



auto output = engine->tensor("output")

int num_classes = output->size(2) - 5

input_width_ = input->size(3)

input_height_ = input->size(2)

tensor_allocator_ = make_shared<MonopolyAllocator<TRT::Tensor>>(max_batch_size * 2)

result.set_value(true)

input->resize_single_dim(0, max_batch_size).to_gpu()

affin_matrix_device.set_stream(stream_)

affin_matrix_device.resize(max_batch_size, 8).to_gpu()

output_array_device.resize(max_batch_size, 1 + MAX_IMAGE_BBOX * NUM_BOX_ELEMENT).to_gpu()

vector<Job> fetch_jobs ·····

int infer_batch_size = fetch_jobs.size()

input->resize_single_dim(0, infer_batch_size)

int ibatch = 0

ibatch < infer_batch_size

++ibatch

auto& mono = job.mono_tensor->data()

affin_matrix_device.copy_from_gpu(affin_matrix_device.offset(ibatch), mono->get_workspace()->gpu(), 6);

input->copy_from_gpu(input->offset(ibatch), mono->gpu(), mono->count());

_____****

engine->forward(false)

output_array_device.to_gpu(false)

int ibatch = 0

▶ batch < infer_batch_size

auto& job = fetch_jobs[ibatch]

float* image_based_output = output->gpu<float>(ibatch)

float* output_array_ptr = output_array_device.gpu<float>(ibatch)

auto affine_matrix = affin_matrix_device.gpu<float>(ibatch)

checkCudaRuntime(cudaMemsetAsync(output_array_ptr, 0, sizeof(int), stream_))

Yolo::decode_kernel_invoker(image_based_

affine_matrix, output_array_ptr, MAX_IMAGE_BBOX, stream_)

ibatch < infer_batch_size

float* parray = output_array_device.cpu<float>(ibatch)

int count = min(MAX_IMAGE_BBOX, (int)*parray)

auto& job = fetch_jobs[ibatch]

auto image_based_boxes = make_shared<BoxArray>()

float* pbox = parray + 1 + i * NUM_BOX_ELEMENT

int label = pbox[5]

int keepflag = pbox[6]

image_based_boxes->emplace_back(pbox[0], pbox[1], pbox[2], pbox[3], pbox[4], label)

job.pro->set_value(image_based_boxes)

output, output->size(1), num_classes, confidence_threshold_, nms_threshold_,

job.mono_tensor->release()

auto& job = fetch_jobs[ibatch]

job.mono_tensor = tensor_allocator_->query()

CUDATools::AutoDevice auto_device(gpu_)

auto& tensor = job.mono_tensor->data()

Size input_size(input_width_, input_height_)

job.additional.compute(image.size(), input_size)

job.mono_tensor == nullptr return false

tensor == nullptr true tensor = make_shared<TRT::Tensor>()

tensor->set_workspace(make_shared<TRT::MixMemory>())