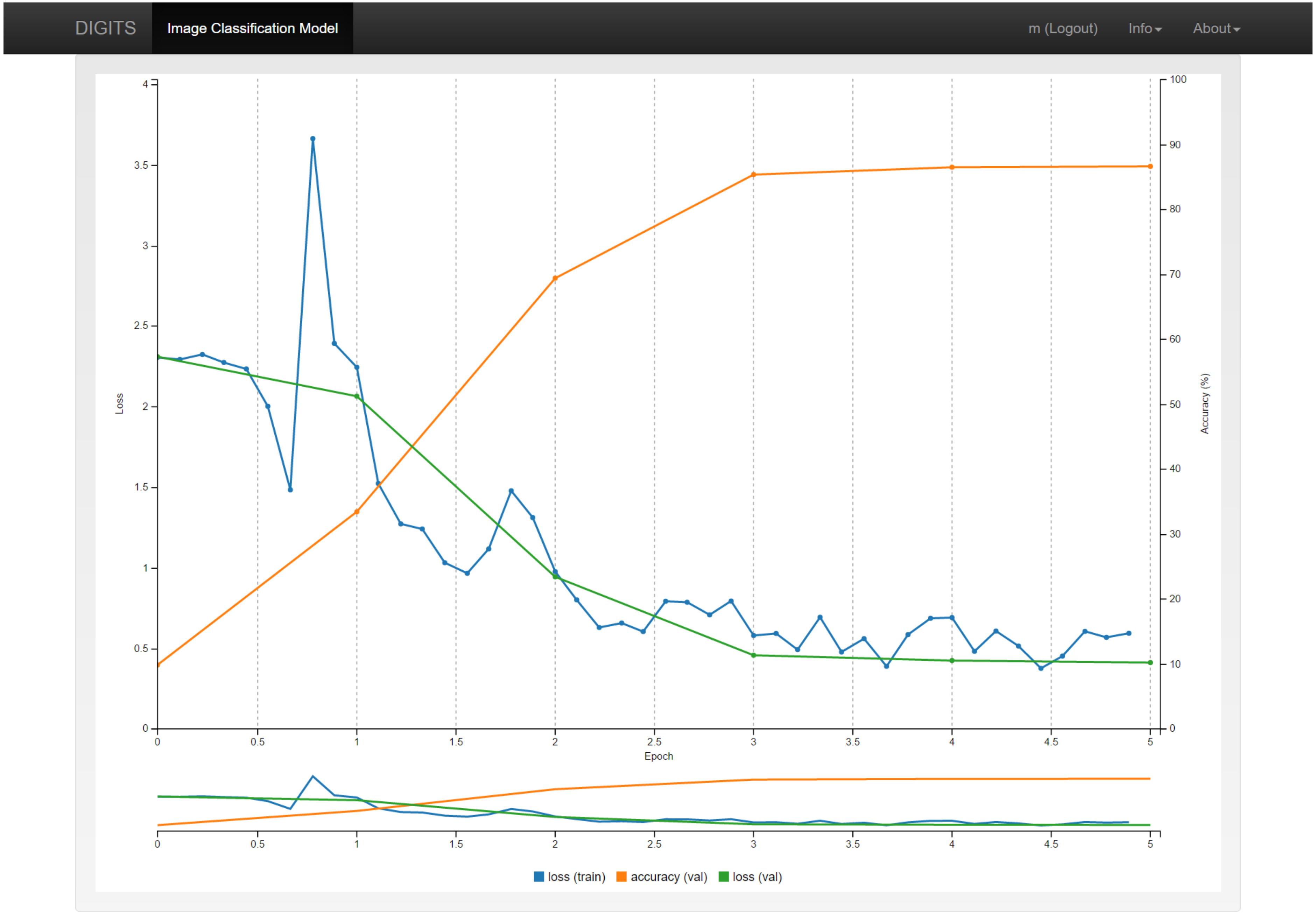
(https://www.nvidia.com/en-us/deep-learning-ai/education/)

## Improving your Model

Now that you've learned to successfully train a model, let's work to train a state of the art model. In this lab, we'll learn the levers that you, as a deep learning practitioner, will use to navigate towards desired results. In the process, we'll start to peel back the layers around the technology that makes this possible.

Let's bring back our handwritten digit classifier. Go to DIGITS' home screen by clicking the DIGITS logo on the top left of the screen. Here, you'll see (at least) two models. Choose the first model you created, in our case, "My first model."

Among other things, DIGITS will display the graph that was generated as the model was being trained.


DIGITS
Image Classification Model
m (Logout)
Info
About
Loss
Accuracy (%)
Epoch
0
0.5
1
1.5
2
2.5
3
3.5
4
4.5
5
10
20
30
40
50
60
70
80
90
100
🟦 loss (train)
🟧 accuracy (val)
🟩 loss (val)

Three quantities are reported: training loss, validation loss, and accuracy. The values of training and validation loss should have decreased from epoch to epoch, although they may jump around some. The accuracy is the measure of the ability of the model to correctly classify the validation data. If you hover your mouse over any of the data points, you will see its exact value. In this case, the accuracy at the last epoch is about 87%. Your results might be slightly different than what is shown here, since the initial networks are generated randomly.

Analyzing this graph, one thing that jumps out is that accuracy is increasing over time and that loss is decreasing. A natural question may be, "will the model keep improving if we let it train longer?" This is the first intervention that we'll experiment with and discuss.

## Study more

Following the advice of parents and teachers everywhere, let's work to improve the accuracy of our model by asking it to study more.

An **epoch** is one complete presentation of the data to be learned to a learning machine. Let's make sense of what is happening during an **epoch.**

1. Neural networks take the first image (or small group of images) and make a prediction about what it is (or they are).
2. Their prediction is compared to the actual label of the image(s).
3. The network uses information about the difference between their prediction and the actual label to adjust itself.
4. The network then takes the next image (or group of images) and make another prediction.
5. This new (hopefully closer) prediction is compared to the actual label of the image(s).
6. The network uses information about the difference between this new prediction and the actual label to adjust again.
7. This happens repeatedly until the network has looked at each image.

Compare this to a human study session using flash cards:

1. A student looks at a first flashcard and makes a guess about what is on the other side.
2. They check the other side to see how close they were.
3. They adjust their understanding based on this new information.
4. The student then looks at the next card and makes another prediction.
5. This new (hopefully closer) prediction is compared to the answer on the back of the card.
6. The student uses information about the difference between this new prediction and the right answer to adjust again.
7. This happens repeatedly until the student has tried each flashcard once.

You can see that one epoch can be compared to one trip through a deck of flashcards.

In the model that we trained, we asked our network for 5 epochs. The blue curve on the graph above shows how far off each prediction was from the actual label.

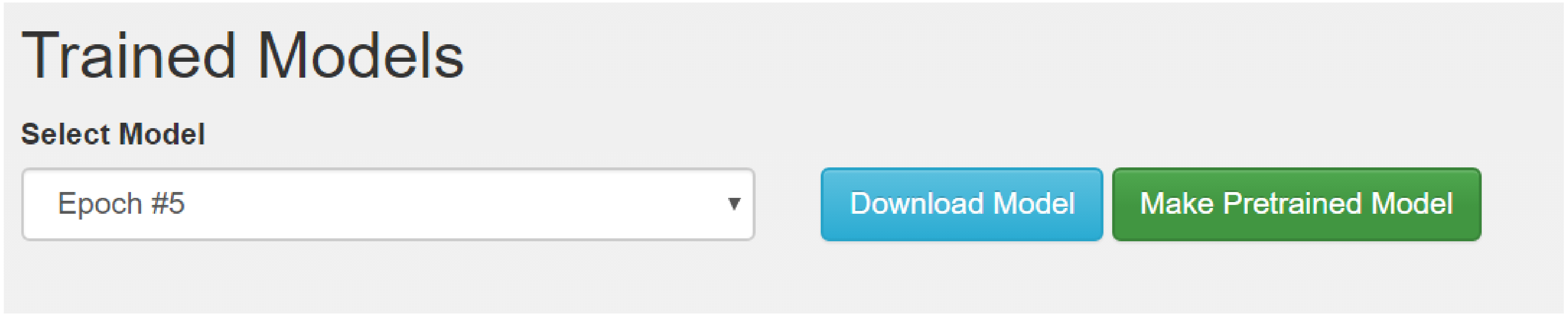
Like a human student, the point of studying isn't just to be able to replicate someone else's knowledge. The green curve shows the difference between the model's predictions and actual labels for NEW data that it hasn't learned from. The orange curve is simple the inverse of that loss.

**Loss** can be measured in many ways, but conceptually, it's simply the difference between predicted and actual labels.

If after taking a test and earning 87%, a human student wanted to improve performance, they might be advised to study more. At this point, they'd go back to their flashcards.

A deep learning practitioner could request more epochs. It *is* possible to start from scratch and request more epochs when creating your model for the first time. However, our model has already spent some time learning! Let's use what it has learned to improve our model instead of start from scratch.

Head back to DIGITS and scroll to the bottom of your model page and click the big green button labeled: "Make Pretrained Model."



This will save two things:

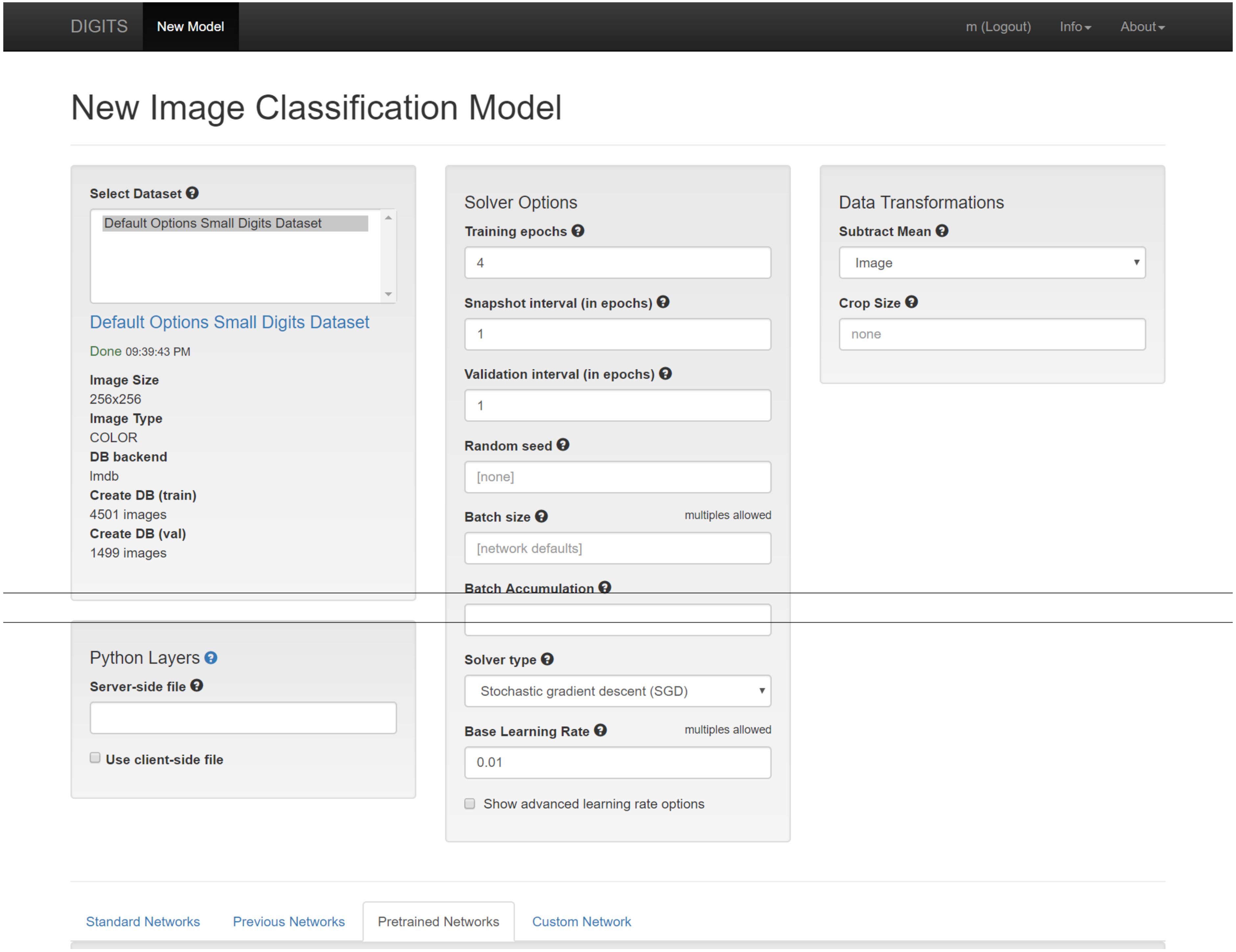
1. The "network architecture" that you chose when selecting "AlexNet."
2. What the model has "learned" in the form of the parameters that have been adjusted as your network worked through the data in the first 5 epochs.

We can now create a new model from this starting point. Go back to DIGITS' home screen and create a new Image Classfication model like before. New Model (Images) -> Classification

- Select the same dataset - (Default Options Small Dataset)
- Choose some number of epochs between 3 and 8. (Note that in creating a model from scratch, this is where you could have requested more epochs originally.)
- This time, instead of choosing a "Standard Network," select "Pretrained Networks."
- Select the pretrained model that you just created, likely "My first model."

- Name your model - We chose "Study more"
- Click Create

Your settings should look like:



New Image Classification Model

**Select Dataset**

Default Options Small Digits Dataset

Default Options Small Digits Dataset

Done 09:39:43 PM

**Image Size**
256x256
**Image Type**
COLOR
**DB backend**
lmdb
**Create DB (train)**
4501 images
**Create DB (val)**
1499 images

Python Layers

**Server-side file**

☐ **Use client-side file**

Solver Options

**Training epochs**
4

**Snapshot interval (in epochs)**
1

**Validation interval (in epochs)**
1

**Random seed**
[none]

**Batch size** multiples allowed
[network defaults]

**Batch Accumulation**

**Solver type**
Stochastic gradient descent (SGD)

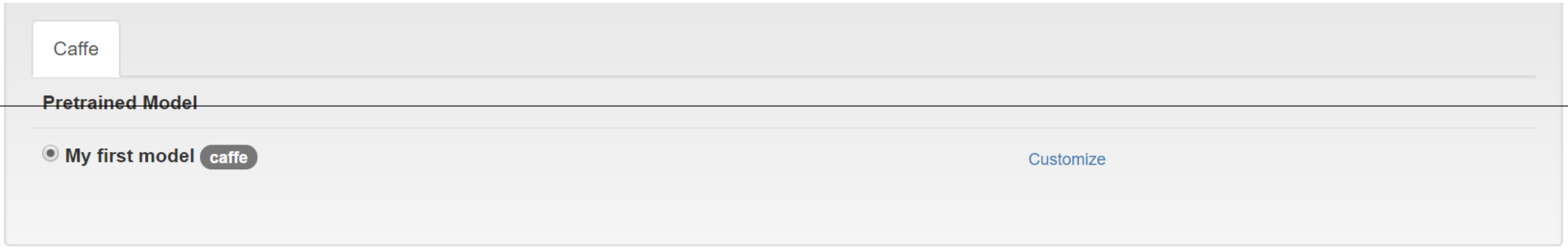
**Base Learning Rate** multiples allowed
0.01

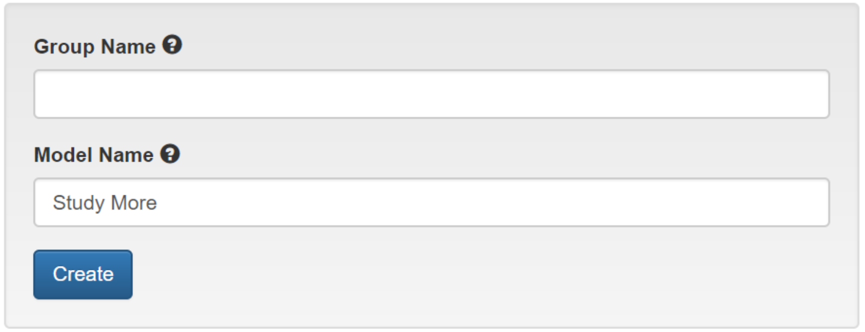
☐ Show advanced learning rate options

Data Transformations

**Subtract Mean**
Image

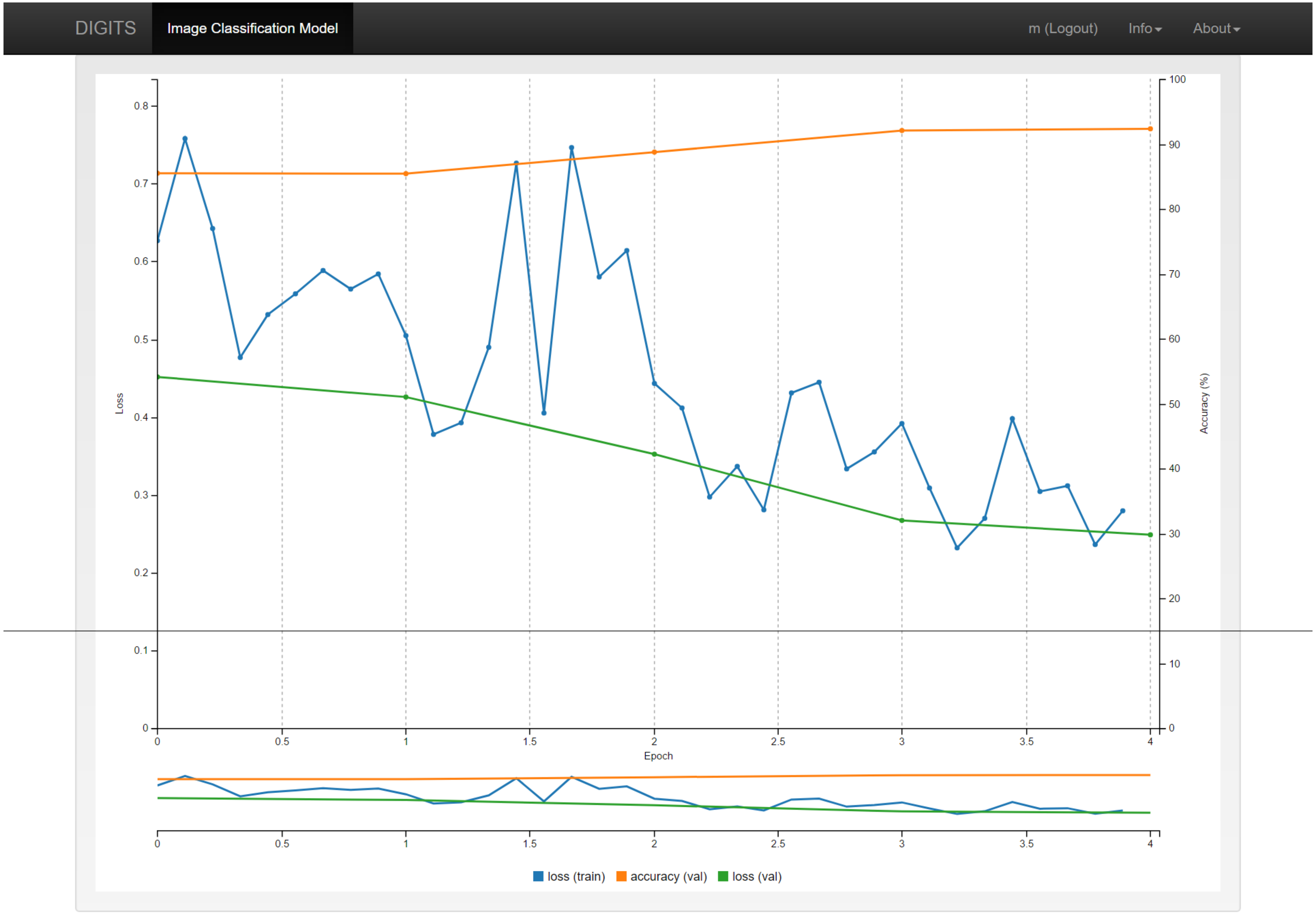
**Crop Size**
none





Model Name: Study More

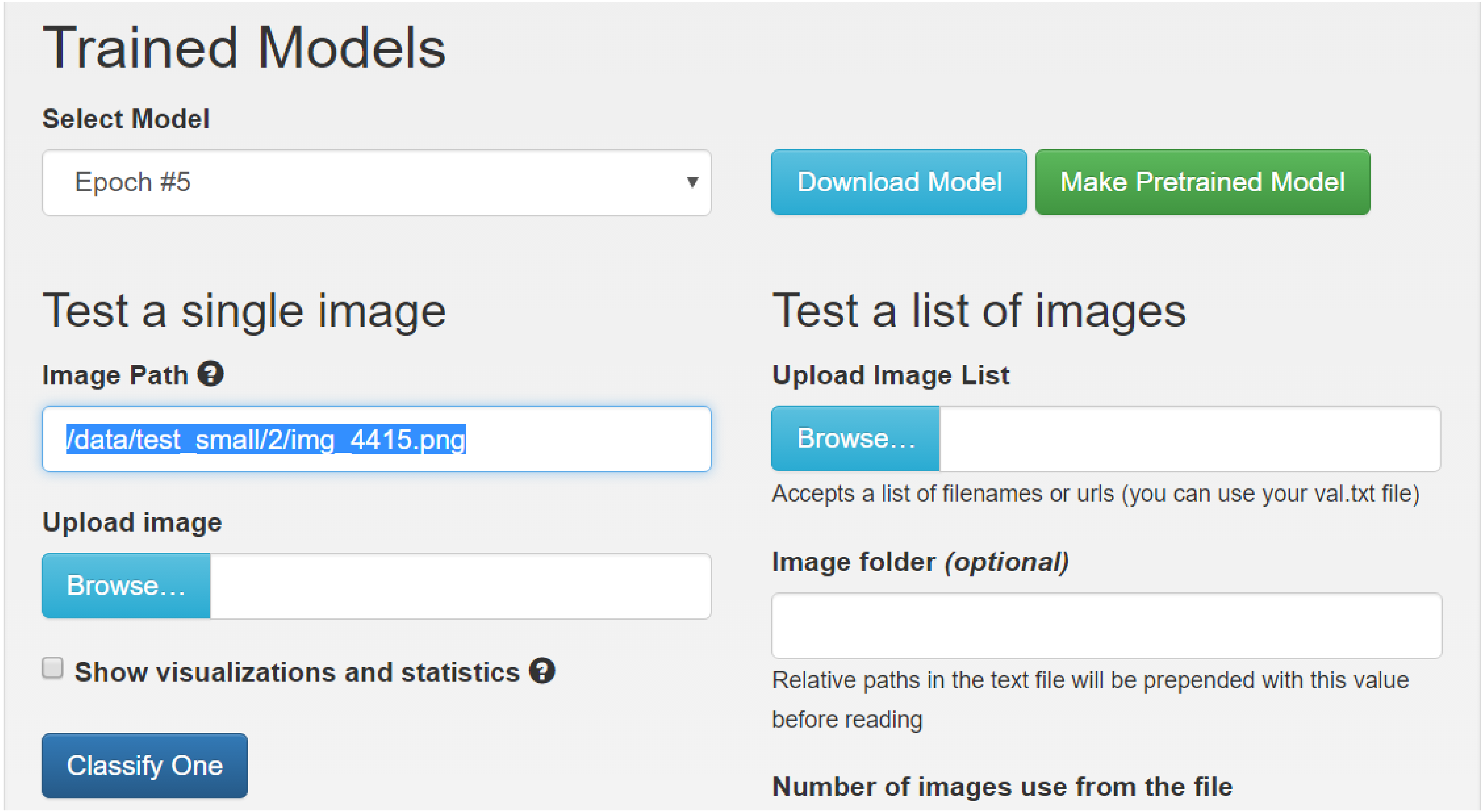
When you create the model, you'll get the following graph.


DIGITS
Image Classification Model
m (Logout)
Info
About
Loss
Accuracy (%)
Epoch
0
0.1
0.2
0.3
0.4
0.5
0.6
0.7
0.8
0
10
20
30
40
50
60
70
80
90
100
0
0.5
1
1.5
2
2.5
3
3.5
4
loss (train)
accuracy (val)
loss (val)

Note the following:

1. As expected, the accuracy starts close to where our first model left off, 86%.
2. Accuracy DOES continue to increase, showing that increasing the number of epochs often does increase performance.
3. The *rate* of increase in accuracy slows down, showing that more trips through the same data can't be the only way to increase performance.

Let's test our new and improved model using the same image from before. At the bottom of our model page, "Test a single image." We'll test the same "2" to compare performance. As a reminder, the image path is:
`/data/test_small/2/img_4415.png`


Trained Models

Select Model: Epoch #5

Test a single image

Image Path: /data/test_small/2/img_4415.png

Upload image

Show visualizations and statistics

Test a list of images

Upload Image List

Accepts a list of filenames or urls (you can use your val.txt file)

Image folder *(optional)*

Relative paths in the text file will be prepended with this value before reading

Number of images use from the file

Recall that our original model correctly identified this as a 2 as well, but with a confidence of 85%. This is clearly a better model.

Feel free to try a few more images by changing the number after "img_ "

Let's try testing the model with a set of images. They are shown below.

image-1-1.jpg

image-2-1.jpg

image-3-1.jpg

image-4-1.jpg

image-7-1.jpg

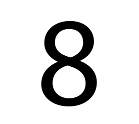


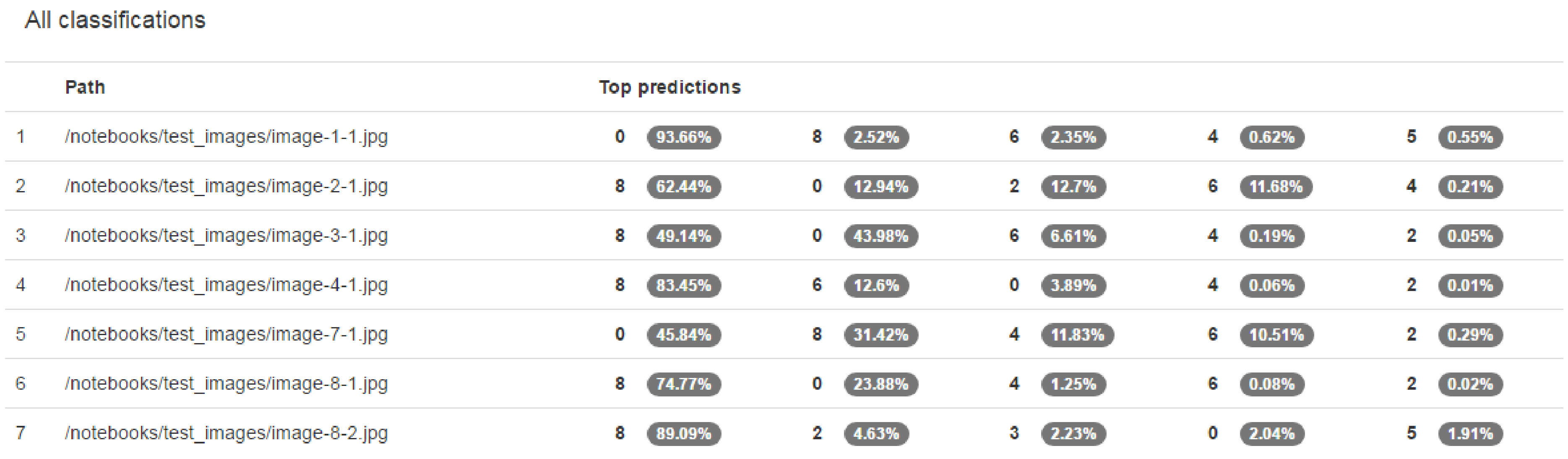
image-8-1.jpg

image-8-2.jpg

We can classify multiple files if we put them in the list. In the link below, execute the code block and a link to the file an_image.list will appear. Right click on an_image.list and save that to a file on your local computer(right click and "Save As"). Remember the directory in which it is saved.

In [ ]:
```
from IPython.display import FileLink, FileLinks
FileLinks('test_images_list')
```

On the right side of the DIGITS model page, there is an option to "test a list of images". Press the button **Browse** and select the `an_image.list` file you just downloaded. Then press the **Classify Many** button. After several seconds, you will see the results from Caffe trying to classify these images with the generated model. In the image name, the first number is the digit in the image (ex. image-3-1.jpg is a 3). Your results should be similar to this:


All classifications

| | Path | Top predictions | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|
| 1 | /notebooks/test_images/image-1-1.jpg | 0 | 93.66% | 8 | 2.52% | 6 | 2.35% | 4 | 0.62% | 5 | 0.55% |
| 2 | /notebooks/test_images/image-2-1.jpg | 8 | 62.44% | 0 | 12.94% | 2 | 12.7% | 6 | 11.68% | 4 | 0.21% |
| 3 | /notebooks/test_images/image-3-1.jpg | 8 | 49.14% | 0 | 43.98% | 6 | 6.61% | 4 | 0.19% | 2 | 0.05% |
| 4 | /notebooks/test_images/image-4-1.jpg | 8 | 83.45% | 6 | 12.6% | 0 | 3.89% | 4 | 0.06% | 2 | 0.01% |
| 5 | /notebooks/test_images/image-7-1.jpg | 0 | 45.84% | 8 | 31.42% | 4 | 11.83% | 6 | 10.51% | 2 | 0.29% |
| 6 | /notebooks/test_images/image-8-1.jpg | 8 | 74.77% | 0 | 23.88% | 4 | 1.25% | 6 | 0.08% | 2 | 0.02% |
| 7 | /notebooks/test_images/image-8-2.jpg | 8 | 89.09% | 2 | 4.63% | 3 | 2.23% | 0 | 2.04% | 5 | 1.91% |

What is shown here is the probability that the model predicts the class of the image. The results are sorted from highest probability to lowest. Our model didn't do so well.

While the accuracy of the model was 87%, it could not correctly classify any of the images that we tested. What can we do to improve the classification of these images?

At this point it's time to be a bit more intentional. After we can successfully train a model, what comes next comes from understanding and experimentation. To build a better understanding of THIS project, we should start with our data. To build a better understanding of anything, we should start with primary sources.

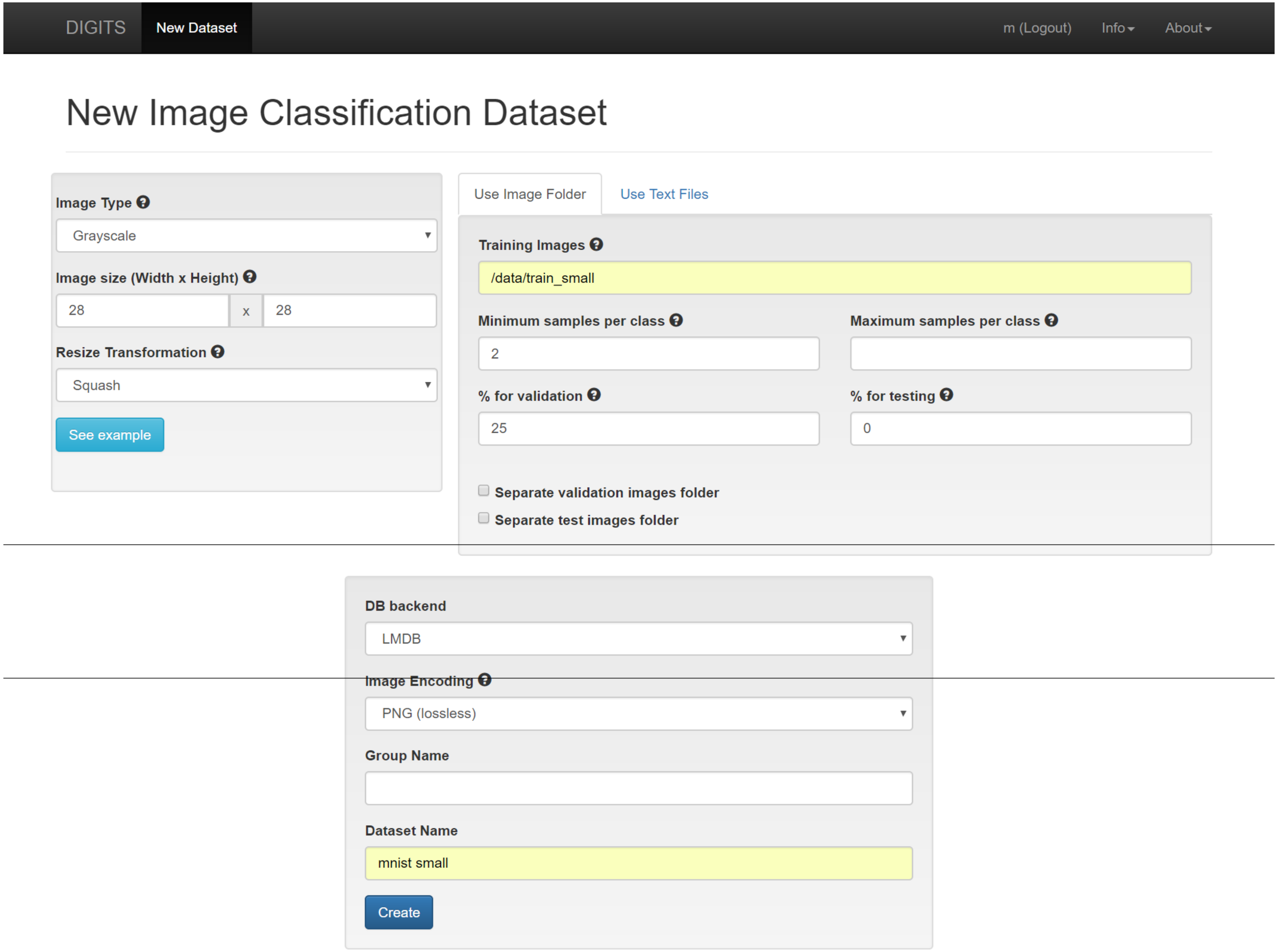
The dataset that we are learning from is a subset of the MNIST (http://yann.lecun.com/exdb/mnist/) dataset. A close read of the documentation would likely yield a lot of insight.

One key observation we'll use is that the images start as 28x28 grayscale images. When we loaded our dataset, we stuck with defaults, which were 256x256 and color. You may have noticed that the images were a bit blurry. In the next section we'll explore the benefits that can come from matching your data to the right model.

## The right model

Let's start from scratch using what we've learned. This time, we'll load our data as 28x28 grayscale images and pick a model that is built to accept that type of data, LeNet. To compare to our previous model, use the total number of epochs that you trained with so far, eg. the 5 in "my first model" and the additional epochs trained from your pretrained model. In my case I'll use 8.

Here's an image of the settings that would create the dataset.



New Image Classification Dataset

Image Type: Grayscale

Image size (Width x Height): 28 x 28

Resize Transformation: Squash

Training Images: /data/train_small

Minimum samples per class: 2

Maximum samples per class:

% for validation: 25

% for testing: 0

Separate validation images folder

Separate test images folder

DB backend: LMDB

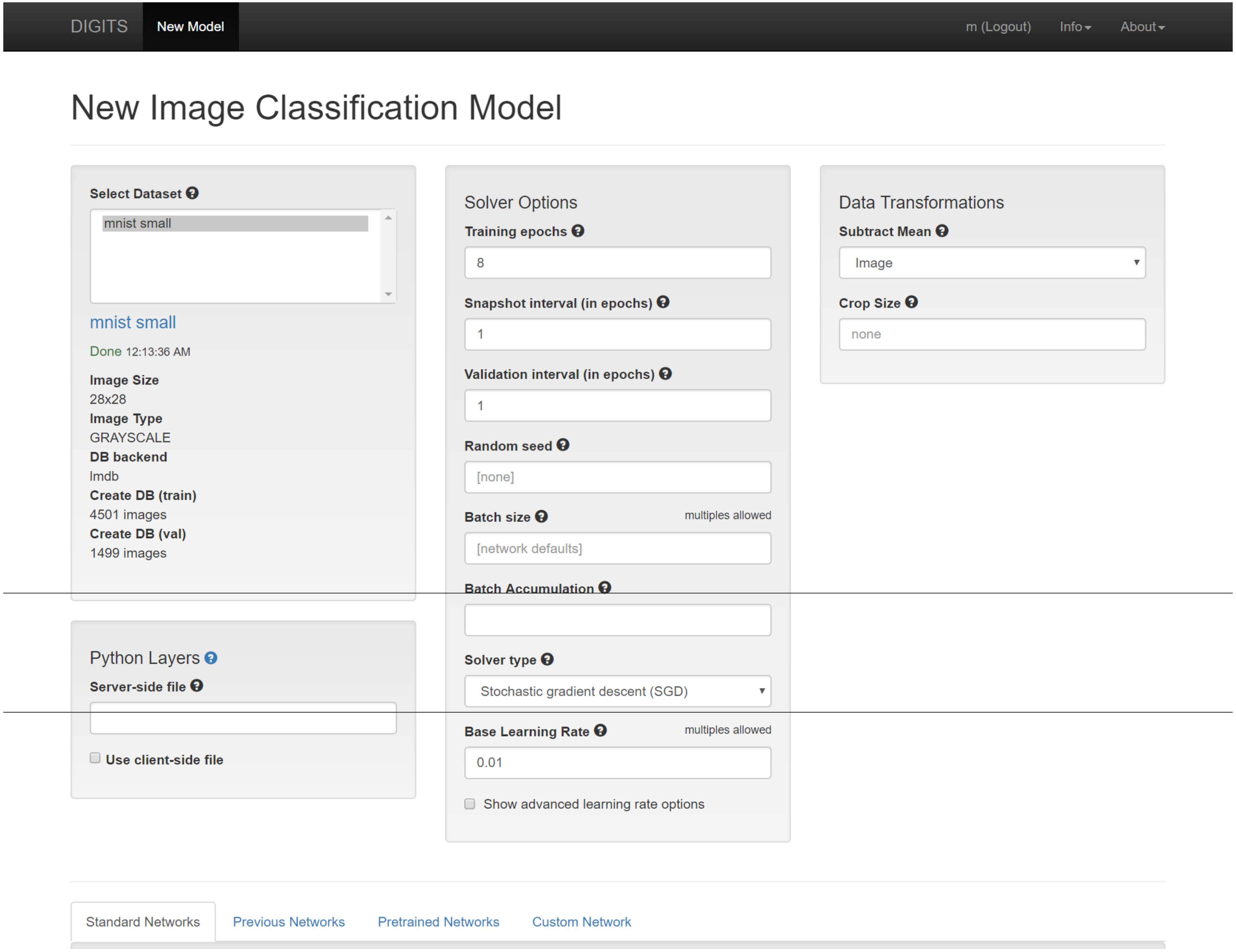
Image Encoding: PNG (lossless)

Group Name:

Dataset Name: mnist small

Feel free to "explore the db" again. I notice immediately that the images are no longer blurry.

Next, create a model using the settings in the image below. Note that the LeNet model is designed to take 28x28 grayscale images. You'll see shortly that this network was actually purpose-built for digit classification.



New Image Classification Model

Select Dataset

mnist small

mnist small

Done 12:13:36 AM

Image Size
28x28
Image Type
GRAYSCALE
DB backend
lmdb
Create DB (train)
4501 images
Create DB (val)
1499 images

Python Layers

Server-side file

Use client-side file

Solver Options

Training epochs: 8

Snapshot interval (in epochs): 1

Validation interval (in epochs): 1

Random seed: [none]

Batch size (multiples allowed): [network defaults]

Batch Accumulation:

Solver type: Stochastic gradient descent (SGD)

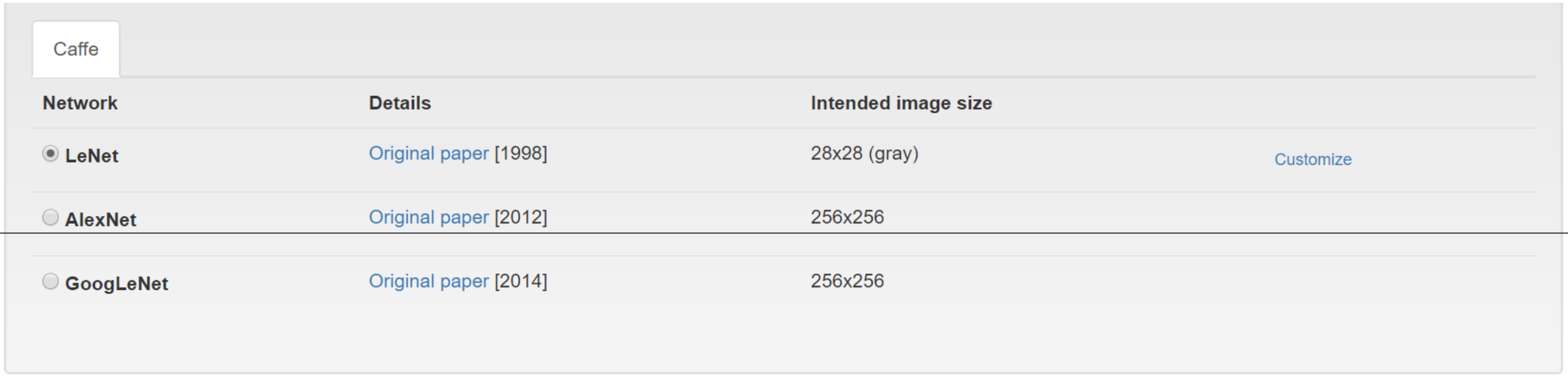
Base Learning Rate (multiples allowed): 0.01

Show advanced learning rate options

Data Transformations

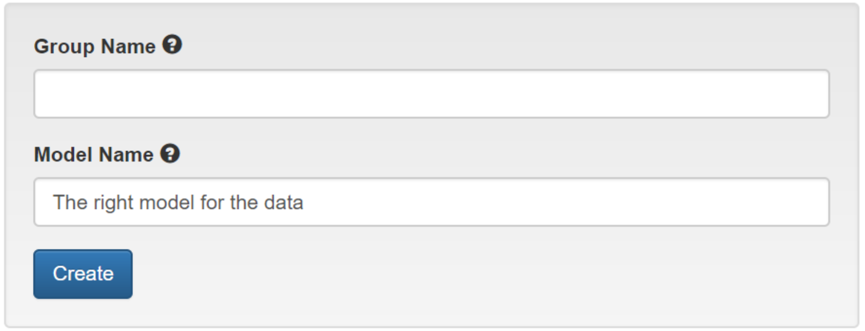
Subtract Mean: Image

Crop Size: none



Caffe

| Network | Details | Intended image size | |
|---|---|---|---|
| ● LeNet | Original paper [1998] | 28x28 (gray) | Customize |
| ○ AlexNet | Original paper [2012] | 256x256 | |
| ○ GoogLeNet | Original paper [2014] | 256x256 | |

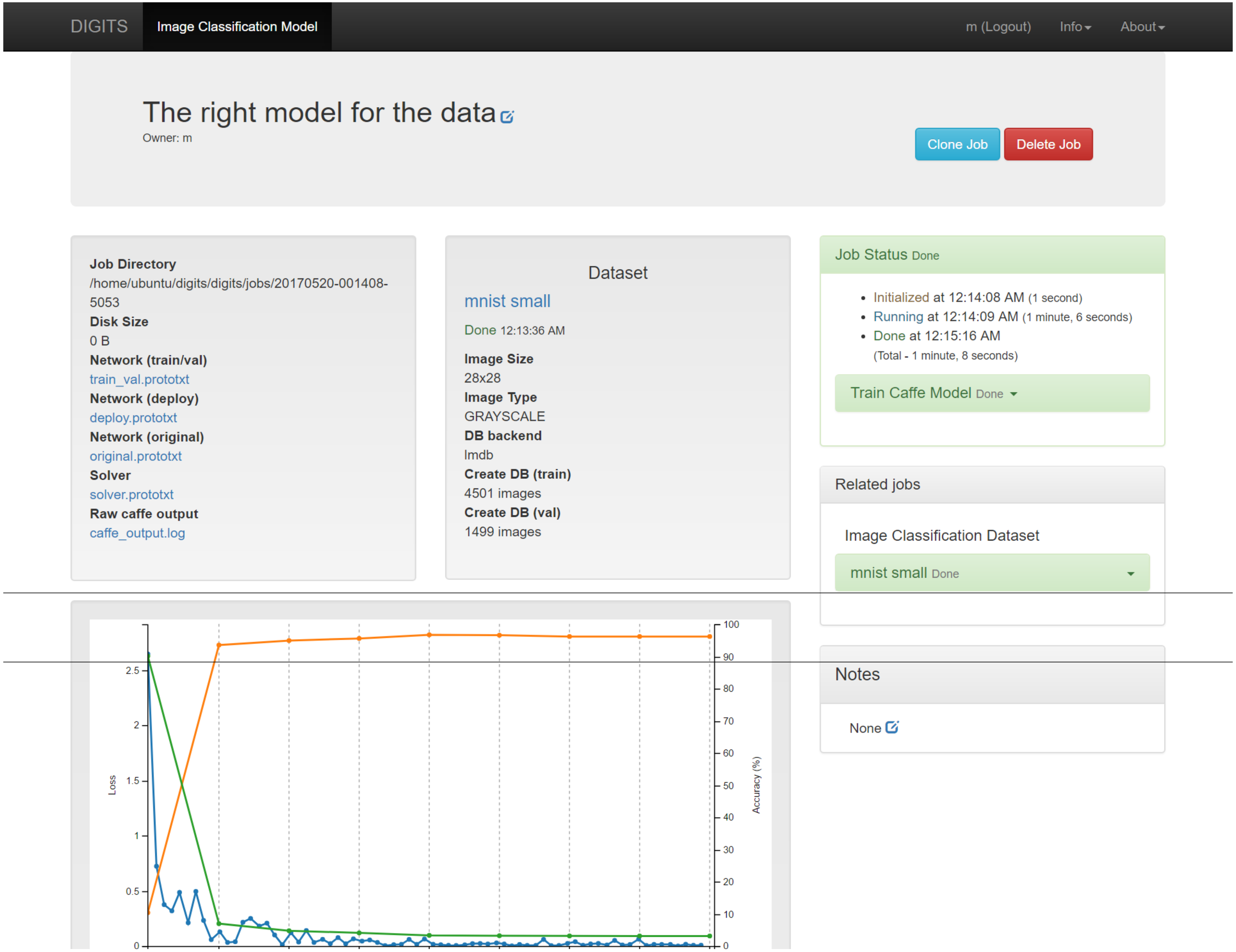


Group Name

Model Name: The right model for the data

Woah. You should have noticed two things.

1. Your model improved performance. Mine was accurate to more than 96%.
2. Your model trained faster. In far less than two minutes, my model ran through 8 epochs.



The right model for the data

Owner: m

**Job Directory**
/home/ubuntu/digits/digits/jobs/20170520-001408-5053
**Disk Size**
0 B
**Network (train/val)**
train_val.prototxt
**Network (deploy)**
deploy.prototxt
**Network (original)**
original.prototxt
**Solver**
solver.prototxt
**Raw caffe output**
caffe_output.log

Dataset

mnist small

Done 12:13:36 AM

**Image Size**
28x28
**Image Type**
GRAYSCALE
**DB backend**
lmdb
**Create DB (train)**
4501 images
**Create DB (val)**
1499 images

Job Status Done

- Initialized at 12:14:08 AM (1 second)
- Running at 12:14:09 AM (1 minute, 6 seconds)
- Done at 12:15:16 AM
(Total - 1 minute, 8 seconds)

Train Caffe Model Done

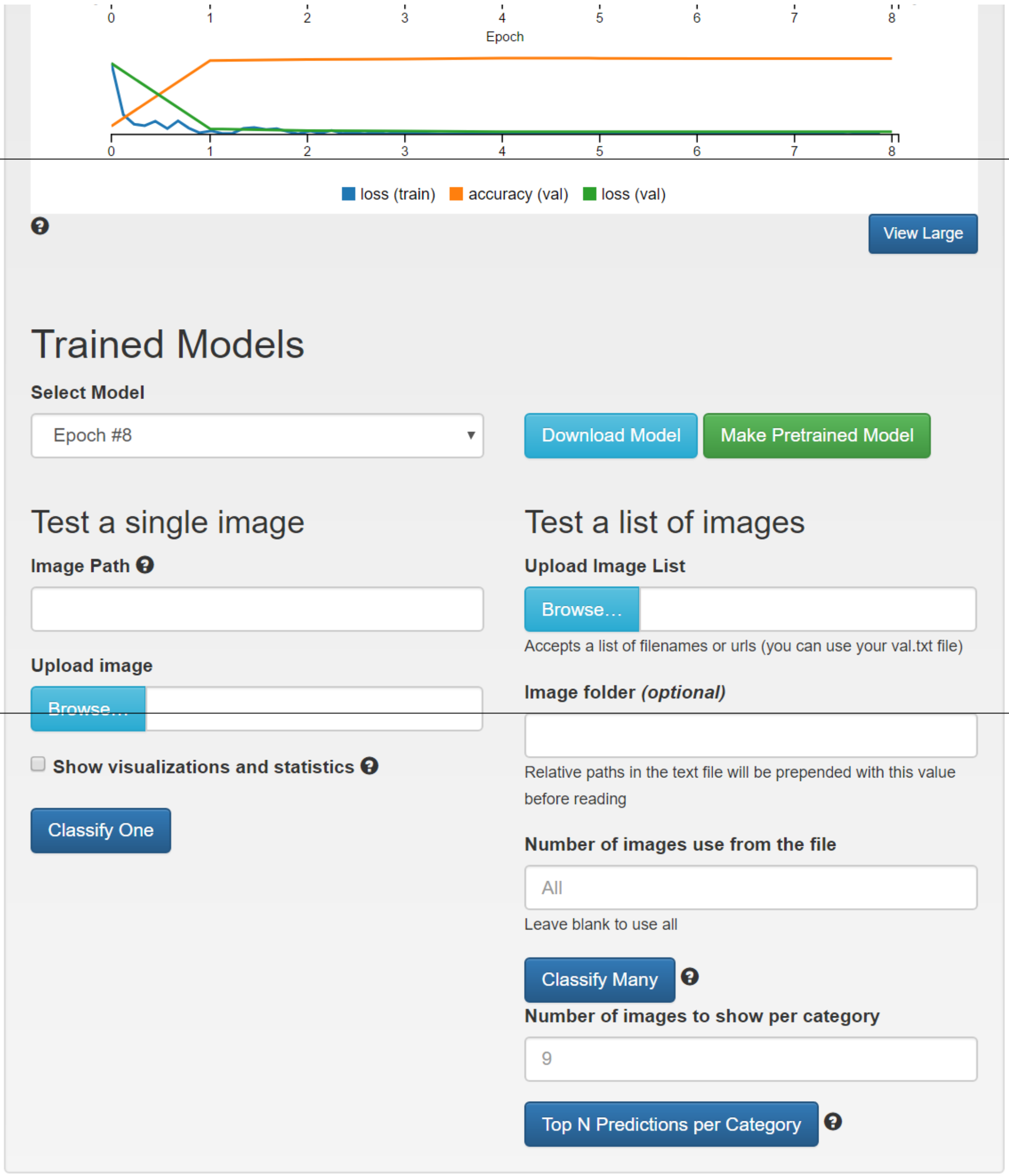
Related jobs

Image Classification Dataset

mnist small Done

Notes

None



## Trained Models

**Select Model**

Epoch #8

Download Model | Make Pretrained Model

### Test a single image

**Image Path**

**Upload image**

Browse…

☐ **Show visualizations and statistics**

Classify One

### Test a list of images

**Upload Image List**

Browse…

Accepts a list of filenames or urls (you can use your val.txt file)

**Image folder *(optional)***

Relative paths in the text file will be prepended with this value before reading

**Number of images use from the file**

All

Leave blank to use all

Classify Many

**Number of images to show per category**

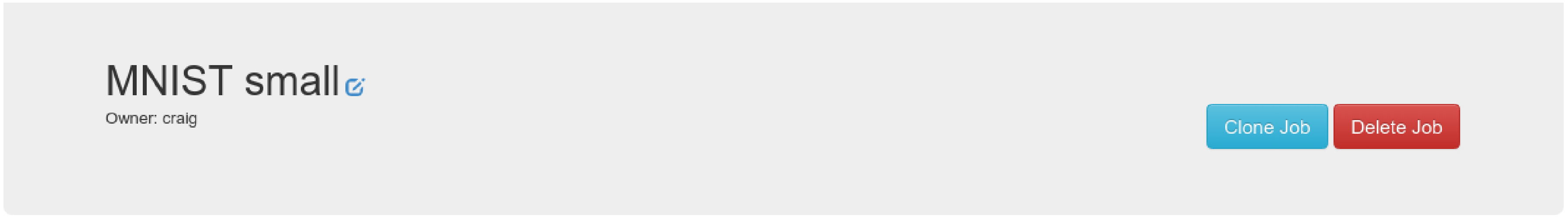
9

Top N Predictions per Category

We haven't done anything to address the problem of more diverse data, but if we can train faster, we can experiment a lot more.

## Train with more data

In our last attempt at training, we only used 10% of the full MNIST dataset. Let's try training with the complete dataset and see how it improves our training. We can use the clone option in DIGITS to simplify the creation of a new job with similar properties to an older job. Let's return to the home page by clicking on **DIGITS** in the upper left corner. Then select **Dataset** from the left side of the page to see all of the datasets that you have created. You will see your **Mnist small** dataset. When you select that dataset, you will be returned to the results window of that job. In the right corner you will see a button: **Clone Job**.



Press the **Clone Job** button.

From here you will see the create dataset template populated with all the options you used when you created the Default Options Small Digits dataset. To create a database with the full MNIST data, change the following settings:

- Training Images - /data/train_full
- Dataset Name - MNIST full

Then press the **Create** button. This dataset is ten times larger than the other dataset, so it will take a few minutes to process.

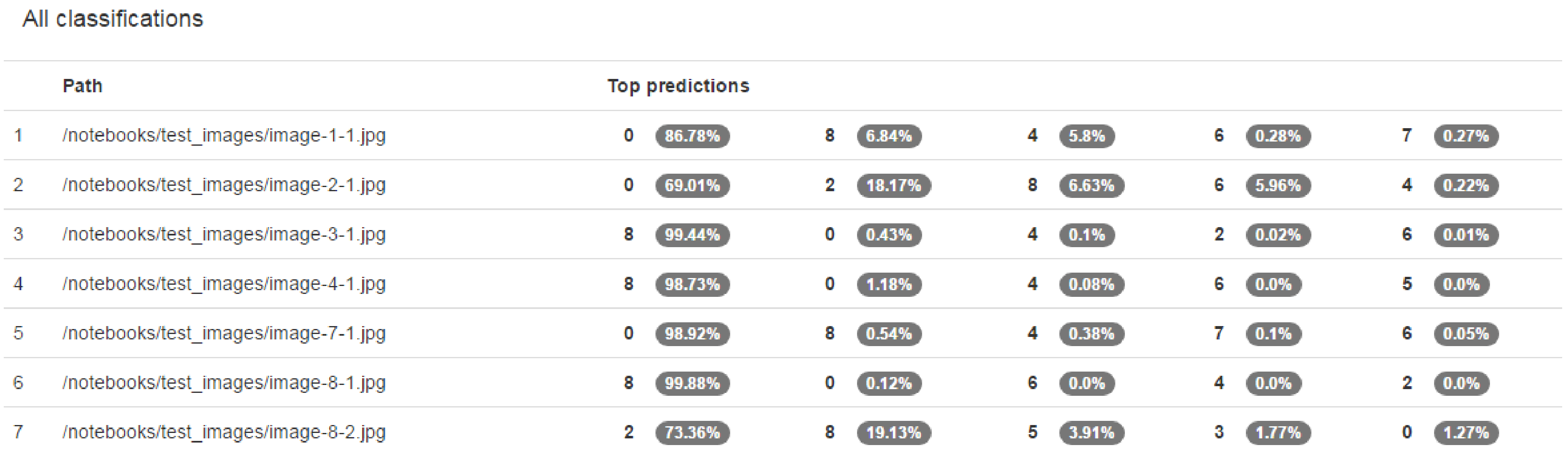
After you have created your new database, follow the same procedure to clone your training model. In the template, change the following values:

- Select the MNIST full dataset
- Change the name to MNIST full

Then create the model.

With much more data, the model will take longer to run. It still should complete in less than a minute. What do you notice that is different about the results? Both the training and validation loss function values are much smaller. In addition, the accuracy of the model is around 99%, possibly greater. That is saying the model is correctly identifying most every image in its validation set. This is a significant improvement. However, how well does this new model do on the challenging test images we used previously?

Using the same procedure from above to classify our set of test images, here are the new results:



All classifications

| | Path | Top predictions | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|
| 1 | /notebooks/test_images/image-1-1.jpg | 0 | 86.78% | 8 | 6.84% | 4 | 5.8% | 6 | 0.28% | 7 | 0.27% |
| 2 | /notebooks/test_images/image-2-1.jpg | 0 | 69.01% | 2 | 18.17% | 8 | 6.63% | 6 | 5.96% | 4 | 0.22% |
| 3 | /notebooks/test_images/image-3-1.jpg | 8 | 99.44% | 0 | 0.43% | 4 | 0.1% | 2 | 0.02% | 6 | 0.01% |
| 4 | /notebooks/test_images/image-4-1.jpg | 8 | 98.73% | 0 | 1.18% | 4 | 0.08% | 6 | 0.0% | 5 | 0.0% |
| 5 | /notebooks/test_images/image-7-1.jpg | 0 | 98.92% | 8 | 0.54% | 4 | 0.38% | 7 | 0.1% | 6 | 0.05% |
| 6 | /notebooks/test_images/image-8-1.jpg | 8 | 99.88% | 0 | 0.12% | 6 | 0.0% | 4 | 0.0% | 2 | 0.0% |
| 7 | /notebooks/test_images/image-8-2.jpg | 2 | 73.36% | 8 | 19.13% | 5 | 3.91% | 3 | 1.77% | 0 | 1.27% |

The model was still only able to classify one of the seven images. While some of the classifications came in a close second, our model's predictive capabilities were not much greater. Are we asking too much of this model to try and classify non-handwritten, often colored, digits with our model?

## Improving Model Results - Data Augmentation

You can see with our seven test images that the backgrounds are not uniform. In addition, most of the backgrounds are light in color whereas our training data all have black backgrounds. We saw that increasing the amount of data did help for classifying the handwritten characters, so what if we include more data that tries to address the contrast differences?

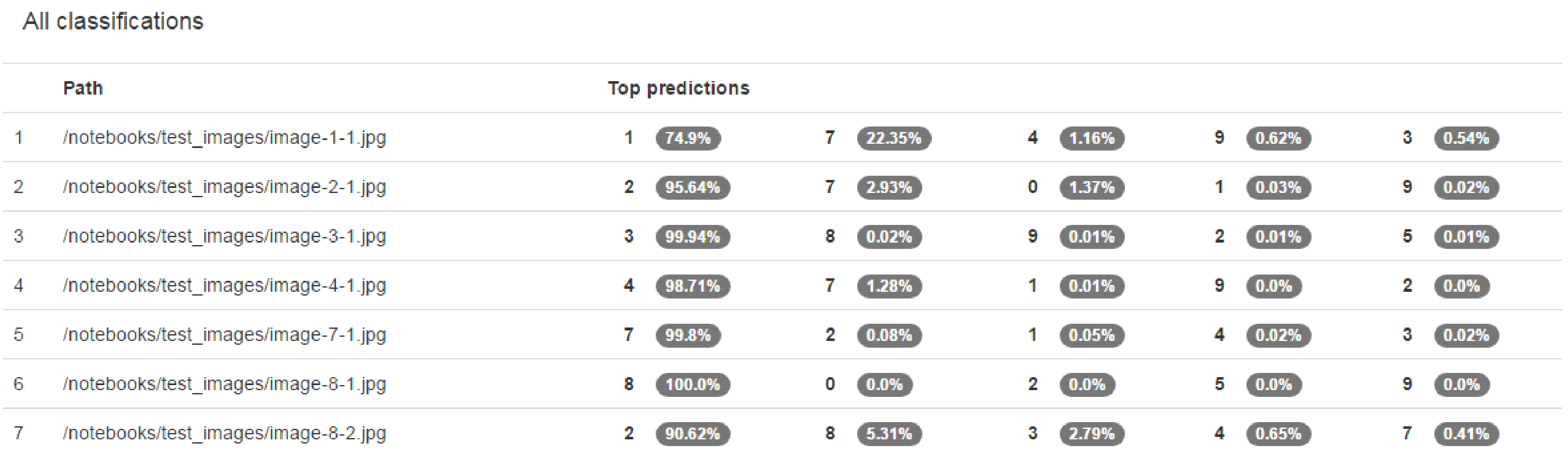
Let's try augmenting our data by inverting the original images. Let's turn the white pixels to black and vice-versa. Then we will train our network using the original and inverted images and see if classification is improved.

To do this, follow the steps above to clone and create a new dataset and model. The directories for the augmented data are:

- Training Images - /data/train_invert

Remember to change the name of your dataset and model. When the new dataset is ready, explore the database. Now you should see images with black backgrounds and white numbers and also white backgrounds and black numbers.

Now train a new model. Clone your previous model results, and change the dataset to the one you just created with the inverted images. Change the name of the model and create a new model. When the training is complete, the accuracy hasn't really increased over the non-augmented image set. In fact, the accuracy may have gone down slightly. We were already at 99% so it is unlikely we were going to improve our accuracy. Did using an augmented dataset help us to better classify our images? Here is the result:



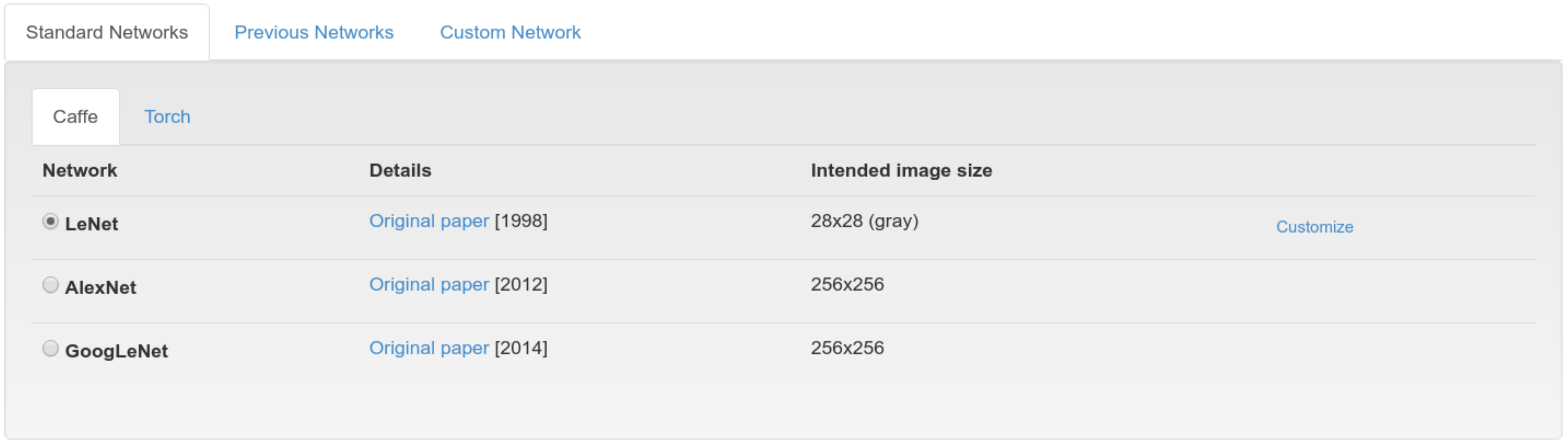
All classifications

| | Path | Top predictions | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|
| 1 | /notebooks/test_images/image-1-1.jpg | 1 | 74.9% | 7 | 22.35% | 4 | 1.16% | 9 | 0.62% | 3 | 0.54% |
| 2 | /notebooks/test_images/image-2-1.jpg | 2 | 95.64% | 7 | 2.93% | 0 | 1.37% | 1 | 0.03% | 9 | 0.02% |
| 3 | /notebooks/test_images/image-3-1.jpg | 3 | 99.94% | 8 | 0.02% | 9 | 0.01% | 2 | 0.01% | 5 | 0.01% |
| 4 | /notebooks/test_images/image-4-1.jpg | 4 | 98.71% | 7 | 1.28% | 1 | 0.01% | 9 | 0.0% | 2 | 0.0% |
| 5 | /notebooks/test_images/image-7-1.jpg | 7 | 99.8% | 2 | 0.08% | 1 | 0.05% | 4 | 0.02% | 3 | 0.02% |
| 6 | /notebooks/test_images/image-8-1.jpg | 8 | 100.0% | 0 | 0.0% | 2 | 0.0% | 5 | 0.0% | 9 | 0.0% |
| 7 | /notebooks/test_images/image-8-2.jpg | 2 | 90.62% | 8 | 5.31% | 3 | 2.79% | 4 | 0.65% | 7 | 0.41% |

By augmenting our dataset with the inverted images, we could identify six of the seven images. While our result is not perfect, our slight change to the images to increase our dataset size made a significant difference.

## Advanced - Improving Model Results by Modifying the Network (Optional)

Augmenting the dataset improved our results, but we are not identifying all our test images. Let's try modifying the LeNet network directly. You can create custom networks to modify the existing ones, use different networks from external sources, or create your own. To modify a network, select the Customize link on the right side of the Network dialog box.



| Network | Details | Intended image size |
|---|---|---|
| ● LeNet | Original paper [1998] | 28x28 (gray) |
| ○ AlexNet | Original paper [2012] | 256x256 |
| ○ GoogLeNet | Original paper [2014] | 256x256 |

This will open an editor with the LeNet model configuration. Scroll through the window and look at the code. The network is defined as a series of layers. Each layer has a name that is a descriptor of its function. Each layer has a top and a bottom, or possibly multiples of each, indicating how the layers are connected. The *type* variable defined what type the layer is. Possibilities include **Convolution**, **Pool**, and **ReLU**. All the options available in the Caffe model language are found in the Caffe Tutorial (http://caffe.berkeleyvision.org/tutorial/).

At the top of the editor, there is a **Visualize** button. Pressing this button will visualize all of the layers of the model and how they are connected. In this window, you can see that the data are initially scaled, there are two sets of Convolution and Pooling layers, and two Inner Products with a Rectilinear Unit (ReLU) connected to the first Inner Product. At the bottom of the network, there are output functions that return the accuracy and loss computed through the network.

We are going to make two changes to the network. First, we are going to connect a ReLU to the first pool. Second, we are going to change the values of num_output to 75 for the first Convolution (conv1) and 100 for the second Convolution (conv2). The ReLU layer definition should go below the pool1 definition and look like:

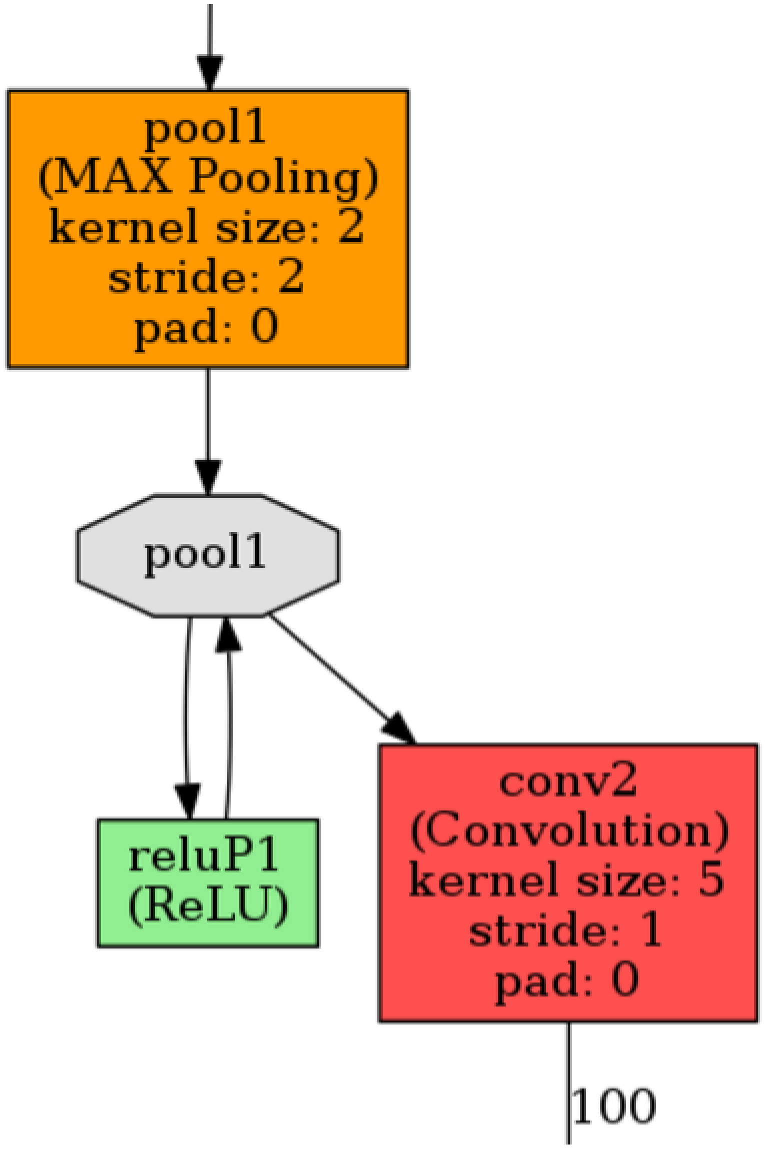
```
layer {
  name: "reluP1"
  type: "ReLU"
  bottom: "pool1"
  top: "pool1"
}
```

The Convolution layers should be changed to look like:

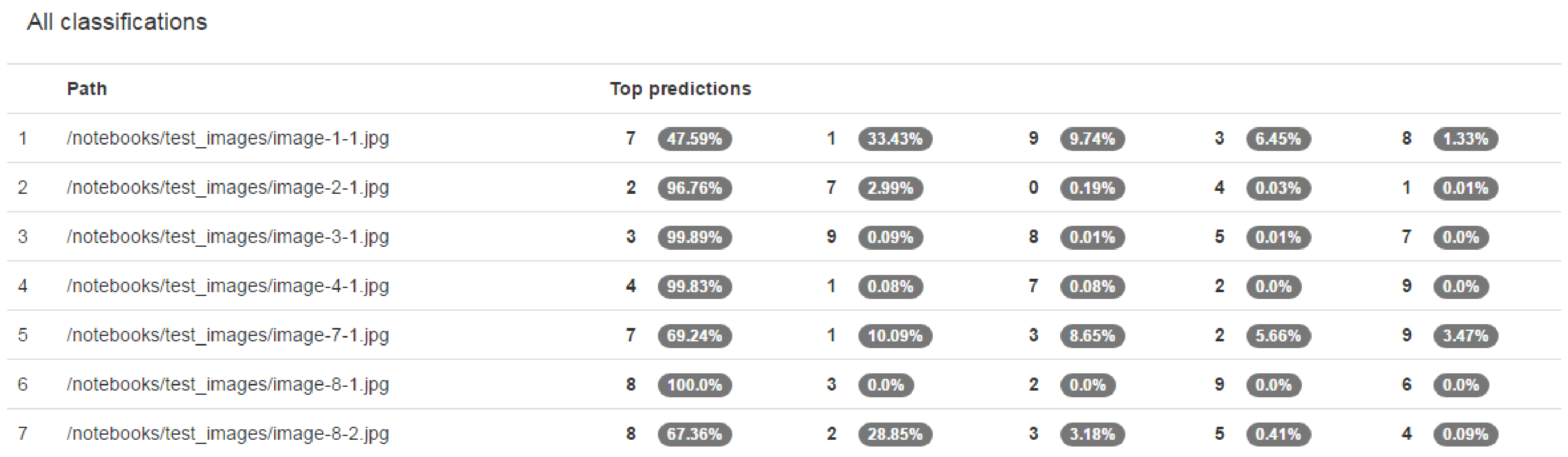
```
layer {
  name: "conv1"
  type: "Convolution"
  bottom: "scaled"
  top: "conv1"
...
  convolution_param {
    num_output: 75
...

layer {
  name: "conv2"
  type: "Convolution"
  bottom: "pool1"
  top: "conv2"
...
  convolution_param {
    num_output: 100
...
```

Note, the ellipis (...) just indicates we removed some of the lines from the layer for brevity. The only change you have to make is to the value of num_output.

After making these changes, visualize your new model. You should see the ReLU unit appear similar to:



Now change the name of the model and press the **Create** button. When it is complete test the data again. The results should be similar to:



All classifications

| | Path | Top predictions | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|
| 1 | /notebooks/test_images/image-1-1.jpg | 7 | 47.59% | 1 | 33.43% | 9 | 9.74% | 3 | 6.45% | 8 | 1.33% |
| 2 | /notebooks/test_images/image-2-1.jpg | 2 | 96.76% | 7 | 2.99% | 0 | 0.19% | 4 | 0.03% | 1 | 0.01% |
| 3 | /notebooks/test_images/image-3-1.jpg | 3 | 99.89% | 9 | 0.09% | 8 | 0.01% | 5 | 0.01% | 7 | 0.0% |
| 4 | /notebooks/test_images/image-4-1.jpg | 4 | 99.83% | 1 | 0.08% | 7 | 0.08% | 2 | 0.0% | 9 | 0.0% |
| 5 | /notebooks/test_images/image-7-1.jpg | 7 | 69.24% | 1 | 10.09% | 3 | 8.65% | 2 | 5.66% | 9 | 3.47% |
| 6 | /notebooks/test_images/image-8-1.jpg | 8 | 100.0% | 3 | 0.0% | 2 | 0.0% | 9 | 0.0% | 6 | 0.0% |
| 7 | /notebooks/test_images/image-8-2.jpg | 8 | 67.36% | 2 | 28.85% | 3 | 3.18% | 5 | 0.41% | 4 | 0.09% |

Were you able to correctly identify them all? If not, why do you think the results were different?

## Next Steps

In our example here, we were able to identify most of our test images successfully. However, that is generally not the case. How would go about improving our model further? Typically, hyper-parameter searches are done to try different values of model parameters such as learning-rate or different solvers to find settings that improve model accuracy. We could change the model to add layers or change some of the parameters within the model associated with the performance of the convolution and pooling layers. In addition, we could try other networks.

## Summary

In this tutorial you were introduced to Deep Learning and all of the steps necessary to classify images including data processing, training, testing, and improving your network through data augmentation and network modifications. In the training phase, you learned about the parameters that can determine the performance of training a network. By training a subset of the MNIST data, a full set, different models, augemented data, etc. you saw the types of options you have to control performance. In testing our model, we found that although the test images were quite different than the training data, we could still correctly classify them.

Now that you have a basic understanding of Deep Learning and how to train using DIGITS, what you do next is limited only by your own imagination. Test what you have learned, there are several good datasets with which to practice. We have included the CalTech 101 (http://www.vision.caltech.edu/Image_Datasets/Caltech101/) dataset at

```
/data/101_ObjectCategories
```

Feel free to run any part of this lab again with that (more complex) dataset.

There is still much more to learn! Start a Deep Learning project or take another course. The Nvidia Deep Learning Institute has a comprehensive library of hands-on labs for both fundamental and advanced topics.

NVIDIA | DEEP LEARNING INSTITUTE

(https://www.nvidia.com/en-us/deep-learning-ai/education/)

## Appendix

For a high performing and generalizable model:

1. Load the dataset stored at

   ```
   /data/train_invert
   ```

   as 28x28 grayscale images.

2. Train a model using the LeNet network for 5 epochs.