MNIST Image Classification with TensorFlow on Cloud AI Platform

This notebook demonstrates how to implement different image models on MNIST using the tf.keras API.

Learning Objectives

1. Understand how to build a Dense Neural Network (DNN) for image classification
2. Understand how to use dropout (DNN) for image classification
3. Understand how to use Convolutional Neural Networks (CNN)
4. Know how to deploy and use an image classification model using Google Cloud's AI Platform

Each learning objective will correspond to a **#TODO** in the notebook, where you will complete the notebook cell's code before running the cell. Refer to the solution notebook)for reference.

First things first. Configure the parameters below to match your own Google Cloud project details.

In [1]:
```
!sudo chown -R jupyter:jupyter /home/jupyter/training-data-analyst
```

In [13]:
```
# Here we'll show the currently installed version of TensorFlow
import tensorflow as tf
print(tf.__version__)
```

```
2.6.0
```

In [14]:
```
from datetime import datetime
import os

PROJECT = "qwiklabs-gcp-02-e914dd157b41"  # REPLACE WITH YOUR PROJECT ID
BUCKET = "qwiklabs-gcp-02-e914dd157b41"  # REPLACE WITH YOUR BUCKET NAME
REGION = "us-central1"  # REPLACE WITH YOUR BUCKET REGION e.g. us-central1
MODEL_TYPE = "cnn"  # "linear", "cnn", "dnn_dropout", or "dnn"

# Do not change these
os.environ["PROJECT"] = PROJECT
os.environ["BUCKET"] = BUCKET
os.environ["REGION"] = REGION
os.environ["MODEL_TYPE"] = MODEL_TYPE
os.environ["TFVERSION"] = "2.6"  # Tensorflow  version
os.environ["IMAGE_URI"] = os.path.join("gcr.io", PROJECT, "mnist_models")
```

Building a dynamic model

In the previous notebook, 1_mnist_linear.ipynb, we ran our code directly from the notebook. In order to run it on the AI Platform, it needs to be packaged as a python module.

The boilerplate structure for this module has already been set up in the folder `mnist_models`. The module lives in the sub-folder, `trainer`, and is designated as a python package with the empty `__init__.py` (`mnist_models/trainer/__init__.py`) file. It still needs the model and a trainer to run it, so let's make them.

Let's start with the trainer file first. This file parses command line arguments to feed into the model.

In [15]:

```
%%writefile mnist_models/trainer/task.py
import argparse
import json
import os
import sys

from . import model


def _parse_arguments(argv):
    """Parses command-line arguments."""
    parser = argparse.ArgumentParser()
    parser.add_argument(
        '--model_type',
        help='Which model type to use',
        type=str, default='linear')
    parser.add_argument(
        '--epochs',
        help='The number of epochs to train',
        type=int, default=10)
    parser.add_argument(
        '--steps_per_epoch',
        help='The number of steps per epoch to train',
        type=int, default=100)
    parser.add_argument(
        '--job-dir',
        help='Directory where to save the given model',
        type=str, default='mnist_models/')
    return parser.parse_known_args(argv)


def main():
    """Parses command line arguments and kicks off model training."""
    args = _parse_arguments(sys.argv[1:])[0]

    # Configure path for hyperparameter tuning.
    trial_id = json.loads(
        os.environ.get('TF_CONFIG', '{}')).get('task', {}).get('trial', '')
    output_path = args.job_dir if not trial_id else args.job_dir + '/'

    model_layers = model.get_layers(args.model_type)
    image_model = model.build_model(model_layers, args.job_dir)
    model_history = model.train_and_evaluate(
        image_model, args.epochs, args.steps_per_epoch, args.job_dir)
```

```
if __name__ == '__main__':
    main()
```

```
Overwriting mnist_models/trainer/task.py
```

Next, let's group non-model functions into a util file to keep the model file simple. We'll copy over the `scale` and `load_dataset` functions from the previous lab.

In [16]:

```
%%writefile mnist_models/trainer/util.py
import tensorflow as tf


def scale(image, label):
    """Scales images from a 0-255 int range to a 0-1 float range"""
    image = tf.cast(image, tf.float32)
    image /= 255
    image = tf.expand_dims(image, -1)
    return image, label


def load_dataset(
        data, training=True, buffer_size=5000, batch_size=100, nclasses=10):
    """Loads MNIST dataset into a tf.data.Dataset"""
    (x_train, y_train), (x_test, y_test) = data
    x = x_train if training else x_test
    y = y_train if training else y_test
    # One-hot encode the classes
    y = tf.keras.utils.to_categorical(y, nclasses)
    dataset = tf.data.Dataset.from_tensor_slices((x, y))
    dataset = dataset.map(scale).batch(batch_size)
    if training:
        dataset = dataset.shuffle(buffer_size).repeat()
    return dataset
```

```
Overwriting mnist_models/trainer/util.py
```

Finally, let's code the models! The tf.keras API accepts an array of layers into a model object, so we can create a dictionary of layers based on the different model types we want to use. The below file has two functions: `get_layers` and `create_and_train_model`. We will build the structure of our model in `get_layers`. Last but not least, we'll copy over the training code from the previous lab into `train_and_evaluate`.

TODO 1: Define the Keras layers for a DNN model
TODO 2: Define the Keras layers for a dropout model
TODO 3: Define the Keras layers for a CNN model

Hint: These models progressively build on each other. Look at the imported `tensorflow.keras.layers` modules and the default values for the variables defined in `get_layers` for guidance.

In [21]:

```
%%writefile mnist_models/trainer/model.py
import os
import shutil

import matplotlib.pyplot as plt
import numpy as np
```

```
import tensorflow as tf
from tensorflow.keras import Sequential
from tensorflow.keras.callbacks import TensorBoard
from tensorflow.keras.layers import (
    Conv2D, Dense, Dropout, Flatten, MaxPooling2D, Softmax)

from . import util


# Image Variables
WIDTH = 28
HEIGHT = 28


def get_layers(
        model_type,
        nclasses=10,
        hidden_layer_1_neurons=400,
        hidden_layer_2_neurons=100,
        dropout_rate=0.25,
        num_filters_1=64,
        kernel_size_1=3,
        pooling_size_1=2,
        num_filters_2=32,
        kernel_size_2=3,
        pooling_size_2=2):
    """Constructs layers for a keras model based on a dict of model types."""
    model_layers = {
        'linear': [
            Flatten(),
            Dense(nclasses),
            Softmax()
        ],
        'dnn': [
            Flatten(),
            Dense(hidden_layer_1_neurons, activation='relu'),
            Dense(hidden_layer_2_neurons, activation='relu'),
            Dense(nclasses),
            Softmax()
        ],
        'dnn_dropout': [
            Flatten(),
            Dense(hidden_layer_1_neurons, activation='relu'),
            Dense(hidden_layer_2_neurons, activation='relu'),
            Dropout(dropout_rate),
            Dense(nclasses),
            Softmax()
                    ],
        'cnn': [
            Conv2D(num_filters_1, kernel_size=kernel_size_1,
                   activation='relu', input_shape=(WIDTH, HEIGHT, 1)),
            MaxPooling2D(pooling_size_1),
            Conv2D(num_filters_2, kernel_size=kernel_size_2,
                   activation='relu'),
            MaxPooling2D(pooling_size_2),
            Flatten(),
            Dense(hidden_layer_1_neurons, activation='relu'),
            Dense(hidden_layer_2_neurons, activation='relu'),
            Dropout(dropout_rate),
            Dense(nclasses),
            Softmax()
```

```
        ]
    }
    return model_layers[model_type]


def build_model(layers, output_dir):
    """Compiles keras model for image classification."""
    model = Sequential(layers)
    model.compile(optimizer='adam',
                  loss='categorical_crossentropy',
                  metrics=['accuracy'])
    return model


def train_and_evaluate(model, num_epochs, steps_per_epoch, output_dir):
    """Compiles keras model and loads data into it for training."""
    mnist = tf.keras.datasets.mnist.load_data()
    train_data = util.load_dataset(mnist)
    validation_data = util.load_dataset(mnist, training=False)

    callbacks = []
    if output_dir:
        tensorboard_callback = TensorBoard(log_dir=output_dir)
        callbacks = [tensorboard_callback]

    history = model.fit(
        train_data,
        validation_data=validation_data,
        epochs=num_epochs,
        steps_per_epoch=steps_per_epoch,
        verbose=2,
        callbacks=callbacks)

    if output_dir:
        export_path = os.path.join(output_dir, 'keras_export')
        model.save(export_path, save_format='tf')

    return history
```

```
Overwriting mnist_models/trainer/model.py
```

Local Training

With everything set up, let's run locally to test the code. Some of the previous tests have been copied over into a testing script `mnist_models/trainer/test.py` to make sure the model still passes our previous checks. On `line 13`, you can specify which model types you would like to check. `line 14` and `line 15` has the number of epochs and steps per epoch respectively.

Moment of truth! Run the code below to check your models against the unit tests. If you see "OK" at the end when it's finished running, congrats! You've passed the tests!

In [22]:

```
!python3 -m mnist_models.trainer.test
```

```
2021-09-29 17:46:02.047772: I tensorflow/core/common_runtime/process_util.cc:14
6] Creating new thread pool with default inter op setting: 2. Tune using inter_o
```

```
p_parallelism_threads for best performance.
2021-09-29 17:46:02.128285: I tensorflow/compiler/mlir/mlir_graph_optimization_p
ass.cc:185] None of the MLIR Optimization Passes are enabled (registered 2)
..
*** Building model for linear ***

Epoch 1/10
100/100 - 5s - loss: 1.3079 - accuracy: 0.6750 - val_loss: 0.7823 - val_accurac
y: 0.8346
Epoch 2/10
100/100 - 1s - loss: 0.6558 - accuracy: 0.8526 - val_loss: 0.5525 - val_accurac
y: 0.8723
Epoch 3/10
100/100 - 1s - loss: 0.5187 - accuracy: 0.8733 - val_loss: 0.4645 - val_accurac
y: 0.8855
Epoch 4/10
100/100 - 1s - loss: 0.4428 - accuracy: 0.8889 - val_loss: 0.4169 - val_accurac
y: 0.8954
Epoch 5/10
100/100 - 2s - loss: 0.4267 - accuracy: 0.8837 - val_loss: 0.3847 - val_accurac
y: 0.8990
Epoch 6/10
100/100 - 2s - loss: 0.4033 - accuracy: 0.8931 - val_loss: 0.3647 - val_accurac
y: 0.9058
Epoch 7/10
100/100 - 2s - loss: 0.3770 - accuracy: 0.8989 - val_loss: 0.3490 - val_accurac
y: 0.9075
Epoch 8/10
100/100 - 1s - loss: 0.3543 - accuracy: 0.9050 - val_loss: 0.3371 - val_accurac
y: 0.9087
Epoch 9/10
100/100 - 1s - loss: 0.3547 - accuracy: 0.9026 - val_loss: 0.3284 - val_accurac
y: 0.9112
Epoch 10/10
100/100 - 2s - loss: 0.3482 - accuracy: 0.9041 - val_loss: 0.3261 - val_accurac
y: 0.9122

*** Building model for dnn ***

Epoch 1/10
100/100 - 5s - loss: 0.6031 - accuracy: 0.8256 - val_loss: 0.2815 - val_accurac
y: 0.9119
Epoch 2/10
100/100 - 2s - loss: 0.2589 - accuracy: 0.9242 - val_loss: 0.1914 - val_accurac
y: 0.9421
Epoch 3/10
100/100 - 2s - loss: 0.1890 - accuracy: 0.9446 - val_loss: 0.1818 - val_accurac
y: 0.9438
Epoch 4/10
100/100 - 2s - loss: 0.1682 - accuracy: 0.9479 - val_loss: 0.1439 - val_accurac
y: 0.9552
Epoch 5/10
100/100 - 1s - loss: 0.1434 - accuracy: 0.9604 - val_loss: 0.1351 - val_accurac
y: 0.9586
Epoch 6/10
100/100 - 2s - loss: 0.1476 - accuracy: 0.9555 - val_loss: 0.1128 - val_accurac
y: 0.9652
Epoch 7/10
100/100 - 1s - loss: 0.1061 - accuracy: 0.9678 - val_loss: 0.1132 - val_accurac
y: 0.9636
Epoch 8/10
```

```
100/100 - 1s - loss: 0.0987 - accuracy: 0.9715 - val_loss: 0.0944 - val_accurac
y: 0.9696
Epoch 9/10
100/100 - 1s - loss: 0.0888 - accuracy: 0.9730 - val_loss: 0.1169 - val_accurac
y: 0.9641
Epoch 10/10
100/100 - 1s - loss: 0.1040 - accuracy: 0.9690 - val_loss: 0.0945 - val_accurac
y: 0.9707

*** Building model for dnn_dropout ***

Epoch 1/10
100/100 - 5s - loss: 0.6661 - accuracy: 0.8030 - val_loss: 0.2774 - val_accurac
y: 0.9195
Epoch 2/10
100/100 - 1s - loss: 0.3203 - accuracy: 0.9064 - val_loss: 0.2158 - val_accurac
y: 0.9371
Epoch 3/10
100/100 - 2s - loss: 0.2357 - accuracy: 0.9313 - val_loss: 0.1702 - val_accurac
y: 0.9497
Epoch 4/10
100/100 - 1s - loss: 0.1894 - accuracy: 0.9456 - val_loss: 0.1711 - val_accurac
y: 0.9482
Epoch 5/10
100/100 - 1s - loss: 0.1811 - accuracy: 0.9488 - val_loss: 0.1347 - val_accurac
y: 0.9597
Epoch 6/10
100/100 - 1s - loss: 0.1628 - accuracy: 0.9508 - val_loss: 0.1181 - val_accurac
y: 0.9648
Epoch 7/10
100/100 - 2s - loss: 0.1140 - accuracy: 0.9661 - val_loss: 0.1158 - val_accurac
y: 0.9623
Epoch 8/10
100/100 - 1s - loss: 0.1140 - accuracy: 0.9669 - val_loss: 0.1031 - val_accurac
y: 0.9664
Epoch 9/10
100/100 - 1s - loss: 0.1203 - accuracy: 0.9613 - val_loss: 0.1054 - val_accurac
y: 0.9687
Epoch 10/10
100/100 - 1s - loss: 0.1121 - accuracy: 0.9671 - val_loss: 0.0974 - val_accurac
y: 0.9698

*** Building model for cnn ***

Epoch 1/10
100/100 - 11s - loss: 0.7101 - accuracy: 0.7699 - val_loss: 0.1707 - val_accurac
y: 0.9477
Epoch 2/10
100/100 - 7s - loss: 0.1877 - accuracy: 0.9414 - val_loss: 0.1442 - val_accurac
y: 0.9583
Epoch 3/10
100/100 - 7s - loss: 0.1298 - accuracy: 0.9635 - val_loss: 0.0855 - val_accurac
y: 0.9724
Epoch 4/10
100/100 - 7s - loss: 0.0953 - accuracy: 0.9726 - val_loss: 0.0588 - val_accurac
y: 0.9817
Epoch 5/10
100/100 - 7s - loss: 0.0961 - accuracy: 0.9715 - val_loss: 0.0552 - val_accurac
y: 0.9819
Epoch 6/10
100/100 - 7s - loss: 0.0736 - accuracy: 0.9767 - val_loss: 0.0499 - val_accurac
```

```
y: 0.9834
Epoch 7/10
100/100 - 7s - loss: 0.0679 - accuracy: 0.9789 - val_loss: 0.0627 - val_accurac
y: 0.9787
Epoch 8/10
100/100 - 8s - loss: 0.0641 - accuracy: 0.9817 - val_loss: 0.0420 - val_accurac
y: 0.9861
Epoch 9/10
100/100 - 8s - loss: 0.0570 - accuracy: 0.9824 - val_loss: 0.0372 - val_accurac
y: 0.9879
Epoch 10/10
100/100 - 7s - loss: 0.0556 - accuracy: 0.9846 - val_loss: 0.0375 - val_accurac
y: 0.9875
...
----------------------------------------------------------------------
Ran 5 tests in 139.239s

OK
```

Now that we know that our models are working as expected, let's run it on the Google Cloud AI Platform. We can run it as a python module locally first using the command line.

The below cell transfers some of our variables to the command line as well as create a job directory including a timestamp.

In [23]:

```
current_time = datetime.now().strftime("%y%m%d_%H%M%S")
model_type = 'cnn' # "linear", "cnn", "dnn_dropout", or "dnn"

os.environ["MODEL_TYPE"] = model_type
os.environ["JOB_DIR"] = "mnist_models/models/{}_{}/".format(
    model_type, current_time)
```

The cell below runs the local version of the code. The epochs and steps_per_epoch flag can be changed to run for longer or shorter, as defined in our `mnist_models/trainer/task.py` file.

In [24]:

```
%%bash
python3 -m mnist_models.trainer.task \
    --job-dir=$JOB_DIR \
    --epochs=5 \
    --steps_per_epoch=50 \
    --model_type=$MODEL_TYPE
```

```
Epoch 1/5
50/50 - 9s - loss: 0.9974 - accuracy: 0.6850 - val_loss: 0.3174 - val_accuracy:
0.9096
Epoch 2/5
50/50 - 5s - loss: 0.3411 - accuracy: 0.8918 - val_loss: 0.1870 - val_accuracy:
0.9415
Epoch 3/5
50/50 - 4s - loss: 0.2117 - accuracy: 0.9382 - val_loss: 0.1254 - val_accuracy:
0.9620
Epoch 4/5
50/50 - 5s - loss: 0.1609 - accuracy: 0.9478 - val_loss: 0.1041 - val_accuracy:
0.9680
Epoch 5/5
```

```
50/50 - 4s - loss: 0.1329 - accuracy: 0.9622 - val_loss: 0.0819 - val_accuracy:
0.9740
2021-09-29 17:48:45.116779: I tensorflow/core/common_runtime/process_util.cc:14
6] Creating new thread pool with default inter op setting: 2. Tune using inter_o
p_parallelism_threads for best performance.
2021-09-29 17:48:45.575441: I tensorflow/core/profiler/lib/profiler_session.cc:1
31] Profiler session initializing.
2021-09-29 17:48:45.575487: I tensorflow/core/profiler/lib/profiler_session.cc:1
46] Profiler session started.
2021-09-29 17:48:45.576288: I tensorflow/core/profiler/lib/profiler_session.cc:1
64] Profiler session tear down.
2021-09-29 17:48:46.275760: I tensorflow/compiler/mlir/mlir_graph_optimization_p
ass.cc:185] None of the MLIR Optimization Passes are enabled (registered 2)
2021-09-29 17:48:49.766089: I tensorflow/core/profiler/lib/profiler_session.cc:1
31] Profiler session initializing.
2021-09-29 17:48:49.766145: I tensorflow/core/profiler/lib/profiler_session.cc:1
46] Profiler session started.
2021-09-29 17:48:49.825858: I tensorflow/core/profiler/lib/profiler_session.cc:6
6] Profiler session collecting data.
2021-09-29 17:48:49.834998: I tensorflow/core/profiler/lib/profiler_session.cc:1
64] Profiler session tear down.
2021-09-29 17:48:49.854250: I tensorflow/core/profiler/rpc/client/save_profile.c
c:136] Creating directory: mnist_models/models/cnn_210929_174838/train/plugins/p
rofile/2021_09_29_17_48_49

2021-09-29 17:48:49.863942: I tensorflow/core/profiler/rpc/client/save_profile.c
c:142] Dumped gzipped tool data for trace.json.gz to mnist_models/models/cnn_210
929_174838/train/plugins/profile/2021_09_29_17_48_49/tensorflow-2-6-20210929-131
042.trace.json.gz
2021-09-29 17:48:49.882437: I tensorflow/core/profiler/rpc/client/save_profile.c
c:136] Creating directory: mnist_models/models/cnn_210929_174838/train/plugins/p
rofile/2021_09_29_17_48_49

2021-09-29 17:48:49.883659: I tensorflow/core/profiler/rpc/client/save_profile.c
c:142] Dumped gzipped tool data for memory_profile.json.gz to mnist_models/model
s/cnn_210929_174838/train/plugins/profile/2021_09_29_17_48_49/tensorflow-2-6-202
10929-131042.memory_profile.json.gz
2021-09-29 17:48:49.885166: I tensorflow/core/profiler/rpc/client/capture_profil
e.cc:251] Creating directory: mnist_models/models/cnn_210929_174838/train/plugin
s/profile/2021_09_29_17_48_49
Dumped tool data for xplane.pb to mnist_models/models/cnn_210929_174838/train/pl
ugins/profile/2021_09_29_17_48_49/tensorflow-2-6-20210929-131042.xplane.pb
Dumped tool data for overview_page.pb to mnist_models/models/cnn_210929_174838/t
rain/plugins/profile/2021_09_29_17_48_49/tensorflow-2-6-20210929-131042.overview
_page.pb
Dumped tool data for input_pipeline.pb to mnist_models/models/cnn_210929_174838/
train/plugins/profile/2021_09_29_17_48_49/tensorflow-2-6-20210929-131042.input_p
ipeline.pb
Dumped tool data for tensorflow_stats.pb to mnist_models/models/cnn_210929_17483
8/train/plugins/profile/2021_09_29_17_48_49/tensorflow-2-6-20210929-131042.tenso
rflow_stats.pb
Dumped tool data for kernel_stats.pb to mnist_models/models/cnn_210929_174838/tr
ain/plugins/profile/2021_09_29_17_48_49/tensorflow-2-6-20210929-131042.kernel_st
ats.pb

2021-09-29 17:49:14.628390: W tensorflow/python/util/util.cc:348] Sets are not c
urrently considered sequences, but this may change in the future, so consider av
oiding using them.
```

Training on the cloud

Since we're using an unreleased version of TensorFlow on AI Platform, we can instead use a Deep Learning Container in order to take advantage of libraries and applications not normally packaged with AI Platform. Below is a simple Dockerlife which copies our code to be used in a TF2 environment.

In [25]:
```
%%writefile mnist_models/Dockerfile
FROM gcr.io/deeplearning-platform-release/tf2-cpu
COPY mnist_models/trainer /mnist_models/trainer
ENTRYPOINT ["python3", "-m", "mnist_models.trainer.task"]
```

```
Writing mnist_models/Dockerfile
```

The below command builds the image and ships it off to Google Cloud so it can be used for AI Platform. When built, it will show up here with the name `mnist_models`. (Click here to enable Cloud Build)

In [26]:
```
!docker build -f mnist_models/Dockerfile -t $IMAGE_URI ./
```

```
Sending build context to Docker daemon   5.11MB
Step 1/3 : FROM gcr.io/deeplearning-platform-release/tf2-cpu
latest: Pulling from deeplearning-platform-release/tf2-cpu

327ec0e7: Pulling fs layer
1b6ebc96: Pulling fs layer
09099ad7: Pulling fs layer
6a8b630b: Pulling fs layer
b700ef54: Pulling fs layer
b3064d0c: Pulling fs layer
01c6f2ce: Pulling fs layer
66c569d3: Pulling fs layer
1fb91b32: Pulling fs layer
14694728: Pulling fs layer
ab248284: Pulling fs layer
e6559d28: Pulling fs layer
c3c444d6: Pulling fs layer
e84e616f: Pulling fs layer
d14c1b75: Pulling fs layer
6fc56e5d: Pulling fs layer
7a3c7a92: Pulling fs layer
e609592a: Pulling fs layer
cea30e25: Pulling fs layer
Digest: sha256:5b5ca9e6be49a477a09c45fbba5dfa8a2461d7fc352368f0a75f01d49e6a7d482
KDownloading  1.095GB/1.319GB
Status: Downloaded newer image for gcr.io/deeplearning-platform-release/tf2-cpu:
latest
 ---> fb8ac72580b0
Step 2/3 : COPY mnist_models/trainer /mnist_models/trainer
 ---> 7c3c71ceeb06
Step 3/3 : ENTRYPOINT ["python3", "-m", "mnist_models.trainer.task"]
 ---> Running in 2b1adf075988
Removing intermediate container 2b1adf075988
 ---> 44b921427bce
Successfully built 44b921427bce
Successfully tagged gcr.io/qwiklabs-gcp-02-e914dd157b41/mnist_models:latest
```

In [27]:
```
!docker push $IMAGE_URI
```

```
Using default tag: latest
The push refers to repository [gcr.io/qwiklabs-gcp-02-e914dd157b41/mnist_models]

c4054cbc: Preparing
7431c89f: Preparing
ad311e6f: Preparing
5f0ad326: Preparing
1a3ac85a: Preparing
b42995cf: Preparing
e5699636: Preparing
ebb0a931: Preparing
e3bb5d74: Preparing
c32f59e4: Preparing
4aa64610: Preparing
3f5b29db: Preparing
ab39d28b: Preparing
e3198a0c: Preparing
d55d2779: Preparing
44f7a682: Preparing
bf18a086: Preparing
37fa3c8c: Preparing
9a4914af: Preparing
3162c7c3: Preparing
3162c7c3: Mounted from deeplearning-platform-release/tf2-cpu latest: digest: sha
256:075ca52e51ead1c38a6815d228eb4df6887e59423efd27ef643ca116c3e0dda8 size: 4713
```

Finally, we can kickoff the AI Platform training job. We can pass in our docker image using the `master-image-uri` flag.

In [28]:

```
current_time = datetime.now().strftime("%y%m%d_%H%M%S")
model_type = 'cnn' # "linear", "cnn", "dnn_dropout", or "dnn"

os.environ["MODEL_TYPE"] = model_type
os.environ["JOB_DIR"] = "gs://{}/mnist_{}_{}/".format(
    BUCKET, model_type, current_time)
os.environ["JOB_NAME"] = "mnist_{}_{}".format(
    model_type, current_time)
```

In [33]:

```
%%bash
echo $JOB_DIR $REGION $JOB_NAME
gcloud ai-platform jobs submit training $JOB_NAME \
    --staging-bucket=gs://$BUCKET \
    --region=$REGION \
    --master-image-uri=$IMAGE_URI \
    --scale-tier=BASIC_GPU \
    --job-dir=$JOB_DIR \
    -- \
    --model_type=$MODEL_TYPE
```

```
gs://qwiklabs-gcp-02-e914dd157b41/mnist_cnn_210929_175215/ us-central1 mnist_cnn
_210929_175215
jobId: mnist_cnn_210929_175215
state: QUEUED
Job [mnist_cnn_210929_175215] submitted successfully.
Your job is still active. You may view the status of your job with the command

  $ gcloud ai-platform jobs describe mnist_cnn_210929_175215
```

```
or continue streaming the logs with the command

  $ gcloud ai-platform jobs stream-logs mnist_cnn_210929_175215
```

Deploying and predicting with model

Once you have a model you're proud of, let's deploy it! All we need to do is give AI Platform the location of the model. Below uses the keras export path of the previous job, but `${JOB_DIR}keras_export/` can always be changed to a different path.

Uncomment the delete commands below if you are getting an "already exists error" and want to deploy a new model.

In [35]:

```
%%bash
MODEL_NAME="mnist"
MODEL_VERSION=${MODEL_TYPE}
MODEL_LOCATION=${JOB_DIR}keras_export/
echo "Deleting and deploying $MODEL_NAME $MODEL_VERSION from $MODEL_LOCATION ..."
#yes | gcloud ai-platform versions delete ${MODEL_VERSION} --model ${MODEL_NAME}
#yes | gcloud ai-platform models delete ${MODEL_NAME}
gcloud config set ai_platform/region global
gcloud ai-platform models create ${MODEL_NAME} --regions $REGION
gcloud ai-platform versions create ${MODEL_VERSION} \
    --model ${MODEL_NAME} \
    --origin ${MODEL_LOCATION} \
    --framework tensorflow \
    --runtime-version=2.6
```

```
Deleting and deploying mnist cnn from gs://qwiklabs-gcp-02-e914dd157b41/mnist_cn
n_210929_175215/keras_export/ ... this will take a few minutes
Updated property [ai_platform/region].
Using endpoint [https://ml.googleapis.com/]
ERROR: (gcloud.ai-platform.models.create) Resource in projects [qwiklabs-gcp-02-
e914dd157b41] is the subject of a conflict: Field: model.name Error: A model wit
h the same name already exists.
- '@type': type.googleapis.com/google.rpc.BadRequest
  fieldViolations:
  - description: A model with the same name already exists.
    field: model.name
Using endpoint [https://ml.googleapis.com/]
ERROR: (gcloud.ai-platform.versions.create) FAILED_PRECONDITION: Field: version.
deployment_uri Error: The provided URI for model files doesn't contain any objec
ts.
- '@type': type.googleapis.com/google.rpc.BadRequest
  fieldViolations:
  - description: The provided URI for model files doesn't contain any objects.
    field: version.deployment_uri
---------------------------------------------------------------------------
CalledProcessError                        Traceback (most recent call last)
/tmp/ipykernel_29243/2033427367.py in <module>
----> 1 get_ipython().run_cell_magic('bash', '', 'MODEL_NAME="mnist"\nMODEL_VERS
ION=${MODEL_TYPE}\nMODEL_LOCATION=${JOB_DIR}keras_export/\necho "Deleting and de
ploying $MODEL_NAME $MODEL_VERSION from $MODEL_LOCATION ... this will take a few
minutes"\n#yes | gcloud ai-platform versions delete ${MODEL_VERSION} --model ${M
ODEL_NAME}\n#yes | gcloud ai-platform models delete ${MODEL_NAME}\ngcloud config
set ai_platform/region global\ngcloud ai-platform models create ${MODEL_NAME} --
regions $REGION\ngcloud ai-platform versions create ${MODEL_VERSION} \\\n    --m
```

```
odel ${MODEL_NAME} \\\n    --origin ${MODEL_LOCATION} \\\n    --framework tensor
flow \\\n    --runtime-version=2.6\n')

/opt/conda/lib/python3.7/site-packages/IPython/core/interactiveshell.py in run_c
ell_magic(self, magic_name, line, cell)
   2401             with self.builtin_trap:
   2402                 args = (magic_arg_s, cell)
-> 2403                 result = fn(*args, **kwargs)
   2404             return result
   2405

/opt/conda/lib/python3.7/site-packages/IPython/core/magics/script.py in named_sc
ript_magic(line, cell)
    140             else:
    141                 line = script
--> 142             return self.shebang(line, cell)
    143
    144         # write a basic docstring:

/opt/conda/lib/python3.7/site-packages/decorator.py in fun(*args, **kw)
    230             if not kwsyntax:
    231                 args, kw = fix(args, kw, sig)
--> 232             return caller(func, *(extras + args), **kw)
    233     fun.__name__ = func.__name__
    234     fun.__doc__ = func.__doc__

/opt/conda/lib/python3.7/site-packages/IPython/core/magic.py in <lambda>(f, *a,
 **k)
    185     # but it's overkill for just that one bit of state.
    186     def magic_deco(arg):
--> 187         call = lambda f, *a, **k: f(*a, **k)
    188
    189         if callable(arg):

/opt/conda/lib/python3.7/site-packages/IPython/core/magics/script.py in shebang
(self, line, cell)
    243             sys.stderr.flush()
    244         if args.raise_error and p.returncode!=0:
--> 245             raise CalledProcessError(p.returncode, cell, output=out, std
err=err)
    246
    247     def _run_script(self, p, cell, to_close):

CalledProcessError: Command 'b'MODEL_NAME="mnist"\nMODEL_VERSION=${MODEL_TYPE}\n
MODEL_LOCATION=${JOB_DIR}keras_export/\necho "Deleting and deploying $MODEL_NAME
$MODEL_VERSION from $MODEL_LOCATION ... this will take a few minutes"\n#yes | gc
loud ai-platform versions delete ${MODEL_VERSION} --model ${MODEL_NAME}\n#yes |
 gcloud ai-platform models delete ${MODEL_NAME}\ngcloud config set ai_platform/r
egion global\ngcloud ai-platform models create ${MODEL_NAME} --regions $REGION\n
gcloud ai-platform versions create ${MODEL_VERSION} \\\n    --model ${MODEL_NAM
E} \\\n    --origin ${MODEL_LOCATION} \\\n    --framework tensorflow \\\n    --r
untime-version=2.6\n'' returned non-zero exit status 1.
```

To predict with the model, let's take one of the example images.

TODO 4: Write a `.json` file with image data to send to an AI Platform deployed model

In [31]:

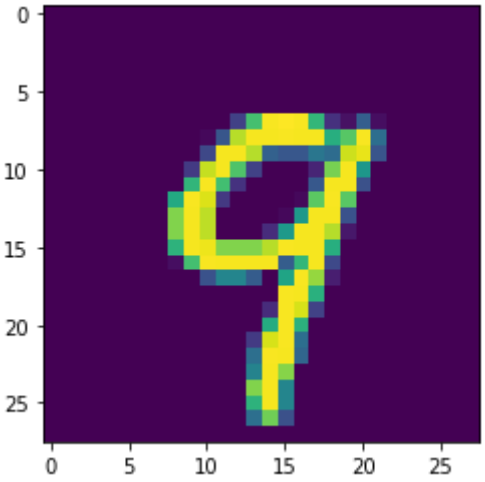
```
import json, codecs
import tensorflow as tf
import matplotlib.pyplot as plt
```

```
from mnist_models.trainer import util

HEIGHT = 28
WIDTH = 28
IMGNO = 12

mnist = tf.keras.datasets.mnist.load_data()
(x_train, y_train), (x_test, y_test) = mnist
test_image = x_test[IMGNO]

jsondata = test_image.reshape(HEIGHT, WIDTH, 1).tolist()
json.dump(jsondata, codecs.open("test.json", "w", encoding = "utf-8"))
plt.imshow(test_image.reshape(HEIGHT, WIDTH));
```



Finally, we can send it to the prediction service. The output will have a 1 in the index of the corresponding digit it is predicting. Congrats! You've completed the lab!

In [32]:

```
%%bash
gcloud ai-platform predict \
    --model=mnist \
    --version=${MODEL_TYPE} \
    --json-instances=./test.json
```

```
Using endpoint [https://ml.googleapis.com/]
ERROR: (gcloud.ai-platform.predict) NOT_FOUND: Field: name Error: The specified
model version was not found.
- '@type': type.googleapis.com/google.rpc.BadRequest
  fieldViolations:
  - description: The specified model version was not found.
    field: name
---------------------------------------------------------------------------
CalledProcessError                        Traceback (most recent call last)
/tmp/ipykernel_29243/1006830016.py in <module>
----> 1 get_ipython().run_cell_magic('bash', '', 'gcloud ai-platform predict
 \\\n    --model=mnist \\\n    --version=${MODEL_TYPE} \\\n    --json-instances
=./test.json\n')

/opt/conda/lib/python3.7/site-packages/IPython/core/interactiveshell.py in run_c
ell_magic(self, magic_name, line, cell)
   2401             with self.builtin_trap:
   2402                 args = (magic_arg_s, cell)
-> 2403                 result = fn(*args, **kwargs)
```

```
   2404             return result
   2405 

/opt/conda/lib/python3.7/site-packages/IPython/core/magics/script.py in named_sc
ript_magic(line, cell)
    140             else:
    141                 line = script
--> 142             return self.shebang(line, cell)
    143 
    144         # write a basic docstring:

/opt/conda/lib/python3.7/site-packages/decorator.py in fun(*args, **kw)
    230             if not kwsyntax:
    231                 args, kw = fix(args, kw, sig)
--> 232             return caller(func, *(extras + args), **kw)
    233     fun.__name__ = func.__name__
    234     fun.__doc__ = func.__doc__

/opt/conda/lib/python3.7/site-packages/IPython/core/magic.py in <lambda>(f, *a,
 **k)
    185     # but it's overkill for just that one bit of state.
    186     def magic_deco(arg):
--> 187         call = lambda f, *a, **k: f(*a, **k)
    188 
    189         if callable(arg):

/opt/conda/lib/python3.7/site-packages/IPython/core/magics/script.py in shebang
(self, line, cell)
    243             sys.stderr.flush()
    244         if args.raise_error and p.returncode!=0:
--> 245             raise CalledProcessError(p.returncode, cell, output=out, std
err=err)
    246 
    247     def _run_script(self, p, cell, to_close):

CalledProcessError: Command 'b'gcloud ai-platform predict \\\n    --model=mnist
 \\\n    --version=${MODEL_TYPE} \\\n    --json-instances=./test.json\n'' return
ed non-zero exit status 1.
```