NN-PREDICTOR分享

CVApp Group, 4paradigm, Inc

Han Feng

2019.08.02

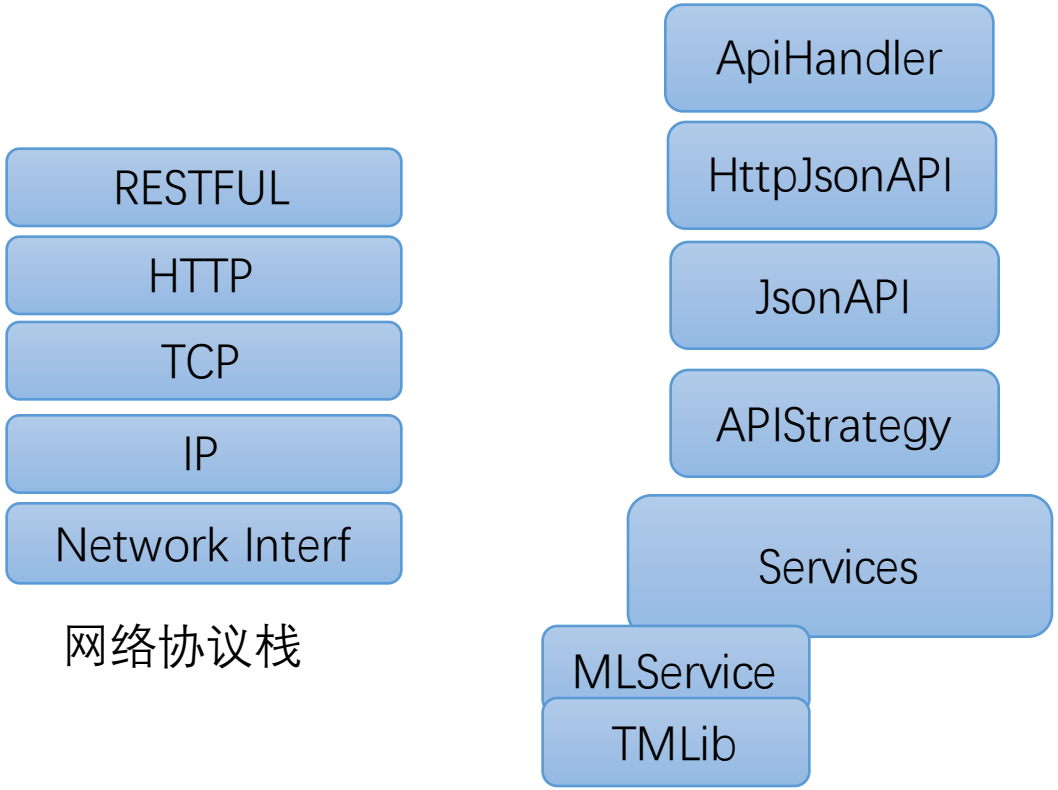
Outline

- Background & Why
- Structure Design
- Internal API Introduce – TF Session
- Internal API Introduce – OpenCV MAT
- Internal API Introduce – Servable/ServableGroup
- Application Develop flow
- Discuss

Background & Why

- Background
  - 模型应用的基础组件
  - SDK服务的基础
  - 性能，安全，个性化的应用，可扩展性
- Why
  - 开源方案的弱项：deepdetect, tf_serving, mxnet serving onnx-serving
  - 更高的性能
  - 更安全
  - 更高的可扩展性
  - 更个性化的控制：Quota,Licenese etc

Structure Design



- Services: 全局托管不同MLService实例的生存状态, 创建, 执行, 删除等, 内部持有MLService实例, 提供被管理对象统一的init/predict/train/clear调用入口。
- APIStrategy: API策略层， 继承Services类， 对外增加提供boot接口
- JsonAPI: 支持Json的API层， 对外提供boot/service_create/service_delete/service_train/service_predict接口, 输入输出数据是JDoc格式。
- HttpJsonAPI: 封装了http最外层， 由APIHandler类提供http方法的回调函数入口。
- APIHandler: http服务回调函数实现入口, 解析http:request, 执行分支回调逻辑， 填充response。

Service Start Flow

- Step 1. main/deepdetect.cc:main
- Step 2. HttpJsonAPI::boot -> (evoke JsonAPI::boot, start_server)
- Step 3. JsonApi::boot
- Step 4. JsonAPI::service_autostart

```cpp
int HttpJsonAPI::boot(int argc, char *argv[]) {
  google::ParseCommandLineFlags(&argc, &argv, true);
  std::signal(SIGINT, terminate);
  JsonAPI::boot(argc, argv);
  return start_server(FLAGS_host, FLAGS_port, FLAGS_nthreads);
}
```

```cpp
int JsonAPI::boot(int argc, char *argv[]) {
  google::ParseCommandLineFlags(&argc, &argv, true);
  if (!FLAGS_config.empty()) {
    service_autostart(FLAGS_config);
  }
  return 0;
}
```

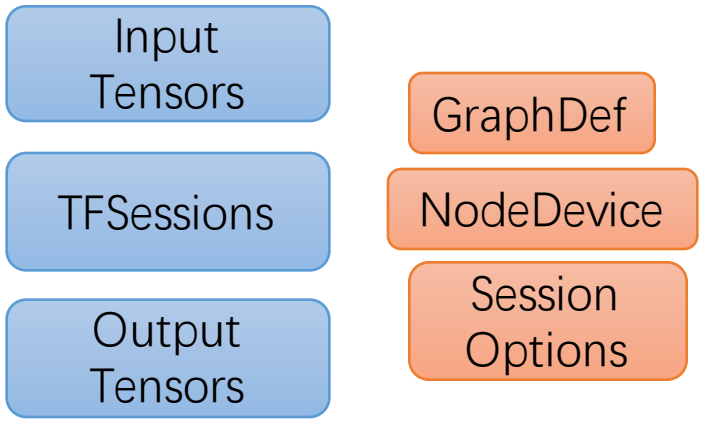
```cpp
int HttpJsonAPI::start_server(const std::string &host,
                              const std::string &port,
                              const int &nthreads) {
  APIHandler ahandler(this);
  http_server::options options(ahandler);
  _dd_server = new http_server(options.address(host)
                               .port(port)
                               .linger(false)
                               .reuse_address(true));
  _ghja = this;
  _gdd_server = _dd_server;
  _logger->info("Running NNPredictor HTTP server on {}:{}", host, port);

  if (!FLAGS_allow_origin.empty()) {
    _logger->info("Allowing origin from {}", FLAGS_allow_origin);
  }

  std::vector<std::thread> ts;
  for (int i = 0; i < nthreads; i++) {
    ts.push_back(std::thread(std::bind(&http_server::run, _dd_server)));
  }

  try {
    _dd_server->run();
  } catch (std::exception &e) {
    _logger->error(e.what());
    return 1;
  }
  for (int i = 0; i < nthreads; i++) {
    ts.at(i).join();
  }
  return 0;
}
```

```
JDoc JsonAPI::service_autostart(const std::string& config_file) {
  if (config_file.empty()) {
    return 0;
  }

  if (!fileops::file_exists(config_file)) {
    _logger->error("JSON autostart file not found: {}", config_file);
    return dd_bad_request_400();
  }

  std::ifstream ifs(config_file);
  if (!ifs.is_open()) {
    _logger->error("Failed opening JSON autostart file {}", config_file);
    return dd_internal_error_500();
  }

  rapidjson::Document d;
  std::stringstream buffer;
  buffer << ifs.rdbuf();
  std::string content = buffer.str();
  d.Parse(content.c_str());
  if (d.HasParseError()) {
    _logger->error("failed to parsing json file {}", config_file);
    std::cout << content << "\n";
    return dd_bad_request_400();
  }

  for (unsigned int i = 0; i < d.Size(); i++) {
    rapidjson::Value& v = d[i];
    if (!v.HasMember("service_name") || !v.HasMember("config")) {
      _logger->warn("failed to parse service id={}", i);
      continue;
    }
    std::string sname = v["service_name"].GetString();
    std::string body = jrender(v["config"]);
    auto status = service_create(sname, body);
    if (status["status"]["code"] == 201) {
      _logger->info("succ to create service name={} id={}", sname, i);
    }
  }
  return dd_created_201();
}
```

Service Predict Flow

- Step 1. HttpJsonAPI::ApiHandler::operator
- Step 2. JsonApi:: service_predict
- Step 3. Service::predict
- Step 4. MLLib::predictor_job
- Step 5. TFLIb::predict
- Step 6. ServerGroup::run

Internal API Introduce – TF Session




```cpp
tensorflow::SessionOptions create_session_options(const APIData& ad) {
  tensorflow::SessionOptions options;
  tensorflow::ConfigProto &config = options.config;
  config.set_allow_soft_placement(true);
  tensorflow::GPUOptions* gpu_options = config.mutable_gpu_options();
  gpu_options->set_allow_growth(false);
  std::string visiable_devices = ad.get("visiable_devices").get<std::string>();
  gpu_options->set_visible_device_list(visiable_devices);
  auto virtual_devices = ad.getv("virtual_devices");
  // todo: check the device size match with physical device
  for (auto& limits_ad : virtual_devices) {
    auto mem_limits = limits_ad.get("limits").get<std::vector<int>>();
    auto vd = gpu_options->mutable_experimental()->add_virtual_devices();
    for (auto mb : mem_limits) {
      vd->add_memory_limit_mb(mb * 1.0);
    }
  }
  return options;
}
```

```cpp
tensorflow::Status Servable::load_graph(int device_id,
                                        const tensorflow::SessionOptions& options) {
  tensorflow::GraphDef graph_def;
  tensorflow::Status status
  if (_parameters.graph_pb.find(".xz") != std::string::npos) {
    std::vector<char> bytes;
    cipher::read_plain_binary(pb_file, bytes);
    status = graph_def.ParseFromArray(bytes.data(), (int)bytes.size());
  } else {
   // support pb file is less than 2gb
   status = ReadBinaryProto(tensorflow::Env::Default(),
                            _parameters.graph_pb, &graph_def);
  }
  if (!status.ok()) { return status; }
  // modify node device information
  std::string device("/gpu:" + std::to_string(device_id));
  for(int i = 0; i < graph_def.node_size(); ++i) {
      graph_def.mutable_node(i)->set_device(device);
  }

  auto session = std::unique_ptr<tensorflow::Session>(
              tensorflow::NewSession(options));
  auto create_status = session->Create(graph_def);
  if (!create_status.ok()) {
    return create_status;
  }
  _sessions.emplace_back(std::move(session));
  return tensorflow::Status::OK();
}
```

Internal API Introduce – OpenCV MAT

Refer src/comon/mat_utils.[h,cc]

- Mat: 核心数据结构, 对外接口clone, copyTo, reshape, ptr;
  内部属性, data, step[0], step[1], depth, channels, elemSize
- Mat主要操作方法:
  1) add, subtract, multiply, divide, addWeighted, sum;
  2) countNonZero, findNonZero, mea, meanStdDev, norm, batchDistance, normalize, reduce;
  3) merge, split, mixChannels, extractChannel, flip, repeat
  4) bitwise_{and, or, xor, not}, absdiff, inRange, min, max
  5) sqrt, pow, exp, log, magnitude
  6) phase, magnitude, checkRange, gemm
  7) transpose, transform, perspectiveTransform
  8) trace, invert, solve, eigen
  9) calcCovarMatrix, PCA, SVD, kmeans
- Draw funcitons:
  1) line, rectangle, circle, ellipse, fillConvexPoly, fillPoly, polylines
  2) clipLine, putText
- Iterator, 各种迭代器, MatConstIterator, NAryMatIterator
- Distance函数: norm, normalize
- Tree: KDTree
- Sequences: seq
- Algorithm:
- Other: CommandLineParser, parallel_for_, Mutex, AutoLock

Mat & helpers

```
### opencv core objects and ops
- template class: Size_, Point_, Vec, Matx, 需要类参数实例化
- DataType: 通过 DataDepth<T>::value 获取数据类型
- <_tp, m, n>Matx: 一个简单版本的矩阵封装, _Tp val[m*n];
- <_tp, m>Vec: 一个简单版本的向量封装
- Complex: 复数类
- Point_: 2-d point, Point2[i,f,d]
- Points3_: 3-d point, Point3[i,f,d]
- Size_: 表示h,w
- Rect_: x, y, w, h
- RotatedRect, center, size, angle
- Scalar_:  Vec<_Tp, 4>, typedef Scalar_<double> Scalar
- Range: start, end
- DataType:  specialized for each primitive numerical type supported by OpenCV
- Ptr: 智能指针, 管理数据信息
- _InputArray: Proxy datatype for passing Mat's and vector<>'s, input数据的抽象基类, 对外接口getMat, getMatVector
- _OutputArray: output数据的抽象接口基类, 对外提供getMatRef,  create
- MatAllocator: 内存分配器
```

Mat & n Dimentional Array

```
if (max_width < MIN_WIDTH) {max_width = MIN_WIDTH;}
const int c = 1;
std::vector<int> size = {n, fixed_h, max_width, c};
rois = cv::Mat(size.size(), size.data(), CV_32F, Scalarf(0.0));
int step0 = 1;
for (unsigned int j = 1; j < size.size(); j++) {
  step0 *= size[j];
}

auto cv_type = c > 1 ? CV_32FC3 : CV_32FC1;
for (unsigned int i = 0; i < temp_mats.size(); ++i) {
  int new_w = widths_(i, 1);
  cv::Mat mapped_mat(fixed_h, max_width, cv_type, rois.data + step0 * 4 * i);
  // be careful, enlarge the marge of the input
  if (new_w < MIN_WIDTH) {
    mapped_mat(cv::Rect(0, 0, MIN_WIDTH, fixed_h)) = 1.0;
    widths_(i, 1) = MIN_WIDTH;
  }
  int start_x = new_w < MIN_WIDTH ? (MIN_WIDTH - new_w) / 2 : 0;
  auto roi = mapped_mat(cv::Rect(start_x, 0, new_w, fixed_h));
  temp_mats[i].copyTo(roi);
  roi.convertTo(roi, cv_type, 1.0 / 255.0);
}
```

Servable/ServableGroup

```
class Servable {
 public:
  Servable() {}
  ~Servable() {
    for (auto& p : _sessions) { p->Close(); p.reset(nullptr); }
    _thread_pool.reset(nullptr);
  }

  // session options are created at process level, shared on all sessions
  void init(const APIData&, const tensorflow::SessionOptions&, int&);
  tensorflow::Status Iinfer(const MatList&, MatList&);
  tensorflow::Status IinferSeq(const MatList&, std::vector<std::string>&);
  tensorflow::Status MinferSeq(const MatList&, std::vector<std::string>&);

 private:
  tensorflow::Status load_graph(int, const tensorflow::SessionOptions&);
  inline int get_random_device() {
    return Random::random_integer(0, _bind_device_cnt - 1);
  }

  ServableParameters _parameters;
  int _bind_device_cnt;
  SessionList _sessions;
  std::unique_ptr<ThreadPool> _thread_pool;
  DISALLOW_COPY_AND_ASSIGN(Servable);
};
```

```
// ServableGroup manage multiple model life cycle
class ServableGroup {
 public:
  ServableGroup() {};
  ~ServableGroup() {};
  virtual void init(const APIData& ad);
  virtual void run(const APIData& ad_in, APIData& ad_out) {}

  Servable* get_servable(int i) {return _servables[i].get();}

 private:
  tensorflow::SessionOptions _options;
  std::vector<std::unique_ptr<Servable>> _servables;
  DISALLOW_COPY_AND_ASSIGN(ServableGroup);
};
```

Servable helpers

```cpp
// util function for mat and tensor transformation
template <typename T>
tensorflow::Tensor Imat2tensor(const cv::Mat& m) {
  // m: >=2 dimentional matrix , type support float or int
  //  shape(n,1) -> shape(n); shape(n,1,1) -> shape(n,1)
  std::vector<tensorflow::int64> shape;
  if (m.cols == 1 && m.dims  == 2) {
    shape.push_back(m.rows);
  } else if (m.dims == 3 && m.size[1] == 1 && m.size[2] == 1) {
    shape.assign(m.size.p, m.size.p + m.dims - 1);
  } else {
    shape.assign(m.size.p, m.size.p + m.dims);
  }

  if (m.channels() > 1) { shape.push_back(m.channels()); }
  auto tensor_shape = absl::MakeSpan(shape);
  tensorflow::Tensor tensor(tensorflow::DataTypeToEnum<T>::value,
                            tensorflow::TensorShape(tensor_shape));
  auto* data_ptr = tensor.flat<T>().data();
  // cv::Mat map_mat(m.dims, m.size.p, m.type(), data_ptr, m.step.p);
  cv::Mat map_mat(m.dims, m.size.p, m.type(), data_ptr);
  m.convertTo(map_mat, m.type());
  return tensor;
}
```

```cpp
// opencv2.4 not support int64 mat, todo: update the opencv3/4
//  new create mat data is continuous, crop will not keep continuous
template <typename T>
cv::Mat Itensor2mat(tensorflow::Tensor& tensor) {
  auto* ptr = tensor.flat<T>().data();
  auto tensor_size_vec = tensor.shape().dim_sizes();
  std::vector<int> size_vec(tensor_size_vec.begin(), tensor_size_vec.end());
  int dims = size_vec.size();
  if (dims == 1) {
    dims = 2;
    size_vec.push_back(1);
  } else if (dims == 0) {
    dims = 2;
    size_vec.push_back(1);
    size_vec.push_back(1);
  }
  cv::Mat m(dims, size_vec.data(), matutils::RawTypeToCvType<T>::value, ptr);
  return m.clone();
}
```

Model Cipher

- Refer src/ext/cipher/aes.hpp

```
static const std::vector<int> RANDOM_PRIME_NUMS = {0,3,23,37,107,139,701};
static const std::vector<int> RANDOM_PRIME_INDEX = {3,2,5,1,6,4,0};
```

```
inline int write_plain_binary(const std::string& file_path,
                              std::vector<char>& bytes) {
  const int n = RANDOM_PRIME_INDEX.size();
  const auto& last_bias = RANDOM_PRIME_NUMS[n - 1];
  const int size = bytes.size();
  std::fstream file;
  file.open(file_path, std::ios::out | std::ios::binary);
  for (auto& i : RANDOM_PRIME_INDEX) {
    int s = RANDOM_PRIME_NUMS[i];
    int e = (s == last_bias) ? size : RANDOM_PRIME_NUMS[i + 1];
    file.write(&bytes[0] + s, e - s);
  }
  file.close();
  return 0;
}
```

```
inline int read_plain_binary(const std::string& file_path,
                             std::vector<char>& bytes) {
  const int n = RANDOM_PRIME_INDEX.size();
  const auto& last_bias = RANDOM_PRIME_NUMS[n - 1];
  std::ifstream file(file_path, std::ios::binary | std::ios::ate);
  if (!file.eof() && !file.fail()) {
    file.seekg(0, std::ios_base::end);
    std::streampos file_size = file.tellg();
    file.seekg(0, std::ios_base::beg);
    bytes.resize(file_size);
    for (auto& i : RANDOM_PRIME_INDEX) {
      int s = RANDOM_PRIME_NUMS[i];
      int e = (s == last_bias) ? int(file_size) : RANDOM_PRIME_NUMS[i + 1];
      file.read(&bytes[0] + s, e - s);
    }
    file.close();
  } else {
    return -1;
  }

  return 0;
}
```

Application Develop Demo

- See src/backend/tf/ocr_v2_app.h/cc