Exercise_Multi_Device_Plugin_and_the_DevCloud

July 1, 2020

1 Exercise: Multi Device Plugin and the DevCloud

Now that we've walked through the process of using the **Multi device plugin** to load a model on all three hardware devices, CPU, GPU, and VPU, you will have the opportunity to do this yourself with the addition of running inference on an image using 3 device combinations.

In this exercise, you will do the following: 1. Write a Python script to load a model and run inference 1000 times on a device on Intel's DevCloud. * Calculate the time it takes to load the model. * Calculate the time it takes to run inference 1000 times. 2. Write a shell script to submit a job to Intel's DevCloud. 3. Submit a job using `qsub` on an **IEI Tank-870** edge node using `MULTI`, run `liveQStat` to view the status of your submitted jobs, and then retrieve and view the results from your job. * One job using `CPU/VPU` as the device. * One job using `GPU/VPU` as the device. * One job using `CPU/GPU/VPU` as the device. 4. Plot and compare the results using bar graphs with `matplotlib` for the following metrics: * Model Loading Time * Inference Time * Frames Per Second (FPS)

Click the **Exercise Overview** button below for a demonstration.

Exercise Overview

IMPORTANT: Set up paths so we can run Dev Cloud utilities You *must* run this every time you enter a Workspace session.

```
In [ ]: %env PATH=/opt/conda/bin:/opt/spark-2.4.3-bin-hadoop2.7/bin:/opt/conda/bin:/usr/local/sb
        import os
        import sys
        sys.path.insert(0, os.path.abspath('/opt/intel_devcloud_support'))
        sys.path.insert(0, os.path.abspath('/opt/intel'))
```

1.1 The Model

We will be using the `vehicle-license-plate-detection-barrier-0106` model for this exercise.

Remember to use the appropriate model precisions for each device:

- GPU - `FP16`
- VPU - `FP16`
- CPU - It is prefered to use `FP32`, but we have to use `FP16` since **GPU** and **VPU** use `FP16`

The model has already been downloaded for you in the `/data/models/intel` directory on Intel's DevCloud.

We will be running inference on an image of a car. The path to the image is `/data/resources/car.png`.

2 Step 1: Creating a Python Script

The first step is to create a Python script that you can use to load the model and perform inference. We'll use the `%%writefile` magic to create a Python file called `inference_on_device.py`. In the next cell, you will need to complete the `TODO` items for this Python script.

`TODO` items:

1. Load the model
2. Get the name of the input node
3. Prepare the model for inference (create an input dictionary)
4. Run inference 1000 times in a loop

If you get stuck, you can click on the **Show Solution** button below for a walkthrough with the solution code.

```
In [ ]: %%writefile inference_on_device.py

        import time
        import cv2
        import numpy as np
        from openvino.inference_engine import IENetwork
        from openvino.inference_engine import IECore
        import argparse

        def main(args):
            model=args.model_path
            model_weights=model+'.bin'
            model_structure=model+'.xml'

            start=time.time()
            model=IENetwork(model_structure, model_weights)

            core = IECore()
            net = core.load_network(network=model, device_name=args.device, num_requests=1)
            load_time=time.time()-start
            print(f"Time taken to load model = {load_time} seconds")

            # Get the name of the input node
            input_name=next(iter(model.inputs))

            # Reading and Preprocessing Image
            input_img=cv2.imread('/data/resources/car.png')
            input_img=cv2.resize(input_img, (300,300), interpolation = cv2.INTER_AREA)
            input_img=np.moveaxis(input_img, -1, 0)

            # Running Inference in a loop on the same image
```

```
    input_dict={input_name:input_img}

    start=time.time()
    for _ in range(1000):
        net.infer(input_dict)

    inference_time=time.time()-start
    fps=100/inference_time

    print(f"Time Taken to run 1000 Inference is = {inference_time} seconds")

    with open(f"/output/{args.path}.txt", "w") as f:
        f.write(str(load_time)+'\n')
        f.write(str(inference_time)+'\n')
        f.write(str(fps)+'\n')

if __name__=='__main__':
    parser=argparse.ArgumentParser()
    parser.add_argument('--model_path', required=True)
    parser.add_argument('--device', default=None)
    parser.add_argument('--path', default=None)

    args=parser.parse_args()
    main(args)
```

Show Solution

2.1 Step 2: Creating a Job Submission Script

To submit a job to the DevCloud, you'll need to create a shell script. Similar to the Python script above, we'll use the `%%writefile` magic command to create a shell script called `inference_model_job.sh`. In the next cell, you will need to complete the `TODO` items for this shell script.

`TODO` items: 1. Create three variables: * `DEVICE` - Assign the value as the first argument passed into the shell script. * `MODELPATH` - Assign the value as the second argument passed into the shell script. * `SAVEPATH` - Assign the value as the third argument passed into the shell script. 2. Call the Python script using the three variable values as the command line argument

If you get stuck, you can click on the **Show Solution** button below for a walkthrough with the solution code.

```
In [ ]: %%writefile inference_model_job.sh
        #!/bin/bash

        exec 1>/output/stdout.log 2>/output/stderr.log

        mkdir -p /output

        DEVICE=$1
```

```
MODELPATH=$2
SAVEPATH=$3

# Run the load model python script
python3 inference_on_device.py  --model_path ${MODELPATH} --device ${DEVICE} --path ${SA

cd /output

tar zcvf output.tgz *
```

Show Solution

2.2 Step 3: Submitting a Job to Intel's DevCloud

In the next three sub-steps, you will write your `!qsub` commands to submit your jobs to Intel's DevCloud to load your model and run inference on the **IEI Tank-870** edge node with an **Intel Core i5** CPU and an **Intel Neural Compute Stick 2** VPU. You will use the **MULTI** device plugin to run inference on three device combinations.

Your `!qsub` command should take the following flags and arguments: 1. The first argument should be the shell script filename 2. `-d` flag - This argument should be `.` 3. `-l` flag - This argument should request an edge node with an **IEI Tank-870**. The default quantity is 1, so the **1** after `nodes` is optional. * **Intel Core i5 6500TE** for your `CPU`. * **Intel HD Graphics 530** for your `IGPU`. * **Intel Neural Compute Stick 2** for your `VPU`.

To get the queue labels for these devices, you can go to this link

4. `-F` flag - This argument should contain the three values to assign to the variables of the shell script:

   - **DEVICE** - Device type for the job: You will have to use `MULTI` with three different combinations of `CPU`,`GPU` or `MYRIAD`.
     - `CPU,MYRIAD`
     - `GPU,MYRIAD`
     - `CPU,GPU,MYRIAD`
   - **MODELPATH** - Full path to the model for the job. As a reminder, the model is located in `/data/models/intel`.
   - **SAVEPATH** - Name of the file you want to save the performance metrics as. These should be named as the following:
     - `cpu_vpu_stats` for the `CPU/VPU` job
     - `gpu_vpu_stats` for the `GPU/VPU` job
     - `cpu_gpu_vpu_stats` for the `CPU/GPU/VPU` job

Note: There is an optional flag, `-N`, you may see in a few exercises. This is an argument that only works on Intel's DevCloud that allows you to name your job submission. This argument doesn't work in Udacity's workspace integration with Intel's DevCloud.

2.3 Step 3a: Running on the CPU and VPU (NCS2)

In the cell below, write the qsub command that will submit your job to both the CPU and VPU (NCS2).

If you get stuck, you can click on the **Show Solution** button below for a walkthrough with the solution code.

```
In [29]: cpu_vpu_job_id_core = !qsub inference_model_job.sh -d . -l nodes=1:tank-870:i5-6500te:i
         print(cpu_vpu_job_id_core[0])
```

```
lgwcMqF2dWniyPIB8tMZvvmVUeFi1Gkq
```

Show Solution

2.3.1 Check Job Status

To check on the job that was submitted, use `liveQStat` to check the status of the job. The cell is locked until this finishes polling 10 times or you can interrupt the kernel to stop it by pressing the stop button at the top:

Column `S` shows the state of your running jobs.

For example: - If `JOB ID`is in Q state, it is in the queue waiting for available resources. - If `JOB ID` is in R state, it is running.

```
In [30]: import liveQStat
         liveQStat.liveQStat()
```

Get Results

Run the next cell to retrieve your job's results.

```
In [31]: import get_results

         get_results.getResults(cpu_vpu_job_id_core[0], filename="output.tgz", blocking=True)
```

```
getResults() is blocking until results of the job (id:lgwcMqF2dWniyPIB8tMZvvmVUeFi1Gkq) are read
Please wait...Success!
output.tgz was downloaded in the same folder as this notebook.
```

Unpack your output files and view stdout.log

```
In [32]: !tar zxf output.tgz
```

```
In [33]: !cat stdout.log
```

```
Time taken to load model = 3.2989161014556885 seconds
Time Taken to run 1000 Inference is = 3.46346116065979 seconds
cpu_vpu_stats.txt
stderr.log
```

View stderr.log

This can be used for debugging

```
In [34]: !cat stderr.log
```

```
inference_on_device.py:15: DeprecationWarning: Reading network using constructor is deprecated.
  model=IENetwork(model_structure, model_weights)
```

2.4 Step 3b: Running on the GPU and VPU (NCS2)

In the cell below, write the qsub command that will submit your job to both the GPU and VPU (NCS2).

If you get stuck, you can click on the **Show Solution** button below for a walkthrough with the solution code.

```
In [35]: gpu_vpu_job_id_core = !qsub inference_model_job.sh -d . -l nodes=1:tank-870:i5-6500te:i
         print(gpu_vpu_job_id_core[0])
```

```
B2q6cOEYjH4SYiG6CtoL1l9UWJWC2j2J
```

Show Solution

2.4.1 Check Job Status

To check on the job that was submitted, use `liveQStat` to check the status of the job. The cell is locked until this finishes polling 10 times or you can interrupt the kernel to stop it by pressing the stop button at the top:

Column `S` shows the state of your running jobs.

For example: - If `JOB ID`is in Q state, it is in the queue waiting for available resources. - If `JOB ID` is in R state, it is running.

```
In [36]: import liveQStat
         liveQStat.liveQStat()
```

Get Results

Run the next cell to retrieve your job's results.

```
In [37]: import get_results

         get_results.getResults(gpu_vpu_job_id_core[0], filename="output.tgz", blocking=True)
```

```
getResults() is blocking until results of the job (id:B2q6cOEYjH4SYiG6CtoL1l9UWJWC2j2J) are read
Please wait...Success!
output.tgz was downloaded in the same folder as this notebook.
```

Unpack your output files and view stdout.log

```
In [38]: !tar zxf output.tgz
```

```
In [39]: !cat stdout.log
```

```
Time taken to load model = 44.424675941467285 seconds
Time Taken to run 1000 Inference is = 5.888828992843628 seconds
gpu_vpu_stats.txt
stderr.log
```

View stderr.log
This can be used for debugging

```
In [40]: !cat stderr.log
```

```
inference_on_device.py:15: DeprecationWarning: Reading network using constructor is deprecated.
  model=IENetwork(model_structure, model_weights)
```

2.5 Step 3c: Running on the CPU, GPU and VPU (NCS2)

In the cell below, write the qsub command that will submit your job to all three devices, CPU, GPU, and VPU (NCS2).

If you get stuck, you can click on the **Show Solution** button below for a walkthrough with the solution code.

```
In [41]: cpu_gpu_vpu_job_id_core = !qsub inference_model_job.sh -d . -l nodes=1:tank-870:i5-6500
         print(cpu_gpu_vpu_job_id_core[0])
```

```
UldZnlQzwSr0Po6opDeDhdhniZeGFFAn
```

Show Solution

2.5.1 Check Job Status

To check on the job that was submitted, use `liveQStat` to check the status of the job. The cell is locked until this finishes polling 10 times or you can interrupt the kernel to stop it by pressing the stop button at the top:

Column `S` shows the state of your running jobs.

For example: - If `JOB ID`is in Q state, it is in the queue waiting for available resources. - If `JOB ID` is in R state, it is running.

```
In [42]: import liveQStat
         liveQStat.liveQStat()
```

Get Results
Run the next cell to retrieve your job's results.

```
In [43]: import get_results

         get_results.getResults(cpu_gpu_vpu_job_id_core[0], filename="output.tgz", blocking=True
```

```
getResults() is blocking until results of the job (id:UldZnlQzwSr0Po6opDeDhdhniZeGFFAn) are read
Please wait...Success!
output.tgz was downloaded in the same folder as this notebook.
```

Unpack your output files and view stdout.log

```
In [44]: !tar zxf output.tgz
```

```
In [45]: !cat stdout.log
```

```
Time taken to load model = 44.03113865852356 seconds
Time Taken to run 1000 Inference is = 3.4590494632720947 seconds
cpu_gpu_vpu_stats.txt
stderr.log
```

View stderr.log
This can be used for debugging

```
In [46]: !cat stderr.log
```

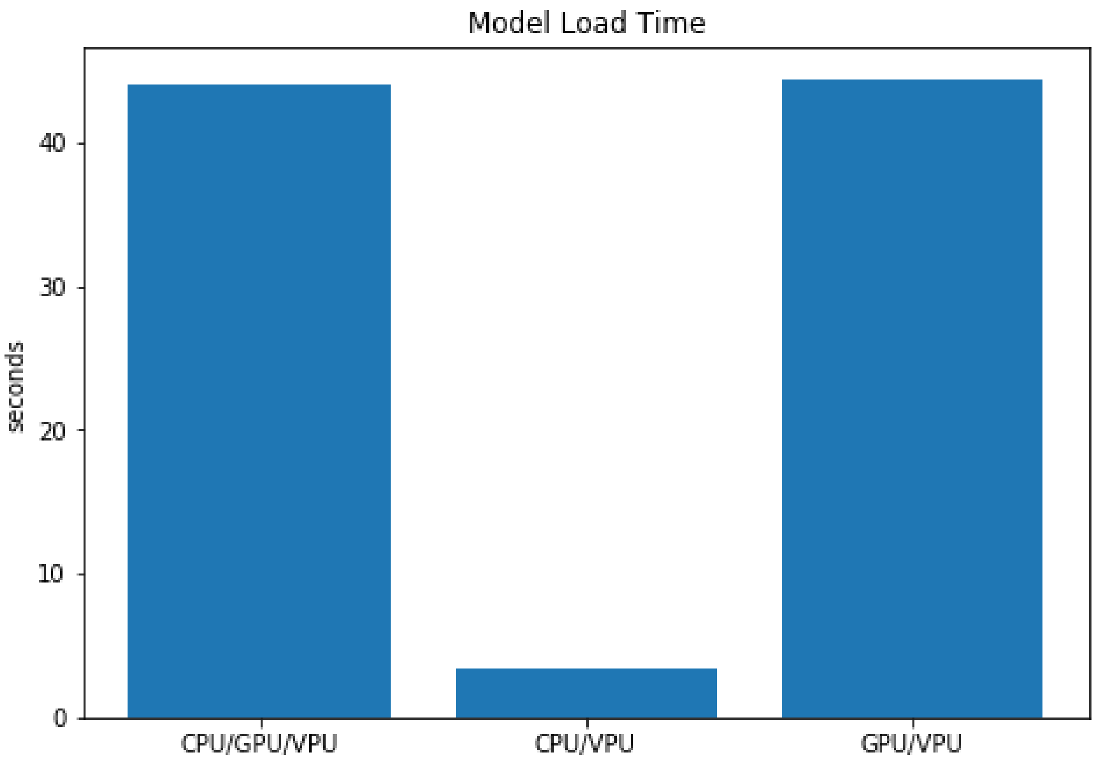
```
inference_on_device.py:15: DeprecationWarning: Reading network using constructor is deprecated.
  model=IENetwork(model_structure, model_weights)
```

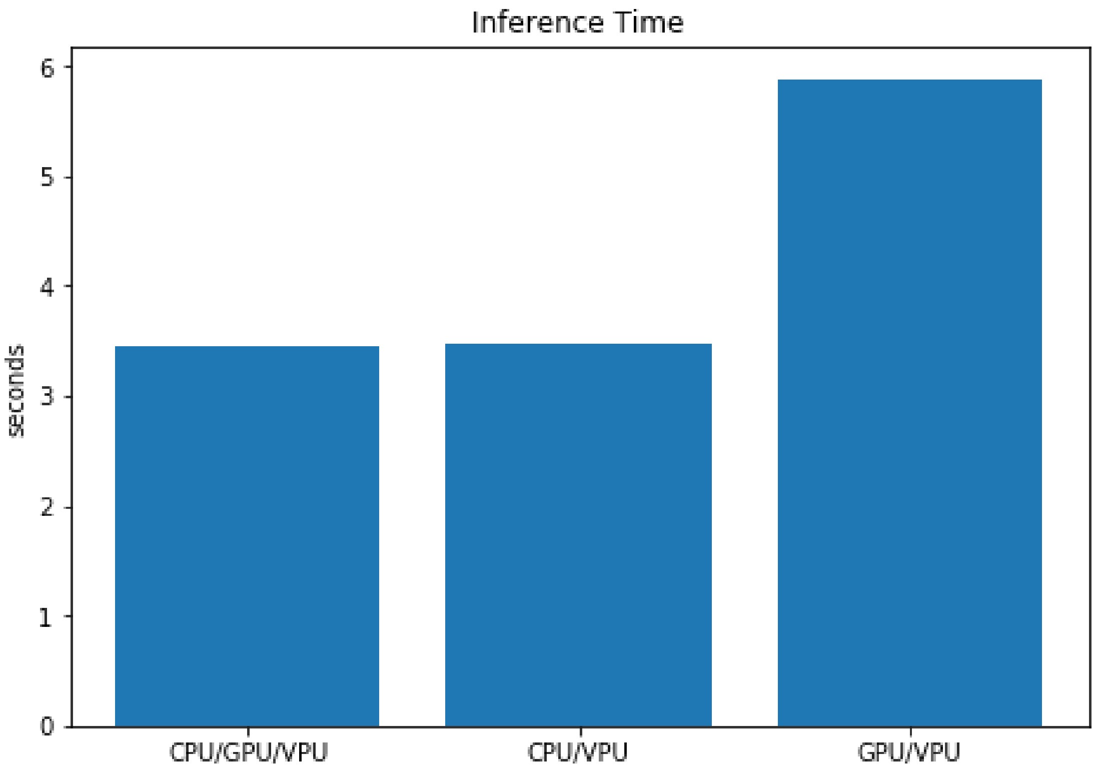
2.6 Step 4: Plot and Compare Results

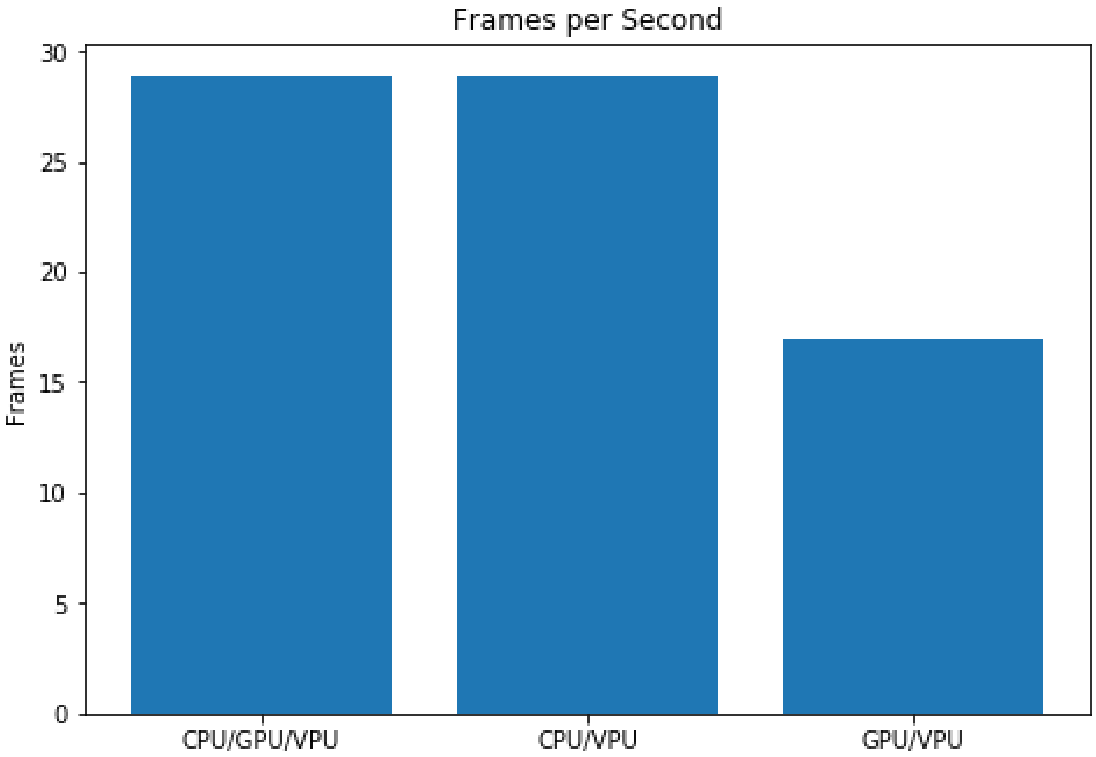
Run the cells below to plot and compare the results.

```
In [47]: import matplotlib.pyplot as plt
```

```
In [48]: def plot(labels, data, title, label):
             fig = plt.figure()
             ax = fig.add_axes([0,0,1,1])
             ax.set_ylabel(label)
             ax.set_title(title)
             ax.bar(labels, data)

         def read_files(paths, labels):
             load_time=[]
             inference_time=[]
             fps=[]

             for path in paths:
                 if os.path.isfile(path):
                     f=open(path, 'r')
                     load_time.append(float(f.readline()))
                     inference_time.append(float(f.readline()))
                     fps.append(float(f.readline()))

             plot(labels, load_time, 'Model Load Time', 'seconds')
             plot(labels, inference_time, 'Inference Time', 'seconds')
             plot(labels, fps, 'Frames per Second', 'Frames')

         paths=['cpu_vpu_stats.txt', 'gpu_vpu_stats.txt', 'cpu_gpu_vpu_stats.txt']
         read_files(paths, ['CPU/VPU', 'GPU/VPU', 'CPU/GPU/VPU'])
```


Model Load Time
seconds
0
10
20
30
40
CPU/GPU/VPU
CPU/VPU
GPU/VPU


Inference Time
seconds
0
1
2
3
4
5
6
CPU/GPU/VPU
CPU/VPU
GPU/VPU


Frames per Second
Frames
0
5
10
15
20
25
30
CPU/GPU/VPU
CPU/VPU
GPU/VPU

```
In [ ]:
```