NVIDIA DIGITS

User Guide

DU-08641-001_v6.1 | September 2021

Table of Contents

Chapter 1. Overview Of DIGITS

The Deep Learning GPU Training System™ (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 NVCaffe™ and 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.

1.1. Contents Of The DIGITS Application

The container image available in the NVIDIA® GPU Cloud™ (NGC) registry and NVIDIA® DGX™ container registry, `nvcr.io`, is pre-built and installed into the `/usr/local/python/` directory.

DIGITS also includes the TensorFlow deep learning framework.

Chapter 2. Downloading DIGITS

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. Ways To Run DIGITS

About this task

There are two ways you can run DIGITS:

1. Running DIGITS
2. Running DIGITS From Developer Zone

3.1. Running DIGITS

About this task

You can run DIGITS in the following ways:

1. Running DIGITS
2. Running DIGITS From Developer Zone
3. Docker® . For more information, see DIGITS on GitHub.

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:21.04-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, `21.01`.
- `<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.

3.2. Running DIGITS From Developer Zone

About this task

For more information about downloading, running, and using DIGITS, see: NVIDIA DIGITS: Interactive Deep Learning GPU Training System.

3.3. Creating A Dataset Using Data From An S3 Endpoint

About this task

DIGITS can be trained on data that is stored on an S3 endpoint. This can be useful for cases in which data has been stored on a different node and you do not want to manually migrate the data over to the node running DIGITS.

Procedure

1. Load the data into S3. As an example, we will use the following dataset:

```
python -m digits.download_data mnist ~/mnist
```

There is a python script called `upload_s3_data.py` which is provided and can be used to upload these files into a configured S3 endpoint. This script and its accompanying configuration file called `upload_config.cfg` is located in the `digits/digits/tools` directory.

```
[S3 Config]
endpoint = http://your-s3-endpoint.com:80
accesskey = 0123456789abcde
secretkey = PrIclctP80KrMi6+UPO9ZYNrkg6ByFeFRR6484qL
bucket = digits
prefix = mnist
```

Where:

- `endpoint` - Specifies the URL of the endpoint where the S3 data will be stored.
- `accesskey` - The access key which will be used to authenticate your access to the endpoint.
- `secretkey` - The secret key which will be used to authenticate your access to the endpoint.

- `bucket` - The name of the bucket where this data should be stored. If it does not exist, it will be created by the script.
- `prefix` - The prefix which will be pre-pended to all of the key names. This will be used later during the creation of the dataset.

2. After the file is configured, run it using:

```
python upload_s3_data.py ~/mnist
```

> **Note:** Depending heavily on your network speed and the computing resources of the S3 endpoint, the upload process will take quite a bit of time to complete.

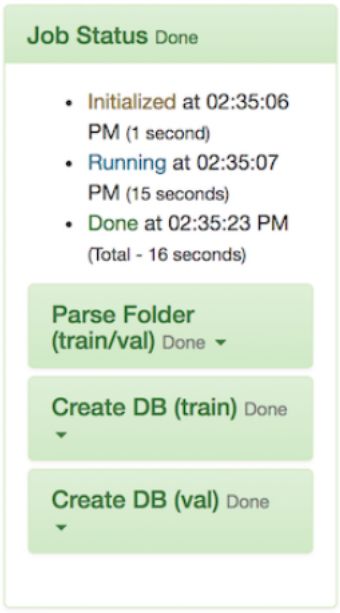
After the upload is complete, all of the keys from the dataset will be uploaded into S3 with the appropriate prefix structure to be used during dataset creation later. For example, in the above configuration, the files would be located in the bucket digits and prefixed with `mnist/train/<0-9>`.

3. Create a dataset within DIGITS.
   a). On the main screen, click **Images > Classification**.
   b). Click the **Use S3 tab** to specify that you want the data to be accessed from an S3 endpoint.

> **Note:** The **Training Images URL** and **Bucket Name** fields may be filled out from the upload configuration fields endpoint and bucket, respectively. The **Training Images Path** consists of the prefix specified during the upload appended by `train/`. For our example, it would be `mnist/train/`. The access key and secret key are the credentials which will be used to access the data from the S3 endpoint.

   Similar to any other dataset, the properties including database backend, image encoding, group name, and dataset name may be specified towards the bottom of the screen. When the dataset has been configured the way you want, click **Create**.
   c). If the job processes correctly, then you have successfully created a dataset from data stored in an S3 endpoint. You will see an image similar to the following:

Figure 1. Confirmation of a successfully created dataset from data stored in an S3 endpoint



You can now proceed to use this dataset to train your model.

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/dgx/index.html. For example, the 19.01 release of DIGITS includes the 19.01 release of the 19.01 release of TensorFlow.

DIGITS is a training platform that can be used with TensorFlow deep learning frameworks. Using either 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 6.0 and later.

4.1.1. Example 1: MNIST

Procedure

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 application has been pulled, you can start DIGITS. 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
   --shm-size=1g --ulimit memlock=-1 --ulimit stack=67108864
   nvcr.io/nvidia/digits:17.04
   ```

   There are a number of options in this command.

   - The first option `-d` tells `nvidia-docker` to run the application in "daemon" mode.
   - The `--name` option names the running application (we will need this later).

- The two `ulimit` options and the `shmem` option are to increase the amount of memory for TensorFlow since it shares data across GPUs using shared memory.
- 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 below.

```
$ 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 application. The DIGITS application has a simple script for downloading the data set into the application. As a check, run the following command to make sure the application 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 application 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.

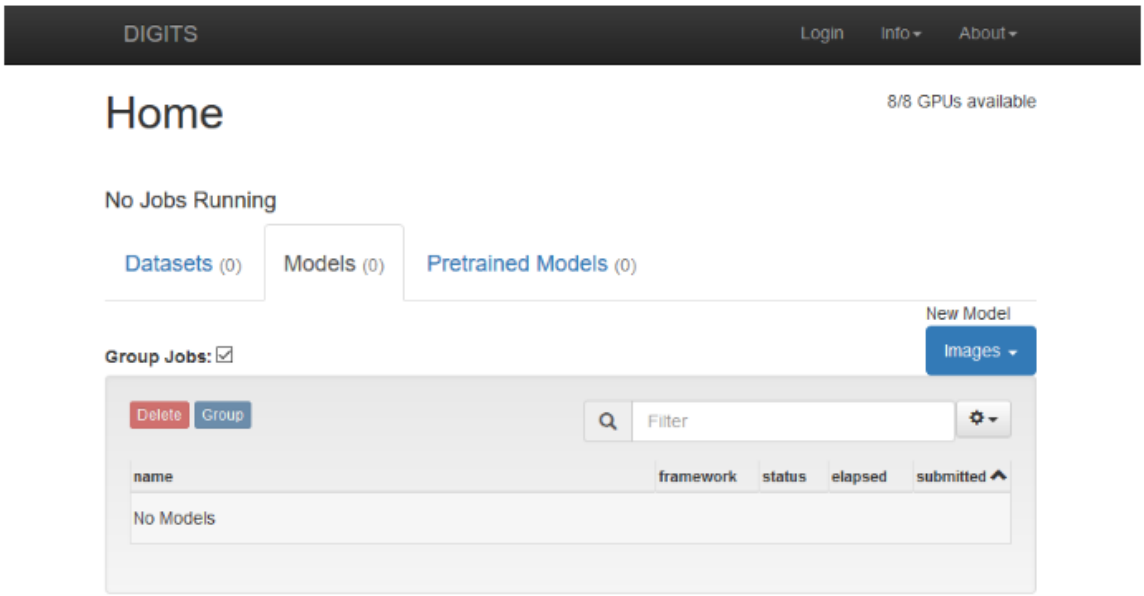
```
# python -m digits.download_data mnist /data/mnist
Downloading url=http://yann.lecun.com/exdb/mnist/train-images-idx3-ubyte.gz ...
Downloading url=http://yann.lecun.com/exdb/mnist/train-labels-idx1-ubyte.gz ...
Downloading url=http://yann.lecun.com/exdb/mnist/t10k-images-idx3-ubyte.gz ...
Downloading 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
```

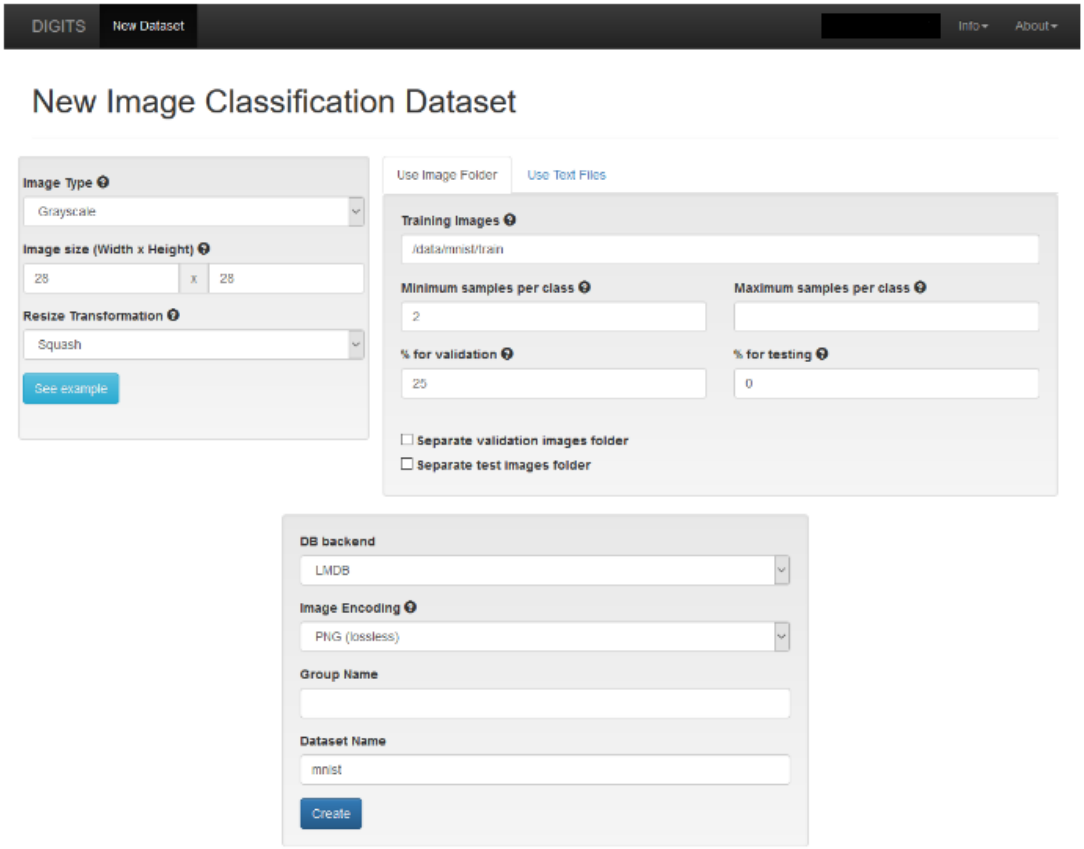
   On the home page of DIGITS, in the top right corner it says that there are 8 of 8 GPUs available on this DGX-1. For a DGX Station this should be 4 or 4 GPUs available. For NVIDIA NGC Cloud Services on a cloud provider, the number of GPUs should match the number for the instance type.

Figure 2. DIGITS home page



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.

Figure 3. New Image Classification Dataset



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 TensorFlow to load the datasets. After the datasets are loaded, your screen should look similar to the following.

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

Figure 4. MNIST top level

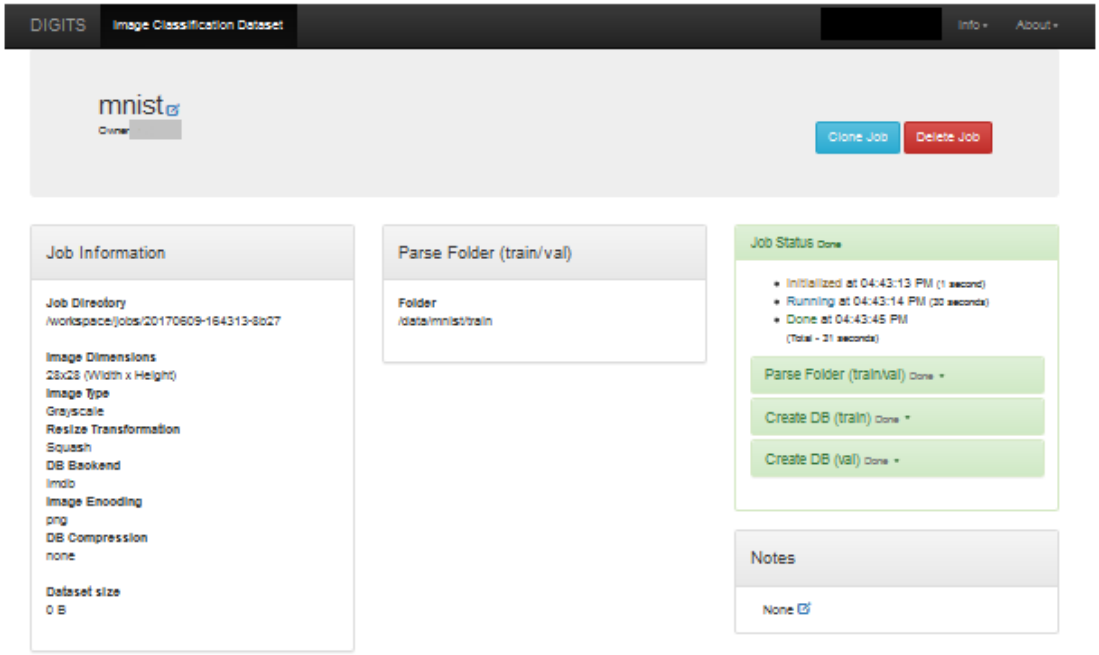


Figure 5. MNIST lower level

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 are 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.

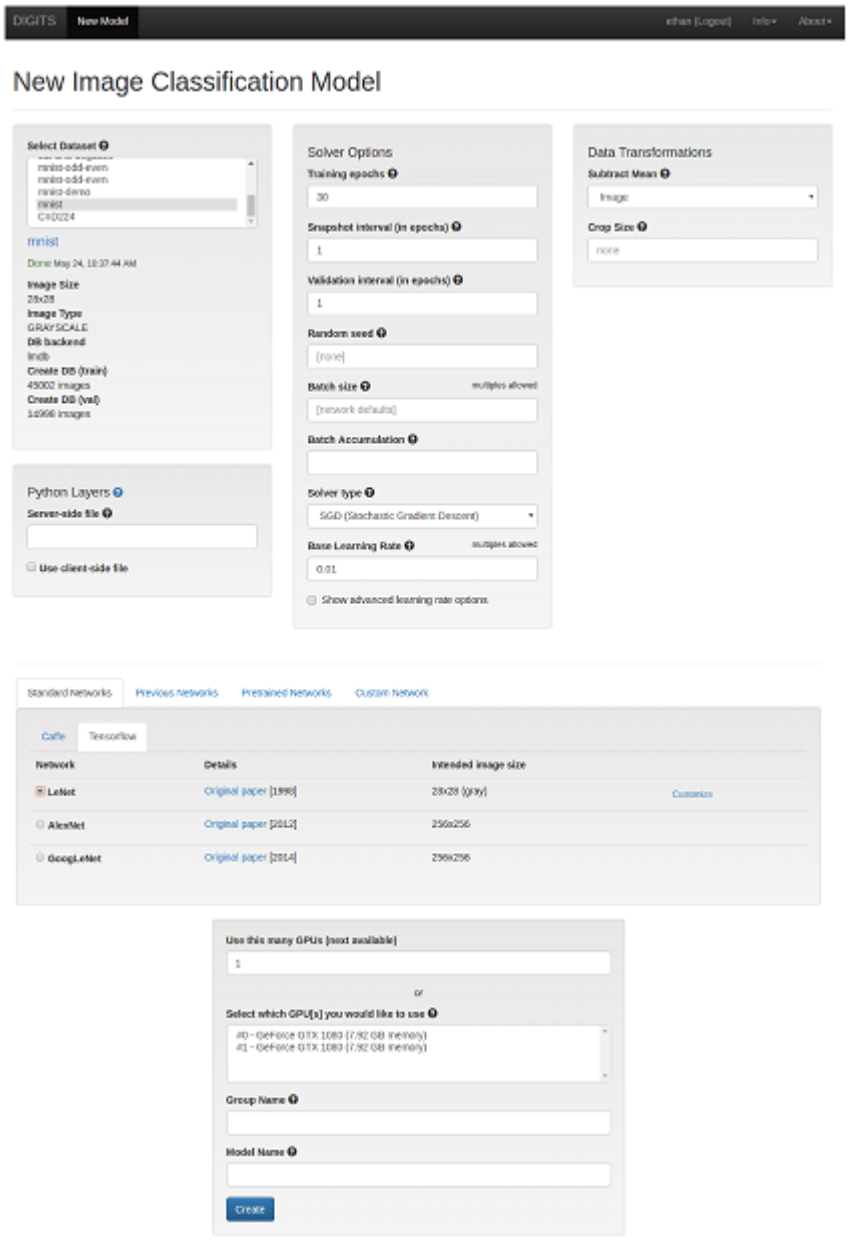
e). In the **Standard Networks** tab, click **TensorFlow** and select the **LeNet** radio button.

Note: DIGITS allows you to use previous networks, pre-trained networks, and customer networks if you want.

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

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

Figure 6. New Image Classification Model top level



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.

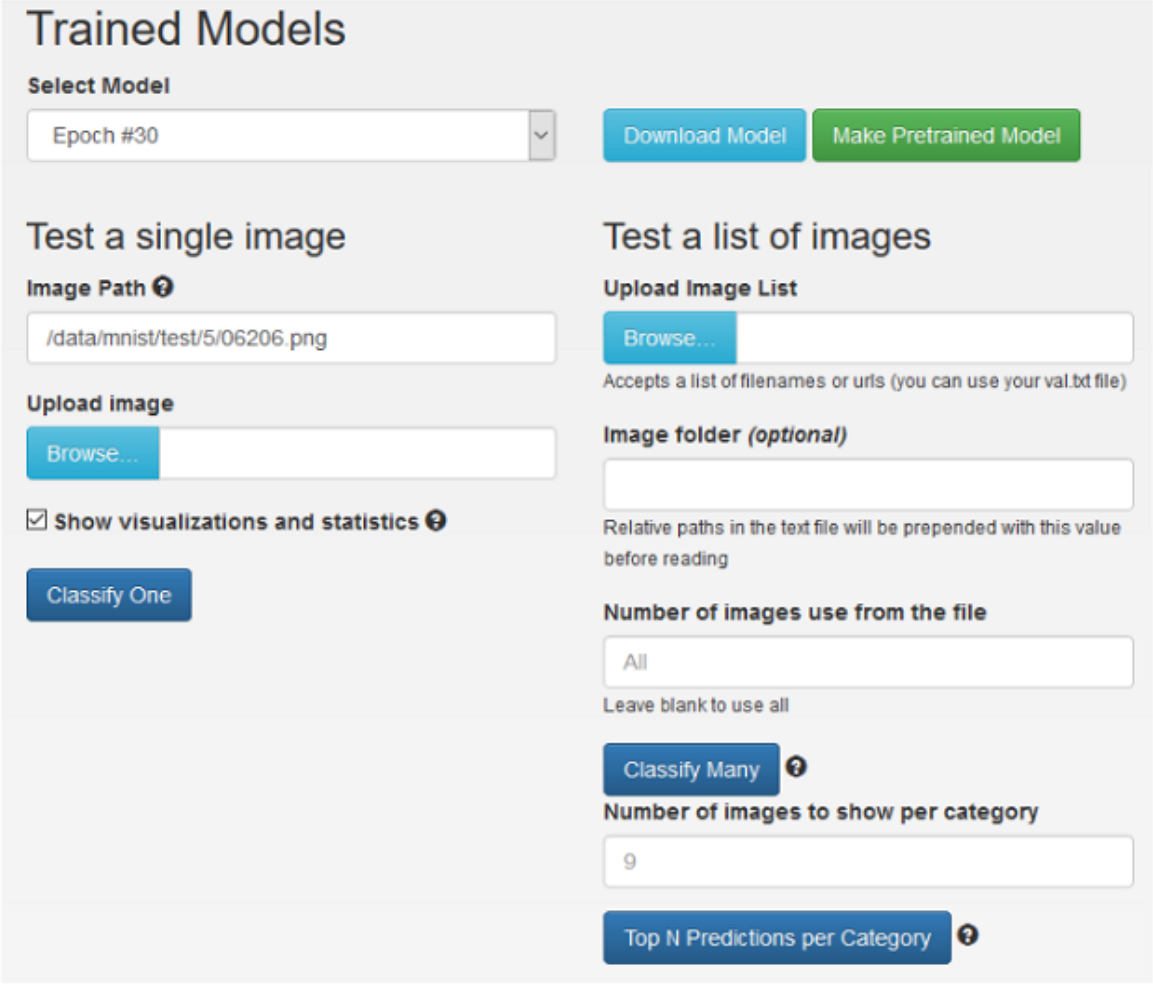
Note: This screen capture has been truncated because the web page is very long.

Figure 7. Image Classification Model

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.

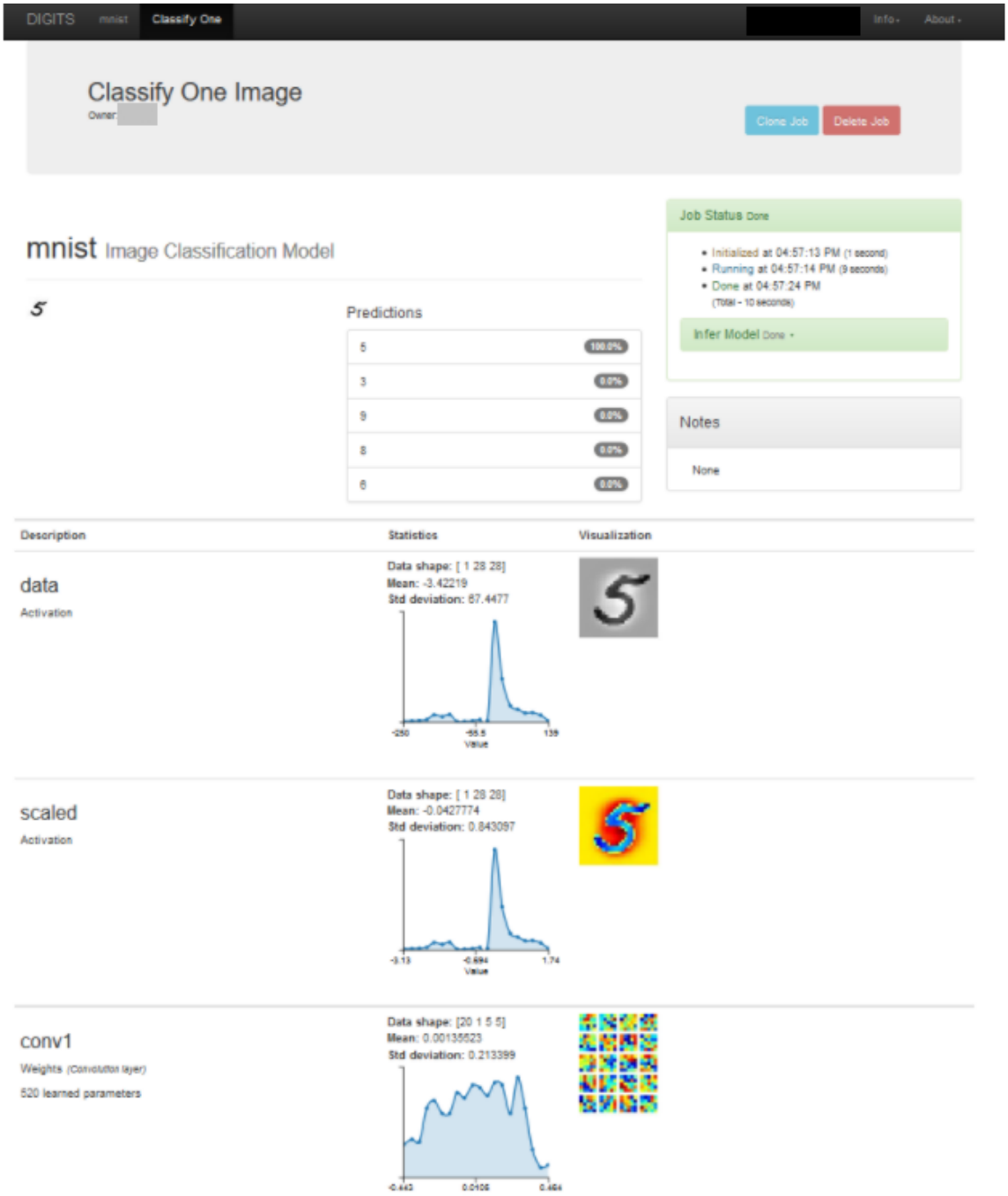
Figure 8. Trained Models



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".

Figure 9. Classify One Image

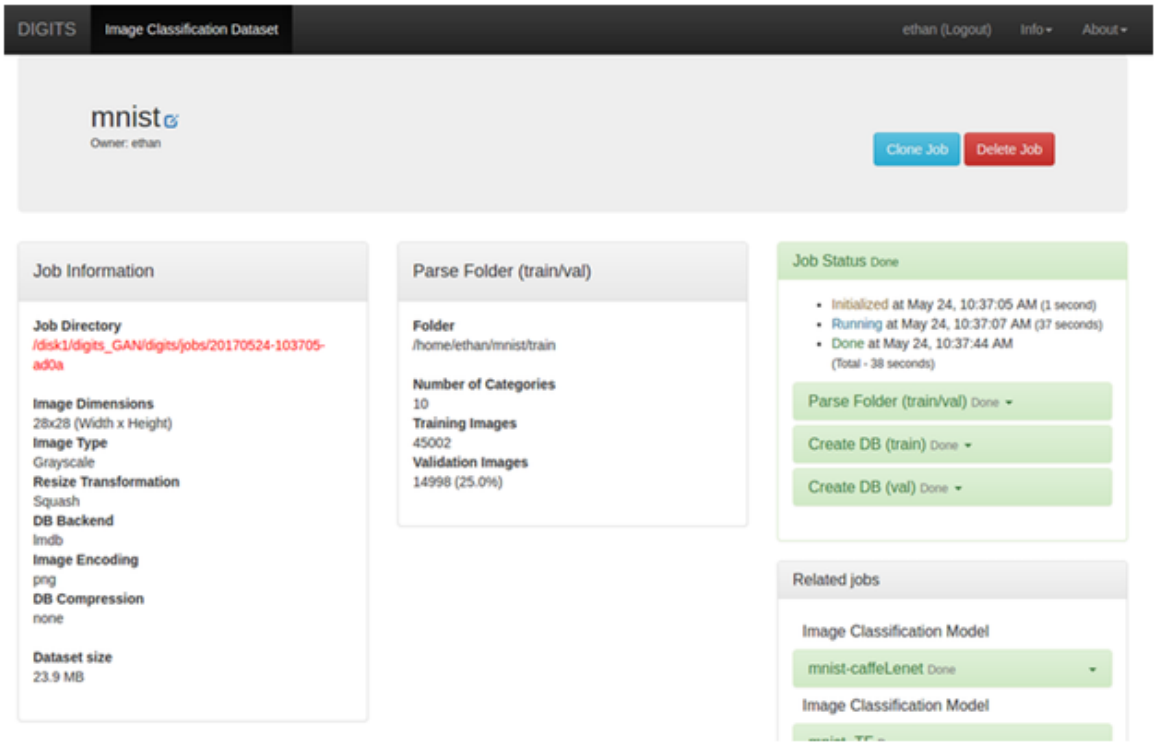


4.1.2. Example 2: Siamese Network

Before you begin

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

Figure 10. Job directory



Procedure

1. Run the Python script available at: GitHub: siamese-TF.py. The script requires the following parameters:

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

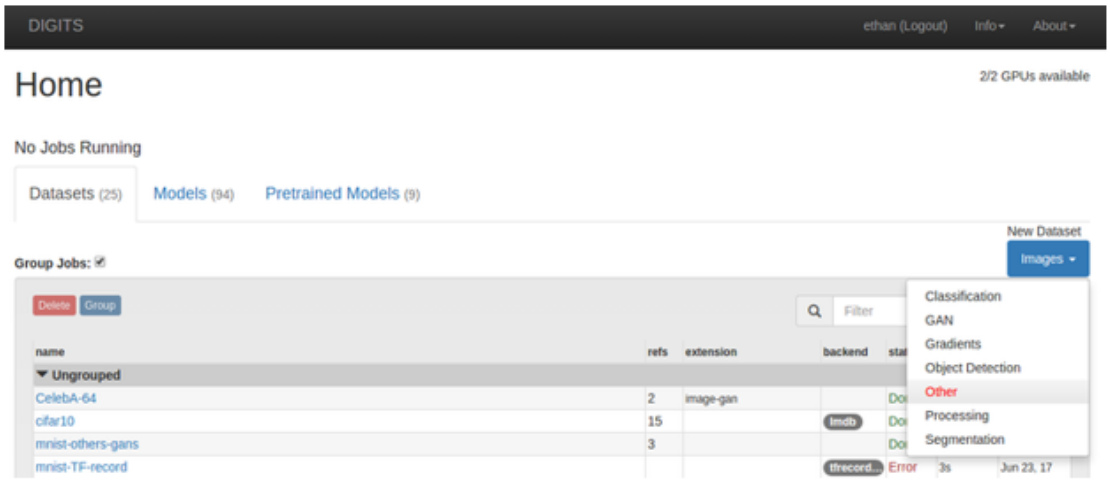
   Where:

   - `<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**.

Figure 11. New dataset

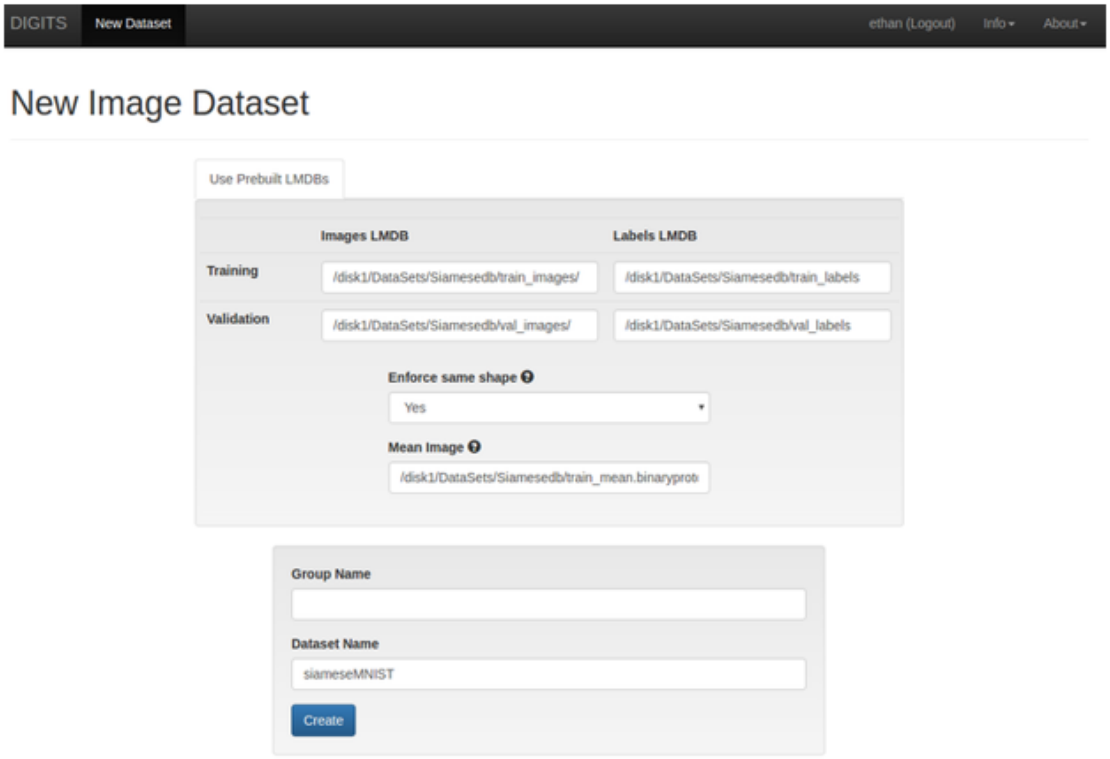


b). Provide the directory paths to the following fields:

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

- The train image database
- The train label database
- The validation image database
- The validation label database
- The train image `train_mean.binaryproto` file

Figure 12. New image dataset



3. Click **New Model > Images > Other** to create the model. In this example, we will use TensorFlow to train our siamese network.
4. Train the model.
   a). Click the **Custom Network** tab and select **TensorFlow**.
   b). Copy and paste the following network definition: GitHub: mnist_siamese_train_test.prototxt
   c). Ensure the Base Learning Rate is set to `0.01`, keep the default settings to the other fields, and click **Train**.

Figure 13. New image model

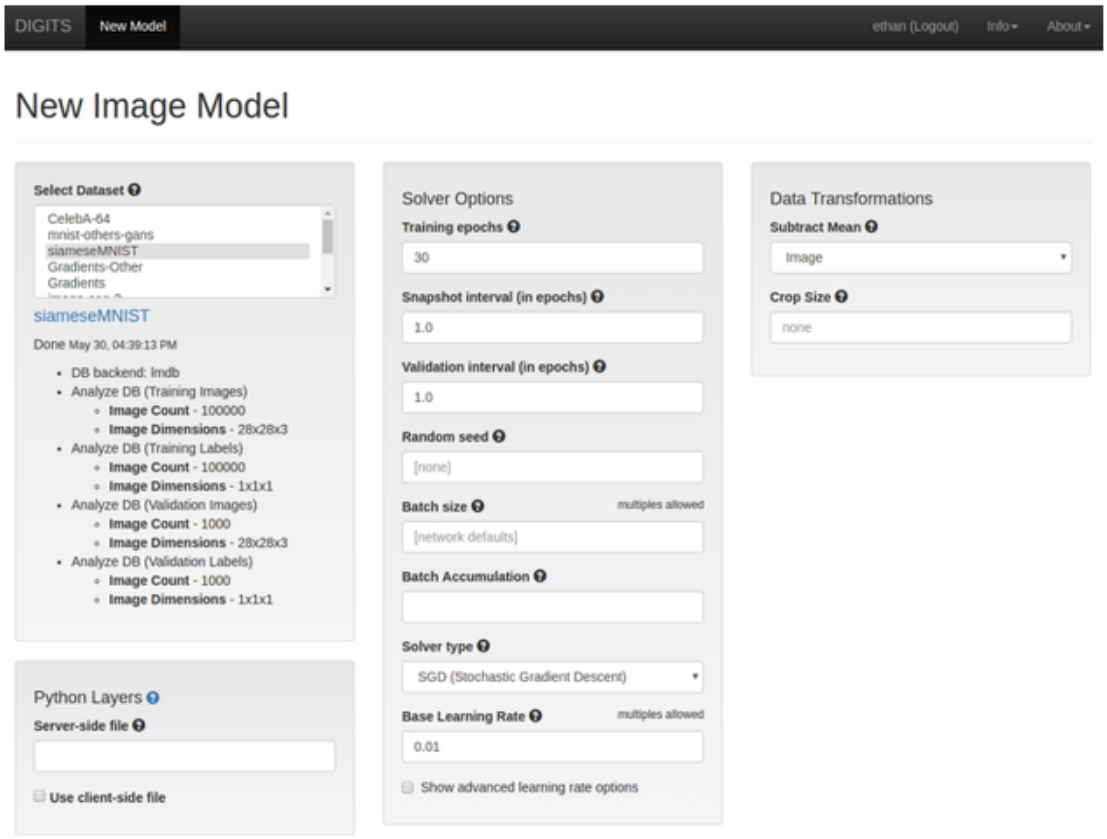


Figure 14. Custom model

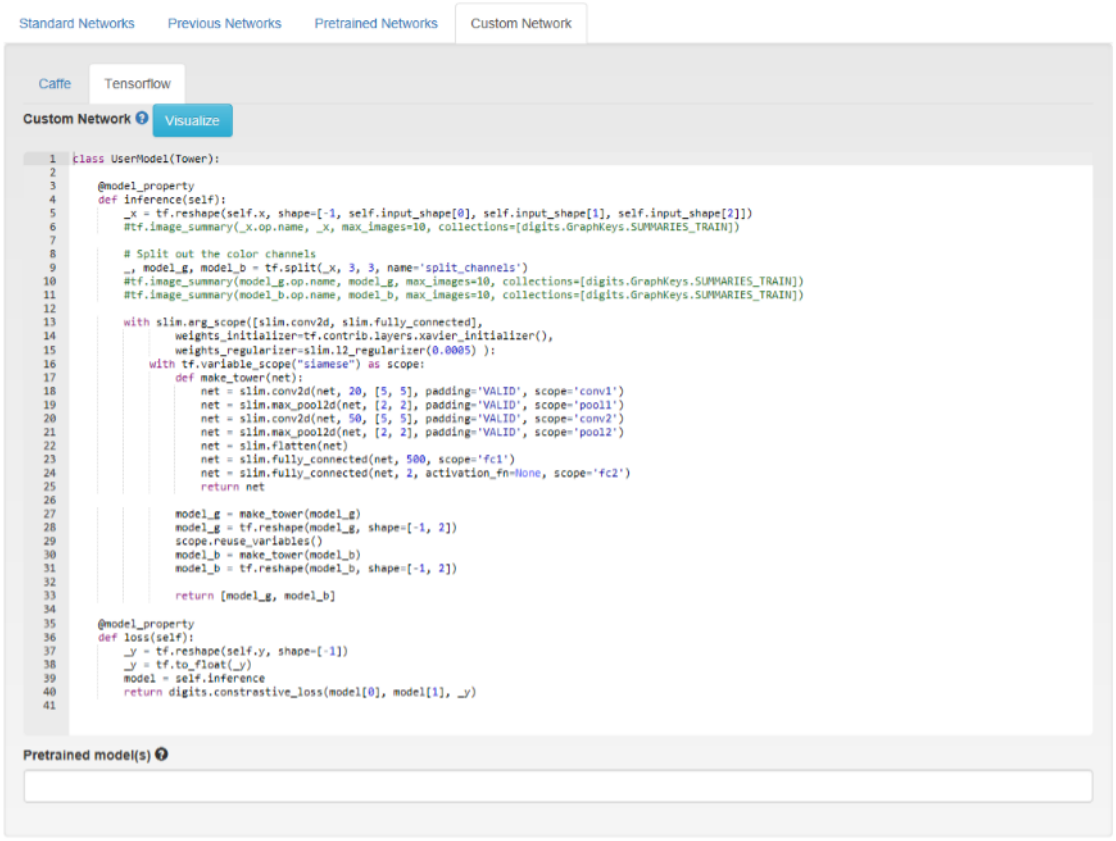
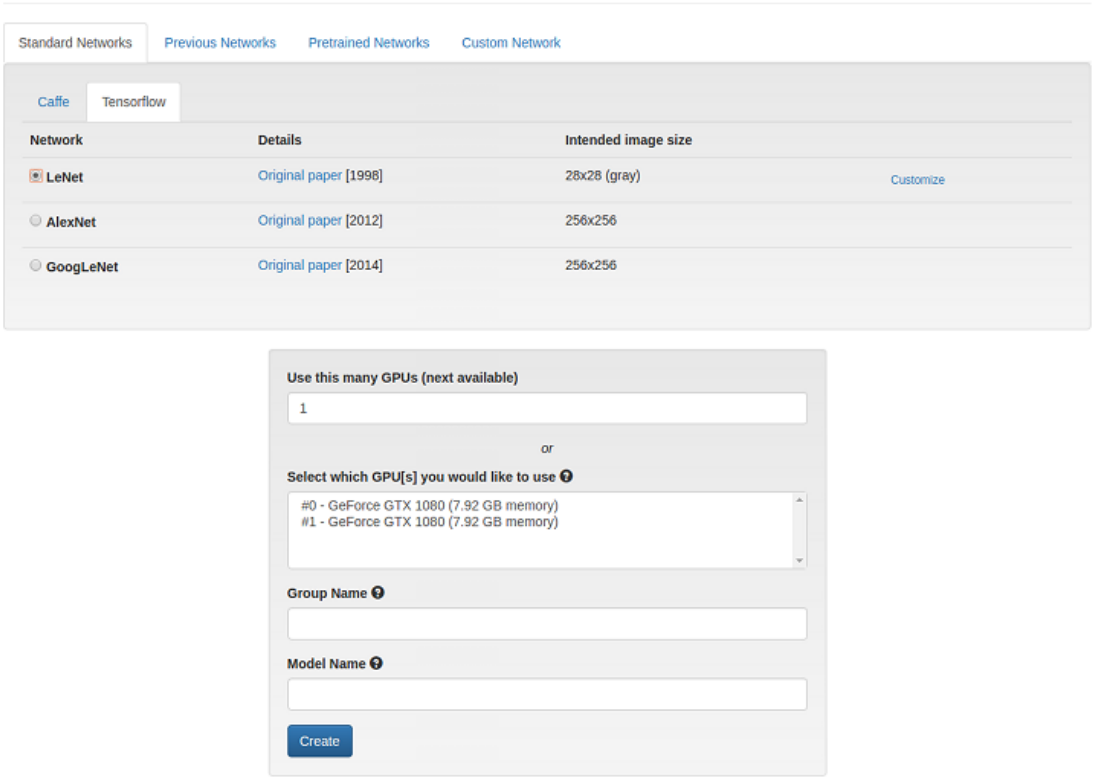
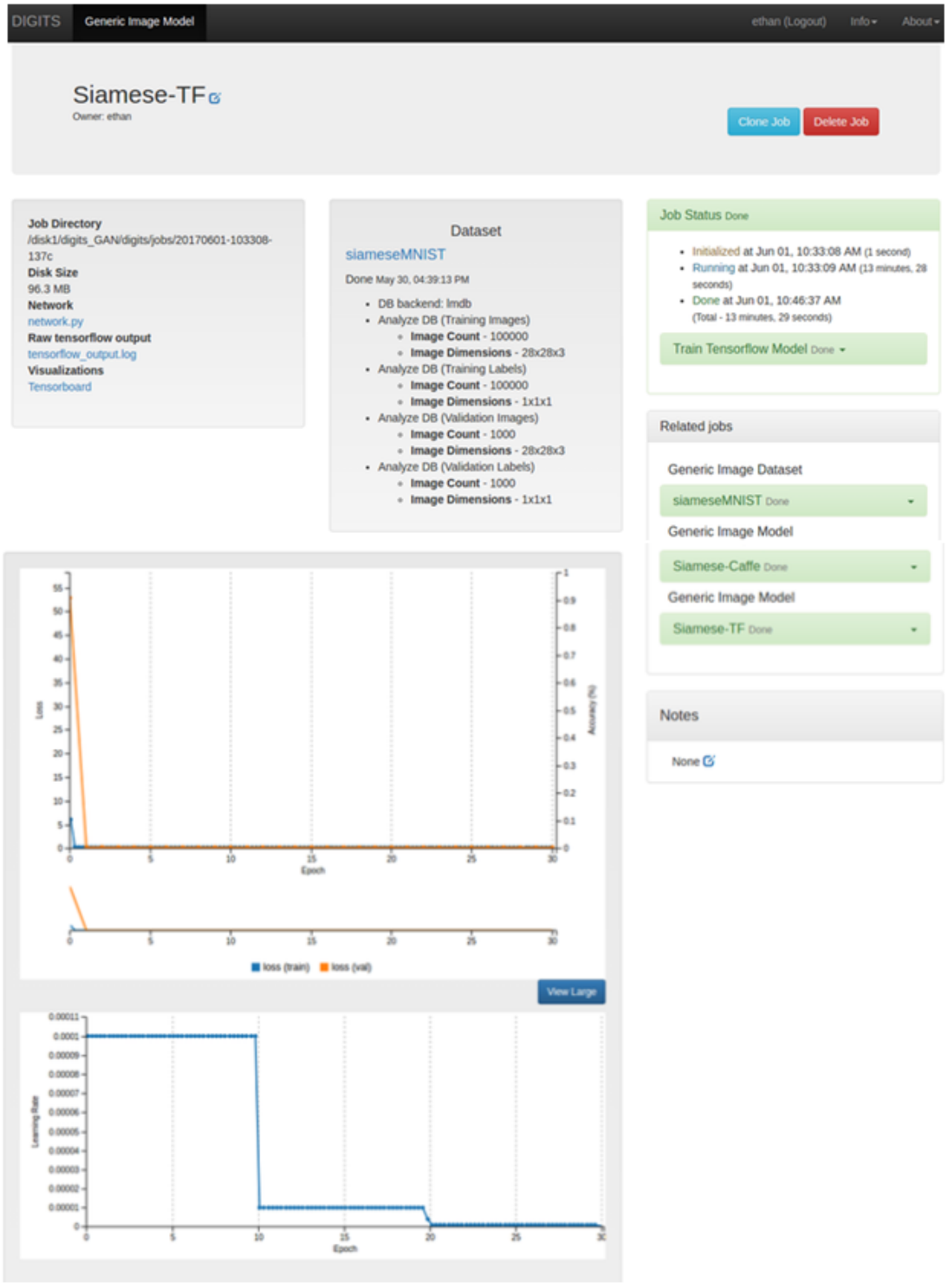


Figure 15. Training on TensorFlow



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

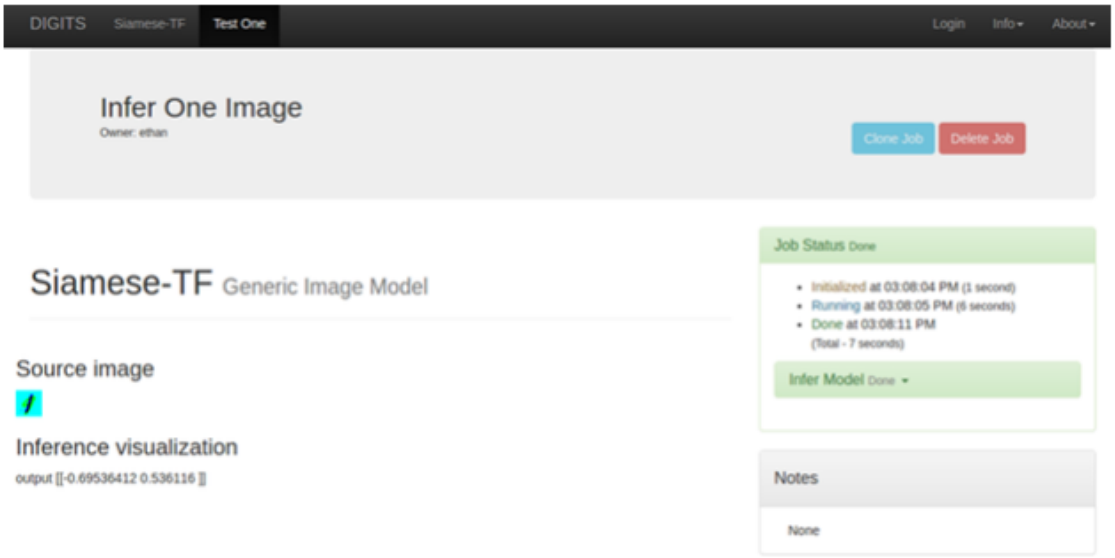
Figure 16. TensorFlow graph output



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

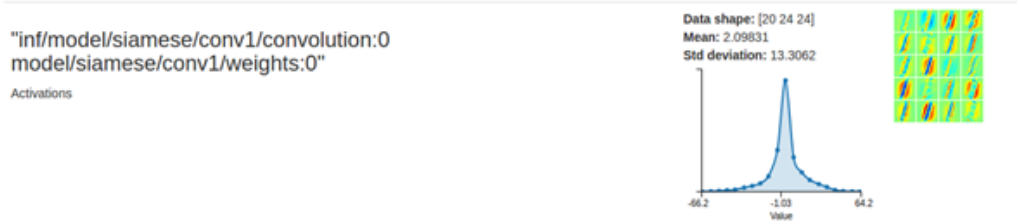
a). Select the **Show visualization and statistics** check box. In order to ensure that the network was trained correctly and everything worked, there are two things you need to confirm are included within the results.

Figure 17. Verify



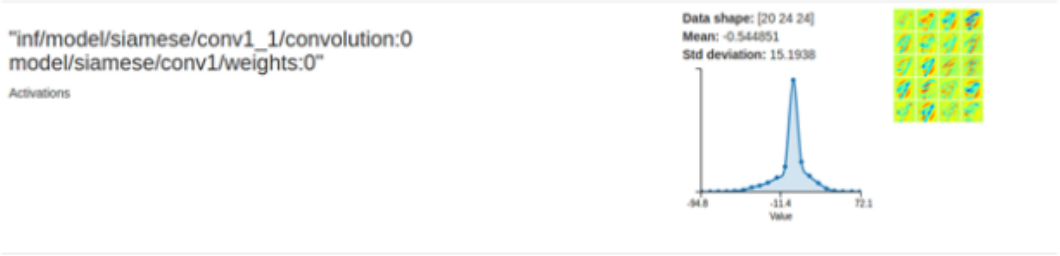
i. Near the top, there is an activation result which highlights one of the numbers that exists in the image. In this example, you will see that the number 1 is highlighted.

Figure 18. Example output



ii. Scroll down to see the inference highlighting the numbers that were seen inside the given image.

Figure 19. Example output



Chapter 5. Examples

Here are some helpful code samples.

5.1. Adjusting the model to input dimensions and number of classes

The following network defines a linear network that takes any 3D-tensor as input and produces one categorical output per class:

```
return function(p)
    -- model should adjust to any 3D-input
    local nClasses = p.nclasses or 1
    local nDim = 1
    if p.inputShape then p.inputShape:apply(function(x) nDim=nDim*x end) end
    local model = nn.Sequential()
    model:add(nn.View(-1):setNumInputDims(3)) -- c*h*w -> chw (flattened)
    model:add(nn.Linear(nDim, nclasses)) -- chw -> nClasses
    model:add(nn.LogSoftMax())
    return {
        model = model
    }
end
```

5.2. Selecting the nn backend

Convolution layers are supported by a variety of backends (e.g. nn, cunn, cudnn, ...). The following code snippet shows how to select between nn, cunn, cudnn based on their availability in the system:

```
if pcall(function() require('cudnn') end) then
   backend = cudnn
   convLayer = cudnn.SpatialConvolution
else
   pcall(function() require('cunn') end)
   backend = nn -- works with cunn or nn
   convLayer = nn.SpatialConvolutionMM
end
local net = nn.Sequential()
lenet:add(backend.SpatialConvolution(1,20,5,5,1,1,0)) -- 1*28*28 -> 20*24*24
lenet:add(backend.SpatialMaxPooling(2, 2, 2, 2)) -- 20*24*24 -> 20*12*12
lenet:add(backend.SpatialConvolution(20,50,5,5,1,1,0)) -- 20*12*12 -> 50*8*8
lenet:add(backend.SpatialMaxPooling(2,2,2,2)) --  50*8*8 -> 50*4*4
lenet:add(nn.View(-1):setNumInputDims(3))  -- 50*4*4 -> 800
```

```
lenet:add(nn.Linear(800,500))  -- 800 -> 500
lenet:add(backend.ReLU())
lenet:add(nn.Linear(500, 10))  -- 500 -> 10
lenet:add(nn.LogSoftMax())
```

5.3. Supervised regression learning

In supervised regression learning, labels may not be scalars like in classification learning. To learn a regression model, a generic dataset may be created using one database for input samples and one database for labels (only 1D row label vectors are supported presently). The appropriate loss function must be specified using the loss internal parameters. For example the following snippet defines a simple regression model on 1x10x10 images using MSE loss:

```
local net = nn.Sequential()
net:add(nn.View(-1):setNumInputDims(3))  -- 1*10*10 -> 100
net:add(nn.Linear(100,2))
return function(params)
    return {
        model = net,
        loss = nn.MSECriterion(),
    }
end
```

5.4. Command Line Inference

DIGITS Lua wrappers may also be used from command line. For example, to classify an image test.png using the snapshot at epoch 30 of a model job 20160707-093158-9ed6 using a dataset 20160117-131355-ba71:

```
th /home/greg/ws/digits/tools/torch/wrapper.lua test.lua --image=test.png
 --network=model --networkDirectory=/home/greg/ws/digits/digits/
jobs/20160707-093158-9ed6 --snapshot=/home/greg/ws/digits/digits/
jobs/20160707-093158-9ed6/snapshot_30_Model.t7 --labels=/home/greg/ws/digits/
digits/jobs/20160117-131355-ba71/labels.txt --mean=/home/greg/ws/digits/digits/
jobs/20160707-093158-9ed6/mean.jpg --subtractMean=image
2016-07-07 09:43:20 [INFO ] Loading mean tensor from /home/greg/ws/digits/digits/
jobs/20160707-093158-9ed6/mean.jpg file
2016-07-07 09:43:20 [INFO ] Loading network definition from /home/greg/ws/digits/
digits/jobs/20160707-093158-9ed6/model
Using CuDNN backend
2016-07-07 09:43:20 [INFO ] For image 1, predicted class 1: 5 (4) 0.99877911806107
2016-07-07 09:43:20 [INFO ] For image 1, predicted class 2: 10 (9)
 0.0011787103721872
2016-07-07 09:43:20 [INFO ] For image 1, predicted class 3: 8 (7)
 3.8778140151408e-05
2016-07-07 09:43:20 [INFO ] For image 1, predicted class 4: 7 (6)
 2.3879254058556e-06
2016-07-07 09:43:20 [INFO ] For image 1, predicted class 5: 1 (0)
 4.1431232489231e-07
```

For classification networks, the Top-5 classes are shown. For each class, the label is shown within brackets. For example predicted class 1: 5 (4) 0.99877911806107 means that the

network predicted the most likely class to be the 5th class in labels.txt with label "4" and probability 99.88%.

For other types of networks, set --allPredictions=yes on the command line to display the raw network output. For example:

```
th /home/greg/ws/digits/tools/torch/wrapper.lua test.lua --image=test.png
 --network=model --networkDirectory=/home/greg/ws/digits/digits/
jobs/20160707-093158-9ed6 --snapshot=/home/greg/ws/digits/digits/
jobs/20160707-093158-9ed6/snapshot_30_Model.t7 --mean=/home/greg/ws/digits/digits/
jobs/20160707-093158-9ed6/mean.jpg --subtractMean=image --allPredictions=yes
2016-07-07 09:46:31 [INFO ] Loading mean tensor from /home/greg/ws/digits/digits/
jobs/20160707-093158-9ed6/mean.jpg file
2016-07-07 09:46:31 [INFO ] Loading network definition from /home/greg/ws/digits/
digits/jobs/20160707-093158-9ed6/model
Using CuDNN backend
2016-07-07 09:46:32 [INFO ] Predictions for image 1:
 [-14.696645736694,-16.256759643555,-16.247381210327,-20.25181388855,-0.0012216567993164,-18.05564
```

5.5. Multi-GPU training

Data parallelism is supported in Torch7 by cunn through the DataParallelTable module. DIGITS provides the number of available GPUs through the ngpus external parameter.

Assuming net is a container that encapsulates the definition of a network, the following snippet may be used to enable data parallelism into a container called `model`:

```
local model
if ngpus>1 then
   model = nn.DataParallelTable(1)  -- Split along first (batch) dimension
   for i = 1, ngpus do
      cutorch.setDevice(i)
      model:add(net:clone(), i)  -- Use the ith GPU
   end
   cutorch.setDevice(1)  -- This is the 'primary' GPU
else
   model = net
end
```

Chapter 6. Troubleshooting

6.1. Support

For the latest Release Notes, see the DIGITS Release Notes Documentation website.

For more information about DIGITS, see:

- DIGITS website
- DIGITS 6.0 project
- GitHub documentation

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

Notice

This document is provided for information purposes only and shall not be regarded as a warranty of a certain functionality, condition, or quality of a product. NVIDIA Corporation ("NVIDIA") makes no representations or warranties, expressed or implied, as to the accuracy or completeness of the information contained in this document and assumes no responsibility for any errors contained herein. NVIDIA shall have no liability for the consequences or use of such information or for any infringement of patents or other rights of third parties that may result from its use. This document is not a commitment to develop, release, or deliver any Material (defined below), code, or functionality.

NVIDIA reserves the right to make corrections, modifications, enhancements, improvements, and any other changes to this document, at any time without notice.

Customer should obtain the latest relevant information before placing orders and should verify that such information is current and complete.

NVIDIA products are sold subject to the NVIDIA standard terms and conditions of sale supplied at the time of order acknowledgement, unless otherwise agreed in an individual sales agreement signed by authorized representatives of NVIDIA and customer ("Terms of Sale"). NVIDIA hereby expressly objects to applying any customer general terms and conditions with regards to the purchase of the NVIDIA product referenced in this document. No contractual obligations are formed either directly or indirectly by this document.

NVIDIA products are not designed, authorized, or warranted to be suitable for use in medical, military, aircraft, space, or life support equipment, nor in applications where failure or malfunction of the NVIDIA product can reasonably be expected to result in personal injury, death, or property or environmental damage. NVIDIA accepts no liability for inclusion and/or use of NVIDIA products in such equipment or applications and therefore such inclusion and/or use is at customer's own risk.

NVIDIA makes no representation or warranty that products based on this document will be suitable for any specified use. Testing of all parameters of each product is not necessarily performed by NVIDIA. It is customer's sole responsibility to evaluate and determine the applicability of any information contained in this document, ensure the product is suitable and fit for the application planned by customer, and perform the necessary testing for the application in order to avoid a default of the application or the product. Weaknesses in customer's product designs may affect the quality and reliability of the NVIDIA product and may result in additional or different conditions and/or requirements beyond those contained in this document. NVIDIA accepts no liability related to any default, damage, costs, or problem which may be based on or attributable to: (i) the use of the NVIDIA product in any manner that is contrary to this document or (ii) customer product designs.

No license, either expressed or implied, is granted under any NVIDIA patent right, copyright, or other NVIDIA intellectual property right under this document. Information published by NVIDIA regarding third-party products or services does not constitute a license from NVIDIA to use such products or services or a warranty or endorsement thereof. Use of such information may require a license from a third party under the patents or other intellectual property rights of the third party, or a license from NVIDIA under the patents or other intellectual property rights of NVIDIA.

Reproduction of information in this document is permissible only if approved in advance by NVIDIA in writing, reproduced without alteration and in full compliance with all applicable export laws and regulations, and accompanied by all associated conditions, limitations, and notices.

THIS DOCUMENT AND ALL NVIDIA DESIGN SPECIFICATIONS, REFERENCE BOARDS, FILES, DRAWINGS, DIAGNOSTICS, LISTS, AND OTHER DOCUMENTS (TOGETHER AND SEPARATELY, "MATERIALS") ARE BEING PROVIDED "AS IS." NVIDIA MAKES NO WARRANTIES, EXPRESSED, IMPLIED, STATUTORY, OR OTHERWISE WITH RESPECT TO THE MATERIALS, AND EXPRESSLY DISCLAIMS ALL IMPLIED WARRANTIES OF NONINFRINGEMENT, MERCHANTABILITY, AND FITNESS FOR A PARTICULAR PURPOSE. TO THE EXTENT NOT PROHIBITED BY LAW, IN NO EVENT WILL NVIDIA BE LIABLE FOR ANY DAMAGES, INCLUDING WITHOUT LIMITATION ANY DIRECT, INDIRECT, SPECIAL, INCIDENTAL, PUNITIVE, OR CONSEQUENTIAL DAMAGES, HOWEVER CAUSED AND REGARDLESS OF THE THEORY OF LIABILITY, ARISING OUT OF ANY USE OF THIS DOCUMENT, EVEN IF NVIDIA HAS BEEN ADVISED OF THE POSSIBILITY OF SUCH DAMAGES. Notwithstanding any damages that customer might incur for any reason whatsoever, NVIDIA's aggregate and cumulative liability towards customer for the products described herein shall be limited in accordance with the Terms of Sale for the product.

VESA DisplayPort

DisplayPort and DisplayPort Compliance Logo, DisplayPort Compliance Logo for Dual-mode Sources, and DisplayPort Compliance Logo for Active Cables are trademarks owned by the Video Electronics Standards Association in the United States and other countries.

HDMI

HDMI, the HDMI logo, and High-Definition Multimedia Interface are trademarks or registered trademarks of HDMI Licensing LLC.

OpenCL

OpenCL is a trademark of Apple Inc. used under license to the Khronos Group Inc.

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

Trademarks

NVIDIA, the NVIDIA logo, and cuBLAS, CUDA, cuDNN, DALI, DIGITS, DGX, DGX-1, DGX-2, DGX Station, DLProf, Jetson, Kepler, Maxwell, NCCL, Nsight Compute, Nsight Systems, NvCaffe, PerfWorks, Pascal, SDK Manager, Tegra, TensorRT, Triton Inference Server, Tesla, TF-TRT, and Volta are trademarks and/or registered trademarks of NVIDIA Corporation in the U.S. and other countries. Other company and product names may be trademarks of the respective companies with which they are associated.

Copyright

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