

FUNDAMENTALS OF DEEP LEARNING FOR COMPUTER VISION

1. Biological inspiration –

This section explains the difference between the workflow of Classical Machine learning and Deep Learning. Machine Learning involves hand designed feature engineering. In Deep Learning, the algorithm itself learn complex features and representation of the data by finding a mapping function between input and output.

Solving problems with deep learning requires identifying some pattern in the world, finding examples that highlight both sides of the pattern (the input and the output), and then letting a "neural network" learn the map between the two. This opens the types of problems where computers can help us to those where we:

1. Have identified a pattern within a problem
2. Have enough data that exemplifies the pattern

In the first game (Louie or not louie) we just performed a fundamental task of computer vision: image classification, where a computer can take an image that it has never seen before and correctly attribute it to the appropriate "class".

GPU Task 1:

First, I created a *model* by choosing a *dataset(Images of beagles)*, a *neural network*, and how many *training epochs* to run. This highlighted two concepts:

1. Deep Neural Networks are flexible algorithms inspired by the human brain that allow practitioners to use training strategies inspired by human learning. The input of an image generated an output of the network's confidence that the image belonged to one of two classes.
2. Deep Neural Networks fall very short of the human brain. After one epoch, our model was predicting no better than chance: 50/50.

NVIDIA CERTIFICATION LEARNINGS

In this task, the model looked at each image once. Each time it saw an image, it generated an output, or prediction, about whether that image contains Louie or does not contain Louie. After each prediction, it learned the right answer, adjusted its model, and then tried again.

Experiment 1 :

Batch size - 2 , Epochs = 5 , Learning rate = 0.01 , Optimizer = SGD, Network = LeNet

Nikhil_GPU1_Exp1 Image Classification Model



Predictions

Louie	51.54%
Not Louie	48.46%

Experiment 2 :

Batch size - 2 , Epochs = 100 , Learning rate = 0.001 , Optimizer = SGD, Network = AlexNet

Select Dataset ?

Images of Beagles

Images of Beagles

Done Nov 02 2017, 06:03:31 PM

Image Size
256x256

Image Type
COLOR

DB backend
Imdb

Create DB (train)
16 images

Solver Options

Training epochs ?
100

Snapshot interval (in epochs) ?
1.0

Validation interval (in epochs) ?
1.0

Random seed ?
[none]

Batch size ? multiples allowed
2

Batch Accumulation ?

Data Transformations

Subtract Mean ?

Image

Crop Size ?
none

Nikhil_GPU1_Exp2 Image Classification Model



Predictions

Louie	100.0%
Not Louie	0.0%

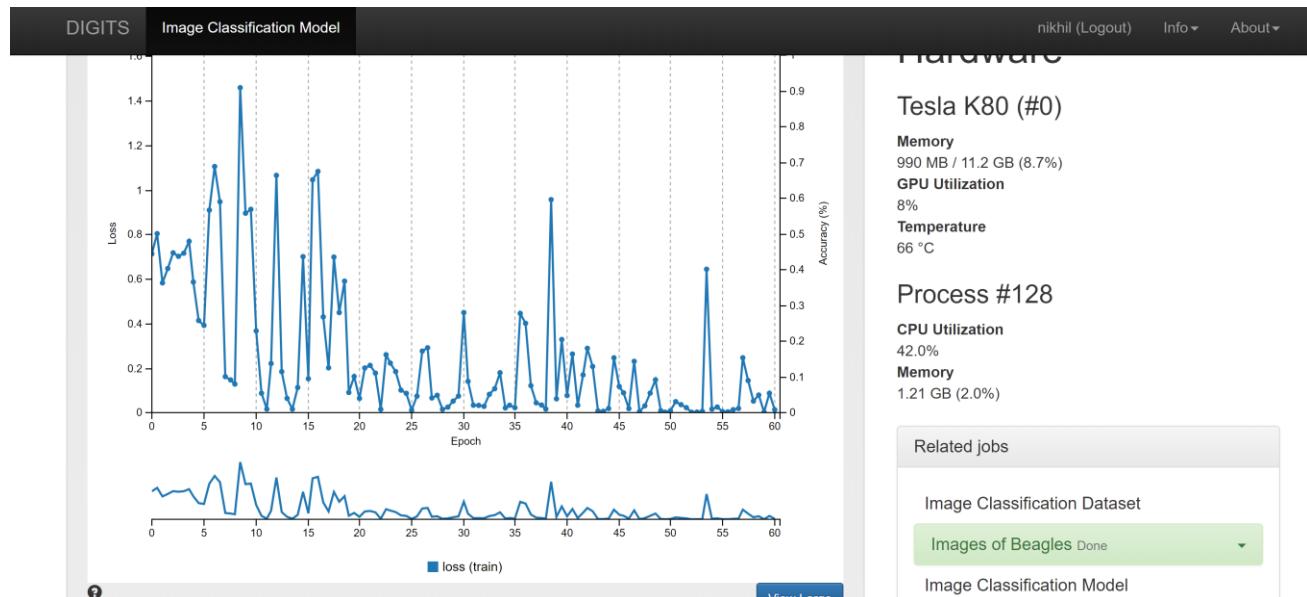
Job Status Done

- Initialized at 12:19:10 AM (1 second)
- Running at 12:19:11 AM (4 seconds)
- Done at 12:19:15 AM (Total - 5 seconds)

Infer Model Done ▾

Notes

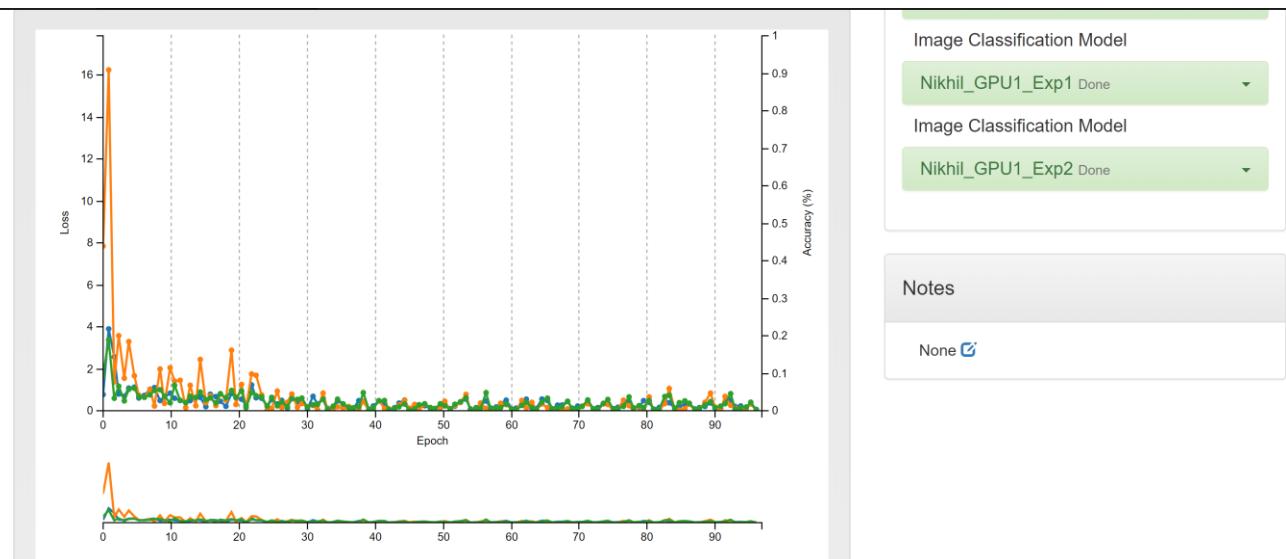
None



Experiment 3 :

Batch size - 2 , Epochs = 100 , Learning rate = 0.005 , Optimizer = Adadelta, Network = GoogleNet

NVIDIA CERTIFICATION LEARNINGS



Nikhil_GPU1_Exp3 Image Classification Model



Predictions

Louie	93.69%
Not Louie	6.31%

Experiment 4 :

Batch size - 2 , Epochs = 100 , Learning rate = 0.001 , Optimizer = Adam, Network = Pretrained Network GPU_Exp3

NVIDIA CERTIFICATION LEARNINGS

Standard Networks Previous Networks Pretrained Networks Custom Network

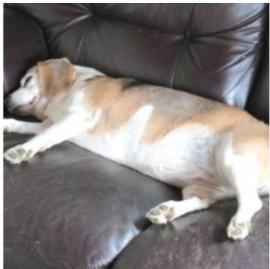
Caffe Torch Tensorflow

Pretrained Model

• Nikhil_GPU1_Exp3 **caffe**

Customize

Nikhil_GPU1_Exp4 Image Classification Model



Predictions

Not Louie	81.74%
Louie	18.26%

Job Status Done

- Initialized at 12:43:32 AM (1 second)
- Running at 12:43:33 AM (3 seconds)
- Done at 12:43:37 AM
(Total - 4 seconds)

Infer Model Done ▾

Notes

None

Out of all the models, AlexNet network model performed the best with a 100% accuracy.

GPU TASK 2 : BIG DATA

In this Task, we combined these three below ingredients to train a neural network that is effective in the real world. We'll measure our success by the model's performance on new data.

1. Deep Neural Networks
2. The GPU
3. Big data

The data that a neural network is exposed to during training is the only guidance it has about the world. When you learned to identify Louie, you began with an understanding of not only dogs, but even of features that might differentiate between dogs (color, size, etc.) A neural network still just interprets data as data, so has to discover which features are useful.

Creating a model that is effective with *new data*

DB backend: LMDB

Image Encoding: PNG (lossless)

Group Name:

Dataset Name: Nikhil_Dogs&Cats

Create

Dataset - Nikhil_Cats&Dogs

Nikhil_Dogs&Cats

Owner: nikhil

Clone Job Abort Job Delete Job

Job Information		Parse Folder (train/val)	Job Status
Job Directory	/dli/data/digits/20190225-005823-c308	Folder	Running
Image Dimensions	256x256 (Width x Height)		• Initialized at 12:58:23 AM (1 second)
Image Type	Color		• Running at 12:58:25 AM
Resize Transformation	Squash	Parse Folder (train/val) Running	
DB Backend	Imdb	Create DB (train) Running	
Image Encoding		45%	
			Estimated time remaining: 4 minutes, 30 seconds
			• Initialized at 12:58:23 AM (3 seconds)

Let's Create Models now and train it on this Dataset and test on 25% validation data from this dataset.

New Image Classification Model

Select Dataset ?

Images of Beagles

Nikhil_Dogs&Cats

Done 01:04:51 AM

Image Size
256x256

Image Type
COLOR

Solver Options

Training epochs ?

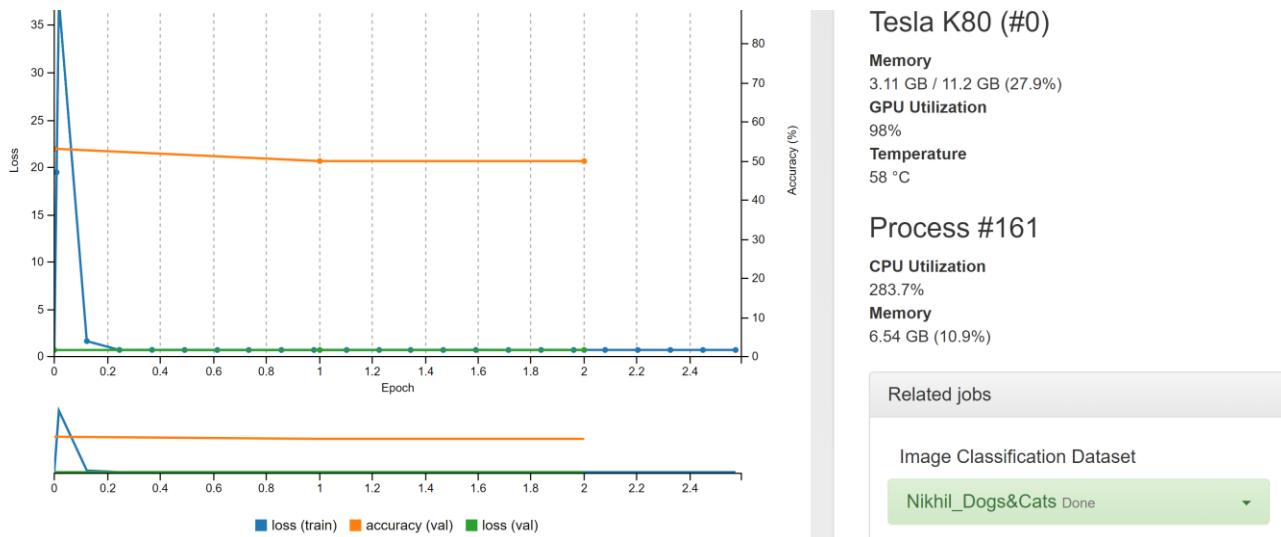
Snapshot interval (in epochs) ?

Validation interval (in epochs) ?

Random seed ?

Experiment 1 -

Batch size - 2, Epochs = 20, Learning rate = 0.01 , Optimizer = Adam, Network = AlexNet

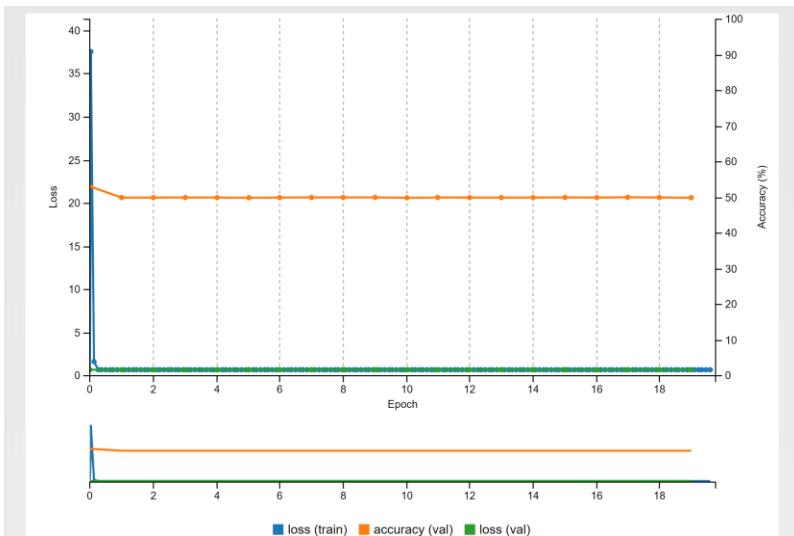


NVIDIA CERTIFICATION LEARNINGS

Job Directory
 /dli/data/digits/20190225-010839-c0ac
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
Nikhil_Dogs&Cats
 Done 01:04:51 AM
Image Size
 256x256
Image Type
 COLOR
DB backend
 lmdb
Create DB (train)
 18750 images
Create DB (val)
 6250 images

Job Status Running
 • Initialized at 01:08:39 AM (1 second)
 • Running at 01:08:40 AM
Train Caffe Model Running ▾
 95%
 Estimated time remaining: 49 seconds
 • Initialized at 01:08:39 AM (1 second)
 • Running at 01:08:40 AM



Tesla K80 (#0)
Memory
 3.11 GB / 11.2 GB (27.9%)
GPU Utilization
 99%
Temperature
 61 °C
Process #161
CPU Utilization
 309.7%
Memory
 8.61 GB (14.4%)
Related jobs
 Image Classification Dataset
 Nikhil_Dogs&Cats Done ▾

With 20 epochs, it took 16 minutes to train the model.

Lets test using a unseen image - /dli/data/BeagleImages/Louie/louie5.JPG

Nikhil_GPU2_Exp1 Image Classification Model



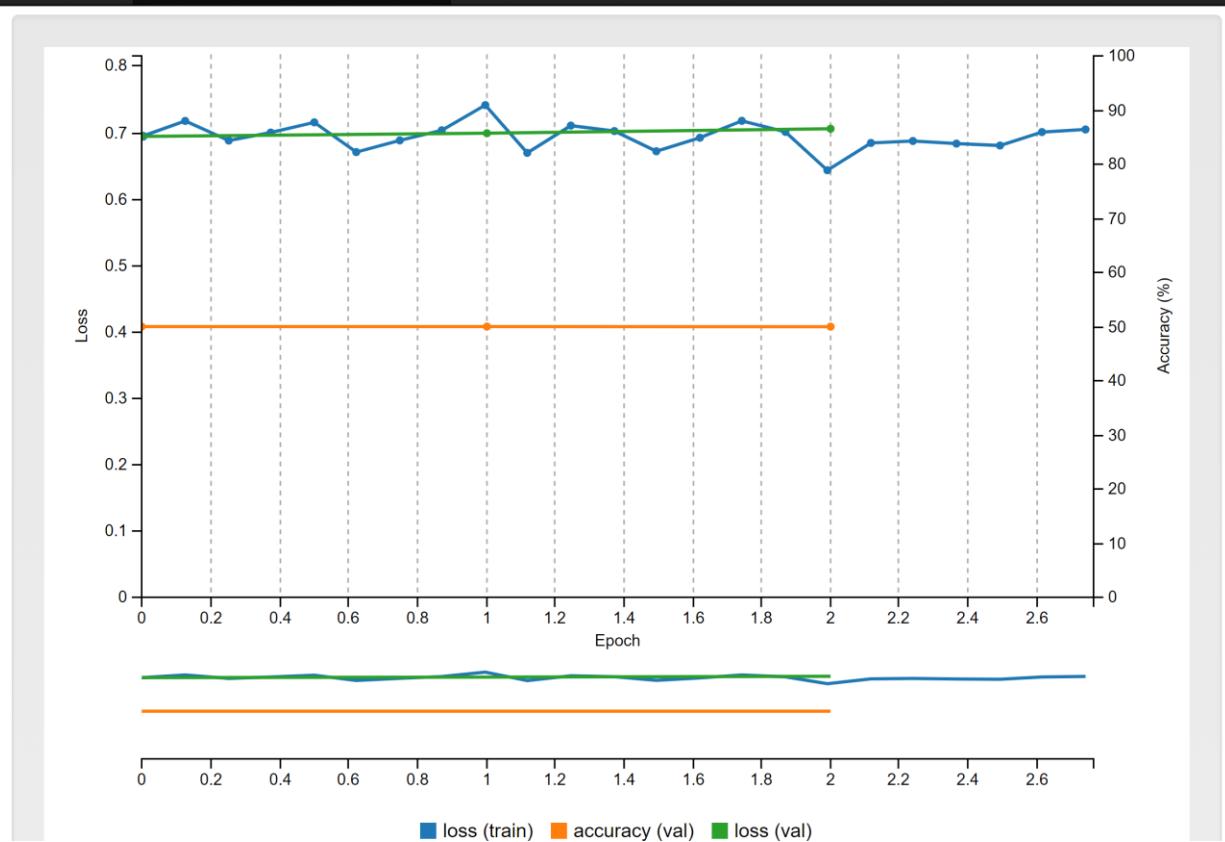
Predictions

dogs	50.01%
cats	49.99%

Accuracy was 50.01%. Lets change some hyperparameters and try again .

Experiment 2 -

Batch size - 16, Epochs = 10, Learning rate = 0.01 , Optimizer = Adam, Network = Pretrained Network GPU2_Exp1



Nikhil_GPU2_Exp2 Image Classification Model



Predictions	
dogs	50.06%
cats	49.94%

Job Status Done

- Initialized at 01:42:30 AM (1 second)
- Running at 01:42:31 AM (4 seconds)
- Done at 01:42:36 AM
(Total - 5 seconds)

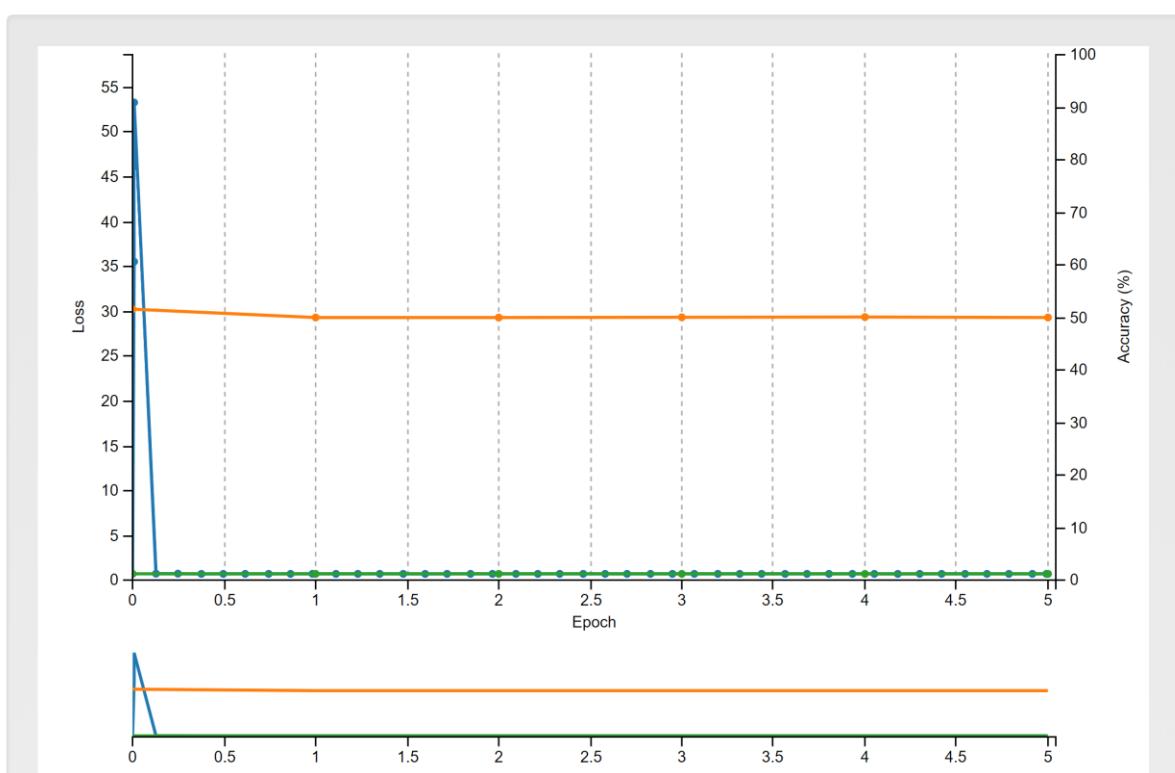
Infer Model Done ▾

Notes

None

Experiment 3 -

Batch size - 64, Epochs = 5, Learning rate = 0.005 , Optimizer = AdaGrad, Network = AlexNet



Nikhil_GPU2_Exp3 Image Classification Model

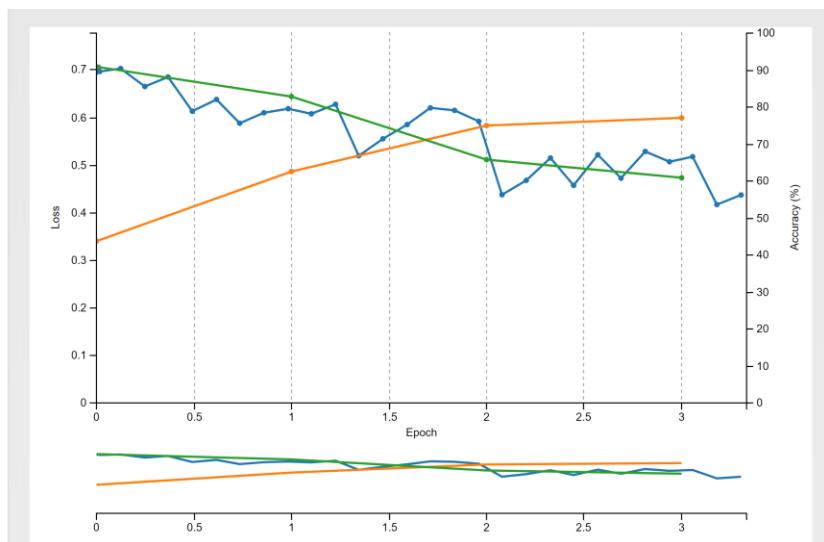


Predictions

cats	50.09%
dogs	49.91%

Experiment 4 -

Epochs = 5, Learning rate = 0.001 , Optimizer = Adam, Network = AlexNet , Subtract Mean - Image



Hardware

Tesla K80 (#0)

Memory
2.9 GB / 11.2 GB (25.9%)
GPU Utilization
99%
Temperature
65 °C

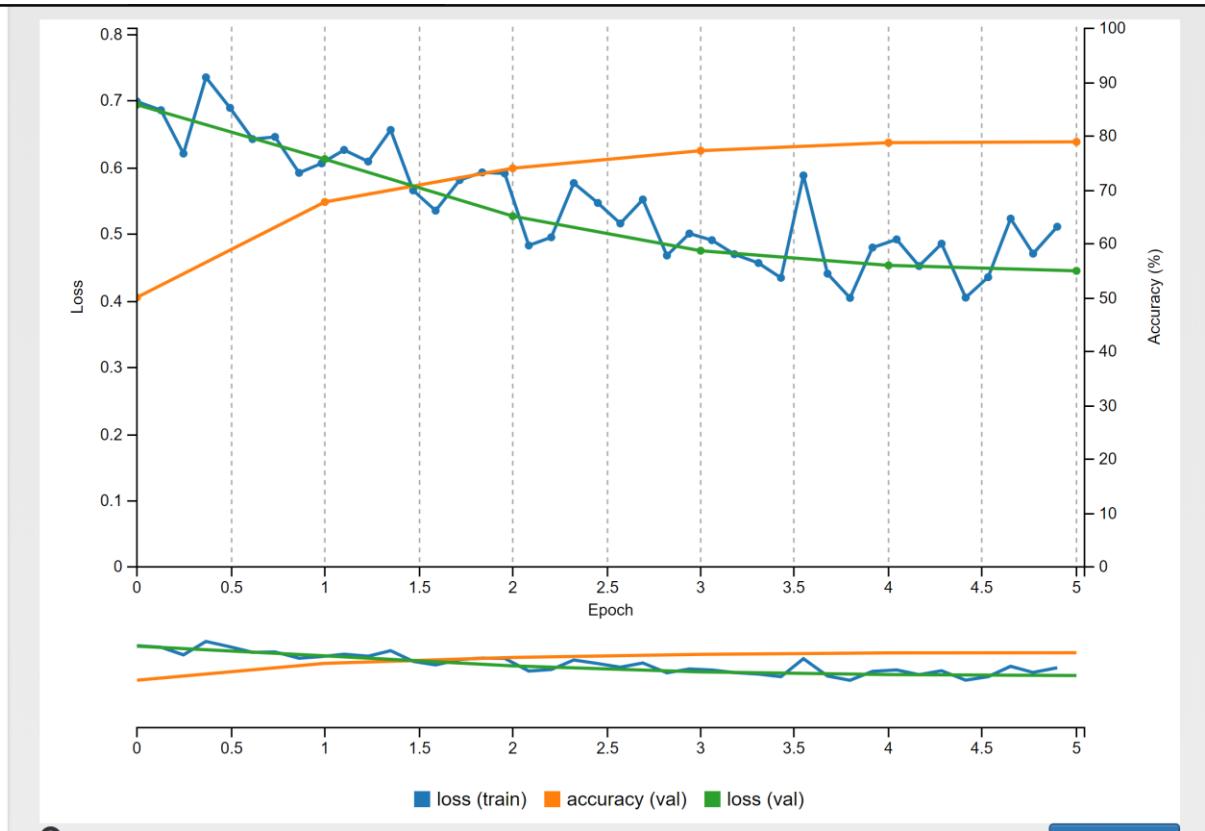
Process #96

CPU Utilization
140.9%
Memory
11.5 GB (19.3%)

Related jobs

Image Classification Dataset

Dogs and Cats Done



Accuracy is the highest in AlexNet model

GPU TASK 3 :

We just trained a neural network model to classify dogs from cats. In this task, we will *deploy* that trained model into a simulator.

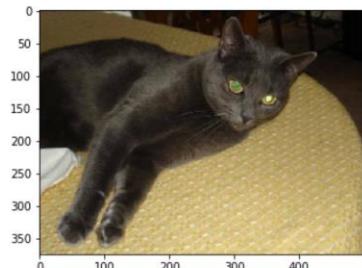
1. How to deploy a trained model into an application.
2. What role a trained model plays within an application.
3. To identify and use the pieces of a trained model

We also saw that a trained network consists of two components:

1. A description of the architecture of the untrained network
2. The weights that were "learned" while the network trained

NVIDIA CERTIFICATION LEARNINGS

```
In [5]: M import matplotlib.pyplot as plt #matplotlib.pyplot allows us to visualize results  
input_image= caffe.io.load_image('/dli/data/dogscats/train/cats/cat.10941.jpg')  
plt.imshow(input_image)  
plt.show()
```



While this is the image we have, it is not the 'input' the network expects.

To prepare data for inference, we're going to follow one golden rule:

Whatever was done prior to training must be done prior to inference

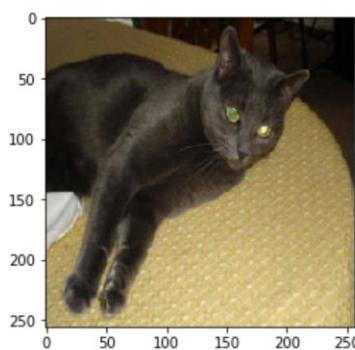
Replace `DATA_JOB_DIR` with the directory you created earlier and execute the code below to set `DATA_JOB_DIR` to the right location and examining what's inside.

```
In [6]: M DATA_JOB_DIR = '/dli/data/digits/20180222-165843-ada0' ## Remember to set this to be the job directory for your model  
!ls $DATA_JOB_DIR  
create_train_db.log labels.txt mean.jpg train.txt val.txt  
create_val_db.log mean.binaryproto status.pickle train_db val_db
```

Again, there is more information here than you need (for now). There is an infinite amount that you *could* know about data science and data prep which will become clear as you work through a variety of deep learning problems. In this case, DIGITS did two steps prior to training. We call this *preprocessing*.

```
print "Sorry cat:( https://media.giphy.com/media/jb8aFEQk3tADS/giphy.gif"  
else:  
    print "Welcome dog! https://www.flickr.com/photos/aidras/5379402670"
```

Input image:



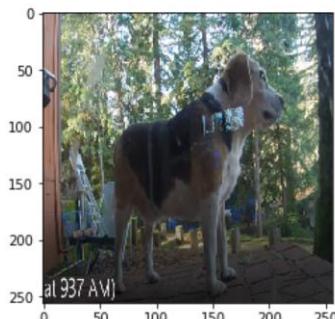
Output:

Sorry cat:(<https://media.giphy.com/media/jb8aFEQk3tADS/giphy.gif>

NVIDIA CERTIFICATION LEARNINGS

```
print "Sorry cat! https://media.giphy.com/media/Jb8aFEQK3tADS/giphy.gif"
else:
    print "Welcome dog! https://www.flickr.com/photos/aidras/5379402670"
```

Input Image:



```
[[ 0.39924237  0.6007576 ]]
```

Output:

```
Welcome dog! https://www.flickr.com/photos/aidras/5379402670
```

Essentially, we've created a simulator for our doggy door challenge. We've created an application that takes an input from a camera, converts it to a data type our network expects, generates an output, and then converts that output into something useful to a user.

And then run our small python application with that image as input below. Ignore most of the output and scroll to the bottom. (Even errors and warnings are fine.)

```
In [14]: █ ipython pythondeployment.py $TEST_IMAGE 2>/dev/null
[[ 0.42844081  0.57155913]]
Output:
Welcome dog! https://www.flickr.com/photos/aidras/5379402670
None
```

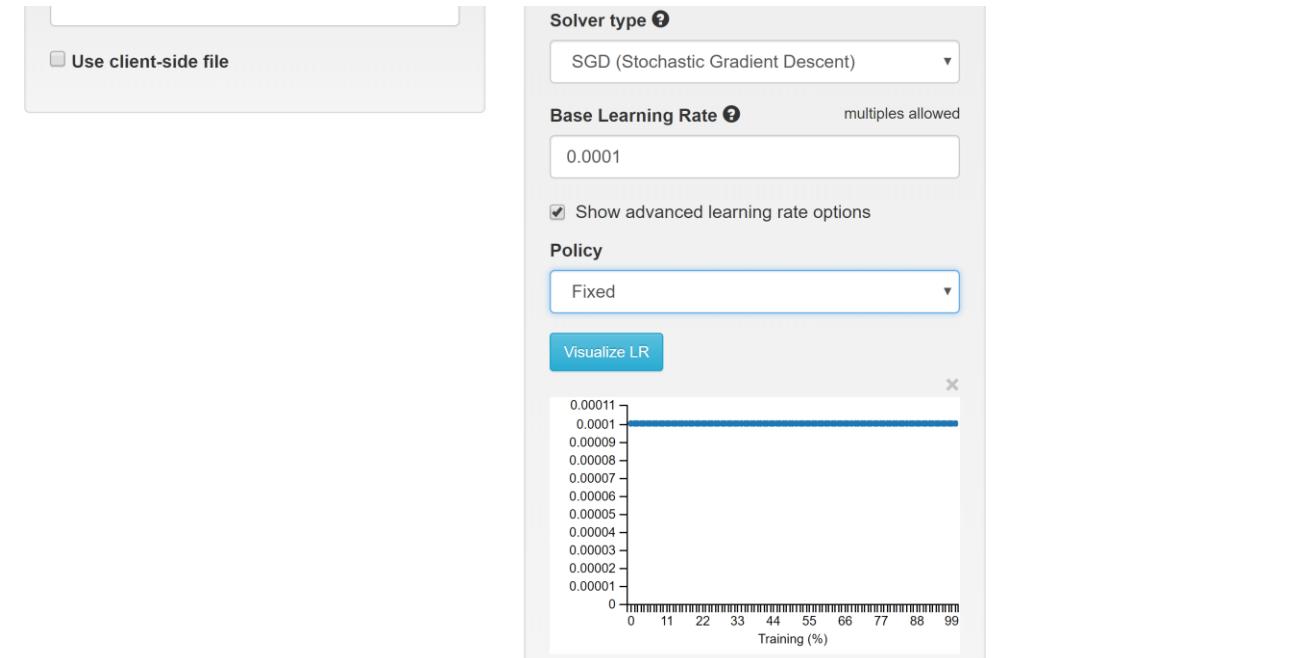


GPU TASK 4:

In this next section, we will engage with some strategies for improving training performance using our existing model. You will learn:

NVIDIA CERTIFICATION LEARNINGS

- To run more training epochs on an existing model, analogous to a human learner *studying more*.



The screenshot shows the DIGITS web interface for creating a new model. At the top, there is a navigation bar with tabs for "DIGITS", "New Model", and user information "nikhil (Logout) Info ▾ About ▾". Below the navigation bar, there are tabs for "Standard Networks", "Previous Networks", "Pretrained Networks" (which is selected), and "Custom Network". Under the "Pretrained Networks" tab, there are three options: "Caffe", "Torch", and "Tensorflow", with "Caffe" being the active choice. Below these tabs is a section titled "Pretrained Model" with a radio button selected for "Dogs vs. Cats" (using "caffe"). To the right of this section is a "Customize" link. At the bottom of the page, there is a modal window for creating a new group and model. It contains fields for "Group Name" (set to "Nikhil") and "Model Name" (set to "Nikhil_GPU4_Exp1"). A "Create" button is located at the bottom of this modal.

NVIDIA CERTIFICATION LEARNINGS

DIGITS Image Classification Model nikhil (Logout) Info ▾ About ▾

Nikhil_GPU4_Exp1

Owner: nikhil

[Clone Job](#) [Abort Job](#) [Delete Job](#)

Job Directory
/dli/data/digits/20190225-025220-d7bb

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

Pretrained Model

Dataset
Dogs and Cats
Done Feb 22 2018, 05:03:28 PM

Image Size
256x256

Image Type
COLOR

DB backend
lmdb

Create DB (train)
18750 images

Create DB (val)
6250 images

Job Status Running

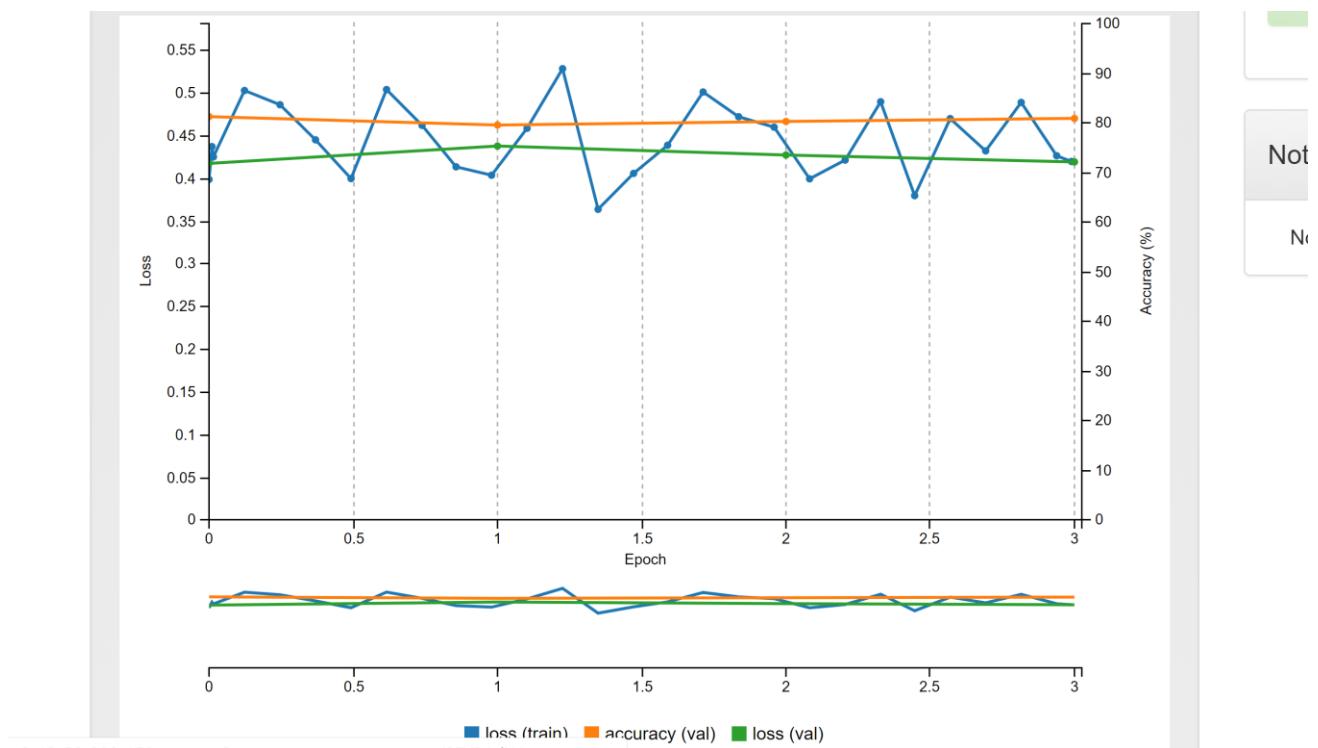
- Initialized at 02:52:20 AM (1 second)
- Running at 02:52:21 AM

Train Caffe Model Running

0%

Estimated time remaining: 44 minutes, 48 seconds

- Initialized at 02:52:20 AM (1 second)
- Running at 02:52:21 AM



- As expected, the accuracy starts close to where our first model left off, 80%.
- Accuracy DOES continue to increase, showing that increasing the number of epochs often does increase performance.
- 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.

Deploying award winning models

Instead of working through the deployment of the model you just retrained, it's time to introduce a shortcut to performance: deploying expert pretrained models.

In this section, you'll learn to deploy other people's networks so that you can get the performance gains of their research, compute time, and data curation.

NVIDIA CERTIFICATION LEARNINGS

Let's use what we already know about deployment to run an image through this model. Start by initializing the model:

```
In [3]: import caffe
import numpy as np
caffe.set_mode_gpu()
import matplotlib.pyplot as plt #matplotlib.pyplot allows us to visualize results

ARCHITECTURE = 'deploy.prototxt'
WEIGHTS = 'bvlc_alexnet.caffemodel'
MEAN_IMAGE = 'ilsvrc_2012_mean.npy'
TEST_IMAGE = '/dli/data/BeagleImages/louietest2.JPG'

# Initialize the Caffe model using the model trained in DIGITS
net = caffe.Classifier(ARCHITECTURE, WEIGHTS) #Each "channel" of our images are 256 x 256
```

Then create an input the network expects. Note that this is different than the *preprocessing* used in the last model. To learn how imagenet was preprocessed, the documentation was clearly presented on the [Caffe website](#):

```
In [4]: #Load the image
image= caffe.io.load_image(TEST_IMAGE)
```

Visualize the output –

```
In [5]: # copy the image data into the memory allocated for the net
net.blobs['data'].data[...] = transformed_image

### perform classification
output = net.forward()

output
```

```
Out[5]: {'prob': array([[ 2.31780262e-09,  2.58294519e-09,  3.19525961e-09,
       2.09216755e-09,  5.00786212e-09,  2.04342987e-09,
       2.37229258e-09,  9.16246246e-11,  4.86508278e-10,
       5.01131137e-09,  1.35739935e-08,  9.87543114e-09,
       3.42600948e-11,  1.11983944e-09,  3.58725938e-10,
       7.21391349e-11,  1.98810324e-09,  3.15641522e-08,
       3.184066570e-08,  7.21174453e-10,  8.27692492e-09,
       1.42624259e-08,  2.83560397e-09,  3.29977219e-08,
       3.40230094e-10,  7.50220686e-09,  9.47929624e-10,
       1.18161825e-09,  2.00934576e-08,  2.79246071e-10,
       1.43229650e-09,  2.25081864e-09,  8.54826698e-09,
       1.04760878e-09,  3.35014100e-10,  1.14679100e-10,
       8.23971502e-10,  2.29677444e-10,  1.93692991e-08,
       3.21901672e-10,  2.40228504e-09,  1.76278014e-09,
       2.23937793e-08,  5.04234487e-10,  4.73921236e-10,
       5.41517942e-10.  1.59301594e-09.  2.94658253e-09.
```

Lets see which class this image belongs to by checking the maximum probability –

Work to make the output useful to a user.

What you see above is an array containing the probabilities that our image belongs to each of 1000 classes. Let's work to make this useful.

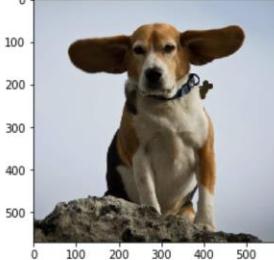
```
In [8]: output_prob = output['prob'][0] # the output probability vector for the first image in the batch
print 'predicted class is:', output_prob.argmax()
```

```
predicted class is: 162
```

This is closer. Take a look at imagenet's classes to see what that number corresponds to [here](#). Better???? Let's add that functionality to our application so we have a useful end-to-end deployment.

```
In [10]: M print ("Input image:")
plt.imshow(image)
plt.show()

print("Output label:" + labels[output_prob.argmax()])

Input image:

Output label:n02088364 beagle
```

GPU TASK 5 : IMAGE DETECTION

The biggest takeaway here is that Deep Neural Networks turn one block of data (tensor) into another block of data (tensor). What that data represents is flexible. Our job is to find a pairing that helps us to solve the problem we've set out to solve, in this case, raw data to the (X,Y) pairings of Louie's boundaries within an image.

Workflow	Input	Output		
Image Classification	Raw Pixel Values	A vector where each index corresponds with the likelihood of the image belonging to each class		
Object Detection	Raw Pixel Values	A vector with (X,Y) pairings for the top-left and bottom-right corner of each object present in the image		
Image Segmentation	Raw Pixel Values	A overlay of the image for each class being segmented, where each value is the likelihood of that pixel belonging to each class		
Text Generation	A unique vector for each 'token' (word, letter, etc.)	A vector representing the most likely next 'token'		
Image Rendering	Raw Pixel Values of a grainy Image	Raw pixel values of a clean image		

We're going to use what's called the "sliding window" approach, where we'll take split out image into small sections which we'll call grid squares. We're run

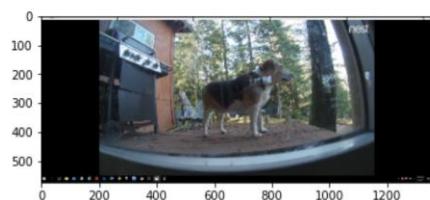
NVIDIA CERTIFICATION LEARNINGS

each grid square through an image classifier. If that grid square contains an image of a dog, we'll have localized Louie in the image.

From there you will work to use deep learning for more and more of your workflow, learning to adjust neural network architecture and finally, a completely new object detection specific workflow.

Next, we'll take an image directly from a front door security camera. Note, this image is larger than 256X256. We want to know where a dog is in this image.

```
In [2]: # Choose a random image to test against
#RANDOM_IMAGE = str(np.random.randint(10))
IMAGE_FILE = '/dli/tasks/task5/task/images/LouieReady.png'
input_image= caffe.io.load_image(IMAGE_FILE)
plt.imshow(input_image)
plt.show()
```



Using our *learned* function: Forward Propagation

```
# Display total time to perform inference
print 'Total inference time: ' + str(end-start) + ' seconds'
```



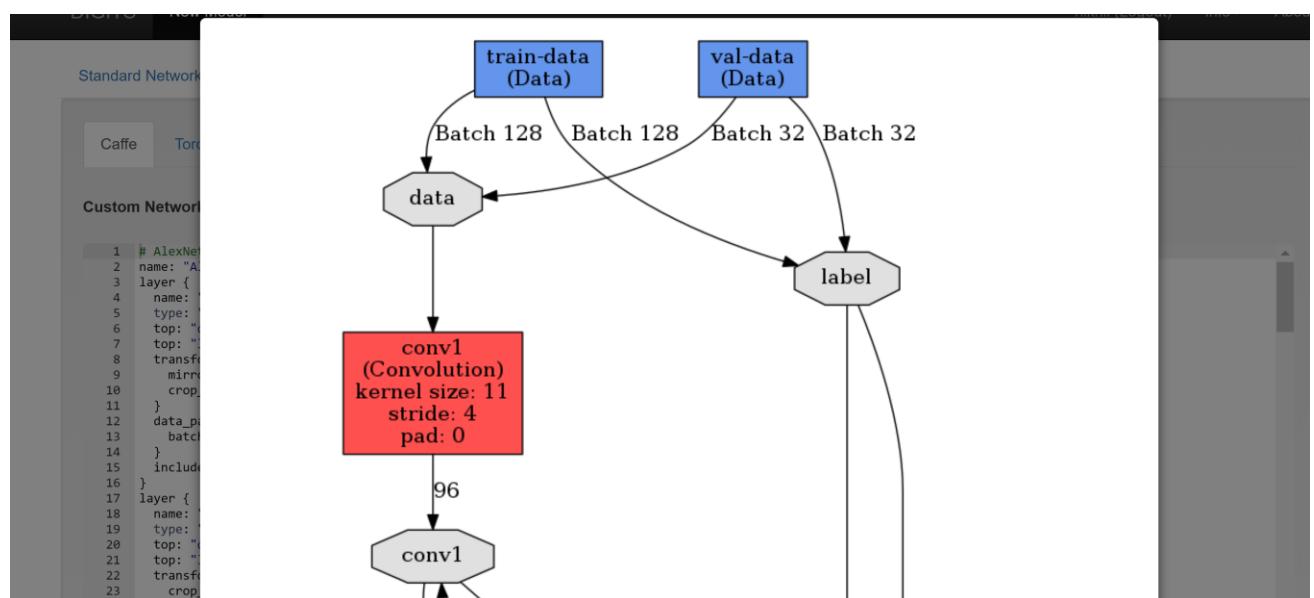
Total inference time: 0.801002979279 seconds



Again, this is the output of a randomly chosen wide area test image along with an array showing the predicted class for each

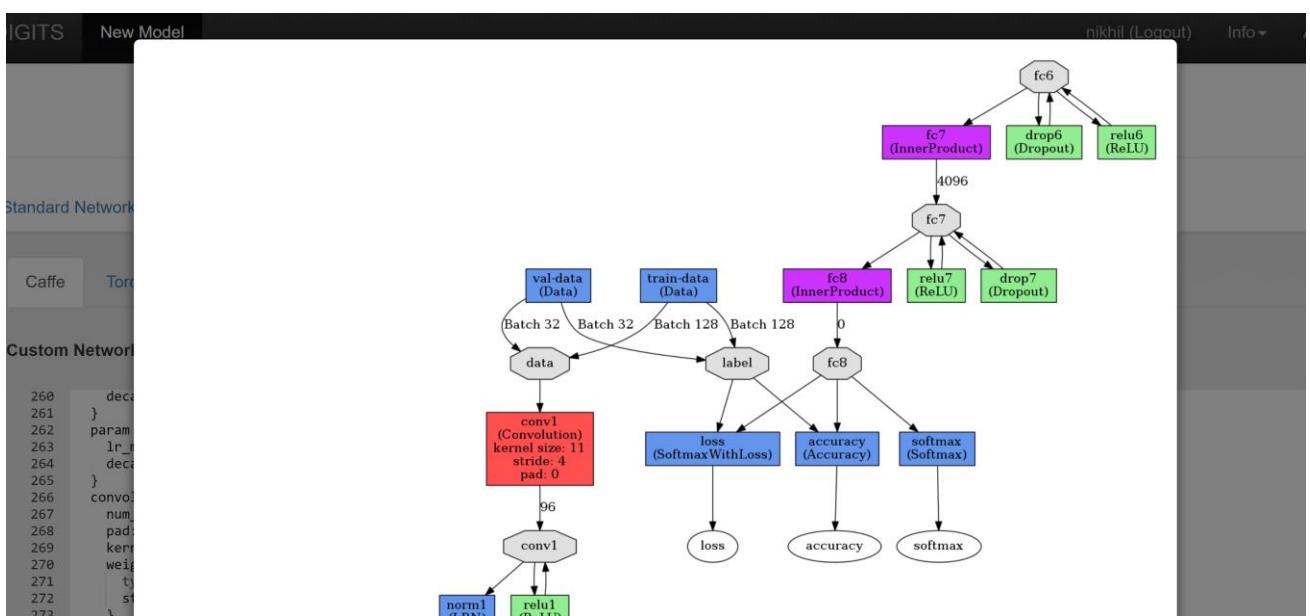
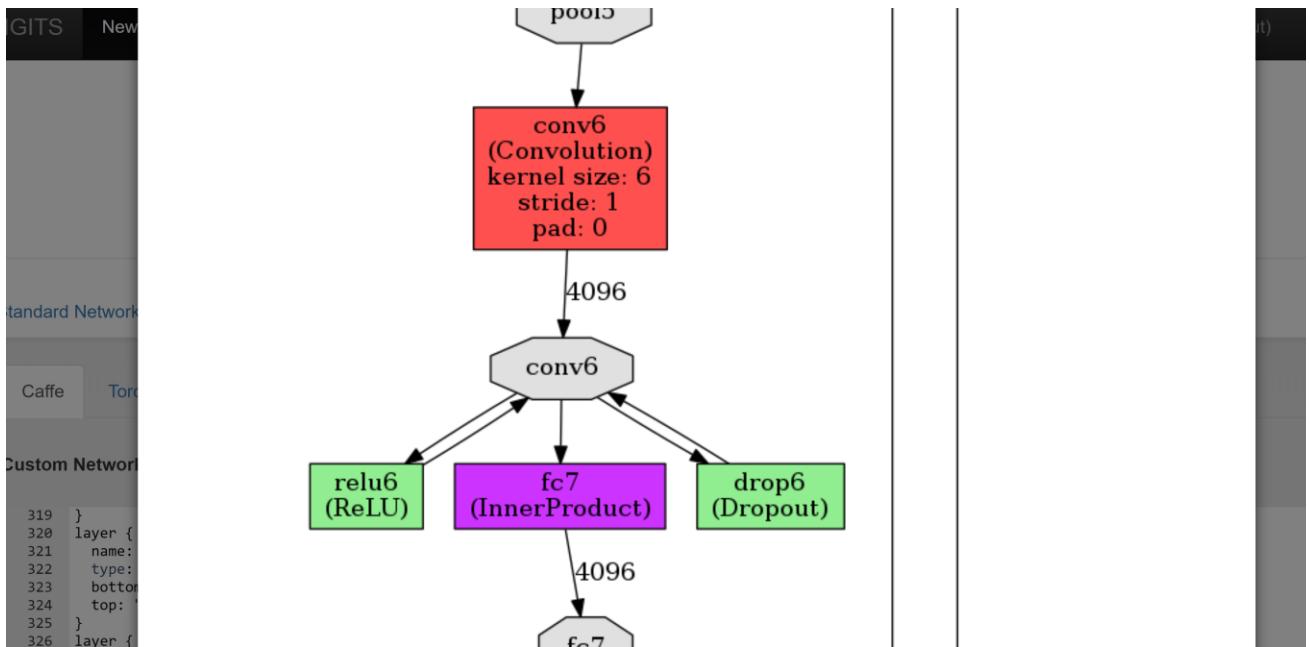
As we have now seen, the advantage of this sliding window approach is that we can train a detector using only patch-based training data (which is more widely available) and that with our current skillset, we can start working to solve the problem.

Approach 2 – rebuilding from an existing Deep learning network



We have changed the layer fc6 to conv6 now -

NVIDIA CERTIFICATION LEARNINGS



NVIDIA CERTIFICATION LEARNINGS

```
376     exclude { stage: "deploy" }
377   }
378   layer 5
```

Pretrained model(s) ?

Group Name ?

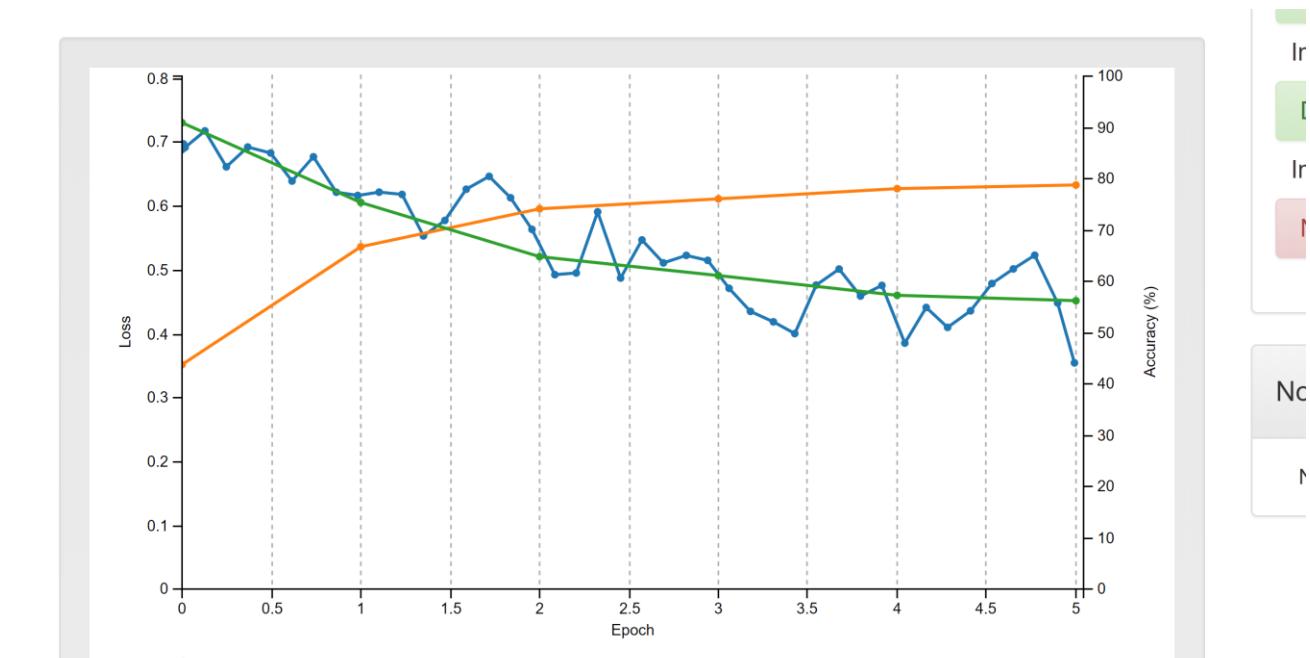
Model Name ?

Nikhil_GPU5_New

Create

Create a new model now with the customized AlexNet -

/dli/data/digits/20190225-041311-3325





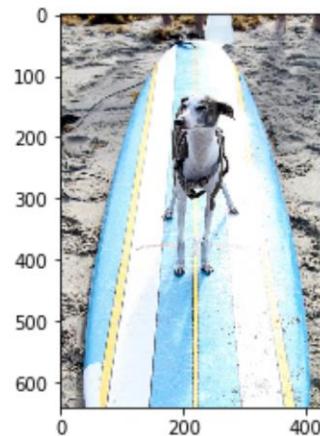
Approach 3: DetectNet

We started this lab with AlexNet, an elegant solution to the very specific problem of *image classification*. We built some python around it and then performed minor brain surgery to accomplish a task that we wouldn't have been able to prior to our work in deep learning, *object detection*.

However, is there an elegant solution to *object detection*? Could we build a model that maps directly from the input you have: photos of various sizes, to the output you want: localization and detection information?

The fact that there is will help us to broaden our definition of Deep Learning and provide some insight into what *other* problems you can solve with it.

```
In [12]: # input_image = caffe.io.load_image('/dli/data/train/images/488156.jpg')
plt.imshow(input_image)
plt.show()
```



Next, its corresponding label:

```
In [13]: !cat '/dli/data/train/labels/488156.txt' # "cat" has nothing to do with the animals, this displays the text
dog 0 0 0 161.29 129.03 291.61 428.39 0 0 0 0 0 0 0
surfboard 0 0 0 31.25 36.44 410.41 640.0 0 0 0 0 0 0 0
```

Note:

DIGITS

New Dataset

nikhil (Logout)

Info ▾

New Object Detection Dataset

Object Detection Dataset Options

Images can be stored in any of the supported file formats ('.png','.jpg','.jpeg','.bmp','.ppm').

Training image folder ?

/dli/data/train/images

Label files are expected to have the .txt extension. For example if an image file is named foo.png the corresponding label file should be foo.txt.

Training label folder ?

/dli/data/train/labels

Validation image folder ?

/dli/data/val/images



New Object Detection Model

Select Dataset [?](#)

coco-dog
Running
• DB backend: lmdb

Solver Options

Training epochs [?](#)

Snapshot interval (in epochs) [?](#)

Validation interval (in epochs) [?](#)

Random seed [?](#)

Data Transformations

Subtract Mean [?](#)

Crop Size [?](#)

Python Layers [?](#)

```
2543     module: 'caffe.layers.detectnet.mean_ap'
2544     layer: 'mAP'
2545     param_str : '640, 640, 16'
2546   }
2547   include: { phase: TEST stage: "val" }
2548 }
2549 }
```

Pretrained model(s) [?](#)

Group Name [?](#)

Model Name [?](#)

[Create](#)

NVIDIA CERTIFICATION LEARNINGS

Job Directory
/dli/data/digits/20190225-044319-6349

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

Pretrained Model
/dli/data/digits/snapshot_iter_38600.caffemodel

Visualizations

Tensorboard

Dataset
coco-dog

Done 04:40:16 AM

- DB backend: Imdb
- Create train_db DB
 - **Entry Count:** 3855
 - **Feature shape:** (3, 640, 640)
 - **Label shape:** (1, 52, 16)
- Create val_db DB
 - **Entry Count:** 1969
 - **Feature shape:** (3, 640, 640)
 - **Label shape:** (1, 51, 16)

Job Status Running

- Initialized at 04:43:19 AM (1 second)
- Running at 04:43:20 AM

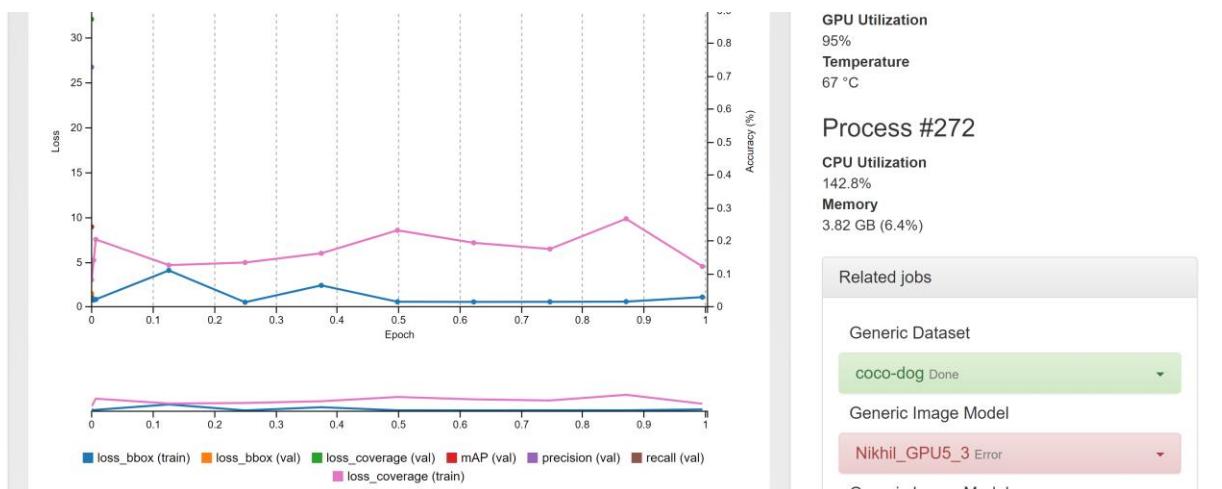
Train Caffe Model Running ▾

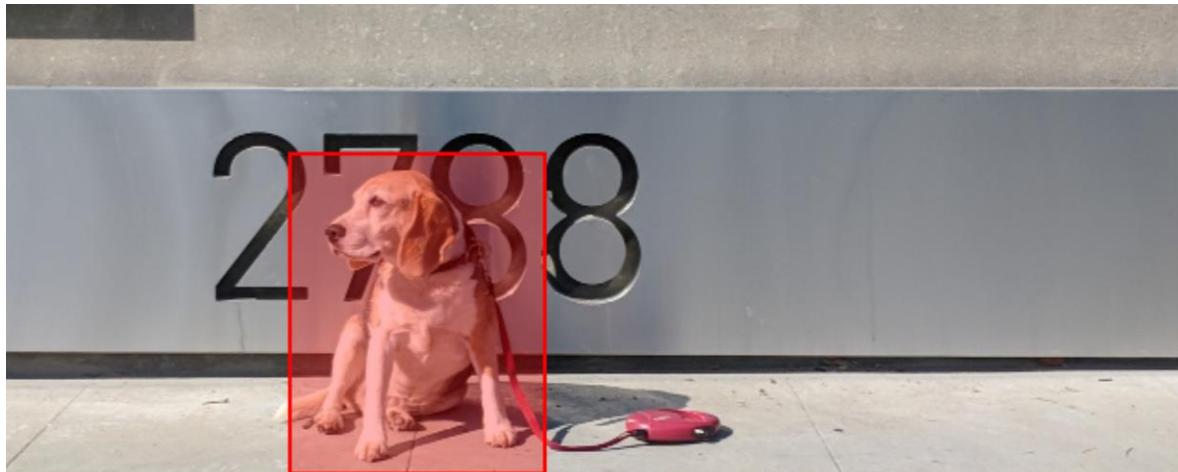
0%

Estimated time remaining: 36 minutes, 49 seconds

- Initialized at 04:43:19 AM (1 second)
- Running at 04:43:20 AM

Hardware





■ bbox-list

[Bookmark this page](#)

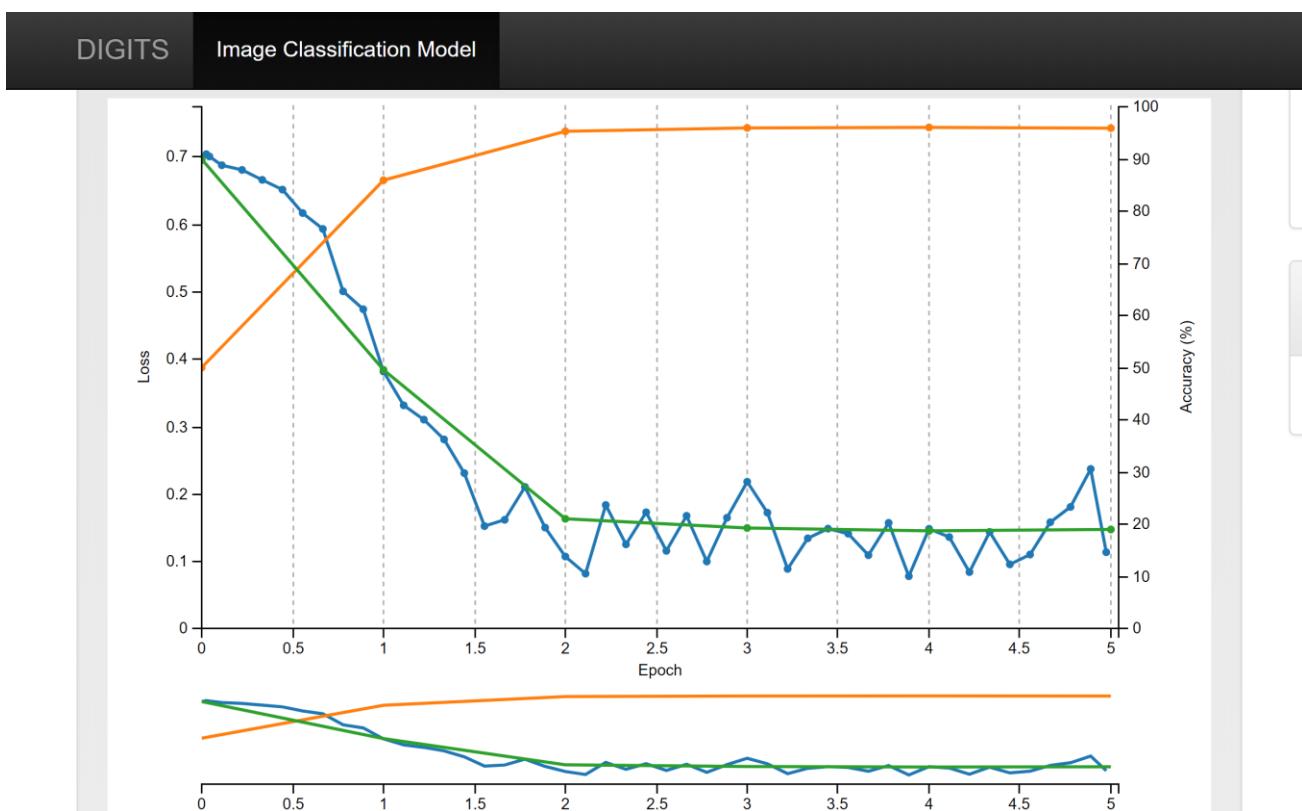
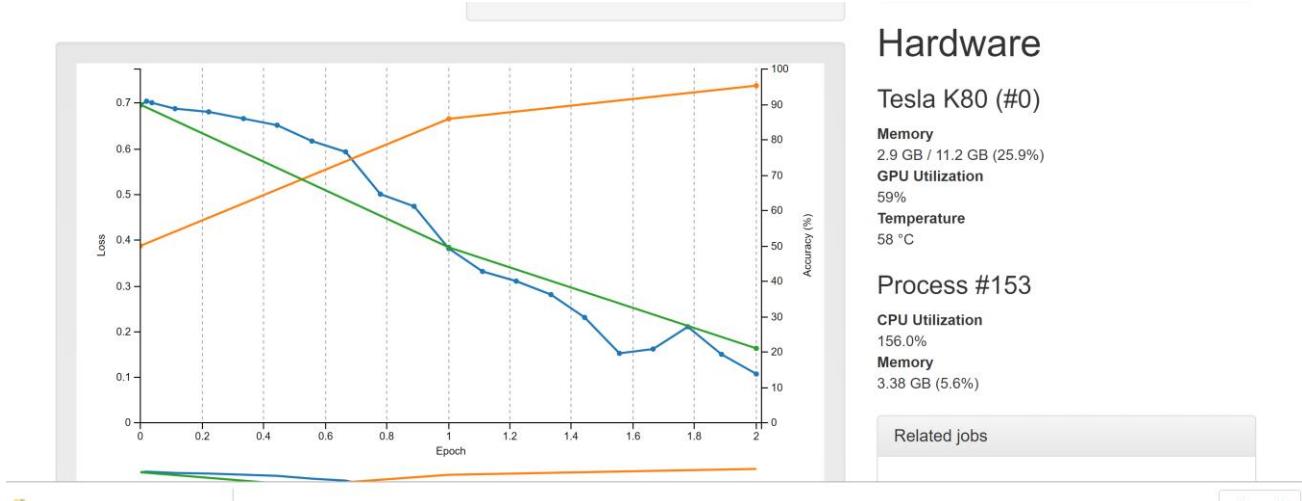
Congrats. You've just completed your first course in deep learning. At this point, you have learned to:

1. Identify the ingredients required for deep learning
2. Train a Deep Neural Network to correctly classify images it has never seen before
3. Deploy Deep Neural Networks into applications
4. Identify techniques for improving the performance of deep learning applications
5. Assess the types of problems that are candidates for deep learning
6. Modify neural networks to change behavior

Go to the next section to attempt a small deep learning problem on your own and earn a certificate showing your mastery of the course.

Assesement -

NVIDIA CERTIFICATION LEARNINGS



Nikhil_Asselement1

Image Classification Model



Predictions

face	93.26%
not face	6.74%

```
In [14]: M !python submission.py '/dli/data/whale/data/train/face/w_1.jpg' #This should return "whale" at the very bottom
I0225 05:34:20.351789 271 net.cpp:1137] Copying source layer relu2 Type:ReLU #blobs=0
I0225 05:34:20.351802 271 net.cpp:1137] Copying source layer norm2 Type:LRN #blobs=0
I0225 05:34:20.351809 271 net.cpp:1137] Copying source layer pool2 Type:Pooling #blobs=0
I0225 05:34:20.351814 271 net.cpp:1137] Copying source layer conv3 Type:Convolution #blobs=2
I0225 05:34:20.352250 271 net.cpp:1137] Copying source layer relu3 Type:ReLU #blobs=0
I0225 05:34:20.352264 271 net.cpp:1137] Copying source layer conv4 Type:Convolution #blobs=2
I0225 05:34:20.352610 271 net.cpp:1137] Copying source layer relu4 Type:ReLU #blobs=0
I0225 05:34:20.352624 271 net.cpp:1137] Copying source layer conv5 Type:Convolution #blobs=2
I0225 05:34:20.352847 271 net.cpp:1137] Copying source layer relu5 Type:ReLU #blobs=0
I0225 05:34:20.352861 271 net.cpp:1137] Copying source layer pool5 Type:Pooling #blobs=0
I0225 05:34:20.352867 271 net.cpp:1137] Copying source layer fc6 Type:InnerProduct #blobs=2
I0225 05:34:20.370242 271 net.cpp:1137] Copying source layer relu6 Type:ReLU #blobs=0
I0225 05:34:20.370275 271 net.cpp:1137] Copying source layer drop6 Type:Dropout #blobs=0
I0225 05:34:20.370280 271 net.cpp:1137] Copying source layer fc7 Type:InnerProduct #blobs=2
I0225 05:34:20.378078 271 net.cpp:1137] Copying source layer relu7 Type:ReLU #blobs=0
I0225 05:34:20.378108 271 net.cpp:1137] Copying source layer drop7 Type:Dropout #blobs=0
I0225 05:34:20.378113 271 net.cpp:1137] Copying source layer fc8 Type:InnerProduct #blobs=2
I0225 05:34:20.378137 271 net.cpp:1129] Ignoring source layer loss
```

NVIDIA CERTIFICATION LEARNINGS

```
In [15]: █ python submission.py '/dli/data/whale/data/train/not_face/w_1.jpg' #This should return "not whale" at the very bottom
I0225 05:34:54.473445 286 net.cpp:1137] Copying source layer relu2 Type:ReLU #blobs=0
I0225 05:34:54.473457 286 net.cpp:1137] Copying source layer norm2 Type:LRN #blobs=0
I0225 05:34:54.473464 286 net.cpp:1137] Copying source layer pool2 Type:Pooling #blobs=0
I0225 05:34:54.473470 286 net.cpp:1137] Copying source layer conv3 Type:Convolution #blobs=2
I0225 05:34:54.473963 286 net.cpp:1137] Copying source layer relu3 Type:ReLU #blobs=0
I0225 05:34:54.473978 286 net.cpp:1137] Copying source layer conv4 Type:Convolution #blobs=2
I0225 05:34:54.474313 286 net.cpp:1137] Copying source layer relu4 Type:ReLU #blobs=0
I0225 05:34:54.474328 286 net.cpp:1137] Copying source layer conv5 Type:Convolution #blobs=2
I0225 05:34:54.474562 286 net.cpp:1137] Copying source layer relu5 Type:ReLU #blobs=0
I0225 05:34:54.474576 286 net.cpp:1137] Copying source layer pool15 Type:Pooling #blobs=0
I0225 05:34:54.474583 286 net.cpp:1137] Copying source layer fc6 Type:InnerProduct #blobs=2
I0225 05:34:54.492488 286 net.cpp:1137] Copying source layer relu6 Type:ReLU #blobs=0
I0225 05:34:54.492439 286 net.cpp:1137] Copying source layer drop6 Type:Dropout #blobs=0
I0225 05:34:54.492445 286 net.cpp:1137] Copying source layer fc7 Type:InnerProduct #blobs=2
I0225 05:34:54.500339 286 net.cpp:1137] Copying source layer relu7 Type:ReLU #blobs=0
I0225 05:34:54.500371 286 net.cpp:1137] Copying source layer drop7 Type:Dropout #blobs=0
I0225 05:34:54.500377 286 net.cpp:1137] Copying source layer fc8 Type:InnerProduct #blobs=2
I0225 05:34:54.500403 286 net.cpp:1129] Ignoring source layer loss
not whale
```

NVIDIA Certification Link -

<https://courses.nvidia.com/certificates/53f76e05c1e346b9878cb3cb3875eb4e>



This certificate is awarded to
NIKHIL KOHLI

for demonstrating competence in the completion of
**FUNDAMENTALS OF DEEP
LEARNING FOR COMPUTER VISION**

Will Ramey
Senior Director, Developer Programs, NVIDIA

2019
Year issued

NVIDIA CERTIFICATION LEARNINGS

NVIDIA CERTIFICATION LEARNINGS

NVIDIA CERTIFICATION LEARNINGS

NVIDIA CERTIFICATION LEARNINGS