# Assignment 2 Part 1

NVIDIA LABS OBSERVATIONS

Shravanthi Sridhar

# Task 1: Image Classification

**Dataset:** Louie and Not Louie

**No of images:** 16 images with 8 of each category.

**Recording of different experiments:**
https://docs.google.com/spreadsheets/d/1ANlieYHWwjxCinbG5UO6CtU2lctQrRmYyK1Q231_aPE/edit#gid=0

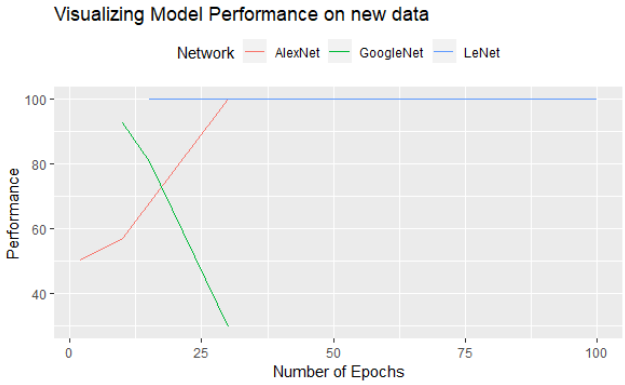
**Inference:**

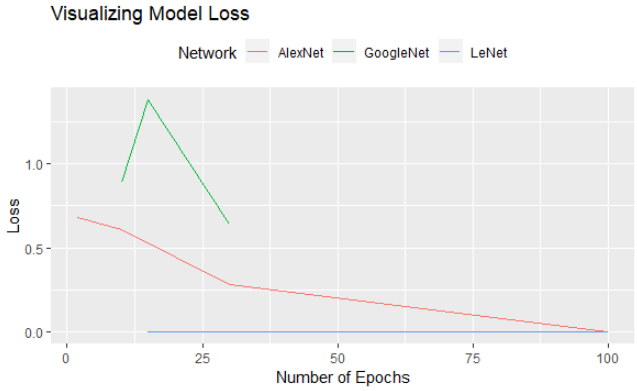
- LeNet gives Model Loss 0 and identifies the new test image with 100% Accuracy and 15 epochs. Also, the model takes only 26 seconds to train.
- AlexNet gives Model Loss 0.28 and Accuracy 100% at 30 epochs with 35secs to train the model.
- GoogLeNet doesn't seem to perform well for this Classification problem when compared to LeNet and AlexNet.

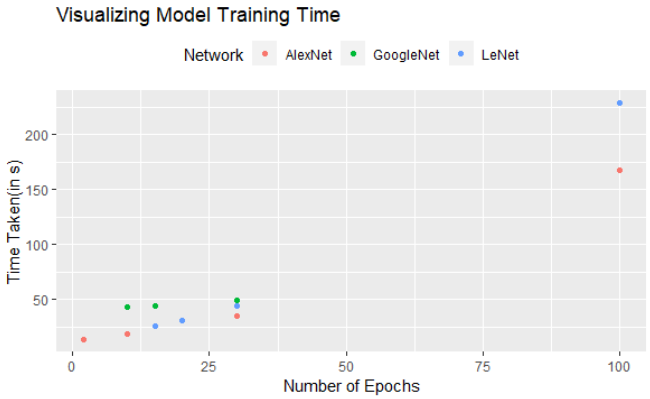
**Comparison between three types of architectures used:**

| Category | LeNet | AlexNet | GoogLeNet |
|---|---|---|---|
| Number of layers | 6 | 7 | 22 |
| Regularization | | Uses dropout to deal with regularization. | Batch Normalization |
| Nonlinearity function used | tanh and Sigmoid | Relu | Relu |
| Number of parameters | 60000 | 60 million | 4 million |
| Layers Configuration | Fully connected layers at the end | Deeper network with more filters per layer along with stacked conv layers | Inception layer and Dimension reduction prior to convolution<br>Parallel paths with different filter sizes and operations meant to capture sparse patterns |
| Type of pooling | Average pooling | Max Pooling | Average Pooling |

## Visual Comparisons:




## Problems:

Small dataset provided for training due to which network leads to overfitting.

## Task 2: Performance improvement along with exposure to more training data

**Dataset:** Dogs and Cats

**No of images:** 18750 labelled images of dogs and cats

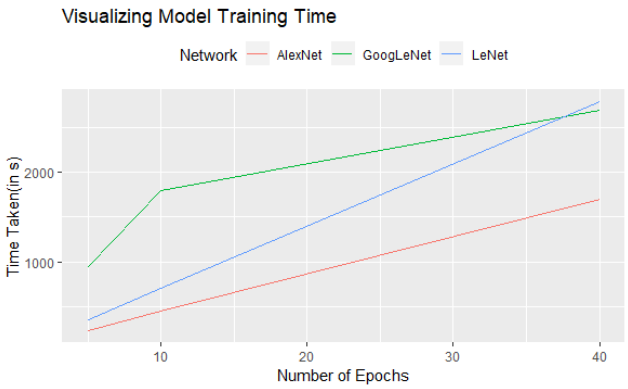
**Recording of different experiments:**

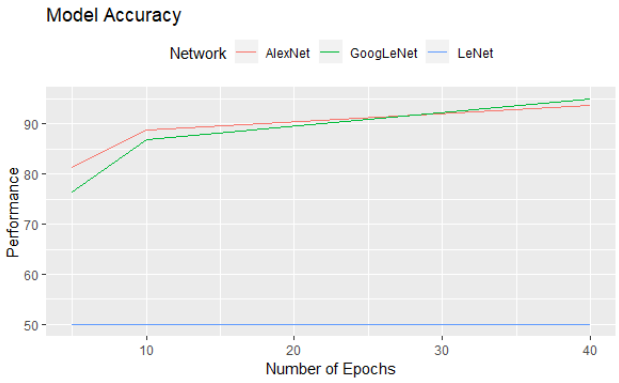
https://docs.google.com/spreadsheets/d/1ANlieYHWwjxCinbG5UO6CtU2lctQrRmYyK1Q231_aPE/edit#gid=532151771

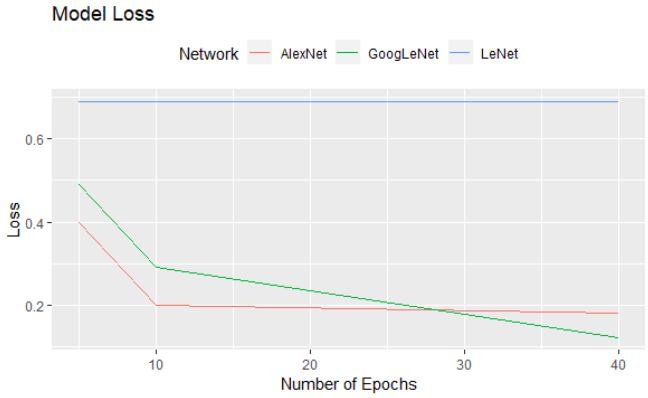
**Observations:**

- With a greater number of data available for training, better model is built post exposure to different variety of data.
- Model Performance assessment methods are studied in this task.
- Comparing Validation Loss and Accuracy of Model, the following are observed:
    - AlexNet takes 28 mins to train model and provides accuracy of 93.8 and Loss .18 with 40 epochs.
    - GoogLeNet takes 44 mins to train model and provides accuracy of 95%, Loss .12 with 15 epochs.
    - Both AlexNet and GoogLeNet give 100% accurate result when tested with a new test image.
    - LeNet takes 46.5 mins to train model and provides accuracy of 50.01% along with Loss .69 at 40 epochs.
    - It is clearly visible that LeNet architecture does perform well with the given dataset for this Image Classification problem.

**Visual Comparisons:**






# Task 3: Model Deployment

**Given Task:** Deploy Dogs Vs Cats model to take inputs from a security camera and open a simulated door if the model returns dog

**Observations:**

- Simulator is created for the task.
- Model generates two files: architecture and weights.
- Images are normalized and resized to reduce the computation complexity.
- Application takes input from camera and converts it to a datatype that network understands, generates and output.
- Output generated is in terms of probability where probability of success or near 1 means that the image is a Dog and vice versa.
- The output generated is then converted to something useful to the user. In this case seeing the image:
    - If it is a Dog, message stating “Welcome dog!” is displayed.
    - Else, for a cat, “Sorry Cat” is displayed.

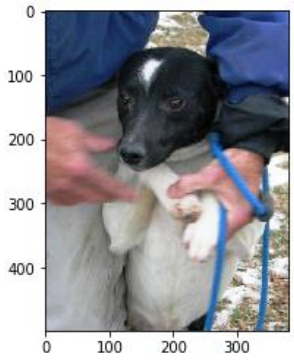
After testing several samples from the given images, it is seen that model is deployed and interpreted in right way as given in task.

Architecture file being deploy.protxt and weights file being in the most recent captures snapshot file.

**Example:**

In case a Dog is captured in camera

Input:

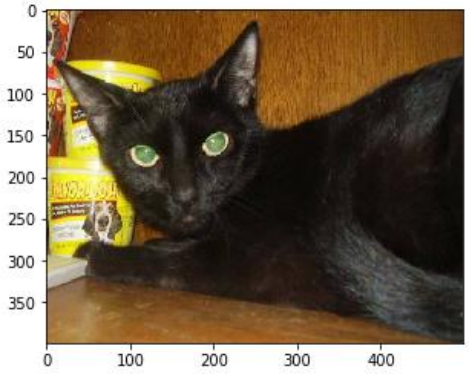



Output:

```
[[ 0.42844081  0.57155913]]
Output:
Welcome dog! https://www.flickr.com/photos/aidras/5379402670
None
```

Incase Cat is captured in camera

Input:




Output:

```
[[ 0.79810351  0.20189646]]
Output:
Sorry cat:( https://media.giphy.com/media/jb8aFEQk3tADS/giphy.gif
None
```

## Task 4: Performance during Training

**Objective:** Improving Training performance with existing model

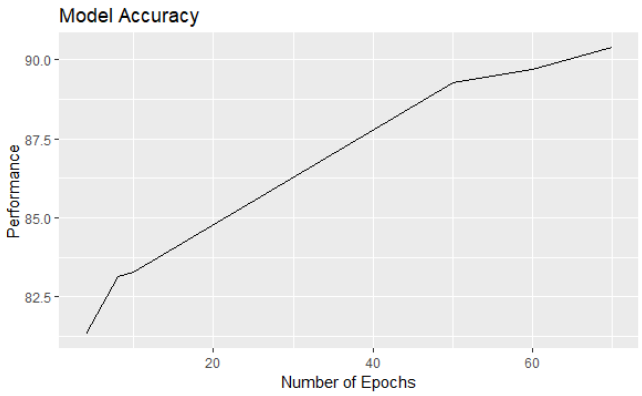
**Dataset:** Dogs and Cats

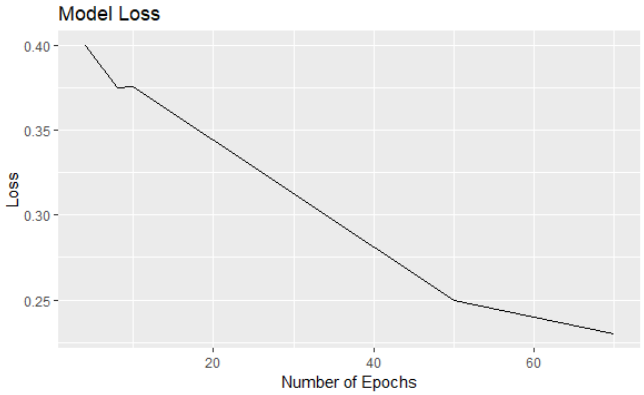
**Recording of different experiments:**
https://docs.google.com/spreadsheets/d/1ANlieYHWwjxCinbG5UO6CtU2lctQrRmYyK1Q231_aPE/edit#gid=299622262

**Observations:**

- Saving existing model saves network architecture used for the specified model.
- Using Pretrained models saves time and lets us continue from where we left at ease.
- Model Performance seems to improve with increase in number of epochs.
    - Model Accuracy increases, and Model Loss decreases.

**Visualizing Model Performance:**






# Task 5: Object Detection

**Approach 1: Sliding window**

Sliding a grid box around image and classifying each image crop in box.

Performance:

This method doesn't work best for detecting the object, dog in this case. The camera input image seems to capture an entire larger grid including too much rather than the dog.

Inference Time taken for this task is .76 s

Problems:

Since sliding window is used with non-overlapping grid squares, it leads to partial identification of object which eventually brings misclassifications into place.

**Approach 2: Modifying Network Architecture**

Initially, AlexNet architecture was used for training model in case od sliding window approach which didn't turn out to be the best option. In this section, the network architecture is adjusted so that model performance seems to improve.

In this approach, the entire image is fed through the trained model instead of grid squares which helps in better object detection.

AlexNet is converted into a fully convolution network in this case.

Performance:

This approach brings improvement in model. The object is identified better than the sliding window approach.

Inference Time taken is 17 secs.

**Approach 3: DetectNet**

It is an End to End structure similar to GoogLeNet Fully Connected Network Architecture.

The network takes about 16 hours to train on a GPU.

For each object in the image, training label captures not only the class of the object but also the coordinates of the corners of its bounding box.

Performance:

This is the best approach applied so far for object detection, but model training time is too much due to the complicated network structure.

## Task 6: Assessment

**Objective:** Build a classifier for classifying image as “Whale” or “not Whale”.

**Task:** Train model with accuracy > 80%

**Model Hyper parameters:**

Learning Rate: .01

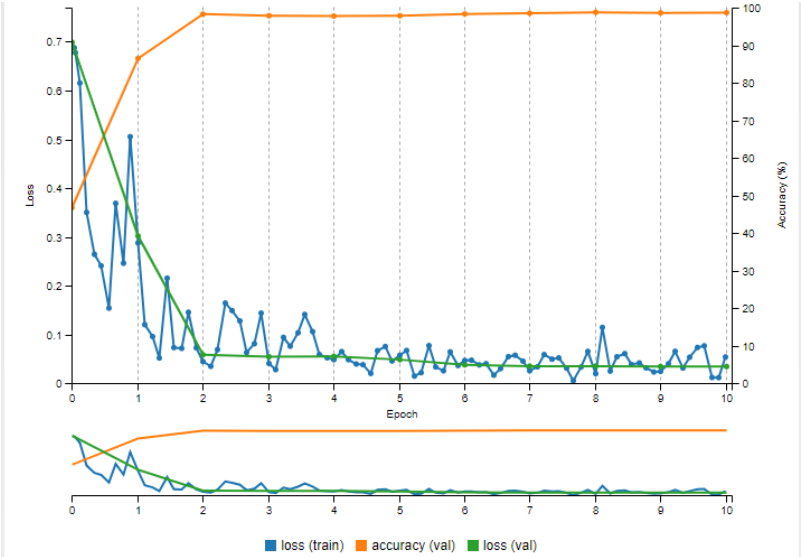
Type: Step Down Learning Rate

Training time: 3 mins 11 secs

Optimizer: SGD

Number of epochs: 10

## Model Performance




Validation Accuracy: 98.8%

Validation Loss: .03

## Output snapshots:

```
In [16]: !python submission.py '/dli/data/whale/data/train/face/w_1.jpg'  #This should return "whale" at the very bottom
I0212 05:15:21.221859   367 net.cpp:1137] Copying source layer conv3 Type:Convolution #blobs=2
I0212 05:15:21.222460   367 net.cpp:1137] Copying source layer relu3 Type:ReLU #blobs=0
I0212 05:15:21.222474   367 net.cpp:1137] Copying source layer conv4 Type:Convolution #blobs=2
I0212 05:15:21.222923   367 net.cpp:1137] Copying source layer relu4 Type:ReLU #blobs=0
I0212 05:15:21.222935   367 net.cpp:1137] Copying source layer conv5 Type:Convolution #blobs=2
I0212 05:15:21.223237   367 net.cpp:1137] Copying source layer relu5 Type:ReLU #blobs=0
I0212 05:15:21.223249   367 net.cpp:1137] Copying source layer pool5 Type:Pooling #blobs=0
I0212 05:15:21.223256   367 net.cpp:1137] Copying source layer fc6 Type:InnerProduct #blobs=2
I0212 05:15:21.245430   367 net.cpp:1137] Copying source layer relu6 Type:ReLU #blobs=0
I0212 05:15:21.245473   367 net.cpp:1137] Copying source layer drop6 Type:Dropout #blobs=0
I0212 05:15:21.245479   367 net.cpp:1137] Copying source layer fc7 Type:InnerProduct #blobs=2

I0212 05:15:21.255370   367 net.cpp:1137] Copying source layer relu7 Type:ReLU #blobs=0
I0212 05:15:21.255398   367 net.cpp:1137] Copying source layer drop7 Type:Dropout #blobs=0
I0212 05:15:21.255403   367 net.cpp:1137] Copying source layer fc8 Type:InnerProduct #blobs=2
I0212 05:15:21.255432   367 net.cpp:1129] Ignoring source layer loss
[[  9.99872565e-01   1.27485429e-04]]
Output:
whale
```

```
In [17]: !python submission.py '/dli/data/whale/data/train/not_face/w_1.jpg'  #This should return "not whale" at the very bottom
I0212 05:15:39.443898   382 net.cpp:1137] Copying source layer pool2 Type:Pooling #blobs=0
I0212 05:15:39.443904   382 net.cpp:1137] Copying source layer conv3 Type:Convolution #blobs=2
I0212 05:15:39.444486   382 net.cpp:1137] Copying source layer relu3 Type:ReLU #blobs=0
I0212 05:15:39.444501   382 net.cpp:1137] Copying source layer conv4 Type:Convolution #blobs=2
I0212 05:15:39.444936   382 net.cpp:1137] Copying source layer relu4 Type:ReLU #blobs=0
I0212 05:15:39.444950   382 net.cpp:1137] Copying source layer conv5 Type:Convolution #blobs=2
I0212 05:15:39.445268   382 net.cpp:1137] Copying source layer relu5 Type:ReLU #blobs=0
I0212 05:15:39.445282   382 net.cpp:1137] Copying source layer pool5 Type:Pooling #blobs=0
I0212 05:15:39.445289   382 net.cpp:1137] Copying source layer fc6 Type:InnerProduct #blobs=2
I0212 05:15:39.467512   382 net.cpp:1137] Copying source layer relu6 Type:ReLU #blobs=0
I0212 05:15:39.467550   382 net.cpp:1137] Copying source layer drop6 Type:Dropout #blobs=0
I0212 05:15:39.467556   382 net.cpp:1137] Copying source layer fc7 Type:InnerProduct #blobs=2
I0212 05:15:39.477481   382 net.cpp:1137] Copying source layer relu7 Type:ReLU #blobs=0
I0212 05:15:39.477524   382 net.cpp:1137] Copying source layer drop7 Type:Dropout #blobs=0
I0212 05:15:39.477530   382 net.cpp:1137] Copying source layer fc8 Type:InnerProduct #blobs=2
I0212 05:15:39.477560   382 net.cpp:1129] Ignoring source layer loss
[[  9.35984775e-04   9.99064088e-01]]
Output:
not whale
```