Caffe2 - Android

Using custom-built Caffe2 library

Now if you have to use a custom built Caffe2 library suitable to your own needs with custom flags in the application, follow the given steps.

1. First we need to build the static library .so files. You can find the script to build Caffe2 for android in the Caffe2 source repository on Github Caffe2 Repository.
2. Use the "build_android.sh" file in the "caffe2/scripts" folder. If you open the file, you can see various flags like "-DANDROID_ABI", "-DUSE_CUDA", "-DUSE_OPENCV", "-DUSE_NNPACK" etc. You can set them as you require or according to your device specifications.
3. Run the script in caffe2 folder and let the library files build. Two new folders "build_android" and "build_host_protoc" will be build.
4. Locate the static library files, "libCaffe2*.a", "libCAFFE2*_NNPACK.a" etc. and copy them to your application source files in the "jniLibs" directory.
5. Now, open the application in android studio and open the "CMakeLists.txt" files in the External Build Files tab. Include all the custom build library in this file.

You can now build the application and make the APK file. Once the application is built, install and run it on the target device with the desired configurations.

Understanding Caffe2 Implementation

Caffe2 is a deep learning framework which is optimised for mobile integrations, flexibility, easy updates and running model on low powered devices.

The basic unit of computation in Caffe2 are the Operators, which are nothing but a more flexible version of the Caffe layers. Caffe2 comes with over 400 different operations and more custom operators can be added as needed. We can see the difference in implementation of the Fully Connected Operators in Caffe and Caffe2 which emphasise the modularity of the design choice.

There are two key objects in the definition of Caffe2:

1. caffe2::NetDef - This is a Google Protobuf instance which encapsulates the computation graph and the pre-trained network weights. There are two type of NetDef object,
    a. init_net : A google protobuf object, which runs a set of operations that deserialises weights.
    b. predict_net : Also a google protobuf object, which describes how to execute the computation graph for each input.
2. caffe2::Predictor - This class is instantiated with an "initialisation" NetDef, for example the init_net.pb, and a "predict" NetDef, example "predict.pb", and execute the "predict" NetDef with the input and returns the output. In simple terms, the Predictor class takes the input and the network definition objects and returns the output.

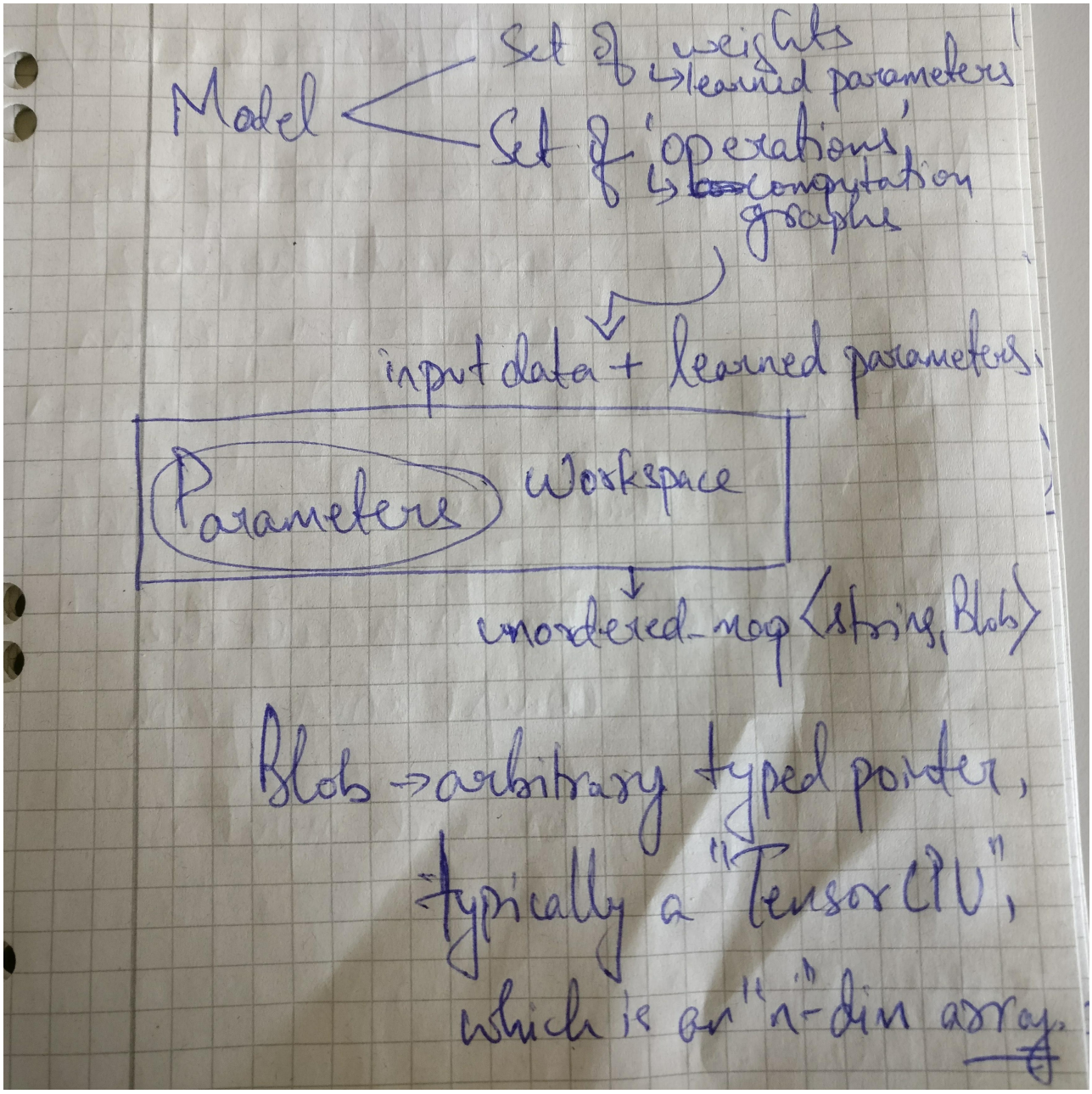
Caffe2 library is composed of:

- A core library composed of the Workspace, Blob, Net and Operator classes.
- An operator library, which contains a range of Operator Implementations.

The complete library is pure C++, with the only non-optional dependencies being:

- Google Protobuf
- Eigen, a BLAS (on Android) which is required for certain primitives.

Caffe2 Models



The figure shows the basic constituent of the Caffe2 model. In summary, Caffe2 models consists of

- Set of **weights**, which are the parameters to be learnt, defined in the init_net protobuf object
- Set of **operations**, which consists of the computation graphs, defined in the predict_net protobuf objects.

Thus, the workspace consists of the parameters which is an `unordered_map<string, Blob>` type, of which the parameter Blob is an arbitrary pointer to, usually, a "TensorCPU", which is an "n"-dimensional array.

Performance

Currently, Caffe2 is optimized for ARM CPUs with NEON which are basically all ARM CPUs released after 2012. They claim to outperform on-board GPUs (the NNPACK ARM CPU implementation outperforms Apple's MPSCNNConvolutions on devices older than iPhone6s).

For convolutional implementations, NNPACK is recommended which is said to give ~2x-3x speed boost to the computations.

For non-convolutional implementation, it is recommended to fall back to BLAS libraries, like Eigen on Android.

Thus, including both NNPACK library and Eigen library provides good performance on ARM CPUs with NEON for the current implementation of Caffe2.

Memory

The total memory usage while running a Predictor instance is the sum of size of the weights and the total size of the activations.

- In Caffe2, there is no 'static' memory allocation.
- All allocations are tied to the Workspace instance owned by the Predictor.

Thus, the amount of memory required depends on the size of the model and only that amount of memory is allocated, as much as required by the Predictor class to run the network.

There are many memory optimizations that are introduced by fb-caffe-extensions, that can be found here FB Caffe extensions.

Predictor, is a simple C++ library containing wrappers that wraps the common pattern of running Caffe:Net in **Multiple Threads sharing weights** in a topological ordering of the graphs.

The Predictor/Optimize.h optimizes memory usage by automatically reusing intermediate activations when it is safe. This saves a lot of memory. More details about the optimizations can be found on the Github page.