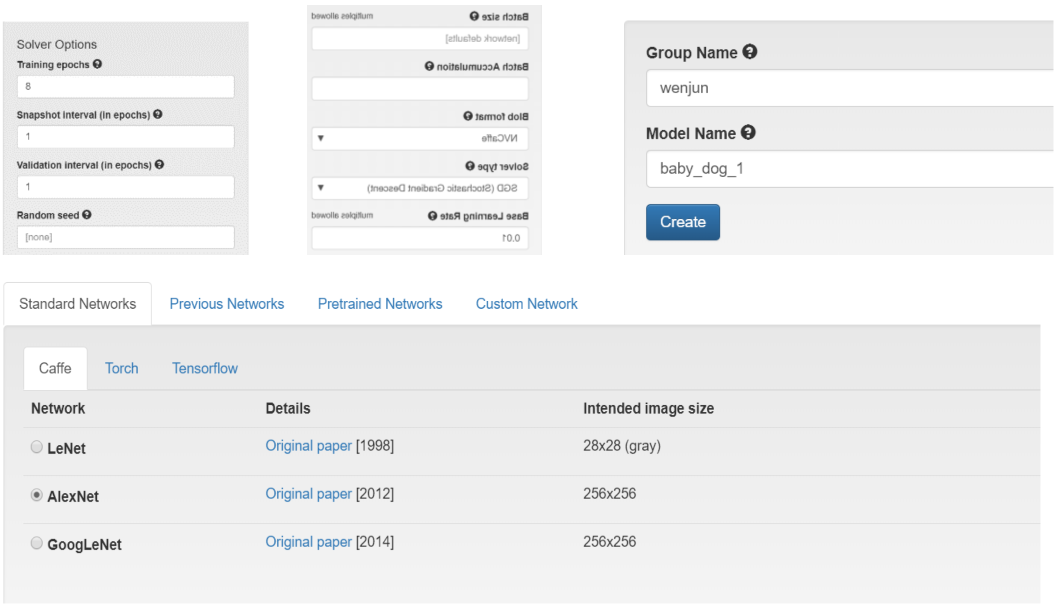
NVIDA DEEP LEARNING COURSE REPORT

1. For the game, I followed the instruction to play a game to train a learning process to identify Louie or not. In this section, I know the learning mechanism for a supervised deep learning process, we should give out the labels to train the model. Setting up a model, and then tell the model what “correct” label for the input data is, and then use it to keep training the model to learn. With more and more labels we tell the model, the model will be more and more smart to identify the targets. So this is the logic direction to train the model to keep learning. All of the things we should do in deep learning are training models with better model and better parameters to get a higher accuracy and output.
2. To teach a neural network to understand the difference between Louie and other dogs, we'll need images for the network to learn from. We've pre-loaded 8 labeled images of Louie and 8 labeled images of other dogs into the dataset called "Images of Beagles." Next I start the parameter of “epoch” being 8, and change it to be 20 and 30 to observe the output of the model. and I keep the other parameters as default values. And then select the deep neural network to train by selecting "AlexNet" from the list of "Standard Networks". AlexNet is intended to be used with 256X256 (color) images, and it is the most popular network for fresh learner as me. So far we get a model to train the data, I name the group as my name “Wenjun”, and the model name as “ baby_dog_1” as the following graph:



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:

Blob format: NVCaffe

Solver type: SGD (Stochastic Gradient Descent)

Base Learning Rate (multiples allowed): 0.01

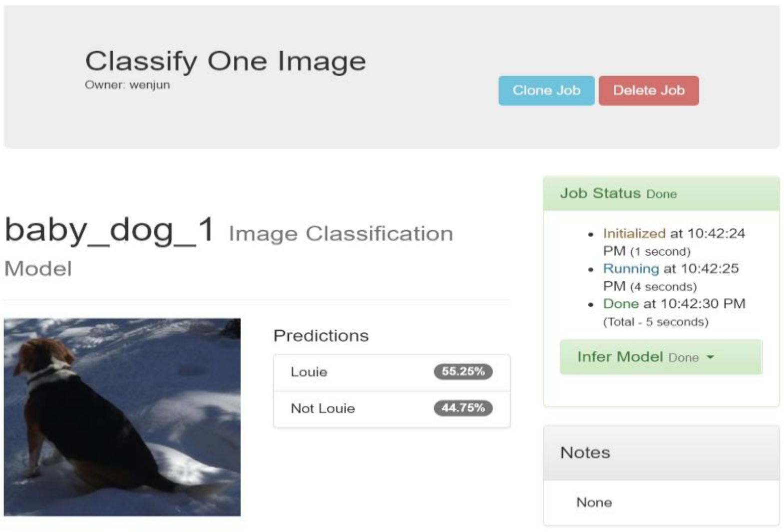
Group Name: wenjun

Model Name: baby_dog_1

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

Figure 1 original parameters

3. After training the model, we should test it, I choose one picture to test the model, and get the result as following:


Classify One Image

Owner: wenjun

Clone Job Delete Job

baby_dog_1 Image Classification Model

Predictions

Louie	55.25%
Not Louie	44.75%

Job Status Done

- Initialized at 10:42:24 PM (1 second)
- Running at 10:42:25 PM (4 seconds)
- Done at 10:42:30 PM (Total - 5 seconds)

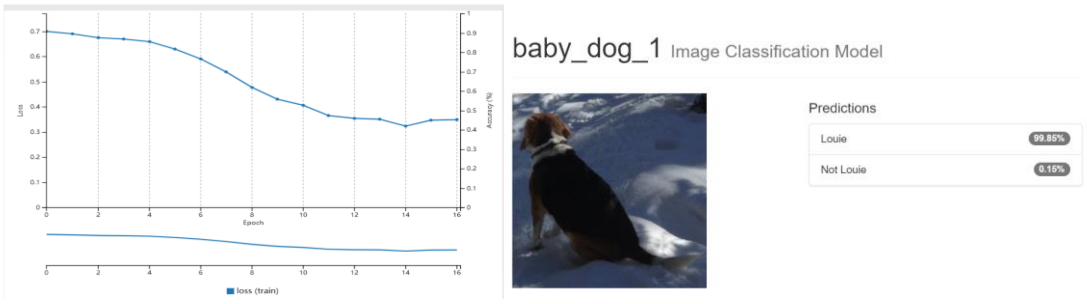
Infer Model Done

Notes

None

Figure 1 epoch=8

The accuracy is around 0.5, because this picture has less information to identify it being Louie. So we should continue to improve the model. I set the epoch=20 to train the model, and test it, then get such a result as following:



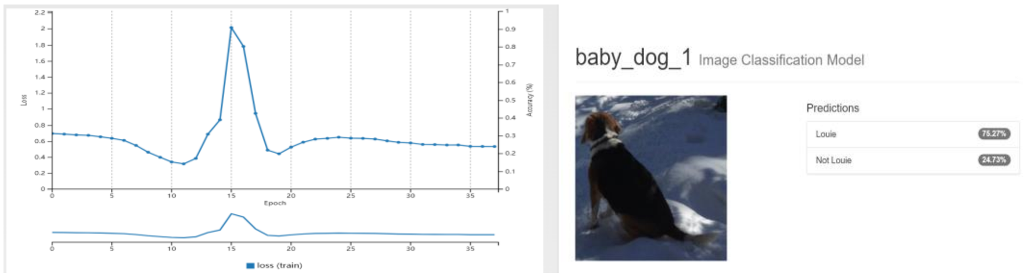
baby_dog_1 Image Classification Model

Predictions

Louie	99.85%
Not Louie	0.15%

Figure 2 epoch=20

So the model with epoch=20 is obviously better than epoch=8 when we input the same image to test the models. So we continue to increase the size of epoch to be 40 to observe the result:



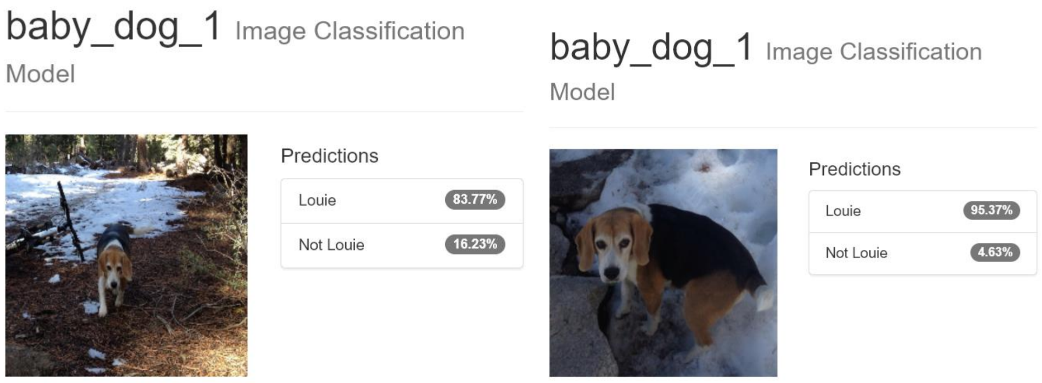
baby_dog_1 Image Classification Model

Predictions

Louie	75.27%
Not Louie	24.73%

Figure 3 epoch=40

The accuracy is lower than epoch=30, and we observe the loss graph, we can see epoch is a cutoff to determine the loss reduction, which means if we increase the size of epoch the model is easy to be overfitting. so we use epoch=20 to test the model with other images which are Louie or not as the following figure 4~6.


baby_dog_1 Image Classification Model

Predictions

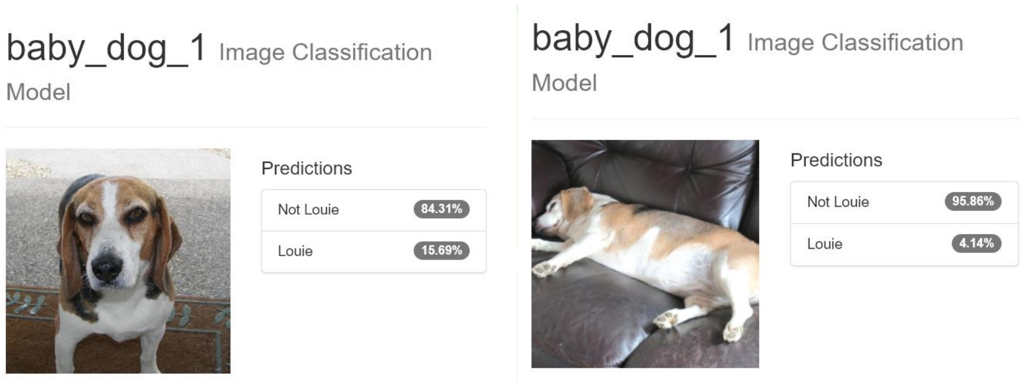
Louie	83.77%
Not Louie	16.23%

baby_dog_1 Image Classification Model

Predictions

Louie	95.37%
Not Louie	4.63%

Figure 4 Real Louie


baby_dog_1 Image Classification Model

Predictions

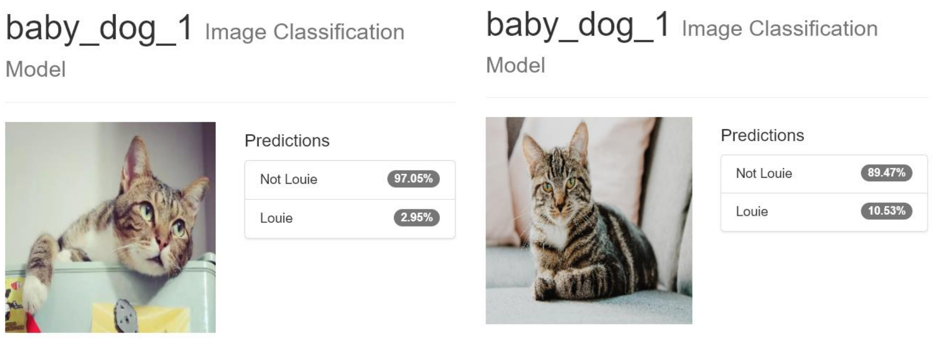
Not Louie	84.31%
Louie	15.69%

baby_dog_1 Image Classification Model

Predictions

Not Louie	95.86%
Louie	4.14%

Figure 5 False Louie


baby_dog_1 Image Classification Model

Predictions

Not Louie	97.05%
Louie	2.95%

baby_dog_1 Image Classification Model

Predictions

Not Louie	89.47%
Louie	10.53%

Figure 6 False Louie

From above pictures, we can see the model with epoch=20 can identify the pictures well. So this is a successful model to classify Louie or not.

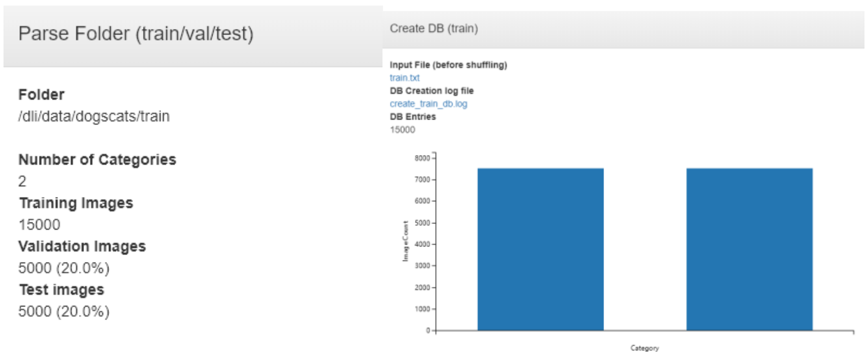
Successfully training a neural network to perform well on new data requires enough data to demonstrate the diversity of the environment where your network should be effective.

Essentially, instead of "learning" the difference between Louie and other dogs, the network has "memorized" which image in the training dataset belongs to which class. So for the previous model which has just one dog to train the model is easily overfitting, and it will be influenced by other factors much easier, like Louie facing a different direction, with a different amount of light, or when he was a different age. So we should optimize the model to adapt the new data and reduce overfitting.

4. Task 2

So we should creating a model that is effective with new data and enlarge the scope of images which are used as input dataset to identify dogs and cats.

I choose another dataset which is from " /dli/data/dogscats/train". And set the % for validation is 25%and 0% for test data. To reduce the time required to prepare the dataset, change "Encoding" from "PNG(lossless)" to "None" and give the dataset a name- dogs and cats.


Parse Folder (train/val/test)

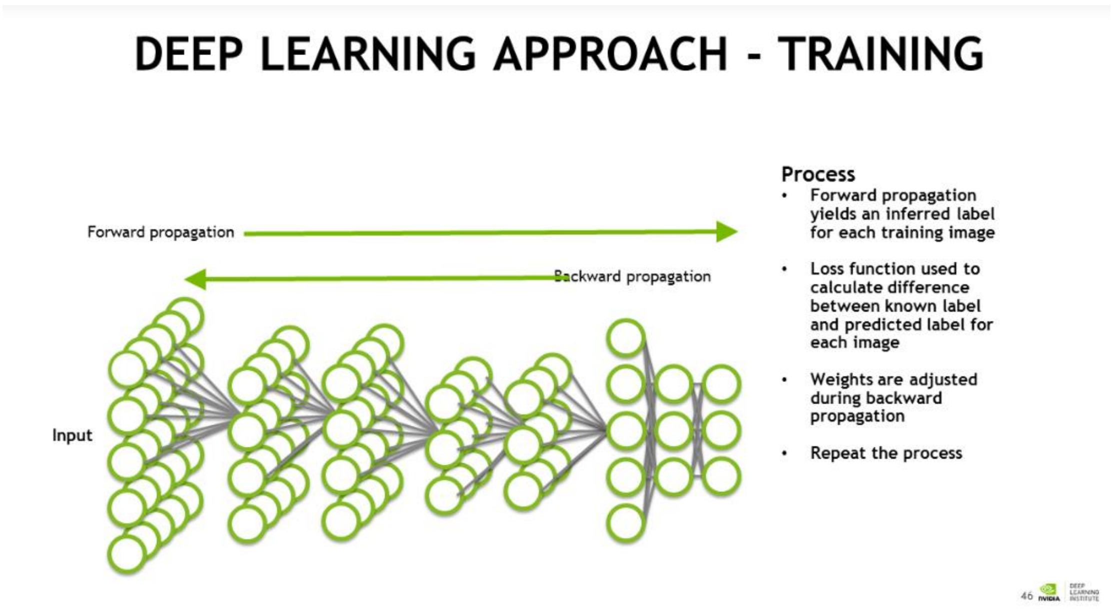
Folder
/dli/data/dogscats/train

Number of Categories
2
Training Images
15000
Validation Images
5000 (20.0%)
Test images
5000 (20.0%)

Create DB (train)

Input File (before shuffling)
train.txt
DB Creation log file
create_train_db.log
DB Entries
15000

Standardize them to the same size to match what the network you are training expects. We'll be training AlexNet again which was designed to take an input of 256X256 color images. Split them into two datasets, where 75% of each class is used for training and 25% is set aside for validation. The structure of algorithm is just like this:

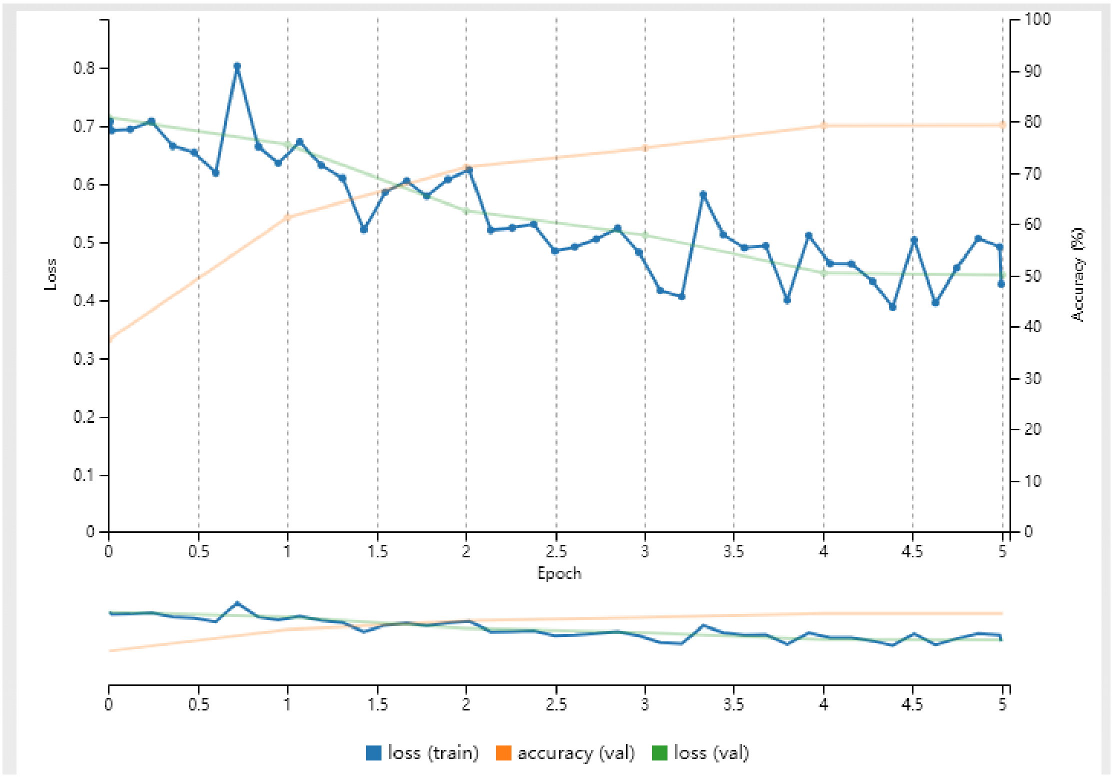

DEEP LEARNING APPROACH - TRAINING

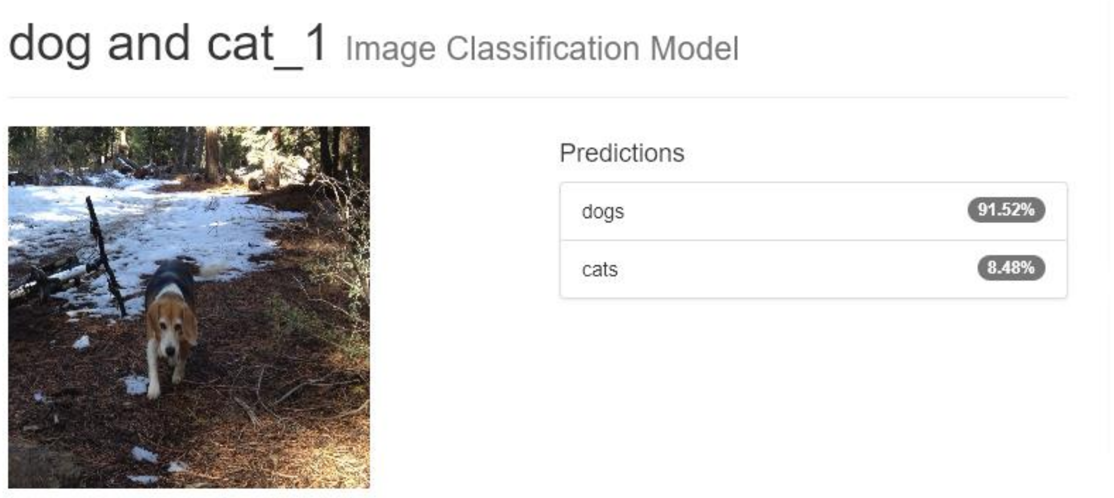
Process

- Forward propagation yields an inferred label for each training image
- Loss function used to calculate difference between known label and predicted label for each image
- Weights are adjusted during backward propagation
- Repeat the process

When we validate the model with validation data, the network *does not learn anything* from the data. Loss is reported but the model itself is unchanged! We can use the same validation dataset to assess our performance on new data over and over again while continuing to treat it like *new* data.

Then we focus on train the model. I set the epoch=5 to obverse the model:





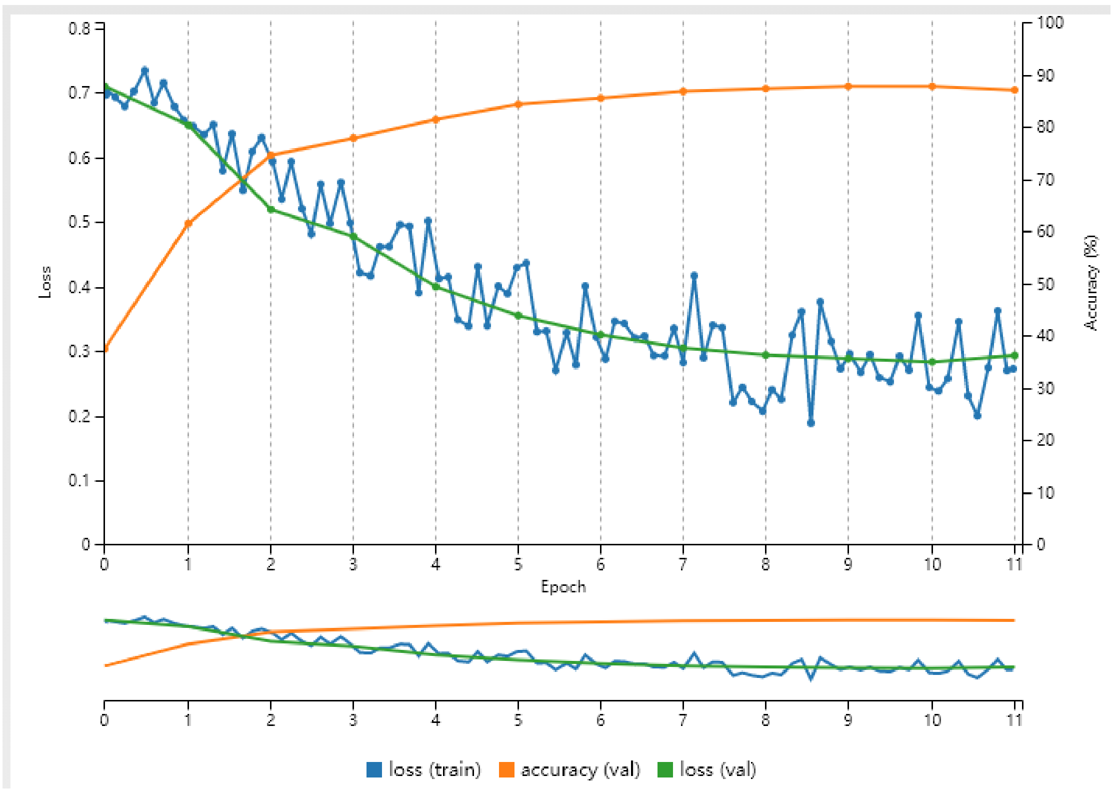
dog and cat_1 Image Classification Model

Predictions

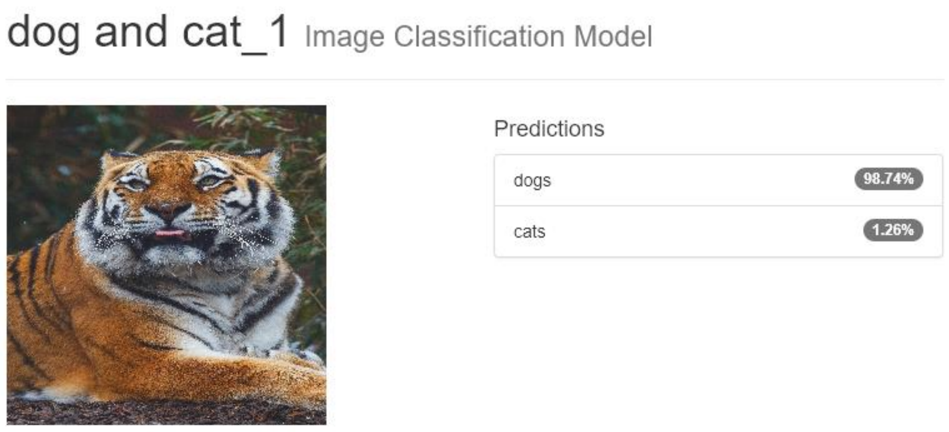
dogs	91.52%
cats	8.48%

I randomly choose a image to classify, the accuracy for dog class is 91.52 which is good enough for this image.

The loss and accuracy of validation seem to be convergent, but we can try higher epoch to see what will happen for accuracy and loss. We set epoch=11, this model performance better and we can draw out a higher accuracy.



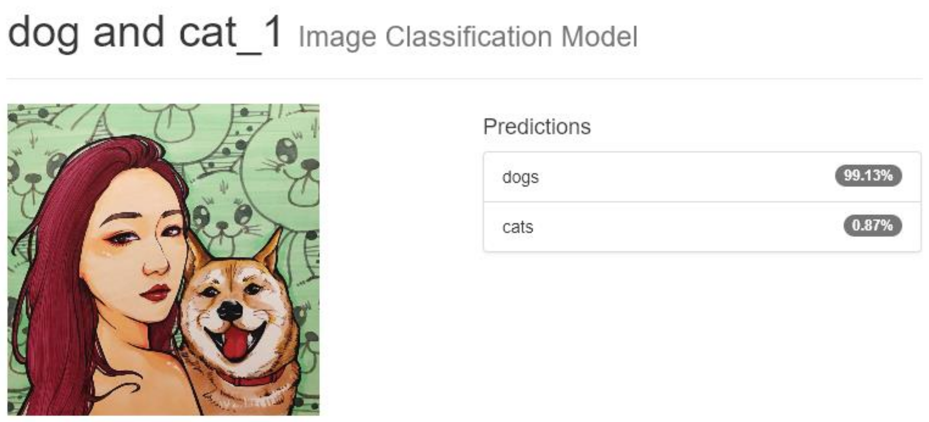
I put two new images (without labels) to test the model, however, the model classify it as a dog with a high accuracy like this:



dog and cat_1 Image Classification Model

Predictions

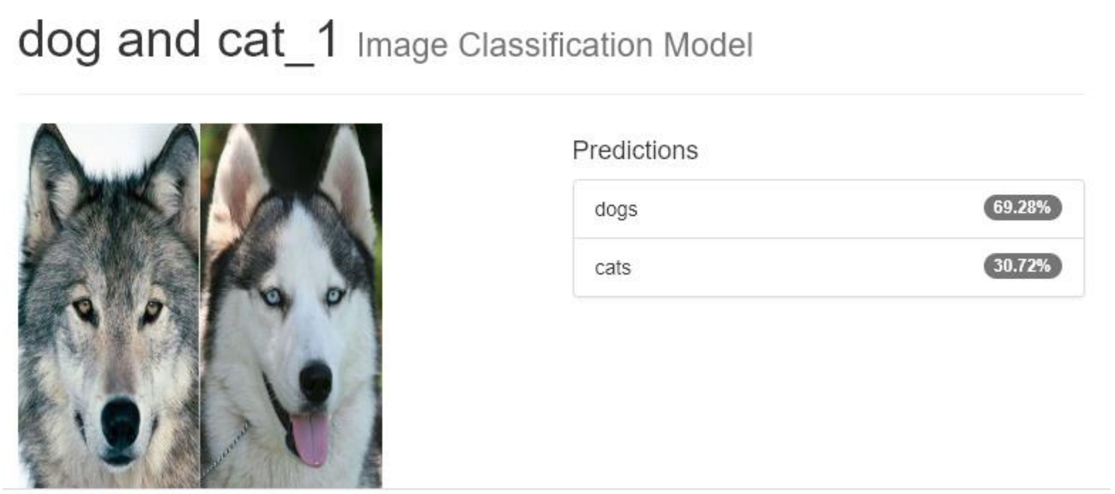
dogs	98.74%
cats	1.26%



dog and cat_1 Image Classification Model

Predictions

dogs	99.13%
cats	0.87%



dog and cat_1 Image Classification Model

Predictions

dogs	69.28%
cats	30.72%

In general, this model is good for the images abviouly like a dog or cat, but it’s hard to identify the complex images. So we should train the model further.

5. GPU task3

In this section, it’s about how Deep Neural Networks, GPUs, and big data have enabled us to solve problems using computing that were previously impossible. For this realistic problem—doggy door, is a interesting case to learn the previous knowledge .

There are three points we should consider when we design the model of doggy door: How to deploy a trained model into an application. What role a trained model plays within an application. To identify and use the pieces of a trained model.

Input the data:

```
MODEL_JOB_DIR = ’/dli/data/digits/20180301-185638-e918’
## Remember to set this to be the job directory for your model
!ls $MODEL_JOB_DIR
```

```
caffe_output.log    snapshot_iter_735.caffemodel    status.pickle
deploy.prototxt     snapshot_iter_735.solverstate   train_val.prototxt
original.prototxt   solver.prototxt
```

Set a classifier:

```
# Initialize the Caffe model using the model trained in DIGITS
net = caffe.Classifier(ARCHITECTURE, WEIGHTS,
                       channel_swap =(2, 1, 0), #Color images have three channels, Red, Green, and Blue.
                       raw_scale=255) #Each pixel value is a number between 0 and 255
                       #Each "channel" of our images are 256 x 256
```

We obtain a new data which should be used to train the model, but before we train a model, we should resized the images.

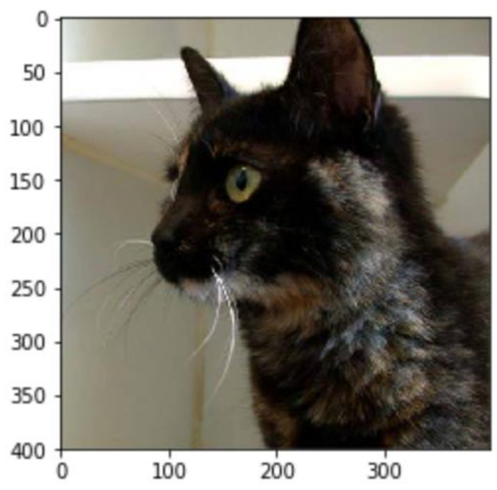
```
mean_image = caffe.io.load_image(DATA_JOB_DIR+’/mean.jpg’)
ready_image = input_image-mean_image
```

after processing data, we can train the model officially:

```
##Create an input our network expects
input_image= caffe.io.load_image('/dli/data/fromnest.PNG')
input_image=cv2.resize(input_image, (256, 256), 0,0)
ready_image = input_image-mean_image
##Treat our network as a function that takes an input and generates an output
prediction = net.predict([ready_image])
print("Input Image:")
plt.imshow(input_image)
plt.show()
print(prediction)
##Create a useful output
print("Output:")
if prediction.argmax()==0:
    print "Sorry cat:( https://media.giphy.com/media/jb8aFEQk3tADS/giphy.gif"
else:
    print "Welcome dog! https://www.flickr.com/photos/aidras/5379402670"
```

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 we put into a new image to test the model like this:

```
TEST_IMAGE = '/dli/data/dogscats/test/5.jpg'
display= caffe.io.load_image(TEST_IMAGE)
plt.imshow(display)
plt.show()
```



```
!python pythondeployment.py $TEST_IMAGE 2>/dev/null
```

```
[[ 0.25521094  0.74478912]]
Output:
Welcome dog! https://www.flickr.com/photos/aidras/5379402670
None
```

That means this model is not good enouth to identify some confusing images which are difficult to identify by people sometimes like the one above.

6. Task 6

Make a conclusion for the knowledge learned in this course:

1 Prepared a dataset for training

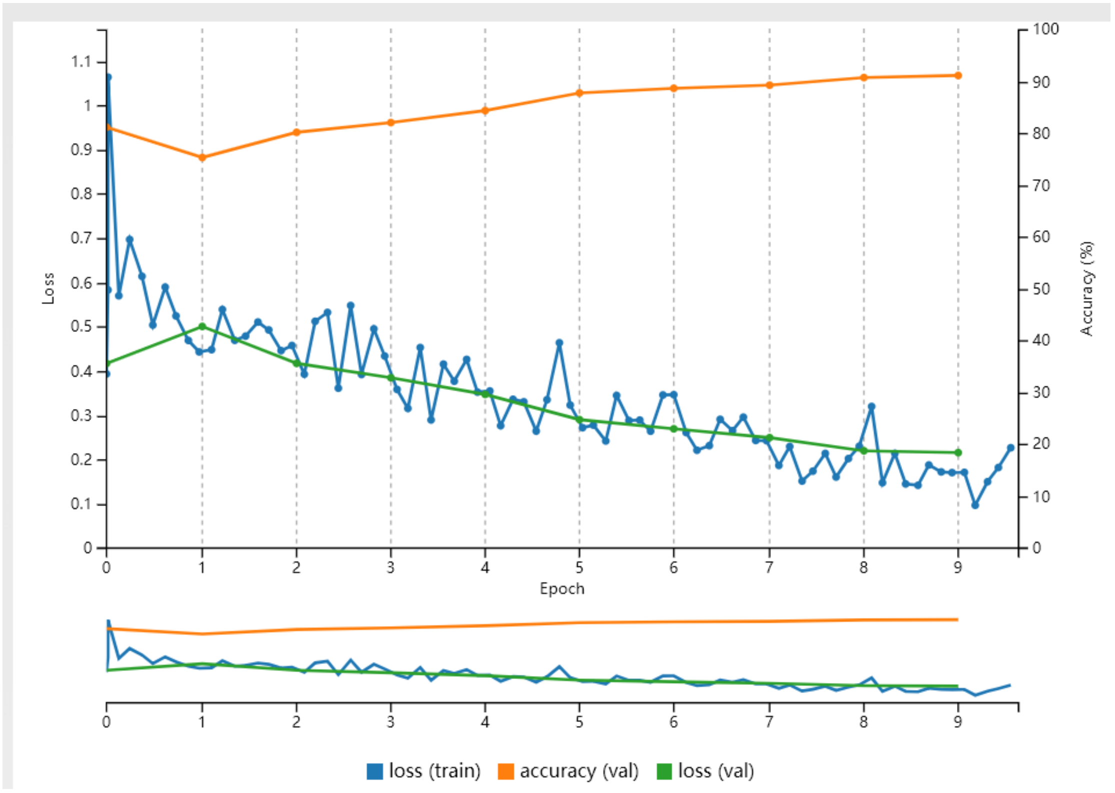
2 Selected a network to train

3 Trained the network

4 Tested the trained model in a training environment

5 Deployed the trained model into an application

I improve the training performance by: run more training epochs on an existing model, search the hyperparameter space to get different parameters to train different models, use the results of others' research, compute, network design, and data to optimize the models. The most popular parameters we should adjust are “epoch” and “learning rate”.



Additionally, we can get some information to improve the models from this net course from the following four aspects：

1) Data - A large and diverse enough dataset to represent the environment where our model should work. Data curation is an art form in itself.

2) Hyperparameters - Making changes to options like learning rate are like changing your training "style." Currently, finding the right hyperparameters is a manual process learned through experimentation. As you build intuition about what types of jobs respond well to what hyperparameters, your performance will increase.

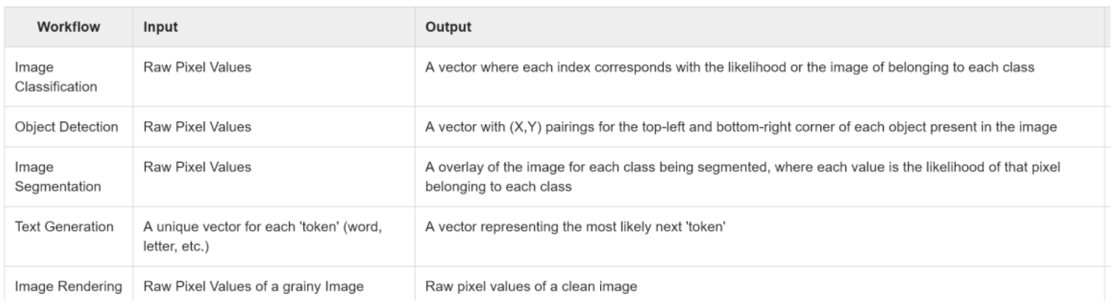
3) Training time - More epochs improve performance to a point. At some point, too much training will result in overfitting (humans are guilty of this too), so this can not be the only intervention you apply.

4) Network architecture - We'll begin to experiment with network architecture in the next section. This is listed as the last intervention to push back against a false myth that to engage in solving problems with deep learning, people need mastery of network

architecture. This field is fascinating and powerful, and improving your skills is a study in math. Not only can we use their network architecture, we can even use their trained weights, acquired through the manipulation of the four levers above: data, hyperparameters, training time, and network architecture. Without any training or data collection, we can *deploy* award winning neural networks. So we can use the excellence models from others to deal with different problems.
And then I learned a tools to draw data from web directly to the server without pulling it to local: !wget http://dl.caffe.berkeleyvision.org/bvlc_alexnet.caffemodel
!wget:https://raw.githubusercontent.com/BVLC/caffe/master/models/bvlc_alexnet/deploy.prototxt

7. task 5

we should prescribe an input/output mapping for object 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. Below is a chart that shows some common input-output mappings.



Workflow	Input	Output
Image Classification	Raw Pixel Values	A vector where each index corresponds with the likelihood or the image of 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

this is very important for the future images research work, we should know what we should do before we do something. A model that specified the location of an object (localization) would aim to output locations as close to where the actual object was as possible. So we should learn how to detect object if we have to handle some raw images without clear object or features or labels.
The basic principle of object detection is called "**sliding window**" approach, where we'll take split out image into small sections which we'll call grid squares. We're run 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. So, in this way, we can draw Louie out from the image, so does the other object.
So object detection is integration of traditional programming and deep learning. we should combine deep learning with traditional computer vision and modify the internals (the actual math) of neural networks and choose the right network for the job.
We use the model and dataset job directories to make use of the model architecture, trained weights, and preprocessing information such as the mean image.
We use "net.predict" to draw out the probability through performing layer after layer of matrix math to transform an image into a vector of probabilities. The prediction from the top left 256X256 is:

```
X = 0
Y = 0

grid_square = input_image[X*256:(X+1)*256,Y*256:(Y+1)*256]
# subtract the mean image (because we subtracted it while
#we trained the network - more on this in next lab)
grid_square -= mean_image
# make prediction
prediction = net.predict([grid_square])
print prediction
```

```
[[ 0.89857852  0.10142148]]
```

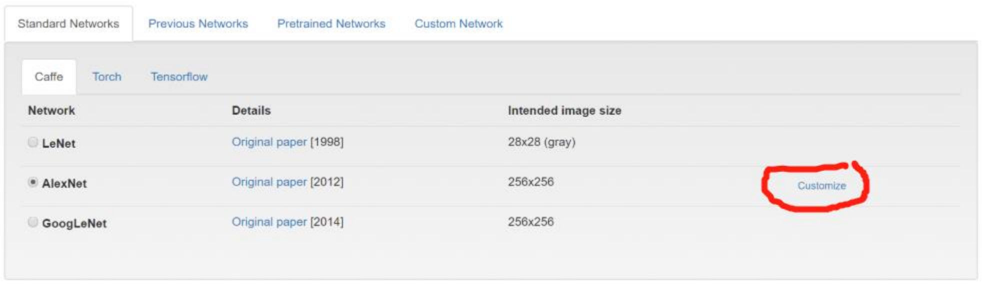
We implement some code around our function to iterate over the image and classify each grid_square to create a heatmap:

```
# Load the input image into a numpy array and display it
input_image = caffe.io.load_image(IMAGE_FILE)
plt.imshow(input_image)
plt.show()

# Calculate how many 256x256 grid squares are in the image
rows = input_image.shape[0]/256
cols = input_image.shape[1]/256

# Initialize an empty array for the detections
detections = np.zeros((rows,cols))

# Iterate over each grid square using the model to make a class prediction
start = time.time()
for i in range(0,rows):
    for j in range(0,cols):
        grid_square = input_image[i*256:(i+1)*256,j*256:(j+1)*256]
        # subtract the mean image
        grid_square -= mean_image
        # make prediction
        prediction = net.predict([grid_square])
        detections[i,j] = prediction[0].argmax()
end = time.time()
```

This is the output of a randomly chosen wide area test image (larger than 256X256) along with an array showing the predicted class for each non-overlapping 256x256 grid square when input in to the model.

Next, we keep the grid square size as 256x256, modify the code to increase the overlap between grid squares and obtain a finer classification map. Modify the code to batch together multiple grid squares to pass in to the network for prediction. 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.

Nest important step is to replace some layers of the AlexNet network to create new functionality AND make sure that the network still trains, which is the right difference between traditional deep learning and object detection process.


Standard Networks | Previous Networks | Pretrained Networks | Custom Network

Caffe | Torch | Tensorflow

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

We can change the value of the “prototext” files to change the structure of the model, this is the first time I know we can change this file to adjust and obverse the model. and we can get a clear structure and the relationship of different layers of other parameters of the model. On one hand, we can use ANY math we want, as long as we keep everything connected so the data can flow from input to output.

We removed a "fully connected" layer, which is a traditional matrix multiplication. While there is a lot to know about matrix multiplication, the one characteristic that we'll address is that matrix multiplication only works with specific matrix sizes. The implications of this for our problem is that we are limited to fixed size images as our input (in our case 256X256).

We added a "convolutional" layer, which is a "filter" function that moves over an input matrix. There is also a lot to know about convolutions, but the one characteristic that we'll take advantage of is that they **do not** only work with specific matrix sizes. The implications of this for our problem are that if we replace all of our fully connected layers with convolutional layers, we can accept any size image.

By converting AlexNet to a "Fully Convolutional Network," we'll be able to feed images of various sizes to our network without first splitting them into grid squares.

Feel free to test other images from the `/dli/data/BeagleImages` folder. You will see that DetectNet is able to accurately detect most dogs with a tightly drawn bounding box and has a very low false alarm rate. It is also worth noting that this model can detect multiple classes of objects! So, we can extend the utilization of this method to other area, such as auto-pilot driving, smart-transportation and so on.

From this course, I learned how to train multiple types of neural networks using DIGITS

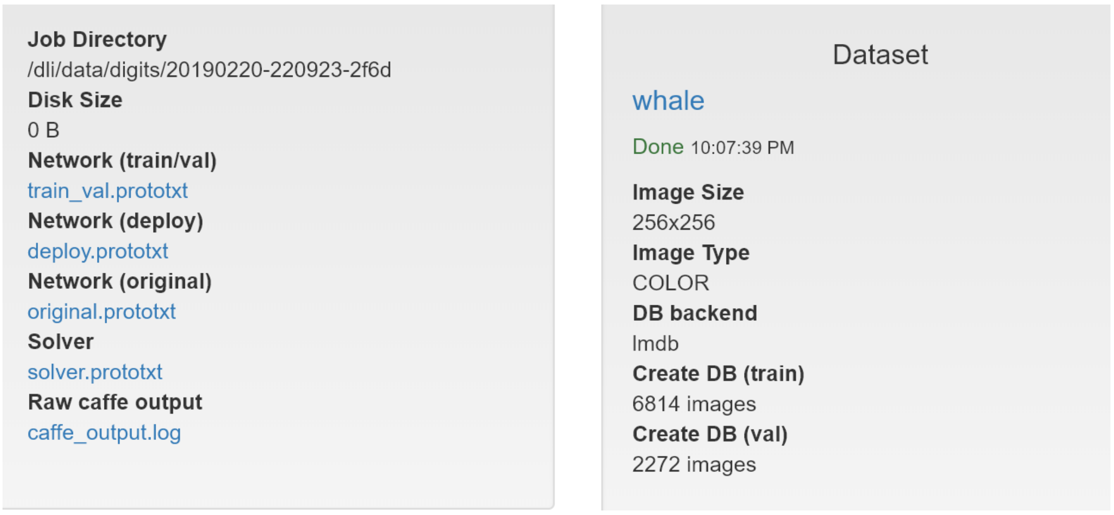
What types of changes you can make to how you train networks to improve performance

How to combine deep learning with traditional programming to solve new problems

How to modify the internal layers of a network to change functionality and limitations

Identify the ingredients required for deep learning, train a Deep Neural Network to correctly classify images it has never seen before, deploy Deep Neural Networks into applications, identify techniques for improving the performance of deep learning applications, assess the types of problems that are candidates for deep learning, modify neural networks to change behavior.

At last, I did a whale classification case, upload the dataset, I set the model with: epochs=5, batch_size=32, blob format=NVCaffe, Solver type = SGD, Base learning rate=0.01, and used AlexNet as the network.



Job Directory
/dli/data/digits/20190220-220923-2f6d
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

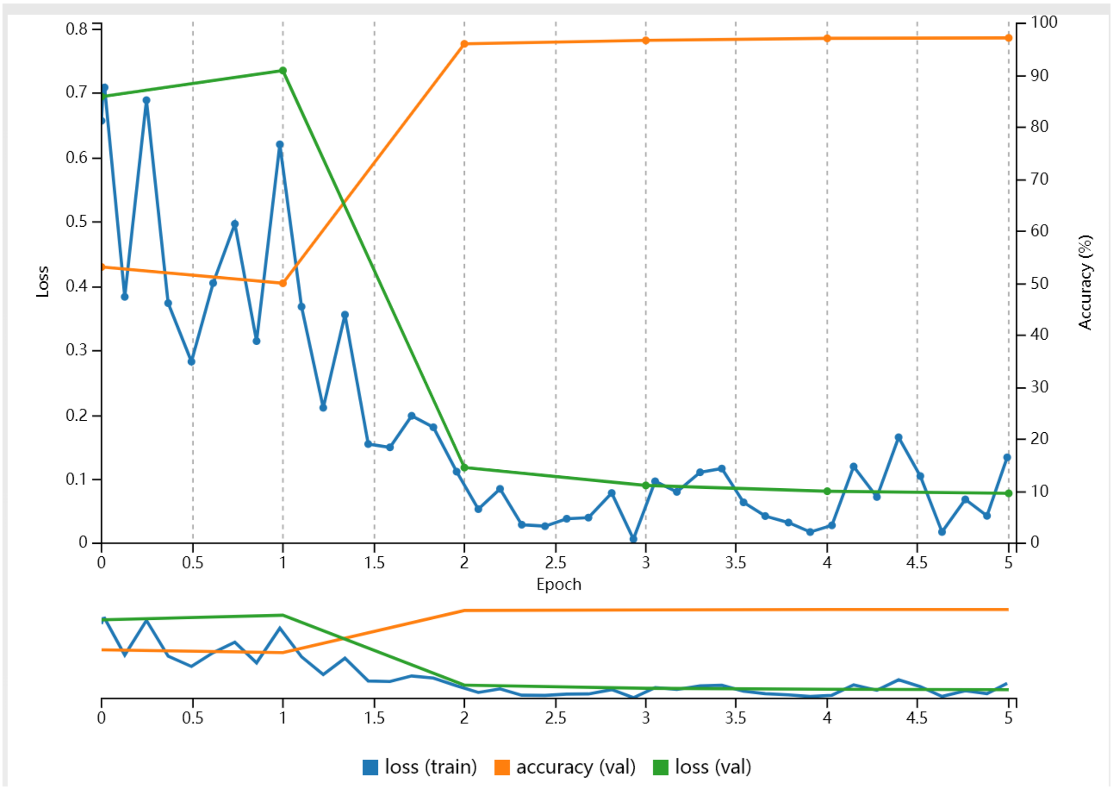
Dataset

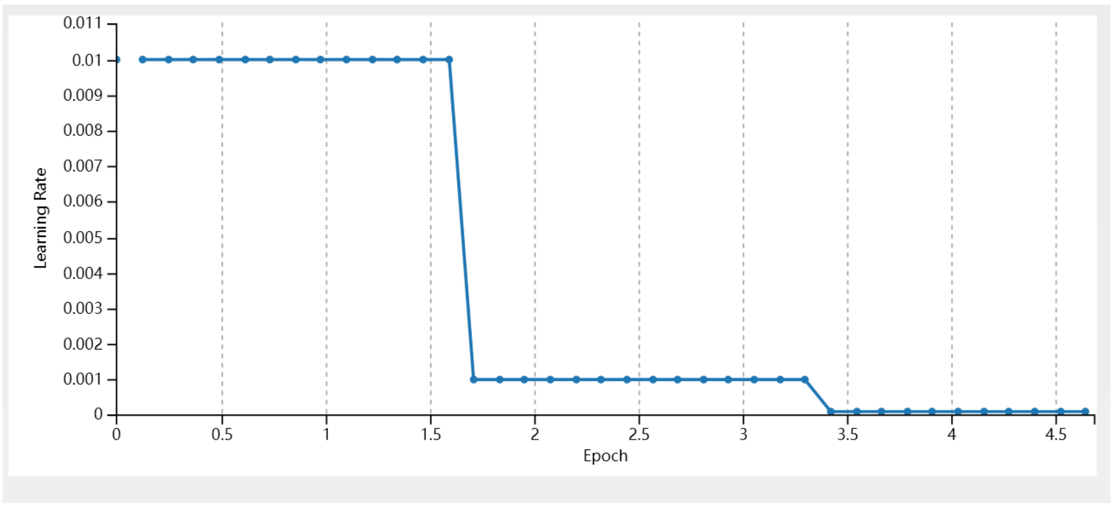
whale

Done 10:07:39 PM

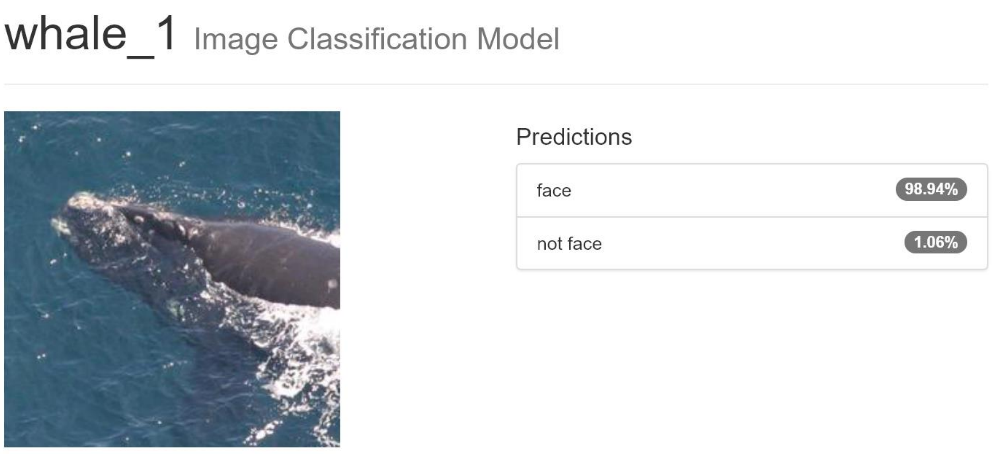
Image Size
256x256
Image Type
COLOR
DB backend
lmdb
Create DB (train)
6814 images
Create DB (val)
2272 images

The model with epochs=5 is as following :





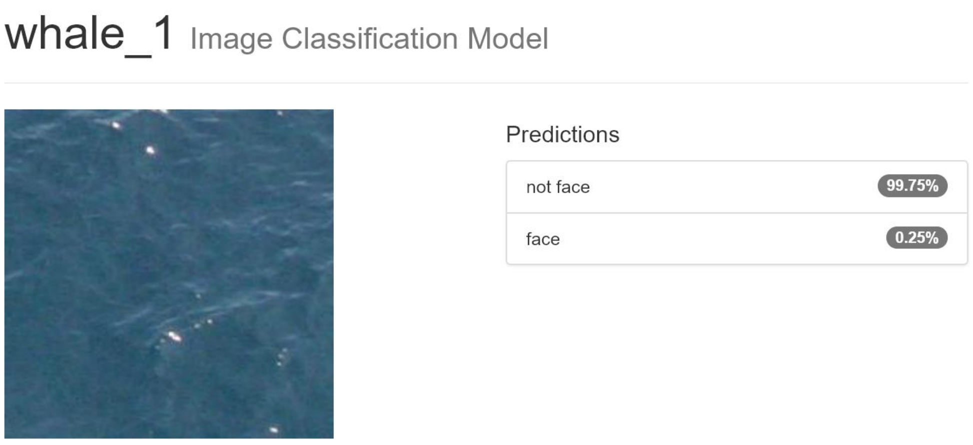
from the values of traing loss and validation loss and accuracy of validation data(orange line), we can arbitrarily to say this model is good for the dataset, but we should also test it by some whale images with or without whale faces as following:



whale_1 Image Classification Model

Predictions

face	98.94%
not face	1.06%

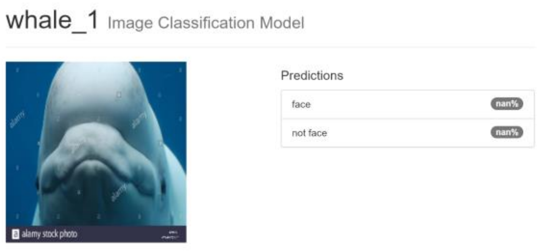


whale_1 Image Classification Model

Predictions

not face	99.75%
face	0.25%

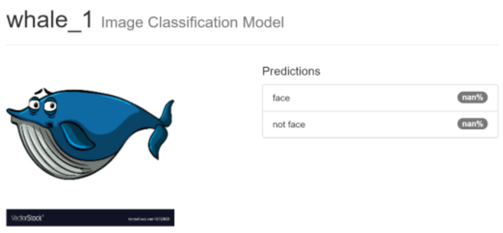
After test the model with two pictures, the model is not bad for classify the whale face, but we should test it further.


whale_1 Image Classification Model

Predictions

face nan%

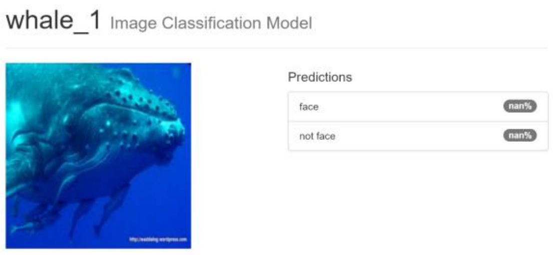
not face nan%


whale_1 Image Classification Model

Predictions

face nan%

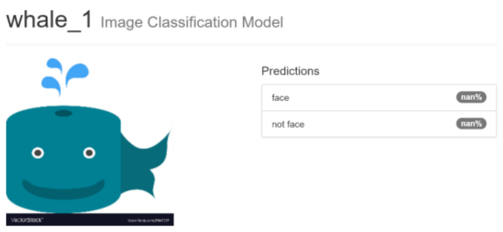
not face nan%


whale_1 Image Classification Model

Predictions

face nan%

not face nan%


whale_1 Image Classification Model

Predictions

face nan%

not face nan%

I used some whale images without labels to test the mode, it can not identify the images as whale or not, so we should continue to improve the model. firstly we shuold enlarge the dataset to learn more labels with features to help the potential model learn more information about whale, and than to set out the proper parameters.