Deep Learning in a Single File for Smart Devices

Deep learning (DL) systems are complex and often have a few of dependencies. It is often painful to port a DL library into different platforms, especially for smart devices. There is one fun way to solve this problem: provide a light interface and ***put all required codes into a single file*** with minimal dependencies. In this tutorial we will give details on how to do the amalgamation. In addition, we will show a demo to run image object recognition on mobile devices.

Amalgamation: Make the Whole System into a Single File

The idea of amalgamation comes from SQLite and other projects, which packs all the codes into a single source file. Then it is only needed to compile that single file to create the library, which makes porting to various platforms much easier. MXNet provides an amalgamation script, thanks to Jack Deng, to combiles all codes needed for prediction using trained DL models into a single `.cc` file, which has around 30K lines of codes. The only dependency required is just a BLAS library.

We also have a minimal version removed BLAS dependency, and the single file can be compiled into JavaScript by using enscripten.

The compiled library can be used by any other programming language easily. The `.h` file contains a light prediction API, porting to another language with a C foreign function interface needs little effect. For example

- Go: https://github.com/jdeng/gomxnet
- Java: https://github.com/dmlc/mxnet/tree/master/amalgamation/jni
- Python: https://github.com/dmlc/mxnet/tree/master/amalgamation/python

To do amalgamation, there are a few things we need to be careful about when building the project:

- Minimize the dependency to other libraries and do.
- Use namespace to encapsulate the types and operators.

- Avoid do commands such as `using namespace xyz` on the global scope.
- Avoid cyclic include dependencies.

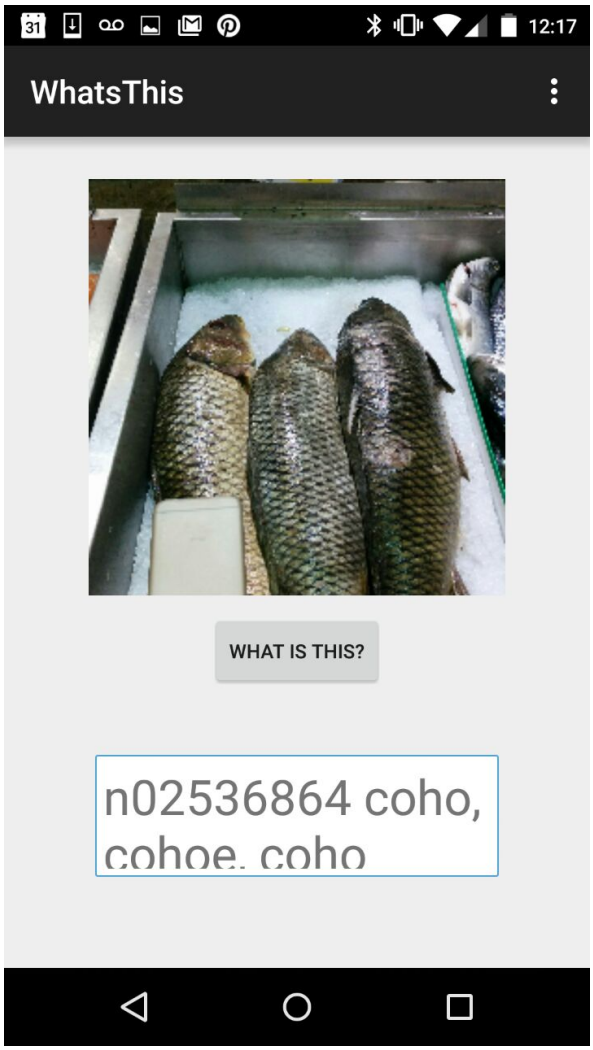
Image Recognition Demo on Mobile

With amalgamation, deploying the system on smart devices (such as Android or iOS) is simple. But there are two additional things we need to consider:

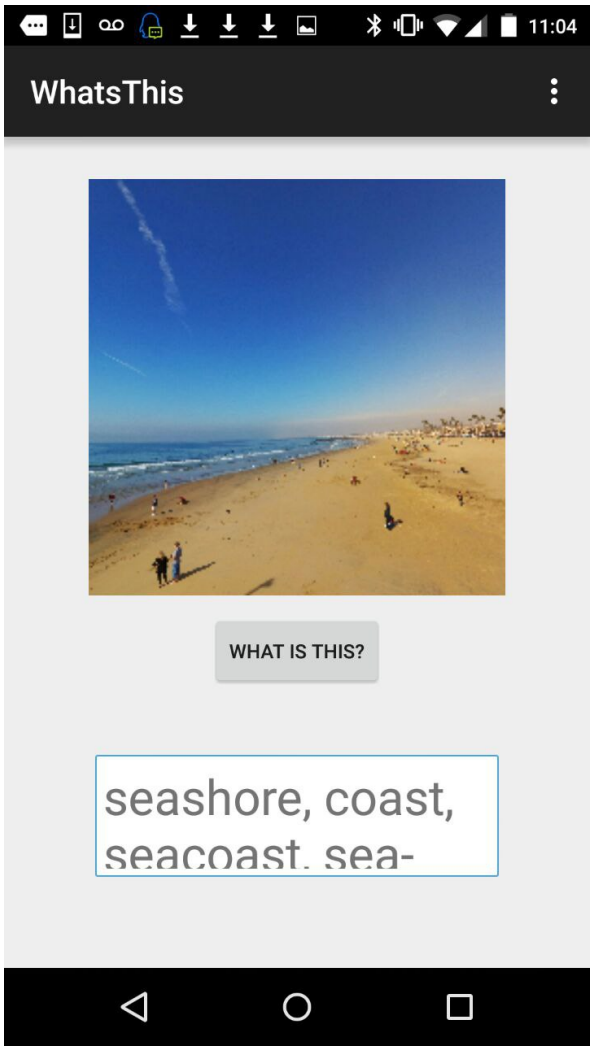
1. The model should be small enough to fit into the device's memory
2. The model should not be too expensive to run given the relative low computational power of these devices

Next we will use the image recognition as an example to show how we try to get such a model. We start with the state-of-the-art inception model. We train it on imagnet dataset, using multiple server machines with GTX 980 cards. The resulted model fits into memory, but we find it can be too expensive to run. Then we remove some layers. then further remove somethings, but now the results are too pool. more explains, and the results table.

Finally, we show an Android example, thanks to Leliana, https://github.com/Leliana/WhatsThis demonstrate how to run on Android.



By using amalgamation, we can easily port the prediction library to mobile devices, with nearly no dependency. Compile on smart platform is no longer a painful task. After compiled library for smart platform, the last thing is call C-API in the target language (Jave/Swift).



Besides pre-trained Inception-BatchNorm network we provided two not bad pretrained models:

We tested on Nexus 5:

	Top-1 Validation on ILSVRC2012	Time	App Size	Runtime Temp Memory Req
FastPoorNet	around 52%, similar to 2011 winner	1s	<10MB	<5MB
Sub InceptionBN	around 64%, similar to 2013 winner	2.7s	<40MB	<10MB
InceptionBN	around 70%	4s-5s	<60MB	10MB

These models are just for demo purpose, as the models are not fine tuned for mobiles, there is definitely great room for improvement. We believe making lightweight, portable and fast deep learning library is fun and interesting, and hope you have fun with the library.

Source code:

https://github.com/Leliana/WhatsThis

Demo APK Download:

- FastPoorNet
- SubInception