# Assignment 1 Part A NVIDIA Object Detection

## GPU Task #1

**Aim**
In this task, we are classifying images of dogs. Our groups' labels are "Louie" and "Not Louie". Louie being a specific dog and Not Louie being any other dog. For this purpose, we are using the popular AlexNet neural network.

**What is AlexNet?**
AlexNet is the name of a convolutional neural network, designed by Alex Krizhevsky. AlexNet competed in the ImageNet Large Scale Visual Recognition Challenge and achieved a top-5 error of 15.3%, more than 10.8 percentage points lower than that of the runner up. AlexNet revolutionized object detection at it's time.

- AlexNet has 5 convolutional layers,
- 3 fully connected layers
- and 1000-way softmax output layer.
- It takes 256x256 color images as input.
- AlexNet uses ReLu activation function.

**AlexNet Architecture**
- Layer 1 is a Convolution Layer
- Layer 2 is a Max Pooling Followed by Convolution
- Layers 3, 4 & 5 are similar to Layer 2
- Layer 6 is fully connected
- Layers 7 & 8 follow on similar lines.

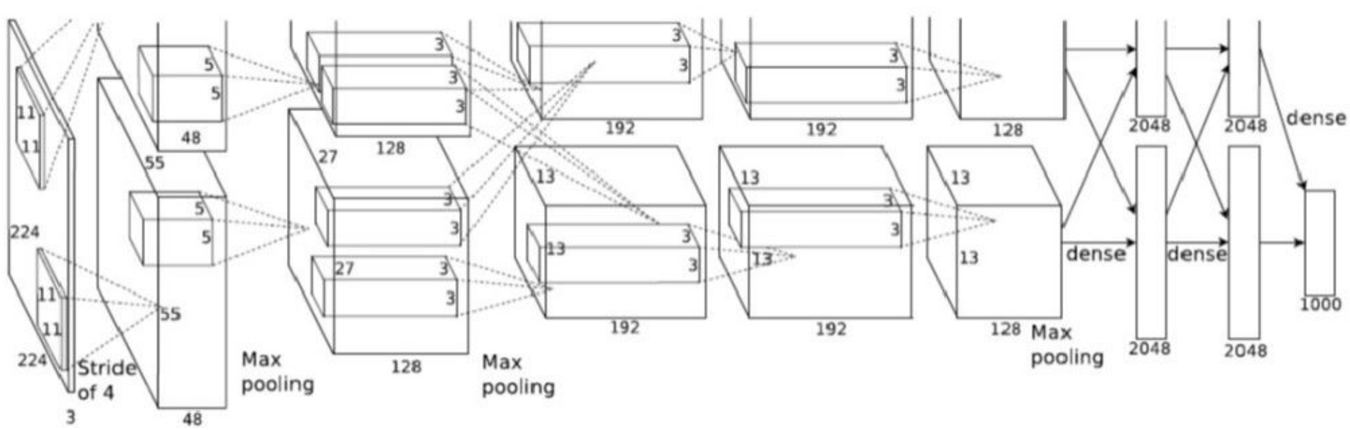


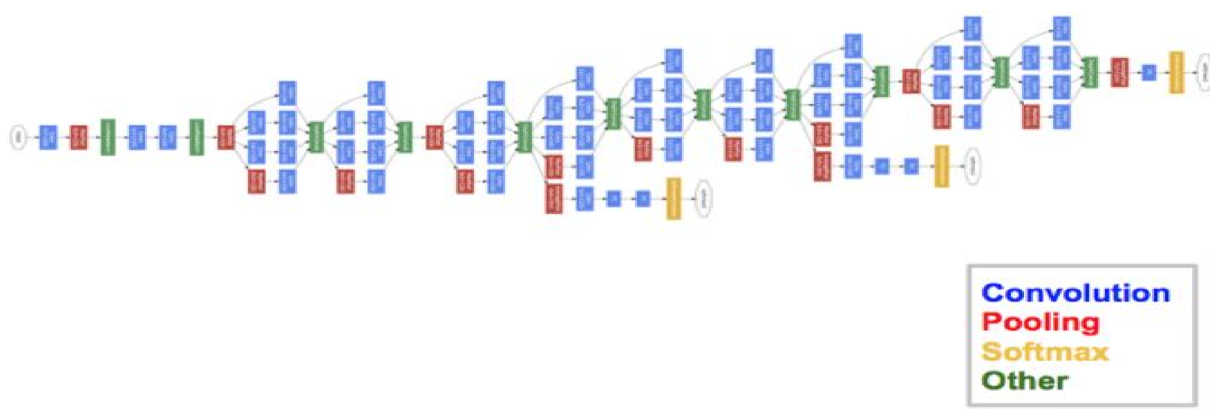

*Diagram obtained from original paper*

**Approach**
We will train the AlexNet neural network with our ‘images_of_beagles’ dataset and we will tune the parameters to obtain the best performance and document our results.

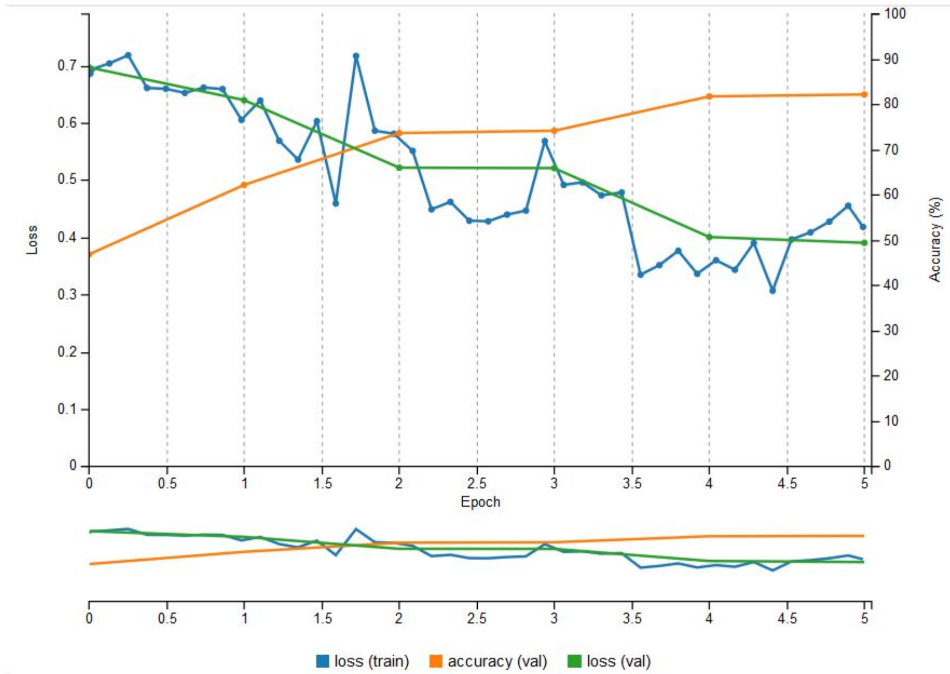
## GPU Task #2

**Aim**
In this task, we build a new dataset for classifying images of dogs and cats into their respective categories. We also create a validation set to determine the performance of the model. Compared to the previous task, we use more data here. We adjust hyperparameters and document the performance of the model. We use AlexNet and GoogLeNet for this task.

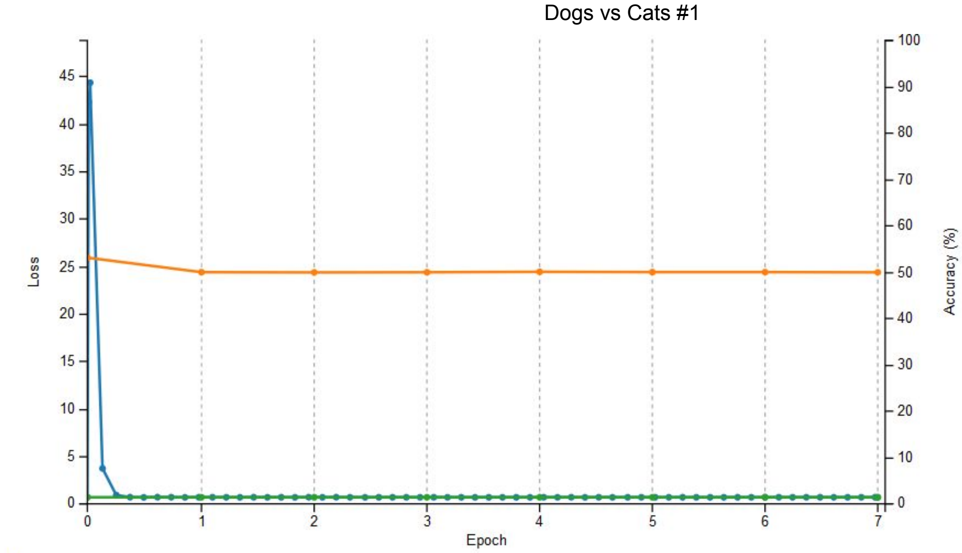
**What is GoogLeNet?**
The winner of the ImageNet Large Scale Visual Recognition Challenge 2014 competition was GoogleNet from Google.
It achieved a top-5 error rate of 6.67%, this is far more efficient than AlexNet. Their architecture consisted of a 22 layer deep CNN but reduced the number of parameters from 60 million (AlexNet) to 4 million.
The network used a CNN inspired by LeNet but implemented a novel element which is dubbed an inception module. It used batch normalization, image distortions and RMSprop.



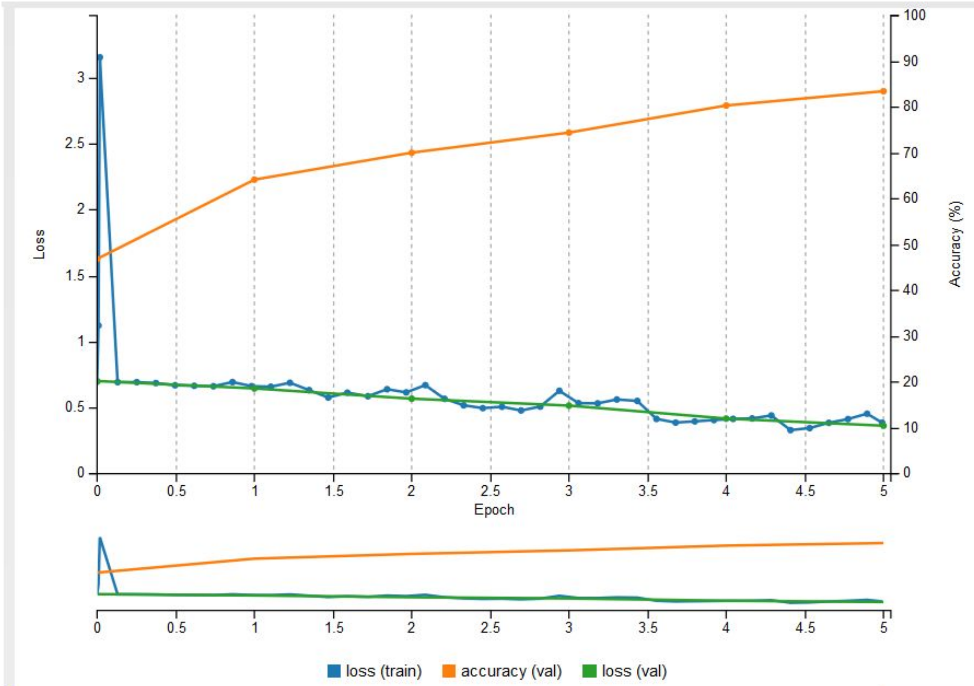

**Approach**
Similar to the previous task, we will use AlexNet and tune it’s parameters. In addition to this, we will also test out the GoogLeNet network.



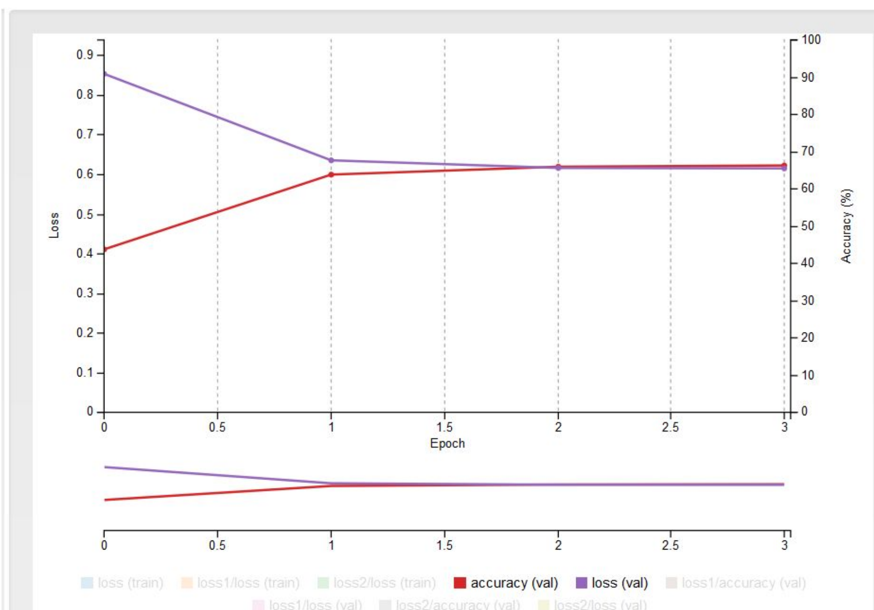

Dogs vs Cats #1



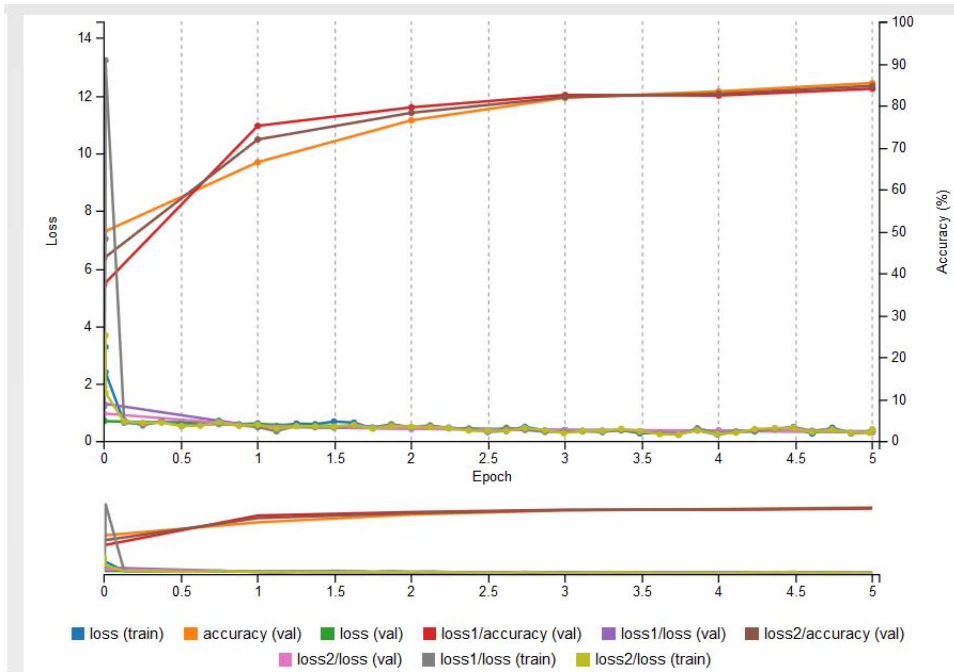

Dogs vs Cats #2




Dogs vs Cats #3




Dogs vs Cats #4




Dogs vs Cats #5

**Results**

| Model | Network | Epochs | Learning Rate | Batch Size | Solver | Accuracy | Loss |
|---|---|---|---|---|---|---|---|
| Dogs vs Cats #1 | AlexNet | 5 | 0.01 Step Down | Default | SGD | 82.2385 | 0.390578 |
| Dogs vs Cats #2 | AlexNet | 7 | 0.01 Step Down | Default | Adam | 49.9681 | 0.69315 |
| Dogs vs Cats #3 | AlexNet | 5 | 0.001 Fixed | Default | AdaGrad | 83.4981 | 0.363066 |
| Dogs vs Cats #4 | GoogLeNet | 3 | 0.01 Step Down | Default | AdaDelta | 66.2884 | 0.614098 |
| Dogs vs Cats #5 | GoogLeNet | 5 | 0.001 Fixed | Default | AdaGrad | 85.501899 | 0.32929 |

## GPU Task #3

**Aim**

In this task, we need to remove a pre-trained model from its environment and deploy it in a real application. We do this using the Caffe Deep Learning framework.

### Approach

We change our output to a readable format. Meaningful post-processing is required because an external application needs to know how to use the output.

#### Forward Propagation: Using your model

This is what we care about. Let's take a look at the function:
`prediction = net.predict([grid_square])` .

Like any function, `net.predict` passes an input, `ready_image` , and returns an output, `prediction` . Unlike other functions, this function isn't following a list of steps, instead, it's performing layer after layer of matrix math to transform an image into a vector of probabilities.

Run the cell below to see the prediction from labeled the labeled data above.

```
# make prediction
prediction = net.predict([ready_image])
print prediction
```

```
[[ 0.70993775  0.29006225]]
```

Interesting, but doesn't contain all that much information. Our network took a normalized 256x256 color image and generated a vector of length 2.

We use this raw output and display it in a meaningful way.

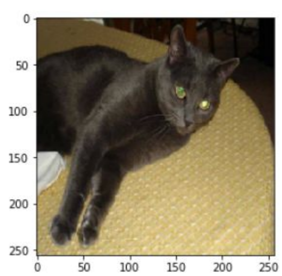
#### Generating a useful output: Postprocessing

At this point, we can really build whatever we want. Your only limit is your programming experience. Before getting creative, let's build something basic. This code will determine whether our network output a higher value for the likelihood of "dog" than it did for "cat." If so, it will display an image that would be appropriate if a dog approached our simulated doggy door. If not, the image represents what we'd want to happen if our network determined a cat was at the door.

```
[49]: print("Input image:")
plt.imshow(input_image)
plt.show()

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"
```

Input image:




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

### Result

Our trained model is now available to be deployed as a python application and the python application will interpret the output of the model in a meaningful way.

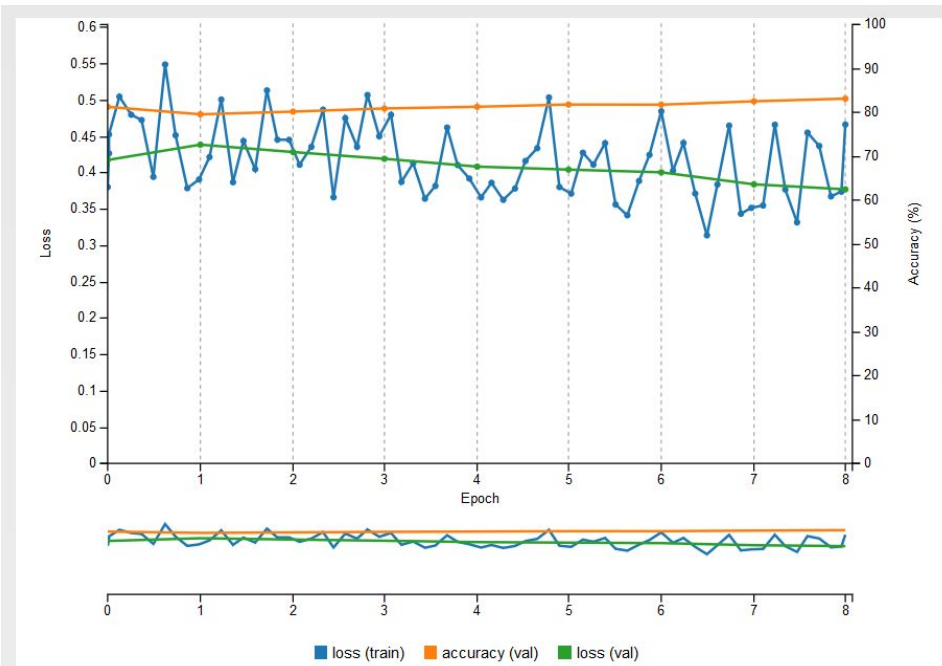
## GPU Task #4

### Aim

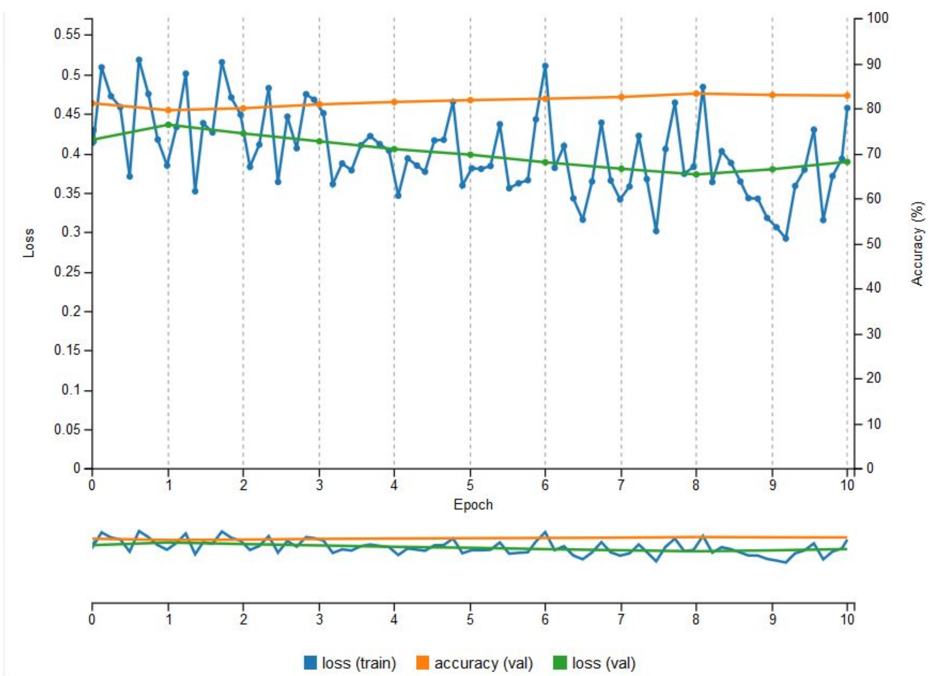
In this task, we would like to improve the performance of a pretrained model by tuning it’s hyperparameters such as learning rate, epochs, solver etc.

**Approach**

- We save a pretrained model (Dogs vs Cats model) which we created in our previous task.
- We change the hyperparameters of this model and document our results.




Dogs vs Cats #6




Dogs vs Cats #7

| Model | Network | Epochs | Learning Rate | Batch Size | Solver | Accuracy | Loss |
|---|---|---|---|---|---|---|---|
| Dogs vs Cats #6 | AlexNet | 8 | 0.0001 Fixed | Default | SGD | 83.1154 | 0.376761 |
| Dogs vs Cats #7 | AlexNet | 10 | 0.0001 Fixed | Default | NAG | 82.956 | 0.389105 |

## GPU Task #5

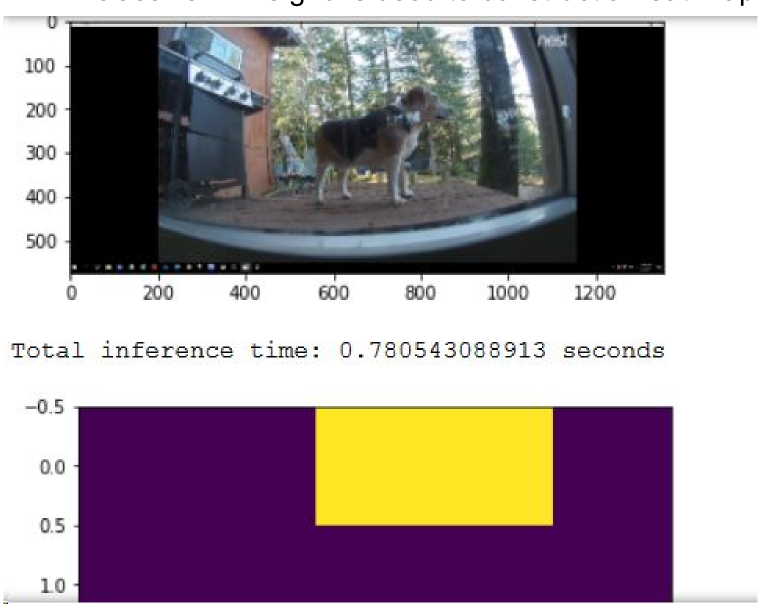
**Aim**

In this task we have the following objectives-

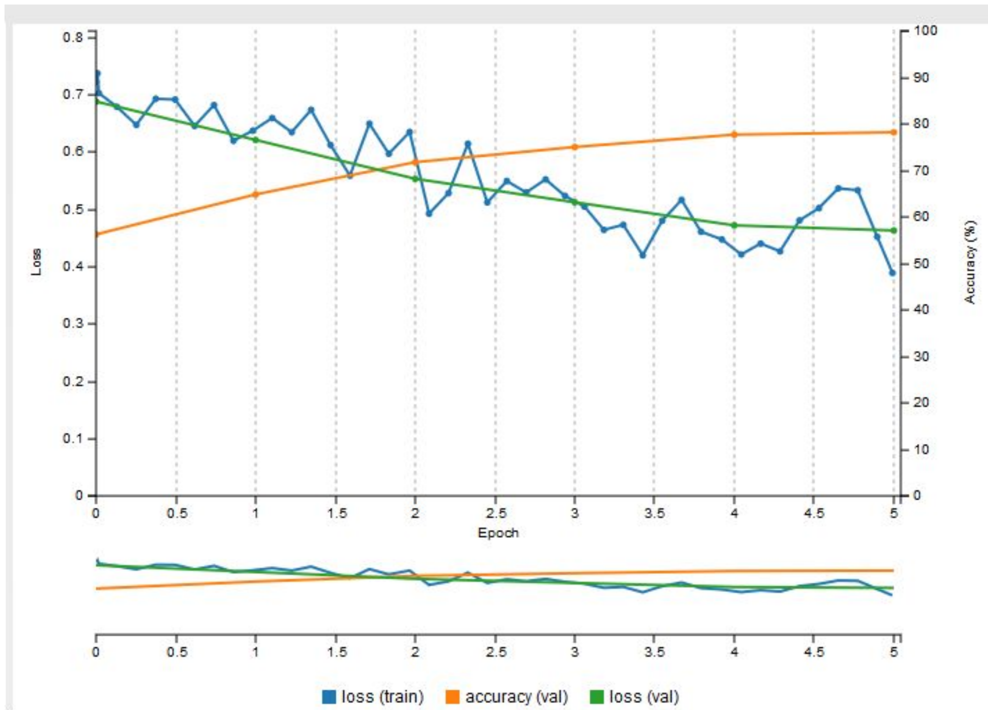
- Combine deep learning with computer vision to detect and localize objects within an image.
- Modifying the internals of an existing neural network.
- Picking the right Neural Network for the task at hand.

**Approach**

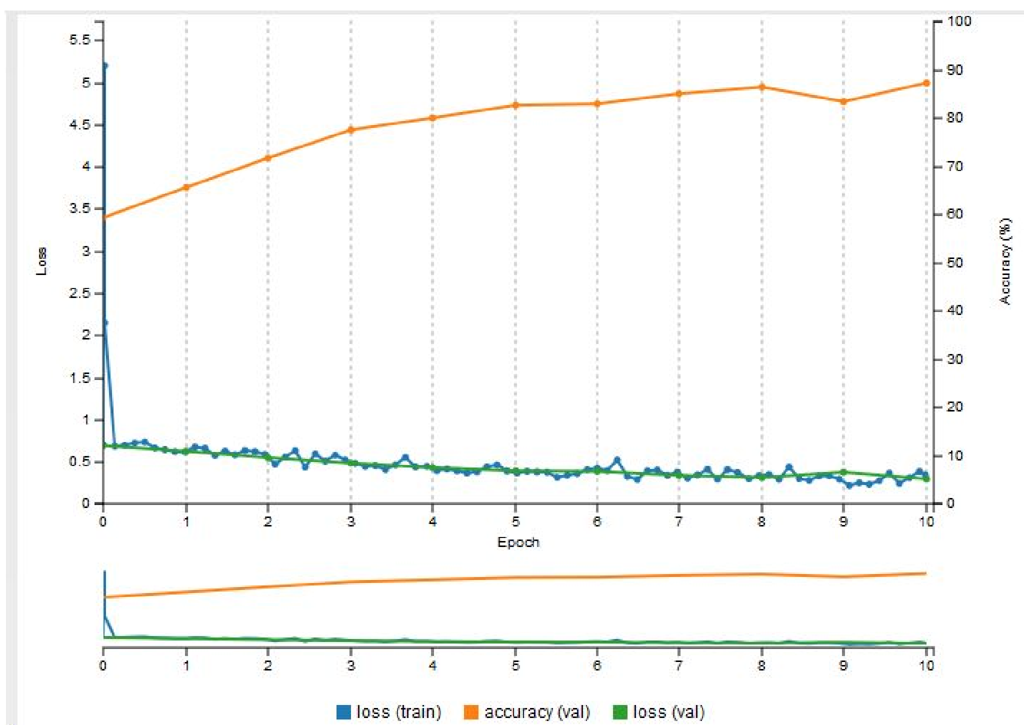
- We use a SoftMax grid square which highlights grids in which a dog appears with a higher probability value. At its core, this block is cutting our input image into 256X256 squares, running each of them through our dog classifier, and creating a new image that is blue where there is not a dog and red where there is one, as determined by our classifier. This grid is used to construct a heat-map as shown below.




- Next, we will attempt to modify the internal working of the AlexNet neural network by changing fully connected layers to convolutional layers.
- We removed a "fully connected" layer, which is a traditional matrix multiplication. We added a "convolutional" layer, which is a "filter" function that moves over an input matrix.
- 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.




Fully Convolutional Layer #1




Fully Convolutional Layer #2

- We will now use DetectNet, which is a Neural Network architecture used by Nvidia.

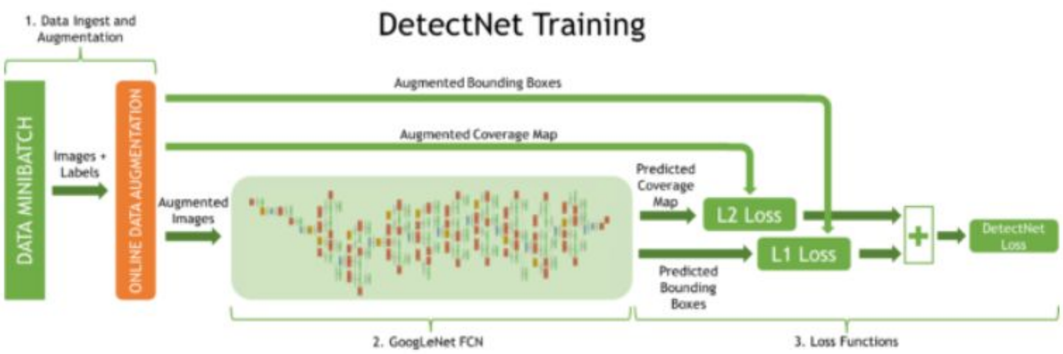
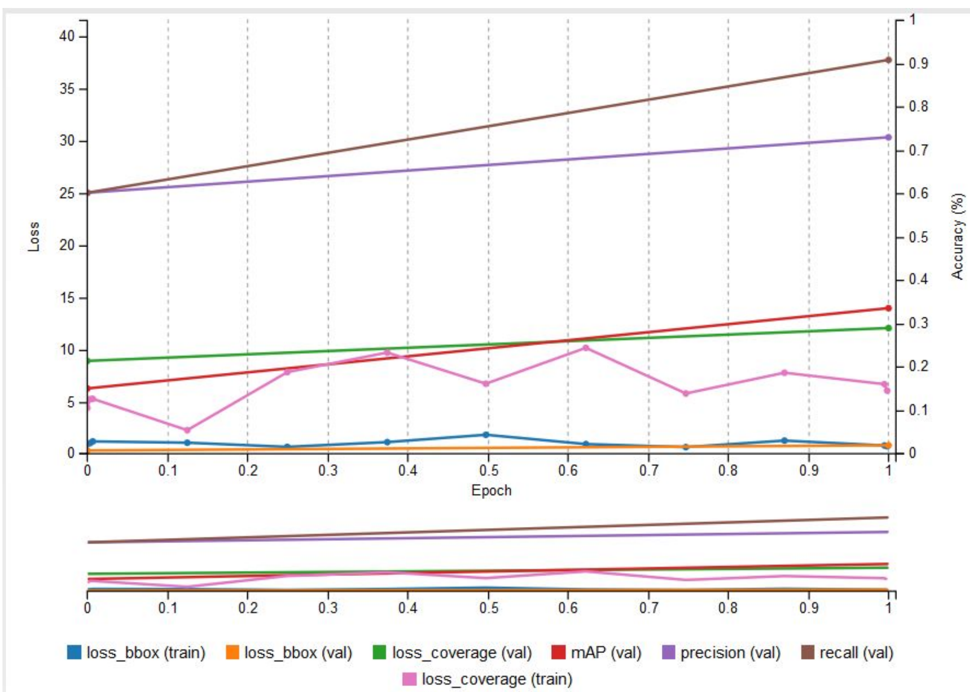
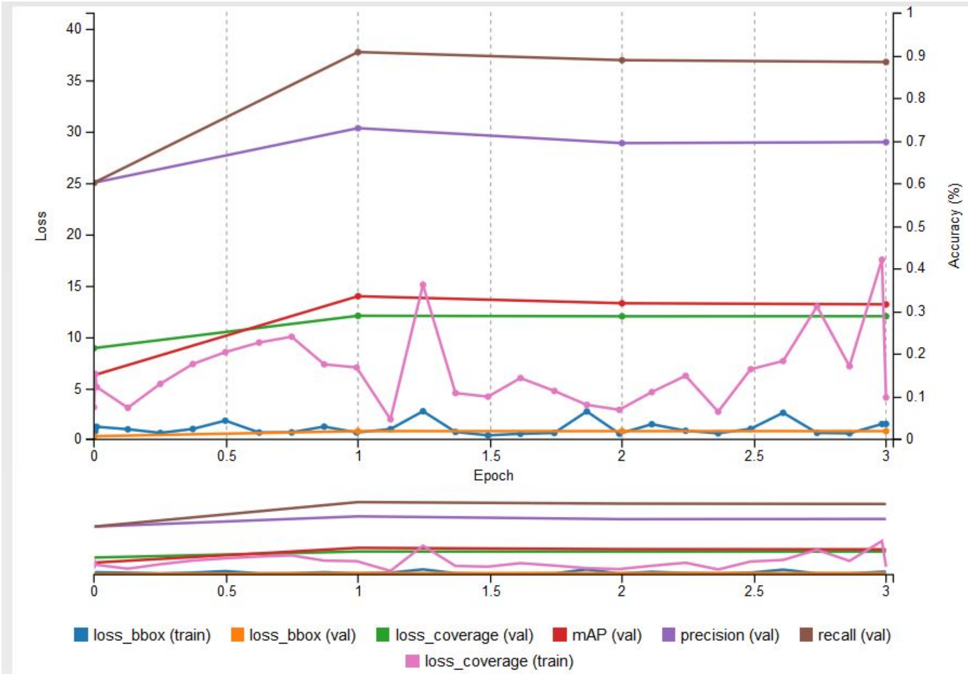



Figure 3: DetectNet structure for training.

A fully-convolutional network (FCN) performs feature extraction and prediction of object classes and bounding boxes per grid square.






## Assessment

### Aim

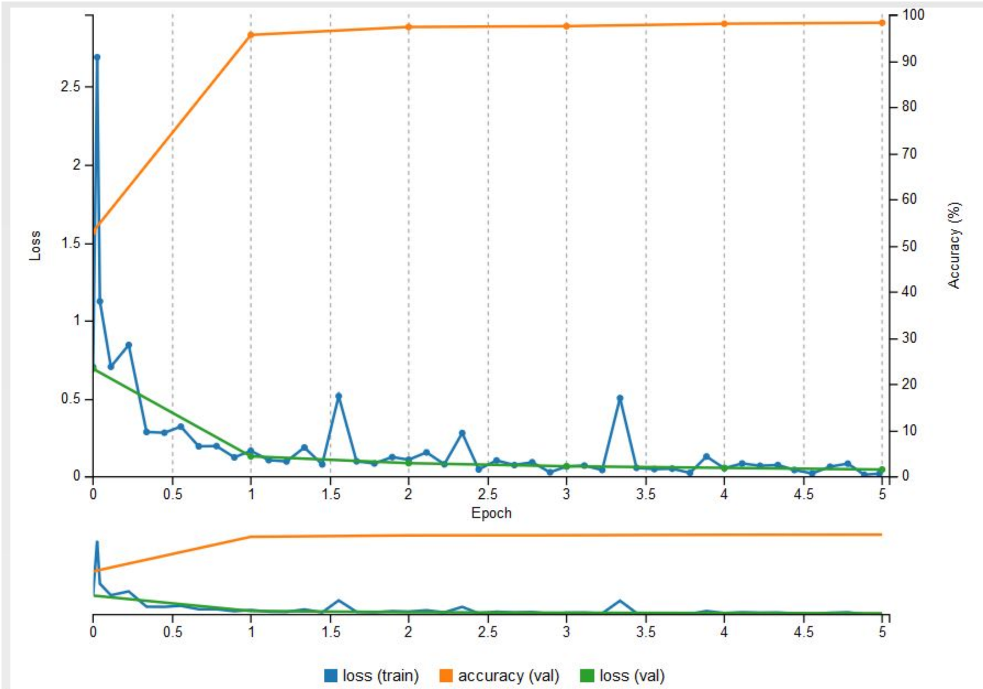
In this task, we use all of the skills we have learned in our past GPU tasks and we build an image classification model for identifying the faces of whales and we deploy it in a python application.

### Approach

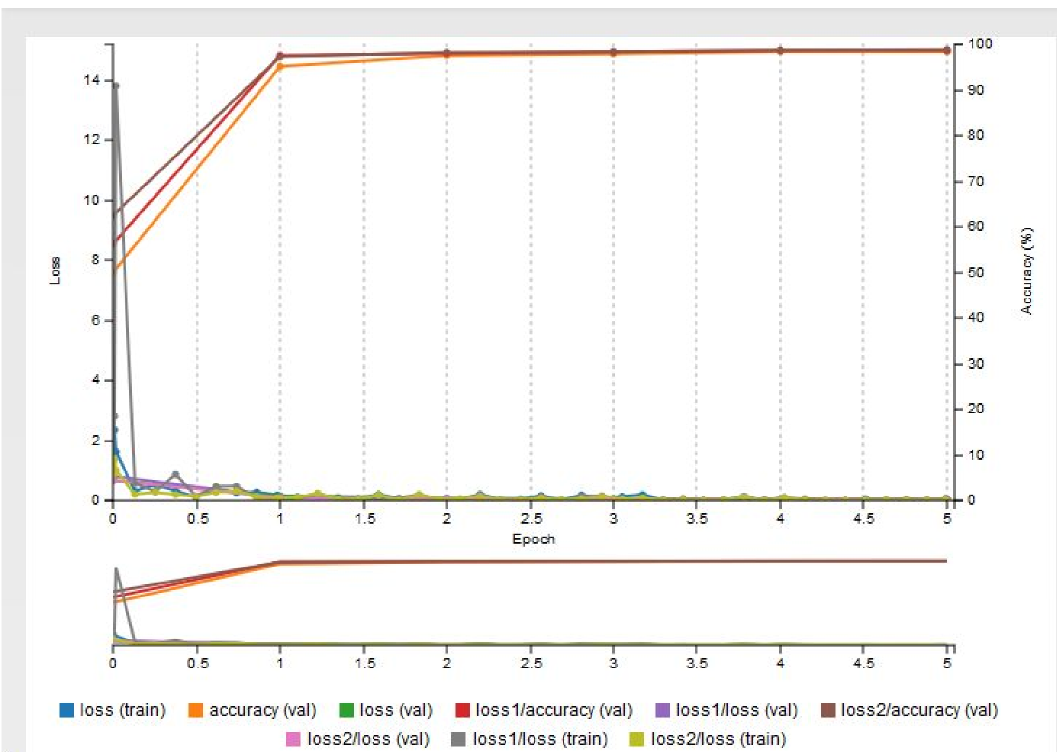
We will train an AlexNet Model for the Whale Faces Dataset, then we will use the Caffe framework on Python to integrate our model and deploy it in a readable format.

Our tasks include:-

- Build a dataset for whale faces
- Run an image classification model on this dataset.
- Tune hyperparameters to obtain best performance.
- Deploy the model in a python application.




Whale Faces #1




Whale Faces #2

```
In [8]: !python submission.py '/dli/data/whale/data/train/face/w_1.jpg'  #This should return "whale" at the very bottom
```

```
I0213 17:02:02.834420   249 net.cpp:1137] Copying source layer relu2 Type:ReLU #blobs=0
I0213 17:02:02.834434   249 net.cpp:1137] Copying source layer norm2 Type:LRN #blobs=0
I0213 17:02:02.834439   249 net.cpp:1137] Copying source layer pool2 Type:Pooling #blobs=0
I0213 17:02:02.834444   249 net.cpp:1137] Copying source layer conv3 Type:Convolution #blobs=2
I0213 17:02:02.834861   249 net.cpp:1137] Copying source layer relu3 Type:ReLU #blobs=0
I0213 17:02:02.834870   249 net.cpp:1137] Copying source layer conv4 Type:Convolution #blobs=2
I0213 17:02:02.835199   249 net.cpp:1137] Copying source layer relu4 Type:ReLU #blobs=0
I0213 17:02:02.835213   249 net.cpp:1137] Copying source layer conv5 Type:Convolution #blobs=2
I0213 17:02:02.835470   249 net.cpp:1137] Copying source layer relu5 Type:ReLU #blobs=0
I0213 17:02:02.835485   249 net.cpp:1137] Copying source layer pool5 Type:Pooling #blobs=0
I0213 17:02:02.835499   249 net.cpp:1137] Copying source layer fc6 Type:InnerProduct #blobs=2
I0213 17:02:02.852859   249 net.cpp:1137] Copying source layer relu6 Type:ReLU #blobs=0
I0213 17:02:02.852895   249 net.cpp:1137] Copying source layer drop6 Type:Dropout #blobs=0
I0213 17:02:02.852908   249 net.cpp:1137] Copying source layer fc7 Type:InnerProduct #blobs=2
I0213 17:02:02.860278   249 net.cpp:1137] Copying source layer relu7 Type:ReLU #blobs=0
I0213 17:02:02.860307   249 net.cpp:1137] Copying source layer drop7 Type:Dropout #blobs=0
I0213 17:02:02.860319   249 net.cpp:1137] Copying source layer fc8 Type:InnerProduct #blobs=2
I0213 17:02:02.860358   249 net.cpp:1129] Ignoring source layer loss
whale
```

```
In [9]: !python submission.py '/dli/data/whale/data/train/not_face/w_1.jpg'  #This should return "not whale" at the very bottom
```

```
I0213 17:02:06.348668   264 net.cpp:1137] Copying source layer norm2 Type:LRN #blobs=0
I0213 17:02:06.348675   264 net.cpp:1137] Copying source layer pool2 Type:Pooling #blobs=0
I0213 17:02:06.348681   264 net.cpp:1137] Copying source layer conv3 Type:Convolution #blobs=2
I0213 17:02:06.349108   264 net.cpp:1137] Copying source layer relu3 Type:ReLU #blobs=0
I0213 17:02:06.349123   264 net.cpp:1137] Copying source layer conv4 Type:Convolution #blobs=2
I0213 17:02:06.349438   264 net.cpp:1137] Copying source layer relu4 Type:ReLU #blobs=0
I0213 17:02:06.349452   264 net.cpp:1137] Copying source layer conv5 Type:Convolution #blobs=2
I0213 17:02:06.349673   264 net.cpp:1137] Copying source layer relu5 Type:ReLU #blobs=0
I0213 17:02:06.349686   264 net.cpp:1137] Copying source layer pool5 Type:Pooling #blobs=0
I0213 17:02:06.349692   264 net.cpp:1137] Copying source layer fc6 Type:InnerProduct #blobs=2
I0213 17:02:06.366268   264 net.cpp:1137] Copying source layer relu6 Type:ReLU #blobs=0
I0213 17:02:06.366299   264 net.cpp:1137] Copying source layer drop6 Type:Dropout #blobs=0
I0213 17:02:06.366312   264 net.cpp:1137] Copying source layer fc7 Type:InnerProduct #blobs=2
I0213 17:02:06.373616   264 net.cpp:1137] Copying source layer relu7 Type:ReLU #blobs=0
I0213 17:02:06.373642   264 net.cpp:1137] Copying source layer drop7 Type:Dropout #blobs=0
I0213 17:02:06.373647   264 net.cpp:1137] Copying source layer fc8 Type:InnerProduct #blobs=2
I0213 17:02:06.373673   264 net.cpp:1129] Ignoring source layer loss

not whale
```

After deployment

## Results

| Model | Network | Epochs | Learning Rate | Batch Size | Solver | Accuracy | Loss |
|---|---|---|---|---|---|---|---|
| Whale Faces #1 | AlexNet | 5 | 0.001 Fixed | Default | AdaGrad | 98.3715 | 0.0466084 |
| Whale Faces #2 | GoogleLeNet | 5 | 0.01 Step Down | Default | SGD | 98.4115 | 0.04533 |