Running ModelSDK on an PyTorch ResNex101 classifier model

How to quantize and compile a PyTorch ResNext101 classifier model using Palette 1.6.

This tutorial shows how to run default Post-Training Quantization (PTQ), evaluate the quantized model and then compiling for the output .tar.gz archive.

Design Flow

Start the SDK docker container - just reply with './' when asked for the work directory:

```
python ./start.py
```

The output in the console should look like this:

```
user@ubmsh:~/tutorials/modelsdk_pytorch$ ./start.py
Set no_proxy to localhost,127.0.0.0
Using port 49152 for the installation.
Checking if the container is already running...
Enter work directory [/home/user/tutorials/modelsdk_pytorch]: ./
Starting the container: palettesdk_1_6_0_Palette_SDK_master_B163
Checking SiMa SDK Bridge Network...
SiMa SDK Bridge Network found.
Creating and starting the Docker container...
b376b8672572336234911037153 72c56d56d0403ca3c0da4d68a3cde2c7c6a27
Successfully copied 3.07kB to
/home/user/tutorials/modelsdk_pytorch/passwd.txt
Successfully copied 3.07kB to
palettesdk_1_6_0_Palette_SDK_master_B163:/etc/passwd
Successfully copied 2.56kB to
/home/user/tutorials/modelsdk_pytorch/shadow.txt
Successfully copied 2.56kB to
palettesdk_1_6_0_Palette_SDK_master_B163:/etc/shadow
Successfully copied 2.56kB to
/home/user/tutorials/modelsdk_pytorch/group.txt
Successfully copied 2.56kB to
palettesdk_1_6_0_Palette_SDK_master_B163:/etc/group
Successfully copied 3.58kB to
/home/user/tutorials/modelsdk_pytorch/sudoers.txt
Successfully copied 3.58kB to
palettesdk_1_6_0_Palette_SDK_master_B163:/etc/sudoers
Successfully copied 2.05kB to
palettesdk_1_6_0_Palette_SDK_master_B163:/home/docker/.simaai/.port
user@b376b8672572:/home$
```

Navigate to the workspace:

```
cd docker/sima-cli
```

The PyTorch model

Download the PyTorch ResNext101 model:

```
cd pyt
python make_resnext.py
cd ..
```

The PyTorch model is trained on the ImageNet dataset using images that have been resized to 224(W)x224(H).

Pre and post-processing

You must know what pre- and post-processing was applied during the training of the floating-point model. For this model the pre-processing was division by 255 to bring all pixels into the range 0 to 1, followed by means subtraction using [0.485, 0.456, 0.406] and division by the standard deviation values of [0.229, 0.224, 0.225].

The post-processing is just simply an argmax function to find the class with the highest probability.

Calibration data

We need a small number of samples from the training dataset to use during the quantization process. These should be randomly chosed from the training dataset. For this tutorial, the calibration samples are not supplied as image files but are instead wrapped into a single numpy file (calib_data.npz) as numpy arrays.

Unpacking the calibration data numpy file is straightforward - it contains a single array called 'x':

```
with np.load(args.calib_data) as data:
    calib_images = data['x']
    print('Number of calibration images: ', calib_images.shape[0])
```

Test data and labels

If we want to test the results of quantization and compare it to the original floating-point model (highly recommended!) then we will need some test data and ground truth labels. Like the calibration data, these are also packed into a single numpy file (test_data.npz) which contains 2 arrays: 'x' is the test data and 'y' are the integer labels corresponding to the 1000 ImageNet classes.

```
with np.load(args.test_data) as data:
    test_images = data['x']
    labels = data['y']
    print('Number of test images: ', test_images.shape[0])
```

> *Note: There are only 50 samples and labels in the test_data.npz numpy file to simplify this tutorial which will impact accuracy results - the complete ImageNet validation dataset of 50k images should actually be used.*

Test the floating-point PyTorch model

An optional step is to run inference of the original PyTorch model and check the output to ensure that it is functional and to create a baseline reference. The PyTorch framework is included in the SDK docker, so we can run the floating-point model. The run_pytorch.py script includes pre- and postprocessing.

```
user@b376b8672572:/home/docker/sima-cli$ python run_pytorch.py
```

The predictions and actual ground truth labels are writen into a file called 'run_pytorch.log' and a final accuracy score is given:

```
2025-05-02 09:49:15 - INFO - Loading model from
./pyt/resnext101_32x8d_wsl.pt
2025-05-02 09:49:16 - INFO - Model loaded and set to evaluation mode
2025-05-02 09:49:16 - INFO - Loading data from ./test_data.npz
2025-05-02 09:49:16 - INFO - Starting evaluation on 50 samples
2025-05-02 09:49:16 - INFO - Sample 0: predicted 65, label 65
2025-05-02 09:49:17 - INFO - Sample 1: predicted 795, label 970
2025-05-02 09:49:17 - INFO - Sample 2: predicted 230, label 230
2025-05-02 09:49:17 - INFO - Sample 3: predicted 809, label 809
2025-05-02 09:49:17 - INFO - Sample 4: predicted 516, label 516
2025-05-02 09:49:18 - INFO - Sample 5: predicted 57, label 57
2025-05-02 09:49:18 - INFO - Sample 6: predicted 334, label 334
2025-05-02 09:49:18 - INFO - Sample 7: predicted 700, label 415
2025-05-02 09:49:18 - INFO - Sample 8: predicted 674, label 674
2025-05-02 09:49:18 - INFO - Sample 9: predicted 332, label 332
2025-05-02 09:49:19 - INFO - Sample 10: predicted 109, label 109
2025-05-02 09:49:19 - INFO - Sample 11: predicted 286, label 286
2025-05-02 09:49:19 - INFO - Sample 12: predicted 370, label 370
2025-05-02 09:49:19 - INFO - Sample 13: predicted 757, label 757
2025-05-02 09:49:19 - INFO - Sample 14: predicted 595, label 595
2025-05-02 09:49:20 - INFO - Sample 15: predicted 147, label 147
2025-05-02 09:49:20 - INFO - Sample 16: predicted 108, label 108
2025-05-02 09:49:20 - INFO - Sample 17: predicted 23, label 23
2025-05-02 09:49:20 - INFO - Sample 18: predicted 478, label 478
2025-05-02 09:49:20 - INFO - Sample 19: predicted 517, label 517
2025-05-02 09:49:21 - INFO - Sample 20: predicted 334, label 334
2025-05-02 09:49:21 - INFO - Sample 21: predicted 167, label 173
2025-05-02 09:49:21 - INFO - Sample 22: predicted 948, label 948
```

```
2025-05-02 09:49:21 - INFO - Sample 23: predicted 727, label 727
2025-05-02 09:49:21 - INFO - Sample 24: predicted 23, label 23
2025-05-02 09:49:22 - INFO - Sample 25: predicted 846, label 846
2025-05-02 09:49:22 - INFO - Sample 26: predicted 270, label 270
2025-05-02 09:49:22 - INFO - Sample 27: predicted 166, label 167
2025-05-02 09:49:22 - INFO - Sample 28: predicted 55, label 55
2025-05-02 09:49:22 - INFO - Sample 29: predicted 858, label 858
2025-05-02 09:49:22 - INFO - Sample 30: predicted 324, label 324
2025-05-02 09:49:23 - INFO - Sample 31: predicted 573, label 573
2025-05-02 09:49:23 - INFO - Sample 32: predicted 150, label 150
2025-05-02 09:49:23 - INFO - Sample 33: predicted 981, label 981
2025-05-02 09:49:23 - INFO - Sample 34: predicted 586, label 586
2025-05-02 09:49:23 - INFO - Sample 35: predicted 887, label 887
2025-05-02 09:49:24 - INFO - Sample 36: predicted 26, label 32
2025-05-02 09:49:24 - INFO - Sample 37: predicted 398, label 398
2025-05-02 09:49:24 - INFO - Sample 38: predicted 795, label 777
2025-05-02 09:49:24 - INFO - Sample 39: predicted 74, label 74
2025-05-02 09:49:24 - INFO - Sample 40: predicted 431, label 516
2025-05-02 09:49:25 - INFO - Sample 41: predicted 756, label 756
2025-05-02 09:49:25 - INFO - Sample 42: predicted 129, label 129
2025-05-02 09:49:25 - INFO - Sample 43: predicted 198, label 198
2025-05-02 09:49:25 - INFO - Sample 44: predicted 256, label 256
2025-05-02 09:49:25 - INFO - Sample 45: predicted 725, label 725
2025-05-02 09:49:26 - INFO - Sample 46: predicted 565, label 565
2025-05-02 09:49:26 - INFO - Sample 47: predicted 166, label 167
2025-05-02 09:49:26 - INFO - Sample 48: predicted 717, label 717
2025-05-02 09:49:26 - INFO - Sample 49: predicted 394, label 394
2025-05-02 09:49:26 - INFO - Evaluation complete: 42 correct out of 50
(84.00%)
```

Quantize & Compile

The run_modelsdk.py script will do the following:

- load the floating-point PyTorch model.
- quantize using pre-processed calibration data and default quantization parameters.
- test the quantized model accuracy using pre-processed images. An accuracy value is generated and should be compared to the value obtained from the floating-point model.
- compile to generate a tar.gz

```
python run_modelsdk.py
```

If this runs correctly, the final output messages in the console will be like this:

```
2025-05-02 04:56:22,533 - afe.backends.mpk.interface - INFO -
============================
2025-05-02 04:56:22,533 - afe.backends.mpk.interface - INFO - Compilation
summary:
2025-05-02 04:56:22,534 - afe.backends.mpk.interface - INFO - ------------
```

```
-----------------
2025-05-02 04:56:22,534 - afe.backends.mpk.interface - INFO - Desired batch
size: 1
2025-05-02 04:56:22,534 - afe.backends.mpk.interface - INFO - Achieved
batch size: 1
2025-05-02 04:56:22,534 - afe.backends.mpk.interface - INFO - -------------
-----------------
2025-05-02 04:56:22,534 - afe.backends.mpk.interface - INFO - Plugin
distribution per backend:
2025-05-02 04:56:22,534 - afe.backends.mpk.interface - INFO -   MLA : 1
2025-05-02 04:56:22,534 - afe.backends.mpk.interface - INFO -   EV74: 5
2025-05-02 04:56:22,534 - afe.backends.mpk.interface - INFO -   A65 : 0
2025-05-02 04:56:22,534 - afe.backends.mpk.interface - INFO - -------------
-----------------
2025-05-02 04:56:22,534 - afe.backends.mpk.interface - INFO - Generated
files: resnext101_32x8d_wsl_mpk.json, resnext101_32x8d_wsl_stage1_mla.elf,
mla.json, preproc.json, resnext101_32x8d_wsl_stage1_mla_stats.yaml,
detess_dequant.json
```

The compiled model is written to build/resnext101_32x8d_wsl/resnext101_32x8d_wsl_mpk.tar.gz.

run_modelsdk.py Code Walkthrough

The PyTorch floating-point model is loaded using the load_model API.

```
    # load the floating-point model
    input_names = ['x']
    input_shapes = [(1, 3, 224, 224)]
    importer_params: ImporterParams = pytorch_source(args.model_path,
input_names, input_shapes)
    loaded_net = load_model(importer_params)
```

The calibration images are read from the calib_data.npz numpy file and then preprocessed before being appended to an iterable variable (a list in the case). Each calibration sample is provided as a dictionary, the key is the name of the input that will have the preprocessed calibration sample applied to it, the value is the preprocessed sample:

```
    # calibration data
    with np.load(args.calib_data) as data:
        calib_images = data['x']
        logger.info("Number of calibration images: %d",
calib_images.shape[0])

    calibration_data = []
    for img in calib_images:
        preproc_image = _preprocessing(img)
        calibration_data.append({input_names[0]: preproc_image})
```

> *It is very important that the same preprocessing used during training of the model is also applied to the calibration data*

The LoadedNet model is then quantized using default quantization parameters. We can also optionally save the quantized model which will allow us to open it with Netron if required:

```
    # quantize
    quant_model = loaded_net.quantize(
        calibration_data=length_hinted(len(calib_images),
calibration_data),
        quantization_config=default_quantization,
        model_name=filename,
        log_level=logging.WARN
    )
    quant_model.save(model_name=filename, output_directory=output_path)
    logger.info("Quantized and saved to %s", output_path)
```

An optional, but highly recommended, step is to evaluate the quantized model to assess the impact of quantization:

```
    # evaluate quantized model
    logger.info("Evaluating quantized model...")
    with np.load(args.test_data) as data:
        test_images = data['x']
        labels = data['y']
        logger.info("Number of test images: %d", test_images.shape[0])

    correct = 0
    for i, img in enumerate(test_images):
        img = _preprocessing(img)
        test_data = {input_names[0]: img}
        prediction = quant_model.execute(test_data, fast_mode=True)
        prediction = np.argmax(prediction)
        if prediction == labels[i]:
            correct += 1
    accuracy = correct / len(labels) * 100
    logger.info("Correct predictions: %d Accuracy: %.2f%%", correct,
accuracy)
```

Then finally we compile the quantized model:

```
    quant_model.compile(
        output_path=output_path,
        batch_size=args.batch_size,
        log_level=logging.INFO)
    logger.info("Compiled model written to %s", output_path)
```

This creates a .tar.gz archive that contains the .elf file that will be executed on the Machine Learning Accelerator (MLA).

Next Steps

Once our model is quantized, evaluated and compiled into a .tar.gz archive, it can now be incorporated into a full pipeline.

- Import the compiled model into the Edgematic tool and build the pipeline using graphical design entry.
- Run the 'mpk project create' command to build a small Gstreamer pipeline as a starting point

```
user@b376b8672572:/home/docker/sima-cli$ mpk project create --model-path
./build/resnext101_32x8d_wsl/resnext101_32x8d_wsl_mpk.tar.gz --input-
resource ./test_data/img_%d.png
```