NVIDIA DEEP LEARNING

GPU TASK 1 :

- This task explains the difference between Machine Learning and Deep Neural Networks.
- It explains the functioning of deep neural networks in the simplest form with Louie classifier example.
- It explains various applications of deep neural networks in real world problems.
- Digits platform was introduced in this task to build a model which classifies the images whether it is Louie or not.
-

No	Model	No of epochs	Result(Classifying as Louie)%
1	Alexnet	3	48.4
2	Lenet	3	100
3	GoogleNet	3	63.4
4	Lenet	50	100
5	Alexnet	50	55.6
6	GoogleNet	50	96.49

- Lenet gave the maximum accuracy but it works only with grey scale images.
- With an increase in no of epochs accuracy increased but when the model was tested with other images the model failed to classify them correctly this was due to overfitting of the model as the data is small.

GPU TASK 2:

- Task 2 mainly deals with overfitting issues by training the model with large data sets.
- Task 2 introduces us to data set creation in Digits platform and explains how to use the data sets created in training a model.
- The data was used and models were built the results are as follows

Model	No of epocs	Train loss	Val loss	Val accuracy	Result(classifying Louie)%
Alexnet	5	0.5	0.45	80	88.2
Google Lenet	5	0.8	0.9	70	69

- Alexnet is hence proved to be the best model for classifying the images.

NVIDIA DEEP LEARNING

GPU TASK3:

- Task 3 mainly deals with deploying the model useful for the end user.
- Explains how to give inputs to the network, The way it expects.
- Converting the output which the model gives into a format which the user can understand.
- To read and process the saved models we need an architecture which can understand and use these models built. So Caffe architecture was introduced in this task.
- We need to provide the path of the model to access its file to the caffee architecture.

```
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
```

- Weights of the model and architecture of the model are needed to use these, so we store them using caffe architecture in different variables

```
ARCHITECTURE = MODEL_JOB_DIR + '/' + 'deploy.prototxt'
WEIGHTS = MODEL_JOB_DIR + '/' + 'snapshot_iter_735.caffemodel'
print ("Filepath to Architecture = " + ARCHITECTURE)
print("Filepath to weights = "+ WEIGHTS)
```

```
Filepath to Architecture = /dli/data/digits/20180301-185638-e918/deploy.prototxt
Filepath to weights = /dli/data/digits/20180301-185638-e918/snapshot_iter_735.caffemodel
```

- We set the mode to GPU as our model consists of hundreds of thousands of operations that can be largely accelerated through parallelization.
- To access the training and testing data we store the dataset job directory in a variable

```
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
```

- Our next step is to remove the mean image from the test image in order to normalize it which in turn reduces the computation time necessary to train.

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

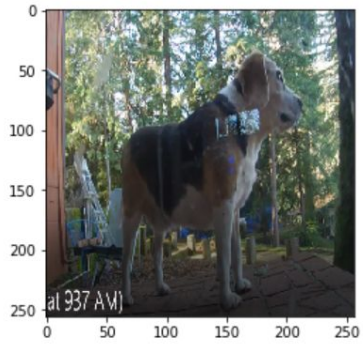
- Now we need to change the output of the model into a format which a user can understand so we process the output of the model

```
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"
```

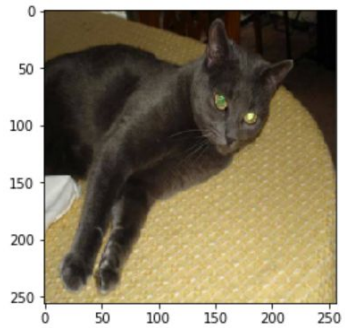
NVIDIA DEEP LEARNING

- Outputs for the test image are as follows.

```
Input Image:
```




```
[[ 0.39924237  0.6007576 ]]
Output:
Welcome dog! https://www.flickr.com/photos/aidras/5379402670

Input image:
```

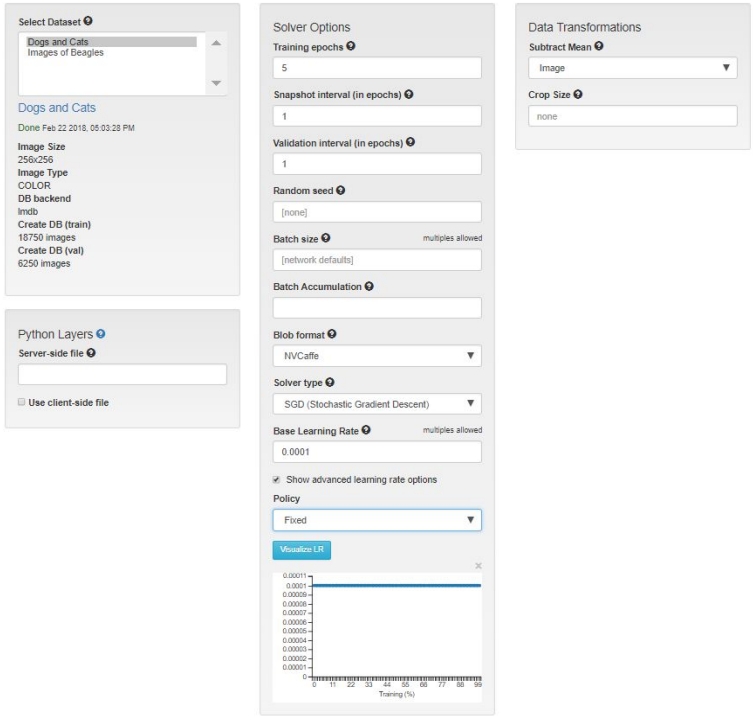



```
Output:
Sorry cat:( https://media.giphy.com/media/jb8aFEQk3tADS/giphy.gif
```

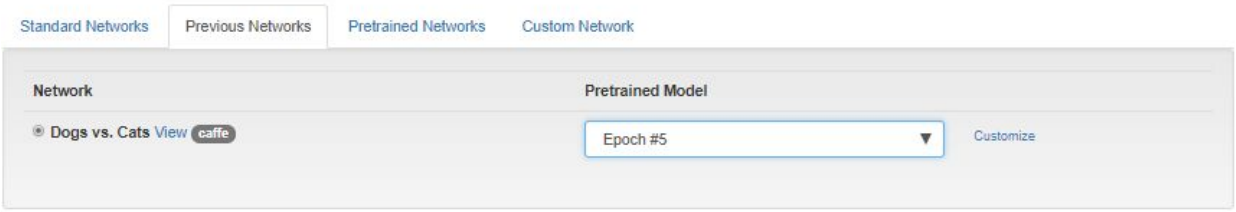
NVIDIA DEEP LEARNING

GPU TASK 4:

- Task 4 mainly deals with improving the performance of the model obtained from task2
- Performance is increased by increasing the number of epochs on the previously trained model and decreasing the learning rate.

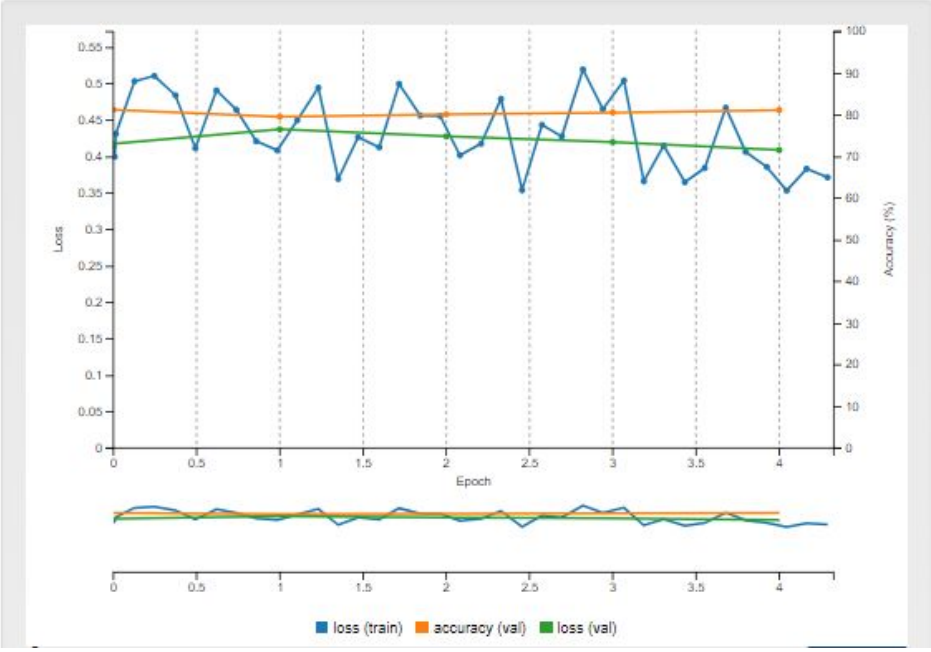


- Using the pre-trained model instead of alexnet or any other networks



- The accuracy of the model is as follows

NVIDIA DEEP LEARNING

- 

- As we have seen that using the pretrained models can improve the accuracy, we now use the award-winning models to improve the accuracy of the models.
- Use of pretrained models and its weights are covered in gpu task 3
- The data to train the model is set and the images are preprocessed to convert them into a shape in which the network expects its input.

```
#Load the image
image= caffe.io.load_image(TEST_IMAGE)
plt.imshow(image)
plt.show()

#Load the mean image
mean_image = np.load(MEAN_IMAGE)
mu = mean_image.mean(1).mean(1)  # average over pixels to obtain the mean (BGR) pixel values

# create transformer for the input called 'data'
transformer = caffe.io.Transformer({'data': net.blobs['data'].data.shape})
transformer.set_transpose('data', (2,0,1))  # move image channels to outermost dimension
transformer.set_mean('data', mu)            # subtract the dataset-mean value in each channel
transformer.set_raw_scale('data', 255)      # rescale from [0, 1] to [0, 255]
transformer.set_channel_swap('data', (2,1,0))  # swap channels from RGB to BGR
# set the size of the input (we can skip this if we're happy with the default; we can also change it later, e.g., for differ
net.blobs['data'].reshape(1,        # batch size
                          3,         # 3-channel (BGR) images
                          227, 227)  # image size is 227x227

transformed_image = transformer.preprocess('data', image)
```

- As we have used the model which was trained on imagenet it consists of 1000 different classes of images so the out put of the model will be the probablility of the image belonging to those 1000 classes separately, We choose the highest probability and classify the image.
- The Implementation of the above process is as follows

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

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

output
```

```
{'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,
          3.18406670e-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,
          3.30536420e-09,   1.88455682e-10,   2.42557197e-10,
          3.91544575e-08,   9.13127296e-10,   7.18308080e-10,
          3.91063715e-09,   1.42117196e-09,   2.83355472e-09,
          8.94371688e-10,   2.96192459e-10,   3.58631702e-09,
          1.15298526e-09,   9.824242402e-08,   9.566139888e-09,
```

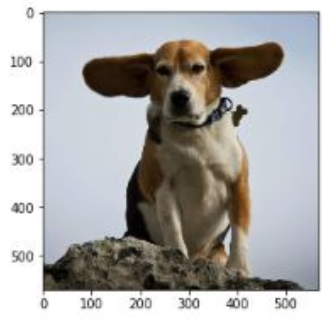
```
!wget https://raw.githubusercontent.com/HoldenCaulfieldRye/caffe/master/data/ilsvrc12/synset_words.txt
labels_file = 'synset_words.txt'
labels = np.loadtxt(labels_file, str, delimiter='\t')

print 'output label:', labels[output_prob.argmax()]
```

```
--2019-02-25 06:34:16--  https://raw.githubusercontent.com/HoldenCaulfieldRye/caffe/master/data/ilsvrc12/synset_words.txt
Resolving raw.githubusercontent.com (raw.githubusercontent.com)... 151.101.200.133
Connecting to raw.githubusercontent.com (raw.githubusercontent.com)|151.101.200.133|:443... connected.
HTTP request sent, awaiting response... 200 OK
Length: 31675 (31K) [text/plain]
Saving to: ‘synset_words.txt’

synset_words.txt    100%[===================>]  30.93K  --.-KB/s    in 0.01s

2019-02-25 06:34:16 (2.86 MB/s) - ‘synset_words.txt’ saved [31675/31675]

output label: n02088364 beagle
```

To get a clean view of what our application does, here is the input and output of our application.

```
print ("Input image:")
plt.imshow(image)
plt.show()

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

```
Input image:
```



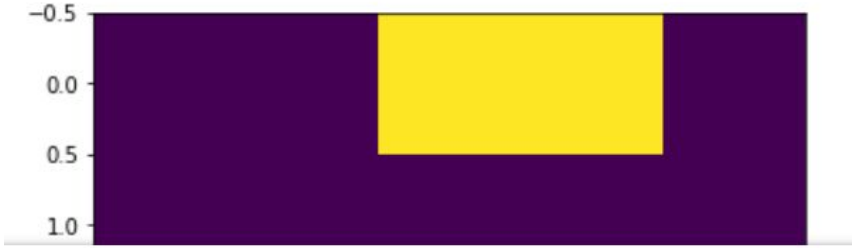

```
Output label:n02088364 beagle
```

NVIDIA DEEP LEARNING

GPU TASK 5:

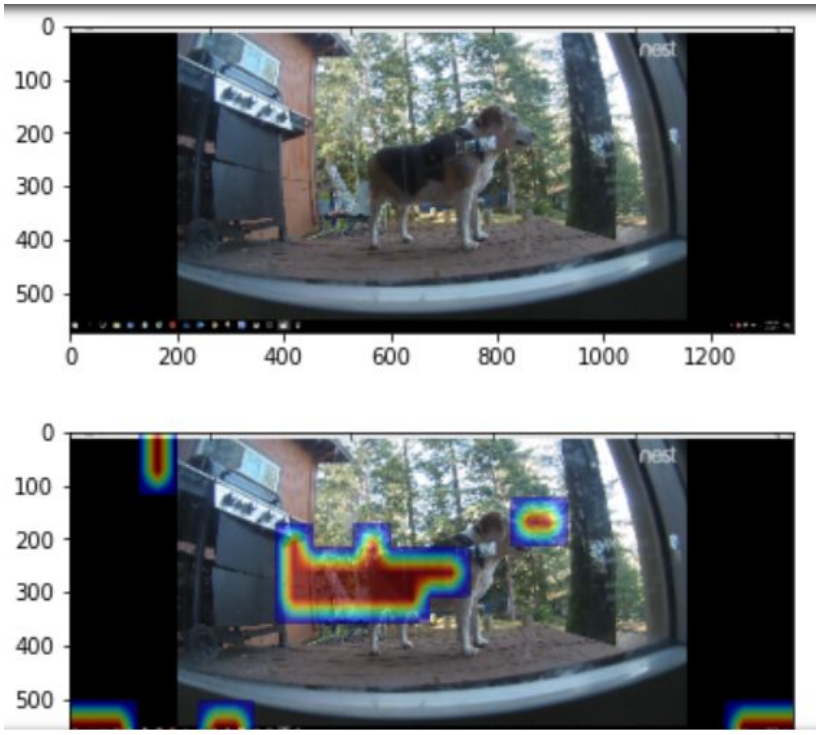
- This section will introduce a more diverse spectrum of computer vision and deep learning workflows.
- Sliding window approach is used to classify images.
- Using the sliding window approach involves deploying an image classifier trained to classify 256X256 portions of an image as either "dog" or "cat" to an application that moves across an image one 256X256 section at a time.
- This approach was built and is applied on a random image, the result is as follows


```
Total inference time: 0.768598079681 seconds
```



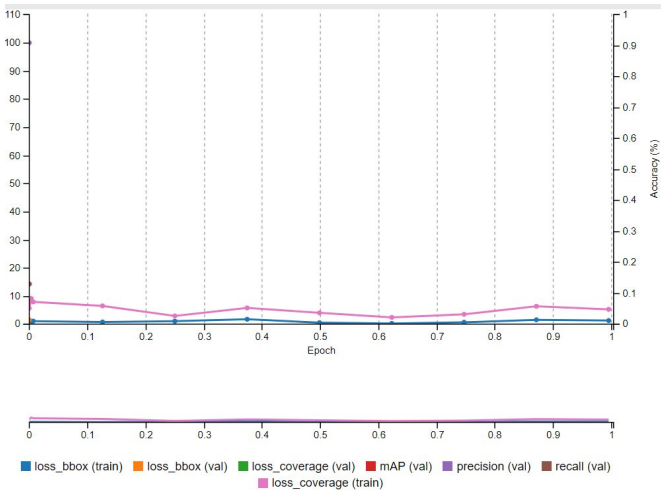
- The model is applied on each grid of the image and the ones which have the highest probability of being a dog is highlighted separating it from the ones which doesn't classify as a dog.
- In the next section, we rebuild from an existing neural network.
- Pretrained model dogsvscats in the gpu task2 is modified by changing the model architecture of Alexnet.
- First, we changed the 6th layer of the network to be convolutional and then hooked up other layers with it to make sure that data flows properly in the architecture.
- Next, we changed the 7th and 8th layers to convolutional layers and thus made a fully convoluted neural network.
- This model is used to classify a image, the results are as follows

NVIDIA DEEP LEARNING



- Time taken to compute it was 0.13 seconds and the classification looks much better than the previous network.
- In the next approach, we detect the images separately in the image highlighting them with a box around them.
- We created a different dataset for this job and named it coco-dog as we did in gpu task2.
- Next, we built a custom model called Densenet which has more convoluted layers in its architecture specially designed for this dataset.
- DetectNet is actually a Fully Convolutional Network (FCN), as we built above, but configured to produce precisely this data representation as its output. The bulk of the layers in DetectNet are identical to the well-known GoogLeNet network.
- The weights of the network are also used with the path provided.
- Training of the model was not performed without weights as it is computationally too expensive and requires a lot of time to build the model from scratch with out weights.
- The model was built and the accuracy is as follows.

NVIDIA DEEP LEARNING



- The model is tested with different images and the result is as follows,