The University of Texas at Austin
Electrical and Computer Engineering
Cockrell School of Engineering

Spring 2021

ECE 379K: Machine Learning and Data Analytics for Edge AI
HW3 Assigned: Mar 30 DUE: Apr 13 (11:59:59pm CST)

Work in groups of two students. At the end of the PDF file add a paragraph where you clearly describe each member's contribution

Introduction

In this homework, you will deploy popular deep learning models on different computational platforms (Odroid MC1 and RaspberryPi 3B+) using two of the most used frameworks for deployment, namely ONNX and TensorFlow Lite. The models will have two versions, each trained using the most popular deep learning frameworks, namely PyTorch and TensorFlow. To compare and contrast various options for training and deployment, you need to report different metrics (e.g., inference latency, accuracy and energy consumption) and analyze the impact of different models deployed on both edge devices. Throughout this homework you will also work with a very popular dataset (CIFAR10) containing 60000 RGB images split into 10 classes.

NOTE:

a) Python 3.7 is required for this assignment as all the necessary libraries for Python 3.7 are already installed on the devices.
b) For all plots in this homework, you need to have a title, labels on both axes, a grid, and a legend explaining the labels for each line in the plot.

Problem 1 [40p]: PyTorch and TensorFlow evaluation

In the first part of this assignment, you will experiment with PyTorch and TensorFlow implementations of VGG11 on the CIFAR10 dataset; this will help you get familiar with both frameworks. As in many real-world software projects, we divide the model code into different folders: ***pytorch*** and ***tensorflow***, and have naming conventions such as ***vgg_pt*** and ***vgg_tf*** to know exactly which framework we use to implement which model. In general, we recommend you to stick to the naming conventions of adding suffixes ***_pt*** for PyTorch models and ***_tf*** for TensorFlow models.

One thing to note is that TensorFlow, starting with TensorFlow 2.0, incorporated a previously independent library called Keras which facilitates easy prototyping and training of DNNs. **The models you will create in this homework will actually be Keras models**, but since Keras functions under the bigger name of TensorFlow, we refer to these models as being TensorFlow models.

Question 1: [12p] Write the TensorFlow equivalent of the VGG11 model and place it in ***HW3_files/models/tensorflow*** folder. For specific details related to this model, consider the PyTorch implementation we give you in ***HW3_files/models/pytorch/vgg_pt.py*** as a starting point.

NOTE:

a) The number of trainable parameters of the PyTorch model and the TensorFlow model should be the same. For TensorFlow, you can use ***model.summary()*** to check the number of trainable parameters. In PyTorch, the number of trainable parameters can be found using the ***summary()*** method from ***torchsummary*** library.
b) Only use ***.add()*** method to add layers to the **Sequential()** model.
c) For activation functions use **Activation()** with the corresponding strings (e.g., ***'relu'*** for ReLU activation, ***'softmax'*** for Softmax activation).
d) When defining ***Conv2D*** and ***Dense*** layers, use appropriate names with the ***'name'*** parameter.

Question 2: [18p] For both PyTorch and TensorFlow implementations of VGG11, train the models using TACC with the given default parameters (i.e., number of epochs, batch size). To do this, you have the help files in the ***HW3_files*** folder; these files are named ***main_pt.py*** and ***main_tf.py.*** By following the ***TODO*** parts, you have to complete the code to make it work. After that, add the necessary code needed for you to complete ***Table 1***.

Use the **Appendix A0** to set up the TACC environment. By following the guidelines in the **Appendix A1**, you should train both models in parallel, on the same computing TACC instance.

Table 1

Framework	Training accuracy [%]	Testing accuracy [%]	Total time for training [s]	Number of trainable parameters	GPU memory during training [MB]
PyTorch					
TensorFlow					

Notes for TensorFlow:

a) To complete the ***TODO*** parts and define the model, you can use the demo code from Lecture 4 as an inspiration source.
b) The ***fit*** method will return its output into a variable called ***history***. To find the loss and accuracy, use ***history.history***, which is a dictionary containing 4 important keys: *loss*, *accuracy*, *val_loss* and *val_accuracy*. Each key contains a list for the values for loss and accuracy for each training epoch.
c) Use ***model.evaluate()*** to evaluate the model on the testing dataset and output the loss and the accuracy of the model (if you mentioned as metric for the ***compile()*** method only ***'accuracy'***).
d) Use ***model.save_weights()*** to save the model weights in the ***.ckpt*** format. As a parameter, give a string: the name of the model with the corresponding ***_tf*** suffix followed by the ***.ckpt*** extension. To load the model later on, define the model instance and then use ***model.load_weights()*** with the same parameter as before.
e) To make the comparison as uniform as possible, use the same loss function and optimizer as the ones used in PyTorch. If in PyTorch they have no parameters, then leave the ones from TensorFlow with the default parameters as well.

Question 3: [10p] Draw (on the same plot) the training accuracy values of both models and, on another plot, their testing accuracies for each epoch. Compare and contrast the results obtained in Q2 and in these plots, explaining the possible tradeoffs of using the two frameworks in terms of time for training, time for inference, GPU memory, complexity of the code needed to define the models and to train them. Explain which framework would you use for training, under which real scenario, while considering GPU memory, training speed, etc.

Problem 2 [60p]: Deployment on the Edge using ONNX

In this part of the homework, you will deploy VGG11 and MobileNet-v1 on edge devices (Odroid MC1 and Raspberry Pi 3B+ platforms) using the Open Neural Network Exchange (ONNX) framework. You can find in the ***HW3_files folder*** the pretrained versions of MobileNet-v1 on CIFAR10 (written using PyTorch and TensorFlow) so you do not have to train them again. Make sure you follow the instructions in **Appendix A2** to install ONNX locally on your machine before you start working on this problem. Use **Appendix A3** to connect to each device correspondingly.

Question 1: [10p] Create two methods inside a new file named ***convert_onnx.py*** to convert the PyTorch and TensorFlow models in ONNX respectively, using the appropriate suffixes (i.e., ***_pt*** for PyTorch models and ***_tf*** for TensorFlow) and the ***.onnx*** file extension for saving the converted models. Consider the following issues:

- For PyTorch, use ***torch.onnx.export()*** method with ***export_params=True*** and ***opset_version=10***. The first three parameters for this function are: the model itself, random input of the appropriate size and the name of the file to export to.
- For TensorFlow, after you define the model and load its weights, use ***convert_keras()*** method from ***keras2onnx*** library. ***convert_keras()*** takes two parameters: the model itself and the model name (obtainable using ***model.name***). After obtaining the ONNX model[1], you can save it using the ***save_model()*** method from the same, ***keras2onnx***, library. ***save_model()*** has two parameters: the previously converted ONNX model and the name of the file to save it to.

Question 2: [20p] Use the ***deploy_onnx.py*** file from the ***HW3_files*** folder to deploy[2] all four ONNX models on the RaspberryPi 3B+ and Odroid MC1 devices. Finish writing the file by completing the ***TODO*** parts. Send all ONNX models (converted in Q1) and the ***deploy_onnx.py*** file to each device over SSH using the ***scp*** command (similar to HW2).

Add some extra code to calculate and report the accuracy of the four ONNX models on the testing dataset. This accuracy must be the same as the one you obtained in Problem 1 for VGG. For MobileNet-v1, the testing accuracy is given in the model definition code for both PyTorch and TensorFlow.

Do inference on the *entire* test dataset available in ***HW3_files/test_deployment***, record the total time required for inference[3] [seconds] and complete ***Table 2***.

Table 2

Framework	Model	Inference time on MC1 [s]/ Accuracy[%]	MC1 RAM memory [MB]	Inference time on RaspberryPi [s]/ Accuracy[%]	RaspberryPi RAM memory [MB]
PyTorch	VGG11				
TensorFlow	VGG11				
PyTorch	MobileNet-v1				
TensorFlow	MobileNet-v1				

NOTE: The memory for RaspberryPi 3B+ and Odroid MC1 is the amount of RAM memory consumed when doing inference. Using ***htop*** check the amount of RAM memory before running inference, and write it down. After you start doing inference, check that value again and write the difference between the current value and the previous one in ***Table 2***.

Question 3: [20p] Draw one plot showing the variation of power consumption over time for both devices and another plot with the temperature measurements of the CPU of both devices. Fill in ***Table 3***.

[1] You can view the human readable representation of the graph using ***onnx.helper.printable_graph(model.graph)*** using the ONNX converted model. To load the ONNX model use the ***load()*** method from the ***onnx*** library.

[2] To run the code on RaspberryPi 3B+ and Odroid MC1 use ***python3 your_code.py***

[3] The actual inference is when ***sess.run()*** is being executed.

Table 3

Framework	Model	MC1 total energy consumption [J]	RaspberryPi total energy consumption [J]
PyTorch	VGG11		
TensorFlow	VGG11		
PyTorch	MobileNet-v1		
TensorFlow	MobileNet-v1		

NOTE: The power measurement for both RaspberryPi 3B+ and Odroid MC1 needs to be done as in HW2. Check **Appendix A4** for how to collect temperature measurements from RaspberryPi. Because you have multiple temperature sensors for Odroid MC1, the temperature you will use throughout this homework will be the average of the 4 big core temperatures.

Question 4: [10p] Based on the data collected in ***Tables 2*** and ***3***, as well as the plots, compare the two types of edge device. Based on the data you collected:

a) Which edge device would you prefer to use for inference? Explain.
b) Would you rather use VGG11 or MobileNet-v1 to be deployed for inference on edge devices? Explain.

BONUS Problem 3 [20p]: ONNX versus TensorFlow Lite

In this problem you will use RaspberryPi 3B+ and Odroid MC1 with two different deployment frameworks: ONNX for the PyTorch models and TensorFlow Lite for the TensorFlow models. If you want to run the code locally, install TensorFlow Lite as suggested here.

Question 1: [4p] Create a new file called ***convert_tflite.py*** and make the conversion from Keras model to TensorFlow Lite. The first step is to get an appropriate converter from ***tensorflow.lite.TFLiteConverter*** library using the ***from_keras_model()*** method which takes the Keras model (with its corresponding weights already loaded) as input. The next step is to obtain the TensorFlow Lite model using the ***convert()*** method from the previously obtained converter. This model can be written as a binary file, naming it using the ***.tflite*** file extension.

Question 2: [8p] Use the ***deploy_tflite.py*** file from ***HW3_files*** to deploy on RaspberryPi 3B+ and Odroid MC1 VGG11 and MobileNet-v1 (the TensorFlow Lite version for each model). Finish writing the file by completing the ***TODO*** parts. Do inference on the entire test dataset available in ***HW3_files/test_deployment***, record the time for inference[4] [seconds] and complete ***Table 4***.

Add extra code to calculate the accuracy of the TensorFlow Lite models on the testing dataset. This accuracy must be the same one you obtained in Problem 1 for VGG. For MobileNet-v1, the testing accuracy is given in the model definition code.

[4] The actual inference is when ***interpreter.invoke()*** is being executed.

Table 4

Framework	Model	Inference time MC1 [s]/ Accuracy[%]	MC1 RAM memory[5] [MB]	Inference time RaspberryPi [s]/ Accuracy[%]	RaspberryPi RAM memory [MB]
TensorFlow	VGG11				
TensorFlow	MobileNet-v1				

Question 3: [8p] Draw one plot with the power consumption over time for both devices and another plot with the thermal measurements of the CPU of both devices. Compare the inference time, memory consumption during inference, energy and thermal readings with the previous ONNX deployment of the TensorFlow models on RaspberryPi 3B+ and Odroid MC1 (from **Problem 2, Q2&3**). Which framework delivers better results on each edge device? Between the PyTorch models deployed with ONNX (from **Problem 2, Q2&3**) and the TensorFlow models deployed with TensorFlow Lite (on both RaspberryPi and Odroid), which framework and models tend to be better? Explain.

Submission Instructions/Hints

1. Please include your solutions into a single zip file named <**GroupNo_FirstName1_LastName1_FirstName2_LastName2**>**.zip**. The zip file should have:
   - A single PDF file containing all your results and discussions.
   - A *readme.txt* file listing the purpose of all your items in the zip file.
   - Your code.
2. You should verify your implementation as you proceed.
3. Before you begin this assignment, please read the entire description carefully to make sure you understand it and have all the tools needed to solve the problems.
4. **Start early!** This homework may take longer than you expect to complete.

Good luck!

[5] The Note from Problem 2 Q2 applies here as well

Appendix

A0. Setting up TACC

Open a terminal (or Windows bash/PowerShell for Windows users) and use the command:

```
ssh <your_TACC_username>@maverick2.tacc.utexas.edu
```

You need to introduce your password and then access the DuoMobile app for a second security code required to access UT TACC instances. While in TACC, **always use the $WORK directory**! To change directory to $WORK use the `cdw`[6] command.

1) Load the Python3 module in the system with the following command:

```
module load python3
```

2) To validate, you can check if you have Python3 by typing:

```
which python3
```

3) We use *virtualenv*, which is installed by default. You can check it using:

```
which virtualenv
```

4) To avoid package conflicts with other python packages and versions on the machine, we create a virtual environment using *virtualenv*:

```
virtualenv -p '<output_of_which_python3>' $WORK/HW3_virtualenv
```

5) We activate the virtual environment:

```
source $WORK/HW3_virtualenv/bin/activate
```

6) Check your current working directory (you will need it next) using `pwd`; it should give you something like:

```
/work/<number>/<your_TACC_username>/maverick2
```

7) Download ***HW3_files.zip*** from Canvas and unzip it.
8) Open a new terminal on your computer and go to the location where you downloaded HW1_files:

```
cd Downloads/
```

9) Move the files from your computer to Maverick2 using `scp`[6]

```
scp -r HW3_files <your_TACC_username>@maverick2.tacc.utexas.edu:<pwd_output>
```

10) After moving the files from your computer to Maverick2, on the terminal for TACC install the required packages from ***requirements.txt:***

```
pip install -r $WORK/HW3_files/requirements.txt
```

11) Edit the following file using `vim`:

```
vim $WORK/HW3_virtualenv/lib/python3.7/site-
packages/tensorflow/python/keras/utils/data_utils.py
```

12) Press INSERT or letter 'I'. Change line 178 from `cache_dir=None` to

```
cache_dir='/work/<number>/<your_TACC_username>/maverick2/HW3_files/dataset_keras'
```

13) Save and exit pressing ESC and then typing

```
:wq
```

14) Make sure to create the `dataset_keras` directory inside `HW3_files` using

```
cd $WORK/HW3_files
mkdir dataset_keras
```

[6] `scp -r` is used to move recursively all subdirectories and files.

A1. Running tasks in parallel on TACC machines

Running a single Python program on a computing node with 4 GPUs is a waste of resources. As this class focuses on efficiency from different perspectives, it comes naturally to teach you how to run commands in parallel, each of them on a different GPU.

1) To run in parallel, in ***config.slurm*** provided in ***HW3_files*** place the two commands you want to run in parallel one after the other.
2) Add at the beginning of a command **CUDA_VISIBLE_DEVICES=0** to make that specific command to run on the first GPU(i.e., GPU0). Do the same, with the rest of the commands, assigning them to run on the other GPUs 1, 2 or 3.
3) Add **"&"** at the end of each command; this will make the processes run in the background.
4) After inserting all commands, write **"wait"**. If the processes are running in the background, you want to wait until all of them finish running and not kill the entire script (with all its running processes) when the first command from the background finishes running.
5) Example:

```
CUDA_VISIBLE_DEVICES=0 python main.py &
CUDA_VISIBLE_DEVICES=1 python main.py &
wait
```

A2. Installing ONNX for Linux

1) The following steps are for starting using Conda. Python 3.7 is highly recommended.
   - `$ sudo apt-get update`
   - `$ sudo apt-get install -y python3 python3-dev python3-setuptools gcc libtinfo-dev zlib1g-dev build-essential cmake libedit-dev libxml2-dev`
   - `$ conda create --name ml python=3.7`
   - `$ conda activate ml`
2) Make sure you have vim installed or run `$ sudo apt install vim`
3) While inside the virtual environment created in step 1), execute the following
   - `$ pip install onnx keras2onnx onnxruntime`

A3. Connecting to edge devices

1) To connect to RaspberryPi 3B+ use

```
ssh pi@<rpi_IP_address>
```

2) To connect to Odroid MC1 use

```
ssh odroid@<odroid_IP_address>
```

A4. Monitoring temperature on RaspberryPi 3B+

1) The code below can be run on RaspberryPi 3B+ using ***python3 code.py***

```
import gpizero
cpu_temp = gpizero.CPUTemperature().temperature
```

2) You can add an infinite loop and log the data to a file. You can also use ***time.sleep()*** to have a constant rate of collecting data.