National Supercomputing Centre

NSCC ASPIRE2A: A Beginner's Guide to Running AI Jobs

Updated: 7-March-2024

Preface

This guide serves as a quick introduction for users to run AI jobs on the NSCC supercomputing facilities and not as a guide to learn AI and its programming.
The guide discusses three main ways of running AI jobs on NSCC facilities, namely, ASPIRE 2A Supercomputer.

Table of Contents

1.Introduction **4**
1.1 Using this Guide 5
1.2 NSCC Supercomputers - ASPIRE 2A AI Systems 4
1.3 Pre-Requisites 6
2. Overview of Artificial Intelligence **6**
2.1 Artificial Intelligence 6
2.2 Machine Learning 7
2.3 Deep Learning and Neural Networks 7
2.4 Artificial Intelligence Libraries 8
3. Different Ways to Run Artificial Intelligence Jobs at NSCC **9**
3.1 Running AI Jobs on the ASPIRE 2A CPU 9
3.2 Running AI Jobs on the ASPIRE 2A GPU 9
3.3 Running AI Jobs on the NSCC AI Cluster 10
4. Exercise 1: First Deep Learning Program on ASPIRE 2A CPU **10**
4.1 Writing your First Deep Learning Program - linear.py 11
Step 1: Import the Relevant Libraries 11
Step 2: Build the Model 12
Step 3: Set the Loss and Optimizer Function 12
Step 4: Initialize Input Data 13
Step 5: Fit the Model 13
Step 6: Prediction 13
4.2 Running your First Deep Learning Program on the ASPIRE 2A CPU 13
4.3 Understanding the output 15
4.4 Takeaways 16
5. Exercise 2: Deep Learning Program on ASPIRE 2A GPU **16**
5.1 Writing your Deep Learning Program - fashion.py 17
Step 1: Import Libraries and Data 18
Step 2: Partition Input Data 18
Step 3: Build the Model 18
Step 4: Compile and Fit the Model 19
Step 5: Test the Model 19
5.2 Running your Deep Learning Program on the ASPIRE 2A GPU 19
5.3 Understanding the Output 20
5.4 Takeaways 21
6. Exercise 3: Convolutional Neural Network Program on NSCC AI System **22**
6.1 Understanding Convolutional Neural Networks 22
6.2 Writing your Convolutional Neural Network Program - convolution.py 23
Step 1: Import Library 24
Step 2: Partition Input Data 25

Step 3: Reshape Input Data 25
Step 4: Build the Model 25
Step 5: Compiling and Fitting the Model 26
Step 6: Testing the Model 27
6.3 Running your Convolutional Neural Network Program on the NSCC AI System 27
6.4 Understanding the Output 28
6.5 Takeaways 29
7. Summary **30**
8. References **30**

1.Introduction

This guide is for new users to begin running Artificial Intelligence (AI) programs on the supercomputers at NSCC (National Supercomputing Centre) [1]. NSCC has two main clusters which are for HPC (High-Performance Computing) jobs on ASPIRE 2A, the pbs101 (normal queue) and pbs102 (ai queue, also called the AI cluster).

1.1 Using this Guide

➔ **This guide is organized into 6 sections:**

- Section 1 (this section) introduces the sections within the document.
- Section 2 provides a quick overview of Artificial Intelligence and the relevant software libraries.
- Section 3 presents the different ways that an AI job can be run on the NSCC resources.
- Section 4 guides you through your very first AI program on the ASPIRE 2A compute nodes.
- Section 5 shows an example of running an AI program on the ASPIRE 2A GPU accelerator.
- Section 6 provides an example of how you can utilize the HPE Apollo 6500 Gen10 plus machine to run AI programs.
- Finally, in Section 7, we have provided some resources for you to further your journey with AI.

1.2 NSCC Supercomputers - ASPIRE 2A AI Cluster

ASPIRE 2A is the new generation HPC system which has a total compute capacity of 3.145 petaflop. (Table 1, the highlighted rows).

➔ **For the AI Cluster:-**

- It contains 18 GPU nodes, totaling 96 A100 GPUs.
- There are 12 GPU nodes which have 4 GPUs, and 6 GPU nodes which are equipped with 8 GPUs.
- All the nodes and storage are connected with the Slingshot high speed network. The Operating System (OS) of each node is Red Hat Enterprise 8.4[3].

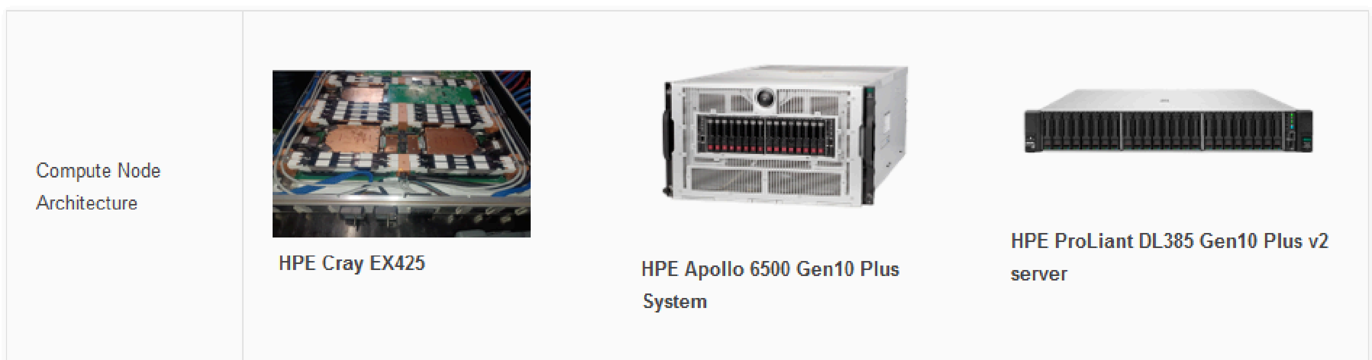

Figure 1.1: Compute Node Architecture of ASPIRE 2A [5]

Table 1. ASPIRE 2A Compute Node Specifications:-

Server	CPU Model	Cores per socket	Socket per server	Total Physical cores per server	Availabl e RAM (DDR4)	GPUs
Standard Compute Node (768 nodes)	Dual-CPU AMD EPYC 7713	64	2	128	512 GB	No GPU
GPU compute node (64 nodes)	Single-CPU AMD EPYC 7713	64	1	64	512 GB	**4x Nvidia A100 40GB**
GPU AI Node (12 nodes)	Single-CPU AMD EPYC 7713	64	1	64	512 GB	**4xNvidia A100 40GB** (11TB nvme)
GPU AI Node (6 nodes)	Dual-CPU AMD EPYC 7713	64	2	128	1 TB	**8xNvidia A100 40GB** (14TB nvme)
Large memory node (12 nodes)	Dual-CPU AMD EPYC 7713	64	2	128	2 TB	No GPU
Large memory node (4 nodes)	Dual-CPU AMD EPYC 7713	64	2	128	4 TB	No GPU
High frequency node (16 nodes)	Dual-CPU AMD EPYC 75F3	32	2	64	512 GB	No GPU

1.3 Pre-Requisites

In this guide, we do not require users to have any background knowledge in AI. However, a basic understanding of Python and bash will be helpful. We are using the open-source programming languages Python and Linux bash shell.

2.1 Artificial Intelligence

Artificial Intelligence refers to the simulation of human intelligence in machines that are programmed to think like humans and mimic their actions. The term may also be applied to any machine that exhibits traits associated with a human mind such as learning and problem-solving.

2.4 Artificial Intelligence Libraries

In this guide, we shall be using two main AI libraries, namely TensorFlow [12] and Keras [13]. For the relatively simple programs in this guide, it is not important to delve into the details of either library. However, for more advanced AI programs, a good understanding of these libraries will be beneficial.

3. Different Ways to Run AI Jobs

In this section, we shall be covering the main ways through which you can run AI jobs on the NSCC resources, which are through the ASPIRE 2A CPU, ASPIRE 2A GPU and AI Cluster. Each approach has similar software and NSCC account requirements, as discussed below. We recommend approaches based on possible use cases. If the jobs plan to run in AI Cluster, the project should have the AI SU.

3.1 Running AI Jobs on the ASPIRE 2A CPU

Some types of AI that do not require GPU power such as below & These types of AI jobs are much fit the CPU nodes of Aspire 2A.

- ➔ Rule-based AI, nearest neighbor
- ➔ Decision trees, Probabilistic AI,
- ➔ Symbolic AI, and etc.

On Aspire 2A, users can create the AI research by different ways such as creating the environment from source codes or pulling a container from Docker Hub. Singularity is supported on Aspire 2a to replace Docker for the safety reason. There are some pre-build containers about AI which are located at `app/apps/containers.` An example of how to use Singularity is available in **Section 4**.

3.2 Running AI Jobs on the ASPIRE 2A GPU

For deep-learning, computer vision and natural language processing, GPU can give huge benefits for training the models. GPUs can perform many computations simultaneously. It is important for AI training because it involves a lot of matrix multiplication and other mathematical operations.

On Aspire 2A, beside the singularity images, there are also some pre-built AI applications, such as Pytorch, Tensorflow which are managed by module files. With the command **`module avail`** `(av` for short`)`, you can see a list of pre-installed software packages and libraries. **Figure 3.1** is a screenshot of the AI software available on ASPIRE 2A.

```
$module av tensorflow
---------------------- /app/apps/modulefiles ---------------------------
tensorflow/1.15.5-hpe     tensorflow/2.7.0-hpe      tensorflow/2.8.1-py3
tensorflow/1.15.5-hpe-gpu tensorflow/2.7.0-hpe-gpu  tensorflow/2.8.1-py3-gpu

$module av pytorch
---------------------- /app/apps/modulefiles ---------------------------
pytorch/1.11.0-hpe     pytorch/1.11.0-hpe-gpu pytorch/1.11.0-py3
pytorch/1.11.0-py3-gpu
```

Figure 3.1: AI software available on NSCC

Kindly user below command to load a software package or library into your environment, **`module load <software package name>`**.

For example: **`module load tensorflow/2.8.1-py3`** loads version 2.8.1 of TensorFlow into your environment. The suffix "-gpu" means the module supports GPU. In section 5, you will see an example of loading the tensorflow environment during the training.

3.3 Running AI Jobs on the NSCC AI Cluster

The NSCC AI Cluster is made up of HPE Apollo 6500 Gen10 Plus servers which are a powerful resource dedicated to AI jobs. Besides the powerful GPUs and CPUs, there is local NVMe storage, which can give huge I/O IOPs during the training processes when reading the data sets. See Table [1].

The local NVME has some of the benefits for AI training:

- Faster data transfer speeds: NVMe can transfer data up to 10 times faster than SATA SSDs and up to 50 times faster than traditional hard drives.
- Lower latency: NVMe has lower latency than other storage interfaces, which means that there is less delay between when a request is made and when the data is returned.
- Higher throughput: NVMe can handle more data per second than other storage interfaces, which can improve the performance of applications that need to process large amounts of data.
- Better scalability: NVMe is scalable, so it can be easily expanded to meet the needs of growing AI workloads.

Then if your AI training needs to deal with lots of data sets, please try to use the local NVMe to achieve better perfoamnce.

4. Exercise 1: First Deep Learning Program on ASPIRE 2A CPU

In this exercise, you will learn the basics of writing and running your first deep learning program on the ASPIRE 2A compute nodes. We will illustrate this using the open-source TensorFlow platform and Keras library. We will also go through the output of the program.

The program is written in Python and the submission script is written in bash. Please use a plain text editor to edit the files. The source codes of examples can be cloned to local. The clone command is:

```
git clone /app/workshops/introductory/ai.examples
```

The program, linear.py [16], uses a neural network algorithm to predict the values of a straight-line graph on a 2-dimensional axis. This program has six main steps.

1. Import the relevant libraries.
2. Build the model,
3. Set the necessary functions and parameters,
4. Initialize our input.
5. Fit and train our model,
6. Get the prediction.

4.1 Writing your First Deep Learning Program - linear.py

Below, we have provided the entire linear.py program. Following that, we have a step by step description and explanation of the code.

```
# Exercise 1 - linear.py
import numpy as np
import keras

# Build the model
model =
keras.Sequential([keras.layers.Dense(units=1,input_shape=[1])])

# Set the loss and optimizer function
model.compile(optimizer='sgd', loss='mean_squared_error')

# Initialize input data
xs = np.array([-1.0, 0.0, 1.0, 2.0, 3.0, 4.0], dtype=float)
```

```
ys = np.array([-2.0, 1.0, 4.0, 7.0, 10.0, 13.0], dtype=float)

# Fit the model
model.fit(xs, ys, epochs=500)
# Prediction
dataIn = np.array([10.0], dtype=float)

print(model.predict(dataIn,1,1))
```

Step 1: Import the Relevant Libraries

```
# Exercise 1 - linear.py
import numpy as np
import keras
```

Open your favorite plain text editor and type in the code as in Step 1 above.
Here, we have imported the critical libraries for the program. NumPy is a mathematical library to handle linear algebra and array processing in deep learning.

Step 2: Build the Model

```
# Build the model
model =
keras.Sequential([keras.layers.Dense(units=1,input_shape=[1])])
```

Next, create the model by adding the above lines of code.

Our model is a *Sequential Model,* which consists of a simple list of layers of nodes. In this model, each layer has a single input and single output. Here, we only have one layer of the node, which is the `layers.Dense` layer. The word `Dense` refers to a regular deeply connected neural network layer, which is the most frequently used layer.

Finally, we have `units=1`, which refers to the number of nodes (i.e. 1 node) and `input_shape = [1]`, which refers to the form of the input data. In this case, our input data is an array of a single value (the x value). Our neural network then outputs the predicted y value. (Note that this input shape refers to a matrix. So, [1,2] would be a matrix with 1 row and 2 columns). With this, we have created our neural network.

Step 3: Set the Loss and Optimizer Function

```
# Set the loss and optimizer function
```

```
model.compile(optimizer='sgd', loss='mean_squared_error')
```

Compile the model as in Step 3 above by adding the appropriate mathematical functions.

In this line, we are setting some basic functions which are the crux of the training process of our neural network. Prior to starting the training process, the neural network will assign random weights. Following that, during training, the model takes the input x values and using the random weights, outputs a predicted y value. Then, the model compares this predicted y value against the actual y value given in the training set. This comparison will be done with the "loss" function, which in this case is the `loss='mean_squared_error'` function.

Next, the weights are readjusted using an optimizer function, i.e. `optimizer='sgd'`. The function will try to adjust the weights such that the next prediction is closer to the expected value of y. This is a high-level and simplified view of Keras. To learn more about the functions, please refer to the official Keras website [18].

Step 4: Initialize Input Data

```
# Initialize input data
xs = np.array([-1.0, 0.0, 1.0, 2.0, 3.0, 4.0], dtype=float)

ys = np.array([-2.0, 1.0, 4.0, 7.0, 10.0, 13.0], dtype=float)
```

Here, define the training data set, with arrays for the x and y values.
The relationship that we are looking for is $y = 3x + 1$

Step 5: Fit the Model

```
# Fit the model
model.fit(xs, ys, epochs=500)
```

Now, train the model by adding the lines of code above.
We are telling the model to begin training with the given training set. In this case, we are telling the computer to train the neural network 500 times. With more epochs (rounds of training), we get higher accuracy. One epoch is when an entire dataset is passed forward and backwards through the neural network once.

Step 6: Prediction

```
# Prediction
```

```
dataIn = np.array([10.0], dtype=float)
print(model.predict(dataIn,1,1))
```

We can then check the accuracy of our model by asking it to predict the value of y when x = 10.0. Intuitively, if the neural network has been trained well, the answer should be 31.0

4.2 Running your First Deep Learning Program on the ASPIRE 2A CPU

1. Save your program as `linear.py`
2. In the same directory, you can create another file, `submit.sh`. This is our submission script:

```
#!/bin/bash
# Exercise 1 submission script - submit.sh
# Below, is the queue
#PBS -q normal
#PBS -j oe
#PBS -l select=1:ncpus=1:mem=1G
#PBS -l walltime=00:10:00
#PBS -P <projectId>
#PBS -N linear_program

# Commands start here
cd ${PBS_O_WORKDIR}
module load tensorflow/2.8.1-py3
python linear.py
```

3. The important parts to focus on for this script are the target queue `#PBS -q normal` and the actual commands.
4. We are using the `normal` queue, which comprises CPUs and GPUs. In this example, we are not making use of GPUs because we are running a simple deep learning job with only one layer, which will run quickly on CPUs. Then in the selection, the variable "ncpus" is set up.
5. As for the commands proper in the script, first, we enter the current working directory `${PBS_O_WORKDIR}`. To find out more about writing and automatically generating PBS scripts, you can refer to the PBS script generator website [19].
6. Then, we load the relevant libraries by using the module interface. In this case, we are loading Singularity.
7. Finally, we execute our program `linear.py`. We do this by executing one of the

provided Singularity images within the NSCC system. In this case, our image contains TensorFlow 1.7.0 and Keras 2.2.0.

8. Submit your job with the command below:

```
qsub submit.sh
```

9. Upon submitting the program, you should see the job ID of your job submission, as can be seen in Figure 4.1 (The blanked-out area is your NSCC user ID). Note that the job ID will be different for you.

```
[linear]$ qsub submit.sh
2129113.pbs101
[linear]$
```

Figure 4.1 The output after submit a job

4.3 Understanding the output

Once the model has finished training, you should see a new file appear in your working directory, something like Figure 4.2 below. The output file is named `linear_program.1254710`, with the name `linear_program` corresponding to the name we set in our submission script, and the number 1254710 corresponding to our job ID.

```
[linear]$ ls
linear_program.o2129113  linear.py  submit.sh
[linear]$
```

Figure 4.2 - Output files when job finished

Open the file using vim using this command `vim linear_program.o<JOBID>`. Scroll through the output and note that the "loss value" (Figure 4.3) decreases with each epoch, signifying that the differences between the predicted and actual values are decreasing.

```
Epoch 1/500
1/1 [=======================] - 2s 2s/step - loss: 22.2192
Epoch 2/500
1/1 [=======================] - 0s 1ms/step - loss: 17.4979
Epoch 3/500
1/1 [=======================] - 0s 781us/step - loss: 13.7831
Epoch 4/500
1/1 [=======================] - 0s 953us/step - loss: 10.8601
Epoch 5/500
1/1 [=======================] - 0s 888us/step - loss: 8.5601
```

Figure 4.3 - Loss from each epoch of linear.py

At the end of the output file (Figure 4.4) note how small the loss value is. We can also see the prediction of our input x = 10.0. However, instead of the expected 31.0, it is a float, 30.996. Our input training set is very small, and TensorFlow considers the probability of the observed relationship holding when it calculates the predicted value, hence giving us a value slightly different from what we expected.

```
Epoch 500/500
1/1 [==============================] - 0s 771us/step - loss:
2.7902e-06
1/1 [==============================] - 0s 63ms/step
[[31.004871]]
```

Figure 4.4 - Final prediction from linear.py

4.4 Takeaways

From this exercise, we hope you have learnt the basics of a simple deep learning program. You should have a better understanding of how we use layers of nodes to build neural networks, and the purpose of loss and optimizer functions in training the models. You should also have a clearer idea of the steps involved in writing deep learning programs, namely importing the relevant libraries, building, compiling and fitting the model to the training data and finally testing the model.

Additionally, you should have a clearer understanding of how to use a Singularity image to run AI programs on the ASPIRE 2A compute nodes. You may explore the other images within the NSCC system to expand the capabilities of your AI programs.

5. Exercise 2: Deep Learning Program on ASPIRE 2A GPU

In this exercise, we will introduce a more complex deep learning program, again using the open-source platform TensorFlow and library Keras. This will allow users to better understand the capabilities and variations of deep learning programs.

Additionally, we shall be covering the script required to run AI programs on the GPU (graphical processing unit), which is an accelerated approach as compared to running AI jobs on the compute nodes, allowing you to run larger and more complex AI programs.

The program is written in Python and the submission script is written in bash.

The program, fashion.py [20], trains a neural network to take an image of a piece of clothing and categorize it into 10 different categories. Our training and testing dataset, Fashion MNIST [17], contains a collection of over 70,000 images of clothing. Like Exercise 1, this program has several main steps, with some changes. First, we import the relevant libraries. Following that, we partition our input data and build our model. Then, we compile and fit the model before finally testing it.

5.1 Writing your Deep Learning Program - fashion.py

Below, we have provided the entire fashion.py program. Following that, we have provided a step by step description and explanation of the code.

```
# Exercise 2 - fashion.py
import tensorflow as tf
import keras

fashion_mnist = keras.datasets.fashion_mnist

# Partition input data
(train_images, train_labels), (test_images, test_labels) =
fashion_mnist.load_data()

# Build the model
model = keras.Sequential([
    keras.layers.Flatten(input_shape=(28,28)),
    keras.layers.Dense(128, activation = tf.nn.relu),
```

```
    keras.layers.Dense(10, activation=tf.nn.softmax)
])

# Compile and fit the model
model.compile(optimizer = tf.keras.optimizers.Adam(),
    loss = 'sparse_categorical_crossentropy',
    metrics=['accuracy'])

model.fit(train_images, train_labels, epochs=5)

# Test the model
test_loss, test_acc = model.evaluate(test_images,test_labels)
print('Test loss: {}, Test accuracy: {}'.format(test_loss,
test_acc*100))
```

Step 1: Import Libraries and Data

```
# Exercise 2 - fashion.py
import tensorflow as tf
import keras

fashion_mnist = keras.datasets.fashion_mnist
```

Open your favorite plain text editor and type in the code as in Step 1 above. Here, we have imported the critical libraries and the dataset for the program.

Step 2: Partition Input Data

```
# Partition input data
(train_images, train_labels), (test_images, test_labels) =
fashion_mnist.load_data()
```

Next, let's split our data into 2 portions. The first will be the training images and the relevant labels, which is what we will be using when building our model. The second is the test set, with which we can check the accuracy of our model.

Step 3: Build the Model

```
# Build the model
model = keras.Sequential([
    keras.layers.Flatten(input_shape=(28,28)),
    keras.layers.Dense(128, activation = tf.nn.relu),
    keras.layers.Dense(10, activation=tf.nn.softmax)
```

```
])
```

Now, let's build our model. Given that we have a much more complex task now, our model will also be more complex as compared to the introductory exercise.

Firstly, we will be building a *Sequential Model* again. Prior to working on the data, the `keras.layers.Flatten(input_shape=(28,28))` function takes in the images (which are of 28 x 28 pixels) and converts them into a one-dimensional set. This is for ease of manipulation as there is a rule of thumb that the first layer in the neural network should follow the shape of the input data. Thus, having 28 x 28 nodes is not feasible and by flattening the images, we are able to build a more feasible neural network.

Next, we have 2 layers of nodes, both of which are `keras.layers.Dense`. Both layers also have an activation function [21], which (in simple terms) is a function that tells each layer of nodes what to do.

Step 4: Compile and Fit the Model

```
# Compile and fit the model
model.compile(optimizer = tf.keras.optimizers.Adam(),
    loss = 'sparse_categorical_crossentropy',
    metrics=['accuracy'])
model.fit(train_images, train_labels, epochs=5)
```

Next, we shall be compiling and fitting our model with only 5 epochs. Note the usage of the loss and optimizer functions.

Step 5: Test the Model

```
# Test the model
test_loss, test_acc = model.evaluate(test_images,test_labels)
print('Test loss: {}, Test accuracy: {}'.format(test_loss,
test_acc*100))
```

Once the model has finished training, we may evaluate its accuracy using our test data.

5.2 Running your Deep Learning Program on the ASPIRE 2A GPU

1. Save your AI program as `fashion.py`
2. In the same directory, create another file, `submit.sh`. This is our submission script.

```
#!/bin/bash
# Exercise 2 submission script - submit.sh
# Below, is the queue
#PBS -q normal
#PBS -j oe
#PBS -l select=1:ngpus=1
#PBS -l walltime=00:10:00
#PBS -P 90000001
#PBS -N fashion_program

# Commands start here
module load tensorflow/2.8.1-py3-gpu
cd ${PBS_O_WORKDIR}
python 2_fashion.py
```

3. The important parts to focus on for this script are the target queue `#PBS -q gpu` and the actual commands.
4. We are using the GPU queue as AI jobs tend to run more efficiently on GPUs instead of CPUs. The GPU accelerated approach allows us to complete complex mathematical jobs in parallel. To find out more about writing and automatically generating PBS scripts, please refer to the PBS script generator website [19].
5. As for the commands, we load the relevant libraries by using the module interface. In this case, we are loading TensorFlow 1.4. Next, we enter our current working directory `${PBS_O_WORKDIR}`.
6. Finally, we run the actual program using Python.
7. Once you have done that and logged into your account, submit your job with this command:

   `qsub submit.sh`
8. Upon submitting the program, you should see the job ID of your job submission, as can be seen in Figure 5.1. [The blanked-out area is your NSCC user ID]. Note that the job ID will be different for you.

```
[fashion]$ ls
fashion.py  submit.sh
[fashion]$ qsub submit.sh
```

```
2129125.pbs101
```

Figure 5.1 - Job ID for Fashion Job Submission

5.3 Understanding the Output

Once the model has finished training, you should see a new file appear in your working directory (as per Figure 5.2). The output file is named `fashion_program.o1255226`, with the name `fashion_program` corresponding to the name we set in our submission script, and the number 1255226 corresponding to our job ID.

```
[fashion]$ ls
fashion_program.o2129125  fashion.py  submit.sh
[fashion]$
```

Figure 5.2 - Output file

Open the file using vim using this command `vim fashion_program.<JOBID>` and scroll through it. You should see something like Figure 5.3 below. Notice that the "loss value" decreases with each epoch, signifying that the differences between the predicted and actual values are decreasing.

```
Epoch 1/5
2023-08-06 22:36:36.656194: I
tensorflow/stream_executor/cuda/cuda_blas.cc:1786]
TensorFloat-32 will be used for the matrix multiplication.
This will only be logged once.
1875/1875 [==============================] - 5s 984us/step -
loss: 3.1328 - accuracy: 0.6751
Epoch 2/5
1875/1875 [==============================] - 2s 966us/step -
loss: 0.7319 - accuracy: 0.7128
Epoch 3/5
1875/1875 [==============================] - 2s 961us/step -
loss: 0.6372 - accuracy: 0.7459
```

Figure *5.3* - Loss from each epoch of fashion.py

At the end of the output file (Figure 5.4), we can see the `test_accuracy` value, which is a relatively low 24.7%, due to the small number of epochs.

```
1875/1875 [==============================] - 2s 959us/step -
loss: 0.6110 - accuracy: 0.7623
Epoch 5/5
1875/1875 [==============================] - 2s 964us/step -
loss: 0.5695 - accuracy: 0.7856
313/313 [==============================] - 0s 934us/step - loss:
0.5703 - accuracy: 0.7962
Test loss: 0.570262610912323, Test accuracy: 79.61999773979187
```

Figure 5.4 - Final test accuracy from fashion.py

5.4 Takeaways

From this exercise, we hope you have learnt more about the possible variations that you can include in your deep learning model. We can have many more layers, hence increasing the complexity of the neural network. Similar to Exercise 1, you should also have a clearer idea of the steps involved in writing deep learning programs, namely importing the relevant libraries, building, compiling and fitting the model to the training data and finally testing the model. In this exercise, we have also introduced the concept of partitioning the training and testing data.

Additionally, you should have a clearer understanding of how to use the module interface to run AI programs on the ASPIRE 2A GPU. You can explore the other TensorFlow modules available on ASPIRE 2A.

6. Exercise 3: Convolutional Neural Network Program on NSCC AI Cluster

In this exercise, we will introduce a different type of neural network, the convolutional neural network. Convolutional neural networks are better at classifying images than the neural network we discussed in Exercise 2.

The program, convolution.py [22], trains a convolutional neural network to take an image of a piece of clothing and categorize it into 10 different categories. Our training and testing dataset, Fashion MNIST [17], contains a collection of over 70,000 images of clothing. While this is largely similar to Exercise 2, we will see a more complex model which provides a better result. There is also a slight change in how we process our input data. First, we import the relevant libraries. Following that, we partition and reshape our input data. Then, we build, compile and fit the model before finally testing it.

6.1 Understanding Convolutional Neural Networks

In Exercise 2, we used a simple neural network as a way to classify images of clothes. This, however, would only work for images which are already centered on the main subject matter of the image. As a result, the neural network would only be able to identify images similar to the ones in Figure 6.1. As users, we would want a program which could correctly identify any shoe, regardless of color, size or orientation of the shoe in the image.

Figure 6.1 - Images from Exercise 2 [23]

To build a more powerful neural network which can classify a larger variety of images, we have to introduce *convolutions* into our earlier neural network. Convolutions are filters which pass over an image during training and process it, thus extracting the common features from images of the same classification. As a result, our neural network will be able to pick out common features, instead of being restricted by the type and size of the image.

6.2 Writing your Convolutional Neural Network Program - convolution.py

We have provided the entire convolution.py program below. A step by step description and explanation of the code is provided below the code.

```
# Exercise 3 - convolution.py
import tensorflow as tf
print(tf.__version__)

# Partition input data
mnist = tf.keras.datasets.fashion_mnist
(training_images,training_labels), (test_images, test_labels)
= mnist.load_data()

# Reshape input data
training_images=training_images.reshape(60000, 28, 28, 1)
training_images=training_images / 255.0
test_images = test_images.reshape(10000, 28, 28, 1)
test_images=test_images / 255.0

#Build the model
model = tf.keras.models.Sequential([
    tf.keras.layers.Conv2D(64,(3,3),activation='relu',
    input_shape=(28, 28, 1)),
    tf.keras.layers.MaxPooling2D(2, 2),
    tf.keras.layers.Conv2D(64, (3, 3), activation='relu'),
    tf.keras.layers.MaxPooling2D(2,2),
    tf.keras.layers.Flatten(),
    tf.keras.layers.Dense(128, activation='relu'),
    tf.keras.layers.Dense(10, activation='softmax')
])
# Compile and fit the model
model.compile(optimizer='adam',
    loss='sparse_categorical_crossentropy',
metrics=['accuracy'])
model.fit(training_images, training_labels, epochs=5)

# Test the model
test_loss,test_accuracy=model.evaluate(test_images,test_label
s)
print('Test loss: {}, Test accuracy: {}'.format(test_loss,
test_accuracy*100))
```

Step 1: Import Library

```
# Exercise 3 - convolution.py
import tensorflow as tf
print(tf.__version__)
```

Open your favorite text editor and type in the code as in Step 1 above.

Here, we have imported TensorFlow for the program. The TensorFlow images available on the Aspire 2A (which we will be using) contain a newer version of TensorFlow, version 2.18.. This version contains Keras as a submodule, allowing users to seamlessly use Keras as a high-level API and TensorFlow as the computational backend. As a result, we will be able to refer to Keras as `tf.keras` (below).

Step 2: Partition Input Data

```
# Partition input data
mnist = tf.keras.datasets.fashion_mnist
(training_images,training_labels), (test_images,
test_labels) = mnist.load_data()
```

Next, let's split our input data into 2 portions. The first will be the training images and the relevant labels, which is what we will be using when building our model. The second is the test set with which we can check the accuracy of our model.

Step 3: Reshape Input Data

```
# Reshape input data
training_images=training_images.reshape(60000, 28, 28, 1)
training_images=training_images / 255.0
test_images = test_images.reshape(10000, 28, 28, 1)
test_images=test_images / 255.0
```

Here, we have to reshape the input images to fit the neural network.

The first convolution (filter) expects a single source of input for both the training and testing phase, rather than having 60000 training and 10000 testing images of dimensions 28x28x1

pixels. As a result, we reshape both sets of images into a single 4-dimensional list.

After that, we also normalize the pixel values. Neural networks generally process inputs using small weight values and using inputs with large integer values can disrupt and slow down the training process. Pixels are unsigned integers in the range between 0 and 255 (represented as a byte). We normalize the value to the 0 - 1 range by dividing with 255 to speed up the training process.

Step 4: Build the Model

```
#Build the model
model = tf.keras.models.Sequential([
    tf.keras.layers.Conv2D(64,(3,3),activation='relu',
    input_shape=(28, 28, 1)),
    tf.keras.layers.MaxPooling2D(2, 2),
    tf.keras.layers.Conv2D(64, (3, 3), activation='relu'),
    tf.keras.layers.MaxPooling2D(2,2),
    tf.keras.layers.Flatten(),
    tf.keras.layers.Dense(128, activation='relu'),
    tf.keras.layers.Dense(10, activation='softmax')
])
```

Now, let's build our model. With the convolutions, we have an even more complex model with several layers.

Firstly, we will be building a *sequential* model again. Prior to working on the data, the `tf.keras.layers.Conv2D(64,(3,3),activation='relu',input_shape= (28, 28, 1))` serves as the convolutional layer. The parameters of this function describe the characteristics of the convolutional layer. In this case, we are generating `64` convolutions (filters), each of dimension `(3 x 3)`. Next, we use the `relu` activation function, which ensures that the layer only returns nonnegative results. We also have the dimensions of the input data.

Following the convolutional layer, we have the pooling layer, which is another mechanism to help programmers to reduce the amount of information in images, while still maintaining the important features. The `tf.keras.layers.MaxPooling2D(2,2)` layer reduces the size of the image by a factor of 4. More specifically, it splits the image into groups of 4 pixels and chooses the largest pixel value, hence the name `MaxPooling`. This iteration and removal process reduces the image size. After that, we repeat the convolution and pooling

once more.

After the convolutions and pooling, we use the same neural network used in Exercise 2. We flatten the images into a one-dimensional set and then pass them through the 2 `keras.layers.Dense` layers, giving us our completed model.

Step 5: Compiling and Fitting the Model

```
# Compile and fit the model
model.compile(optimizer='adam',
    loss='sparse_categorical_crossentropy',
metrics=['accuracy'])
model.fit(training_images, training_labels, epochs=5)
```

Next, we shall be compiling and fitting our model with only 5 epochs. Note the usage of the loss and optimizer functions.

Step 6: Testing the Model

```
# Test the model
test_loss,test_accuracy=model.evaluate(test_images,test_label
s)
print('Test loss: {}, Test accuracy: {}'.format(test_loss,
test_accuracy*100))
```

Once the model has finished training, we may evaluate its accuracy using our test data.

6.3 Running your Convolutional Neural Network Program on the NSCC AI Cluster

1. To use the NSCC AI System, you must have a special account with NSCC.
2. Save your AI program as `convolution.py`
3. In the same directory, create another file, `submit.sh`. This is our submission script.

```
#!/bin/sh
# Below, is the queue
#PBS -q ai
```

```
#PBS -j oe
# Number of cores
#PBS -l select=1:ngpus=1
#PBS -l walltime=00:25:00
#PBS -P 90000001
#PBS -N convolution_program

# Start of commands
cd $PBS_O_WORKDIR
module load singularity/3.10.0
image="/app/apps/containers/tensorflow/tensorflow-nvidia-22.0
4-tf2-py3.sif"

singularity run --nv -B /scratch,/app,/data $image python
3_convolution.py
```

4. The important parts to focus on for this script are the target queue `#PBS -q ai`, the number of cores `#PBS -l select=1:ngpus=8`, project ID `#PBS -P <Project_ID>` and the actual commands.

5. We are using the "ai" queue to showcase the efficiency and power of the HPE Apollo 6500 machine. In addition, you have to use your assigned project ID, which you may apply for through the NSCC website [25].

6. We are also using 8 GPUs and 40 CPU cores on each node. If you decide to use fewer GPUs, you will have to reduce the number of CPUs by five times the number of GPUs. For example, with one GPU, you will have five CPUs, or `#PBS -l select=1:ngpus=1`. To find out more about writing and automatically generating PBS scripts, please refer to the PBS script generator website [19]

7. As for the commands, we enter the working directory and load the TensorFlow image. To see the available images on the NSCC system, you can type `nscc-docker images` in the command line. Following that, we run the Docker image and run the `convolution.py` in the container.

8. Submit your job with this command:

```
$qsub submit.sh
```

9. Upon submitting the program, you should see the job ID of your job submission, as can be seen in Figure 6.2. [The blanked-out area is your NSCC user ID]. Note that the job ID will

be different for you.

```
[convolution]$ ls
convolution.py  submit.sh
[convolution]$ qsub submit.sh
2129229.pbs101
```

Figure 6.2 - Job ID for Convolution Job Submission

6.4 Understanding the Output

Once the model has finished training, you should see a new file appear in your working directory, as per Figure 6.3 below. The output file is named `convolution_program.o2129229`, with the name `convolution_program` corresponding to the name we set in our submission script, and the number 1328360 corresponding to our job ID.

```
[convolution]$ ls
convolution_program.o2129229  convolution.py  submit.sh
[convolution]$
```

Figure 6.3 - Output file

Open the file using vim with this command `vim convolution_program.o<JOBID>` and scroll through it. You should see something like Figure 6.4 below. At the end of the output file (Figure 6.4), we can see the test_accuracy value, which is much higher than Exercise 2 90.77%. We can attribute this to the usage of convolutions, which increase the accuracy and speed of our program.

```
Test loss: 0.2551686465740204, Test accuracy: 90.77000021934509
=================================================================
```

Figure 6.4 - Loss from each epoch of fashion.py

6.5 Takeaways

From this exercise, we hope you have learnt more about convolutions and how beneficial they can be in image classification. Similar to Exercise 2, you should also have a clearer idea of the steps involved in writing deep learning programs, namely importing the relevant libraries, building, compiling and fitting the model to the training data and finally testing the model. In this exercise, we have also introduced the concept of reshaping the input data.

Additionally, you should have a clearer understanding of how to run AI programs on the NVIDIA GPU. You may explore the other Docker images available on the NGC (https://catalog.ngc.nvidia.com/?filters=&orderBy=weightPopularDESC&query=) and convert it to Singularity Image.

To convert the Docker image to singularity image.

```
module load singularity
singularity build mypytorch22.09.sif
docker://nvcr.io/nvidia/pytorch:22.09-py3
singularity exec --nv mypytorch22.09.sif python my_python_script.py <args>
```

7. Summary

With the knowledge from the introductory and advanced tutorials, you should be able to get started on exploring more advanced Deep Learning and Machine Learning tutorials. With the increasing complexity and time requirements of AI programs, you may explore using the AI Cluster to run larger and more complex AI programs. If you are new to programming, you may refer to the Python [8] and bash [9] references. You may also continue with the next set of video tutorials following Exercise 1 and 2 [26].

8. References

[1] NSCC, [Online]. Available: https://www.nscc.sg/ [Accessed 10 Dec 2020]

[2] "NSCC - Software/Hardware Information", National Supercomputing Centre Singapore, [Online]. Available: https://help.nscc.sg/softwarehardware-information/ [Accessed 10 Dec 2020].

[3] "Basic Linux Tutorial", National Supercomputing Centre Singapore, [Online].

Available: https://help.nscc.sg/wp-content/uploads/2016/03/BasicLinuxTutorial-v0.1.pdf [Accessed 10 Dec 2020].

[4] NSCC, "Getting Started with NSCC Supercomputing on ASPIRE 1".

[5] NSCC, "Software/Hardware Information", [Online]. Available: https://help.nscc.sg/softwarehardware-information/ [Accessed 10 Dec 2020].

[6] NVIDIA, "DGX-1 Infographic", [Online]. Available: https://www.nvidia.com/content/dam/en-zz/Solutions/Data-Center/dgx-1/dgx-1-print-infographic-738238-nvidia-web.pdf [Accessed 10 Dec 2020].

[7] NSCC, "AI System Quick Start Guide", 22 August 2019. [Online]. Available: https://help.nscc.sg/wp-content/uploads/AI_System_QuickStart.pdf [Accessed 10 Dec 2020].

[8] Learn Python, "Welcome Page", [Online]. Available: https://www.learnpython.org/ [Accessed 10 Dec 2020].

[9] Linux Config, "Bash Scripting Tutorial for Beginners", [Online]. Available: https://linuxconfig.org/bash-scripting-tutorial-for-beginners [Accessed 10 Dec 2020].

[10] Y. Gavrilova, "Artificial Intelligence vs. Machine Learning vs. Deep Learning: Essentials", 8 April 2020. [Online]. Available: https://serokell.io/blog/ai-ml-dl-difference [Accessed 10 Dec 2020].

[11] C. Nicholson, "A Beginner's Guide to Neural Networks and Deep Learning", [Online]. Available: https://pathmind.com/wiki/neural-network [Accessed 10 Dec 2020].

[12] TensorFlow, [Online]. Available: https://www.tensorflow.org/ [Accessed 10 Dec 2020].

[13] Keras, [Online]. Available: https://keras.io/ [Accessed 10 Dec 2020].

[14] Sylabs, [Online]. Available: https://sylabs.io/docs/ [Accessed 10 Dec 2020].

[15] Environment Modules, [Online]. Available: https://modules.readthedocs.io/en/latest/module.html [Accessed 10 Dec 2020].

[16] TensorFlow, "Intro to Machine Learning(ML Zero to Hero - Part 1)", 31 August 2019. [Online]. Available: https://www.youtube.com/watch?v=KNAWp2S3w94 [Accessed 10 Dec 2020].

[17] TensorFlow, "fashion_mnist", [Online]. Available: https://www.tensorflow.org/datasets/catalog/fashion_mnist [Accessed 10 Dec 2020].

[18] Keras, "API Reference", [Online]. Available: https://keras.io/api/ [Accessed 10 Dec 2020].

[19] NSCC, "PBS Script Generator", [Online]. Available: https://tishyakhanna97.github.io/astarWebsite/index.html [Accessed 10 Dec 2020].

[20] TensorFlow, "Basic Computer Vision with ML(ML Zero to Hero - Part 2)", 4 September 2019. [Online]. Available: https://www.youtube.com/watch?v=bemDFpNooA8 [Accessed 10 Dec 2020].

[21] Keras, "Layer activation functions", [Online]. Available:

https://keras.io/api/layers/activations/ [Accessed 10 Dec 2020].
[22] TensorFlow, "Introducing convolutional neural networks (ML Zero to Hero - Part 3)", [Online]. Available: https://www.youtube.com/watch?v=x_VrgWTKkiM [Accessed 10 Dec 2020].
[23] TensorFlow, "What are Convolutions?", [Online]. Available: https://codelabs.developers.google.com/codelabs/tensorflow-lab3-convolutions/#1 [Accessed 10 Dec 2020].
[24] TensorFlow, "Using Convolutions", [Online]. Available: https://codelabs.developers.google.com/codelabs/tensorflow-lab3-convolutions/#2 [Accessed 10 Dec 2020].
[25] NSCC, "Contact us - Help", [Online]. Available: https://help.nscc.sg/contact-us/ [Accessed 10 Dec 2020].
[26] TensorFlow, "Build an image classifier (ML Zero to Hero - Part 4)", [Online]. Available: https://www.youtube.com/watch?v=u2TjZzNuly8&vl=en [Accessed 10 Dec 2020].