NVIDIA DIGITS Container

Getting Started Guide

DU-09193-001 _v1.0 | September 2021

Table of Contents

Chapter 1. Overview

DIGITS (the Deep Learning GPU Training System) is a web app for training deep learning models, and currently supports the TensorFlow framework. DIGITS puts the power of deep learning into the hands of engineers and data scientists.

DIGITS is not a framework. DIGITS is a wrapper for TensorFlow; which provides a graphical web interface to those frameworks rather than dealing with them directly on the command-line.

DIGITS can be used to rapidly train highly accurate deep neural network (DNNs) for image classification, segmentation, object detection tasks, and more. DIGITS simplifies common deep learning tasks such as managing data, designing and training neural networks on multi-GPU systems, monitoring performance in real time with advanced visualizations, and selecting the best performing model from the results browser for deployment. DIGITS is completely interactive so that data scientists can focus on designing and training networks rather than programming and debugging.

DIGITS is available through multiple channels such as:

- GitHub download
- NVIDIA's Docker repository, `nvcr.io`

This guide walks you through getting up-and-running with the DIGITS container downloaded from NVIDIA's Docker repository. To install the DIGITS application by itself, see the DIGITS Installation Guide.

The container image available in the NVIDIA Docker repository, `nvcr.io`, is pre-built and installed into the `/usr/local/python/` directory.

DIGITS also includes the NVIDIA TensorFlow deep learning framework.

Chapter 2. Pulling The Container

Before you can pull a container from the NGC Registry, you must have Docker and nvidia-docker installed. For DGX users, this is explained in Preparing to use NVIDIA Containers Getting Started Guide.

For users other than DGX, follow the NVIDIA® GPU Cloud™ (NGC) registry nvidia-docker installation documentation based on your platform.

You must also have access and be logged into the NGC Registry as explained in the NGC Getting Started Guide.

There are four repositories where you can find the NGC docker containers.

`nvcr.io/nvidia`
: The deep learning framework containers are stored in the `nvcr.io/nvidia` repository.

`nvcr.io/hpc`
: The HPC containers are stored in the `nvcr.io/hpc` repository.

`nvcr.io/nvidia-hpcvis`
: The HPC visualization containers are stored in the `nvcr.io/nvidia-hpcvis` repository.

`nvcr.io/partner`
: The partner containers are stored in the `nvcr.io/partner` repository. Currently the partner containers are focused on Deep Learning or Machine Learning, but that doesn’t mean they are limited to those types of containers.

Chapter 3. Running DIGITS

About this task

On your system, before running the application, use the `docker pull` command to ensure an up-to-date image is installed. Once the pull is complete, you can run the application. This is because nvidia-docker ensures that drivers that match the host are used and configured for the container. Without nvidia-docker, you are likely to get an error when trying to run the container.

Procedure

1. Issue the command for the applicable release of the container that you want. The following command assumes you want to pull the latest container.
   ```
   docker pull nvcr.io/nvidia/digits:19.xx-tensorflow
   ```
2. Open a command prompt and paste the pull command. The pulling of the container image begins. Ensure the pull completes successfully before proceeding to the next step.
3. Run the application. A typical command to launch the application is:
   ```
   docker run --gpus all -it --rm –v local_dir:container_dir
   nvcr.io/nvidia/digits:<xx.xx>-<framework>
   ```

   Where:

   - `-it` means interactive
   - `--rm` means delete the application when finished
   - `–v` means mount directory
   - `local_dir` is the directory or file from your host system (absolute path) that you want to access from inside your container. For example, the `local_dir` in the following path is `/home/jsmith/data/mnist`.
     ```
     -v /home/jsmith/data/mnist:/data/mnist
     ```
     If you are inside the container, for example, `ls /data/mnist`, you will see the same files as if you issued the `ls /home/jsmith/data/mnist` command from outside the container.
   - `container_dir` is the target directory when you are inside your container. For example, `/data/mnist` is the target directory in the example:
     ```
     -v /home/jsmith/data/mnist:/data/mnist
     ```
   - `<xx.xx>` is the container version. For example, `19.01`.

- `<framework>` is the framework that you want to pull. For example, `tensorflow`.

a). To run the server as a daemon and expose port 5000 in the container to port 8888 on your host:

```
docker run --gpus all --name digits -d -p 8888:5000
nvcr.io/nvidia/digits:<xx.xx>-<framework>
```

> **Note:** DIGITS 6.0 uses port 5000 by default.

b). To mount one local directory containing your data (read-only), and another for writing your DIGITS jobs:

```
docker run --gpus all --name digits -d -p 8888:5000 -v
/home/username/data:/data -v /home/username/digits-
jobs:/workspace/jobs nvcr.io/nvidia/digits:<xx.xx>-<framework>
```

> **Note:** In order to share data between ranks, NVIDIA® Collective Communications Library ™ (NCCL) may require shared system memory for IPC and pinned (page-locked) system memory resources. The operating system's limits on these resources may need to be increased accordingly. Refer to your system's documentation for details.
>
> In particular, Docker containers default to limited shared and pinned memory resources. When using NCCL inside a container, it is recommended that you increase these resources by issuing:
>
> ```
> --shm-size=1g --ulimit memlock=-1
> ```
>
> in the command line to:
>
> ```
> docker run --gpus all
> ```

4. See `/workspace/README.md` inside the container for information on customizing your DIGITS application.

   For more information about DIGITS, see:

   - DIGITS website
   - DIGITS project
   - nvidia-docker documentation

> **Note:** There may be slight variations between the Dockerhub images and this image.

Chapter 4. Deep Learning Frameworks for DIGITS

The DIGITS application in the NVIDIA Docker repository, `nvcr.io`, comes with DIGITS, but also comes with TensorFlow. You can read the details in the container release notes here (http://docs.nvidia.com/deeplearning/digits/). For example, the 21.01 release of DIGITS includes the 21.01 release of TensorFlow.

DIGITS is a training platform that can be used with NVIDIA TensorFlow deep learning frameworks. Using any of these frameworks, DIGITS will train your deep learning models on your dataset.

The following sections include examples using DIGITS with a TensorFlow backend.

4.1. TensorFlow for DIGITS

TensorFlow for DIGITS works with DIGITS v6.0 and later.

4.1.1. Example 1: MNIST

The MNIST dataset comes with the DIGITS application.

1. The first step in training a model with DIGITS and TensorFlow is to pull the DIGITS container from the nvcr.io registry (be sure you are logged into the appropriate registry).
```
$ docker pull nvcr.io/nvidia/digits:17.04
```
2. After the container has been pulled, you can start DIGITS on the DGX system. Because DIGITS is a web-based frontend for TensorFlow, we will run the DIGITS application in a non-interactive way using the following command.
```
docker run --gpus all -d --name digits-17.04 -p 8888:5000 nvcr.io/nvidia/
digits:17.04
```

   There are a number of options in this command.

   - The first option "-d" tells Docker to run the container in "daemon" mode.
   - The "--name" option "names" the running container (we will need this later).
   - The "-p 8888:5000" option maps the DIGITS port 5000 to port 8888 (you will see how this is used below).

After you run this command you need to find the IP address of the DIGITS node. This can be found by running the command `ifconfig` as shown here:

```
$ ifconfig
docker0: flags=4163<UP,BROADCAST,RUNNING,MULTICAST>  mtu 1500
        inet 192.168.99.1  netmask 255.255.255.0  broadcast 0.0.0.0
        inet6 fe80::42:5cff:fefb:1c30  prefixlen 64  scopeid 0x20<link>
        ether 02:42:5c:fb:1c:30  txqueuelen 0  (Ethernet)
        RX packets 22649  bytes 5171804 (4.9 MiB)
        RX errors 0  dropped 0  overruns 0  frame 0
        TX packets 29088  bytes 123439479 (117.7 MiB)
        TX errors 0  dropped 0 overruns 0  carrier 0  collisions 0

enp1s0f0: flags=4163<UP,BROADCAST,RUNNING,MULTICAST>  mtu 1500
        inet 10.31.229.99  netmask 255.255.255.128  broadcast 10.31.229.127
        inet6 fe80::56ab:3aff:fed6:614f  prefixlen 64  scopeid 0x20<link>
        ether 54:ab:3a:d6:61:4f  txqueuelen 1000  (Ethernet)
        RX packets 8116350  bytes 11069954019 (10.3 GiB)
        RX errors 0  dropped 9  overruns 0  frame 0
        TX packets 1504305  bytes 162349141 (154.8 MiB)
        TX errors 0  dropped 0 overruns 0  carrier 0  collisions 0
...
```

In this case, we want the Ethernet IP address since that is the address of the web server for DIGITS (10.31.229.56 for this example). Your IP address will be different.

3. We now need to download the MNIST data set into the container. The DIGITS container has a simple script for downloading the data set into the container. As a check, run the following command to make sure the container is running.

```
$ docker ps -a
CONTAINER ID    IMAGE                        ...  NAMES
c930962b9636    nvcr.io/nvidia/digits:17.04 ...  digits-17.04
```

The application is running and has the name that we gave it (`digits-17.04`).

Next you need to "shell" into the running container from another terminal on the system.

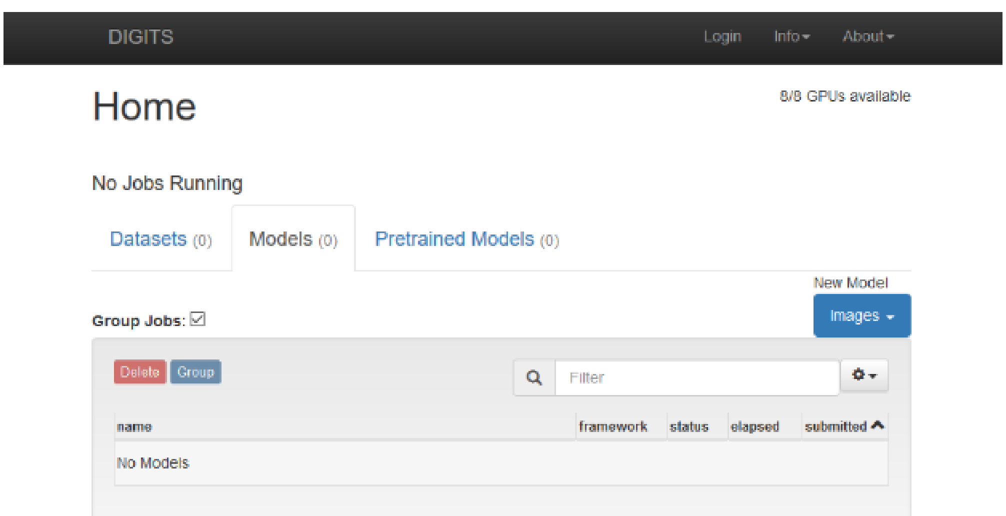
```
$ docker exec -it digits-17.04 bash
root@XXXXXXXXXXXX:/workspace#
```

We want to put the data into the directory `/data/mnist`. There is a simple Python script in the application that will do this for us. It downloads the data in the correct format as well.

```
# python -m digits.download_data mnist /data/mnist
Downloading url=http://yann.lecun.com/exdb/mnist/train-images-idx3-ubyte.gz ...
Downloading url=http://yann.lecun.com/exdb/mnist/train-labels-idx1-ubyte.gz ...
Downloading url=http://yann.lecun.com/exdb/mnist/t10k-images-idx3-ubyte.gz ...
Downloading url=http://yann.lecun.com/exdb/mnist/t10k-labels-idx1-ubyte.gz ...
Uncompressing file=train-images-idx3-ubyte.gz ...
Uncompressing file=train-labels-idx1-ubyte.gz ...
Uncompressing file=t10k-images-idx3-ubyte.gz ...
Uncompressing file=t10k-labels-idx1-ubyte.gz ...
Reading labels from /data/mnist/train-labels.bin ...
Reading images from /data/mnist/train-images.bin ...
Reading labels from /data/mnist/test-labels.bin ...
Reading images from /data/mnist/test-images.bin ...
Dataset directory is created successfully at '/data/mnist'
Done after 13.4188599586 seconds.
```

4. You can now open a web browser to the IP address from the previous step. Be sure to use port 8888 since we mapped the DIGITS port from 5000 to port 8888. For this example, the URL would be the following.

   ```
   10.31.229.56:8888
   ```

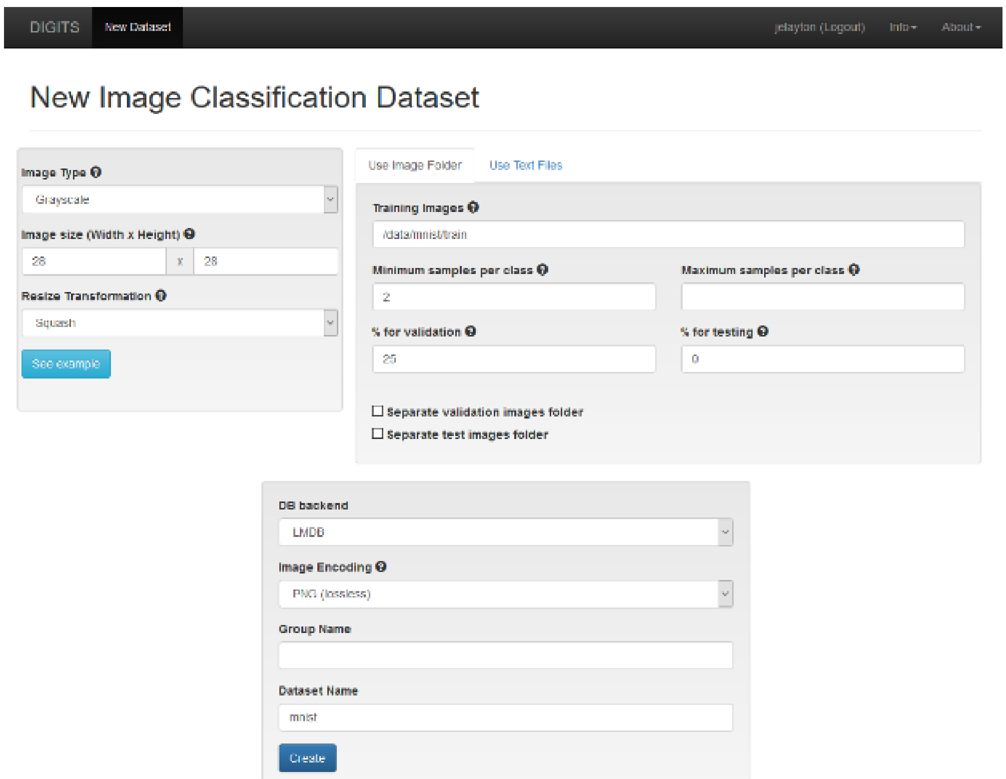


This is the home page of DIGITS. Notice that in the top right corner it says that there are 8 of 8 GPUs available on this DGX-1. For a DGX Station this should be 4 or 4 GPUs available. For NVIDIA NGC Cloud Services on a cloud provider, the number of GPUs should match the number for the instance type.

5. Load a dataset. We are going to use the MNIST dataset as an example since it comes with the application.

   a). Click the **Datasets** tab.

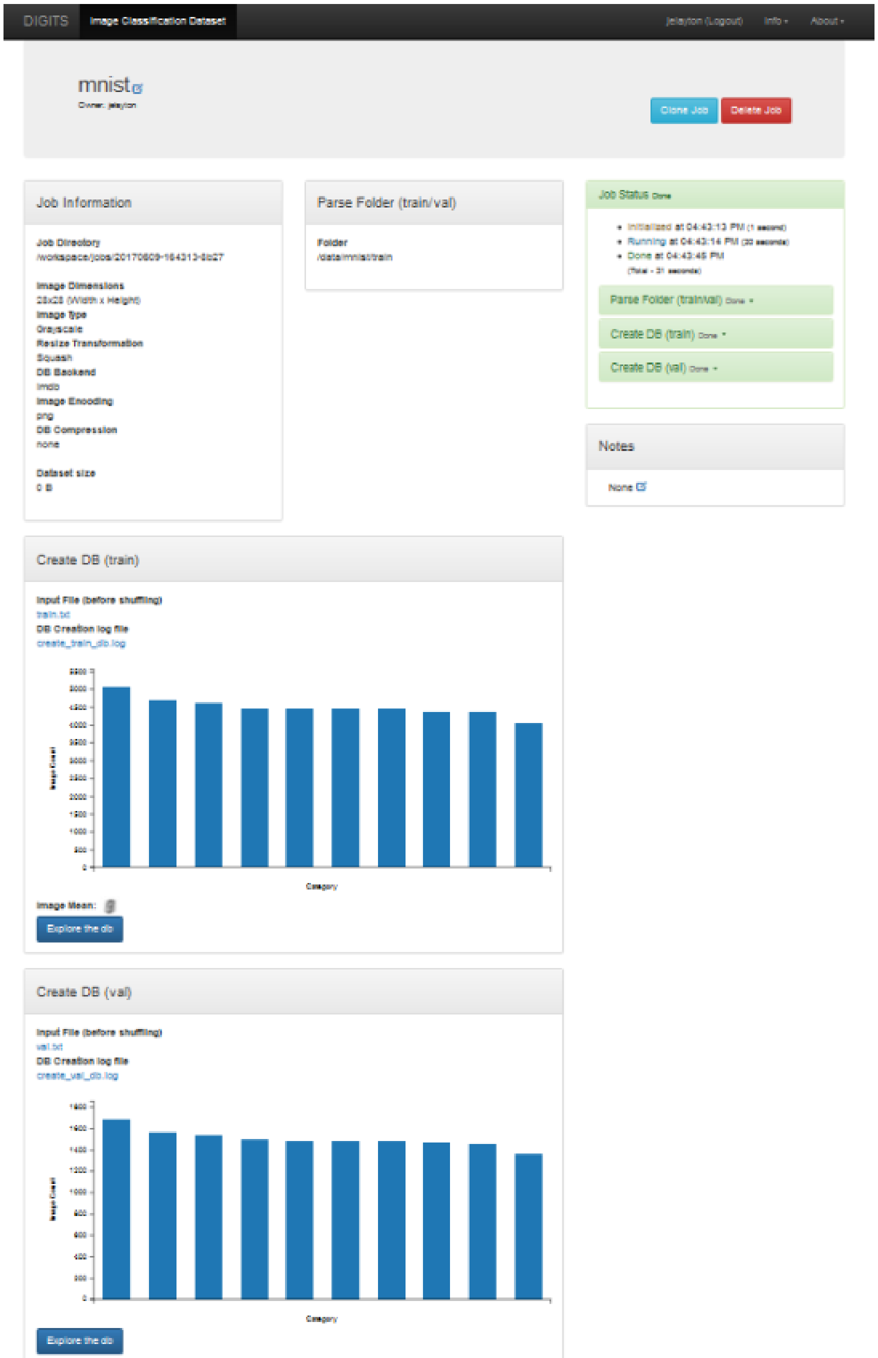
   b). Click the **Images** drop down menu and select **Classification**. If DIGITS asks for a user name, you can enter anything you want. The New Image Classification Dataset window displays. After filling in the fields, your screen should look like the following.



   c). Provide values for the **Image Type** and the **Image size** as shown in the above image.

   d). Give your dataset a name in the **Dataset Name** field. You can name the dataset anything you like. In this case the name is just "mnist".

   e). Click **Create**. This tells DIGITS to tell Caffe to load the datasets. After the datasets are loaded, your screen should look similar to the following.



> **Note:** There are two sections that allow you to "explore" the db (database). The Create DB (train) is for training data and Create DB (val) is for validating data. In either of these displays, you can click **Explore the db** for the training set.

6. Train a model. We're going to use Yann Lecun's LeNet model as an example since it comes with the application.

   a). Define the model. Click **DIGITS** in the upper left corner to be taken back to the home page.

b). Click the **Models** tab.

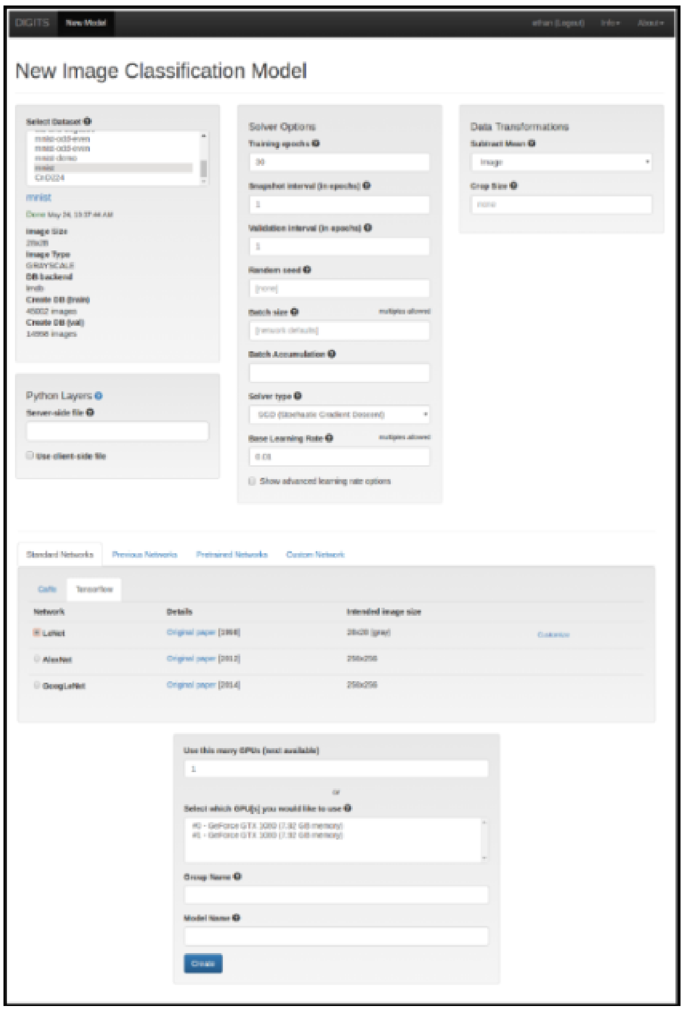
c). Click the **Images** drop down menu and select **Classification**. The New Image Classification Model window displays.

d). Provide values for the **Select Dataset** and the training parameter fields.

e). In the **Standard Networks** tab, click **Caffe** and select the **LeNet** radio button.

> **Note:** DIGITS allows you to use previous networks, pretrained networks, and customer networks if you want.

f). Click **Create**. The training of the LeNet model starts.

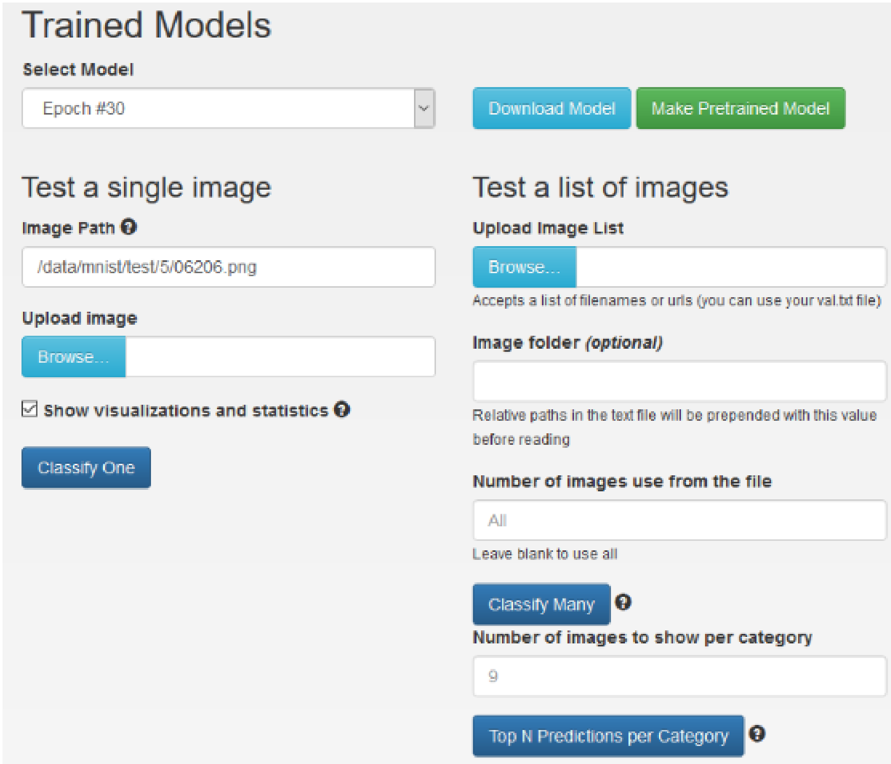


During the training, DIGITS displays the history of the training parameters, specifically, the loss function for the training data, the accuracy from the validation data set, and the loss function for the validation data. After the training completes, (all 30 epochs are trained), your screen should look similar to the following.

> **Note:** This screen capture has been truncated because the web page is very long.

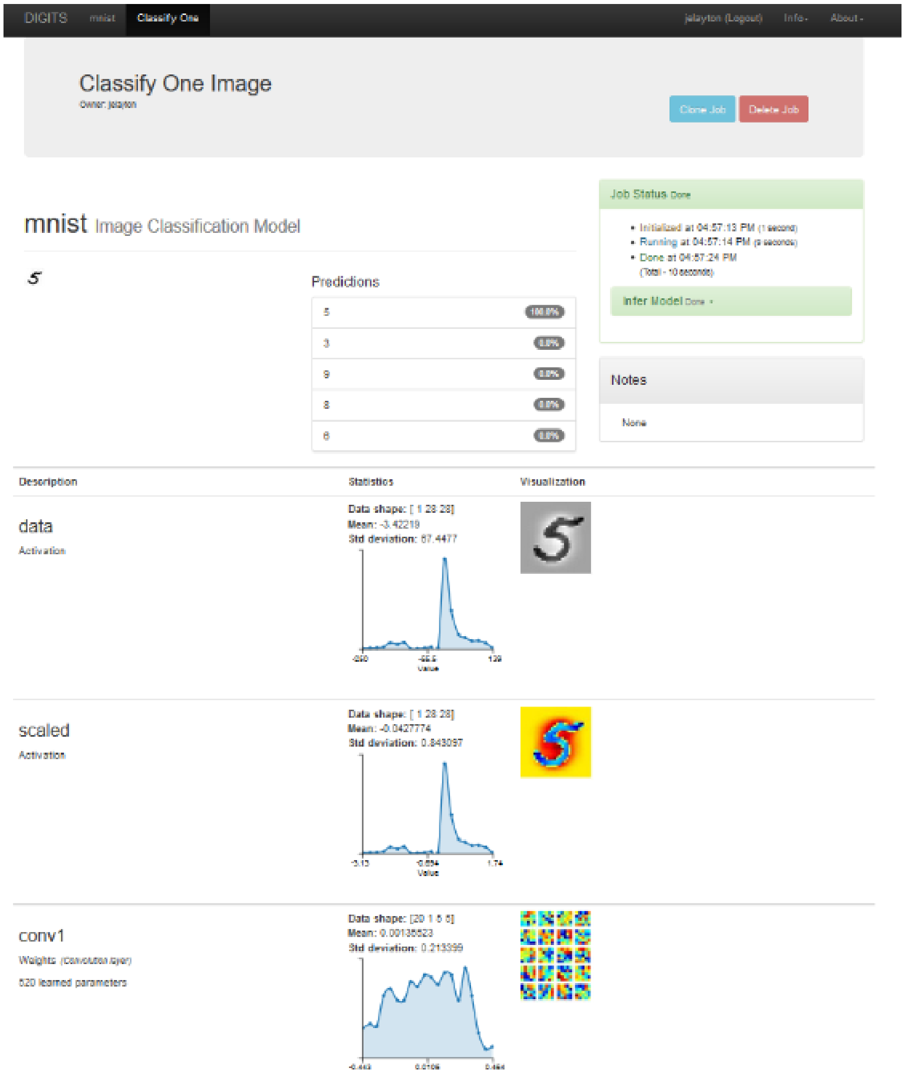
7. Optional: You can test some images (inference) against the trained model by scrolling to the bottom of the web page. For illustrative purposes, a single image is input from the test data set. You can always upload an image if you like. You can also input a list of test images if you want.

   The screen below does inference against a test image called `/data/mnist/test/5/06206.png`. Also, select the **Statistics and Visualizations** checkbox to ensure that you can see all of the details from the network as well as the network prediction.



Note: You can select a model from any of the epochs if you want. To do so, click the **Select Model** drop down arrow and select a different epoch.

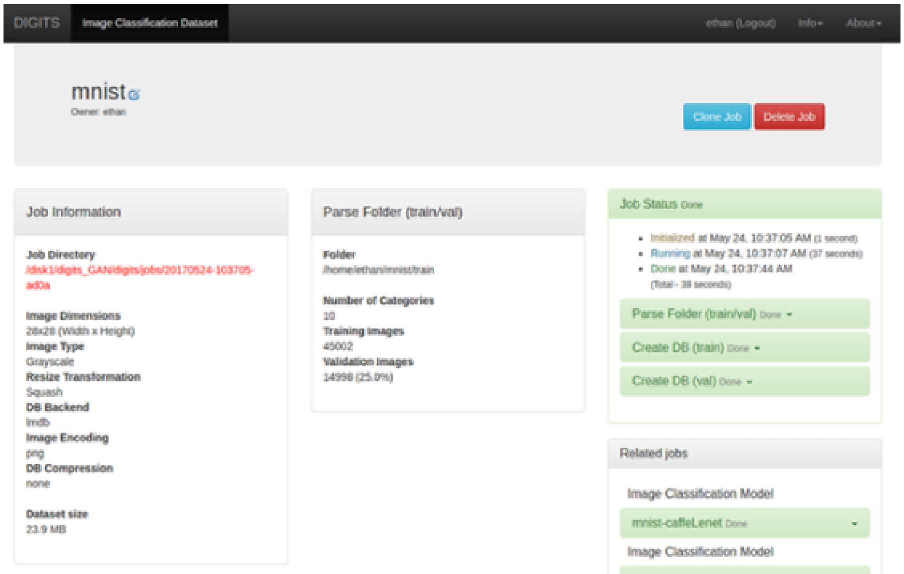
8. Click **Classify One**. This opens another browser tab and displays predictions. The screen below is the output for the test image that is the number "5".



4.1.2. Example 2: Siamese Network

Prerequisites:

1. In order to train a Siamese dataset, you must first have the MNIST dataset. To create the MNIST dataset, see the "TensorFlow MNIST example".
2. Remember the Job Directory path, since this is needed in this task.



Procedure:

1. Execute the Python script available from: https://github.com/NVIDIA/DIGITS/blob/master/examples/siamese/create_db.py. The script requires the following parameters:

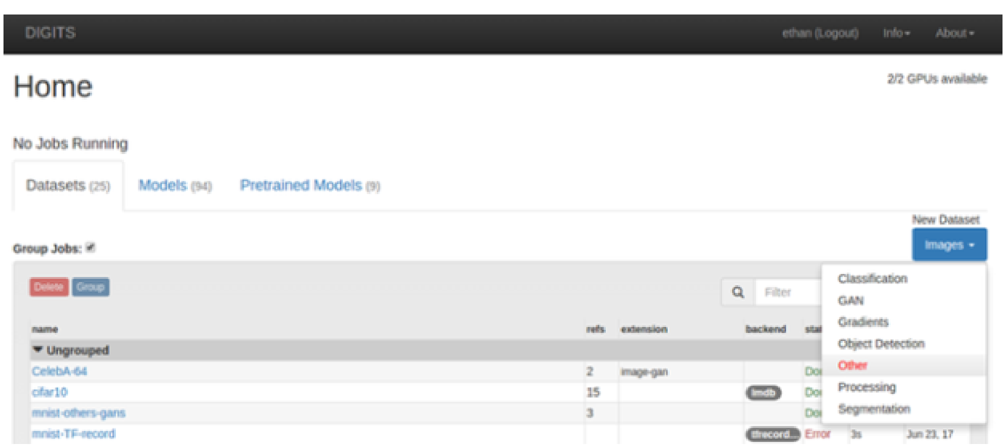
```
Create_db.py <where to save results><the job directory> -c <how many samples>
```

   Where:

   - `<where to save results>` is the directory path where you want to save your output.
   - `<the job directory>` is the name of the directory that you took note of in the prerequisites.
   - `<how many samples>` is where you define the number of samples. Set this number to `100000`.

2. Create the Siamese dataset.

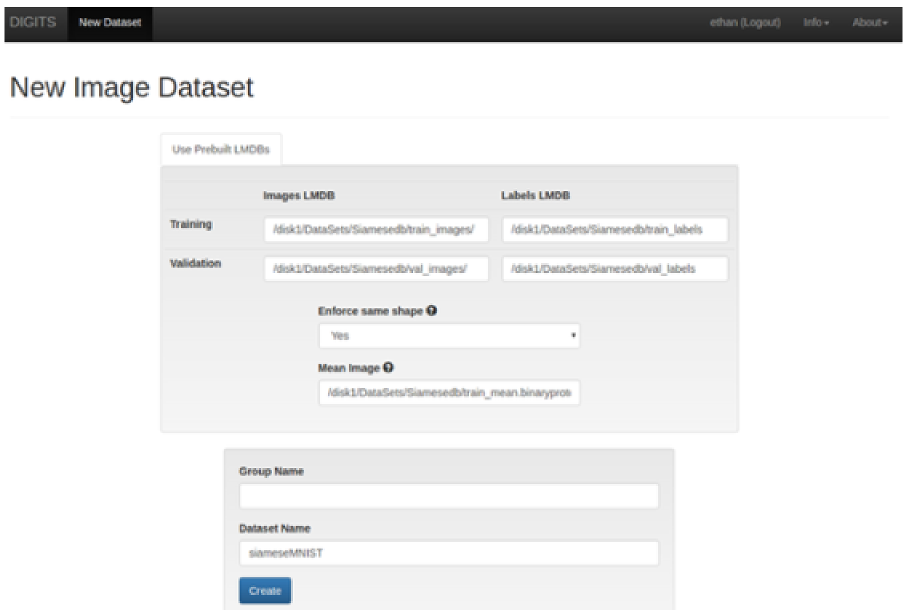
   a). On the Home page, click **New Dataset > Images > Other**.



b). Provide the directory paths to the following fields:

i. The train image database
ii. The train label database
iii. The validation image database
iv. The validation image database
v. The train image train_mean.binaryproto file

> **Note:** The directory path should be the same location that was specified in <where to save results>.

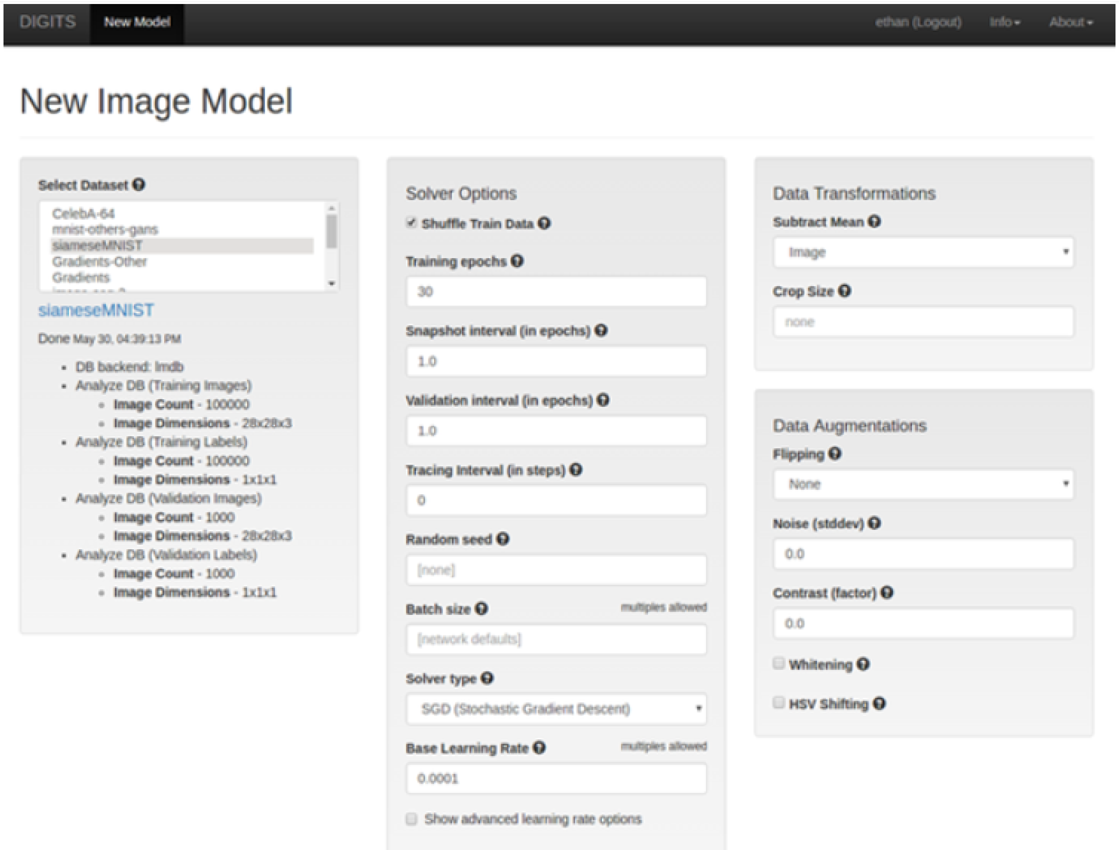


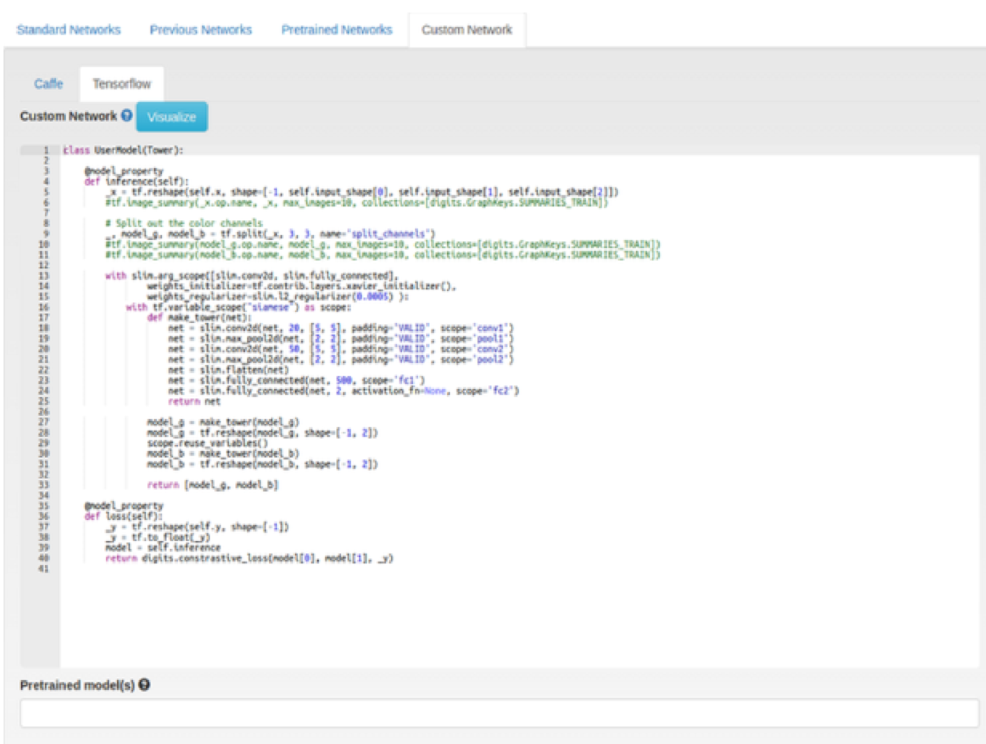
3. Click **New Model > Images > Other** to create the model. In this example, we will use Caffe to train our Siamese network.
4. Test the model.

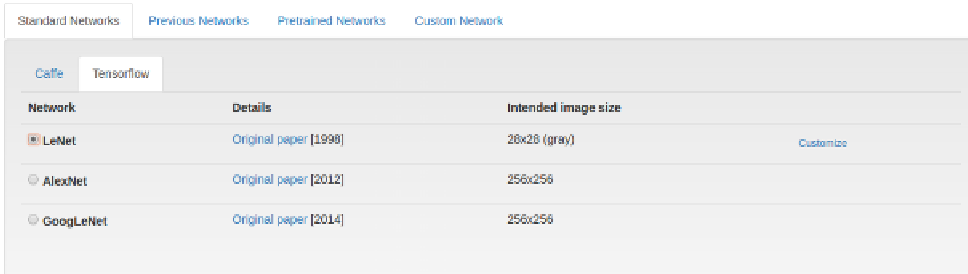
a). Click the **Custom Network** tab and select **TensorFlow**.

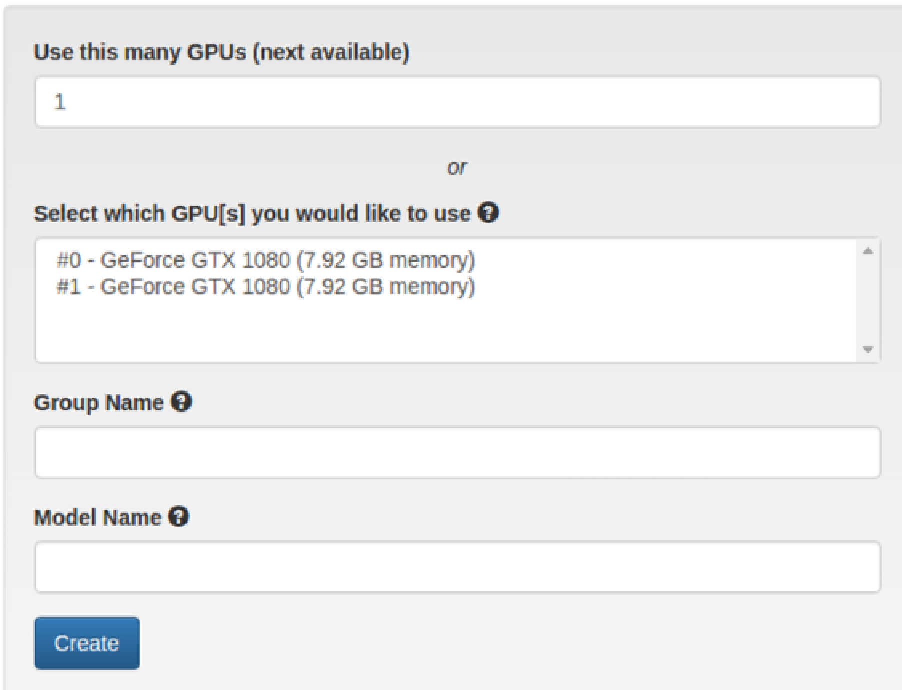
b). Copy and paste the following network definition: https://github.com/NVIDIA/DIGITS/blob/master/examples/siamese/siamese-TF.py.

c). Ensure the Base Learning Rate is set to 0.01, keep the default settings to all other fields, and click **Train**.

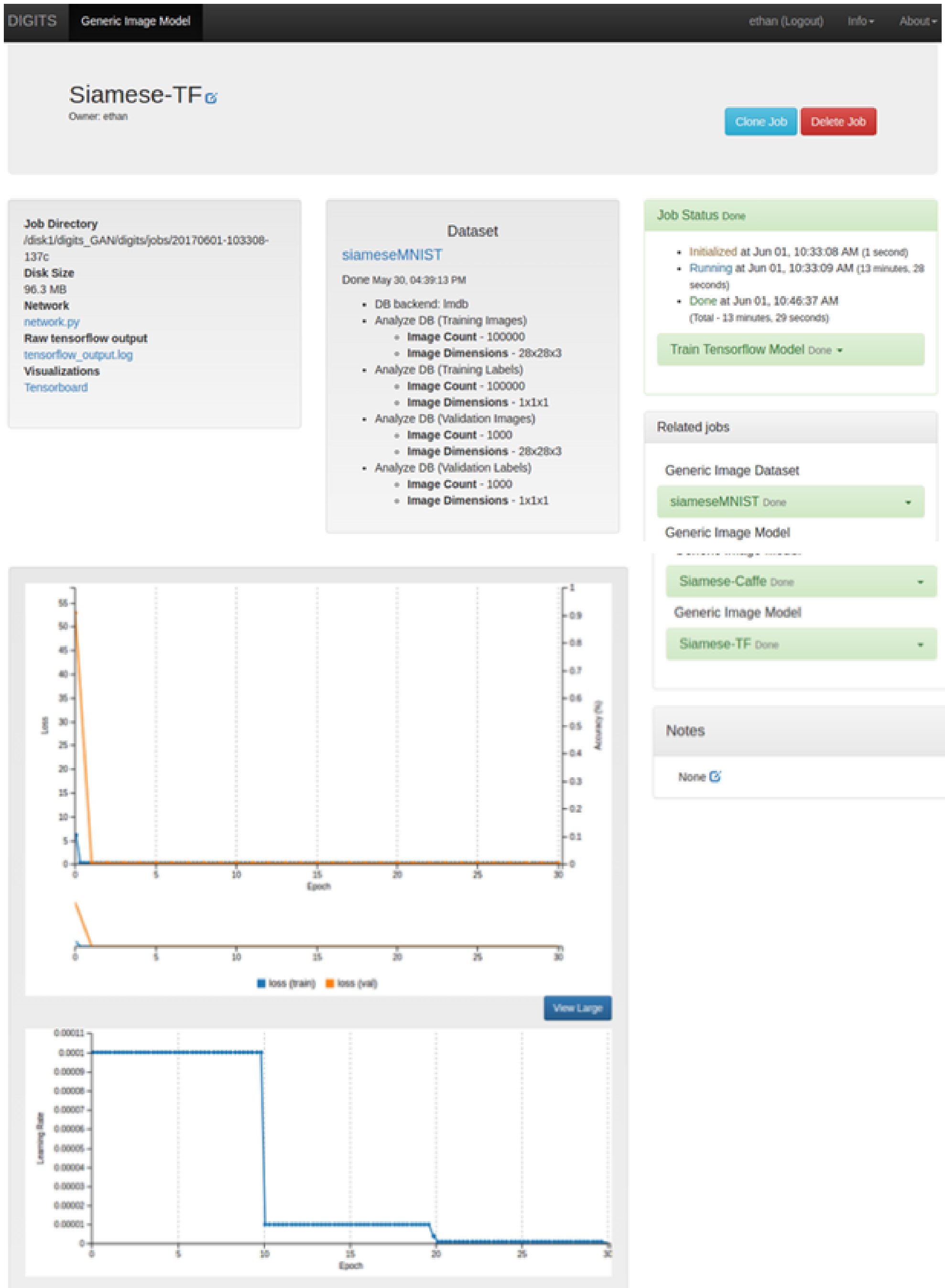






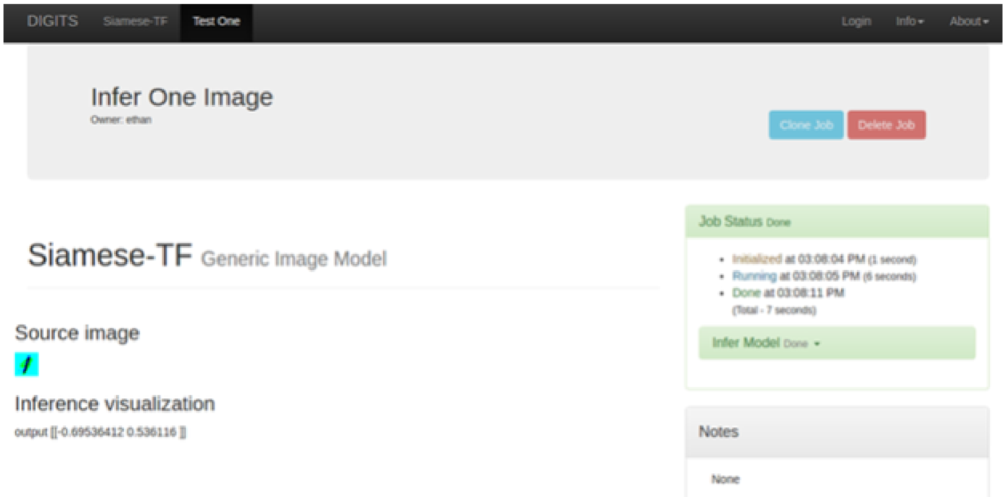


After the model is trained, the graph output should look similar to the following:

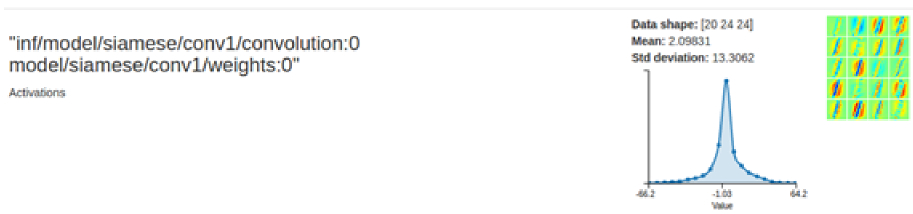


5. Test an image by uploading one from the same directory location that you specified in the `<where to save results>` path.

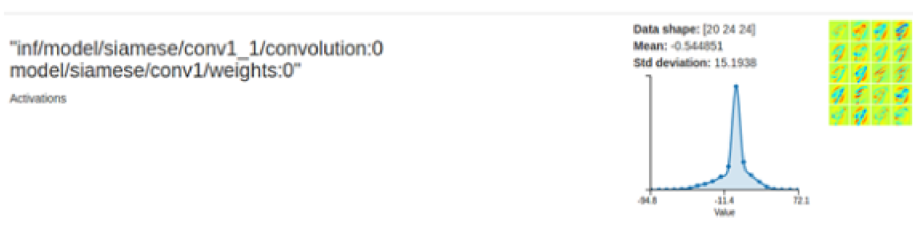
   a). Select the **Show visualization and statistics** check box. In order to ensure that the network was trained correctly and everything worked, scroll down to verify the results.



i. Near the top, there is an activation result which highlights one of the numbers that exists in the image. In this example, you will see that the number 1 is highlighted.



ii. Scroll down to see another activation result which highlights the other number that exists in the image. In this example, you will see that the number 9 is highlighted.



Chapter 5. Support

For the latest Release Notes, see the DIGITS Release Notes Documentation website: (http://docs.nvidia.com/deeplearning/digits/digits-release-notes/index.html).

For more information about DIGITS, see:

- DIGITS website (https://developer.nvidia.com/digits)
- DIGITS project (https://github.com/NVIDIA/DIGITS/blob/digits-5.0/README.md)
- GitHub documentation (https://github.com/NVIDIA/nvidia-docker/wiki/DIGITS)

Note: There may be slight variations between the NVIDIA-docker images and this image.

NVIDIA Corporation | 2788 San Tomas Expressway, Santa Clara, CA 95051
http://www.nvidia.com

NVIDIA Corporation | 2788 San Tomas Expressway, Santa Clara, CA 95051
http://www.nvidia.com