Qualcomm ThunderSoft®

SNPE Training Part 3

Aaron Yue

2021

Contents

- Review
- SNPE Benchmarking
- AI demos - Object Detector Demo
  - Part 1 : Get Camera Stream by Camera2
  - Part 2 : JNI Interface
  - Part 3 : Integrating SNPE
- AI demos - Face Recognition
- Solve SNPE Problem Skills
- Q & A

Review

Question 1

3 Steps to Use SNPE API ?

Answer

- Step 1: Build SNPE instance
- Step 2: Fill input tensor with data
- Step 3: Inference(execute) and get results

SNPE Benchmarking

Overview

- The benchmark shipped in the SNPE SDK consists of a set of python scripts that runs a network on a target Android/LinuxEmbedded device and collect performance metrics. It uses executables and libraries found in the SDK package to run a DLC file on target, using a set of inputs for the network, and a file that points to that set of inputs.

- The input to the benchmark scripts is a configuration file in JSON format. The SDK ships with a configuration file for running the AlexNet model that is created in the SNPE SDK. SDK users are encouraged to create their own configuration files and use the benchmark scripts to run on target to collect timing and memory consumption measurements.

Overview

The configuration file allows the user to specify:

- Name of the benchmark (i.e., AlexNet)
- Host path to use for storing results
- Device paths to use (where to push the necessary files for running the benchmark)
- Device to run the benchmark on (only one device is supported per run)
- Hostname/IP of remote machine to which devices are connected
- Number of times to repeat the run
- Model specifics (name, location of dlc, location of inputs)
- SNPE runtime configuration(s) to use (combination of CPU, GPU, GPU_FP16 and DSP)
- Which measurements to take ("mem" and/or "timing")
- Profiling level of measurements ("off", "basic", "moderate" or "detailed")

Command Line Parameters

- To see all of the command line parameters use the "-h" option when running snpe_bench.py

```
usage: snpe_bench.py [-h] -c CONFIG_FILE [-o OUTPUT_BASE_DIR_OVERRIDE]
                     [-v DEVICE_ID_OVERRIDE] [-r HOST_NAME] [-a]
                     [-t DEVICE_OS_TYPE_OVERRIDE] [-d] [-s SLEEP]
                     [-b USERBUFFER_MODE] [-p PERFPROFILE] [-l PROFILINGLEVEL]
                     [-json] [-cache]
```

Running the Benchmark

- Prerequisites
  - The SNPE SDK has been set up following the SNPE Setup chapter.
  - The Tutorials Setup has been completed. That means all inception_v3 and alexnet assets have been downloaded and decompressed.
  - Optional: If the device is connected to remote machine, the remote adb server setup needs to be done by the user.
- Running AlexNet that is Shipped with the SDK
- snpe_bench.py is the main benchmark script to measure and report performance statistics. Here is how to use it with the AlexNet model and data that is created in the SDK.
  - cd $SNPE_ROOT/benchmarks
  - python3 snpe_bench.py -c alexnet_sample.json -a

    where

    -a benchmarks on the device connected

Viewing the Results (csv File or json File)

- All results are stored in the "HostResultDir" that is specified in the configuration json file. The benchmark creates time-stamped directories for each benchmark run. All timing results are stored in microseconds.
- For your convenience, a "latest_results" link is created that always points to the most recent run.
  - # In alexnet_sample.json, "HostResultDir" is set to "alexnet/results"
  - cd $SNPE_ROOT/benchmarks/alexnet/results
  - # Notice the time stamped directories and the "latest_results" link.
  - cd $SNPE_ROOT/benchmarks/alexnet/results/latest_results
  - # Notice the .csv file, open this file in a csv viewer (Excel, LibreOffice Calc)
  - # Notice the .json file, open the file with any text editor

CSV Benchmark Results File

- The CSV file contains results similar to the example below. Some measurements may not be apparent in the CSV file. To get all timing information, profiling level needs to be set to detailed. By default, profiling level is basic. Note that colored headings have been added for clarity.

CSV 1: Configuration

This section contains information about:

- SNPE SDK version used to generate the benchmark run
- Name & Model:Dlc model name and path to the DLC file
- Device Info device information
- etc.

SNPE SDK version:	1.35.0.698								
Configuration used:		1							
Name	AlexNet								
HostRootPath	/home/user/Workspace/Framework/SNPE/snpe-1.35.0.698/benchmarks/alexnet								
HostResultsDir	/home/user/Workspace/Framework/SNPE/snpe-1.35.0.698/benchmarks/alexnet/results								
Model:Dlc	/home/user/Workspace/Framework/SNPE/snpe-1.35.0.698/models/alexnet/dlc/bvlc_alexnet.dlc								
...									
Device Info:									
ro.product.name	aikit								
				DSP_timing(2 runs)				CPU_timing(2 runs)			
				avg (us)	max (us)	min (us)	runtime	avg (us)	max (us)	min (us)	runtime
		Load		189	193	186	CPU	396	578	214	CPU
		Deserialize		393515	393600	393431	CPU	407482	411082	403882	CPU
	2	Create		3574698	3619728	3529669	CPU	426386	432298	420474	CPU
		Init		3968706	4013692	3923720	CPU	840086	842014	838158	CPU
		De-Init		43575	49752	37398	CPU	105390	105570	105210	CPU
		Create Network(s)		2892845	2937568	2848123	CPU	54397	56804	51991	CPU
		RPC Init Time		197716	198585	196847	CPU	0	0	0	CPU
		Snpe Accelerator Init Time		183164	184038	182290	CPU	0	0	0	CPU
		Accelerator Init Time		182131	183012	181250	CPU	0	0	0	CPU
		Total Inference Time		19664	19711	19617	CPU|DSP	103385	104477	102293	CPU
		Forward Propagate		19651	19697	19605	CPU|DSP	102238	103356	101120	CPU
	3	RPC Execute		19481	19521	19442	CPU|DSP	0	0	0	CPU
		Snpe Accelerator		18863	18900	18826	CPU|DSP	0	0	0	CPU
		Accelerator		18563	18609	18518	CPU|DSP	0	0	0	CPU
		Misc Accelerator		101	102	100	CPU|DSP	0	0	0	CPU
		layer_000 (Name:data Type:data)		145	148	142	DSP	1	1	1	CPU
		layer_001 (Name:conv1 Type:convol		0	0	0	DSP	12340	12368	12312	CPU
	4	layer_002 (Name:relu1 Type:neuron)		1496	1499	1493	DSP	163	125	121	CPU
		...						5			
		layer_020 (Name:fc8 Type:fully_con		469	487	452	DSP	2876	2914	2838	CPU
		layer_021 (Name:prob Type:softmax		66	67	65	DSP	24	24	24	CPU

CSV Benchmark Results File (cont.)

CSV 2: Initialization metrics

This section contains measurements concerning model initialization. Profiling level affects the amount of measurements collected. No metrics are collected for profiling off. Metrics for basic and detailed are shown below:

Profiling level: Basic

- Load measures the time required to load the model's metadata.
- Deserialize measures the time required to deserialize the model's buffers.
- Create measures the time spent to create SNPE network(s) and initialize all layers for the given model. Detailed breakdown of create time can be retrieved with profiling level set to detailed.
- Init measures the time taken to build and configure SNPE. This time includes time measured Load, Deserialize, and Create.
- De-Init measures the time taken to de-initialized SNPE.

SNPE SDK version:	1.35.0.698								
Configuration used:		1							
Name	AlexNet								
HostRootPath	/home/user/Workspace/Framework/SNPE/snpe-1.35.0.698/benchmarks/alexnet								
HostResultsDir	/home/user/Workspace/Framework/SNPE/snpe-1.35.0.698/benchmarks/alexnet/results								
Model:Dlc	/home/user/Workspace/Framework/SNPE/snpe-1.35.0.698/models/alexnet/dlc/bvlc_alexnet.dlc								
...									
Device Info:									
ro.product.name	aikit								
			DSP_timing(2 runs)				CPU_timing(2 runs)			
			avg (us)	max (us)	min (us)	runtime	avg (us)	max (us)	min (us)	runtime
		Load	189	193	186	CPU	396	578	214	CPU
		Deserialize	393515	393600	393431	CPU	407482	411082	403882	CPU
	2	Create	3574698	3619728	3529669	CPU	426386	432298	420474	CPU
		Init	3968706	4013692	3923720	CPU	840086	842014	838158	CPU
		De-Init	43575	49752	37398	CPU	105390	105570	105210	CPU
		Create Network(s)	2892845	2937568	2848123	CPU	54397	56804	51991	CPU
		RPC Init Time	197716	198585	196847	CPU	0	0	0	CPU
		Snpe Accelerator Init Time	183164	184038	182290	CPU	0	0	0	CPU
		Accelerator Init Time	182131	183012	181250	CPU	0	0	0	CPU
		Total Inference Time	19664	19711	19617	CPU|DSP	103385	104477	102293	CPU
		Forward Propagate	19651	19697	19605	CPU|DSP	102238	103356	101120	CPU
	3	RPC Execute	19481	19521	19442	CPU|DSP	0	0	0	CPU
		Snpe Accelerator	18863	18900	18826	CPU|DSP	0	0	0	CPU
		Accelerator	18563	18609	18518	CPU|DSP	0	0	0	CPU
		Misc Accelerator	101	102	100	CPU|DSP	0	0	0	CPU
		layer_000 (Name:data Type:data)	145	148	142	DSP	1	1	1	CPU
		layer_001 (Name:conv1 Type:convol	0	0	0	DSP	12340	12368	12312	CPU
	4	layer_002 (Name:relu1 Type:neuron)	1496	1499	1493	DSP	123	125	121	CPU
		...					5			
		layer_020 (Name:fc8 Type:fully_conr	469	487	452	DSP	2876	2914	2838	CPU
		layer_021 (Name:prob Type:softmax	66	67	65	DSP	24	24	24	CPU

CSV Benchmark Results File (cont.)

CSV 2: Initialization metrics

Profiling level: Moderate or Detailed

- Load measures the time required to load the model's metadata.
- Deserialize measures the time required to deserialize the model's buffers.
- Create measures the time spent to create SNPE network(s) and initialize all layers for the given model. Detailed breakdown of create time can be retrieved with profiling level set to detailed.
- Init measures the time taken to build and configure SNPE. This time includes time measured Load, Deserialize, and Create.
- De-Init measures the time taken to de-initialized SNPE.
- Create Network(s) measures the total time required to create all network(s) for the model. Models that are partitioned will result in multiple networks being created.
- RPC Init Time measures the entire time spent on RPC and accelerators used by SNPE. This time includes time measured in Snpe Accelerator Init Time and Accelerator Init Time. Currently only available for DSP and AIP runtime. Will appear as 0 for other runtimes.
- Snpe Accelerator Init Time measures the total time spent by SNPE to prepare the data for the accelerator process such as GPU, DSP, AIP. Currently only available for DSP and AIP runtime. Will appear as 0 for other runtimes.
- Accelerator Init Time measures the total processing time spent on the accelerator core, which may include different hardware resources. Currently only available for DSP and AIP runtime. Will appear as 0 for other runtimes.

SNPE SDK version:	1.35.0.698								
Configuration used:		1							
Name	AlexNet								
HostRootPath	/home/user/Workspace/Framework/SNPE/snpe-1.35.0.698/benchmarks/alexnet								
HostResultsDir	/home/user/Workspace/Framework/SNPE/snpe-1.35.0.698/benchmarks/alexnet/results								
Model:Dlc	/home/user/Workspace/Framework/SNPE/snpe-1.35.0.698/models/alexnet/dlc/bvlc_alexnet.dlc								
...									
Device Info:									
ro.product.name	aikit								
			DSP_timing(2 runs)				CPU_timing(2 runs)			
			avg (us)	max (us)	min (us)	runtime	avg (us)	max (us)	min (us)	runtime
		Load	189	193	186	CPU	396	578	214	CPU
		Deserialize	393515	393600	393431	CPU	407482	411082	403882	CPU
	2	Create	3574698	3619728	3529669	CPU	426386	432298	420474	CPU
		Init	3968706	4013692	3923720	CPU	840086	842014	838158	CPU
		De-Init	43575	49752	37398	CPU	105390	105570	105210	CPU
		Create Network(s)	2892845	2937568	2848123	CPU	54397	56804	51991	CPU
		RPC Init Time	197716	198585	196847	CPU	0	0	0	CPU
		Snpe Accelerator Init Time	183164	184038	182290	CPU	0	0	0	CPU
		Accelerator Init Time	182131	183012	181250	CPU	0	0	0	CPU
		Total Inference Time	19664	19711	19617	CPU|DSP	103385	104477	102293	CPU
		Forward Propagate	19651	19697	19605	CPU|DSP	102238	103356	101120	CPU
	3	RPC Execute	19481	19521	19442	CPU|DSP	0	0	0	CPU
		Snpe Accelerator	18863	18900	18826	CPU|DSP	0	0	0	CPU
		Accelerator	18563	18609	18518	CPU|DSP	0	0	0	CPU
		Misc Accelerator	101	102	100	CPU|DSP	0	0	0	CPU
		layer_000 (Name:data Type:data)	145	148	142	DSP	1	1	1	CPU
		layer_001 (Name:conv1 Type:convol	0	0	0	DSP	12340	12368	12312	CPU
	4	layer_002 (Name:relu1 Type:neuron)	1496	1499	1493	DSP	123	125	121	CPU
		...						5		
		layer_020 (Name:fc8 Type:fully_conr	469	487	452	DSP	2876	2914	2838	CPU
		layer_021 (Name:prob Type:softmax	66	67	65	DSP	24	24	24	CPU

CSV Benchmark Results File (cont.)

CSV 3: Execution metrics

This section contains measurements concerning the execution of one inference pass of the neural network model. Profiling level affects the amount of measurements collected.

Profiling level: Basic

- Total Inference Time measures the entire execution time of one inference pass. This includes any input and output processing, copying of data, etc. This is measured at the start and end of the execute call.

Profiling level: Moderate or Detailed

- Total Inference Time measures the entire execution time of one inference pass. This includes any input and output processing, copying of data, etc. This is measured at the start and end of the execute call.
- Forward Propogate measures the time spent executing one inference pass excluding processing overheads on one of the accelerator cores. For example, in the case of the GPU this represents the execution time of all the GPU kernels running on the GPU HW.
- RPC Execute measures the entire time spent on RPC and accelerators used by SNPE. This time includes time measured in Snpe Accelerator and Accelerator. Currently only available for DSP and AIP runtime. Will appear as 0 for other runtimes.
- Snpe Accelerator measures the total time spent by SNPE to setup processing for the accelerator. Currently only available for DSP and AIP runtime. Will appear as 0 for other runtimes.
- Accelerator measures the total execution time spent on the accelerator core, which may include different hardware resources. Currently only available for DSP and AIP runtime. Will appear as 0 for other runtimes.

SNPE SDK version:	1.35.0.698								
Configuration used:									
Name	AlexNet								
HostRootPath	/home/user/Workspace/Framework/SNPE/snpe-1.35.0.698/benchmarks/alexnet								
HostResultsDir	/home/user/Workspace/Framework/SNPE/snpe-1.35.0.698/benchmarks/alexnet/results								
Model:Dlc	/home/user/Workspace/Framework/SNPE/snpe-1.35.0.698/models/alexnet/dlc/bvlc_alexnet.dlc								
...									
Device Info:									
ro.product.name	aikit								
		DSP_timing(2 runs)				CPU_timing(2 runs)			
		avg (us)	max (us)	min (us)	runtime	avg (us)	max (us)	min (us)	runtime
	Load	189	193	186	CPU	396	578	214	CPU
	Deserialize	393515	393600	393431	CPU	407482	411082	403882	CPU
2	Create	3574698	3619728	3529669	CPU	426386	432298	420474	CPU
	Init	3968706	4013692	3923720	CPU	840086	842014	838158	CPU
	De-Init	43575	49752	37398	CPU	105390	105570	105210	CPU
	Create Network(s)	2892845	2937568	2848123	CPU	54397	56804	51991	CPU
	RPC Init Time	197716	198585	196847	CPU	0	0	0	CPU
	Snpe Accelerator Init Time	183164	184038	182290	CPU	0	0	0	CPU
	Accelerator Init Time	182131	183012	181250	CPU	0	0	0	CPU
	Total Inference Time	19664	19711	19617	CPU|DSP	103385	104477	102293	CPU
	Forward Propagate	19651	19697	19605	CPU|DSP	102238	103356	101120	CPU
3	RPC Execute	19481	19521	19442	CPU|DSP	0	0	0	CPU
	Snpe Accelerator	18863	18900	18826	CPU|DSP	0	0	0	CPU
	Accelerator	18563	18609	18518	CPU|DSP	0	0	0	CPU
	Misc Accelerator	101	102	100	CPU|DSP	0	0	0	CPU
	layer_000 (Name:data Type:data)	145	148	142	DSP	1	1	1	CPU
	layer_001 (Name:conv1 Type:convol	0	0	0	DSP	12340	12368	12312	CPU
4	layer_002 (Name:relu1 Type:neuron)	1496	1499	1493	DSP	163	125	121	CPU
	...								5
	layer_020 (Name:fc8 Type:fully_conr	469	487	452	DSP	2876	2914	2838	CPU
	layer_021 (Name:prob Type:softmax	66	67	65	DSP	24	24	24	CPU

CSV Benchmark Results File (cont.)

CSV 3: Execution metrics
Profiling level: Detailed

- Misc Accelerator measures the total execution time spent on the accelerator core for optimization elements not specified by SNPE. For example, in the case of DSP this represents the exection time spent on additional layers added by the accelerator to acheive optimal performance. Currently only available for DSP and AIP runtime. Will appear as 0 for other runtimes.

SNPE SDK version:	1.35.0.698								
Configuration used:									
Name	AlexNet								
HostRootPath	/home/user/Workspace/Framework/SNPE/snpe-1.35.0.698/benchmarks/alexnet								
HostResultsDir	/home/user/Workspace/Framework/SNPE/snpe-1.35.0.698/benchmarks/alexnet/results								
Model:Dlc	/home/user/Workspace/Framework/SNPE/snpe-1.35.0.698/models/alexnet/dlc/bvlc_alexnet.dlc								
...									
Device Info:									
ro.product.name	aikit								
		DSP_timing(2 runs)				CPU_timing(2 runs)			
		avg (us)	max (us)	min (us)	runtime	avg (us)	max (us)	min (us)	runtime
	Load	189	193	186	CPU	396	578	214	CPU
	Deserialize	393515	393600	393431	CPU	407482	411082	403882	CPU
2	Create	3574698	3619728	3529669	CPU	426386	432298	420474	CPU
	Init	3968706	4013692	3923720	CPU	840086	842014	838158	CPU
	De-Init	43575	49752	37398	CPU	105390	105570	105210	CPU
	Create Network(s)	2892845	2937568	2848123	CPU	54397	56804	51991	CPU
	RPC Init Time	197716	198585	196847	CPU	0	0	0	CPU
	Snpe Accelerator Init Time	183164	184038	182290	CPU	0	0	0	CPU
	Accelerator Init Time	182131	183012	181250	CPU	0	0	0	CPU
	Total Inference Time	19664	19711	19617	CPU|DSP	103385	104477	102293	CPU
	Forward Propagate	19651	19697	19605	CPU|DSP	102238	103356	101120	CPU
3	RPC Execute	19481	19521	19442	CPU|DSP	0	0	0	CPU
	Snpe Accelerator	18863	18900	18826	CPU|DSP	0	0	0	CPU
	Accelerator	18563	18609	18518	CPU|DSP	0	0	0	CPU
	Misc Accelerator	101	102	100	CPU|DSP	0	0	0	CPU
	layer_000 (Name:data Type:data)	145	148	142	DSP	1	1	1	CPU
	layer_001 (Name:conv1 Type:convol	0	0	0	DSP	12340	12368	12312	CPU
4	layer_002 (Name:relu1 Type:neuron)	1496	1499	1493	DSP	123	125	121	CPU
	...								
	layer_020 (Name:fc8 Type:fully_conr	469	487	452	DSP	2876	2914	2838	CPU
	layer_021 (Name:prob Type:softmax	66	67	65	DSP	24	24	24	CPU

5

CSV Benchmark Results File (cont.)

CSV 4: Model Layer Names

- This section contains the list of names of each layer of the neural network model.
- Note that this information will only be present for DSP/GPU runtime only if the profiling level is set to detailed..

SNPE SDK version:	1.35.0.698								
Configuration used:									
Name	AlexNet								
HostRootPath	/home/user/Workspace/Framework/SNPE/snpe-1.35.0.698/benchmarks/alexnet								
HostResultsDir	/home/user/Workspace/Framework/SNPE/snpe-1.35.0.698/benchmarks/alexnet/results								
Model:Dlc	/home/user/Workspace/Framework/SNPE/snpe-1.35.0.698/models/alexnet/dlc/bvlc_alexnet.dlc								
...									
Device Info:									
ro.product.name	aikit								
			DSP_timing(2 runs)				CPU_timing(2 runs)			
			avg (us)	max (us)	min (us)	runtime	avg (us)	max (us)	min (us)	runtime
		Load	189	193	186	CPU	396	578	214	CPU
		Deserialize	393515	393600	393431	CPU	407482	411082	403882	CPU
	2	Create	3574698	3619728	3529669	CPU	426386	432298	420474	CPU
		Init	3968706	4013692	3923720	CPU	840086	842014	838158	CPU
		De-Init	43575	49752	37398	CPU	105390	105570	105210	CPU
		Create Network(s)	2892845	2937568	2848123	CPU	54397	56804	51991	CPU
		RPC Init Time	197716	198585	196847	CPU	0	0	0	CPU
		Snpe Accelerator Init Time	183164	184038	182290	CPU	0	0	0	CPU
		Accelerator Init Time	182131	183012	181250	CPU	0	0	0	CPU
		Total Inference Time	19664	19711	19617	CPU|DSP	103385	104477	102293	CPU
		Forward Propagate	19651	19697	19605	CPU|DSP	102238	103356	101120	CPU
	3	RPC Execute	19481	19521	19442	CPU|DSP	0	0	0	CPU
		Snpe Accelerator	18863	18900	18826	CPU|DSP	0	0	0	CPU
		Accelerator	18563	18609	18518	CPU|DSP	0	0	0	CPU
		Misc Accelerator	101	102	100	CPU|DSP	0	0	0	CPU
		layer_000 (Name:data Type:data)	145	148	142	DSP	1	1	1	CPU
		layer_001 (Name:conv1 Type:convol	0	0	0	DSP	12340	12368	12312	CPU
	4	layer_002 (Name:relu1 Type:neuron)	1496	1499	1493	DSP	123	125	121	CPU
		...					5			
		layer_020 (Name:fc8 Type:fully_conr	469	487	452	DSP	2876	2914	2838	CPU
		layer_021 (Name:prob Type:softmax	66	67	65	DSP	24	24	24	CPU

CSV Benchmark Results File (cont.)

CSV 5: Model Layer Times

- Each row in this section represents the execution time of each layer of the neural network model. For each runtime the average, min and max values are represented.

SNPE SDK version:	1.35.0.698								
Configuration used:									
Name	AlexNet								
HostRootPath	/home/user/Workspace/Framework/SNPE/snpe-1.35.0.698/benchmarks/alexnet								
HostResultsDir	/home/user/Workspace/Framework/SNPE/snpe-1.35.0.698/benchmarks/alexnet/results								
Model:Dlc	/home/user/Workspace/Framework/SNPE/snpe-1.35.0.698/models/alexnet/dlc/bvlc_alexnet.dlc								
...									
Device Info:									
ro.product.name	aikit								
			DSP_timing(2 runs)				CPU_timing(2 runs)			
			avg (us)	max (us)	min (us)	runtime	avg (us)	max (us)	min (us)	runtime
		Load	189	193	186	CPU	396	578	214	CPU
		Deserialize	393515	393600	393431	CPU	407482	411082	403882	CPU
	2	Create	3574698	3619728	3529669	CPU	426386	432298	420474	CPU
		Init	3968706	4013692	3923720	CPU	840086	842014	838158	CPU
		De-Init	43575	49752	37398	CPU	105390	105570	105210	CPU
		Create Network(s)	2892845	2937568	2848123	CPU	54397	56804	51991	CPU
		RPC Init Time	197716	198585	196847	CPU	0	0	0	CPU
		Snpe Accelerator Init Time	183164	184038	182290	CPU	0	0	0	CPU
		Accelerator Init Time	182131	183012	181250	CPU	0	0	0	CPU
		Total Inference Time	19664	19711	19617	CPU|DSP	103385	104477	102293	CPU
		Forward Propagate	19651	19697	19605	CPU|DSP	102238	103356	101120	CPU
	3	RPC Execute	19481	19521	19442	CPU|DSP	0	0	0	CPU
		Snpe Accelerator	18863	18900	18826	CPU|DSP	0	0	0	CPU
		Accelerator	18563	18609	18518	CPU|DSP	0	0	0	CPU
		Misc Accelerator	101	102	100	CPU|DSP	0	0	0	CPU
		layer_000 (Name:data Type:data)	145	148	142	DSP	1	1	1	CPU
		layer_001 (Name:conv1 Type:convol	0	0	0	DSP	12340	12368	12312	CPU
	4	layer_002 (Name:relu1 Type:neuron)	1496	1499	1493	DSP	123	125	121	CPU
		...					5			
		layer_020 (Name:fc8 Type:fully_conr	469	487	452	DSP	2876	2914	2838	CPU
		layer_021 (Name:prob Type:softmax	66	67	65	DSP	24	24	24	CPU

Run Benchmarking

Running …

An Object Detector Demo

Part 1 : Get Camera Stream by Camera2

Get Camera Stream: Add Camera Permissions

- In order to open camera, the application should have the privilege of using Camera. Modify the AndroidManifest.xml and add contents below:

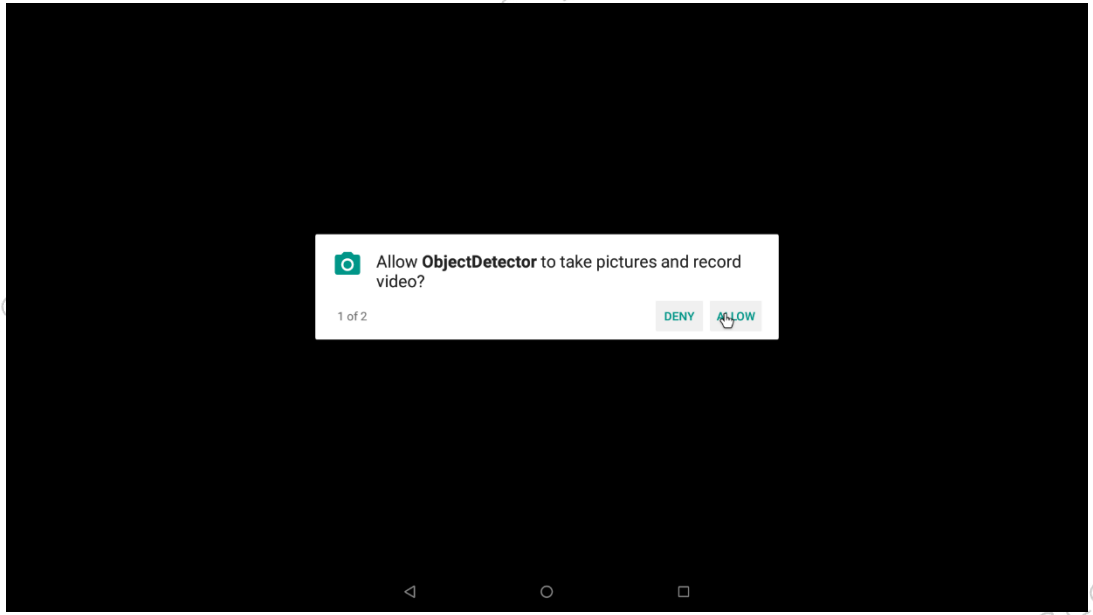
```
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
          package="com.thundersoft.ai.objectdetector">
    <uses-permission android:name="android.permission.CAMERA"/>
    <uses-permission android:name="android.permission.WRITE_EXTERNAL_STORAGE"/>
</manifest>
```

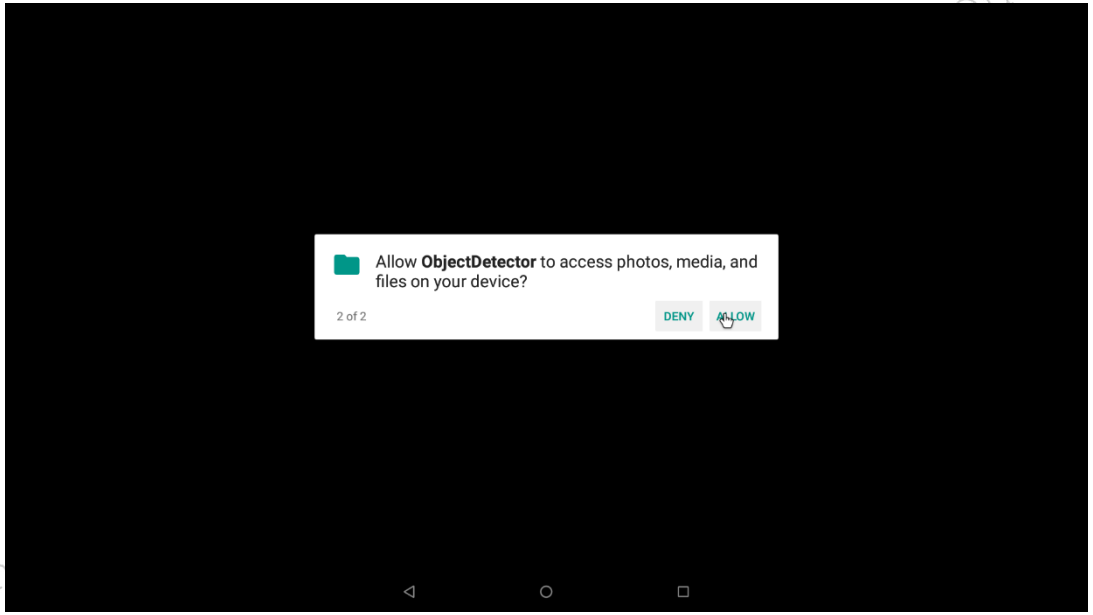
- Call requestPersionsion() in onCreate():

```
private void requestPermission() {
    if (Build.VERSION.SDK_INT >= Build.VERSION_CODES.M) {
        if (shouldShowRequestPermissionRationale(PERMISSION_CAMERA) ||
            shouldShowRequestPermissionRationale(PERMISSION_STORAGE)) {
            Toast.makeText(CameraActivity.this,
            "Camera AND storage permission are required for this demo", Toast.LENGTH_LONG).show();
        }
        requestPermissions(new String[] {PERMISSION_CAMERA, PERMISSION_STORAGE}, PERMISSIONS_REQUEST);
    }
}
```

Get Camera Stream: Add Camera Permissions (cont.)

- When first open the application, please allow it to take pictures and record video.





Get Camera Stream: Open Camera

- Get CameraManager instance and camera ID list

```
final CameraManager manager = (android.hardware.camera2.CameraManager) getSystemService(Context.CAMERA_SERVICE);
String[] cameraIDs = manager.getCameraIdList();
if (cameraIDs != null && cameraIDs.length > 0) {
    mSupportedCameraInfo = new String[cameraIDs.length];
    for (int i=0; i<cameraIDs.length; i++) {
        mSupportedCameraInfo[i] = mCameraInfo[Integer.valueOf(cameraIDs[i])];
    }
}
```

- Open camera

```
final Activity activity = getActivity();
final CameraManager manager = (CameraManager) activity.getSystemService(Context.CAMERA_SERVICE);
if (!cameraOpenCloseLock.tryAcquire(2500, TimeUnit.MILLISECONDS)) {
    throw new RuntimeException("Time out waiting to lock camera opening.");
}
if (ActivityCompat.checkSelfPermission(this.getContext(), Manifest.permission.CAMERA) != PackageManager.PERMISSION_GRANTED) {
    return;
}
manager.openCamera(cameraId, stateCallback, backgroundHandler);
```

Get Camera Stream: Add Layout for Camera Preview

```
<?xml version="1.0" encoding="utf-8"?>
<androidx.constraintlayout.widget.ConstraintLayout xmlns:android="http://schemas.android.com/apk/res/android"
    …
    tools:context=“.MainActivity”>
    <TextureView
        android:id="@+id/camera_preview"
        android:layout_width="0dp"
        android:layout_height="0dp"
        app:layout_constraintBottom_toBottomOf="parent"
        app:layout_constraintLeft_toLeftOf="parent"
        app:layout_constraintRight_toRightOf="parent"
        app:layout_constraintTop_toTopOf="parent" />
</androidx.constraintlayout.widget.ConstraintLayout>
```

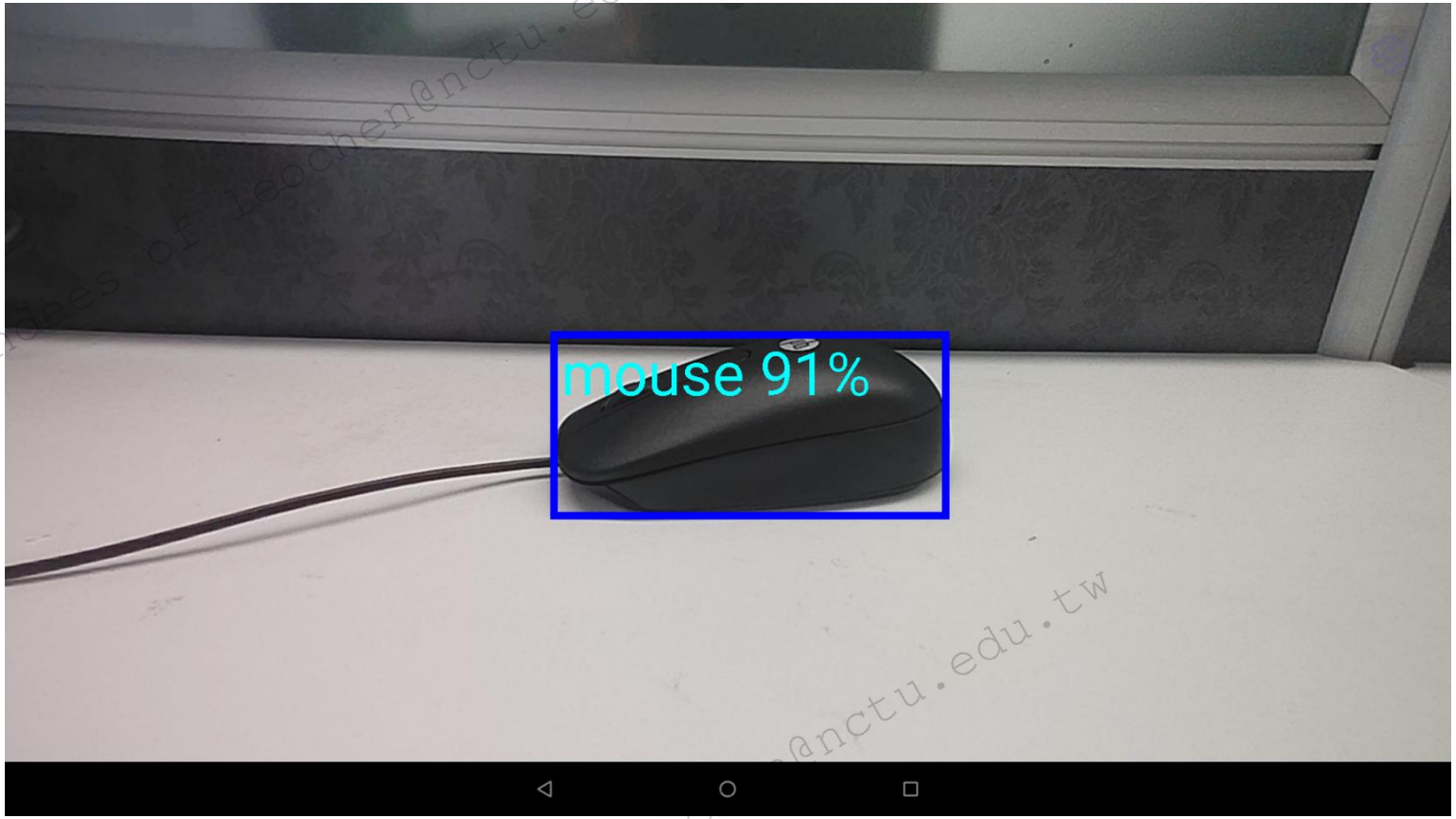
Get Camera Stream: Start Preview

```
final SurfaceTexture texture = textureView.getSurfaceTexture();
assert texture != null;

// We configure the size of default buffer to be the size of camera preview we want.
texture.setDefaultBufferSize(previewSize.getWidth(), previewSize.getHeight());

// This is the output Surface we need to start preview.
final Surface surface = new Surface(texture);

// We set up a CaptureRequest.Builder with the output Surface.
previewRequestBuilder = cameraDevice.createCaptureRequest(CameraDevice.TEMPLATE_PREVIEW);
previewRequestBuilder.addTarget(surface);

// Create the reader for the preview frames.
previewReader =   ImageReader.newInstance(previewSize.getWidth(), previewSize.getHeight(), ImageFormat.YUV_420_888, 2);
previewReader.setOnImageAvailableListener(imageListener, backgroundHandler);
previewRequestBuilder.addTarget(previewReader.getSurface());
```

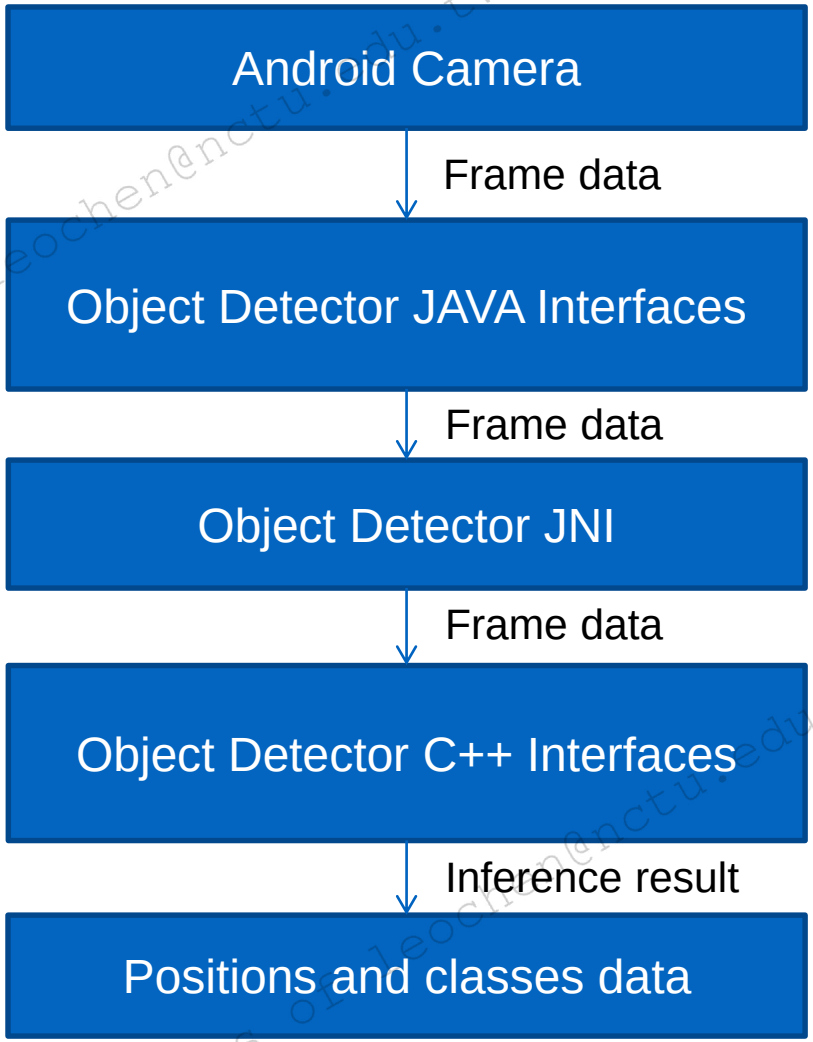
Get Camera Stream: Start Preview (cont.)

```
// Here, we create a CameraCaptureSession for camera preview.
cameraDevice.createCaptureSession(  Arrays.asList(surface, previewReader.getSurface()),
    new CameraCaptureSession.StateCallback() {
    @Override
    public void onConfigured(final CameraCaptureSession cameraCaptureSession) {
        // The camera is already closed
        if (null == cameraDevice) {
            return;
        }
        // When the session is ready, we start displaying the preview.
        captureSession = cameraCaptureSession;
        try {  // Auto focus should be continuous for camera preview.
            previewRequestBuilder.set(   CaptureRequest.CONTROL_AF_MODE,
CaptureRequest.CONTROL_AF_MODE_CONTINUOUS_PICTURE);
            // Flash is automatically enabled when necessary.
            previewRequestBuilder.set(CaptureRequest.CONTROL_AE_MODE, CaptureRequest.CONTROL_AE_MODE_ON_AUTO_FLASH);
            // Finally, we start displaying the camera preview.
            previewRequest = previewRequestBuilder.build();
            captureSession.setRepeatingRequest(previewRequest, captureCallback, backgroundHandler);
        } catch (final CameraAccessException e) {
            LOGGER.e(e, "Exception!");
        }
}
    @Override
    public void onConfigureFailed(final CameraCaptureSession cameraCaptureSession) {
        showToast("Failed");
    }
},  null);
```

Get Camera Stream: Start Preview (cont.)



Object Detection Demo Flow



An Object Detector Demo

Part 2 : JNI Interface

Object Detector JNI Interface

```
package com.qualcomm.ai.objectdetector;

public class ObjectDetector {
    public static final int DEVICE_CPU = 0;
    public static final int DEVICE_GPU = 1;
    public static final int DEVICE_DSP = 2;
    public static final int DEVICE_APU = 3;
    public void init(int device) {
        mPtr = nInit(device);
    }
    public void reInit(int device) {
        mPtr = nReInit(mPtr, device);
    }
    public ClassificationData[] detect(int[] rgb, int width, int height, int channel) {
        return nDetect(mPtr, rgb, width, height, channel);
    }
    public void setConfidence(int value) {
        nSetConfidence(mPtr, value);
    }
    private static native long nInit(int device);
    private static native long nReInit(long obj, int device);
    private static native ClassificationData[] nDetect(long obj, int[] rgb, int w, int h, int c);
    private static native void nSetConfidence(long obj, int value);
    public native boolean nSetAdspLibraryPath(String path);
    private long mPtr;
}
```

An Object Detector Demo

Part 3 : Integrating SNPE

Build SNPE Instance

```
    std::unique_ptr<zdl::DlContainer::IDlContainer> container;
    container =
zdl::DlContainer::IDlContainer::open(zdl::DlSystem::String
(NIKE_MODEL_PATH));
    zdl::SNPE::SNPEBuilder snpeBuilder(container.get());

outputLayers.append("Postprocessor/BatchMultiClassNonMax
Suppression");
    outputLayers.append("add");
    zdl::DlSystem::Runtime_t runtime;
    switch (device) {
        case CPU:
            runtime = zdl::DlSystem::Runtime_t::CPU;
            break;
        case GPU:
            runtime = zdl::DlSystem::Runtime_t::GPU;
            break;
        case DSP:
            runtime = zdl::DlSystem::Runtime_t::DSP;
            break;
        case APU:
            runtime =
zdl::DlSystem::Runtime_t::AIP_FIXED8_TF;
            break;
        default:
            runtime = zdl::DlSystem::Runtime_t::GPU;
            break;
    }
```

```
    if
(!zdl::SNPE::SNPEFactory::isRuntimeAvailable(runtime) ) {
        if
(zdl::SNPE::SNPEFactory::isRuntimeAvailable(zdl::DlSystem::R
untime_t::CPU) ) {
            runtime = zdl::DlSystem::Runtime_t::CPU;
        } else if
(zdl::SNPE::SNPEFactory::isRuntimeAvailable(zdl::DlSystem::R
untime_t::GPU) ) {
            // GPU_FLOAT32_16_HYBRID, rather than
GPU_FLOAT16
            runtime = zdl::DlSystem::Runtime_t::GPU;
        } else {
            runtime = zdl::DlSystem::Runtime_t::DSP;
        }
    }

    zdl::DlSystem::PerformanceProfile_t profile =
zdl::DlSystem::PerformanceProfile_t::HIGH_PERFORMANCE;

    snpe = snpeBuilder.setOutputLayers(outputLayers)
            .setRuntimeProcessor(runtime)
            .setCPUFallbackMode(true)
            .setPerformanceProfile(profile)
            .build();
```

Fill InputTensor(ITensor) and Execute the Network

```
const auto strList = snpe->getInputTensorNames();
auto inputDims = snpe->getInputDimensions((*strList).at(0));
const zdl::DlSystem::TensorShape& inputShape = *inputDims;
size_t rank = inputShape.rank();
int input_size = 1;
for (size_t i=0; i<rank; i++) {
    input_size *= inputShape[i];
    LOGI("input shape %zu : %zu", i, inputShape[i]);
}
inTensor = zdl::SNPE::SNPEFactory::getTensorFactory().createTensor(
        inputDims
);
inMap.add((*strList).at(0), inTensor.get());
zdl::DlSystem::Version_t Version = zdl::SNPE::SNPEFactory::getLibraryVersion();
LOGD("Version:%s", Version.toString().c_str());
```

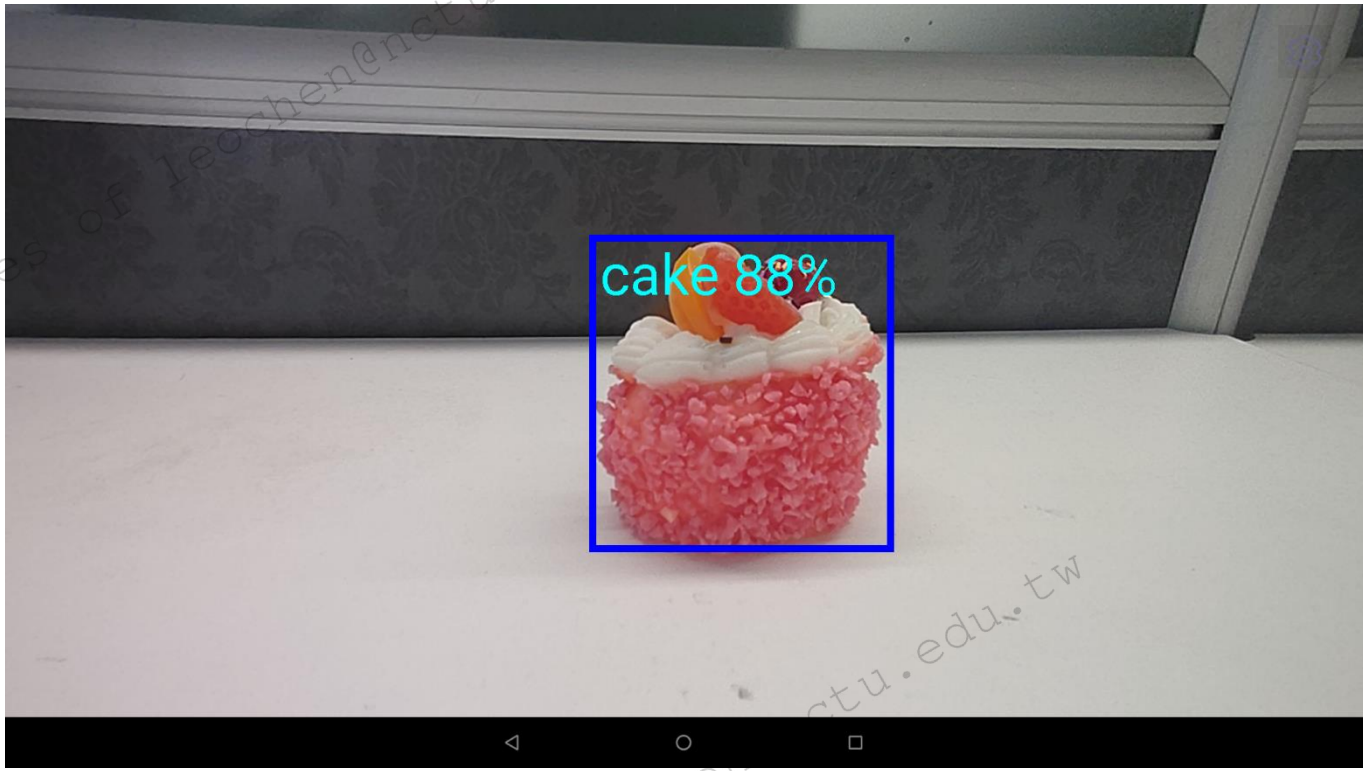
```
std::vector<CLASSIFY_DATA> result;
if (NULL == snpe) {
    LOGE("can not init err!");
    return result;
}
cv::Mat input;
cv::Mat resize_mat;
cv::resize(img, resize_mat, cv::Size(300, 300));
cv::cvtColor(resize_mat, input, CV_BGRA2RGB);
cv::Mat input_norm(MODEL_INPUT_H, MODEL_INPUT_W, CV_32FC3, inTensor.get()->begin().dataPointer());
input.convertTo(input, CV_32F);
cv::normalize(input, input_norm, -1.0f, 1.0f, cv::NORM_MINMAX);
zdl::DlSystem::ITensor* outBoxes = nullptr;
zdl::DlSystem::ITensor* outScores = nullptr;
zdl::DlSystem::ITensor* outClasses = nullptr;
bool ret = snpe->execute(inMap, outMap);
```

Get the Execute Result

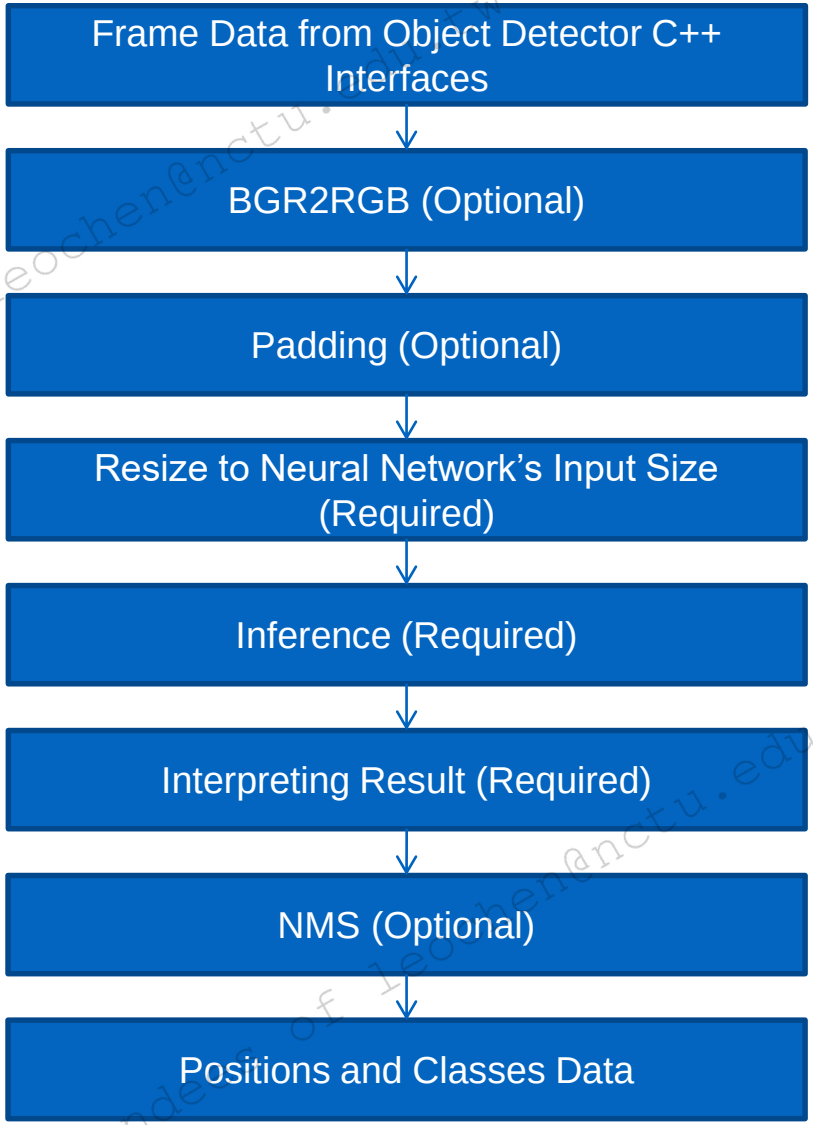
```
if (!ret) {
    const char* const err = zdl::DlSystem::getLastErrorString();
    LOGE("!!!!!! ERROR code :%s", err);
    return result;
}
outBoxes =
outMap.getTensor("Postprocessor/BatchMultiClassNonMax
Suppression_boxes");
outScores =
outMap.getTensor("Postprocessor/BatchMultiClassNonMax
Suppression_scores");
outClasses = outMap.getTensor("detection_classes:0");
zdl::DlSystem::TensorShape boxesShape = outBoxes-
>getShape();
int boxesBatch = boxesShape[0];
int boxesChannel = boxesShape[1];
float x_scale = img.cols / 300.0;
float y_scale = img.rows / 300.0;
const float* score_data = &*(outScores->cbegin());
const float* class_data = &*(outClasses->cbegin());
const float* box_data = &*(outBoxes->cbegin());
```

```
for (int i=0; i<boxesBatch; i++) {
    for (int j=0; j<boxesChannel; j++) {
    float confidence = *score_data++;
    float flabel = *class_data++;
    float yMin = *box_data++;
    float xMin = *box_data++;
    float yMax = *box_data++;
    float xMax = *box_data++;
    if (confidence > mConfidenceThreshold) {
        LOGD("result box[%.3f,%.3f,%.3f,%.3f] lable=%d,
conf=%.3f [%.3f]",xMin, yMin, xMax, yMax, (int)flabel,
confidence, mConfidenceThreshold);
        CLASSIFY_DATA temp;
        temp.rect[0] = xMin * 300 * x_scale;
        temp.rect[1] = yMin * 300 * y_scale;
        temp.rect[2] = xMax * 300 * x_scale - temp.rect[0];
        temp.rect[3] = yMax * 300 * y_scale - temp.rect[1];
        temp.confidence = confidence;
        temp.label = (int)flabel;
        result.push_back(temp);
    }
}
```

Run Demo



Data Flow during Preprocessing, Inference and Postprocessing



AI demos - Face Recognition

Face SDK

- Face SDK is a high-performance middleware adapter for face deep learning algorithms. Face SDK provides developing convenience for application developers. Face SDK can be used on Android device. High-performance enabled on select Qualcomm processors from the 400 tier and above. High-performance algorithm optimization is provided by GPU or DSP(depends on platform). Face SDK provides capabilities as face detection, face recognition, age, gender, emotion detection and eyes blink etc.

- Download method:
  - http://www.45smart.com/app/download

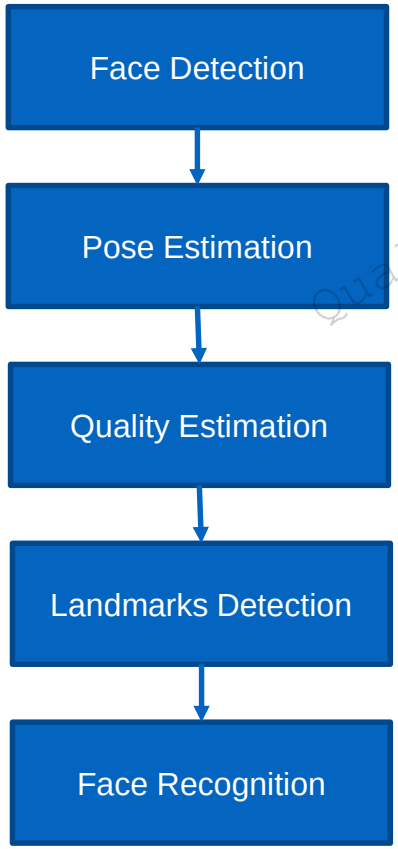
- Face SDK provides C++ and Java Interfaces. It mainly includes two parts: Image API and Video API.We can use the API interfaces above to get face detection, face recognition, face attributes recognition, live detecion results.Currently, the AI engine only supports Qualcomm® Neural Processing SDK and it will support Tensorflow Lite in the future. Face SDK can run on both Linux and Android operation system.

Sample: Face ID WorkFlow

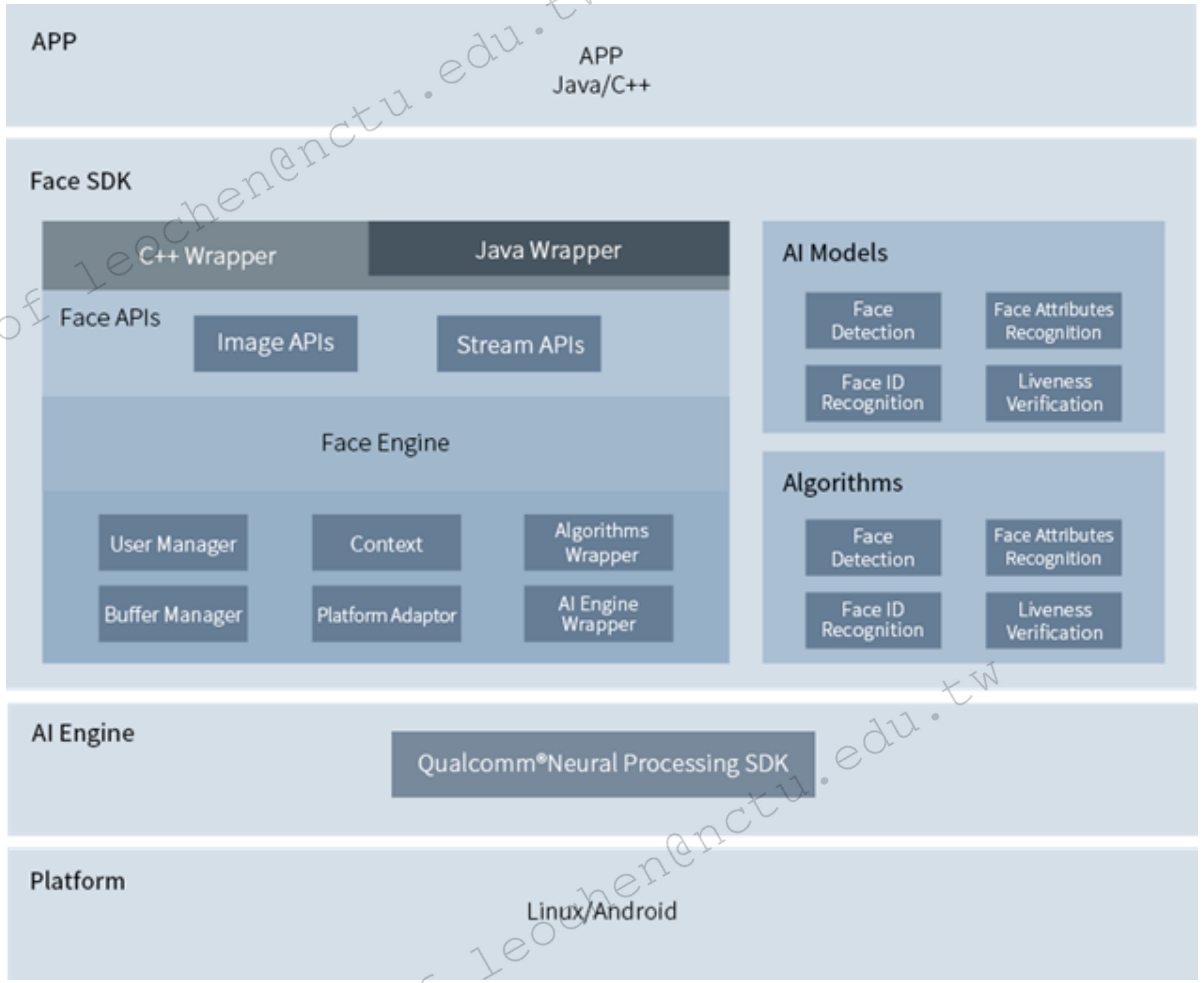
- The face ID workflow has been shown on the right. It includes 5 main parts, face detection module, pose estimation module, quality estimation module, landmarks detection module and face recognition module.

- In order to do face recognition, the algorithm first needs to detect where the faces are. Then, estimate the pose of faces and quality of image. If the face posture is within valid range and image's quality is good enough, the algorithm will do landmarks detection. Finally, the face ID can be got after executing face recognition module.

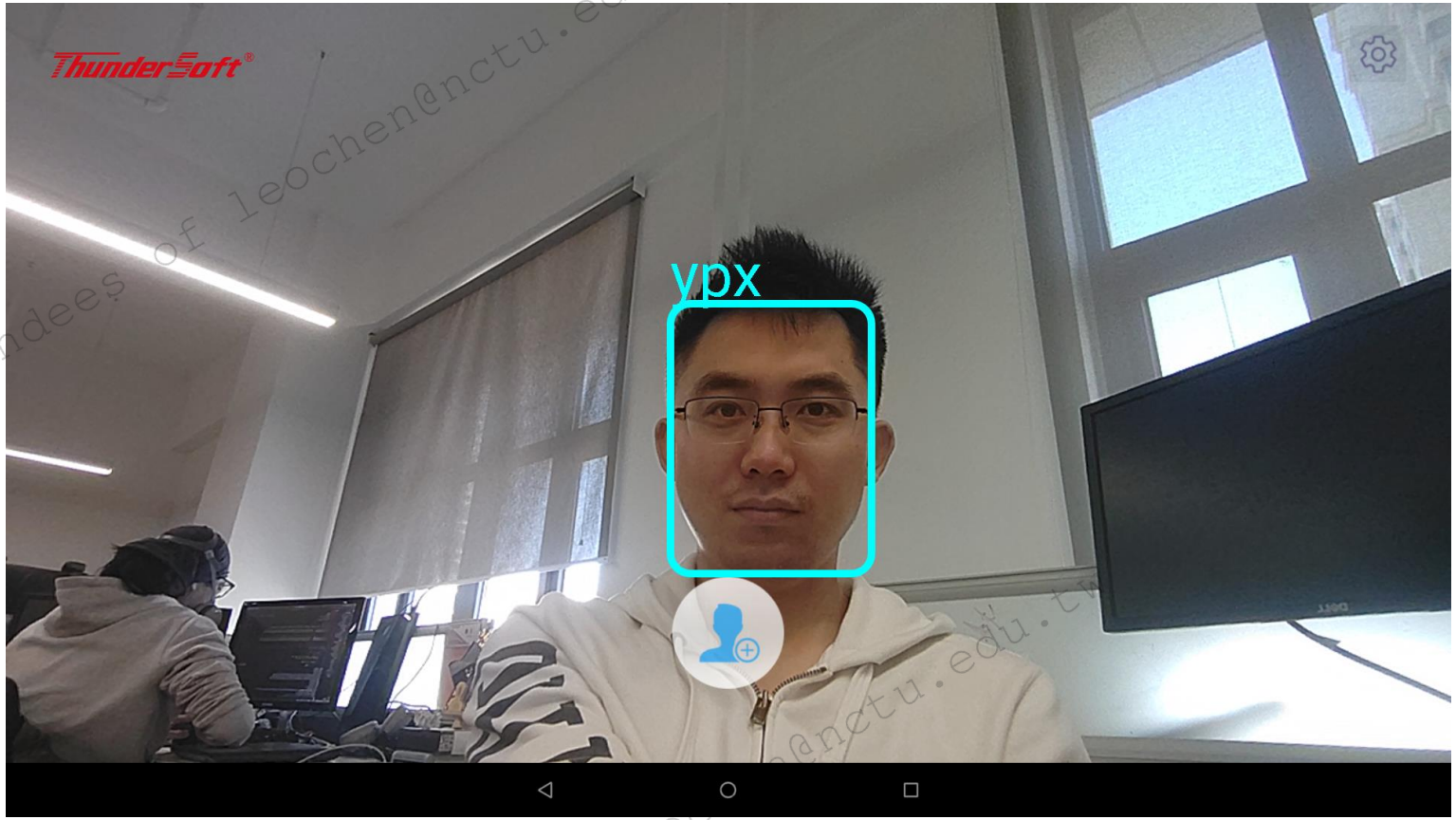
- In conclusion, there are various algorithms running together. For low level chipset SDM 450, its calculation capability is limited. Optimizing each algorithm module plays a key role in good user experience. Doing optimizations for neural networks, reducing workloads and improved running performance are very important.



Face SDK framework



Running Face Recognition Demo



Solve SNPE Problem Skills

Tips 1 How to Deal with Loading Model Failed

- 1. Use getLastErrorString() to check and fix it by analyzing logs printing.
- 2. Use CPU runtime to check what error occurs.
- 3. Check Limitations items.
  - <workspace>/snpe-<version>/doc/html/limitations.html

Tips 2 Focus MaxPerGPUSize When Using GPU Runtime

- 1. The MaxPerGPUSize is dependent on Qualcomm Adreno™ GPU type and the values are given below and you can also find it in limitations chapter.
  - A330: 8192
  - A430, A530, A630: 16384

- 2. Optimizing and reducing parameters in deep learning networks to reduce total amount of calculations.

Tips 3 Optimize Inference Speed

- 1. Optimize your Model
  - A) Reduce parameters of neural network
  - B) Less to use FC layer
- 2. Use proper runtime combinations

Tips 4 Choose a Quantized One or Non-Quantized One

- Two conditions:
- For quantized model:
  - CPU/GPU runtime --> Load Slow
  - DSP runtime --> Load quickly
- For non-quantized model:
  - CPU/GPU runtime --> Load Quickly
  - DSP --> Load Slow

Tips 5 Does X86 Support SNPE?

- Yes, but it only supports CPU runtime. SNPE SDK also provides an example for running on X86 platform.

Q & A

ThunderSoft®

Edge Computing & On-Device AI Enabler

Qualcomm

Thank You