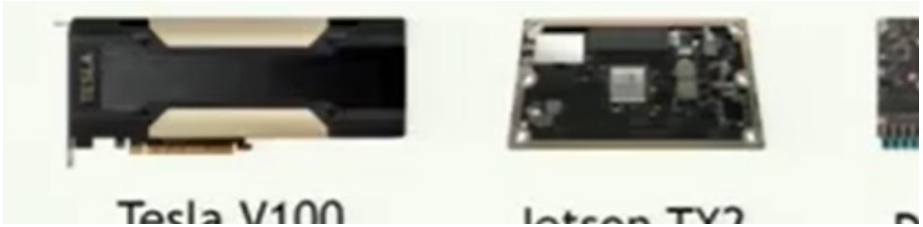
- Convolution: 2D
- Activation: ReLU, tanh and sigmoid
- Pooling: max and average
- ElementWise: sum, product or max of two tensors
- LRN: cross-channel only
- Fully-connected: with or without bias
- SoftMax: cross-channel only
- Deconvolution

TensorRT làm thế nào để tăng tốc



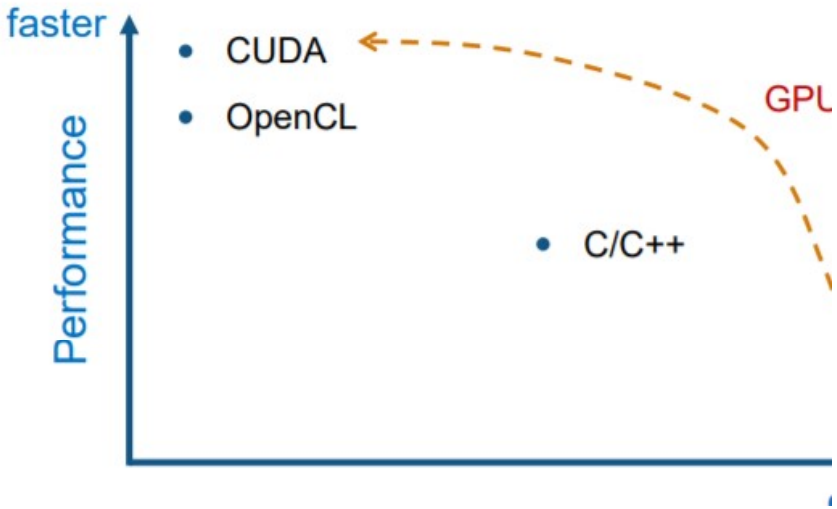
Lựa chọn thiết bị và tham số

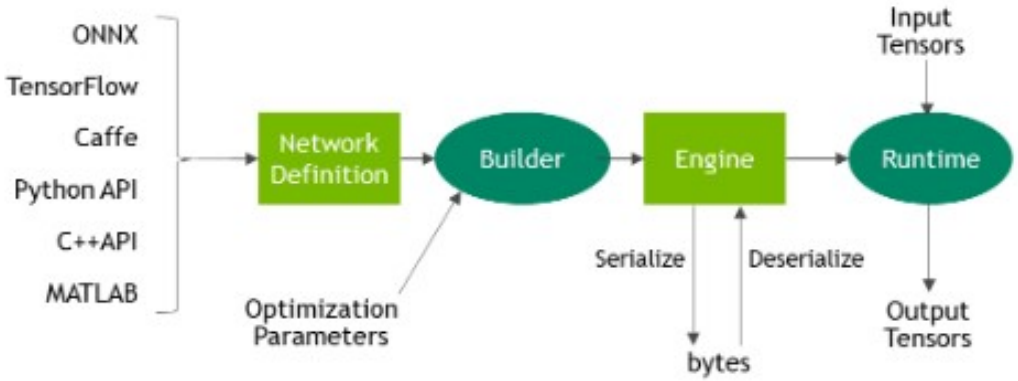
- Lựa chọn Batch size
- Kích thước đầu vào
- Trọng số



Môi trường lập trình

- CUDA® enabled NVIDIA® GPU
- NVIDIA cuDNN and TensorRT library





Ví dụ Chuyển đổi code

Scalarized MATLAB

```
for i = 1:length(x)
    z(i) = a .* x(i) + y(i);
end
```

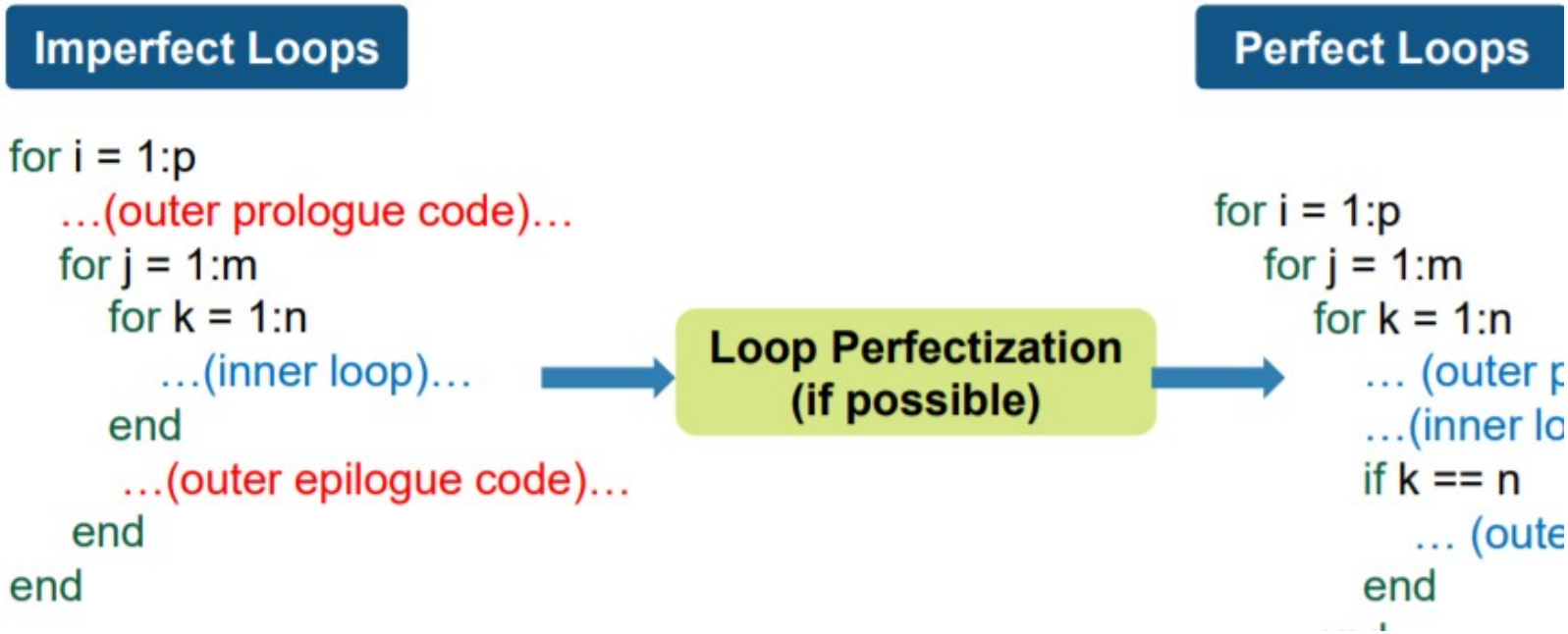
GPU Coder

Vectorized MATLAB

```
z = a .* x + y;
```

```
static __global__ __launch_bounds__(512, 1)
  const real32_T *x, real32_T a, real_T *z)
{
  int i = blockIdx.x*blockDim.x + threadIdx.
  if (!(i >= 1048576)) {
    z[i] = (real_T)(a * x[i] + y[i]);
  }
}

void saxpy(real32_T a, const real32_T x[1048
           real_T z[1048576])
{
  real32_T *gpu_y;
  real32_T *gpu_x;
  real_T *gpu_z;
  cudaMalloc(&gpu_z, 8388608UL);
  cudaMalloc(&gpu_x, 4194304UL);
  cudaMalloc(&gpu_y, 4194304UL);
  cudaMemcpy((void *)gpu_y, (void *)&y[0], 4
  cudaMemcpy((void *)gpu_x, (void *)&x[0], 4
  saxpy_kernel1<<<dim3(2048U, 1U, 1U), dim3(
    gpu_z);
```

Tối ưu code chuyển đổi

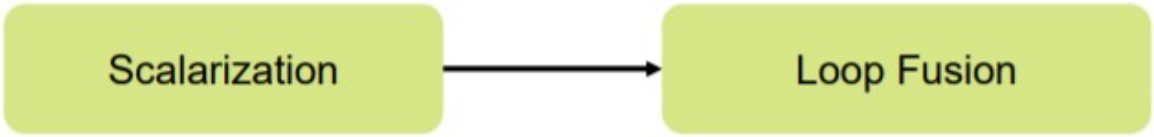


Vector hóa tổ chức dữ liệu

```
output(:, 1) = (input(:, 1) – x_im) .* factor;
```

```
for i = 1:M
   diff(i) = input(i, 1) – x_im(i);
end
for a = 1:M
   output(i, 1) = diff(i) * factor(i);
end
```

```
for i = 1:M
   diff(i) = input(i, 1) – x_im(i);
   output(i, 1) = diff(i) * factor(i);
end
```

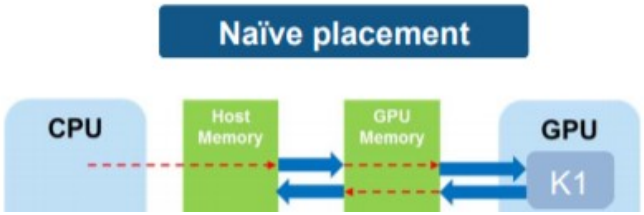


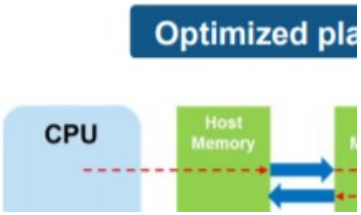
Tổ chức trao đổi dữ liệu giữa các vùng bộ nhớ

```
A = …
…
for i = 1:N
   … A(i)
end
…

imfilter
…
```

```
A = …
…
cudaMemc
kernel1<
cudaMemc
…
cudaMemc
imfilter
cudaMemc
…
```

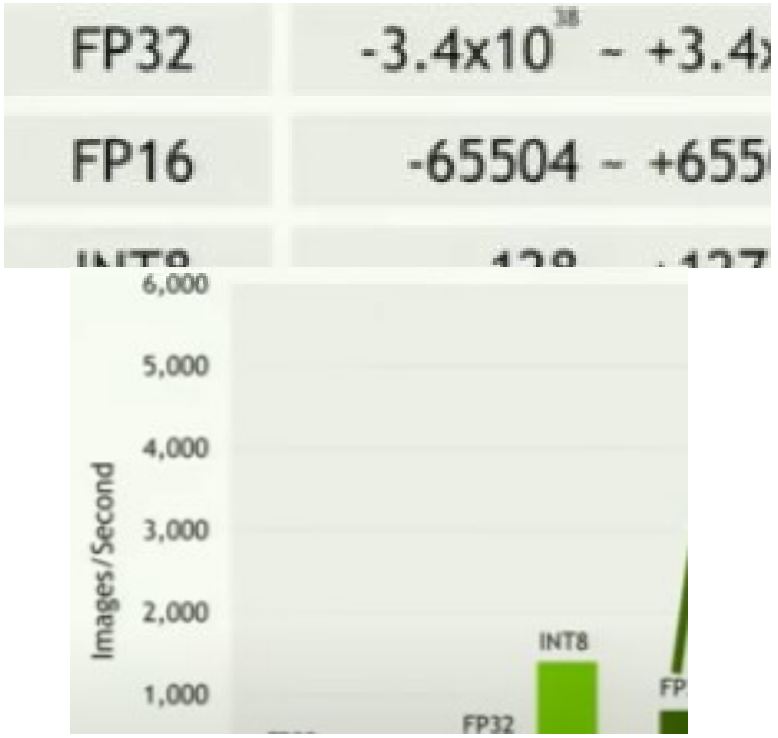
Where is the ideal placement of memcpy





Định dạng dữ liệu

- Type: FP32 (32-bit floating point, or single precision), FP16 (16-bit floating point, or half precision), INT32 (32-bit integer representation) and INT8 (8-bit representation).



Tiếp theo

DEMO

Tài liệu tham khảo

- https://developer.nvidia.com/blog/speed-up-inference-tensorrt/
- https://rse.shef.ac.uk/assets/slides/2018-07-19-dl-cv/deployment.pdf
- https://docs.nvidia.com/deeplearning/tensorrt/sample-support-guide/index.html
- https://docs.nvidia.com/deeplearning/tensorrt/developer-guide/
- https://on-demand.gputechconf.com/gtc/2018/presentation/s8480-gpu-coder-automatic-cuda-and-tensorrt-code-generation-from-matlab.pdf
- https://www.highload.ru/2017/abstracts/2985.html