Neural Networks Speed-up and Compression

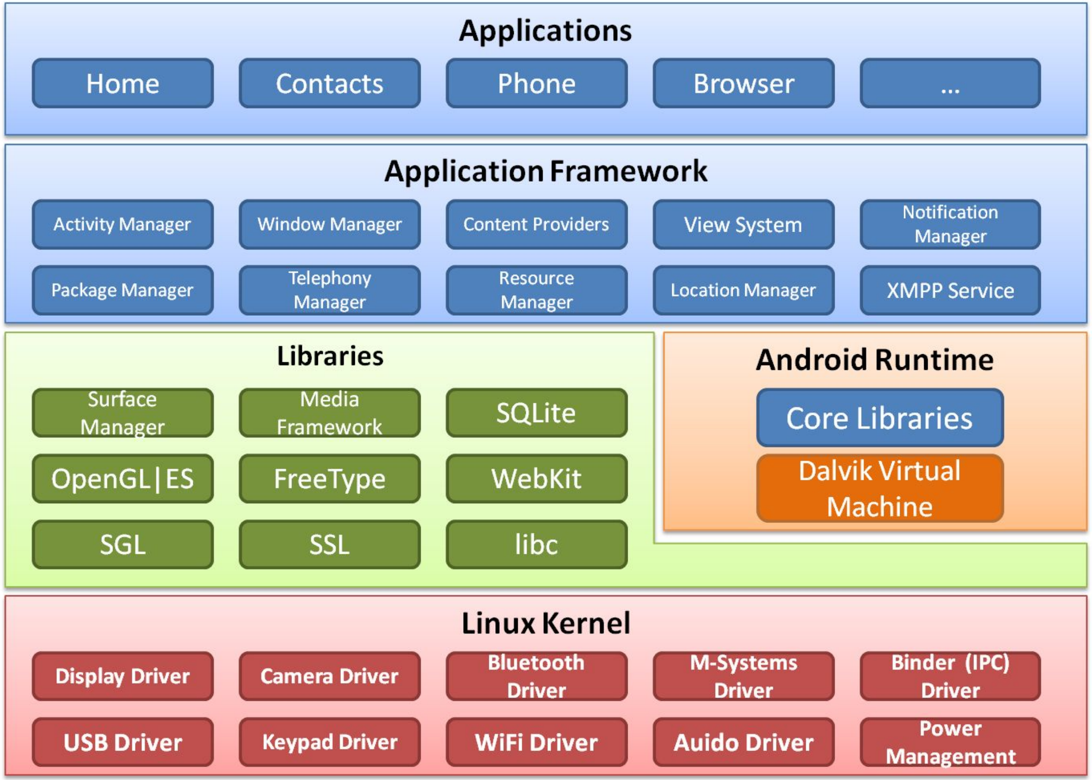
Lecture 7: Inference on mobile devices

Outline

- Android System
- Model preparation for quantization
  - Fusion of Linear, Normalization, Activation layers
  - Statistics collections for quantization
  - Quantization
  - Conversion to TorchScript
- Model inference using
  - mobile terminal
  - mobile application using mobile emulator
  - mobile application using real mobile device

Android System

Android System



http://techbooster.jpn.org/

Model Inference using Mobile Terminal

Problem Formulation

What we want to do?

- PyTorch has its pre-build libraries for neural networks inference on mobile devices
- We want to take out model trained on the computer and convert it to the format which can be understood by PyTorch pre-built library
- Then we measure inference time of our model on mobile device

What we should do to run the model in mobile terminal?

- We connect mobile to our laptop
  - Cable connection + ADB
  - ADB - interface to connect to mobile terminal (aka SSH)

- We need
  - to build (or load already built) a container that contains all required libraries to run our TorchScript model
  - transfer this container to mobile device (via ADB)
  - transfer our TorchScript model to mobile device (via ADB)

Model Inference using Mobile Terminal

Model Preprocessing

On your computer: build TorchScript model

```python
import torch
import torchvision
from torch.utils.bundled_inputs import (
  augment_model_with_bundled_inputs)
from torch.utils.mobile_optimizer import optimize_for_mobile

# Load PyTorch model
model = torchvision.models.resnet18(pretrained=True)
model.eval()

# Generate input image
example = torch.zeros(1, 3, 224, 224)

# Save model graph to Torch script format
script_module = torch.jit.trace(model, example)



# Optimize for mobile PyTorch operations that are supported by Android framework
# If operations are not supported, they remain unchanged
script_module_optimized = optimize_for_mobile(script_module)

# Create a joint input consisting of model and input image
augment_model_with_bundled_inputs(script_module_optimized, [(example,)])

# Save binary file with model on the computer
torch.jit.save(script_module_optimized, "./resnet18.pt")
```

Note: you can generate this model in Google Colab and load it to the computer

Model Inference using Mobile Terminal

Mobile Preparation & Packages Installation

Become a developer of your mobile. Install ADB

On your mobile do:

- Settings -> System -> About phone -> Build number (tap 7 times!) -> PIN -> You are developer of now!!!
- Settings -> Developers options -> Activate USB Debugging -> You can connect your phone using cable now!!!

On your computer do:

- Install **ADB** on your computer
  - MAC: “brew install android-platform-tools”
  - Ubuntu: “apt-get install android-sdk-platform-tools”
- Check if adb has been installed:
  - “which adb”

- Connect your mobile to the computer using cable

Verify your device is connected to the mobile

```
juliagusak@Julias-MacBook-Pro ~ % adb devices
* daemon not running; starting now at tcp:5037
* daemon started successfully
List of devices attached
BH900NK8BZ      unauthorized
```

Note: When you do “adb devices” for the first time, you’ll need to choose a checkbox to trust this computer in the future and press “OK” (in the pop-up window you will see RSA key of your computer)

Now you are ready to connect to the mobile terminal!!!
You do that through your computer terminal

```
juliagusak@Julias-MacBook-Pro ~ % adb shell
H8324:/ $
```

Push model to mobile

```
juliagusak@Julias-MacBook-Pro ~ % adb push Downloads/resnet18.pt /data/local/tmp/
Downloads/resnet18.pt: 1 file pushed, 0 skipped. 16.7 MB/s (47357282 bytes in 2.698s)
juliagusak@Julias-MacBook-Pro ~ %
```

Push binary file with Profiler to mobile

- Check which CPU processor you have on your mobile
    - New processor: aarch64
    - Old processor: armeabiv7

```
H8324:/ $ cat proc/cpuinfo

Processor       : AArch64 Processor rev 12 (aarch64)
processor       : 0
BogoMIPS        : 38.40
Features        : fp asimd evtstrm aes pmull sha1 sha2 crc32 atomics fphp asimdhp
CPU implementer : 0x51
CPU architecture: 8
CPU variant     : 0x7
CPU part        : 0x803
CPU revision    : 12
```

- Load binary file with profiler

```
juliagusak@Julias-MacBook-Pro ~ % adb push Downloads/speed_benchmark_torch-arm64-v8a.file /data/local/tmp/speed_benchmark
Downloads/speed_benchmark_torch-arm64-v8a.fil...skipped. 17.3 MB/s (34134824 bytes in 1.887s)
juliagusak@Julias-MacBook-Pro ~ %
```

Benchmark your model

Add execution rights to our benchmark tool

```
juliagusak@Julias-MacBook-Pro ~ % adb shell
H8324:/ $ cd /data/local/tmp
H8324:/data/local/tmp $ ls
resnet18.pt  speed_benchmark
H8324:/data/local/tmp $ chmod +x speed_benchmark
```

Run using random input data

```
134|H8324:/data/local/tmp $ ./speed_benchmark --model=resnet18.pt --input_dims="1,3,224,224" --input_type=float --warmup=5 --iter=2
Starting benchmark.
Running warmup runs.
Main runs.
Main run finished. Microseconds per iter: 213371. Iters per second: 4.68668
H8324:/data/local/tmp $
```

Run using loaded data

“--use_bundle_input” - index of the tensor in the list

```
127|H8324:/data/local/tmp $ ./speed_benchmark --model=resnet18.pt  --use_bundled_input=0 --warmup=5 --iter=20
```

Compare float and quantized models

```
[H8324:/data/local/tmp $ ./speed_benchmark --model=resnet18.pt --use_bundled_input=0 --warmup=5 --iter=20
Starting benchmark.
Running warmup runs.
Main runs.
Main run finished. Microseconds per iter: 83016.9. Iters per second: 12.0457
[H8324:/data/local/tmp $ ./speed_benchmark --model=resnet18_quantized.pt --use_bundled_input=0 --warmup=5 --iter=20
Starting benchmark.
Running warmup runs.
Main runs.
Main run finished. Microseconds per iter: 54546.9. Iters per second: 18.3328
```

Compare smartphone and emulator

You can build **speed_benchmark_torch** by yourself.

for smartphone: ANDROID_ABI=arm64-v8a,

for emulator: ANDROID_ABI=x86

```
export ANDROID_HOME=path/to/android/sdk
export ANDROID_NDK=path/to/android/ndk
export JAVA_HOME=path/to/jdk
export GRADLE_HOME=path/to/gradle
export PATH=$ANDROID_SDK/emulator:$ANDROID_SDK/tools:$ANDROID_HOME/platfo
rm-tools:$PATH

# install pytorch dependences
conda install ...

# download pytorch source
git clone --recursive https://github.com/pytorch/pytorch
cd pytorch

# build android
rm -rf build_android
# to test on real device use ANDROID_ABI=arm64-v8a in command below
# on android virtual device: ANDROID_ABI=x86
BUILD_PYTORCH_MOBILE=1 ANDROID_ABI=x86 ./scripts/build_android.sh -DBUILD
_BINARY=ON -DUSE_VULKAN=OFF
```

Compare smartphone and emulator

```python
import torch
from torchvision.models.quantization import resnet18
from torchvision.models.quantization import resnet18 as resnet18q
from torch.utils.mobile_optimizer import optimize_for_mobile
from torch.quantization import fuse_modules

layers_to_fuse = [["conv1", "bn1", "relu"]]
for i in range(1, 5):
    for j in range(2):
        for k in range(1, 3):
            layers_to_fuse += [[f'layer{i}.{j}.conv{k}', f'layer{i}.{j}.bn{k}']]

model = resnet18(pretrained=True)
fuse_modules(model, layers_to_fuse, inplace=True)

torchscript_model = torch.jit.script(model)
torchscript_model_optimized = optimize_for_mobile(torchscript_model)
torchscript_model_optimized._save_for_lite_interpreter("model.ptl")

model = resnet18q(pretrained=True, quantize=True)

torchscript_model = torch.jit.script(model)
torchscript_model_optimized = optimize_for_mobile(torchscript_model)
torchscript_model_optimized._save_for_lite_interpreter("qmodel.ptl")
```

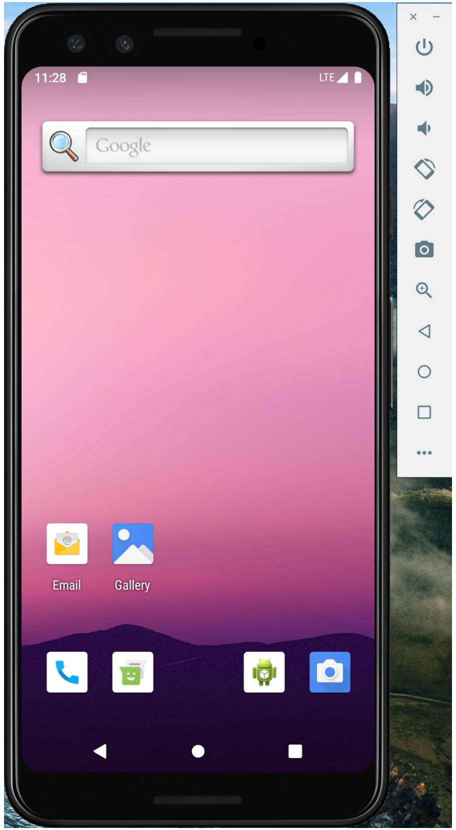
Using optimized lite interpreter model makes inference about 60% faster than the non-optimized lite interpreter model, which is about 6% faster than the non-optimized full jit model

Compare smartphone and emulator

Start and check emulator

```
(pta) [redacted]@mac mobile2 $ adb devices -l
List of devices attached
emulator-5554          device product:sdk_phone_x86 model:Android_SDK_built_for_
x86 device:generic_x86 transport_id:1
```

```
# push benchmark and models to device
adb push pytorch/build_android/bin/speed_benchmark_torch /data/local/tmp
adb push model.ptl /data/local/tmp
adb push qmodel.ptl /data/local/tmp
```



Compare smartphone and emulator

On smartphone

```
(pta) [redacted]@mac mobile2 $ adb shell "/data/local/tmp/speed_benchmark_torch --model=/data/loc
al/tmp/model.ptl" --input_dims="1,3,224,224" --input_type="float" --iter=200 --warmup=100
Starting benchmark.
Running warmup runs.
Main runs.
Main run finished. Microseconds per iter: 63806.8. Iters per second: 15.6723
(pta) [redacted]@mac mobile2 $ adb shell "/data/local/tmp/speed_benchmark_torch --model=/data/loc
al/tmp/qmodel.ptl" --input_dims="1,3,224,224" --input_type="float" --iter=200 --warmup=100
Starting benchmark.
Running warmup runs.
Main runs.
Main run finished. Microseconds per iter: 55956.6. Iters per second: 17.871
```

In emulator

```
(pta) [redacted]@mac mobile2 $ adb shell "/data/local/tmp/speed_benchmark_torch --model=/data/loca
l/tmp/model.ptl" --input_dims="1,3,224,224" --input_type="float" --iter=200 --warmup=100
Starting benchmark.
Running warmup runs.
Main runs.
Main run finished. Microseconds per iter: 76739. Iters per second: 13.0312
(pta) [redacted]@mac mobile2 $ adb shell "/data/local/tmp/speed_benchmark_torch --model=/data/loca
l/tmp/qmodel.ptl" --input_dims="1,3,224,224" --input_type="float" --iter=200 --warmup=100
Starting benchmark.
Running warmup runs.
Main runs.
Main run finished. Microseconds per iter: 47326.9. Iters per second: 21.1297
```

Model Inference using Mobile Application and Mobile Emulator

What should we do to write an application that use our pretrained model?

- We want to have an application that
  - takes an input image from assets/mobile camera
  - takes the NN model we transfer to the device
  - displays an answer to the user
- Application contains 3 main components
  - UI (user interface): defines where to put buttons, images in the application screen(written in .xml)
  - Logic: what happens when you interact with user interface (e.g., push run button to call forward pass of the pretrained model; it is written in Java/Kotlin)
  - Gradle: how to build your application with all needed libraries and created Logic and UI (gradle)

Tool To Build Application

To build application (write UI, Logic, Gradle) we can use Android Studio, which is an IDE that contains

- Visual layout editor: to implement UI (write .xml file)
- Create emulator
  - Emulator - simulator of mobile device (on your laptop you see a “mobile” and can push button on it like on a real one)
- Code editor: to implement Logic (write .java files)
- Profiler: collect information about
  - CPU, memory, internet, energy usage
- Build system: application builder

source: https://developer.android.google.cn/studio

Case 1. Image Segmentation in Emulator

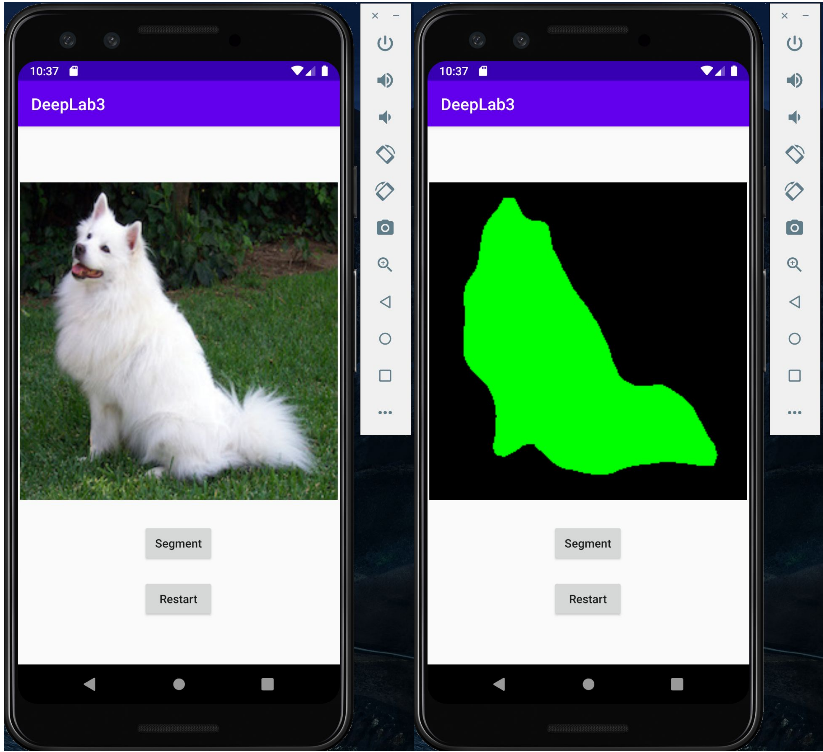
Task: image segmentation

Input source: 2 preloaded images (dog and woman with sheeps)

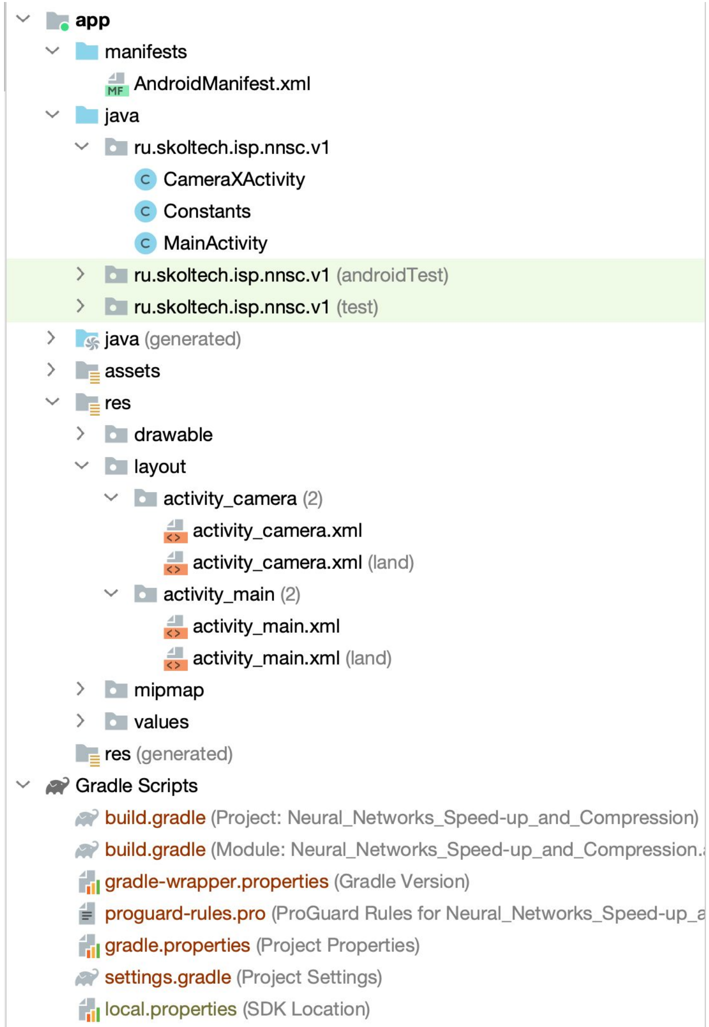
Model: DeepLabV3 with ResNet-50 backbone

Demonstration: using Emulator

image segmentation mask



Project Folder Tree Structure



Manifest

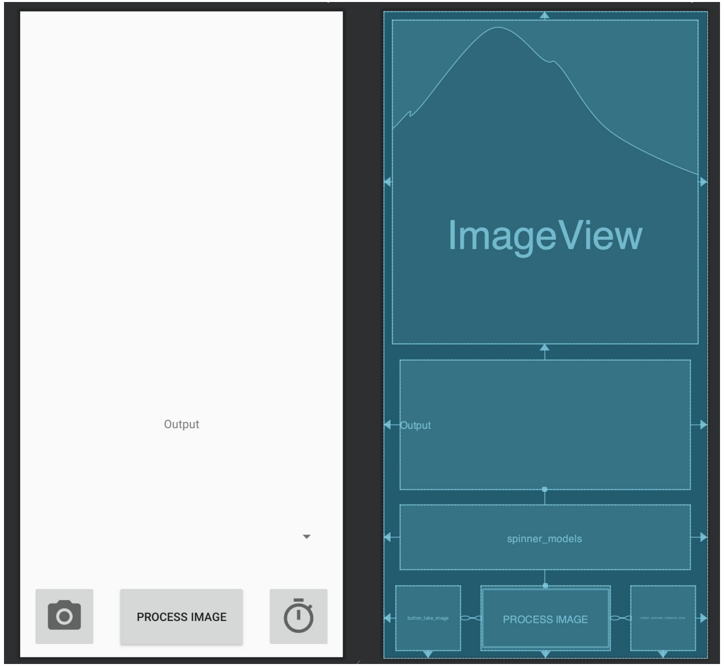
```xml
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    package="ru.skoltech.isp.nnsc.v1">

    <application
        android:allowBackup="true"
        android:icon="@mipmap/ic_launcher"
        android:label="Neural Networks Speed-up and Compression"
        android:roundIcon="@mipmap/ic_launcher_round"
        android:supportsRtl="true"
        android:theme="@style/Theme.AppCompat.Light.NoActionBar">
        <activity android:name=".MainActivity">
            <intent-filter>
                <action android:name="android.intent.action.MAIN" />

                <category android:name="android.intent.category.LAUNCHER" />
            </intent-filter>
        </activity>
        <activity android:name=".CameraXActivity"/>
    </application>

    <uses-permission android:name="android.permission.CAMERA" />

</manifest>
```

- The app's package name
- The components of the app, which include all activities, services, broadcast receivers, and content providers.
- The permissions that the app needs in order to access protected parts of the system or other apps.
- The hardware and software features the app requires, which affects which devices can install the app from Google Play.

Gradle

```
plugins {
    id 'com.android.application'
}

android {
    compileSdkVersion 30
    buildToolsVersion "32.0.0"

    defaultConfig {
        applicationId "ru.skoltech.isp.nnsc.v1"
        minSdkVersion 26
        targetSdkVersion 30
        versionCode 1
        versionName "1.0"

        testInstrumentationRunner 'androidx.test.runner.AndroidJUnitRunner'
    }

    buildTypes {...}
    compileOptions {...}
}

dependencies {

    implementation 'org.pytorch:pytorch_android_lite:1.10.0'
    implementation 'org.pytorch:pytorch_android_torchvision_lite:1.10.0'

    implementation 'androidx.exifinterface:exifinterface:1.3.0-rc01'
    def camerax_version = "1.0.1"
// CameraX core library using camera2 implementation
    implementation "androidx.camera:camera-camera2:$camerax_version"
// CameraX Lifecycle Library
    implementation "androidx.camera:camera-lifecycle:$camerax_version"
// CameraX View class
    implementation "androidx.camera:camera-view:1.0.0-alpha27"
```

User Interface (UI)



Logic

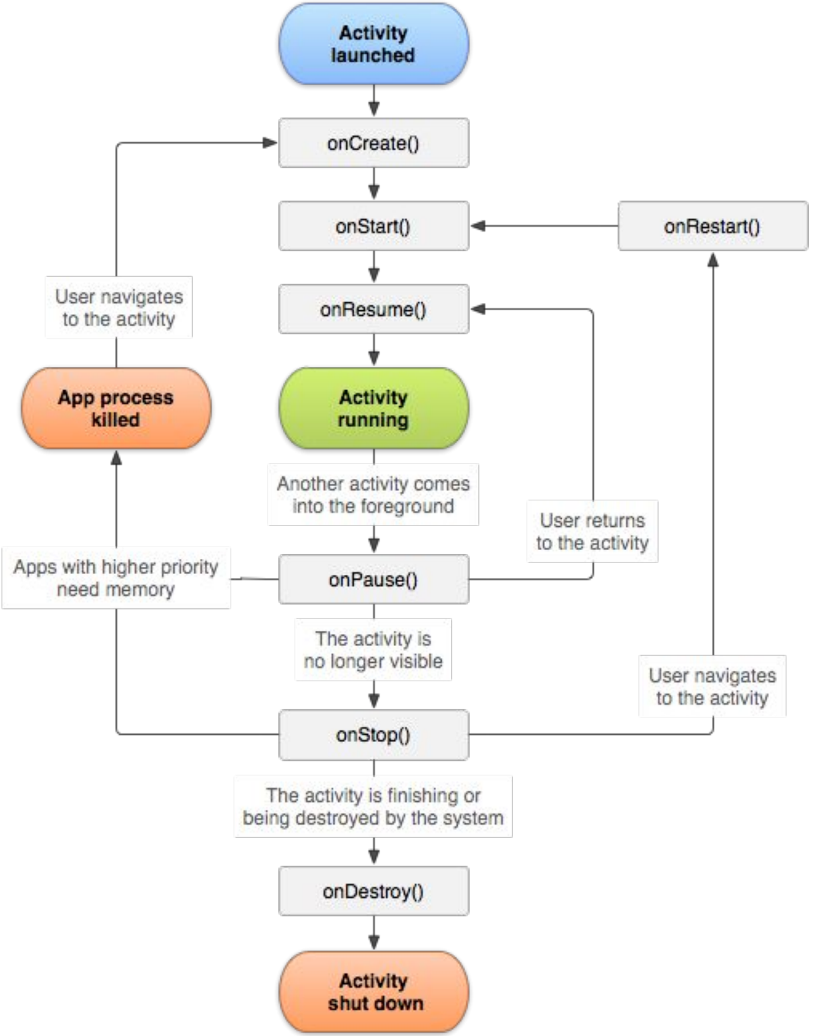
```java
public class MainActivity extends AppCompatActivity  implements AdapterView.OnItemSelectedListener {

    private final int WARMUP = 100;
    private final int ITER = 200;

    private static final String TAG = "MainActivity";
    private static final String mImageName = "image_to_process.jpg";
    private ImageView mImageView;
    private Bitmap mBitmap = null;
    private Module mModule = null;


    @Override
    protected void onCreate(Bundle savedInstanceState) {...}

    @Override
    protected void onResume() {...}

    @Override
    protected void onDestroy() {...}

    @Override
    public void onSaveInstanceState(@NonNull Bundle outState) {...}


    public void onItemSelected(AdapterView<?> parent, View view, int pos, long id) {...}

    public void onNothingSelected(AdapterView<?> parent) {}

    @Override
    protected void onActivityResult(int requestCode, int resultCode, Intent data) {...}

    private void loadModel() {...}

    @WorkerThread  //Denotes that the annotated method should only be called on a worker thread.
    protected void analyzeImage() {...}

    @WorkerThread  //Denotes that the annotated method should only be called on a worker thread.
    protected void estimateInferenceTime() {...}
```

Activity

onCreate(): fires when the system first creates the activity.

onStart(): app prepares for the activity to enter the foreground (visible to the user) and become interactive.

onResume(): activity comes to the foreground

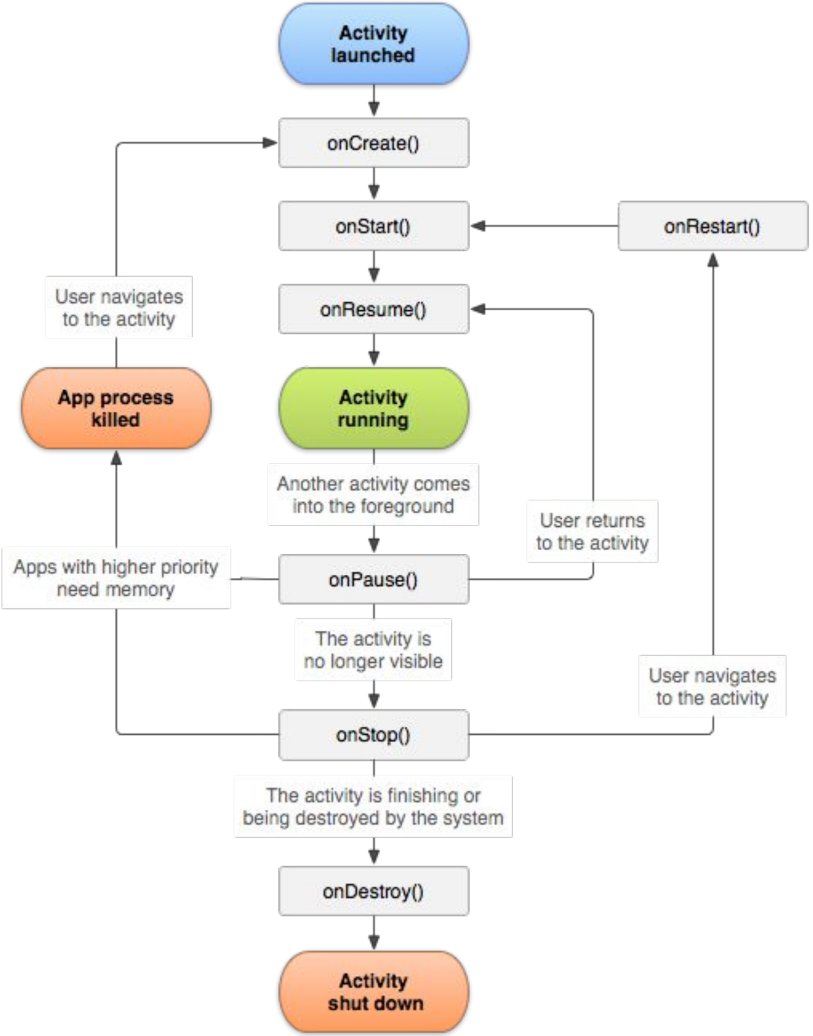


Activity

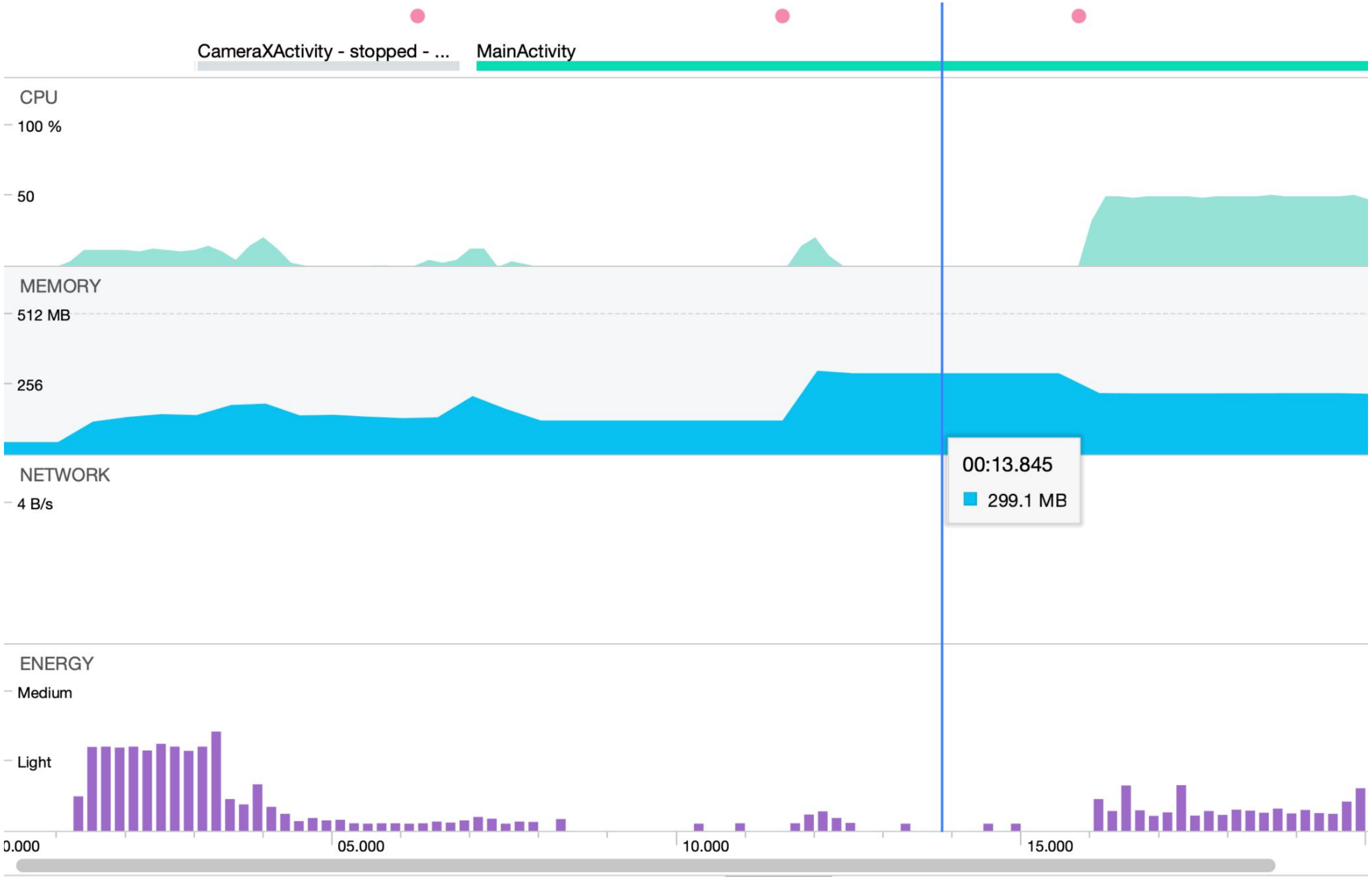
onPause(): indicates that the activity is no longer in the foreground

onStop(): activity is no longer visible to the user

onDestroy() is called before the activity is destroyed



Profiler


CameraXActivity - stopped - ...
MainActivity
CPU
100 %
50
MEMORY
512 MB
256
00:13.845
299.1 MB
NETWORK
4 B/s
ENERGY
Medium
Light
0.000
05.000
10.000
15.000

Model Inference using Mobile Application and Real Mobile device

Case 2. Image Classification on Smartphone

Task: image classification, inference time measurement

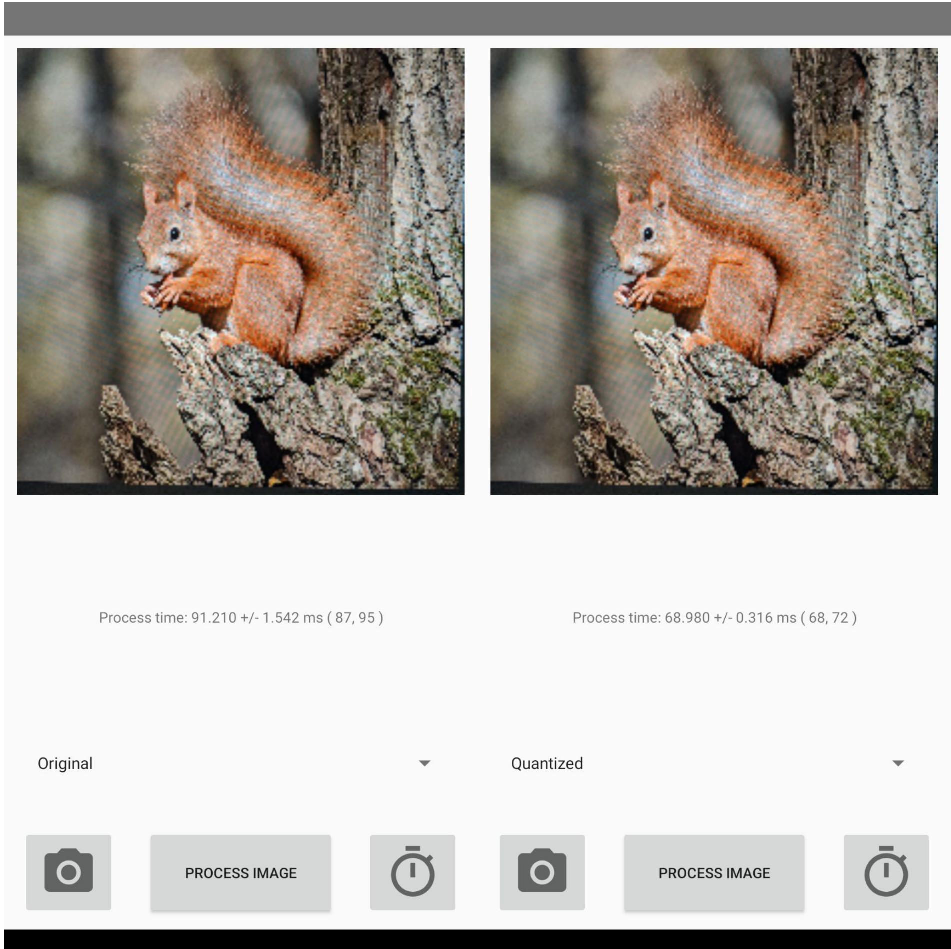
Input source: photo from smartphone camera

Model: ResNet-18 (original and quantized)

Demonstration: using smartphone

warm-up: 100 iter,

measurements: 200 iter



Sources

- https://pytorch.org/mobile/android/