



Hewlett Packard
Enterprise

Using HPE AI and Machine Learning

Lab Guide

Rev. 22.21



© Copyright 2022 Hewlett Packard Enterprise Development LP

The information contained herein is subject to change without notice. The only warranties for Hewlett Packard Enterprise products and services are set forth in the express warranty statements accompanying such products and services. Nothing herein should be construed as constituting an additional warranty. Hewlett Packard Enterprise shall not be liable for technical or editorial errors or omissions contained herein.

This is a Hewlett Packard Enterprise copyrighted work that may not be reproduced without the written permission of Hewlett Packard Enterprise. You may not use these materials to deliver training to any person outside of your organization without the written permission of Hewlett Packard Enterprise.

Printed in the United States of America

Using HPE AI and Machine Learning

Rev. 22.21



Contents

Module 1—Labs: Explore Deep Learning (DL) Training

Lab overview	1
Module 1—Lab 1: Explore DL Training with Constant Hyperparameters	1
Task 1: Get started in the remote lab environment	1
Task 2: Access the HPE Machine Learning Development Environment WebUI.....	3
Task 3: Run a Jupyter Notebook	5
Task 4: Explore an experiment run in HPE Machine Learning Development Environment.....	9
Task 5: Check the trained model accuracy using the Jupyter Notebook	15
Task 6: Explore a Convolutional Neural Network (CNN).....	16
Module 1—Lab 2: Explore DL Training with HPO	21
Task 1: Log into the WebUI	21
Task 2: Explore a DL experiment that uses HPO.....	22
Task 3: View the trained model in action	27

Module 2—Activity: Articulate Benefits of HPE Machine Learning Development Environment

Activity overview	29
Task 1: Read the customer scenario	29
Task 2: Articulate benefits.....	29

Module 3—Lab: Explore the HPE Machine Learning Development Environment Architecture

Lab overview	31
Task 1: View the Cluster	31
Task 2: Use the HPE Machine Learning Development Environment CLI	33

Module 4—Labs: Run Experiments on an HPE Machine Learning Development Environment Cluster

Lab overview	37
---------------------------	-----------

Module 4—Lab 1: Port Model Code for HPE Machine Learning Development Environment.....	37
Task 1: Port the code.....	37
Task 2: Validate the code works.....	49
Troubleshooting hints.....	51
Module 4—Lab 2: Run an Experiment with a Single, Non-Distributed Trial	53
Task 1: Edit the experiment config file.....	53
Task 2: Run the experiment.....	54
Task 3: Monitor the training	55
Task 4: View your trained model in action in a Jupyter Notebook.....	59
Module 4—Lab 3: Run an Experiment with a Single, Distributed Trial	61
Task 1: Create the experiment	61
Task 2: View the experiment.....	63
Task 3: View your trained model in action in a Jupyter Notebook.....	64
Module 4—Lab 4: Run an Experiment that Uses Adaptive ASHA.....	67
Task 1: Create the experiment	67
Task 2: View the experiment.....	70
Task 3: View your trained model in action in a Jupyter Notebook.....	75
Task 4: Terminate your JupyterLab environment.....	77
Module 6—Activity: Engage with a Customer	
Activity overview.....	79
Task 1: Read the customer scenario	79
Task 2: Consider discovery questions	79
Task 3: Read more about the customer	80
Task 4: Start to qualify the customer and size the solution	80
Task 5: Articulate the benefits of HPE Machine Learning Development solutions.....	83



Explore Deep Learning (DL) Training

Module 1—Labs

Lab overview

These labs will introduce you to the environment in which you will work for the rest of the lab. You will also explore the machine learning/deep learning (ML/DL) training process by looking at the results of experiments that have already run in your lab environment.

Module 1—Lab 1: Explore DL Training with Constant Hyperparameters

In this lab, you will log into the HPE Machine Learning Development Environment WebUI and view information about an "experiment" and "trial" completed on the cluster. In other words, someone used the cluster to train a deep learning (DL) model, and you will look at metrics collected during the training process.

The lab actually uses the open-source Determined AI version of the software. However, the functionality that you explore is the same.

Task 1: Get started in the remote lab environment

You will complete the labs in this course in a remote environment.

Your instructor will provide you with this information. Record it here:

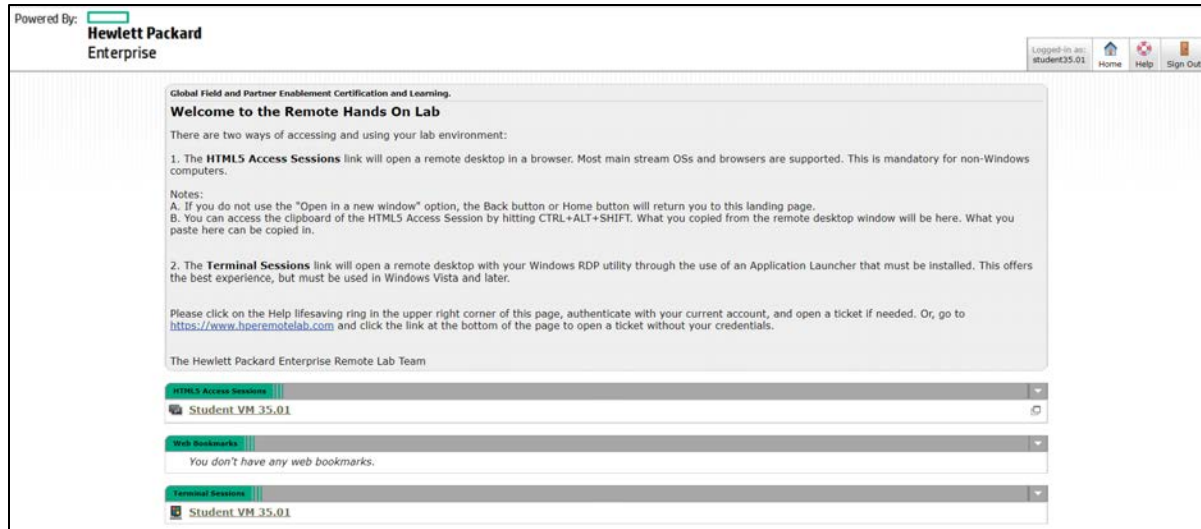
URL: _____

Seat number: _____

Username: _____

Password: _____

1. Access the URL and log in.
2. You will see two choices for accessing the environment. The top choice (HTML5) opens a new browser tab in which you can access the management server. The bottom choice (Terminal) establishes an RDP session to the management server. The HTML5 choice lets you use your current keyboard layout in the environment. The terminal choice lets you copy to and from the remote desktop. However, you need to install a client for the terminal. You can complete the steps of the lab using either method.

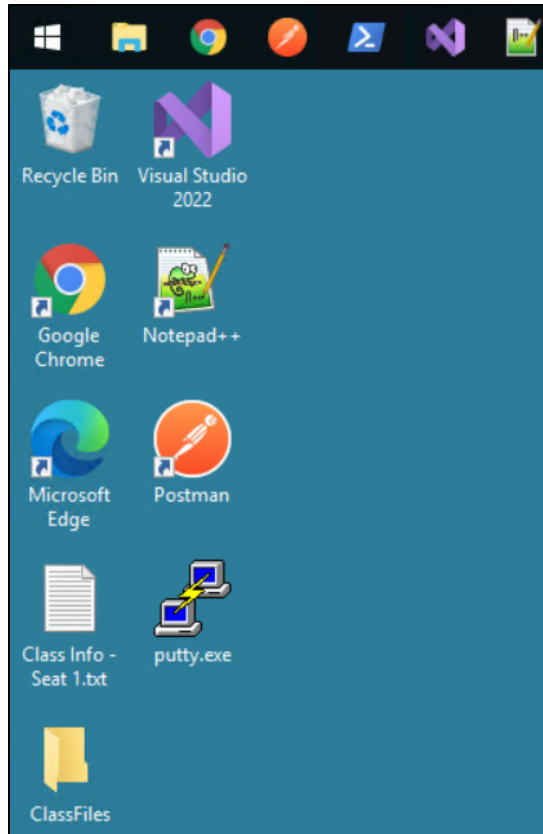


3. After you click one of the options, you will see the management server desktop. The server provides the point of access to the HPE Machine Learning Development Environment cluster.

Warning!

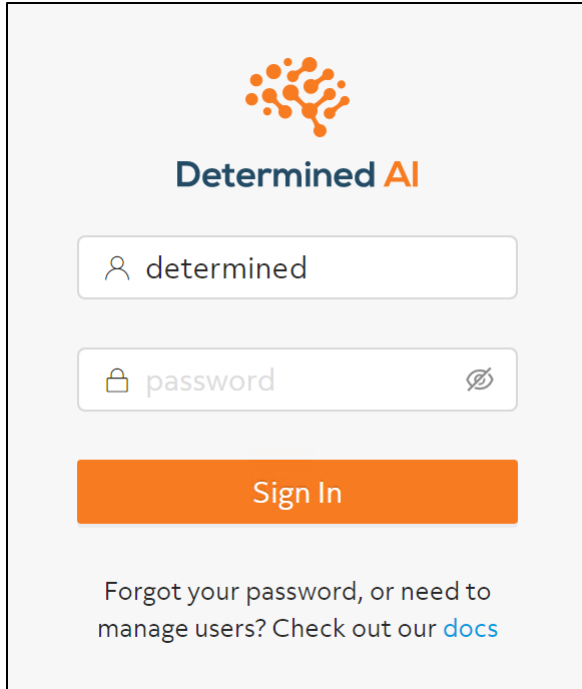
You are sharing the HPE Machine Learning Development Environment cluster with the rest of your classmates. Please take care to follow the instructions in this guide exactly. When you get into the Machine Learning Development Environment UI, you will see several "experiments" that already ran on the cluster. View them as instructed, but do NOT change or delete them.

4. On the desktop find the Class Info file. This file has the usernames and passwords that you will use throughout these labs.
5. On the desktop find the ClassFiles folder. This folder contains files that you will use during the labs such as files for running experiments. Whenever the lab instructions tell you to find a file in the ClassFiles folder, look here.

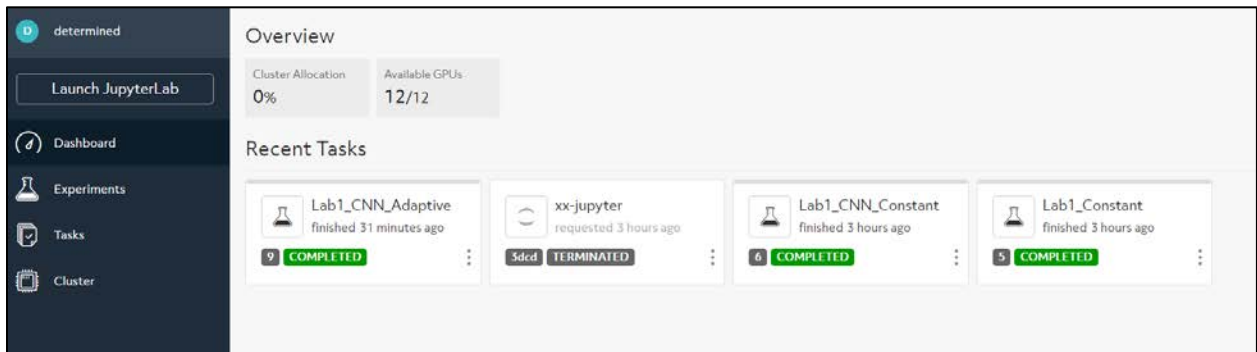


Task 2: Access the HPE Machine Learning Development Environment WebUI

1. The HPE Machine Learning Development Environment software is already installed on a cluster of servers in your lab environment. One of these servers, called the conductor, hosts the WebUI for the cluster. You will now access this WebUI.
2. Open the Chrome browser and navigate to: <http://cluster.hpe.local:8080>
3. Log in with the credentials provided in the Class Info file.



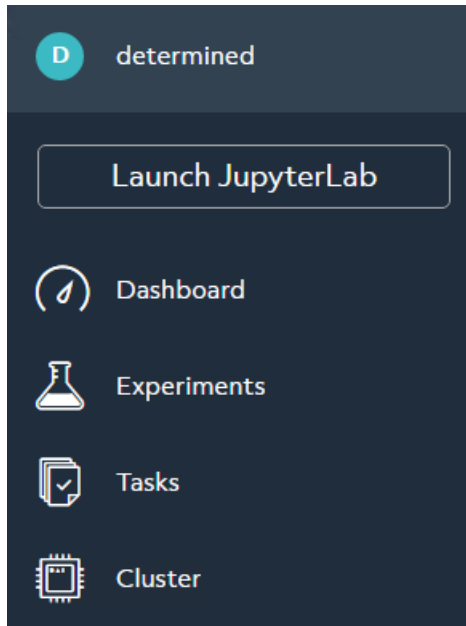
4. You should see the WebUI dashboard.



Task 3: Run a Jupyter Notebook

You will use a Jupyter Notebook to start your DL training exploration.

1. In the left navigation bar, click **Launch JupyterLab**.



2. Fill out the basic settings for your JupyterLab environment:
 - For **Template**, select **jupyter-nb-cpu**.
 - For **Name**, specify **xx-jupyter**, replacing xx with your seat number.
 - For **Resource Pool**, select **lab-compute-pool**.
 - For **Slots**, specify **0**.

Important

Specifying 0 slots is important. This tells JupyterLab to run on CPU, rather than a GPU slot. You need to leave the GPU slots open for use later.

Launch JupyterLab ✕

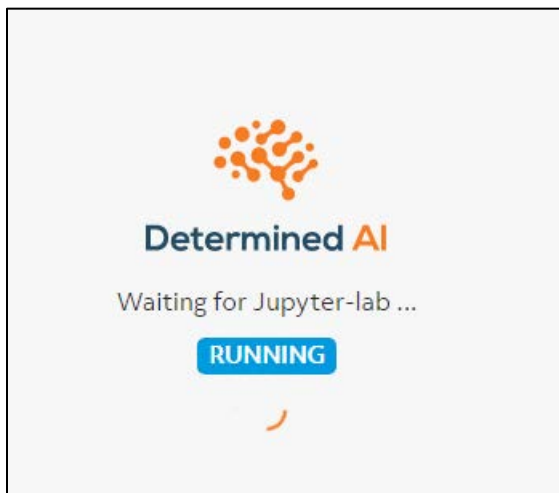
Template

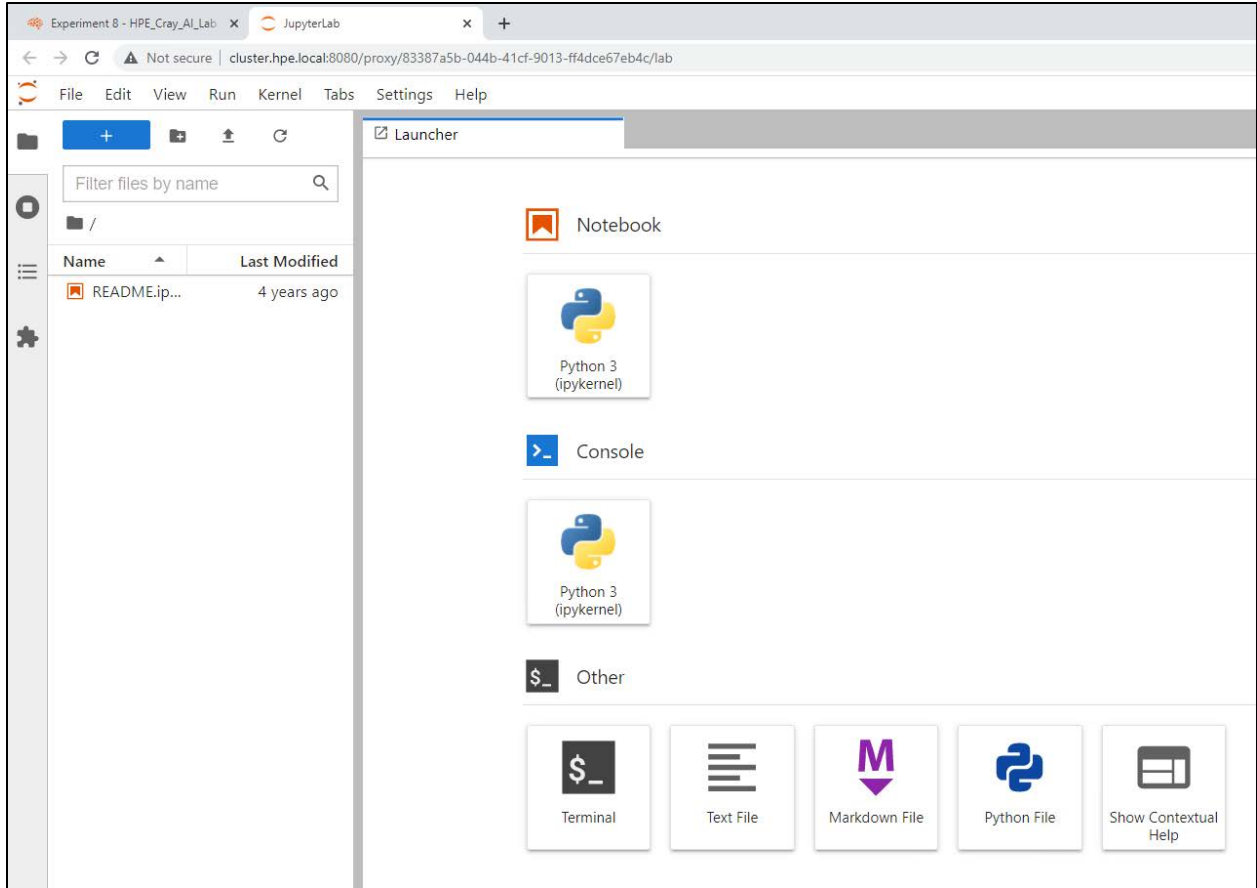
Name

Resource Pool

Slots

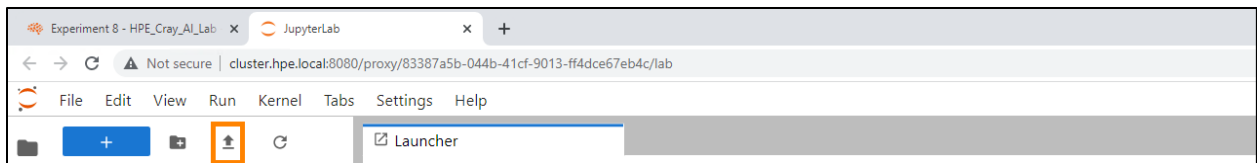
3. Click **Launch**.
4. A new browser tab opens. You will see a "Running" message for a minute or two. You should then see the JupyterLab window.





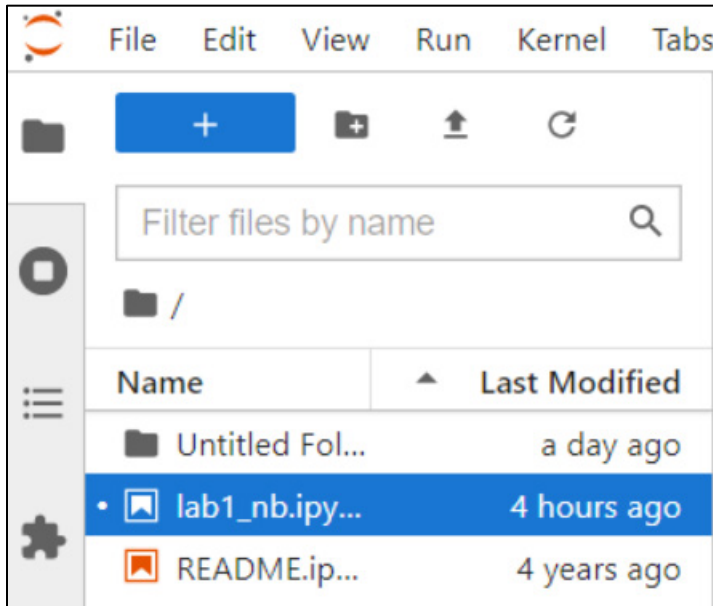
5. Upload the provided Notebook.

- a. In the second to top JupyterLab ribbon, click the up arrow icon to upload a file.



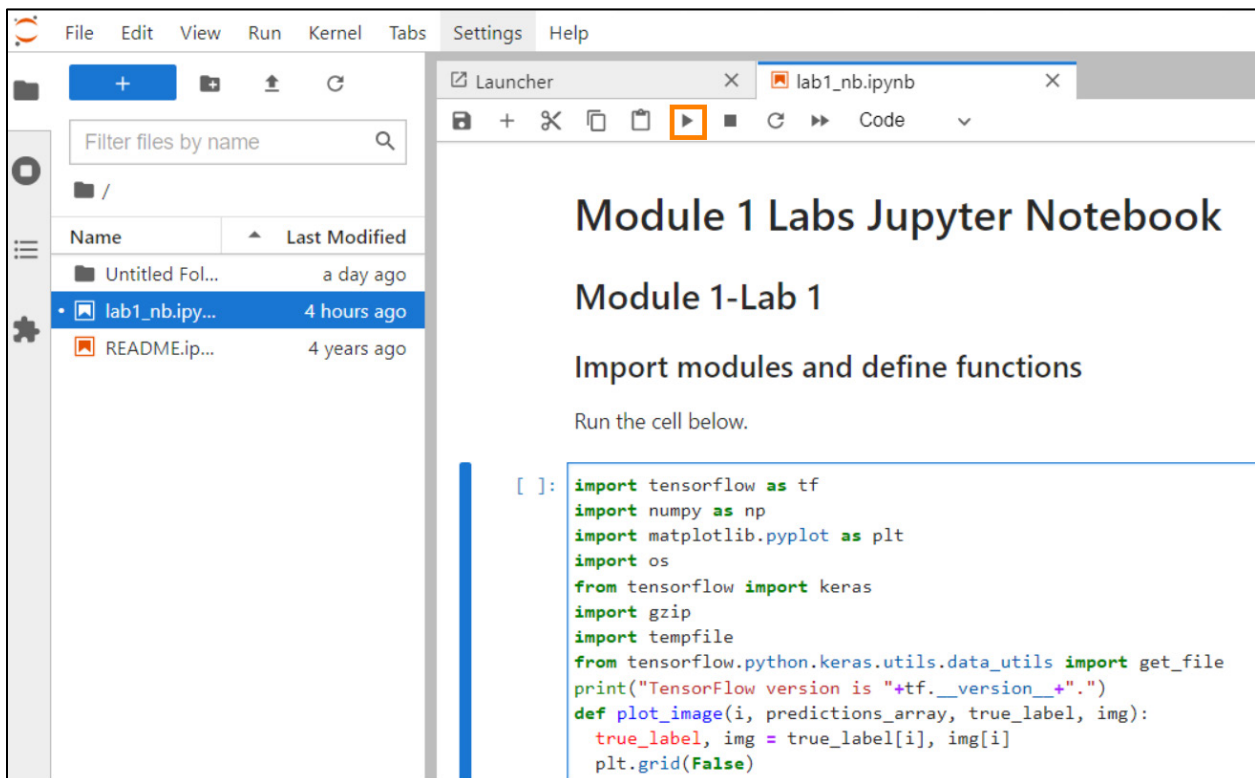
- b. Browse to ClassFiles, select **lab1_nb.ipynb**, and click **Open**.

6. The Notebook will display in the left field. Double-click its name to open the Notebook.



7. You will now run several cells to explore a simple artificial neural network (ANN) model. This model is intended for image classification. You will begin by examining its architecture and see its attempt at image classification when *untrained*.

- a. Begin by placing your cursor in the first code cell, which has an empty square bracket next to it.
- b. Click the Play icon to run the cell.



- c. You will see an asterisk appear in the square bracket, indicating that the cell is running. After the cell finishes running, you will see a number in the square bracket.

```
[1]: import tensorflow as tf
import numpy as np
import matplotlib.pyplot as plt
import os
from tensorflow import keras
```

- d. Above the *next* cell, you will also see output from the code.

```
TensorFlow version is 2.4.3.
Display functions defined.
```

- e. Continue following the instructions in the Notebook to run the next three cells. When you reach the cell that tells you to stop, return to these instructions.

As you proceed through the cells, you will see a few errors related to the fact that you are running TensorFlow on CPU, rather than a GPU. Because you are just running a few images through the model for illustration purposes, and not performing intensive training, the CPU will provide enough power.

You can ignore these messages. The output that you can ignore is shown below.

Under cell 3:

```
2022-01-25 19:09:04.744463: I tensorflow/compiler/jit/xla_cpu_device.cc:41] Not creating XLA devices, tf_xla_enable_xla_devices not set
2022-01-25 19:09:04.744742: I tensorflow/core/platform/cpu_feature_guard.cc:142] This TensorFlow binary is optimized with oneAPI Deep Neural Network Library (oneDNN) to use the following CPU instructions in performance-critical operations: AVX AVX512F FMA
To enable them in other operations, rebuild TensorFlow with the appropriate compiler flags.
```

Under cell 4:

```
2022-01-25 19:09:04.867057: I tensorflow/compiler/mlir/mlir_graph_optimization_pass.cc:116] None of the MLIR optimization passes are enabled (registered 2)
2022-01-25 19:09:04.888177: I tensorflow/core/platform/profile_utils/cpu_utils.cc:112] CPU Frequency: 2100000000 Hz
```

Task 4: Explore an experiment run in HPE Machine Learning Development Environment

- 1. In the Notebook, you observed a defined artificial neural network (ANN) model. What layers did the model have?

- 2. You saw that the untrained model could not successfully classify images yet. However, someone has already run an "experiment" in HPE Machine Learning Development Environment. This experiment is associated with a single "trial." That trial trained the same model that you examined in the Notebook.

You will now view the record of the experiment in the Machine Learning Development Environment WebUI. This record gives insight into the training process and metrics collected during that process.

Note

You do not have to define the untrained model in a Notebook before training the model on Machine Learning Development Environment. You were just using the Notebook to explore what this model does.

3. Return to the browser tab with the Web UI. In the left navigation bar, click **Experiments**.
4. You should see three experiments with a Completed state.
5. Click **Lab1_Constant**.

ID	Name	Labels	Start Time	Duration	Trials	State	Resource Pool	Progress	Archived	User
9	Lab1_CNN_Adaptive		1 hour ago	39m 30s	512	COMPLETED	lab-compute-pool			
6	Lab1_CNN_Constant		3 hours ago	2m 5s	1	COMPLETED	lab-compute-pool			
5	Lab1_Constant		3 hours ago	1m 25s	1	COMPLETED	lab-compute-pool			

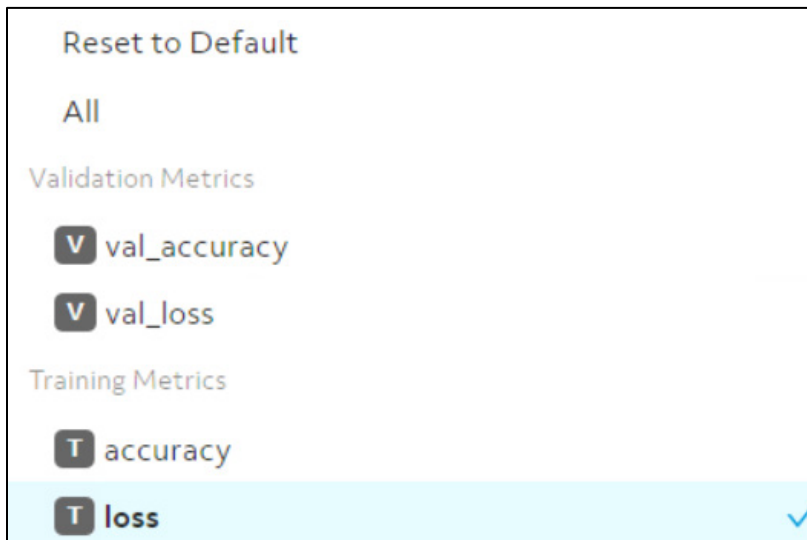
6. This experiment is associated with a single "trial," or training process. You should be at the **Overview** tab.

Here you see an overview of metrics collected during the training process. The graph's x-axis shows the number of batches on which the model was trained. The graph's y-axis shows a metric about the model's performance.



HPE Machine Learning Development Environment collects two broad categories of metrics:

- **Training metrics** report on the model's current performance on training data. The environment collects these metrics frequently. These metrics generally include:
 - **Loss**—The distance between the model's output and the correct output (as defined by data labels). The backward pass (optimization) attempts to reduce loss for the next pass of data. The lower the loss, the better the model is doing.
 - **Accuracy**—The accuracy percentage for the model's output (for example, .8 = 80%)
 - **Validation metrics** report on the model's performance when tested on validation data. The environment typically collects these metrics less frequently. These metrics might also include loss and accuracy.
7. Notice that the graph initially shows "val_accuracy," which is a validation metric. As you see, the accuracy increased as the model was trained on more data.
 8. From the **Metrics** drop-down menu, under **Training Metrics**, select **loss**. Then under **Validation Metrics**, select **val_accuracy** to temporarily turn that metric off.



9. Click away from the menu. You should see the graph update to show the change in loss across the training process.



10. You can hover over the line to see exact values. Record the loss that you see for these batches:

100 _____

200 _____

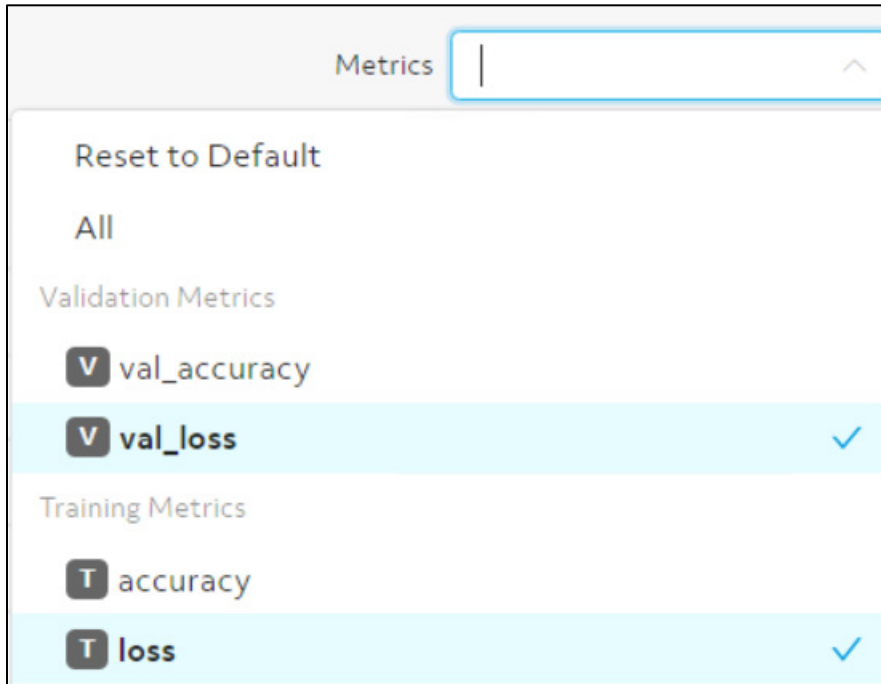
5050 _____

9375 _____

11. As you see, the loss reduces quickly at first and then much more gradually.

This model was also validated on a different data set periodically throughout the training process. You can check the validation results with validation metrics.

12. In the **Metrics** drop-down menu, under **Validation metrics**, select **val_loss** and keep **loss** selected, too.

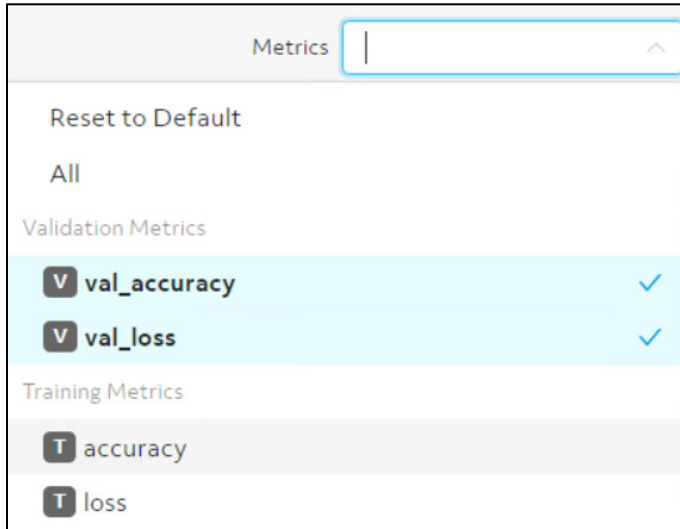


13. Click away and observe the graph.



14. Check the validation_loss and training loss for the same batch number. Based on what you learned in the module, why do you think they are different?

15. In the **Metrics** drop-down menu, under **Validation metrics**, select **accuracy** and **validation_loss**. Select **loss** to turn off that metric.

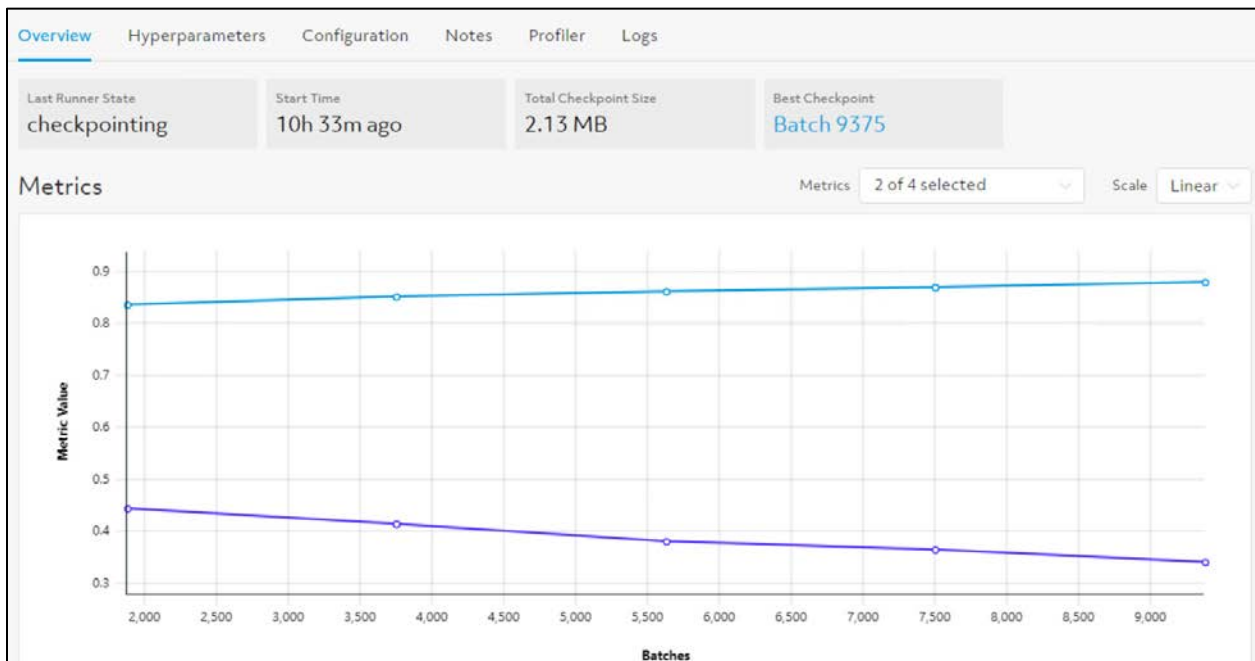


16. Record the best values that you see:

Batch number with best values: _____

Accuracy: _____

Validation loss: _____



Task 5: Check the trained model accuracy using the Jupyter Notebook

As you saw, the final trained model has about 88% accuracy. You will now use the Jupyter Notebook to see the trained model in action.

Important

You must run the Notebook in order. If for some reason you need to reload the Notebook and start over, make sure to go back to the beginning and run the cells in order.

1. Return to the browser tab with the Jupyter Notebook. (If you accidentally closed this tab, you can easily restore it. In the HPE Machine Learning Development Environment Web UI, click **Tasks**. Click **xx-jupyter** in which xx is your seat number.)
2. Locate the section that says "Start here at Module 1-Lab 1 Task 5."

Start here at Module 1-Lab 1 Task 5

Load and use the trained model from "Lab1_Constant"

You explored the "Lab1_Constant" experiment in the Web UI. As you saw, the model accuracy improved over the course of the training until it achieved about 88% accuracy. You will now load the trained model and see it in action.

Run the cell below. As you will see, the model is now successfully classifying most images.

```
[ ]: ann_model.load_weights("/tmp/determined-checkpoint/fa11c0ab-85b8-4b26-8130-1430207312b8/determined-keras-model-weights").expect_p  
print("\033[1m" + "Trained model loaded." + "\033[0m")  
probability_model = tf.keras.Sequential([ann_model,  
                                         tf.keras.layers.Softmax()])  
predictions = probability_model.predict(test_images)
```

- Run the cell below. The trained model loads and processes validation data. You should see that the model can classify most of the images correctly.



Task 6: Explore a Convolutional Neural Network (CNN)

As you learned in the module, a CNN model can work well for tasks such as image classification. You will next explore a CNN model that has been trained on the same dataset that the previous model was.

- You should still be in the Jupyter Notebook. Read the information about the CNN. Answer these questions:

How many convolutional (Conv2D) layers does this model have? _____

How many filters do each of these layers use? _____

2. Run the cell below to instantiate the model.

Examine a Convolutional Neural Network (CNN)

In the code below, examine a CNN model, defined for the same image classification purposes as the previous. This CNN model has:

- 2 hidden Conv2D layers:
 - The first layer both defines the input shape (creating an implicit input layer) and defines a convolutional layer with 64 3x3 filters
 - The second defines a convolutional layer with 128 3x3 filters
- Two layers that reformat the outputs for the next layer (MaxPool2D and Flatten)
- 1 hidden Dense layer, which has 128 nodes
- 1 output (Dense) layer, which has the same number of nodes as there are classes for images

Run the cell.

```
[6]: cnn_model = tf.keras.Sequential(
      [
        tf.keras.layers.Conv2D(64, (3,3), input_shape=(28,28,1), padding="same", activation="relu"),
        tf.keras.layers.Conv2D(128, (3,3), padding="same", activation="relu"),
        tf.keras.layers.MaxPool2D(2,2),
        tf.keras.layers.Flatten(),
        tf.keras.layers.Dense(128, activation="relu"),
        tf.keras.layers.Dense(10),
      ]
    )
    print("Untrained model defined.")
```

Untrained model defined.

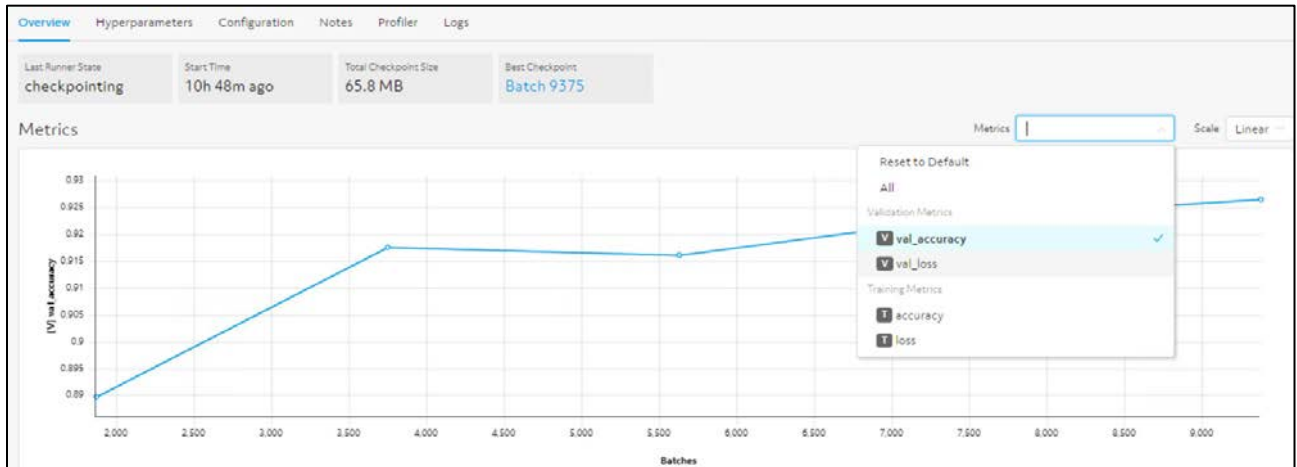
Stop

Return to the Module 1-Lab 1 instructions.

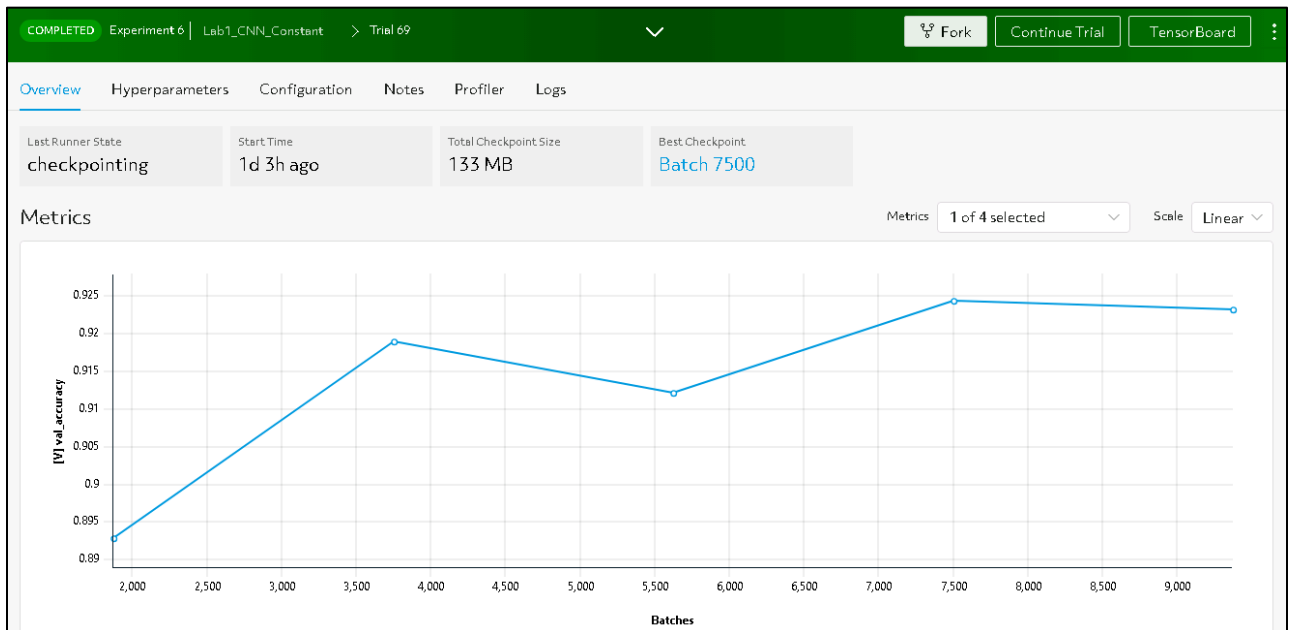
3. Someone has already run an experiment on HPE Machine Learning Development Environment to train this same model. Return to the browser tab to see that experiment.
4. In the left navigation bar, click **Experiments**.
5. Click **Lab1_CNN_Constant**.

ID	Name	Labels	Start Time	Duration	Trials	State	Resource Pool	Progress	Archived	User
9	Lab1_CNN_Adaptive		1 hour ago	39m 30s	512	COMPLETED	lab-compute-pool			
6	Lab1_CNN_Constant		3 hours ago	2m 5s	1	COMPLETED	lab-compute-pool			
5	Lab1_Constant		3 hours ago	1m 25s	1	COMPLETED	lab-compute-pool			

6. In the **Metrics** drop-down menu, select **val_accuracy**.



7. As you see, the **val_accuracy** actually went down slightly at the end of the training process. However, HPE Machine Learning Development Environment preserved a "checkpoint" of the best model.



8. Scroll down to the **Workloads** section. Record the best val_accuracy. As you see, it is higher than that for the previous trained model (from the Lab1_Constant experiment).

Workloads		Show	Has Checkpoint or Validation
Batches	val_accuracy	Checkpoint	
9375	0.923200		
7500	0.924400		
5625	0.912200		
3750	0.919000		
1875	0.892900		

9. Click the **Hyperparameters** tab. A hyperparameter is a parameter that affects the training process itself. For example, in this case, the hyperparameters indicate how many nodes to use in the dense hidden layer and how many filters to use in each convolutional layer. This experiment uses constant hyperparameters.

Overview	Hyperparameters	Configuration	Notes	Profiler	Logs
Hyperparameter	Value				
dense1	128				
filters1	64				
filters2	128				
global_batch_size	32				

- Now you will see this trained model in action. Return to the browser tab with the Jupyter Notebook. (If you accidentally closed this tab, you can easily restore it. In the HPE Machine Learning Development Environment Web UI, click **Tasks**. Click **xx-jupyter** in which xx is your seat number.)

Important

You must run the Notebook in order. If for some reason you need to reload the Notebook and start over, make sure to go back to the beginning and run the cells in order

- Run the next cell to load the trained CNN model and see it in action. You should see that it classifies more images correctly than the previous model.

```
[7]: cnn_model.load_weights("/tmp/determined-checkpoint/a68e4ef1-f4a8-4d05-8592-ed8d40ef32b3/determined-keras-model-weights").expect_partial()
print("\033[1m"+"Trained CNN model loaded."+"\033[0m")
ch_test_images = np.expand_dims(test_images, axis=-1)
print("Channel dimension added to grayscale images to ensure proper input in the CNN.")
probability_model = tf.keras.Sequential([cnn_model,
                                        tf.keras.layers.Softmax()])
predictions = probability_model.predict(ch_test_images)
num_rows = 8
num_cols = 3
num_images = num_rows*num_cols
plt.figure(figsize=(2*num_cols, 2*num_rows))
for i in range(num_images):
    plt.subplot(num_rows, 2*num_cols, 2*i+1)
    plot_image(i, predictions[i], test_labels, ch_test_images)
    plt.subplot(num_rows, 2*num_cols, 2*i+2)
    plot_value_array(i, predictions[i], test_labels)
plt.tight_layout()
print("\033[1m"+"Trained CNN model classifications"+"\033[0m")
plt.show()
```

Trained CNN model loaded.
 Channel dimension added to grayscale images to ensure proper input in the CNN.
 Trained CNN model classifications

Ankle boot 100% (Ankle boot) 0 1 2 3 4 5 6 7 8 9
 Pullover 100% (Pullover) 0 1 2 3 4 5 6 7 8 9
 Trousers 100% (Trousers) 0 1 2 3 4 5 6 7 8 9
 Trousers 100% (Trousers) 0 1 2 3 4 5 6 7 8 9
 Shirt 97% (Shirt) 0 1 2 3 4 5 6 7 8 9
 Trousers 100% (Trousers) 0 1 2 3 4 5 6 7 8 9

- You can continue to run through the Notebook to learn a bit more about the filters that this CNN is using. Follow the instructions in the Notebook. Stop after running three cells when the Notebook tells you to stop.

- But keep the Notebook open.

YOU HAVE COMPLETED THIS LAB.

Module 1—Lab 2: Explore DL Training with HPO

In this lab, you will explore hyperparameter optimization (HPO) with the Adaptive ASHA searcher.

This lab uses a similar CNN model to the previous lab, and this model was trained on the same dataset to classify fashion images. However, the ML engineers wanted to explore some different hyperparameters in efforts to further improve the accuracy of the final trained model. They decided to try:

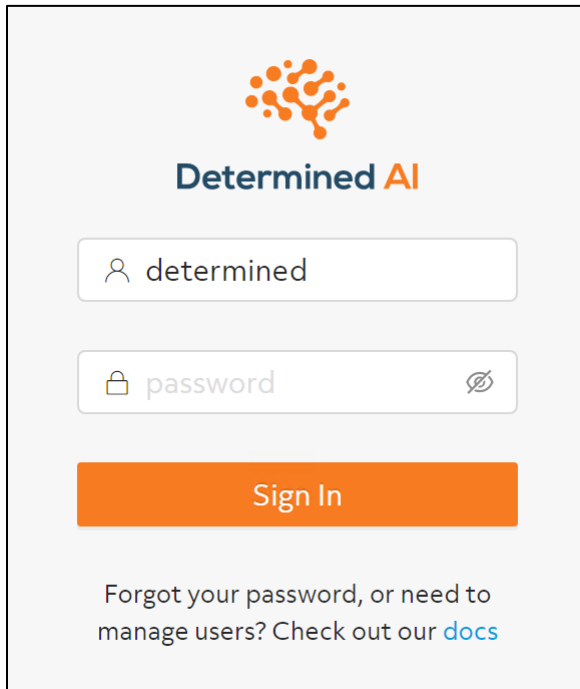
- Try out different number of filters/nodes for the hidden layers
- Try various values for two "dropout" layers, one after the second convolutional layers and one after the hidden dense layer


The dropout layer tells the model to ignore the outputs from randomly selected nodes in the previous layer. The dropout value specifies what percentage to ignore (.1 = 10%). Ignoring different nodes at different times helps the training process improve all nodes and the final model to become more robust.

As you learned in the module, ML engineers can run an experiment that searches for the hyperparameters using HPO. Such an experiment consists of multiple trials, each of which uses different hyperparameters. Your environment features such an experiment that has already been run.

Task 1: Log into the WebUI

1. Log into the HPE remote lab environment.
2. Open the Chrome browser and navigate to: `http://cluster.hpe.local:8080`
3. Log in with the credentials provided in the Class Info file.




Determined AI

Sign In

Forgot your password, or need to manage users? Check out our [docs](#)

Task 2: Explore a DL experiment that uses HPO

In this task, you will explore the Adaptive ASHA experiment that has already been run. You will view metrics collected during the training process and answer some questions.

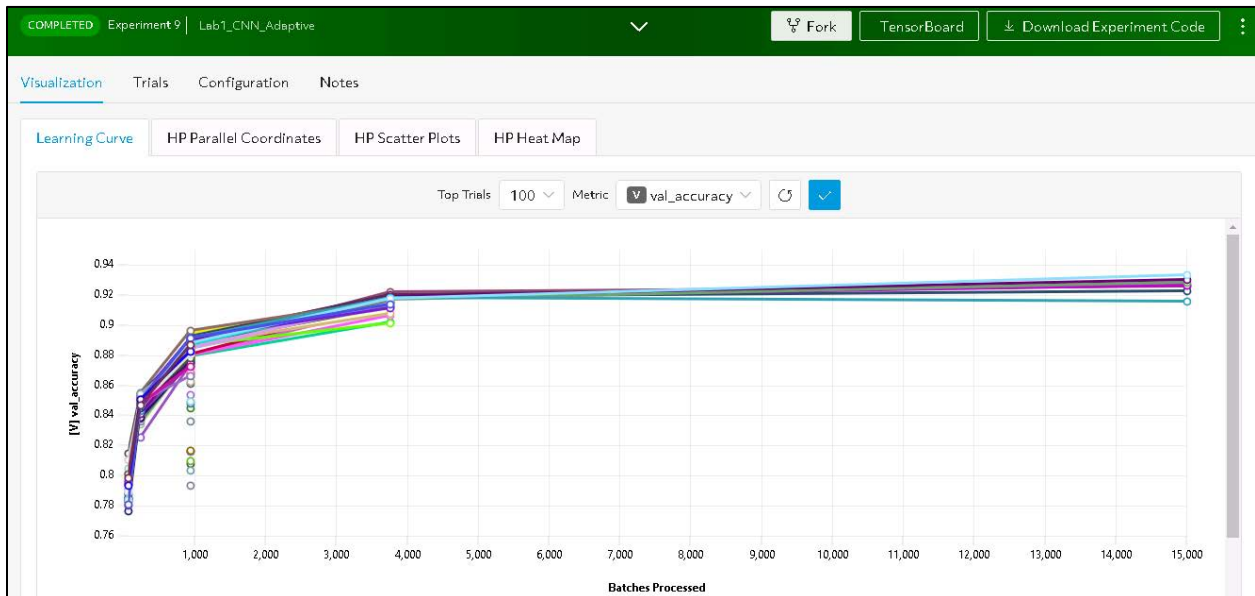
1. In the left navigation bar, click **Experiments**.
2. Click **Lab1_CNN_Adaptive**. This experiment consisted of multiple trials that used Adaptive ASHA for HPO.

ID	Name	Labels	Start Time	Duration	Trials	State	Resource Pool	Progress	Archived	User
9	Lab1_CNN_Adaptive		1 hour ago	39m 30s	512	COMPLETED	lab-compute-pool			
6	Lab1_CNN_Constant		3 hours ago	2m 5s	1	COMPLETED	lab-compute-pool			
5	Lab1_Constant		3 hours ago	1m 25s	1	COMPLETED	lab-compute-pool			

3. You will be at the **Visualization** tab, where you can see more details about the many trials run by this experiment. These details report on the top-performing 100 trials in the experiment.

The visualization graphs the number of batches on which the trial trained the model on the x-axis. It graphs the validation accuracy on the y-axis. Each trial has a dot indicating its validation accuracy when the trial was validated. Trials that were validated multiple times have lines that connect their dots together. Note these features:

- There are fewer lines at the end than at the beginning. This is because ASHA stops most trials at each rung and only promotes a few.
- Some trials are simply dots. This is because the Adaptive ASHA searcher stopped the trials after the first validation (rung 1). They were not performing well.
- You do not see dots at the far left of the graph. That is because this experiment had 512 trials, but the visualization only shows the top 100. There were actually 412 other trials that performed worse than the ones you see here; many of these trials trained the model for only 58 or 234 batches.



- You can find the most accurate trial in the list below. Scroll down and click the **val_accuracy** column to sort by highest accuracy.
- Record these results:

Top trial ID: _____

Top trial accuracy: _____

<input type="checkbox"/>	Trial ID	val_accuracy	dense1	dropout1	dropout2	filters1	filters2
<input type="checkbox"/>	1550	0.933400	497	0.336463	0.286505	39	94
<input type="checkbox"/>	1337	0.930300	188	0.067885	0.253736	47	96
<input type="checkbox"/>	1302	0.928800	383	0.097605	0.107460	50	110
<input type="checkbox"/>	1216	0.926800	307	0.038680	0.249888	52	94

- Is that accuracy higher than that for the Lab1_CNN_Const trial?

- Click the **Trials** tab.

Here you see a list of the trials, the number of batches of data on which they were trained, and the trial's best and latest validation metrics.

- Click **Batches** to sort by that column. Use the numbers at the bottom of the page to click through the just a few of the pages. As you see, most trials were trained briefly on just 58 batches, some on 234, a few more on 3750 batches, and only 8 out of 512 on the full data set (15000 batches). This is a key characteristic of Adaptive ASHA HPO.

<input type="checkbox"/>	ID	State	Batches	Best Validation Metric	Latest Validation Metric	Start Time	Duration	Checkpoint
<input type="checkbox"/>	Trial 1103	COMPLETED	58	0.723900	0.723900	1 day ago	38m 38s	⋮
<input type="checkbox"/>	Trial 1104	COMPLETED	58	0.770200	0.770200	1 day ago	38m 38s	⋮
<input type="checkbox"/>	Trial 1105	COMPLETED	58	0.766000	0.766000	1 day ago	38m 38s	⋮
<input type="checkbox"/>	Trial 1112	COMPLETED	58	0.760300	0.760300	1 day ago	37m 51s	⋮
<input type="checkbox"/>	Trial 1119	COMPLETED	58	0.663500	0.663500	1 day ago	37m 17s	⋮
<input type="checkbox"/>	Trial 1120	COMPLETED	58	0.751300	0.751300	1 day ago	37m 12s	⋮
<input type="checkbox"/>	Trial 1124	COMPLETED	58	0.762700	0.762700	1 day ago	37m 5s	⋮
<input type="checkbox"/>	Trial 1134	COMPLETED	58	0.746200	0.746200	1 day ago	36m 23s	⋮
<input type="checkbox"/>	Trial 1137	COMPLETED	58	0.766500	0.766500	1 day ago	36m 20s	⋮
<input type="checkbox"/>	Trial 1140	COMPLETED	58	0.768700	0.768700	1 day ago	36m 3s	⋮

< 1 2 3 4 5 ... 52 > 10 / page

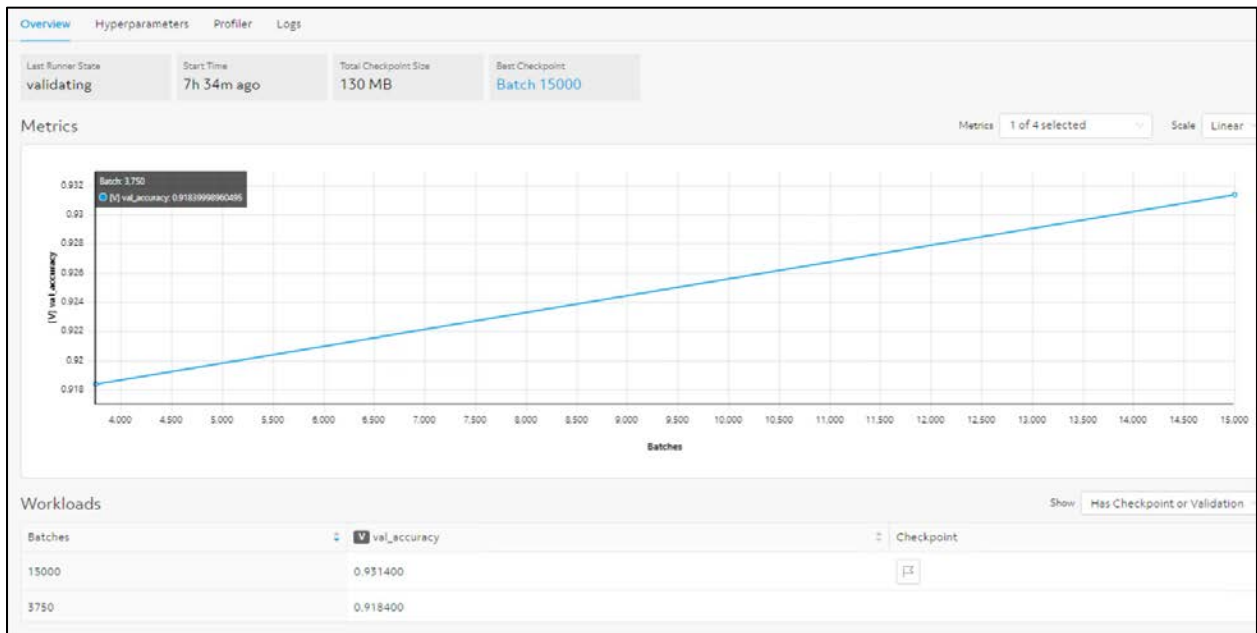
You might notice that the trials trained on the least data have some of the longest durations. The trials were not actually active that full time. Instead Adaptive ASHA paused them, and let other trials run, while waiting to determine if the lower rung trial might be promoted.

- Return to page 1. Then click **Best Validation Metric** to sort by that column. The best trial (with the ID that you noted before) comes to the top of the list.

10. Click the ID for the top trial.

ID	State	Batches	Best Validation Metric	Latest Validation Metric	Start Time	Duration	Checkpoint
Trial 1550	COMPLETED	15000	0.933400	0.933400	1 day ago	5m 59s	
Trial 1337	COMPLETED	15000	0.930300	0.930300	1 day ago	9m 28s	
Trial 1302	COMPLETED	15000	0.928800	0.928800	1 day ago	13m 23s	
Trial 1216	COMPLETED	15000	0.926800	0.926800	1 day ago	12m 6s	
Trial 1176	COMPLETED	15000	0.926700	0.926700	1 day ago	3m	
Trial 1109	COMPLETED	15000	0.926000	0.926000	1 day ago	10m 21s	
Trial 1146	COMPLETED	15000	0.923200	0.923200	1 day ago	15m 42s	
Trial 1158	COMPLETED	3750	0.919800	0.919800	1 day ago	10m 46s	
Trial 1342	COMPLETED	15000	0.918600	0.915900	1 day ago	9m 27s	
Trial 1325	COMPLETED	3750	0.918400	0.918400	1 day ago	7m 12s	

11. As you see, you enter a view of the trial itself, which is very similar to the view of the trial in the experiment with just one trial.



12. Click **Hyperparameters**. Here you see the particular hyperparameters that this trial used.

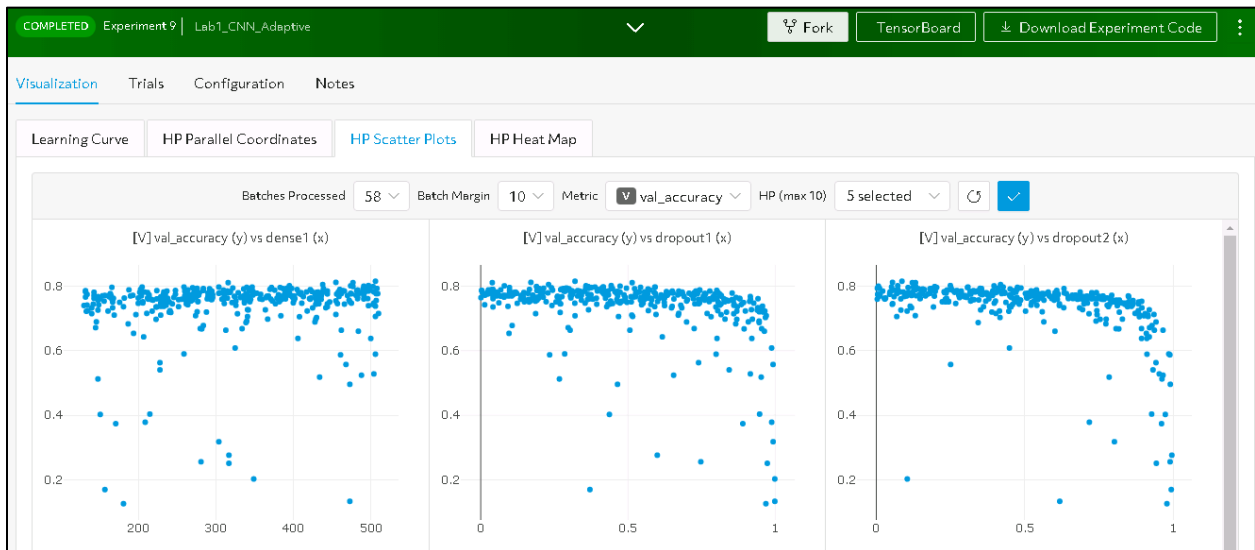


13. In the green bar at the top, click the **Experiment 9** to return to the experiment view.

14. Click **HP Scatter Plots**, which shows the relationship between each type of hyperparameter and the validation loss.

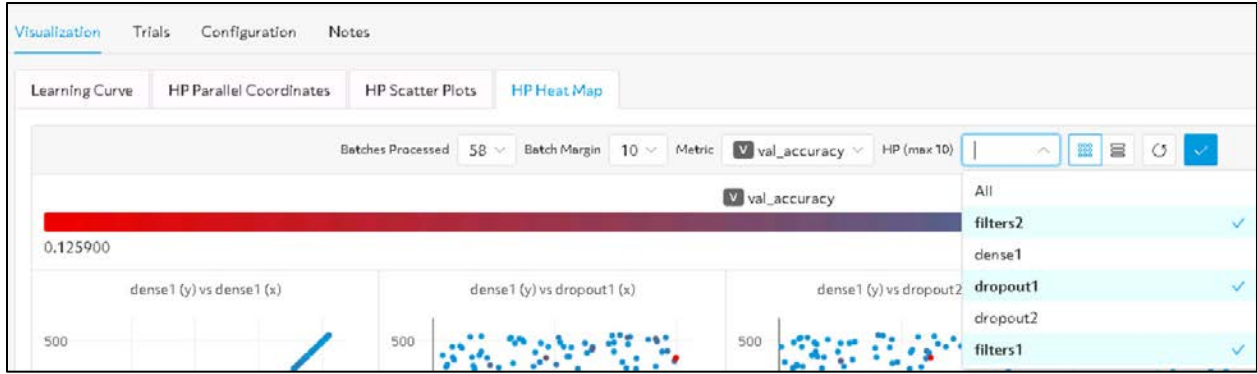
Each graph shows a dot for each trial. The dot is located on the x-axis based on its hyperparameter value and on the y-axis based on its validation accuracy after processing the number of batches indicated in **Batches Processed**. (You can select different values from that menu.)

You can see trends such as that many dropout values work well up until they reach about 1, which is too high a number for this hyperparameter.



15. Click **HP Heat Maps**. The heat maps can help you look at how two hyperparameters interact to affect the validation accuracy.

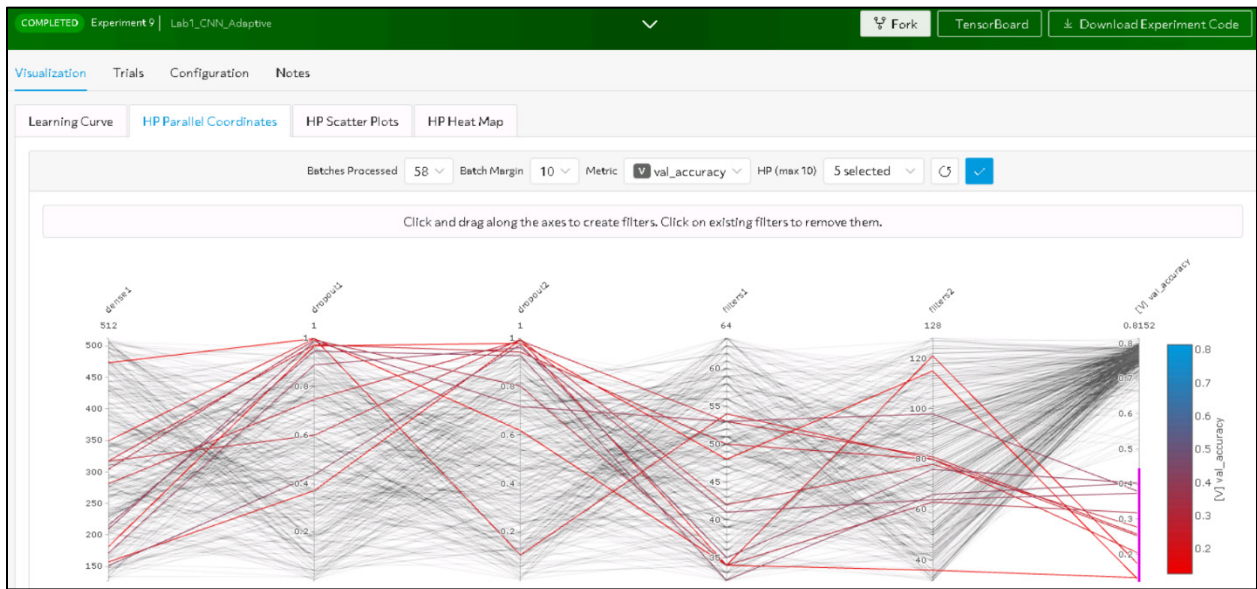
16. You can select which hyperparameters you want to examine. Click the **HP** drop-down menu and select **filters2**, **dropout1**, and **filters1**. Then click the blue check mark.



17. Click **HP Parallel Coordinates**.

18. At first this view might seem hard to navigate, but you can apply filters to focus on what you want to see.

19. On the rightmost axis, click and drag to draw a line between 0 and .4, which is for trials with an accuracy between 0 and 40%. As you see, only the lines for trials that meet these criteria remain. You can see patterns in which combinations of hyperparameters led to these poor results.



Task 3: View the trained model in action

1. Now you will see this trained model in action. Return to the browser tab with the Jupyter Notebook. (Note that if you closed this tab, you can restore it by clicking **Tasks** and then clicking xx-jupyter in which, xx is your seat number.)

Important

You must run the Notebook in order. If for some reason you need to reload the Notebook and start over, make sure to go back to the beginning and run the cells in order.

2. Find the section labeled "Module 1-Lab 2."
3. Run the cell to load the trained CNN model and see it in action. You should see that it classifies all the images in this set correctly.

```
[8]: cnn_adaptive_model = tf.keras.Sequential([
      tf.keras.layers.Conv2D(64, (3,3), input_shape=(28,28,1), padding="same", activation="relu"),
      tf.keras.layers.Conv2D(128, (3,3), padding="same", activation="relu"),
      tf.keras.layers.MaxPool2D(2,2),
      tf.keras.layers.Dropout(.25),
      tf.keras.layers.Flatten(),
      tf.keras.layers.Dense(256, activation="relu"),
      tf.keras.layers.Dropout(.389),
      tf.keras.layers.Dense(10,)]
      cnn_adaptive_model.load_weights("/tmp/determined-checkpoint/b045bc96-326d-42b0-9f9d-40c0d1e5b10e/determined-keras-model-weights").expect_partial()
      print("\033[1m"+"Best model discovered in Adaptive ASHA experiment loaded"+" \033[0m")
      probability_model = tf.keras.Sequential([cnn_adaptive_model,
      tf.keras.layers.Softmax()])
      predictions = probability_model.predict(ch_test_images)
      num_rows = 8
      num_cols = 3
      num_images = num_rows*num_cols
      plt.figure(figsize=(2*2*num_cols, 2*num_rows))
      for i in range(num_images):
          plt.subplot(num_rows, 2*num_cols, 2*i+1)
          plot_image(i, predictions[i], test_images)
          plt.subplot(num_rows, 2*num_cols, 2*i+2)
          plot_value_array(i, predictions[i], test_labels)
      plt.tight_layout()
      print("\033[1m"+"Classifications by CNN model trained with Adaptive ASHA"+" \033[0m")
      print("\033[1m"+"This model has an accuracy over 93% and is now classifying all of these images correctly."+" \033[0m")
      plt.show()
```

Best model discovered in Adaptive ASHA experiment loaded
 Classifications by CNN model trained with Adaptive ASHA
 This model has an accuracy over 93% and is now classifying all of these images correctly.

Ankle boot 100% (Ankle boot) Pullover 100% (Pullover) Trousers 100% (Trousers)

YOU HAVE COMPLETED THIS LAB



Articulate Benefits of HPE Machine Learning Development Environment

Module 2—Activity

Activity overview

You will begin by reading a customer scenario. You will then list benefits that you think will resonate with the customer.

Task 1: Read the customer scenario

This automobile manufacturer wants to become more competitive by enhancing the self-driving features in its vehicles. Computer vision will help the vehicles detect pedestrians, warn drivers of cars in other lanes, and even interpret traffic signals.

The manufacturer has a vast amount of data for training its computer vision models. However, the ML/DL team is only able to use a small amount of this data because the training process takes so long. The customer complains that training a model takes about eight days. The long delays make it hard for the ML/DL team to move projects forward or to be sure that they have created the most accurate models. It's frustrating to have to wait eight days and only then discover an issue.

The customer wants to enhance the training environment and deploy a lot more GPUs to speed up the process.

Currently each training process uses one GPU at a time, and the team uses a point solution for handling scheduling GPU time. The team members would love to use multiple GPUs for training a model, but they are concerned about the amount of time that it will take them to reprogram their models to use distributed training. They are spending a lot of time fine-tuning their code and managing processes such as hyperparameter optimization.

The IT team supporting the ML/DL team is also struggling. The IT team members do not understand the ML/DL workload requirements very well, and they struggle to assemble the resources and environments the ML/DL team needs.

Task 2: Articulate benefits

Think about this customer's pain points. Based on what you learned during the lecture, make a list of HPE Machine Learning Development Environment benefits that you think will resonate with this customer. Include proof points, as relevant.

Module 2—Activity: Articulate Benefits of HPE Machine Learning Development Environment

Lined writing area for articulating benefits of HPE Machine Learning Development Environment.

YOU HAVE COMPLETED THIS LAB.



Explore the HPE Machine Learning Development Environment Architecture


Module 3—Lab

Lab overview

In this lab you will explore the HPE Machine Learning Development Environment architecture. Your lab environment consists of Machine Learning Development Environment software deployed as on-prem cluster. You are sharing this cluster with your classmates. The cluster consists of a conductor and three agents. In this lab, you will explore this cluster. You will also set up the Machine Learning Development Environment CLI to access the cluster.

Task 1: View the Cluster

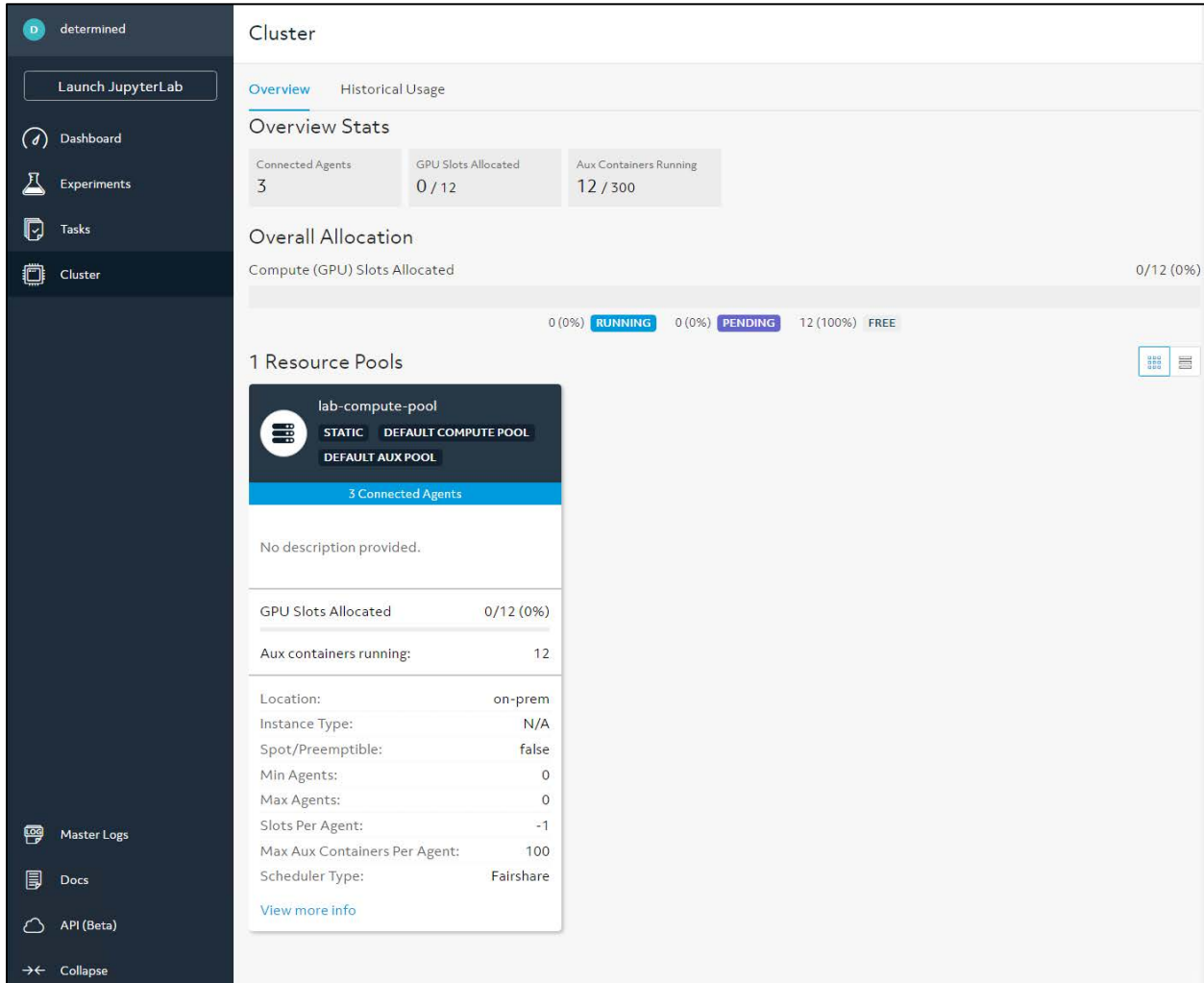
1. Log into the HPE remote lab environment using the credentials provided by your instructor.
2. Open Chrome and access the cluster WebUI at <http://cluster.hpe.local:8080>.
3. Log in with the credentials provided in the Class Info file.


Determined AI

Sign In

Forgot your password, or need to manage users? Check out our [docs](#)

4. In the left navigation menu, click the **Cluster** tab.
5. You should see one resource pool named "lab-compute-pool."



6. Answer these questions about lab-compute-pool.

What is the pool location? _____

How many agents does this pool have? _____

What slot type does it have (GPU or CPU)? _____

How many total slots does it have? _____

7. See at the top of the page that the cluster supports up to 300 aux containers. Each agent supports 100. Note that in the real world, you would typically have a separate auxiliary pool. However, for the purposes of minimizing equipment for the lab, the compute pool is also running auxiliary containers.
8. You will probably see that several aux containers are running. Based on what you learned in the module, what task do you think these aux containers are doing?

9. You can verify your answer by viewing the tasks on the cluster. In the left navigation bar, click **Tasks**.

Short ID	Type	Name	Start Time	State	Resource Pool	User
166533f4		11-jupyter	7 minutes ago	RUNNING	lab-compute-pool	D
ac6aad61		10-jupyter	7 minutes ago	RUNNING	lab-compute-pool	D
c49e6e95		09-jupyter	7 minutes ago	RUNNING	lab-compute-pool	D
0a6d19f2		08-jupyter	7 minutes ago	RUNNING	lab-compute-pool	D
45e372e4		07-jupyter	7 minutes ago	RUNNING	lab-compute-pool	D
fa17f836		06-jupyter	7 minutes ago	RUNNING	lab-compute-pool	D
d98765f3		05-jupyter	7 minutes ago	RUNNING	lab-compute-pool	D
24972107		04-jupyter	8 minutes ago	RUNNING	lab-compute-pool	D
c17edbd4		03-jupyter	8 minutes ago	RUNNING	lab-compute-pool	D
ea8487c8		02-jupyter	8 minutes ago	RUNNING	lab-compute-pool	D

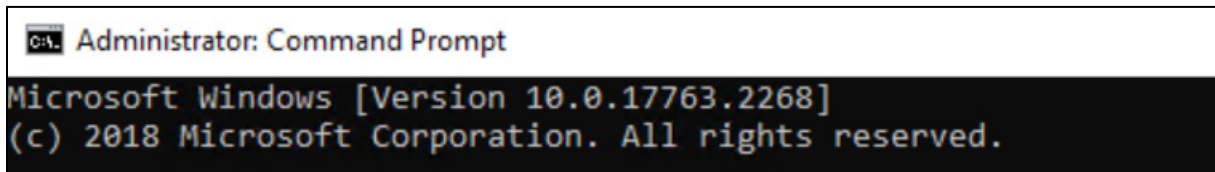
Task 2: Use the HPE Machine Learning Development Environment CLI

Your management server already has the HPE Machine Learning Development Environment CLI installed on it. You will now configure the environmental variable to reference your lab cluster's conductor. You will also practice entering a few commands.

1. From your management (jump) server, open the command prompt, which should be pinned to the task bar at the top of the view.



2. Make sure that the command prompt runs as administrator.



3. Enter this command, which permanently adds the environmental variable:

```
C:\Users\Administrator>setx DET_MASTER cluster.hpe.local
```

4. Close the command prompt and re-open it for the variable to take effect.

5. Enter this command to see a list of agents in the lab cluster:

```
C:\Users\Administrator>det agent list
```

Agent ID Label	Registered Time Addresses	Slots	Containers	Resource Pool	Enabled	Draining
403eb081387b 	2021-11-16 19:02:51+0000 10.8.99.51	4		12 lab-compute-pool	True	False
c7a52e115b37 	2021-11-16 19:02:51+0000 10.8.99.53	4		0 lab-compute-pool	True	False
f07c4a1de80c 	2021-11-16 19:02:48+0000 10.8.99.52	4		0 lab-compute-pool	True	False

Note

If the environmental variable failed to apply properly, you will see this error: [31mFailed to list agents: Forbidden(): Please contact your administrator in order to access this resource.

Make sure that you closed the command prompt and then reopened. If closing and re-opening do not work, try entering this command again, carefully checking for typos: **setx DET_MASTER cluster.hpe.local**. Then close and re-open the command prompt.

6. Enter this command to view the conductor (master) config:

```
C:\Users\Administrator>det master config
```

7. Scroll up through the output and note the "lab-compute-pool" settings.

- As you saw earlier, the pool supports 100 aux containers per agent.
- Because the agents have GPUs, they also contribute GPU slots (no settings required in the pool).
- Why is the provider null? (Hint: think about the pool location.)

```
resource_pools:
- description: ''
  max_aux_containers_per_agent: 100
  pool_name: lab-compute-pool
  provider: null
  task_container_defaults: null
```

8. Continue to scroll up and note that the "lab-compute-pool" is defined as the default aux and default compute pool.

```
resource_manager:
  default_aux_resource_pool: lab-compute-pool
```

```
default_compute_resource_pool: lab-compute-pool
scheduler:
  fitting_policy: best
  type: fair_share
type: agent
```

9. Continue to scroll up and note that checkpoints are stored to a shared file system on the agents.

```
checkpoint_storage:
  host_path: /tmp
  propagation: null
  save_experiment_best: 0
  save_trial_best: 1
  save_trial_latest: 1
  storage_path: determined-checkpoint
type: shared_fs
```

10. Based on the checkpoint_storage settings that you see, what is the path on the agents for where checkpoints will be saved?

YOU HAVE COMPLETED THIS LAB.



Run Experiments on an HPE Machine Learning Development Environment Cluster

Module 4—Labs

Lab overview

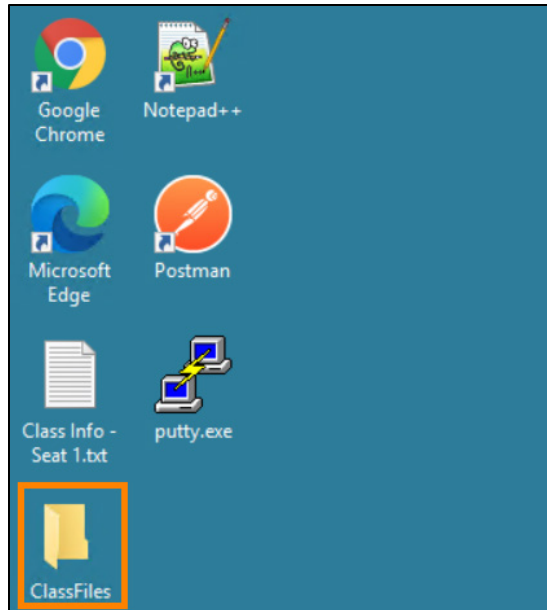
In these labs you will see how HPE Machine Learning Development Environment makes it easy for ML engineers to train deep learning (DL) models on one or more GPUs. You will further see how this environment simplifies hyperparameter optimization (HPO).

Module 4—Lab 1: Port Model Code for HPE Machine Learning Development Environment

In this lab you will practice moving TensorFlow Keras source code into the HPE Machine Learning Development Environment template for TensorFlow Keras model training. You will finish the lab by validating that your code works and Machine Learning Development Environment can run an experiment with it.

Task 1: Port the code

1. Log into the remote lab environment using the URL and credentials provided by your instructor.
2. Open the ClassFiles folder on the desktop.



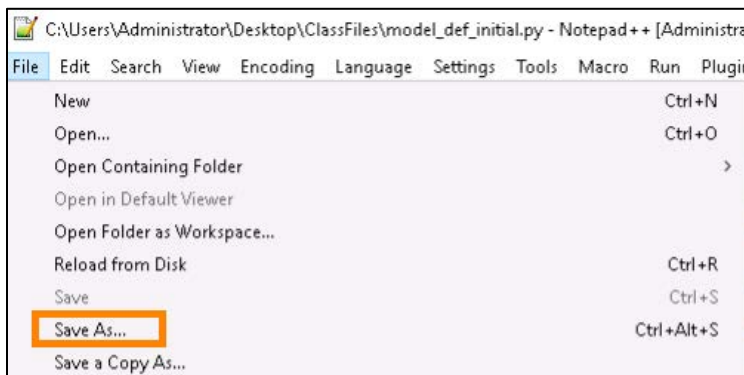
3. Open model_def_initial.
4. You will see the TensorFlow Keras template open in Notepad++.
 - Note the import statements at the top.
 - Note the class definition. The class name is "FashionMNISTTrial" and the class type is "TFKerasTrial."
 - Note the four methods defined under the class.
 - The first method is simply defining the context (which is pulled in from the experiment config).
 - The second method builds and returns the model.
 - The third method builds and returns training data (images plus labels).
 - The fourth method builds and returns validation data (images plus labels).

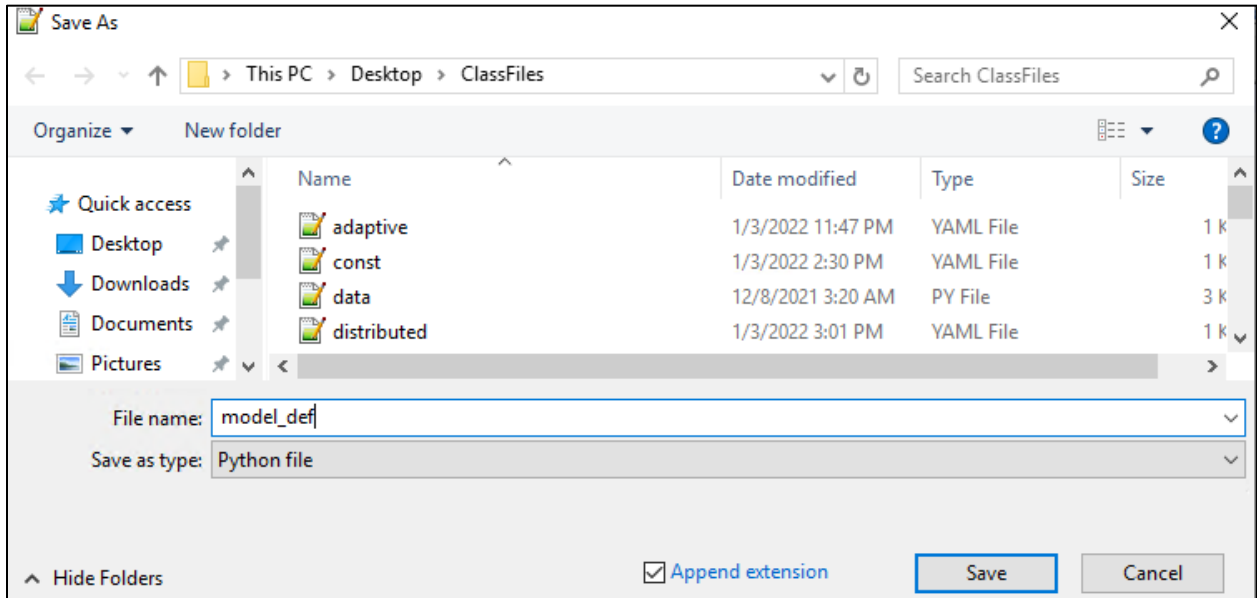
- These methods and the return statements have been defined for you.

```

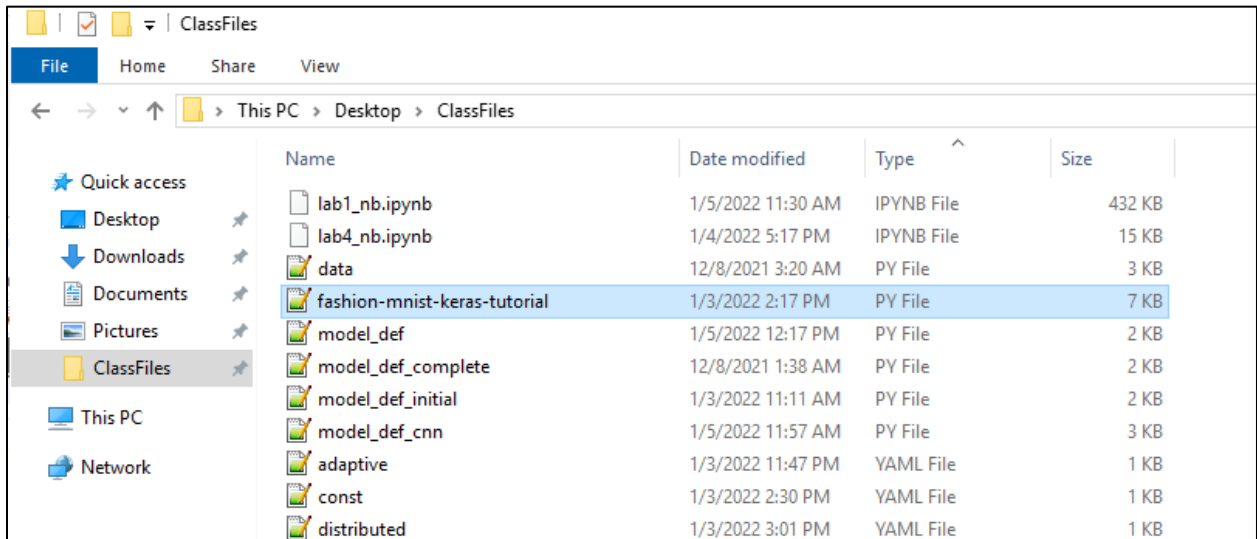
1 """
2 This example shows how to use Determined to implement an image
3 classification model for the Fashion-MNIST dataset using tf.keras.
4
5 Based on: https://www.tensorflow.org/tutorials/keras/classification.
6
7 After about 5 training epochs, accuracy should be around > 85%.
8 This mimics the original implementation. Continue training or increase
9 the number of epochs to increase accuracy.
10 """
11
12
13 import tensorflow as tf
14 from tensorflow import keras
15 from determined.keras import TFKerasTrial, TFKerasTrialContext
16
17
18 class FashionMNISTTrial(TFKerasTrial):
19     def __init__(self, context: TFKerasTrialContext) -> None:
20         # Initialize the trial class.
21         self.context = context
22
23     def build_model(self):
24         # Define and compile model graph.
25
26         return model
27
28     def build_training_data_loader(self):
29         # Create the training data loader. This should return a keras.Sequence,
30         # a tf.data.Dataset, or NumPy arrays.
31
32         return train_images, train_labels
33
34     def build_validation_data_loader(self):
35         # Create the validation data loader. This should return a keras.Sequence,
36         # a tf.data.Dataset, or NumPy arrays.
37
38         return test_images, test_labels
    
```

5. You will now copy code into this template. Prepare by saving the file as **model_def**.





- Next you will open the file with the source code, from which you will copy pieces into "model_def." Return to the ClassFiles folder and double-click **fashion-mnist-keras-tutorial**.



7. Scroll down to line 16, which relates to loading data. Select and copy the line.

```

1 # TensorFlow and tf.keras
2 import tensorflow as tf
3
4 # Helper libraries
5 import numpy as np
6 import matplotlib.pyplot as plt
7
8 print(tf.__version__)
9
10 #This sets a variable to reduce the length of the code below. You can skip this.
11
12 fashion_mnist = tf.keras.datasets.fashion_mnist
13
14 #This is the code for loading data. It belongs under the methods for building training data and validation data.
15
16 (train_images, train_labels), (test_images, test_labels) = fashion_mnist.load_data()
17
18 #The code below gives the classes to which images are assigned user-readable names. It then lets the user explore data shape
19 #You do not need this code for the training process, so do not copy it over.
20
21 class_names = ['T-shirt/top', 'Trouser', 'Pullover', 'Dress', 'Coat',
22               'Sandal', 'Shirt', 'Sneaker', 'Bag', 'Ankle boot']
23
24
25 train_images.shape
26 len(train_labels)
27 train_labels
28 test_images.shape
29 len(test_labels)
30 plt.figure()
31 plt.imshow(train_images[0])
32 plt.colorbar()
33 plt.grid(False)
34 plt.show()
35
36 #The code below processes the image data, ensuring that values fall between 0 and 1.
37 #Copy the first line at the end of the method for building training data and the second line at the end of the method for b
38
39 train_images = train_images / 255.0
40
41 test_images = test_images / 255.0
42

```

Python file | length: 6,309 | lines: 187 | Ln: 16 | Col: 85 | Sel: 84 | 1 | Windows (CRLF) | ANSI | INS

8. Return to the Notepad++ tab for `model_def.py`. Paste the line to *two* places:

- In the empty line above "return train_images, train_labels." Make sure to begin pasting with your cursor directly above "return."
- In the empty line above "return test_images, test_labels." Make sure to begin pasting with your cursor directly above "return."

```

1  """
2  This example shows how to use Determined to implement an image
3  classification model for the Fashion-MNIST dataset using tf.keras.
4
5  Based on: https://www.tensorflow.org/tutorials/keras/classification.
6
7  After about 5 training epochs, accuracy should be around > 85%.
8  This mimics the original implementation. Continue training or increase
9  the number of epochs to increase accuracy.
10 """
11
12
13 import tensorflow as tf
14 from tensorflow import keras
15 from determined.keras import TFKerasTrial, TFKerasTrialContext
16
17
18 class FashionMNISTTrial(TFKerasTrial):
19     def __init__(self, context: TFKerasTrialContext) -> None:
20         # Initialize the trial class.
21         self.context = context
22
23     def build_model(self):
24         # Define and compile model graph.
25
26         return model
27
28     def build_training_data_loader(self):
29         # Create the training data loader. This should return a keras.Sequence,
30         # a tf.data.Dataset, or NumPy arrays.
31         (train_images, train_labels), (test_images, test_labels) = fashion_mnist.load_data()
32         return train_images, train_labels
33
34     def build_validation_data_loader(self):
35         # Create the validation data loader. This should return a keras.Sequence,
36         # a tf.data.Dataset, or NumPy arrays.
37         (train_images, train_labels), (test_images, test_labels) = fashion_mnist.load_data()
38         return test_images, test_labels

```

9. Now edit those lines for this environment. You will delete references to the testing data for the first method and to the training data for the second method. You will also change the method names to reference a data.py file provided as part of the HPE tutorial. (This file has data loading methods based on the MNIST tutorial's methods.)
 - a. In line 31, **delete**:
 , (test_images, test_labels)
 - b. In that same line, change "fashion_mnist.load_data()" to:
 data.load_training_data()
 - c. Line 31 should now read: **(train_images, train_labels) = data.load_training_data()**
 - d. In line 37, **delete**:
 (train_images, train_labels),
 - e. In that same line, change "fashion_mnist.load_data()" to:
 data.load_validation_data()
 - f. Line 37 should now read: **(test_images, test_labels) = data.load_validation_data()**
10. To enable referencing the data.py file, add this text under "from determined.keras import TFKerasTrial, TFKerasTrialContext":

import data

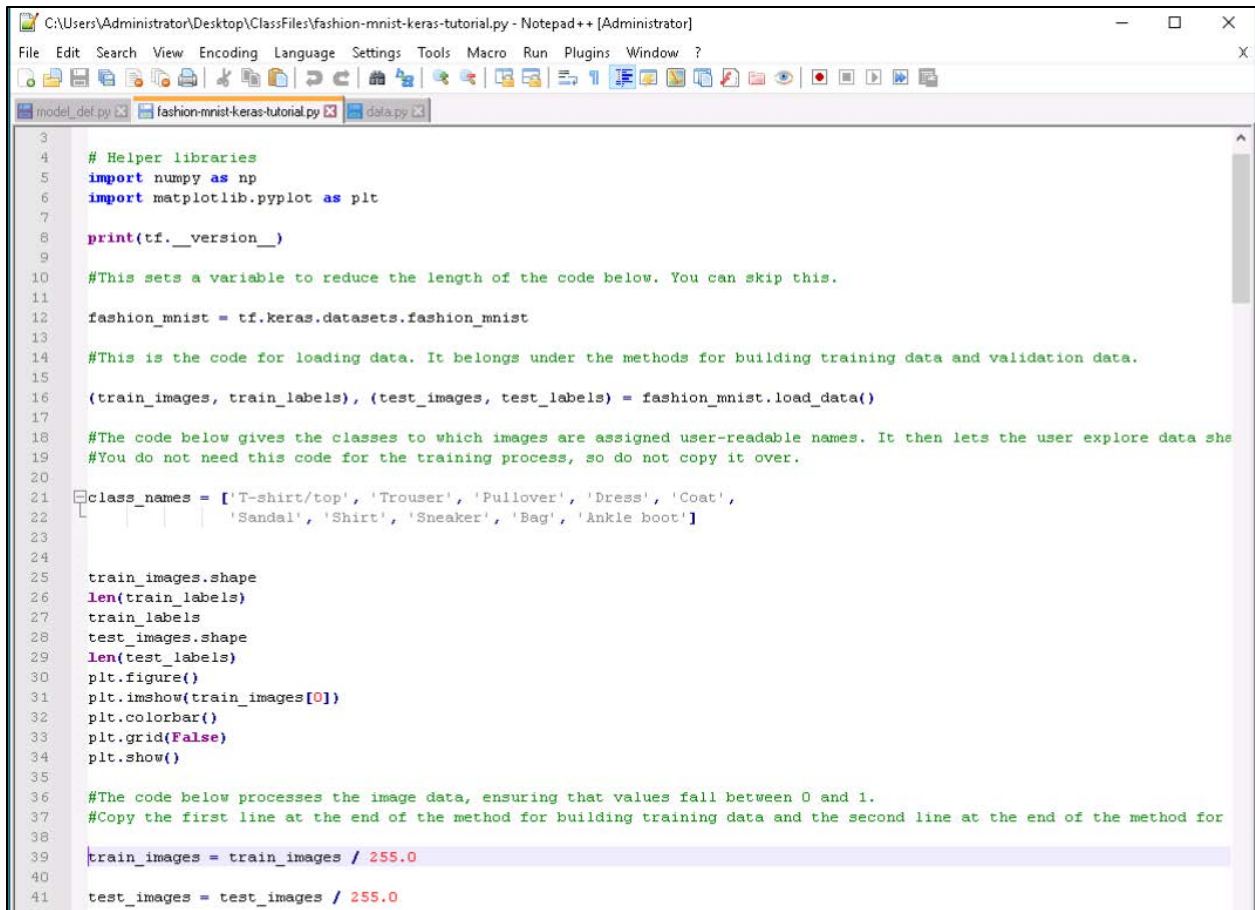
```

13 import tensorflow as tf
14 from tensorflow import keras
15 from determined.keras import TFKerasTrial, TFKerasTrialContext
16 import data
17
18 class FashionMNISTTrial(TFKerasTrial):
19     def __init__(self, context: TFKerasTrialContext) -> None:
20         # Initialize the trial class.
21         self.context = context
22
23     def build_model(self):
24         # Define and compile model graph.
25
26         return model
27
28     def build_training_data_loader(self):
29         # Create the training data loader. This should return a keras.Sequence,
30         # a tf.data.Dataset, or NumPy arrays.
31         (train_images, train_labels) = data.load_training_data()
32         return train_images, train_labels
33
34     def build_validation_data_loader(self):
35         # Create the validation data loader. This should return a keras.Sequence,
36         # a tf.data.Dataset, or NumPy arrays.
37         (test_images, test_labels) = data.load_validation_data()
38         return test_images, test_labels

```

11. Now you will copy in some more code for loading the data.

- a. Return to the Notepad++ tab for fashion-mnist-keras-tutorial.py.
- b. Select and copy all of line 39: **train_images = train_images / 255.0**



```
C:\Users\Administrator\Desktop\ClassFiles\fashion-mnist-keras-tutorial.py - Notepad++ [Administrator]
File Edit Search View Encoding Language Settings Tools Macro Run Plugins Window ?
model_def.py fashion-mnist-keras-tutorial.py data.py
3
4 # Helper libraries
5 import numpy as np
6 import matplotlib.pyplot as plt
7
8 print(tf.__version__)
9
10 #This sets a variable to reduce the length of the code below. You can skip this.
11
12 fashion_mnist = tf.keras.datasets.fashion_mnist
13
14 #This is the code for loading data. It belongs under the methods for building training data and validation data.
15
16 (train_images, train_labels), (test_images, test_labels) = fashion_mnist.load_data()
17
18 #The code below gives the classes to which images are assigned user-readable names. It then lets the user explore data she
19 #You do not need this code for the training process, so do not copy it over.
20
21 class_names = ['T-shirt/top', 'Trouser', 'Pullover', 'Dress', 'Coat',
22               'Sandal', 'Shirt', 'Sneaker', 'Bag', 'Ankle boot']
23
24
25 train_images.shape
26 len(train_labels)
27 train_labels
28 test_images.shape
29 len(test_labels)
30 plt.figure()
31 plt.imshow(train_images[0])
32 plt.colorbar()
33 plt.grid(False)
34 plt.show()
35
36 #The code below processes the image data, ensuring that values fall between 0 and 1.
37 #Copy the first line at the end of the method for building training data and the second line at the end of the method for
38
39 train_images = train_images / 255.0
40
41 test_images = test_images / 255.0
```


- c. Return to the model_def.py tab.
- d. At the end of line 31, which begins "(train_images," press **[Enter]**. Then paste in the line.
- e. Return to the fashion-mnist-keras-tutorial.py tab; select and copy all of line 41: **test_images = test_images / 255.0**
- f. Return to the model_def.py tab.
- g. At the end of line 38, which begins "(test_images," press **[Enter]**. Then paste the line.

```

4
5   Based on: https://www.tensorflow.org/tutorials/keras/classification.
6
7   After about 5 training epochs, accuracy should be around > 85%.
8   This mimics the original implementation. Continue training or increase
9   the number of epochs to increase accuracy.
10  """
11
12
13  import tensorflow as tf
14  from tensorflow import keras
15  from determined.keras import TFKerasTrial, TFKerasTrialContext
16  import data
17
18  class FashionMNISTTrial(TFKerasTrial):
19      def __init__(self, context: TFKerasTrialContext) -> None:
20          # Initialize the trial class.
21          self.context = context
22
23      def build_model(self):
24          # Define and compile model graph.
25
26          return model
27
28      def build_training_data_loader(self):
29          # Create the training data loader. This should return a keras.Sequence,
30          # a tf.data.Dataset, or NumPy arrays.
31          (train_images, train_labels) = data.load_training_data()
32          train_images = train_images / 255.0
33          return train_images, train_labels
34
35      def build_validation_data_loader(self):
36          # Create the validation data loader. This should return a keras.Sequence,
37          # a tf.data.Dataset, or NumPy arrays.
38          (test_images, test_labels) = data.load_validation_data()
39          test_images = test_images / 255.0
40          return test_images, test_labels

```

12. Now you are ready for pasting in the code for building the model.

- a. Return to the fashion-mnist-keras-tutorial.py tab. Select and copy lines 57 through 65 with the code for creating and compiling the model.

```

55 #The code below defines and then compiles the model. Copy it under the method for building the model.
56
57 model = tf.keras.Sequential([
58     tf.keras.layers.Flatten(input_shape=(28, 28)),
59     tf.keras.layers.Dense(128, activation='relu'),
60     tf.keras.layers.Dense(10)
61 ])
62
63 model.compile(optimizer='adam',
64               loss=tf.keras.losses.SparseCategoricalCrossentropy(from_logits=True),
65               metrics=[tf.keras.metrics.SparseCategoricalAccuracy(name="accuracy")])
    
```

- b. Return to the model_def.py tab.
- c. Place your cursor at the *beginning* of line 25 (above "return model.") It is best to start with *no* indent so the indents do not get misaligned.
- d. Paste the copied text.
- e. Now you need to properly indent this code under the "build_model" class. Select lines 25-33.

```

23     def build_model(self):
24         # Define and compile model graph.
25     model = tf.keras.Sequential([
26         tf.keras.layers.Flatten(input_shape=(28, 28)),
27         tf.keras.layers.Dense(128, activation='relu'),
28         tf.keras.layers.Dense(10)
29     ])
30
31     model.compile(optimizer='adam',
32                  loss=tf.keras.losses.SparseCategoricalCrossentropy(from_logits=True),
33                  metrics=[tf.keras.metrics.SparseCategoricalAccuracy(name="accuracy")])
34     return model
    
```

- f. Press **[Tab]** and then **[Tab]** again.
- g. Make sure that the indents match exactly what you see below.

```

23     def build_model(self):
24         # Define and compile model graph.
25         model = tf.keras.Sequential([
26             tf.keras.layers.Flatten(input_shape=(28, 28)),
27             tf.keras.layers.Dense(128, activation='relu'),
28             tf.keras.layers.Dense(10)
29         ])
30
31         model.compile(optimizer='adam',
32                       loss=tf.keras.losses.SparseCategoricalCrossentropy(from_logits=True),
33                       metrics=[tf.keras.metrics.SparseCategoricalAccuracy(name="accuracy")])
34     return model
    
```

13. Now you need to wrap the model and optimizer so that HPE Machine Learning Development Environment can apply its own functions to them.
 - a. Add a new line below the "model" definition. The line must be below the **])** line and at the same indent level as "model" above.
 - b. Add this code on the new line: **model = self.context.wrap_model(model)**
 - c. Press **[Enter]** twice.
 - d. Add this code on the new line to create the optimizer: **optimizer = tf.keras.optimizers.Adam()**
 - e. Press **[Enter]**.
 - f. Add this code on the new line to wrap the optimizer: **optimizer = self.context.wrap_optimizer(optimizer)**
 - g. Change the model.compile method to reference the optimizer variable. In the line that begins "model.compile," change 'adam' to: **optimizer** *without quotation marks*.

```

23 def build_model(self):
24     # Define and compile model graph.
25     model = tf.keras.Sequential([
26         tf.keras.layers.Flatten(input_shape=(28, 28)),
27         tf.keras.layers.Dense(128, activation='relu'),
28         tf.keras.layers.Dense(10)
29     ])
30     model = self.context.wrap_model(model)
31
32     optimizer = tf.keras.optimizers.Adam()
33     optimizer = self.context.wrap_optimizer(optimizer)
34
35     model.compile(optimizer=optimizer,
36                  loss=tf.keras.losses.SparseCategoricalCrossentropy(from_logits=True),
37                  metrics=[tf.keras.metrics.SparseCategoricalAccuracy(name="accuracy")])
38     return model

```

14. Replace the hyperparameter for the number of nodes in the Dense layer with the method for pulling the "dense1" hyperparameter from the experiment config.

In line 27 (the *first* line that begins "tf.keras.layers.Dense"), replace "128" with: **self.context.get_hparam("dense1")**

```

def build_model(self):
    # Define and compile model graph.
    model = tf.keras.Sequential([
        tf.keras.layers.Flatten(input_shape=(28, 28)),
        tf.keras.layers.Dense(self.context.get_hparam("dense1"), activation='relu'),
        tf.keras.layers.Dense(10)

```

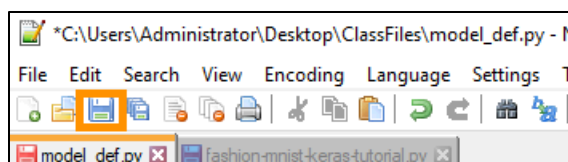
15. Make sure that your file looks like the figure below.

```

1  """
2  This example shows how to use Determined to implement an image
3  classification model for the Fashion-MNIST dataset using tf.keras.
4
5  Based on: https://www.tensorflow.org/tutorials/keras/classification.
6
7  After about 5 training epochs, accuracy should be around > 85%.
8  This mimics the original implementation. Continue training or increase
9  the number of epochs to increase accuracy.
10 """
11
12
13 import tensorflow as tf
14 from tensorflow import keras
15 from determined.keras import TFKerasTrial, TFKerasTrialContext
16 import data
17
18 class FashionMNISTTrial(TFKerasTrial):
19     def __init__(self, context: TFKerasTrialContext) -> None:
20         # Initialize the trial class.
21         self.context = context
22
23     def build_model(self):
24         # Define and compile model graph.
25         model = tf.keras.Sequential([
26             tf.keras.layers.Flatten(input_shape=(28, 28)),
27             tf.keras.layers.Dense(self.context.get_hparam("dense1"), activation='relu'),
28             tf.keras.layers.Dense(10)
29         ])
30         model = self.context.wrap_model(model)
31
32         optimizer = tf.keras.optimizers.Adam()
33         optimizer = self.context.wrap_optimizer(optimizer)
34
35         model.compile(optimizer=optimizer,
36                       loss=tf.keras.losses.SparseCategoricalCrossentropy(from_logits=True),
37                       metrics=[tf.keras.metrics.SparseCategoricalAccuracy(name="accuracy")])
38         return model
39
40     def build_training_data_loader(self):
41         # Create the training data loader. This should return a keras.Sequence,
42         # a tf.data.Dataset, or NumPy arrays.
43         (train_images, train_labels) = data.load_training_data()
44         train_images = train_images / 255.0
45         return train_images, train_labels
46
47     def build_validation_data_loader(self):
48         # Create the validation data loader. This should return a keras.Sequence,
49         # a tf.data.Dataset, or NumPy arrays.
50         (test_images, test_labels) = data.load_validation_data()
51         test_images = test_images / 255.0
52         return test_images, test_labels

```

16. Save the model_def.py file.



17. You can quickly verify that your code does not have syntax errors.

- a. Open the command prompt.
- b. Move to the Desktop\ClassFiles folders.

```
C:\Users\Administrator>cd desktop\classfiles
```

- c. Enter this command to run the code and see output:

```
C:\Users\Administrator\Desktop\ClassFiles>python model_def.py
```

```
Traceback (most recent call last):
```

```
File "C:\Users\Administrator\Desktop\ClassFiles\model_def.py", line 13, in <module>
```

```
import tensorflow as tf
```

```
ModuleNotFoundError: No module named 'tensorflow'
```

- d. You should see an error that there is no module named "tensorflow."
- e. Leave the command prompt open.

If you received the expected error, move on to Task 2. If instead you see an error about indents or syntax, return to model_def.py in Notepad++. Use the error message and the figure on the previous page to help you adjust the code. Save and try to run the code again as in step 17c.

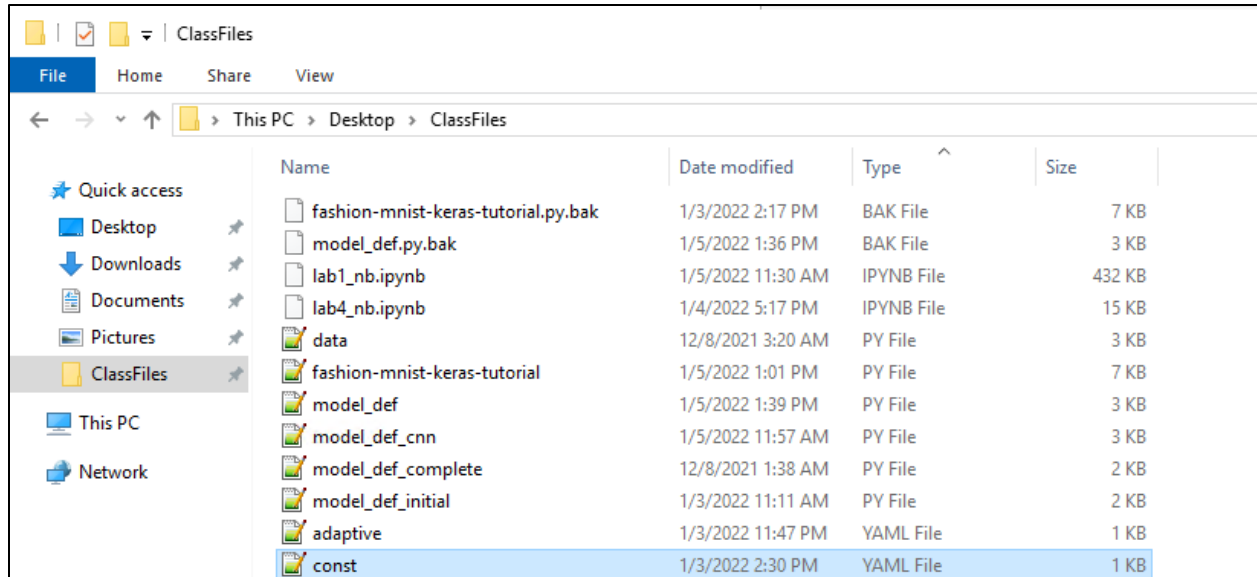
Important

If you continue to struggle, you can delete your "model_def.py." Open "model_def_complete.py," which is provided in the ClassFiles folder. Save that file as "model_def.py."

Task 2: Validate the code works

You will now run an experiment with the code to validate completely that it is working correctly.

1. You can now close the two files open in Notepad++.
2. Return to the ClassFiles folder.
3. Open the **const** file with Notepad++.



- In the first line of the config, change XX- to your seat number (such as 01 or 12). *Make sure to leave a space after the colon.*

name: XX-const

- Note that this experiment file specifies a bind mount, which allows the container to pull training and validation data from a shared file system on the agents. Do *not* make any changes here.

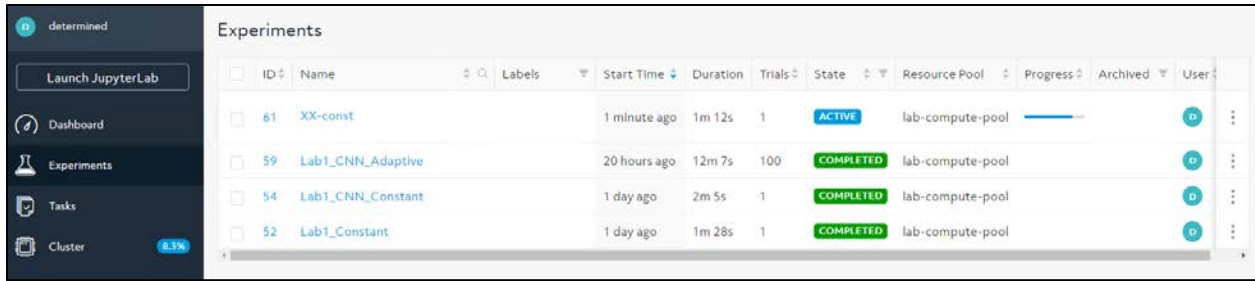
bind_mounts:

- host_path: /tmp/data
- container_path: /tmp/data
- read_only: true

- Save the file.
- Return to the command prompt. Enter this command to run the experiment. **Make sure to include the period at the end of the command.**

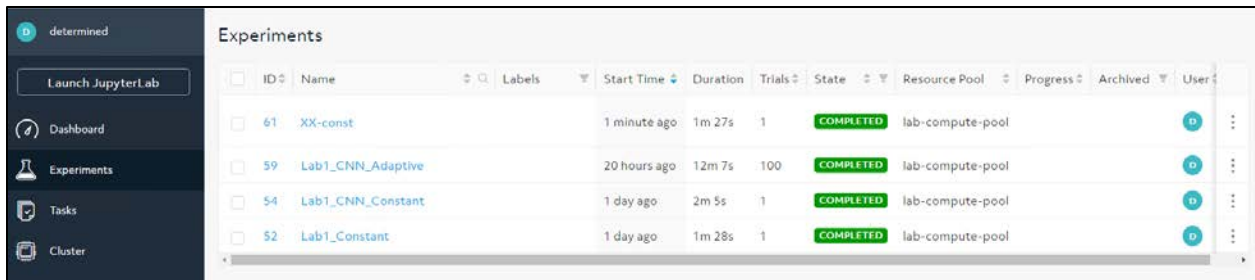
C:\Users\Administrator\Desktop\ClassFiles> **det experiment create const.yaml .**

- Leave the command prompt open.
- Open Chrome and navigate to <http://cluster.hpe.local:8080>.
- Log in with the credentials in the Class Info file.
- In the left navigation bar, click **Experiments**.
- Look for your experiment in the list based on your seat number. You should see that the experiment has one trial associated with it. Wait a minute or two and verify that the progress bar appears.



13. Wait a minute or two and verify that the experiment has completed.

14. Leave the WebUI open.



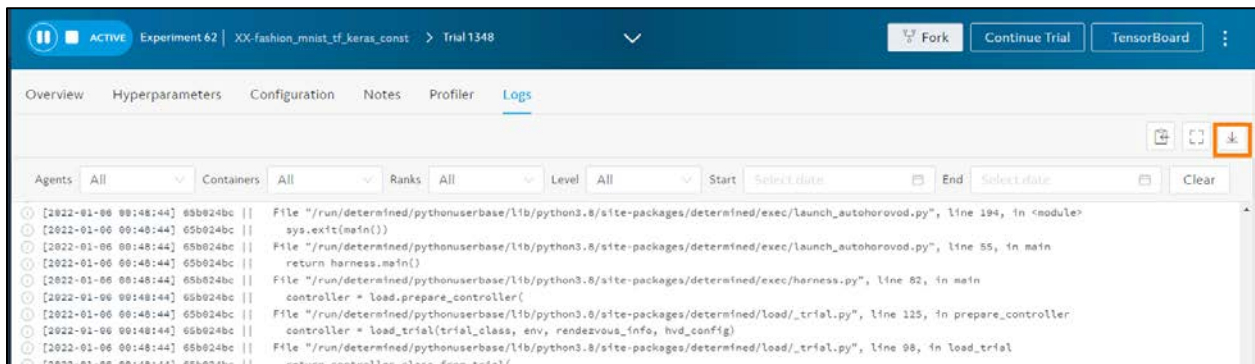
If it takes more than two or three minutes for your experiment to progress or complete, the code might have an error. Read the section below for troubleshooting hints.

YOU HAVE COMPLETED THIS LAB.

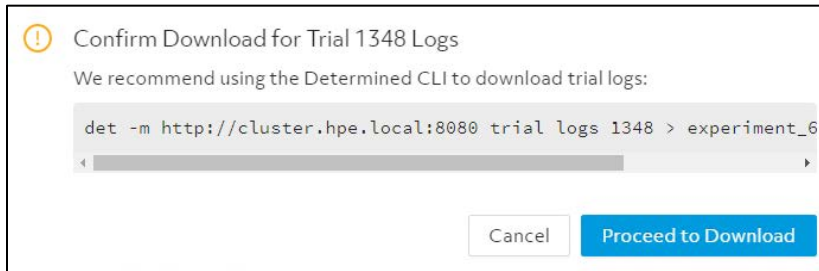
Troubleshooting hints

You can look for clues to the code error as follows:

1. Click the name of your experiment.
2. Click the **Logs** tab.
3. Click the download icon.



4. Click **Proceed to Download**.



5. Find the file in the downloads folder and open it.

6. Search for "error." Look for clues as to the issue.

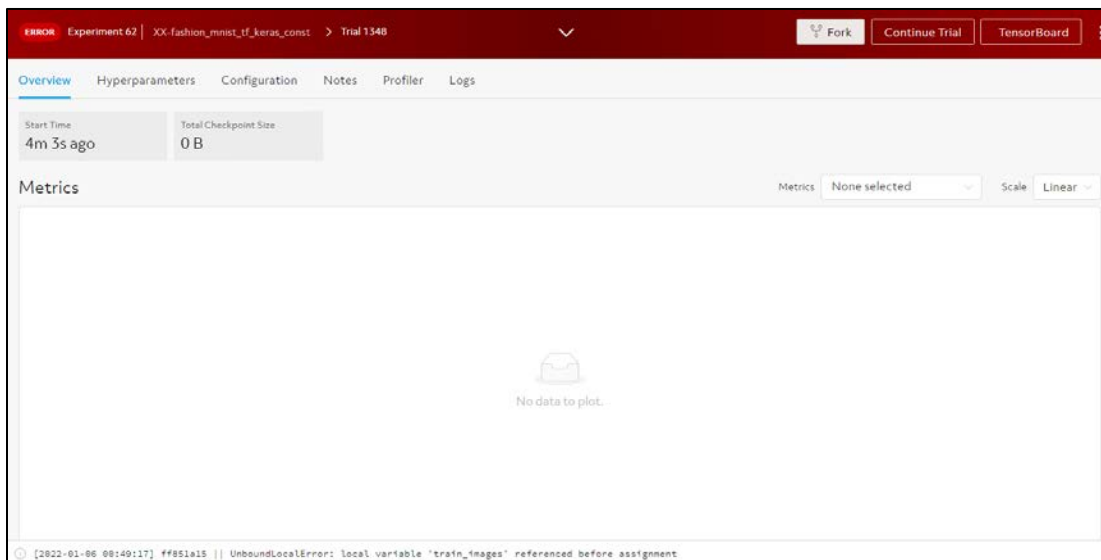
The figure below shows an example. One of the variables in the model code is incorrect.

```
[2022-01-05T18:54:24Z] 12f74ca3 || File "/run/determined/workdir/model_def.py", line 44, in build_training_data_loader
[2022-01-05T18:54:24Z] 12f74ca3 ||     train_images = train_images / 255.0
[2022-01-05T18:54:24Z] 12f74ca3 || UnboundLocalError: local variable 'train_images' referenced before assignment
```

You can return to the model code and search for that variable. In this example, the error was that line 43 set "test_images, test_labels" instead of "train_images, train_labels" as it should have. Therefore, "train_images" in the next line was not defined.

```
40 def build_training_data_loader(self):
41     # Create the training data loader. This should return a keras.Sequence,
42     # a tf.data.Dataset, or NumPy arrays.
43     (test_images, test_labels) = data.load_training_data()
44     train_images = train_images / 255.0
45     return train_images, train_labels
```

7. If you wait for the experiment to have the Error status, you can also look at the bottom of the **Overview** tab for the issue.



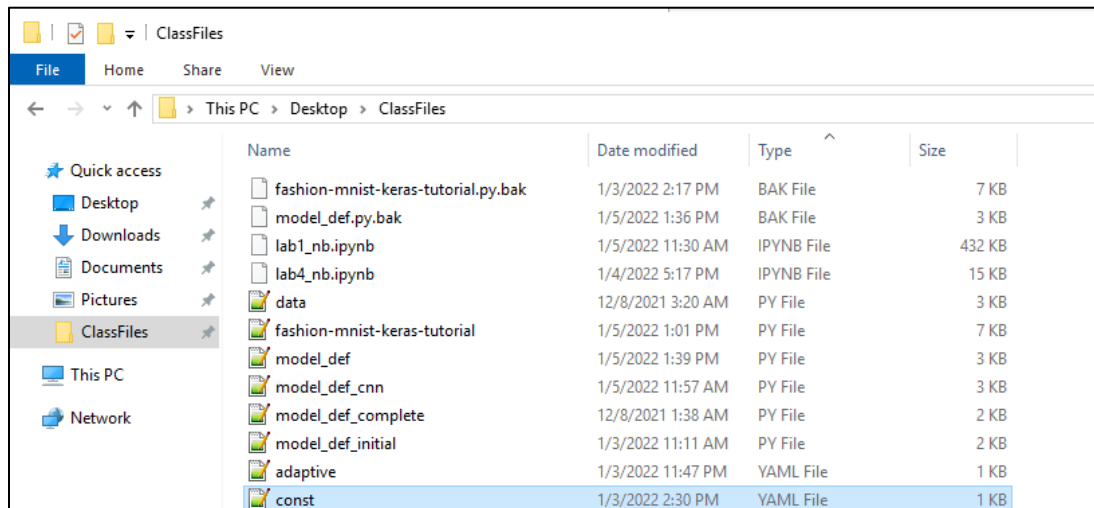
YOU HAVE COMPLETED THIS LAB.

Module 4—Lab 2: Run an Experiment with a Single, Non-Distributed Trial

In this lab, you will examine the settings in `const.yaml` more closely. You will adjust these settings to add a min validation period. You will then run the experiment again. After the experiment concludes, you will explore information about it collected in the WebUI. You will further find the checkpoint UUID for the best model created by the training process. You will then run a Jupyter Notebook, in which you can see that checkpointed model in action.

Task 1: Edit the experiment config file

1. Log into the HPE remote lab environment.
2. Access the ClassFiles folder and open the `const` file.



3. You should see the YAML file in Notepad++.
4. Add a "b" to the name as follows:

`name: XXb-const`

5. Answer this question:

What is the searcher name, and what does this mean? _____

6. Edit the length that the trial runs. Under `max_length`, edit the line as shown:

`max_length:`

`epochs: 8`

7. Add a validation period so that you can validate the trial periodically before it completes. Add these lines to the end of the file. Remember to include two spaces at the beginning of the second line.

`min_validation_period:`

`epochs: 1`

- The default checkpoint policy is to checkpoint the validated model if it is the best model. You will accept that default for this lab.

```

1  name: XXb-const
2  bind_mounts:
3  - host_path: /tmp/data
4    container_path: /tmp/data
5    read_only: true
6  hyperparameters:
7    global_batch_size: 32
8    densel: 128
9    records_per_epoch: 60000
10 searcher:
11   name: single
12   metric: val_accuracy
13   smaller_is_better: false
14   max_length:
15     epochs: 8
16   entrypoint: model_def:FashionMNISTTrial
17   min_validation_period:
18     epochs: 1
    
```

- Save and close the file.

Task 2: Run the experiment

- Return to the command prompt and make sure you are at this command prompt:

```
C:\Users\Administrator\Desktop\ClassFiles>
```

- Enter this command:

```
C:\Users\Administrator\Desktop\ClassFiles> det experiment create const.yaml .
```

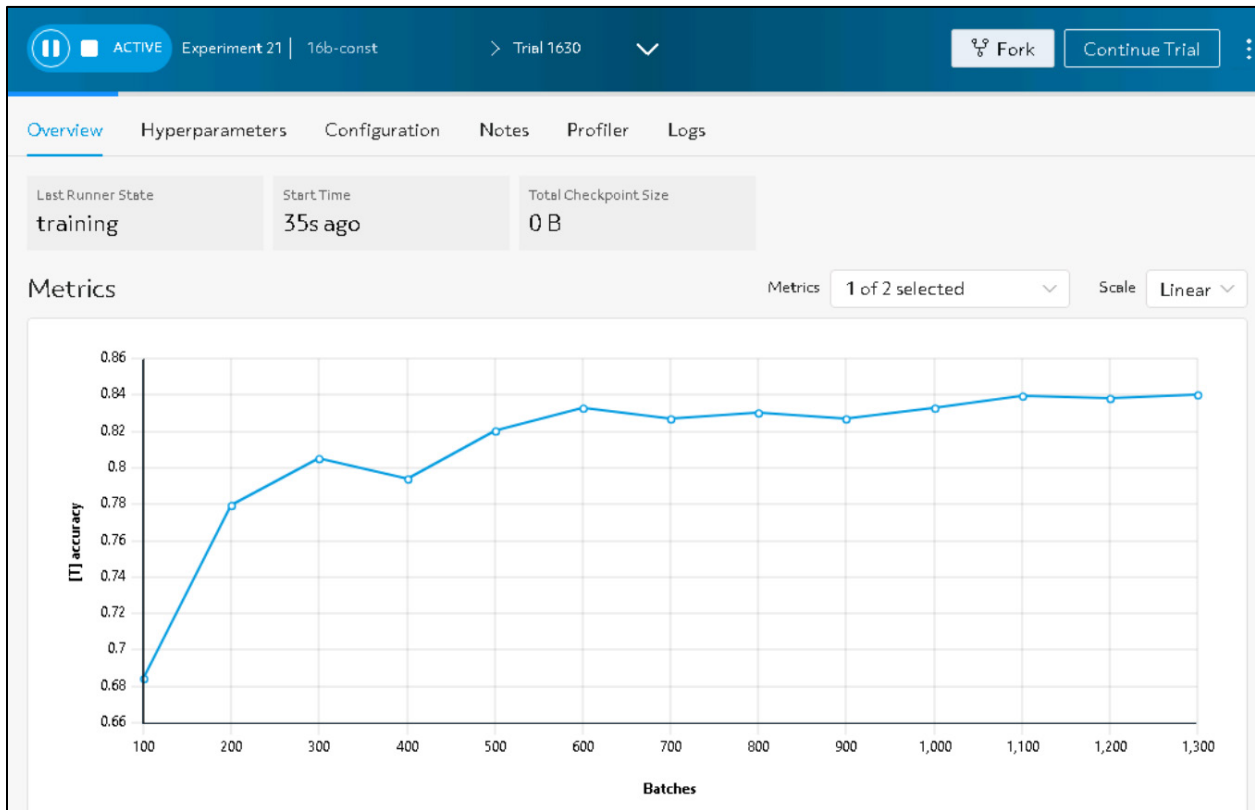
- Make sure that the experiment was created.
- Leave the command prompt open.

Task 3: Monitor the training

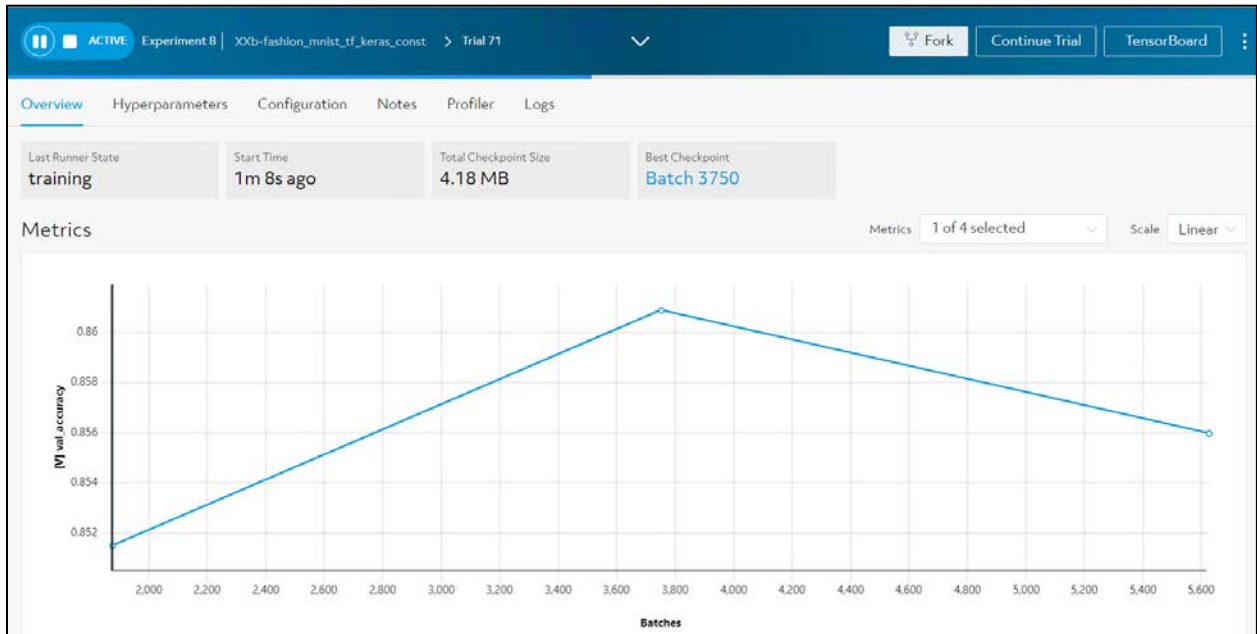
1. Return to the WebUI. If necessary, log in again.
2. Click the experiment name that begins with XXb, in which XX is your seat number.

ID	Name	Labels	Start Time	Duration	Trials	State	Resource Pool	Progress	Archived	User
8	XXb-fashion_mnist_tf_keras_const		32 seconds ago	32s	1	ACTIVE	lab-compute-pool			
7	XX-fashion_mnist_tf_keras_const		15 minutes ago	1m 24s	1	COMPLETED	lab-compute-pool			
4	Lab1_Adaptive		2 days ago	9m 4s	64	COMPLETED	lab-compute-pool			
3	Lab1_Constant		2 days ago	1m 44s	1	COMPLETED	lab-compute-pool			

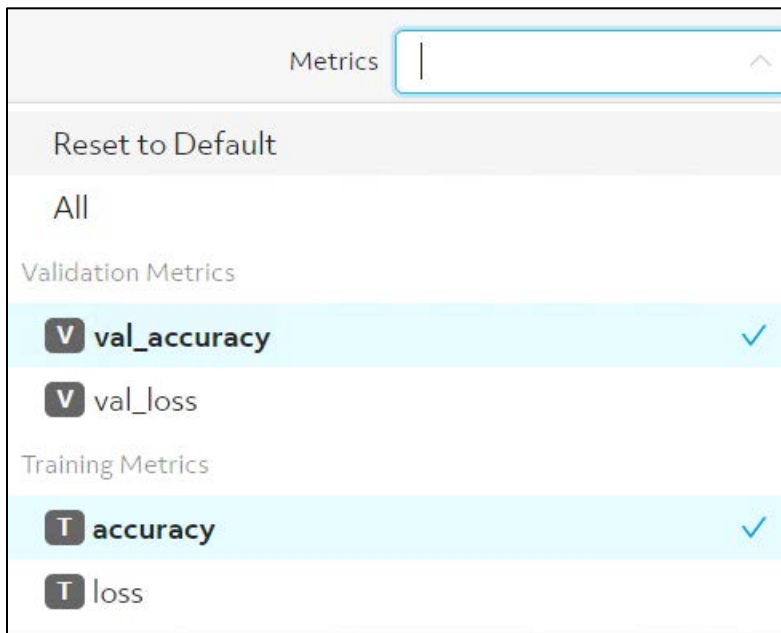
3. Initially the graph is showing the training accuracy metric. You can see this graph changing in real time as the experiment runs. Note, though, that it will take a minute or two for the container to come up and the trial to start to run.



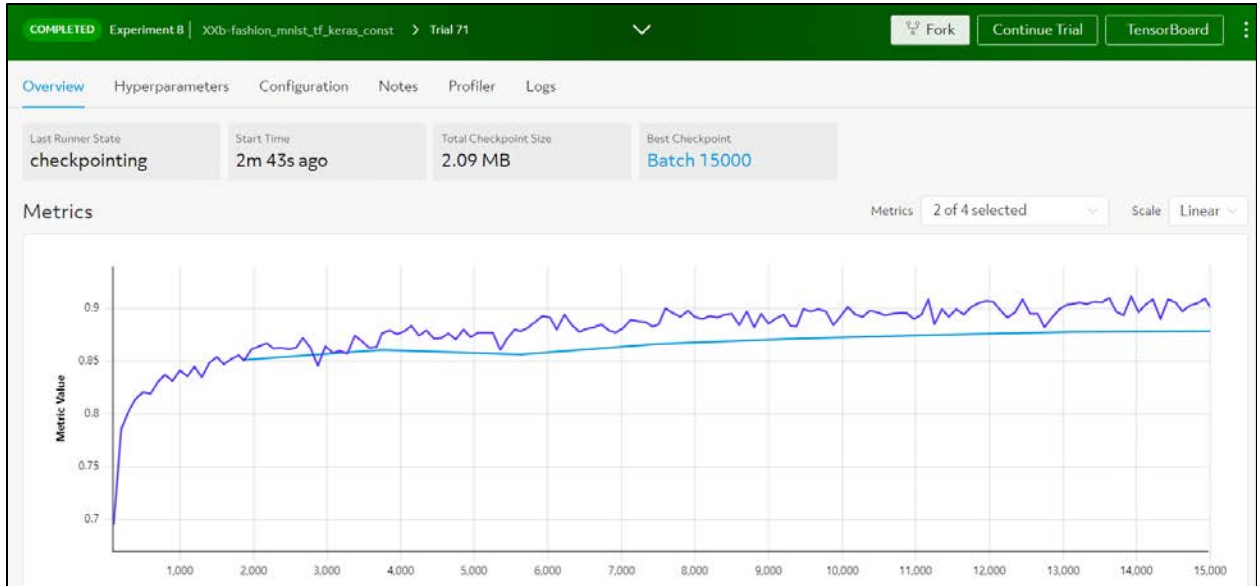
4. When the first validation occurs, the graph automatically switches to showing the `val_accuracy`.



5. From the **Metrics** menu, under **training**, select **accuracy**. Leave **val_accuracy** selected. (The accuracy metric indicates the model accuracy on training data while the `val_accuracy` metric indicates the model's accuracy on validation data.)



- Click away from the menu. The conductor has stored one data point for each training workload (100 batches). You should see that the accuracy sharply increased at first and then gradually increased with ups and downs on the way.



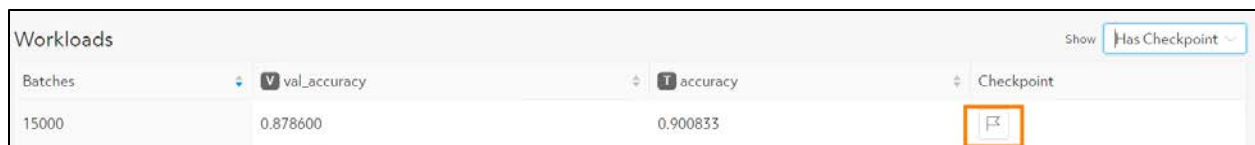
- Scroll down to the bottom of the window, which shows a list of training workloads and their metrics for the metric types selected above.
- Initially this list only shows workloads after which validation or checkpoint occurred. If you make it here before the training completes, you might see multiple batches with checkpoints. But once the training completes, you will only see one or two. Think about why that is the case.

Batches	val_accuracy	accuracy	Checkpoint
11250	0.879900	0.886667	☐
9375	0.877900	0.890833	☐
7500	0.864300	0.892083	☐
5625	0.857800	0.871667	☐

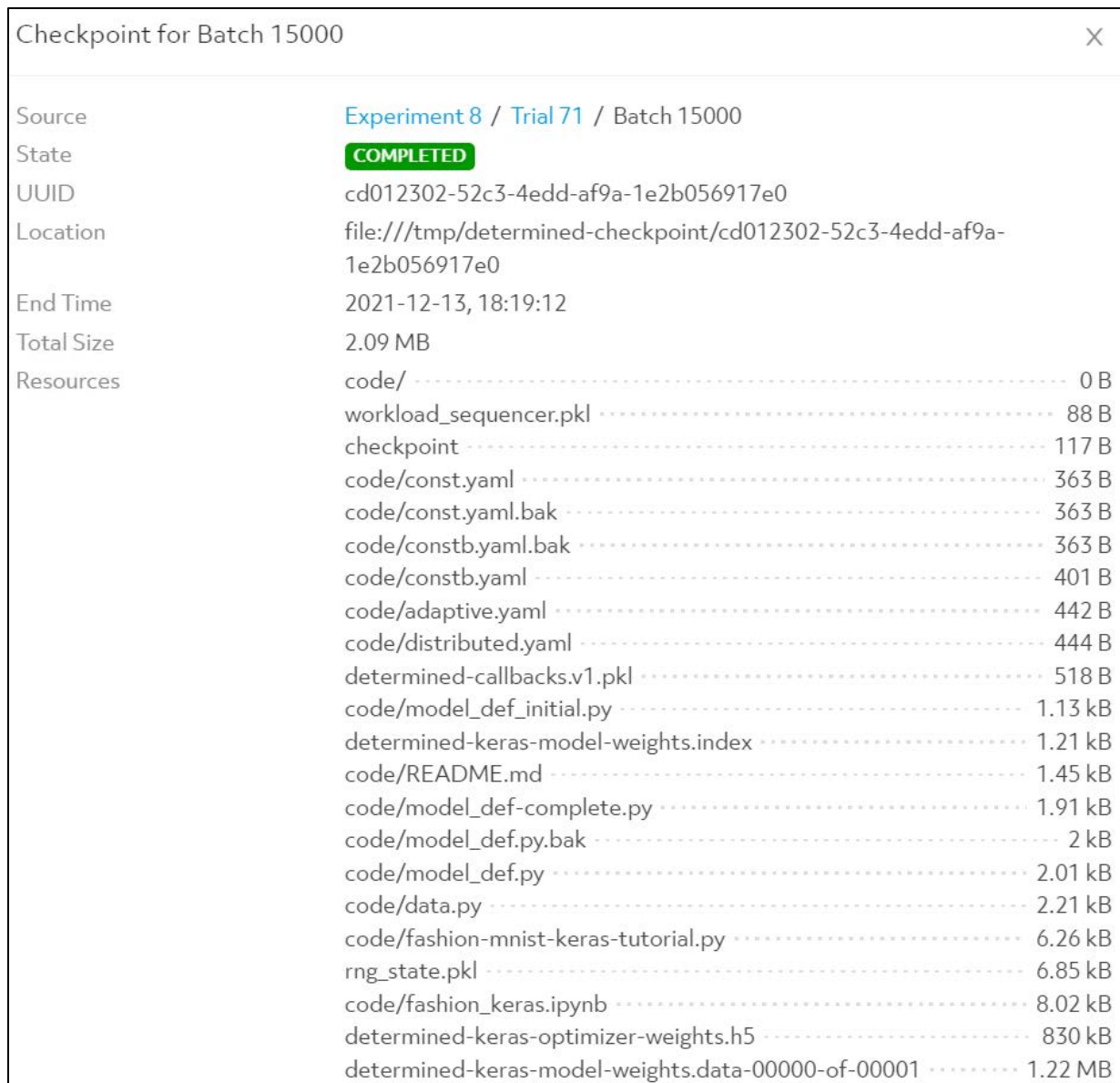
- Select the **Show** menu and select **All**. You will then see all of the workloads.

Batches	val_accuracy	accuracy	Checkpoint
15000	0.878600	0.900833	☐
14925		0.909063	
14825		0.904688	
14725		0.902812	
14625		0.897187	
14525		0.905312	

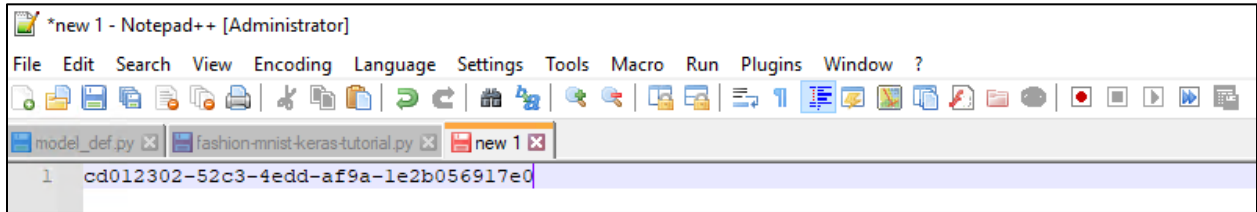
10. In the **Show** menu, select **Has Checkpoint** again to focus on the checkpoints.
11. Once the training has completed (15000 batches), you might see one or two workloads with checkpoints. If you see two, identify the one with the higher accuracy. Click the checkpoint flag icon.



12. You will see information about the checkpoint, including where the checkpoint is stored and the names of the stored files. These files describe the trained model and its training context.
13. Copy the UUID for later reference.
 - a. Select and copy all of the UUID (which begins with "cd" and ends with "e0" in this example, but will be different for you).



- b. Open Notepad++ and select **File > New**.
- c. Paste in the UUID.

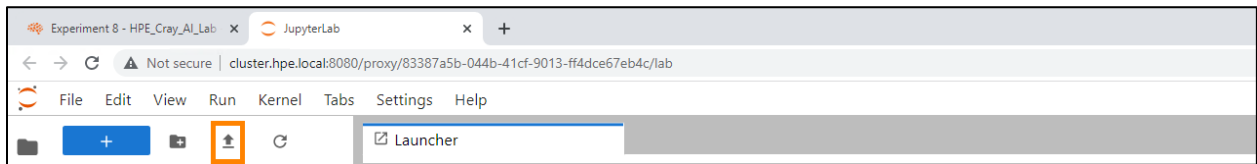


- d. Save the file to Desktop\ClassFiles as **myuuids**.

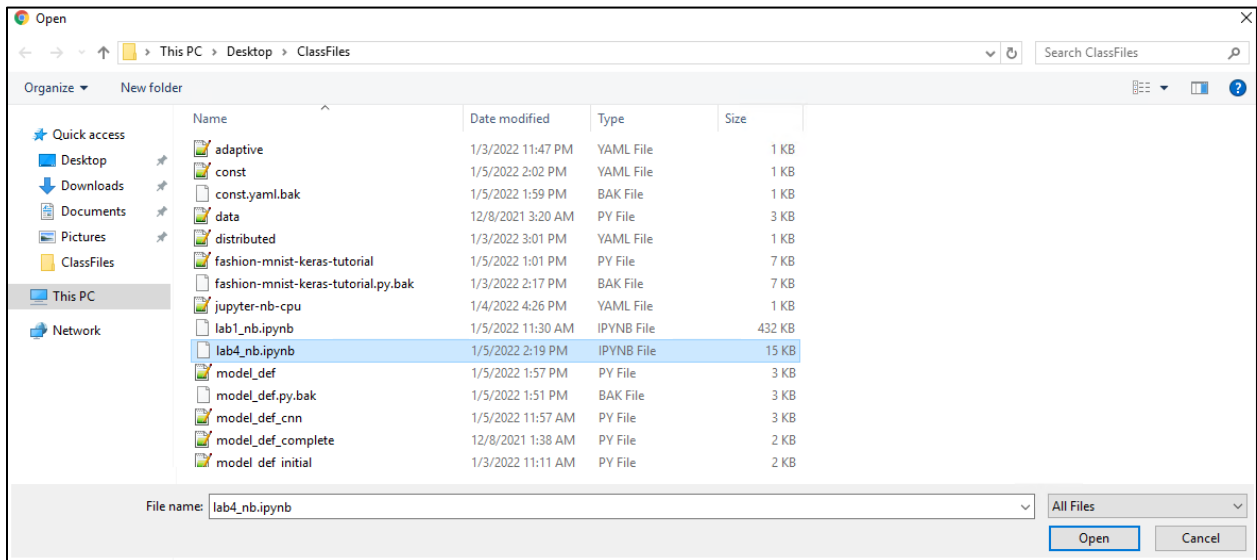
Task 4: View your trained model in action in a Jupyter Notebook

You will now return to your JupyterLab tab, open a new Notebook, and use that Notebook to try out your trained model. Like the model you explored in the Module 1 labs, this model identifies images of clothing and classifies them according to the type of clothing. The Notebook is provided to you in the ClassFiles.

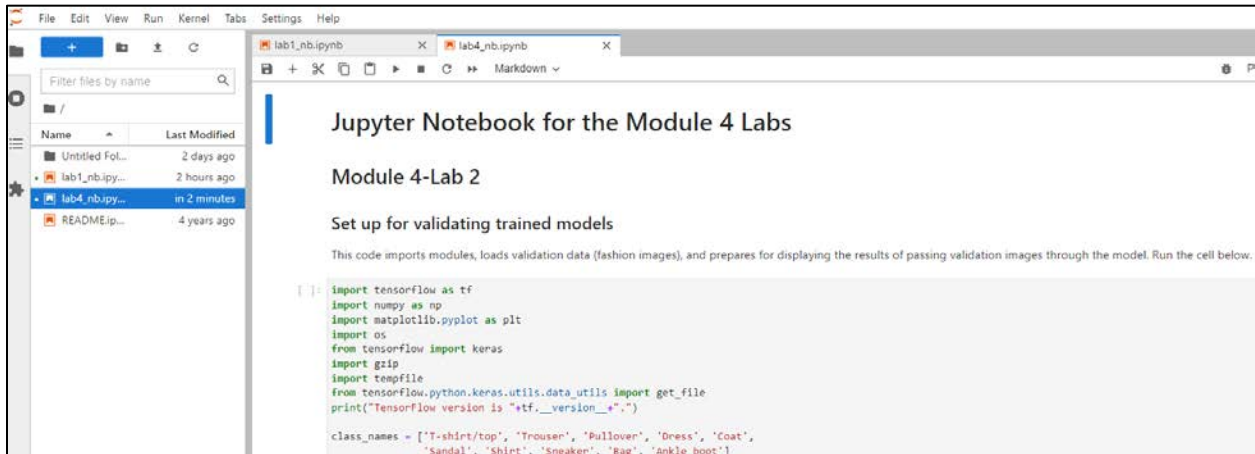
1. Return to the JupyterLab tab. (If you accidentally closed it, you can re-open it by clicking **Tasks** in the Web UI. Then click the name of your task in the list: **xx-jupyter** in which xx is your seat number.)
2. Upload the provided Notebook.
 - a. In the second to top JupyterLab ribbon, click the up arrow icon to upload a file.



- b. Browse to ClassFiles, select **lab4_nb.ipynb**, and click **Open**.



3. The Notebook will display in the left field. Double-click its name to open the Notebook.



4. Run through the Notebook, following the instructions within the Notebook. You will run three cells to:

- Set up the modules and functions used in the Notebook.
- Load your trained model using the checkpoint UUID that you collected; **make sure to leave the quotation marks around the UUID**. The example below shows an example UUID, but you must input *your* UUID.

```
const_uuid = "950d3f2b-fac7-4c33-a446-6ac38c06cc30"
```

- Run fashion images through your model for it to classify.
- **Stop when you are told that you have completed Module 4—Lab 2.**

Note

When you run the third cell, you might see the error shown below. You are running the Notebook on a CPU rather than GPU, but that is okay for the purposes of the lab. Ignore this error.

```

2022-01-12 17:23:19.691357: I tensorflow/compiler/mlir/mlir_graph_optimization_pass.cc:116] None of the MLIR optimization passes are enabled (registered 2)
2022-01-12 17:23:19.713328: I tensorflow/core/platform/profile_utils/cpu_utils.cc:112] CPU Frequency: 2100000000 Hz
    
```

5. Leave the JupyterLab tab open. You will want it for later labs.

YOU HAVE COMPLETED THIS LAB.

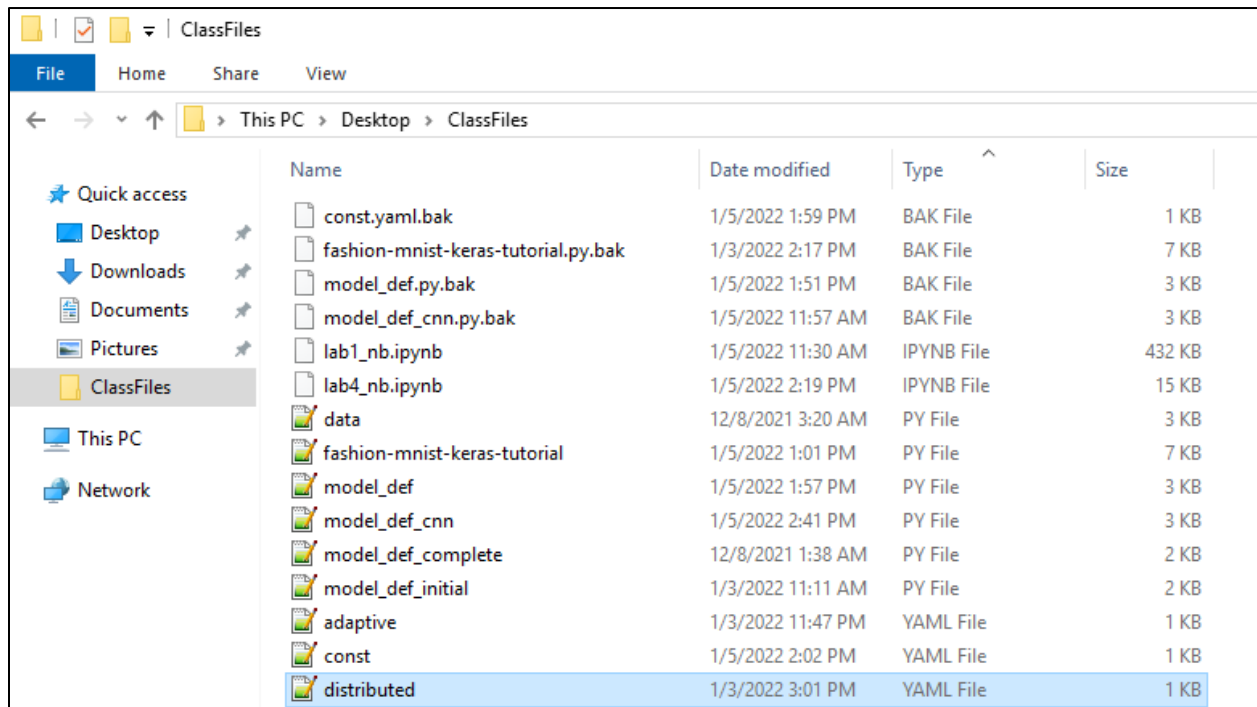
Module 4—Lab 3: Run an Experiment with a Single, Distributed Trial

In this lab, you will run an experiment that features a distributed trial. (Again this experiment will feature a single trial with a set of constant hyperparameters.)

In this scenario, the ML engineers want to improve the image classification app by using a convolutional neural network (CNN) model. They also want to train the model on more data. Anticipating the extra computational demands of these changes to the training process, the ML engineers want to distribute the trial across multiple GPUs.

Task 1: Create the experiment

1. Log into the HPE remote lab environment.
2. Open the ClassFiles folder.
3. Open the **distributed** file.



4. In the first line of the config, change XX to your seat number:

name: xx-distributed

5. As you see, this file is similar to the "const" file that you edited earlier. Because you are using new code for this experiment, it points to "model_def_cnn" for the entrypoint, and it has some additional hyperparameters. The max_length is also longer.

Note

Users can also run distributed training on the *same* code that they use for non-distributed training. You are simply trying out new code in this lab.

6. To set up distributed training, you just need to make two changes:

- Under "resources," change "slots_per_trial" to **4**.
- Under "hyperparameters," change "global_batch_size" to **128**.

```

name: XX-distributed
bind_mounts:
  - host_path: /tmp/data
    container_path: /tmp/data
    read_only: true
hyperparameters:
  global_batch_size: 128
  densel1: 128
  filters1: 32
  filters2: 32
  dropout1: .45
  dropout2: .65
resources:
  slots_per_trial: 4
  records_per_epoch: 59968
searcher:
  name: single
  metric: val_accuracy
  smaller_is_better: false
  max_length:
    epochs: 12
entrypoint: model_def_cnn:FashionMNISTTrial
min_validation_period:
  epochs: 2
    
```

7. Save and close the file.

8. Access the command prompt and make sure that you are at the correct prompt:

```
C:\Users\Administrator\Desktop\ClassFiles>
```

9. Enter this command:

```
C:\Users\Administrator\Desktop\ClassFiles> det experiment create distributed.yaml .
```

10. Make sure that the experiment is created, and leave the command prompt open.

Task 2: View the experiment

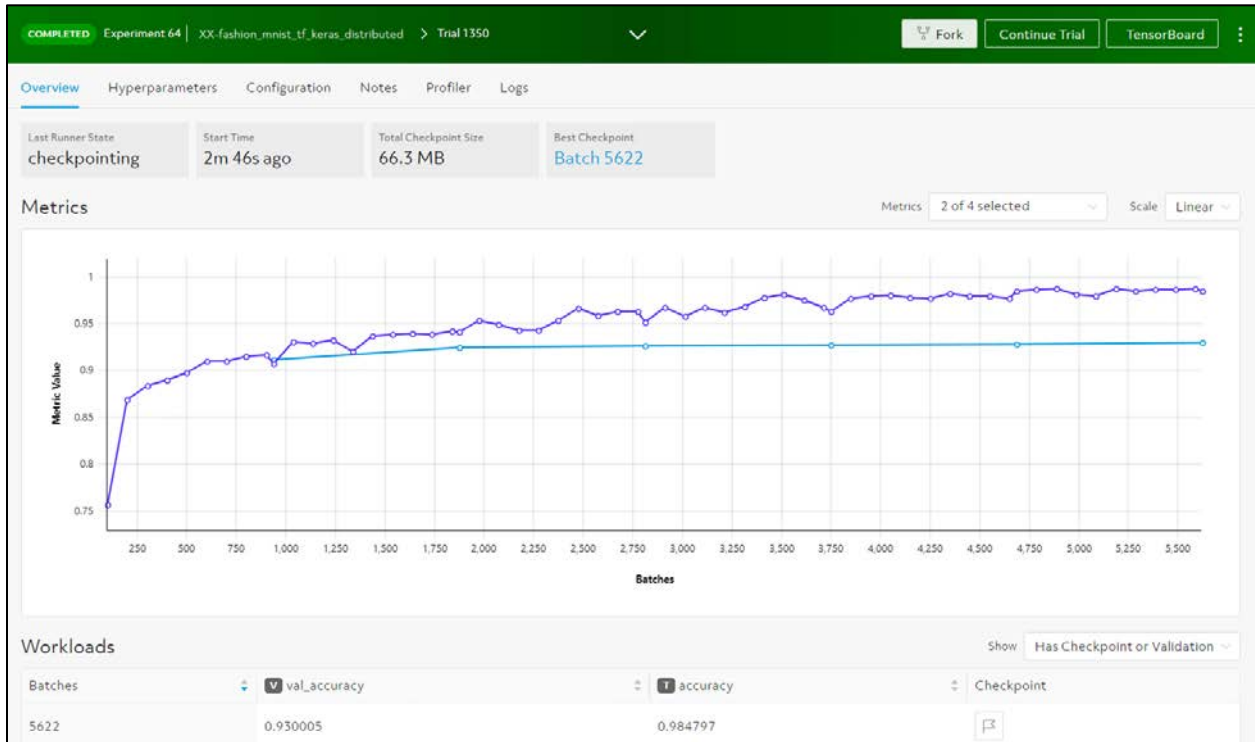
1. Open Chrome and navigate to `http://cluster.hpe.local:8080`.
2. Log in with the credentials in the Class Info file.
3. In the left navigation bar, click **Experiments**.
4. Look for your experiment in the list. You should see that the experiment has an Active state.
5. Your experiment might *not* have a trial associated with it yet. The lab cluster only has resources to run some of the distributed trials at a time. Therefore, you might need to wait 2 to 10 minutes for your experiment's turn to schedule a trial.
6. Find an experiment that does have a trial associated with it. You can watch its progress bar to see the rapid training process.

ID	Name	Labels	Start Time	Duration	Trials	State	Resource Pool	Progress	Archived	User
46	XX-distributed		20 seconds ago	18s	0	ACTIVE	lab-compute-pool			U
45	12-distributed		30 seconds ago	28s	0	ACTIVE	lab-compute-pool			U
44	14-distributed		36 seconds ago	34s	0	ACTIVE	lab-compute-pool			U
43	15-distributed		44 seconds ago	41s	0	ACTIVE	lab-compute-pool			U
42	09-distributed		52 seconds ago	49s	0	ACTIVE	lab-compute-pool			U
41	13-distributed		59 seconds ago	56s	0	ACTIVE	lab-compute-pool			U
40	07-distributed		1 minute ago	1m 3s	0	ACTIVE	lab-compute-pool			U
39	11-distributed		1 minute ago	1m 10s	1	ACTIVE	lab-compute-pool	<div style="width: 20%;"></div>		U
38	08-distributed		1 minute ago	1m 17s	1	ACTIVE	lab-compute-pool	<div style="width: 30%;"></div>		U
37	10-distributed		1 minute ago	1m 24s	0	ACTIVE	lab-compute-pool			U

7. As necessary, take a 5-10 minute break and then return to see if your experiment has completed.
8. Click your experiment's name to see the details.

ID	Name	Labels	Start Time	Duration	Trials	State	Resource Pool	Progress	Archived	User
95	XX-distributed		11 minutes ago	8m 55s	1	COMPLETED	lab-compute-pool			U
92	14-distributed		11 minutes ago	7m 3s	1	COMPLETED	lab-compute-pool			U

- You can select metrics and examine the results as you learned how to do in previous labs. In the view below, "val_accuracy" and "accuracy" are selected.



Task 3: View your trained model in action in a Jupyter Notebook

You will now return to your JupyterLab tab, open a new Notebook, and use that Notebook to try out your trained model.

- In the **Overview** tab for your experiment, scroll to the **Workloads** section.
- Find the workload associated with the best val_accuracy. In the **Checkpoint** column for that workload, click the flag.



- Copy the UUID.

Checkpoint for Batch 5622 ✕

Source Experiment 64 / Trial 1350 / Batch 5622

State **COMPLETED**

UUID **950d3f2b-fac7-4c33-a446-6ac38c06cc30**

Location file:///tmp/determined-checkpoint/950d3f2b-fac7-4c33-a446-6ac38c06cc30

4. Paste the UUID to the second line in your "myuuids" file.
5. Return to the JupyterLab tab and the "lab4_nb" Notebook.
6. Make sure that you are at the "Module 4-Lab 3" section.

Module 4-Lab 3

Load your model trained on multiple GPUs

In line 1 below, replace your_uuid with the checkpoint UUID that you collected in this lab. Make sure to leave "" around the UUID. Then run the cell.

```
[ ]: distributed_uuid = "your_uuid"
distributed_model = tf.keras.Sequential([
    tf.keras.layers.Conv2D(32, (3,3), input_shape=(28,28,1), padding="same", activation="relu"),
    tf.keras.layers.Conv2D(32, (3,3), padding="same", activation="relu"),
    tf.keras.layers.MaxPool2D(2,2),
    tf.keras.layers.Dropout(0.45),
    tf.keras.layers.Flatten(),
    tf.keras.layers.Dense(128, activation="relu"),
    tf.keras.layers.Dropout(0.65),
    tf.keras.layers.Dense(10),])
distributed_model.load_weights("/tmp/determined-checkpoint/"+distributed_uuid+"/determined-keras-model-weights").expect_pa
```

Important

You must run the Notebook in order. If for some reason you have reloaded the Notebook, or you did not complete Module 4—Lab 2, go back to the beginning of the Notebook and run the first code cell *before* you proceed to the Module 4—Lab 3 section.

7. Proceed through the Notebook, following the instructions within the Notebook. You will:

- Paste in your UUID.

Make sure to paste in *your* UUID that you copied in *this* lab (second line in the myuuids file). Also make sure to leave the quotation marks around the UUID.

```
distributed_uuid = "950d3f2b-fac7-4c33-a446-6ac38c06cc30"
```

- Run the cell to load your model.
- Run the next cell to see how well your trained model can classify images.

```
distributed_model.load_weights("/tmp/determined-checkpoint/"+distributed_uid+"/determined-keras-model-weights").expect_partial()
```

[20]: <tensorflow.python.training.tracking.util.CheckpointLoadStatus at 0x7fb738613550>

Run images through the trained model

Run the cell below to run the images through the trained model and display the results.

```
[21]: ch_test_images = np.expand_dims(test_images, axis=-1)
distributed_probability_model = tf.keras.Sequential([distributed_model,
                                                    tf.keras.layers.Softmax()])
distributed_predictions = distributed_probability_model.predict(ch_test_images)
plt.figure(figsize=(2*2*num_cols, 2*num_rows))
for i in range(start_image, end_image):
    index = i % num_images
    plt.subplot(num_rows, 2*num_cols, 2*index+1)
    plot_image(i, distributed_predictions[i], test_labels, test_images)
    plt.subplot(num_rows, 2*num_cols, 2*index+2)
    plot_value_array(i, distributed_predictions[i], test_labels)
plt.tight_layout()
plt.show()
```

- **Stop when you are told that you have completed Module 4—Lab 3.** (But leave the JupyterLab tab open for the next lab.)

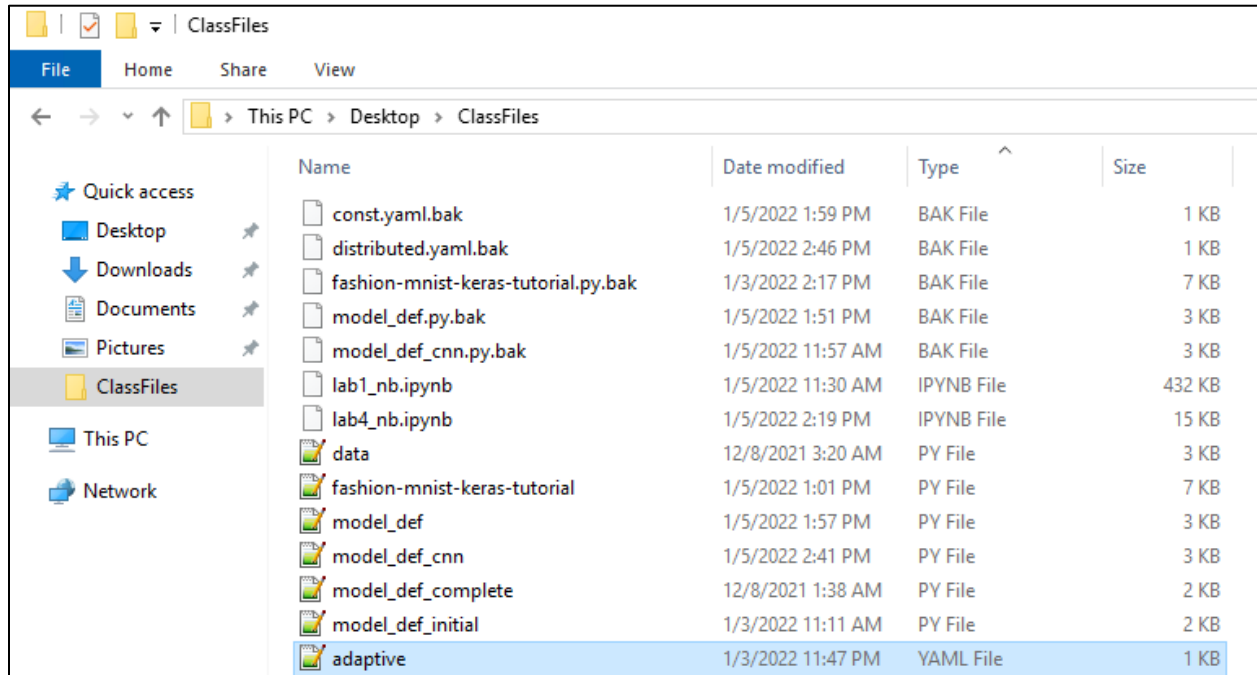
YOU HAVE COMPLETED THIS LAB.

Module 4—Lab 4: Run an Experiment that Uses Adaptive ASHA

In this lab you will run an adaptive experiment that uses the Adaptive ASHA searcher.

Task 1: Create the experiment

1. Log into the HPE remote lab environment.
2. Open the ClassFiles folder.
3. Open the **adaptive** file.



4. In the first line of the config, replace XX with your seat number:

name: XX-adaptive

5. View the searcher settings. Note that "adaptive_asha" is specified.
6. This experiment is using a relatively small number for max_trials and max_length in order to keep the training time reasonable for the lab, especially since you are sharing the environment with many other learners.
7. This experiment is using 2 for the divisor so that you can see several ASHA runs, even though you are not running many trials. (The default divisor, 4, often works better when you have a more trials.)
8. This experiment is using the default settings for Adaptive ASHA mode and max_runs. Do you remember what those are?

9. The "max_concurrent_trials" setting is currently set to 1. Think about what this means in terms of how many trials can run at once. Does this alter if your classmates are running their experiments at the same time?

10. Change "max_concurrent_trials" to 0.

11. Now how many trials can your experiment run at once? Does this alter if your classmates are running their experiments at the same time? What if some of your classmates forget to change the "max_concurrent_trials" settings. (The lab-compute-pool is using fair-share scheduling.)

12. Note that several of the hyperparameters are now variables. This experiment will find good values for these hyperparameters.

```

1  name: xx-adaptive
2  bind_mounts:
3  - host_path: /tmp/data
4    container_path: /tmp/data
5    read_only: true
6  hyperparameters:
7    global_batch_size: 32
8    filters1:
9      type: int
10     minval: 32
11     maxval: 128
12   filters2:
13     type: int
14     minval: 32
15     maxval: 128
16   dense1:
17     type: int
18     minval: 128
19     maxval: 512
20   dropout1: .45
21   dropout2: .65
22   records_per_epoch: 60000
23   searcher:
24     name: adaptive_asha
25     metric: val_accuracy
26     smaller_is_better: false
27   max_length:
28     batches: 3200
29     max_trials: 14
30     divisor: 2
31     max_concurrent_trials: 0
32   checkpoint_storage:
33     save_experiment_best: 1
34     save_trial_best: 0
35     save_trial_latest: 0
36   entrypoint: model_def_cnn:FashionMNISTTrial

```


13. Save and close the file.

14. Access the command prompt and make sure that you are at the correct prompt:

```
C:\Users\Administrator\Desktop\ClassFiles>
```

15. Enter this command:

```
C:\Users\Administrator\Desktop\ClassFiles> det preview-search adaptive.yaml
```

16. The output shows the number of trials that are trained on various amounts of data.

Important

Your output might not match the output shown in the figure exactly due to random decisions in rounding. This is okay. The text below explains how to interpret the example output shown in the figure. You can follow similar principles to interpret the output that *you* see.

In this example, the output shows:

- One ASHA process with two rungs (shown in lines 1 and 2 below)
- One ASHA process with three rungs (shown in lines 7-10 below)
- One ASHA process with four rungs (shown in lines 3-6 below)

As you see, the process with the most rungs has the most trials associated with it ($3+2+1+1 = 7$ trials). All of these trials train their models first on just 400 batches. Then about half of the trials are promoted ($2+1+1 = 4$) and continue up to 800 batches. Two trials are stopped there, while the other two ($1+1 = 2$) are promoted and continue to 1600 batches. Then the single best trial is promoted and finishes training its model on 3200 batches.

You can similarly analyze the other rungs in the output.

```
C:\Users\Administrator\Desktop\ClassFiles>det preview-search adaptive.yaml
@[33mMaster version 0.17.2 is less than CLI version 0.17.4. Consider upgrading the master.@[0m
@[32mUsing search configuration:@[0m
name: adaptive_asha
metric: val_accuracy
smaller_is_better: false
max_length:
  batches: 3200
max_trials: 14
divisor: 2
max_concurrent_trials: 1

This search will create a total of 14 trial(s).
  Trials | Breakdown
-----|-----
  2 | 1 x train 1600 batch(es)
  1 | 1 x train 1600 batch(es), 1 x train 3200 batch(es)
  3 | 1 x train 400 batch(es)
  2 | 1 x train 400 batch(es), 1 x train 800 batch(es)
  1 | 1 x train 400 batch(es), 1 x train 800 batch(es), 1 x train 1600 batch(es)
  1 | 1 x train 400 batch(es), 1 x train 800 batch(es), 1 x train 1600 batch(es), 1 x train 3200 batch(es)
  2 | 1 x train 800 batch(es)
  1 | 1 x train 800 batch(es), 1 x train 1600 batch(es)
  1 | 1 x train 800 batch(es), 1 x train 1600 batch(es), 1 x train 3200 batch(es)
C:\Users\Administrator\Desktop\ClassFiles>
```

17. Enter this command:

```
C:\Users\Administrator\Desktop\ClassFiles> det experiment create adaptive.yaml .
```

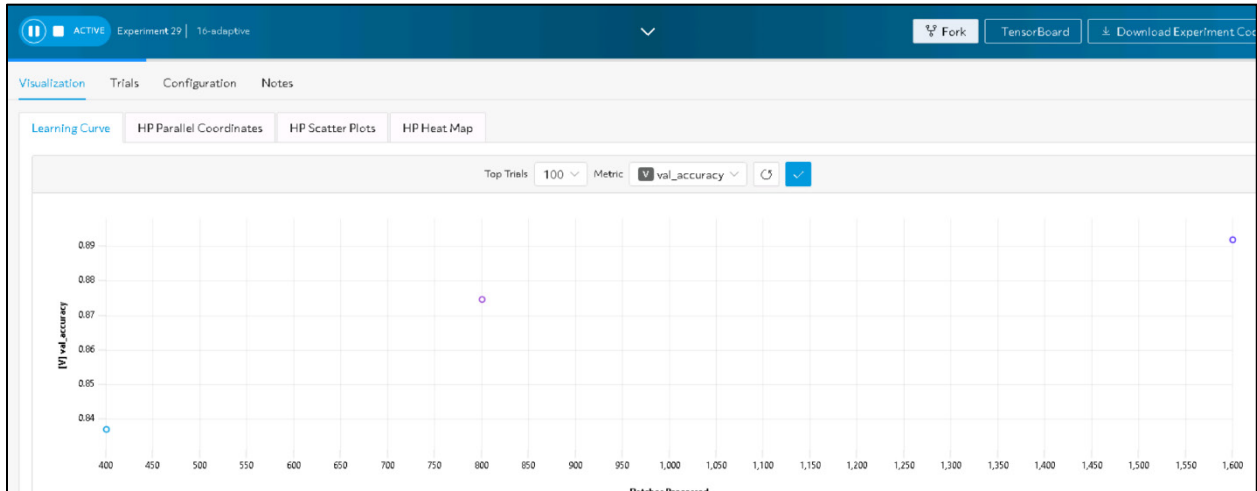
18. Verify that the experiment was created and leave the command prompt open.

Task 2: View the experiment

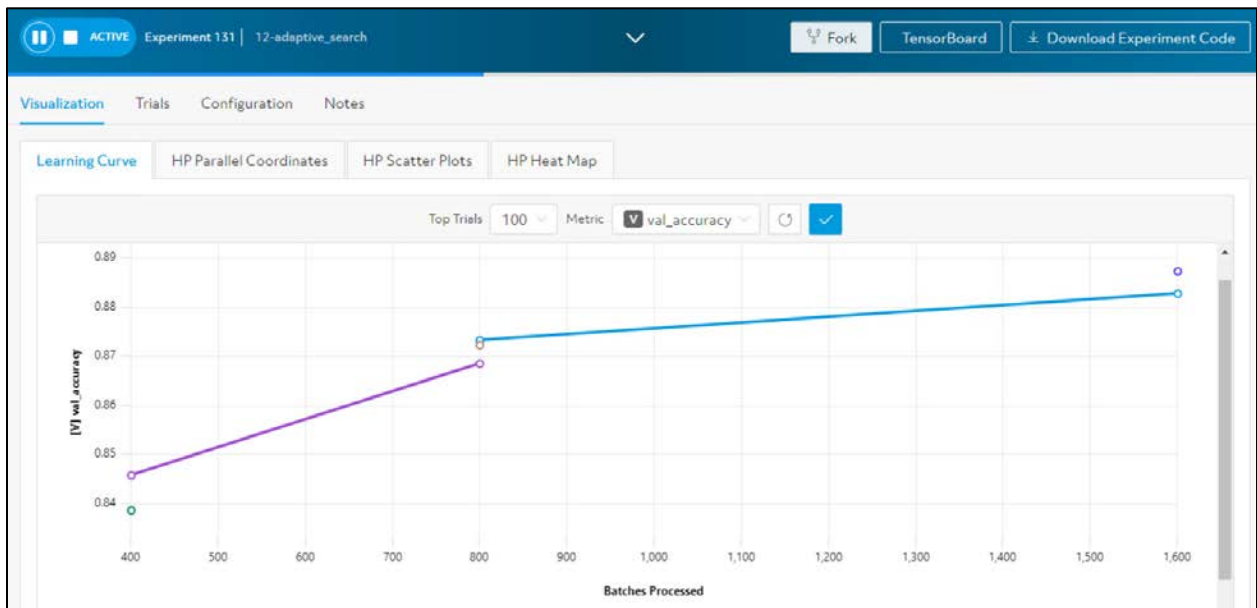
1. Access Chrome and return to the WebUI.
2. If necessary, log in with the credentials in the Class Info file.
3. In the left navigation bar, click **Experiments**.
4. Look for your experiment in the list.
 - If you see any number over 0 in the Trials column, proceed to step 5.
 - If you see 0 in the Trials column, you have more classmates running experiments than the cluster has GPUs. You will need to wait for your trials to be scheduled. Take a 5 minute break. Then return and see whether any trials have been scheduled for your experiment. If so, proceed to step 5.
5. Click the name of your experiment.

ID	Name	Labels	Start Time	Duration	Trials	State	Resource Pool	Progress	Archived	User
105	XX-adaptive		21 seconds ago	21s	1	ACTIVE	lab-compute-pool			D
104	06-adaptive		27 seconds ago	28s	1	ACTIVE	lab-compute-pool			D
103	04-adaptive		34 seconds ago	34s	1	ACTIVE	lab-compute-pool			D
102	09-adaptive		41 seconds ago	41s	1	ACTIVE	lab-compute-pool			D
101	03-adaptive		47 seconds ago	48s	1	ACTIVE	lab-compute-pool			D
100	08-adaptive		54 seconds ago	54s	2	ACTIVE	lab-compute-pool			D
99	07-adaptive		1 minute ago	1m 1s	2	ACTIVE	lab-compute-pool			D
98	01-adaptive		1 minute ago	1m 7s	3	ACTIVE	lab-compute-pool			D
97	10-adaptive		1 minute ago	1m 15s	4	ACTIVE	lab-compute-pool			D
96	12-adaptive		1 minute ago	1m 22s	6	ACTIVE	lab-compute-pool			D
95	05-adaptive		1 minute ago	1m 28s	8	ACTIVE	lab-compute-pool			D
94	02-adaptive		1 minute ago	1m 35s	8	ACTIVE	lab-compute-pool			D

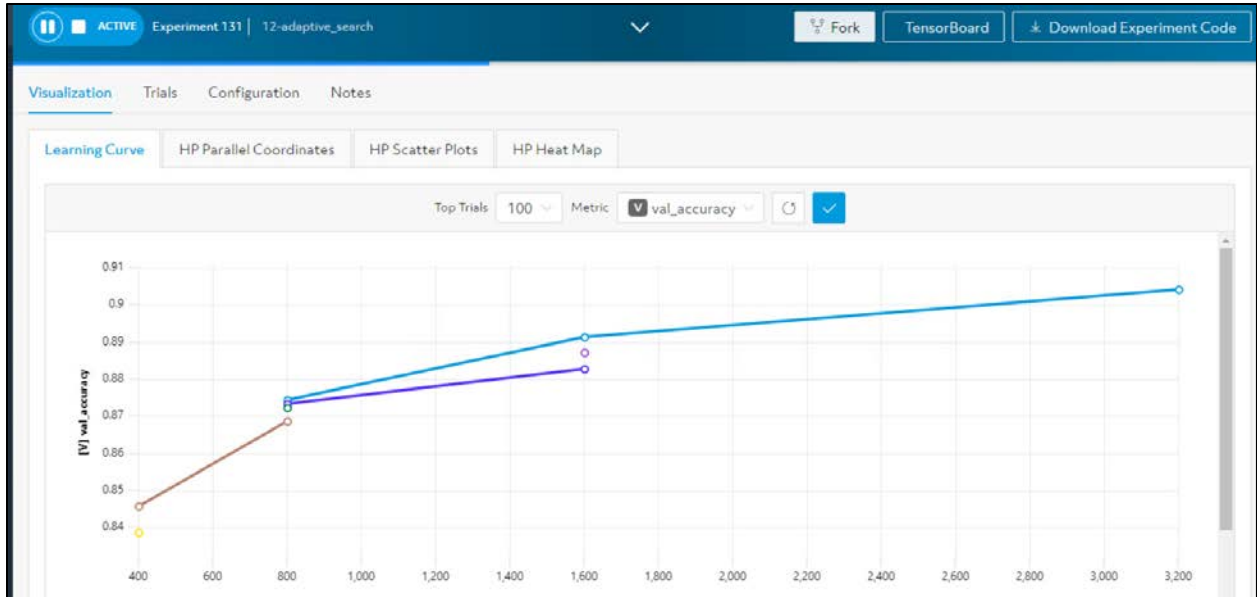
6. Make sure that you are at the **Visualization** tab. Wait a couple minutes for some data to appear.
7. The visualization graphs the validation accuracy (y-axis) versus number of batches on which the model has been trained (x-axis). Each trial has its own line; however, if the trial has only one data point, it shows as a circle. Initially, you will probably see just one or two circles.



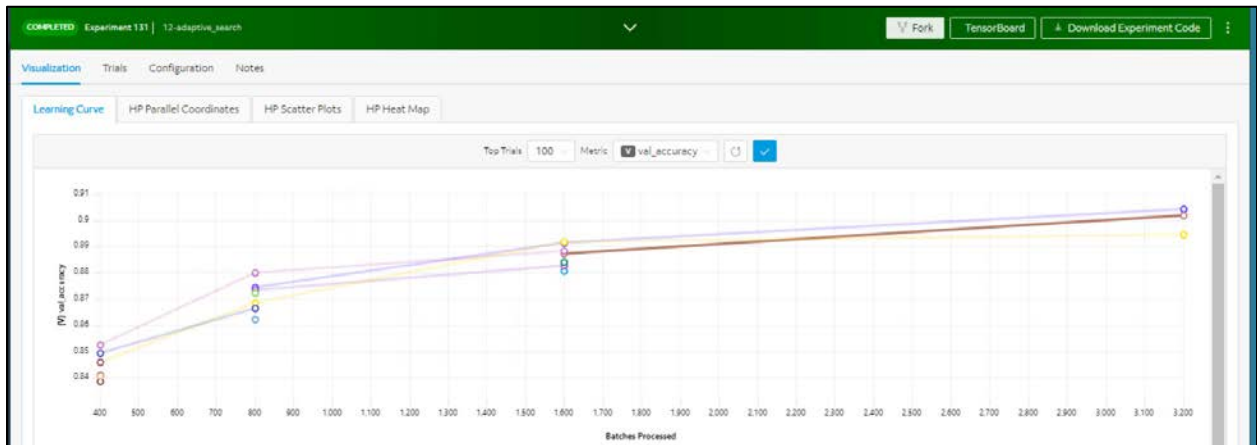
8. Check back in about 2-3 minutes. You will see more circles added as the experiment adds trials and validates them. Eventually you will see lines for some trials; Adaptive ASHA has "promoted" those trials and trained them on more data.



- Continue to check in every 2 to 5 minutes until the experiment is completed. This will probably take about 15 minutes if all of your classmates are running their experiments at the same time. Otherwise, it might finish faster.



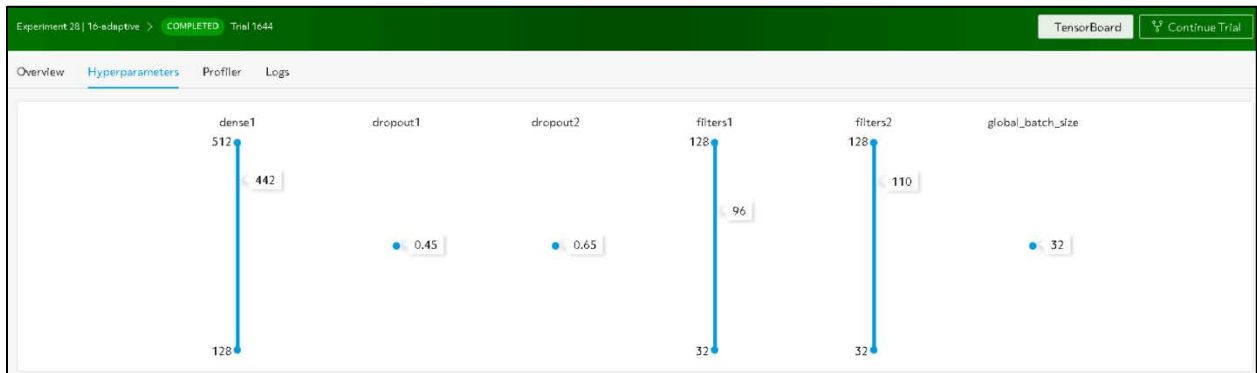
- When the experiment is complete, you should see about three trials that were trained on the full amount of data, while the other trials all stopped before that point.



11. Click the **Trials** tab.
12. Click the **Best validation metric** column to sort by that column.

ID	State	Batches	Best Validation Metric	Latest Validation Metric	StartTime	Duration	Checkpoint
Trial 1644	COMPLETED	3200	0.909500	0.909500	8 minutes ago	1m 16s	
Trial 1650	COMPLETED	3200	0.905000	0.905000	4 minutes ago	1m 20s	
Trial 1641	COMPLETED	3200	0.904300	0.904300	9 minutes ago	4m 1s	
Trial 1642	COMPLETED	3200	0.894100	0.894100	9 minutes ago	1m 6s	
Trial 1638	COMPLETED	1600	0.889800	0.889800	10 minutes ago	3m 56s	
Trial 1645	COMPLETED	1600	0.888600	0.888600	7 minutes ago	4m 38s	
Trial 1637	COMPLETED	1600	0.887700	0.887700	10 minutes ago	2m 50s	
Trial 1639	COMPLETED	800	0.863500	0.863500	10 minutes ago	6m 31s	
Trial 1646	COMPLETED	800	0.863500	0.863500	7 minutes ago	48s	
Trial 1643	COMPLETED	800	0.860300	0.860300	8 minutes ago	1m 30s	

13. Find the trial at the top and note its validation accuracy.
14. Click the name of the trial with the highest validation accuracy.
15. Click the **Hyperparameters** tab. Here you can see the hyperparameters used in this particular trial.



16. Access your **myuuids** file. Add three lines:
 - dense1:
 - filters1:
 - filters2:
17. After the colon for each of those lines, record *your* value for that hyperparameter as shown in the **Hyperparameters** tab for your trial.
18. Save the file.

To keep the experiment length manageable for the lab, you set the `max_length` rather low, so even the best trial has not trained the model long enough for the model to achieve the highest accuracy that it could. You will now continue the best trial so that you can increase the model accuracy.

19. Click the **Continue Trial** button at the top right of the page.
20. Append **-best** to the **Experiment name**.
21. Set **Max batches** to **18750**.
22. Click **Show Full Config**.

Continue Trial 3664

* Experiment name: 01-adaptive_search-best

* Max batches: 18750

Show Full Config Continue Trial

23. Change line 62, under `min_validation_period`, to **batches: 3750**
24. Click **Continue Trial**.

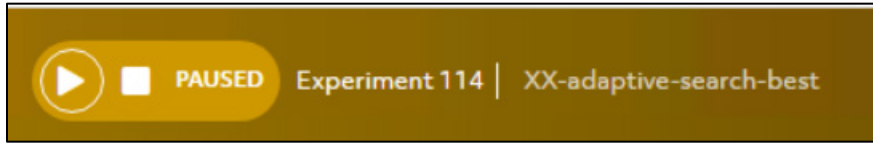
Continue Trial 3589

```

49   type: const
50   val: 76
51   filters2:
52     type: const
53     val: 51
54   global_batch_size:
55     type: const
56     val: 32
57   labels: []
58   max_restarts: 5
59   min_checkpoint_period:
60     batches: 0
61   min_validation_period:
62     batches: 3750
63   name: 12-adaptive_search-best
64   optimizations:
65     aggregation_frequency: 1
66     auto_tune_tensor_fusion: false
67     average_aggregated_gradients: true
68     average_training_metrics: false
69     grad_updates_size_file: null
    
```

Show Simple Config Continue Trial

25. The trial begins in the paused state. Click the play icon to start it.



26. Take a break for 5 or so minutes. Then check if the experiment has completed.

Task 3: View your trained model in action in a Jupyter Notebook

You will now return to your JupyterLab tab, open a new Notebook, and use that Notebook to try out your trained model.

1. You should still be in the **Overview** tab for your continued experiment.
2. Scroll to the Workloads section and find the workload with the best val_accuracy. The accuracy will probably be a bit higher than in Module 4—Lab 3. Record your accuracy:

3. In the **Checkpoint** column for the workload with the best val_accuracy, click the flag.

Workloads				Show	Has Checkpoint or Validation
Batches	val_accuracy	accuracy	Checkpoint		
5622	0.930005	0.984797			

4. Copy the UUID and paste it in a new line at the end of your myuuids file.

Checkpoint for Batch 15000 ✕

Source Experiment 67 / Trial 1364 / Batch 15000

State **COMPLETED**

UUID 0f6897c0-232d-471a-b4b7-9bb246dab112

Location file:///tmp/determined-checkpoint/0f6897c0-232d-471a-b4b7-9bb246dab112

5. Return to the JupyterLab tab and the "lab4_nb" Notebook.

6. Make sure that you are at the "Module 4-Lab 4" section.

Important

You must run the Notebook in order. If for some reason you have reloaded the Notebook, or you did not complete the previous Module 4 labs, go back to the beginning of the Notebook and run the first code cell.

Module 4-Lab 4**Load the model created with Adaptive ASHA HPO**

- In line 1 below, change `your_dense1` to the number that you recorded for the `dense1` hyperparameter.
- In line 2 below, change `your_filters1` to the number that you recorded for the `filters1` hyperparameter.
- In line 3 below, change `your_filters2` to the number that you recorded for the `filters2` hyperparameter.
- In line 4 below, replace `your_uuid` with the UUID you collected in this lab. Make sure to keep "" around the UUID.

Then run the cell.

```
[ ]: dense1 = your_dense1
filters1 = your_filters1
filters2 = your_filters2
adaptive_uuid = "your_uuid"
adaptive_model = tf.keras.Sequential([
    tf.keras.layers.Conv2D(filters1, (3,3), input_shape=(28,28,1), padding="same", activation="relu"),
    tf.keras.layers.Conv2D(filters2, (3,3), padding="same", activation="relu"),
    tf.keras.layers.MaxPool2D(2,2),
    tf.keras.layers.Dropout(0.45),
    tf.keras.layers.Flatten(),
    tf.keras.layers.Dense(dense1, activation="relu"),
    tf.keras.layers.Dropout(0.65),
    tf.keras.layers.Dense(10),])
adaptive_model.load_weights("/tmp/determined-checkpoint/"+adaptive_uuid+"/determined-keras-model-weights").expect_partial()
```

7. Proceed through the Notebook, following the instructions within the Notebook. You will:

- As instructed, replace `your_dense1`, `your_filters1`, `your_filters2`, and `your_uuid` with the values you pasted into the `myuuids` file earlier. **Be careful to leave the quotation marks (" ") around the UUID, but no quotation marks around the `dense1`, `filters1`, and `filters2` values. The figure below shows one example, but you MUST input the values that you collected earlier.**

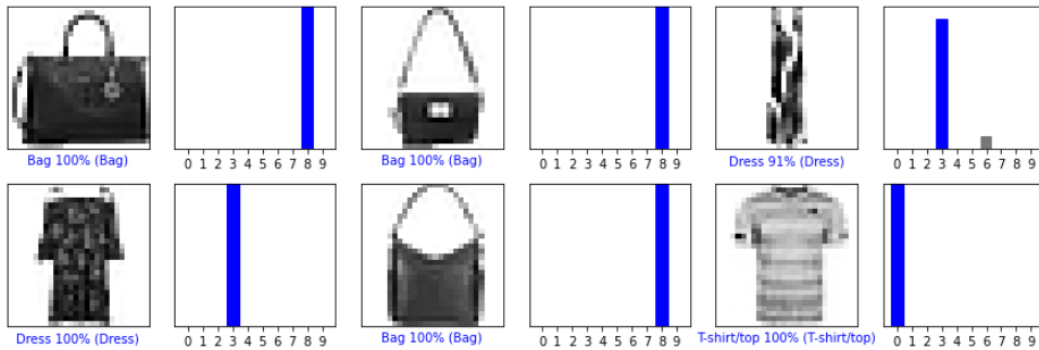
```
dense1 = 442
filters1 = 96
filters2 = 110
adaptive_uuid = "38ccce1c-8944-4ad9-a2fd-b7aab0d87aa0"
```

- Run the cell to load your model.
- Run the next cell to see how well your trained model can classify images.

Run images through the trained model

Run the cell below to run the images through the trained model and display the results.

```
[30]: ch_test_images = np.expand_dims(test_images, axis=-1)
      adaptive_probability_model = tf.keras.Sequential([adaptive_model,
      tf.keras.layers.Softmax()])
      adaptive_predictions = adaptive_probability_model.predict(ch_test_images)
      plt.figure(figsize=(2*2*num_cols, 2*num_rows))
      for i in range(start_image, end_image):
          index = i % num_images
          plt.subplot(num_rows, 2*num_cols, 2*index+1)
          plot_image(i, adaptive_predictions[i], test_labels, test_images)
          plt.subplot(num_rows, 2*num_cols, 2*index+2)
          plot_value_array(i, adaptive_predictions[i], test_labels)
      plt.tight_layout()
      plt.show()
```



- Optionally, follow the instructions for the next cells to try out your models on more images.
- Optionally, follow the instructions for the next cells to view the config for the notebook environment.

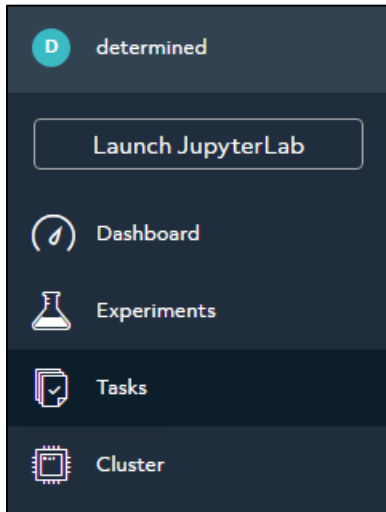
Task 4: Terminate your JupyterLab environment

You have completed the labs. In this task, you will terminate your JupyterLab environment to clean up for the end of the course.

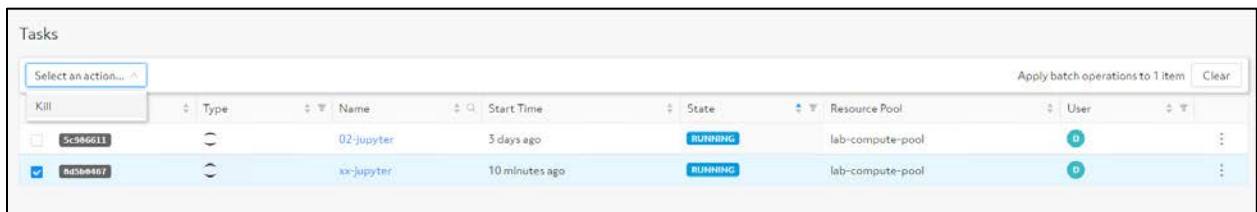
Important

Terminate your JupyterLab task before the lab session ends.

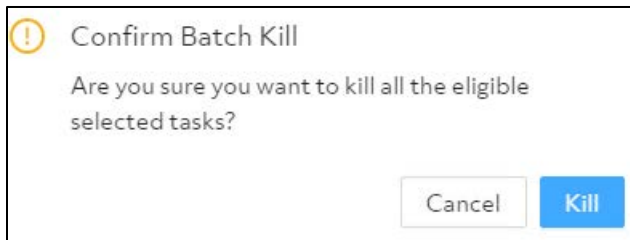
1. In the WebUI left navigation bar, click **Tasks**.



2. Select the check box next to **xx-jupyter**, in which XX is your seat number.
3. In the Select an action drop-down menu, select **Kill**.



4. In the window that pops up, click **Kill**.



5. Verify that the task terminates.



YOU HAVE COMPLETED THIS LAB



Engage with a Customer

Module 6—Activity

Activity overview

You will review a customer scenario. You will then answer several questions about how to engage with the customer and eventually size the solution.

Task 1: Read the customer scenario

You have a longstanding relationship with a healthcare organization. This organization is both a large regional healthcare provider and also supports a broad range of research initiatives. You have recently refreshed the organization’s data center with HPE ProLiant Gen 10 Plus servers and Primera storage.

In your discussions with your IT contacts, you have uncovered an additional opportunity. The senior IT manager has explained that the research department is ramping up its drug investigations with deep learning (DL) and has hired some ML engineers to help. Now researchers and ML engineers are working together on using DL to characterize new drugs and how well they are performing during trials.

But keeping up with the DL users’ demands is proving a nightmare for IT. Originally, IT assigned each user a machine with one GPU. But the department continued to hire, and the DL research team has expanded to 10 members. IT has scrambled to find more GPU-enabled machines for the new research team members.

But the team members now say that they want even more GPUs *and* that they don’t love the current set up. They want an environment that will help them use multiple GPUs for one training process. They want an environment that helps them work together and share results. The senior IT manager isn’t sure exactly what that means and expresses frustration that the requirements always seem to be changing.

Task 2: Consider discovery questions

You have asked to be put in touch with some of the members of the research team as well as the head of the research department.

1. What are some discovery questions or topics that you will plan to discuss with the research team (researchers and ML engineers)?

-
-
2. What are some discovery questions or topics that you will plan to discuss with the head of the research department?

Task 3: Read more about the customer

After meeting with research team, you have learned some additional information.

The DL users want more GPUs because training models is taking days or even weeks. They simply cannot work efficiently when they have to wait days before moving to the next step of the process. The team is looking into ways to implement data parallelization, but the ML experts on the team are *not* looking forward to adding code for that. In fact, the team members generally feel like they are spending too much time on “side tasks,” like setting up their machines to support new ML tools or scripting to manage the DL training process.

Rather than each work on their own machines, the team would like to pool GPUs so that they have more available when they need them. Ideally they would like 40 GPUs. The team is investigating scheduling software for managing the pool of GPUs; again, the team would prefer *not* to build the scheduling app themselves.

But pooling GPUs is not the only thing that the team members mean when they say that they want an environment that helps them work together. They’re also trying to figure out better ways to share data and reproduce each other’s results.

The team has thought about using the cloud to obtain more GPUs quickly. But the CIO wants to keep these particular workloads on-prem for regulatory reasons.

The head of the research department is excited about the benefits that she’s already seen from the pilot DL project. She’s willing to advocate for investing more. It does seem to her that the DL team and the projects in general are experiencing some growing pains. The team has five projects underway, and from her standpoint, it’s hard to tell why some projects are taking longer than others.

Task 4: Start to qualify the customer and size the solution

Answer these questions.

1. Is this customer in the early, active, or advanced stage for DL?

2. Is this customer qualified for HPE Machine Learning Development Environment? Is the customer qualified for HPE Machine Learning Development System? Explain your reasoning.

3. Answer these additional questions:

- Will you recommend that the customer deploy the solution in the cloud or on-prem?

- How many agents will you recommend and how many GPUs per agent?

Hint: The agents in an HPE Machine Learning Development System solution have eight GPUs each.

4. For an on-prem solution, remember that you should generally lead with HPE Machine Learning Development System. You will now practice creating a solution in HPE One Configuration Advanced.
 - a. Log into the HPE Partner Portal.
 - b. Search for and open One Configuration Advanced.
 - c. Create a new normal configuration.
 - d. In the product field, begin to type HPE Machine Learning Development System. Select this product.
 - e. Click **Create** to start the wizard.
 - f. Follow the steps in the wizard. Use these guidelines:
 - The customer’s ML engineers have indicated that their workloads require high memory and a very large number of cores.
 - The customer wants high availability for the management plane.
 - The customer does not need the optional storage nodes.
 - Use the link to the SharePoint with the cable matrix for guidance in which cable types to use. Look in the row with the number of compute nodes you are recommended.
 - Select two 42U racks and two HPE G2 Basic 3Ph 22kVA PDUs.
 - For other choices, use those recommended in the wizard.
 - g. Because you are not creating a real order, you do not need to finish the solution. Briefly review your solution, though, and take notes on the solution components.

If you do not have access to One Configuration Advanced, use what you learned in the module to take notes on the choices you would make for this customer’s HPE Machine Learning Development System solution.

Task 5: Articulate the benefits of HPE Machine Learning Development solutions

Prepare an informal presentation on the benefits of HPE Machine Learning Development solutions to this customer. Think about how these solutions help these stakeholders meet their goals and overcome their challenges:

- Senior IT manager
- Research team (researchers/ML engineers)
- Head of research department

You can take notes in the space below.

PAGE INTENTIONALLY LEFT BLANK



To learn more about HPE solutions, visit
www.hpe.com

© 2022 Hewlett Packard Enterprise Development LP. The information contained herein is subject to change without notice. The only warranties for Hewlett Packard Enterprise products and services are set forth in the express warranty statements accompanying such products and services. Nothing herein should be construed as constituting an additional warranty. Hewlett Packard Enterprise shall not be liable for technical or editorial errors or omissions contained herein.