ML2 Takehome project

Implementation of Lenet5 with Libtorch

Anthony Jung

Contents

- Libtorch Lenet5
  - How to define custom module properly
- Project Overall
  - Torchserve
  - Airflow

Libtorch Lenet5

Lenet5

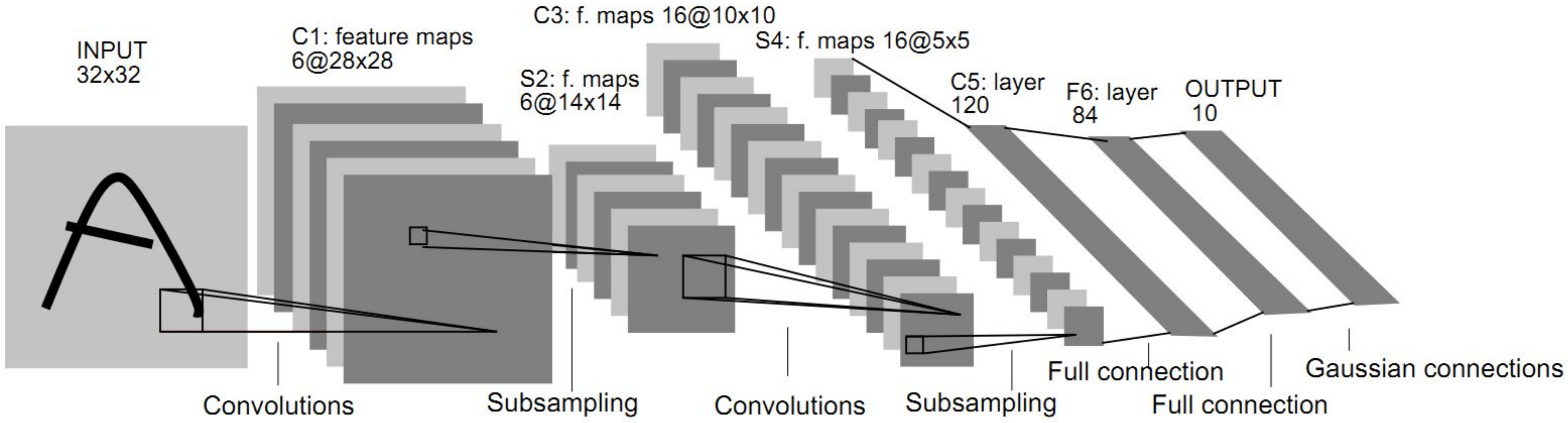


Fig. 2. Architecture of LeNet-5, a Convolutional Neural Network, here for digits recognition. Each plane is a feature map, i.e. a set of units whose weights are constrained to be identical.

gradient-based learning applied to document recognition, Yann LeCun et al.

Model

layer	kind	num filter	filter size	stride	padding
c1	conv	6	(5,5)	1	valid
s2	max pooling	-	(2,2)	2	valid
c3	conv	16	(5,5)	1	valid
s4	max pooling	-	(2,2)	2	valid
c5	conv	120	(5,5)	1	valid
f6	dense	84	-	-	-
output	softmax	10	-	-	-

Implementation

```
struct Lenet5Impl : nn::Module
{
public:
 Lenet5Impl()
     : c1(nn::Conv2dOptions(1, 6, /*kernel_size=*/{5, 5}).padding({2, 2})),
       c3(nn::Conv2dOptions(6, 16, /*kernel_size=*/{5, 5})),
       c5(nn::Conv2dOptions(16, 120, /*kernel_size=*/{5, 5})),
       f6(120, 84),
       output(84, 10)
 {
   register_module("c1", c1);
   register_module("c3", c3);
   register_module("c5", c5);
   register_module("f6", f6);
   register_module("output", output);
 }
```

Implementation

```
 Tensor forward(Tensor x)
 {
   x = max_pool2d(relu(c1(x)), {2, 2}, {2, 2});
   x = max_pool2d(relu(c3(x)), {2, 2}, {2, 2});
   x = relu(c5(x));
   x = x.view({x.size(0), -1});
   x = relu(f6(x));
   x = log_softmax(output(x), /*dim=*/1);

   return x;
 }

 nn::Conv2d c1, c3, c5;
 nn::Linear f6, output;
};
TORCH_MODULE(Lenet5);
```

Class Definition

```
 Tensor forward(Tensor x)
 {
   x = max_pool2d(relu(c1(x)), {2, 2}, {2, 2});
   x = max_pool2d(relu(c3(x)), {2, 2}, {2, 2});
   x = relu(c5(x));
   x = x.view({x.size(0), -1});
   x = relu(f6(x));
   x = log_softmax(output(x), /*dim=*/1);

   return x;
 }

 nn::Conv2d c1, c3, c5;
 nn::Linear f6, output;
};
TORCH_MODULE(Lenet5);
```

MACRO : TORCH_MODULE (pimpl.h)

```
#define TORCH_MODULE_IMPL(Name, ImplType)                             \
 class Name : public torch::nn::ModuleHolder<ImplType> { /* NOLINT */ \
  public:                                                             \
   using torch::nn::ModuleHolder<ImplType>::ModuleHolder;             \
   using Impl = ImplType;                                             \
 }

#define TORCH_MODULE(Name) TORCH_MODULE_IMPL(Name, Name##Impl)
```

ModuleHolder Class (pimpl.h)

```
template <typename Contained>
class ModuleHolder : torch::detail::ModuleHolderIndicator {
protected:
 std::shared_ptr<Contained> impl_;
public:
 using ContainedType = Contained;

 ModuleHolder() : impl_(default_construct()) {
   static_assert(
       std::is_default_constructible<Contained>::value,
       "You are trying to default construct a module which has "
       "no default constructor. Use = nullptr to give it the empty state "
       "(e.g. `Linear linear = nullptr;` instead of `Linear linear;`).");
 }
 /* implicit */ ModuleHolder(std::nullptr_t) : impl_(nullptr) {}
```

std::shared_ptr(in contrast to std::unique_ptr)

std::shared_ptr is a smart pointer that retains shared ownership of an object through a pointer. Several shared_ptr objects may own the same object.

The object is destroyed and its memory deallocated when either of the following happens:

- the last remaining shared_ptr owning the object is destroyed;
- the last remaining shared_ptr owning the object is assigned another pointer via operator= or reset().

Constructor Initializer

```
struct Lenet5Impl : nn::Module
{
public:
 Lenet5Impl()
     : c1(nn::Conv2dOptions(1, 6, /*kernel_size=*/{5, 5}).padding({2, 2})),
       c3(nn::Conv2dOptions(6, 16, /*kernel_size=*/{5, 5})),
       c5(nn::Conv2dOptions(16, 120, /*kernel_size=*/{5, 5})),
       f6(120, 84),
       output(84, 10)
 {
   register_module("c1", c1);
   register_module("c3", c3);
   register_module("c5", c5);
   register_module("f6", f6);
   register_module("output", output);
 }
```

Constructor (pimpl.h)

```
template <
    typename Head,
    typename... Tail,
    typename = typename std::enable_if<
        !(torch::detail::is_module_holder_of<Head, ContainedType>::value &&
          (sizeof...(Tail) == 0))>::type>
explicit ModuleHolder(Head&& head, Tail&&... tail)
    : impl_(new Contained(
          std::forward<Head>(head),
          std::forward<Tail>(tail)...)) {}
```

Register Modules

```
struct Lenet5Impl : nn::Module
{
public:
 Lenet5Impl()
     : c1(nn::Conv2dOptions(1, 6, /*kernel_size=*/{5, 5}).padding({2, 2})),
       c3(nn::Conv2dOptions(6, 16, /*kernel_size=*/{5, 5})),
       c5(nn::Conv2dOptions(16, 120, /*kernel_size=*/{5, 5})),
       f6(120, 84),
       output(84, 10)
 {
   register_module("c1", c1);
   register_module("c3", c3);
   register_module("c5", c5);
   register_module("f6", f6);
   register_module("output", output);
 }
```

register_module (module.h)

```
OrderedDict<std::string, std::shared_ptr<Module>> children_;
...
template <typename ModuleType>
std::shared_ptr<ModuleType> Module::register_module(
   std::string name,
   std::shared_ptr<ModuleType> module) {
 TORCH_CHECK(!name.empty(), "Submodule name must not be empty");
 TORCH_CHECK(
     name.find('.') == std::string::npos,
     "Submodule name must not contain a dot (got '",
     name,
     "')");
 auto& base_module = children_.insert(std::move(name), std::move(module));
 return std::dynamic_pointer_cast<ModuleType>(base_module);
}
```

Trainer : Torch Way

```
template <typename DataLoader>
void train(
   size_t epoch,
   Lenet5 &model,
   torch::Device device,
   DataLoader &data_loader,
   torch::optim::Optimizer &optimizer,
   size_t dataset_size)
```

Trainer : Torch Way

```
{
 model->train();
 size_t batch_idx = 0;
 for (auto &batch : data_loader)
 {
   auto data = batch.data.to(device),
targets = batch.target.to(device);
   optimizer.zero_grad();
   auto output = model->forward(data);
   auto loss = torch::nll_loss(output,
targets);
   AT_ASSERT(!std::isnan(loss.template
item<float>()));
   loss.backward();
   optimizer.step();
```

```
   if (batch_idx++ % kLogInterval == 0)
   {
     std::printf(
         "\rTrain Epoch: %ld [%5ld/%5ld]
Loss: %.4f",
         epoch,
         batch_idx * batch.data.size(0),
         dataset_size,
         loss.template item<float>());
   }
 }
}
```

Tester : Torch Way

```
template <typename DataLoader>
void test(
   Lenet5 &model,
   torch::Device device,
   DataLoader &data_loader,
   size_t dataset_size)
```

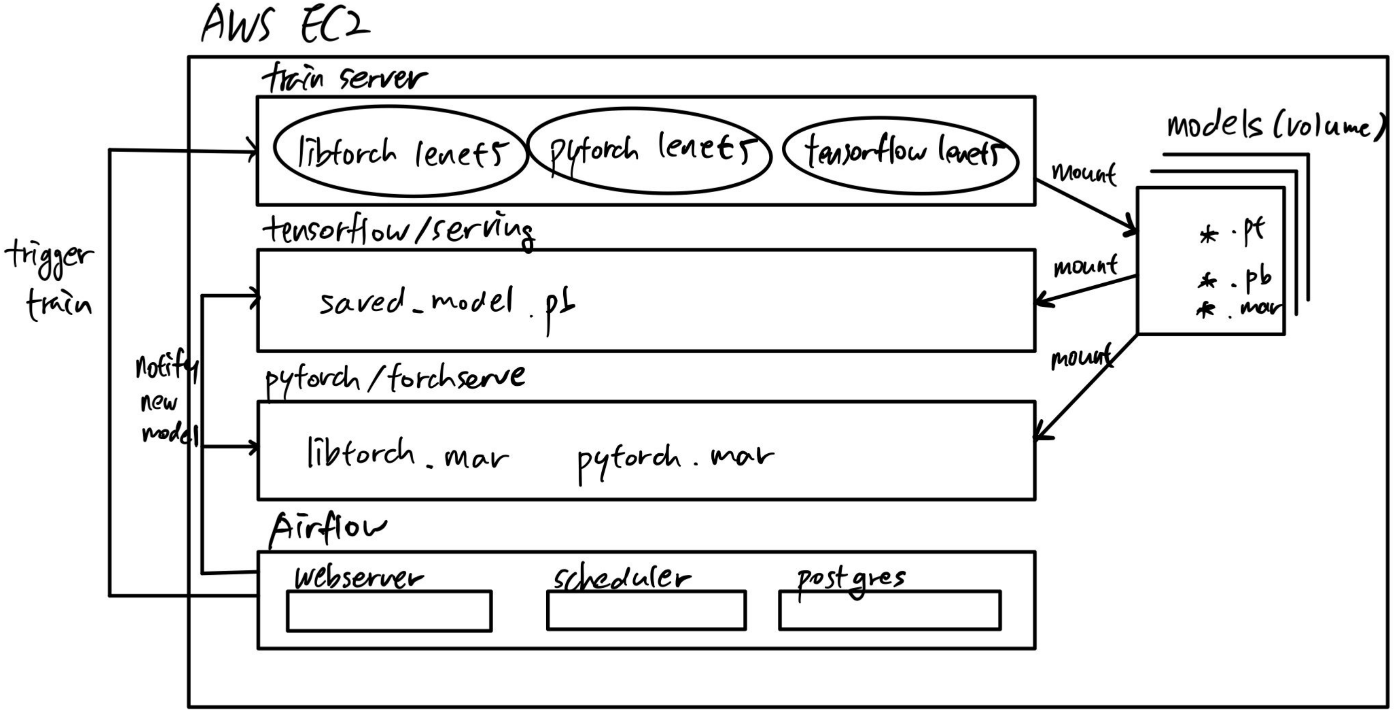
Tester : Torch Way

```
{
 torch::NoGradGuard no_grad;
 model->eval();
 double test_loss = 0;
 int32_t correct = 0;
 for (const auto &batch : data_loader)
 {
   auto data = batch.data.to(device), targets =
batch.target.to(device);
   auto output = model->forward(data);
   test_loss += torch::nll_loss(output, targets,
                    /*weight=*/{},
                    torch::Reduction::Sum)
                    .template item<float>();
   auto pred = output.argmax(1);
   correct += pred.eq(targets).sum().template
item<int64_t>();
 }
```

```
 test_loss /= dataset_size;
 std::printf(
     "\nTest set: Average loss: %.4f | Accuracy:
%.3f\n",
     test_loss,
     static_cast<double>(correct) / dataset_size);
}
```

Project Overall

Docker Containers



docker-compose

https://github.com/anthony0727/ml2_takehome/blob/master/docker-compose.yml

- run containers
- mount volumes
- CMake
- pull && build docker images
- pip3 install -r requirements.txt
- start server
- ...

API : Tensorflow Serving vs Torchserve

```
tf.keras.models.save_model(
    model,
    export_path,
    overwrite=True,
    include_optimizer=True,
    save_format=None,
    signatures=None,
    options=None
)
signature_def: {
  key   : "my_prediction_signature"
  value: {
    inputs: {
       ...
    }
    outputs: {
       ...
    }
    method_name:
"tensorflow/serving/predict"
  }
}
```

```
class MNISTDigitClassifier(ImageClassifier):
   """
   MNISTDigitClassifier handler class. This handler extends
class ImageClassifier from image_classifier.py, a
   default handler. This handler takes an image and returns
the number in that image.
   Here method postprocess() has been overridden while
others are reused from parent class.
   """


   image_processing = transforms.Compose([
       # transforms.Resize(28),
       transforms.ToTensor(),
       transforms.Normalize((0.5,), (0.5,))
   ])

   def postprocess(self, data):
       return data.argmax(1).tolist()
```

Facing 503 Internal Error...

```
2020-09-04 12:36:07,574 [INFO ] W-9003-lenet5_1.0-stdout
org.pytorch.serve.wlm.WorkerLifeCycle -
torch.nn.modules.module.ModuleAttributeError: 'RecursiveScriptModule' object has
no attribute 'forward'
```

Torchserve

Model:

Models could be a `script_module` (JIT saved models) or `eager_mode_models`.

These models can provide custom pre- and post-processing of data along with any other model artifacts such as state_dicts.

Models can be loaded from cloud storage or from local hosts.

...

Torch::save (serialize.h)

```
template <typename Value, typename... SaveToArgs>
void save(const Value& value, SaveToArgs&&... args) {
 serialize::OutputArchive archive(
     std::make_shared<jit::CompilationUnit>());
 archive << value;
 archive.save_to(std::forward<SaveToArgs>(args)...);
}
```

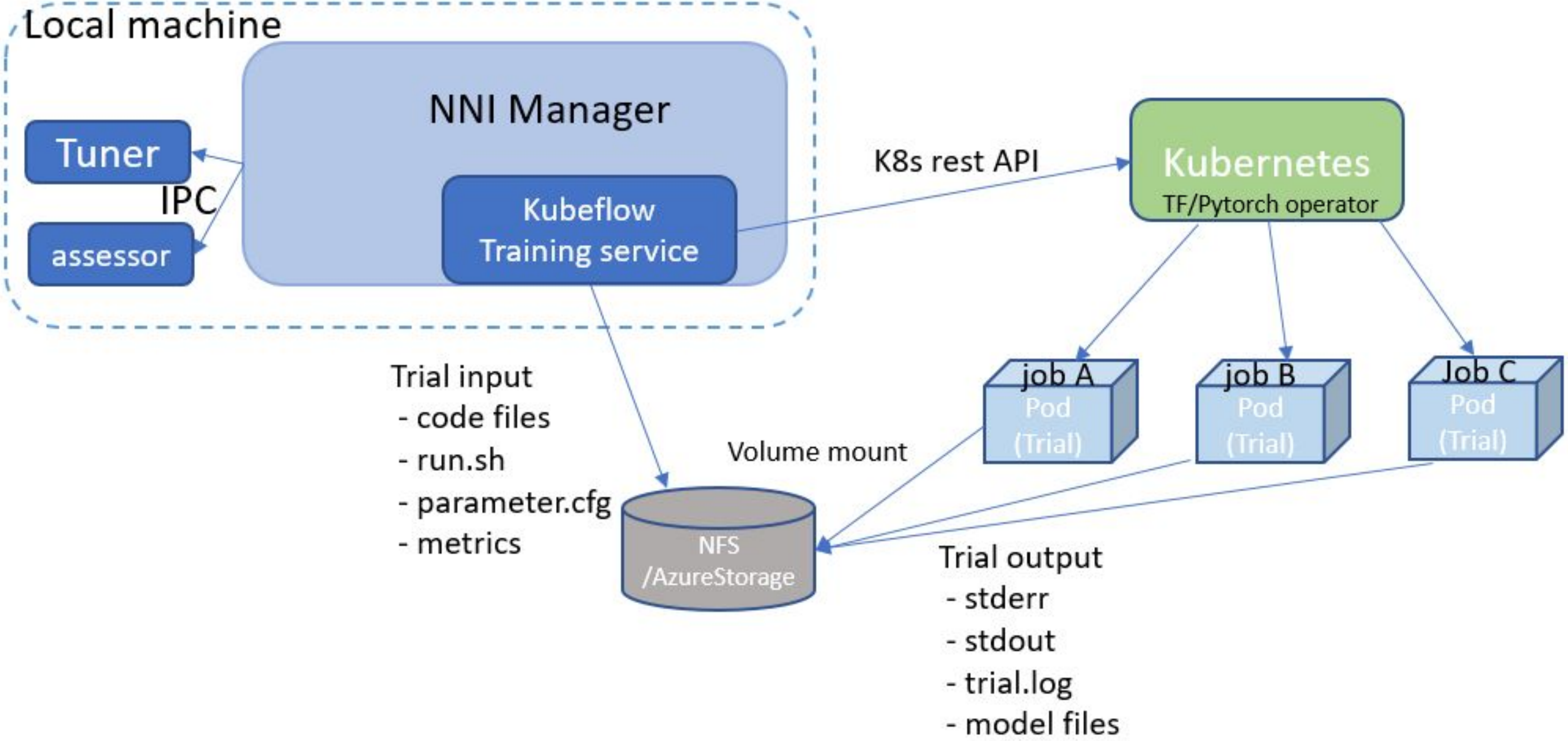
Try tensorflow serving benchmark for now...

https://github.com/anthony0727/ml2_takehome/blob/master/notebook/lenet5_benchmark.ipynb

Application

- Torchserve : web server for many purpose(even for development purpose. ‘1’ server can serve ‘n’ models)
- Dockerized torchserve container : scalable web service
- Airflow : Continuous transfer learning, failover plan for each task, systemized machine learning platform
- Libtorch : High latency requiring deep learning app

NNI + Kubeflow + Kubernetes(k8s)



Thank you!