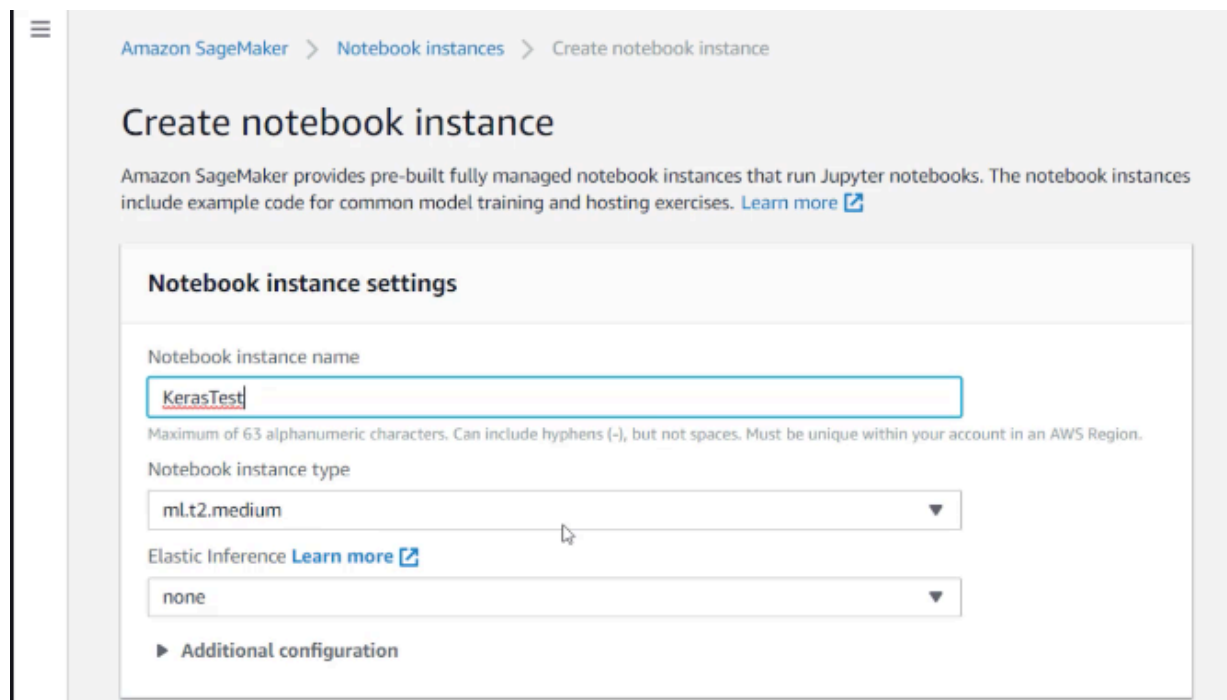


Modeling - 4 - Implementation & Operation Lab

Use SM to run our own custom model

We will use the CNN Tensorflow model on MNIST dataset from previous exercise and adapt it to run in SageMaker

1/ Create a SM Notebook instance



Amazon SageMaker > Notebook instances > Create notebook instance

Create notebook instance

Amazon SageMaker provides pre-built fully managed notebook instances that run Jupyter notebooks. The notebook instances include example code for common model training and hosting exercises. [Learn more](#)

Notebook instance settings

Notebook instance name

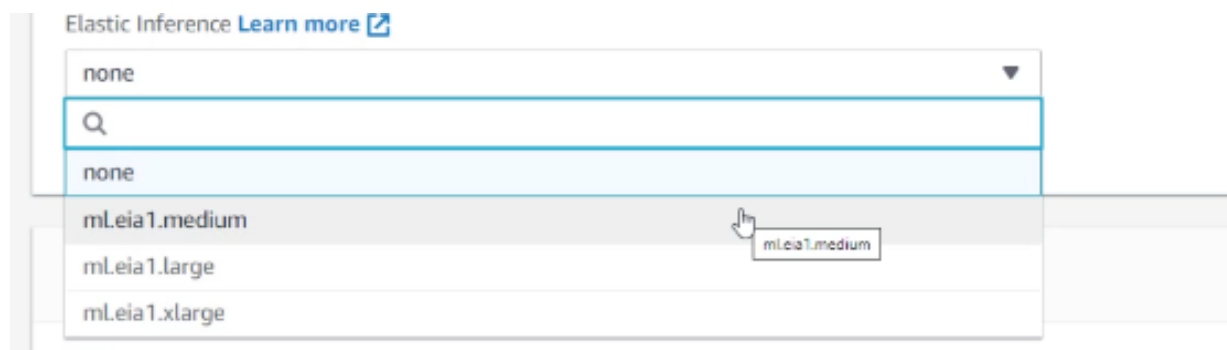
Maximum of 63 alphanumeric characters. Can include hyphens (-), but not spaces. Must be unique within your account in an AWS Region.

Notebook instance type

Elastic Inference [Learn more](#)

► Additional configuration

To note we can add Elastic Inferences to accelerate the Notebook



Elastic Inference [Learn more](#)

Q

none

mLeia1.medium

mLeia1.large

mLeia1.xlarge

Security

IAM role
Notebook instances require permissions to call other services including SageMaker and S3. Choose a role or let us create a role with the [AmazonSageMakerFullAccess](#) IAM policy attached.

Choose an IAM role ▼

Root access - optional

☒ **Enable** - Give users root access to the notebook

☐ **Disable** - Don't give users root access to the notebook
Lifecycle configurations always have root access

Encryption key - optional
Encrypt your notebook data. Choose an existing KMS key or enter a key's ARN.

No Custom Encryption ▼

► **Network - optional**

Allow IAM to create a new role
We can specify specific S3 buckets

Create an IAM role ✕

Passing an IAM role gives Amazon SageMaker permission to perform actions in other AWS services on your behalf. Creating a role here will grant permissions described by the [AmazonSageMakerFullAccess](#) IAM policy to the role you create.

The IAM role you create will provide access to:

☒ **S3 buckets you specify - optional**

☒ **Specific S3 buckets**

Example: bucket-name-1, bucket-name-2, bucket-name-3

Comma delimited. ARNs, "*" and "/" are not supported.

☐ **Any S3 bucket**
Allow users that have access to your notebook instance access to any bucket and its contents in your account.

☐ **None**

☒ Any S3 bucket with "sagemaker" in the name

☒ Any S3 object with "sagemaker" in the name

☒ Any S3 object with the tag "sagemaker" and value "true" [See Object tagging](#)

☒ S3 bucket with a Bucket Policy allowing access to SageMaker [See S3 bucket policies](#)

Cancel Create role

But for our example, we will select None and can still access it if we use "sagemaker" in the name

Encryption key - optional
 Encrypt your notebook data. Choose an existing KMS key or enter a key's ARN.

☐ No Custom Encryption

☒ No Custom Encryption

☐ Enter a KMS key ARN

We can optionally set up a private VPC (we won't do it here)

▼ Network - optional

VPC - optional
 Your notebook instance will be provided with SageMaker provided internet access because a VPC setting is not specified.

☒ No VPC

☐

☐ No VPC

Default vpc-8f2f96f5 (172.31.0.0/16)

We now have our notebook instance ready

Amazon SageMaker > Notebook instances

Notebook instances Actions ▼ Create notebook instance

	Name ▼	Instance	Creation time ▼	Status ▼	Actions
<input type="radio"/>	KerasTest	ml.t2.medium	Sep 26, 2019 14:20 UTC	InService	Open Jupyter Open JupyterLab

Open Jupiter notebook

jupyter Open JupyterLab Quit

Files Running Clusters SageMaker Examples

Select items to perform actions on them. Upload New ▼ ↺

☐ 0 ☐ /

Name ▼ Last Modified File size

The notebook list is empty.

Upload the notebook and the python file

Name	Date modified	Type	Size
Keras-CNN-Tuning.ipynb	9/25/2019 3:24 PM	Canopy Document	27 KB
keras-mnist-sagemaker.ipynb	9/26/2019 9:48 AM	Canopy Document	76 KB
mnist-train-cnn.py	9/25/2019 3:55 PM	Canopy Document	4 KB
TF-IDF.json	9/25/2019 3:55 PM	JSON File	36 KB

Type: Canopy Document
Size: 3.65 KB
Date modified: 9/25/2019 3:55 PM

jupyter

Open JupyterLab

Quit

Files Running Clusters SageMaker Examples Conda

Select items to perform actions on them

Upload New ↻

	Name	Last Modified	File size
<input type="checkbox"/>	keras-mnist-sagemaker.ipynb	seconds ago	77.6 kB
<input type="checkbox"/>	mnist-train-cnn.py	seconds ago	3.74 kB

Training and deploying our Keras CNN on SageMaker

Based on <https://aws.amazon.com/blogs/machine-learning/train-and-deploy-keras-models-with-tensorflow-and-apache-mxnet-on-amazon-sagemaker/>

We modified our Keras MNIST example from the previous exercise into the mnist-train-cnn.py script that you must upload into this Notebook's directory. It's the same code as the previous exercise, but with a little bit of extra stuff to allow hyperparameters to be passed in as arguments from SageMaker.

```
In [1]: import sagemaker

sess = sagemaker.Session()
role = sagemaker.get_execution_role()
```

Save the MNIST dataset to disk

```
In [2]: import os
import keras
import numpy as np
from keras.datasets import mnist
(x_train, y_train), (x_val, y_val) = mnist.load_data()

os.makedirs("./data", exist_ok = True)

np.savez('./data/training', image=x_train, label=y_train)
np.savez('./data/validation', image=x_val, label=y_val)

Using TensorFlow backend.

Downloading data from https://s3.amazonaws.com/img-datasets/mnist.npz
11493376/11490434 [=====] - 0s 0us/step
```

Upload MNIST data to S3

Note that sess.upload_data automatically creates an S3 bucket that meets the security criteria of starting with "sagemaker-".

```
In [3]: prefix = 'keras-mnist'

training_input_path = sess.upload_data('data/training.npz', key_prefix=prefix+'/training')
validation_input_path = sess.upload_data('data/validation.npz', key_prefix=prefix+'/validation')

print(training_input_path)
print(validation_input_path)

s3://sagemaker-us-east-1-669815420608/keras-mnist/training/training.npz
s3://sagemaker-us-east-1-669815420608/keras-mnist/validation/validation.npz
```

To note, we allowed our notebook to access s3 if the bucket includes "sagemaker" name which is the case here

“Local” Run as a Test (on the sagemaker instance)

Test out our CNN training script locally on the notebook instance

We'll test out running a single epoch, just to make sure the script works before we start spending money on P3 instances to train it further.

```
In [4]: !pygmentize mnist-train-cnn.py
```

```
In [5]: from sagemaker.tensorflow import TensorFlow

tf_estimator = TensorFlow(entry_point='mnist-train-cnn.py',
                          role=role,
                          train_instance_count=1,
                          train_instance_type='local',
                          framework_version='1.12',
                          py_version='py3',
                          script_mode=True,
                          hyperparameters={'epochs': 1})
```

To start the training, call fit()

```
In [6]: tf_estimator.fit({'training': training_input_path, 'validation': validation_input_path})
```

```
Creating tmpqust9ati_algo-1-qr56z_1 ...
Attaching to tmpqust9ati_algo-1-qr56z_12mdone
algo-1-qr56z_1 | 2019-09-26 14:34:08,225 sagemaker-containers INFO    Imported framework sagemaker_tensorflow_container.train
ing
algo-1-qr56z_1 | 2019-09-26 14:34:08,231 sagemaker-containers INFO    No GPUs detected (normal if no gpus installed)
algo-1-qr56z_1 | 2019-09-26 14:34:08,453 sagemaker-containers INFO    No GPUs detected (normal if no gpus installed)
algo-1-qr56z_1 | 2019-09-26 14:34:08,472 sagemaker-containers INFO    No GPUs detected (normal if no gpus installed)
algo-1-qr56z_1 | 2019-09-26 14:34:08,486 sagemaker-containers INFO    Invoking user script
algo-1-qr56z_1 | Training Env:
algo-1-qr56z_1 | {
algo-1-qr56z_1 |   "additional_framework_parameters": {},
algo-1-qr56z_1 |   "channel_input_dirs": {
algo-1-qr56z_1 |     "training": "/opt/ml/input/data/training",
algo-1-qr56z_1 |     "validation": "/opt/ml/input/data/validation"
algo-1-qr56z_1 |   }
algo-1-qr56z_1 | }
algo-1-qr56z_1 | Train on 60000 samples, validate on 10000 samples
algo-1-qr56z_1 | Epoch 1/1
algo-1-qr56z_1 |   - 92s - loss: 0.1974 - acc: 0.9417 - val_loss: 0.0519 - val_acc: 0.9840
algo-1-qr56z_1 | Validation loss      : 0.051937680732656734
algo-1-qr56z_1 | Validation accuracy: 0.984
algo-1-qr56z_1 | WARNING:tensorflow:From /usr/local/lib/python3.6/dist-packages/tensorflow/python/saved_model/simple_save.py:85: calling SavedModelBuilder.add_meta_graph_and_variables (from tensorflow.python.saved_model.builder_impl) with legacy_in
it_op is deprecated and will be removed in a future version.
algo-1-qr56z_1 | Instructions for updating:
algo-1-qr56z_1 | Pass your op to the equivalent parameter main_op instead.
algo-1-qr56z_1 | 2019-09-26 14:35:46,794 sagemaker-containers INFO    Reporting training SUCCESS
tmpqust9ati_algo-1-qr56z_1 exited with code 0
Aborting on container exit...
```

Validation accuracy of 98.4% with just 1 epoch - not bad

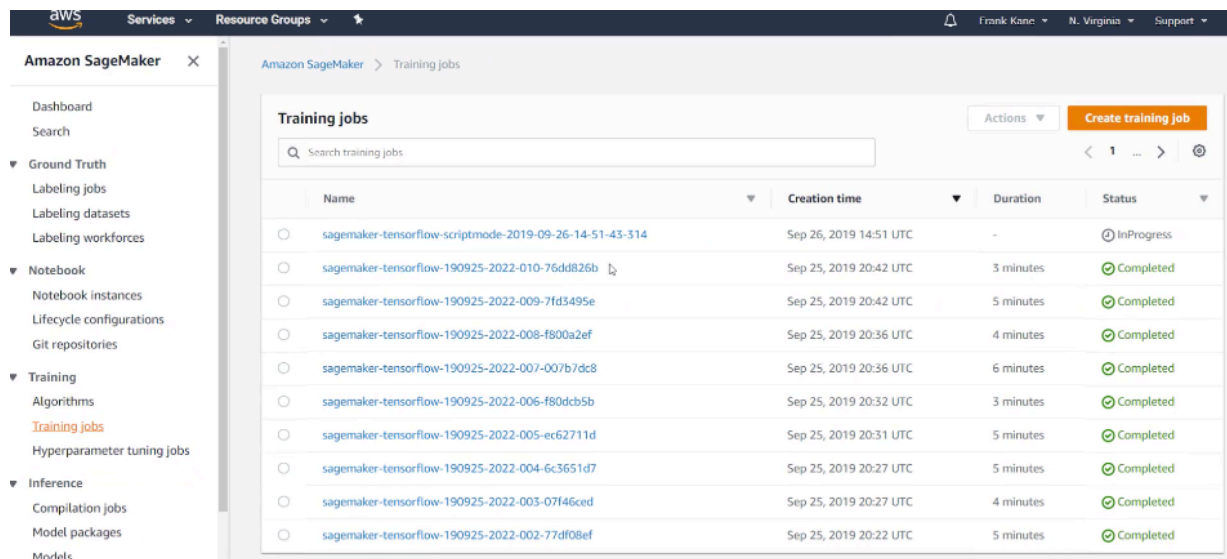
Now train this on a dedicated GPU instance and over 10 epochs
=> change instance_type and hyper parameter epoch

Train on a GPU instance with 10 epochs

Now that we're sure it works, we can start spending money on beefier servers to train our CNN over more epochs.

```
In [7]: tf_estimator = TensorFlow(entry_point='mnist-train-cnn.py',
                                role=role,
                                train_instance_count=1,
                                train_instance_type='ml.p3.2xlarge',
                                framework_version='1.12',
                                py_version='py3',
                                script_mode=True,
                                hyperparameters={
                                    'epochs': 10,
                                    'batch-size': 32,
                                    'learning-rate': 0.001
                                })
```

In the console, we can see the training jobs



The screenshot shows the Amazon SageMaker console interface. On the left is a navigation sidebar with categories like Ground Truth, Notebook, Training, and Inference. The 'Training' section is expanded, showing 'Training jobs' as the selected option. The main panel displays a table of training jobs. The table has columns for Name, Creation time, Duration, and Status. Most jobs are marked as 'Completed' with a green checkmark icon. One job is marked as 'InProgress' with a circular arrow icon.

Name	Creation time	Duration	Status
sagemaker-tensorflow-scriptmode-2019-09-26-14-51-43-314	Sep 26, 2019 14:51 UTC	-	InProgress
sagemaker-tensorflow-190925-2022-010-76dd826b	Sep 25, 2019 20:42 UTC	3 minutes	Completed
sagemaker-tensorflow-190925-2022-009-7fd3495e	Sep 25, 2019 20:42 UTC	5 minutes	Completed
sagemaker-tensorflow-190925-2022-008-f800a2ef	Sep 25, 2019 20:36 UTC	4 minutes	Completed
sagemaker-tensorflow-190925-2022-007-007b/dc8	Sep 25, 2019 20:36 UTC	6 minutes	Completed
sagemaker-tensorflow-190925-2022-006-f80dcb5b	Sep 25, 2019 20:32 UTC	3 minutes	Completed
sagemaker-tensorflow-190925-2022-005-ec62711d	Sep 25, 2019 20:31 UTC	5 minutes	Completed
sagemaker-tensorflow-190925-2022-004-6c3651d7	Sep 25, 2019 20:27 UTC	5 minutes	Completed
sagemaker-tensorflow-190925-2022-003-07f46ced	Sep 25, 2019 20:27 UTC	4 minutes	Completed
sagemaker-tensorflow-190925-2022-002-77df08ef	Sep 25, 2019 20:22 UTC	5 minutes	Completed

```
In [0]: tf_estimator.fit({'training': training_input_path, 'validation': validation_input_path})

Epoch 4/10
- 7s - loss: 0.0480 - acc: 0.9853 - val_loss: 0.0306 - val_acc: 0.9899
Epoch 5/10
- 7s - loss: 0.0407 - acc: 0.9878 - val_loss: 0.0286 - val_acc: 0.9916
Epoch 6/10
- 7s - loss: 0.0351 - acc: 0.9886 - val_loss: 0.0261 - val_acc: 0.9923
Epoch 7/10
- 7s - loss: 0.0295 - acc: 0.9903 - val_loss: 0.0255 - val_acc: 0.9925
Epoch 8/10
- 7s - loss: 0.0276 - acc: 0.9911 - val_loss: 0.0282 - val_acc: 0.9917
Epoch 9/10
- 7s - loss: 0.0242 - acc: 0.9922 - val_loss: 0.0294 - val_acc: 0.9917
Epoch 10/10
- 7s - loss: 0.0211 - acc: 0.9931 - val_loss: 0.0342 - val_acc: 0.9917
Validation loss : 0.03421558839528279
Validation accuracy: 0.9917
WARNING:tensorflow:From /usr/local/lib/python3.6/dist-packages/tensorflow/python/saved_model/simple_save.py:85: calling SavedModelBuilder.add_meta_graph_and_variables (from tensorflow.python.saved_model.builder_impl) with legacy_init_op is deprecated and will be removed in a future version.
```

Got to an accuracy of 99+=%

We can now deploy the model

We will use a CPU instance this time. And we will be adding an EI to help accelerate the inference (Elastic Inference instance)

Deploy the model

To keep costs low, we'll deploy our model to a C5 instance to make inferences. You could also use GPU instances which would be faster, but at higher cost. Note we are also using an Elastic Inference accelerator here.

```
In [10]: import time

tf_endpoint_name = 'keras-tf-mnist-'+time.strftime("%Y-%m-%d-%H-%M-%S", time.gmtime())

tf_predictor = tf_estimator.deploy(initial_instance_count=1,
                                   instance_type='ml.c5.large',
                                   accelerator_type='ml.eia1.medium',
                                   endpoint_name=tf_endpoint_name)
```

And now make predictions, selecting 5 random image and classifying them with `tf_predictor.predict()`

Make predictions with the deployed model

```
In [10]: %matplotlib inline
import random
import matplotlib.pyplot as plt

num_samples = 5
indices = random.sample(range(x_val.shape[0] - 1), num_samples)
images = x_val[indices]/255
labels = y_val[indices]

for i in range(num_samples):
    plt.subplot(1,num_samples,i+1)
    plt.imshow(images[i].reshape(28, 28), cmap='gray')
    plt.title(labels[i])
    plt.axis('off')

prediction = tf_predictor.predict(images.reshape(num_samples, 28, 28, 1))['predictions']
prediction = np.array(prediction)
predicted_label = prediction.argmax(axis=1)
print('Predicted labels are: {}'.format(predicted_label))
```

Predicted labels are: [0 6 0 9 1]



Now doing Hyper parameter Tuning with `tuner.fit()`

Find the best hyperparameters with Automatic Model Tuning

This is by far the most expensive part of this exercise; we're going to use a fair amount of time on P3 instances here. If you're worried about your AWS costs, skip the rest of this notebook and just shut down your SageMaker notebook instance now.

```
In [12]: tf_estimator = TensorFlow(entry_point='mnist-train-cnn.py',
                                   role=role,
                                   train_instance_count=1,
                                   train_instance_type='ml.p3.2xlarge',
                                   framework_version='1.12',
                                   py_version='py3',
                                   script_mode=True
                                   )
```

Test range of epochs from 5 to 20,, learning rate from 0.001 and 0.1 using a logarithmic scale...

```
In [13]: from sagemaker.tuner import IntegerParameter, CategoricalParameter, ContinuousParameter, HyperparameterTuner

hyperparameter_ranges = {
    'epochs': IntegerParameter(5, 20),
    'learning-rate': ContinuousParameter(0.0001, 0.1, scaling_type='Logarithmic'),
    'batch-size': IntegerParameter(32, 1024),
}

objective_metric_name = 'val_acc'
objective_type = 'Maximize'
metric_definitions = [{'Name': 'val_acc', 'Regex': 'val_acc: ([0-9\\.]+)'}]

tuner = HyperparameterTuner(tf_estimator,
                             objective_metric_name,
                             hyperparameter_ranges,
                             metric_definitions,
                             max_jobs=10,
                             max_parallel_jobs=2,
                             objective_type=objective_type)
```

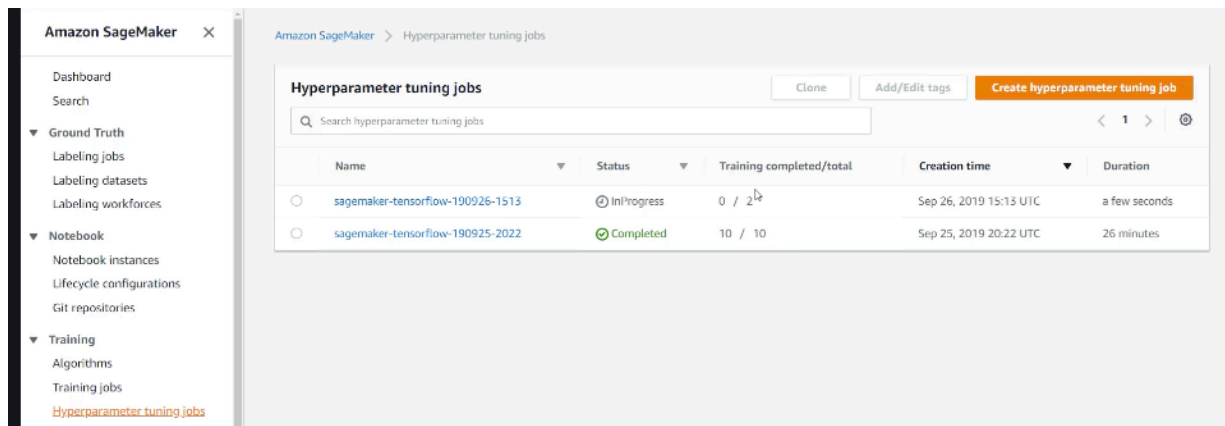
We try to optimize based on val_acc (accuracy on validation set), trying to Maximize it

max_parallel_job = 2

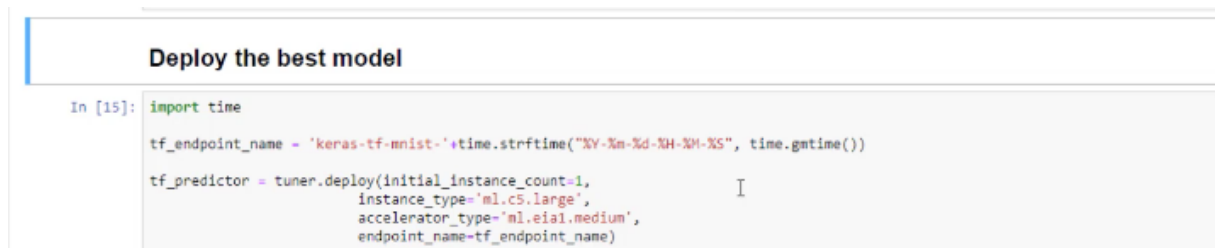
Tuning with 2 jobs at a time => because we learn from previous experiments and don't want too much parallelism

Run the tuner

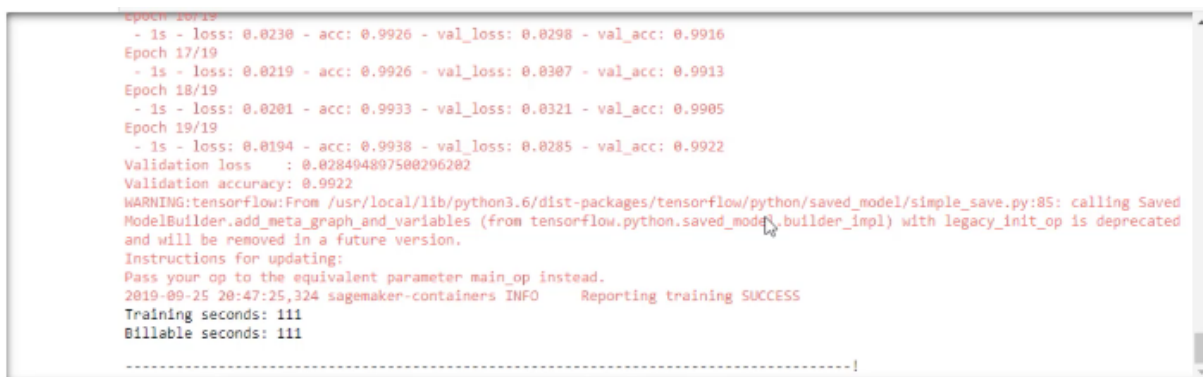
```
In [16]: tuner.fit({'training': training_input_path, 'validation': validation_input_path})
```



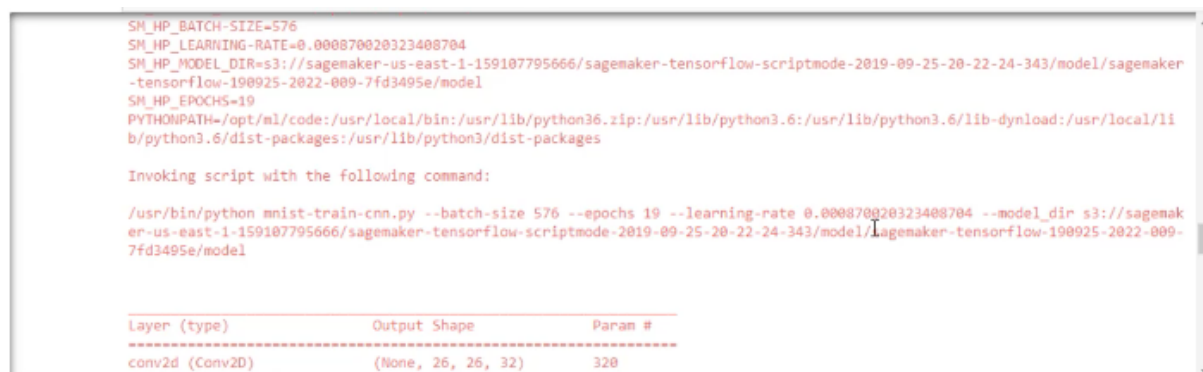
Now deploy the best model with `tuner.deploy()`



Final validation accuracy: 99.2%



Hyper parameters that is used:



Predict again

Predict again

```
In [16]: %matplotlib inline
import random
import matplotlib.pyplot as plt

num_samples = 5
indices = random.sample(range(x_val.shape[0] - 1), num_samples)
images = x_val[indices]/255
labels = y_val[indices]

for i in range(num_samples):
    plt.subplot(1,num_samples,i+1)
    plt.imshow(images[i].reshape(28, 28), cmap='gray')
    plt.title(labels[i])
    plt.axis('off')

prediction = tf_predictor.predict(images.reshape(num_samples, 28, 28, 1))['predictions']
prediction = np.array(prediction)
predicted_label = prediction.argmax(axis=1)
print('Predicted labels are: {}'.format(predicted_label))
```

Predicted labels are: [1 2 6 9 5]

