

Abstract

In this project, a network will be trained for classifying real world objects into predefined classes. In addition to training a network on the supplied dataset, a different network will be chosen and trained using self acquired data. The quality and quantity of data acquired will be discussed.

The network would be able to successfully classify the objects in the self collected data into 4 classes with reasonable accuracy.

Introduction

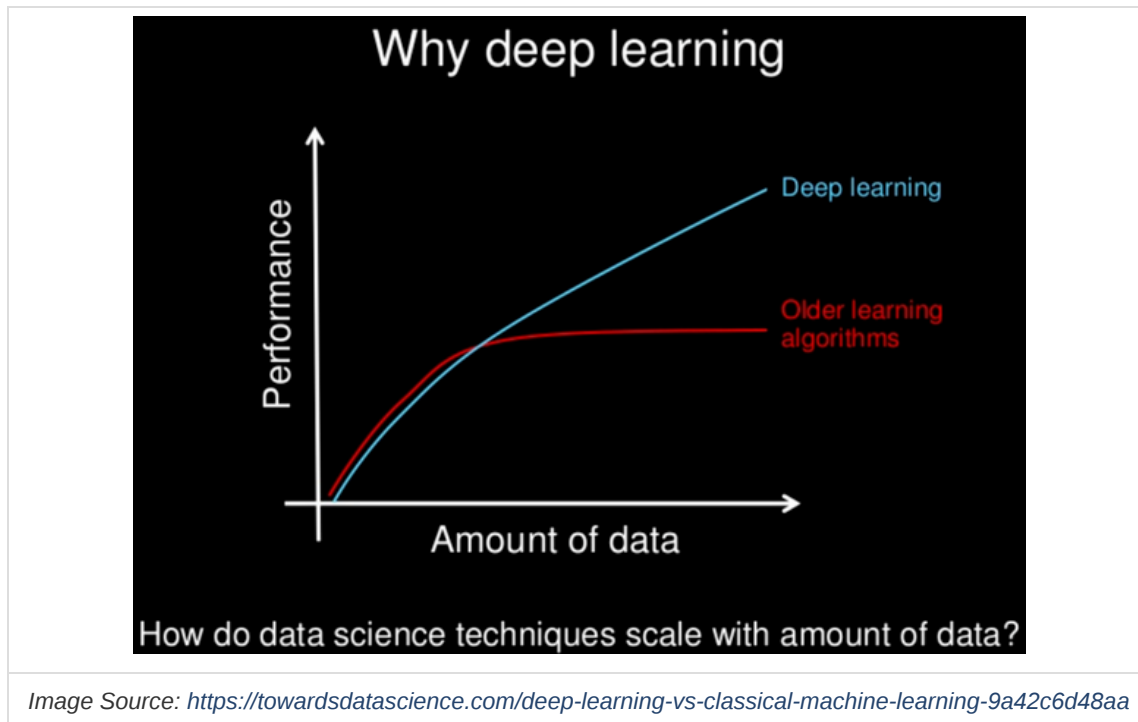
The classification of objects in the 2D image had been tried to be solved by a lot of classical methods but deep learning was able to become the new State of the Art technology. Deep learning shines when there is a lot of data.

The advantages of deep learning :

1. Scales well with increase in data.
2. No in-depth feature detection rules of the data needs to be hard coded.
3. Adaptable and transferable to the new problem datasets.

In the defense of the classical algorithms[1], classical algorithms are better than deep learning in some cases:

1. Needs no training time.
2. Financially and computationally cheap to run.
3. Easier to implement: The parameter tuning in the case of the classical algorithms is more straight-forward because of the thorough understanding of the inner working of them. In case of the deep learning, even the researchers do not understand the inner working of the DL network(*almost a blackbox*).



To investigate the effectiveness of the inference models with random everyday objects that are used extensively by the humans, it was decided to classify the everyday objects on the work desk for this inference task.

Classes - **Wallet, Mouse, NailCutter, Nothing**

All the inference tasks essentially have similar steps that need to be followed. **Nvidia DIGITS is a system to smoothen this workflow.** This project will employ the Nvidia DIGITS workflow for developing the inference system.

Nvidia DIGITS is a way to quickly:

- **Build a dataset** that the network can take as input .
- **Define, customize and visualize** a network architecture.
- **Train the network** with the dataset.

Nvidia DIGITS provides a cleaner interface to do all of the above, through the web browser.

The common inference tasks are:

1. Classification: Answers the *what* in the image
2. Detection: Answers the *where* in the image
3. Segmentation: Pixel level labeling of the images.

Formulation

Transfer Learning

In practice, it is very rare that a CovNet is trained from scratch (with random initialization) because of the unavailability of the vast amount of dataset and compute power needed.

Andrew Ng once explained neural network as it being like a rocket ship. Rocket ship needs a powerful engine and lot of fuel. Powerful engine but less fuel fails to make it to the orbit. Weak engine but lot of fuel fails to even liftoff. Analogously, DL models are engines and dataset is the fuel.

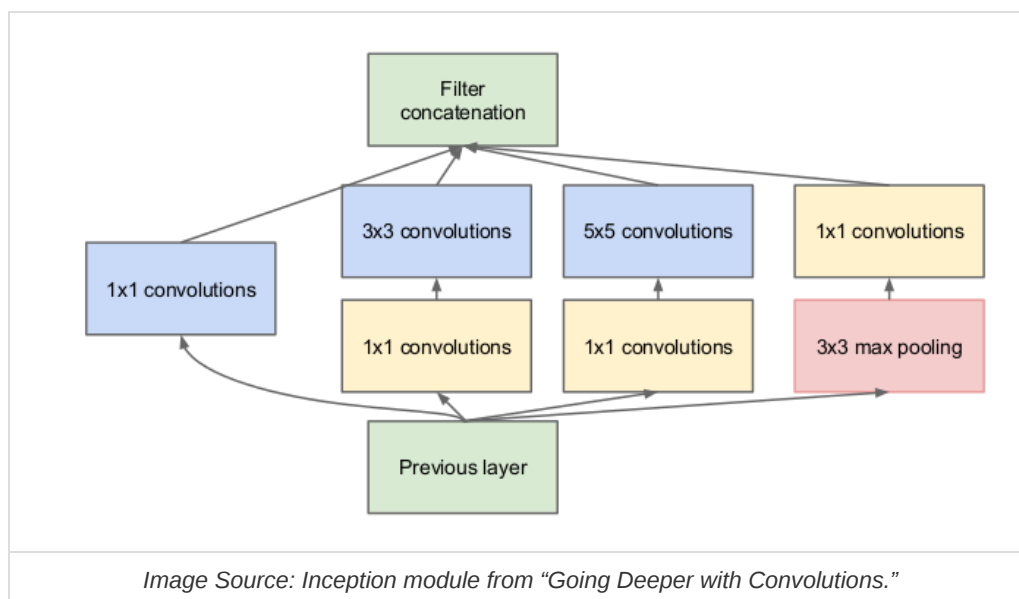
Transfer learning is the finetuning of a network pretrained on a large dataset of the problem similar to the original problem.

In the project transfer learning was leveraged. A GoogLeNet that was pretrained on the ImageNet(1000 Image classes) was used for classifying the everyday objects in the self acquired data.

The [GoogLeNet](#) was used because it has 22 layers and 12 times less parameters than the AlexNet because of:

1. The absence of the fully connected later at the end.
2. Use of the concept of a **local mini network** inside a bigger network called the **Inference module**. The inference module played the major role in the parameter reduction. This solved the problem of deciding which convolution to use - 1x1, 3x3, 5x5, 7x7. It enabled the network to use **all of the convolutions concatenated depth wise**.

The naive **Inference module** used the depth wise concatenation of the 1x1 , 3x3 , 5x5 convolutional layers along with a 3x3 max pooling . The max pooling layer led to the depth staying the same or increasing in the output layer of the Inference module. This problem was solved by 1x1 convolutional layer with the 3x3, 5x5 and more importantly the max pooling layer. The number of 1x1 convolutional layer **helped reduce the dimensionality of the output layer** because the depth of output layer will just be the number of 1x1 convolutional layers.



GoogLeNet was faster and more accurate than the AlexNet. **The idea around the development of GoogLeNet was that it should be able to run even in the smartphone.**(*calculation budget around 1.5 billion multiply-adds on prediction*)

Also, SGD performed better than the RMSProp and was used in the finetuning of the network.

Data Acquisition

Number of images

Number of Training Images

Mouse	Wallet	Nail Cutter	Nothing
317	216	64	25

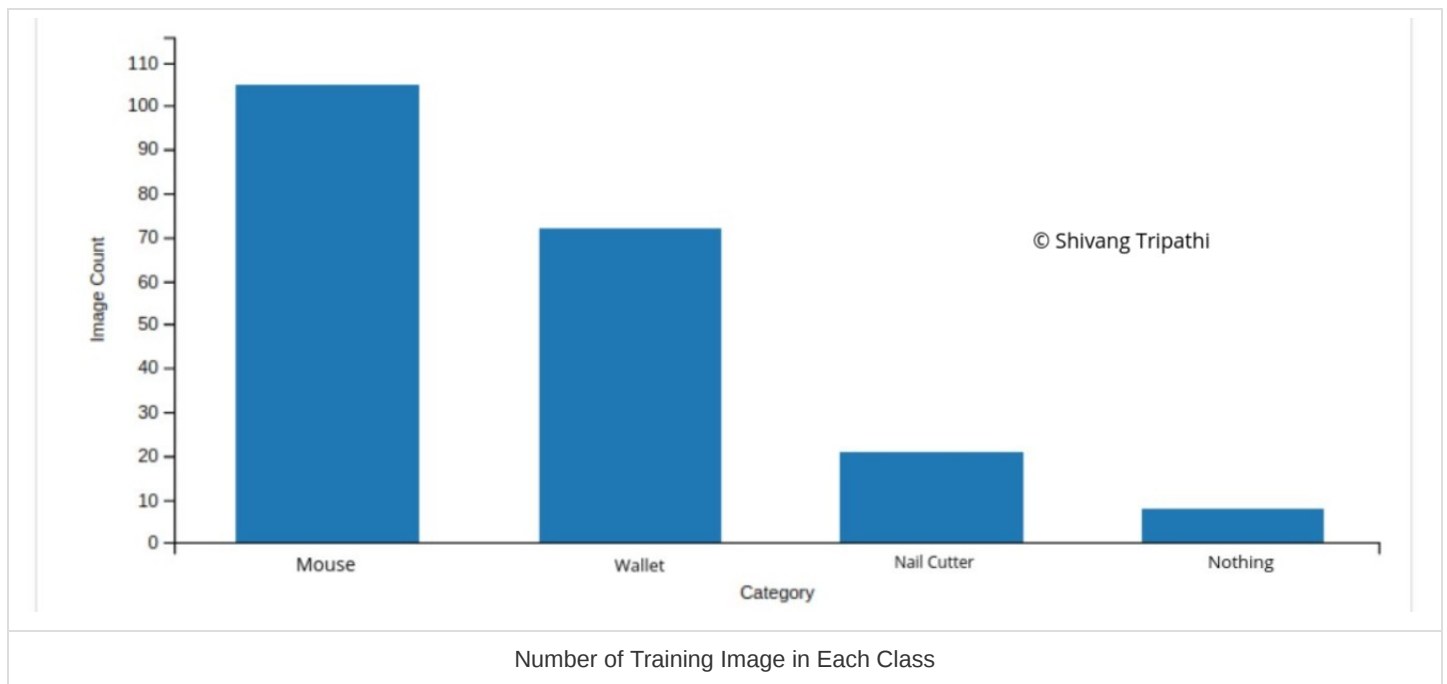
Training Images Distribution: This distribution of images for each class was used for training the network.



Number of Test Images

Mouse	Wallet	Nail Cutter	Nothing
105	72	21	8

Test Images Distribution: This distribution of images was used for the validation of the network. The network validates its accuracy after each run from these images.



The collected image data was augmented by using the rotation of:

- 90 degrees clockwise
- 90 degrees counter- clockwise
- 180 degrees

As a result, the number of images in the dataset quadrupled.

As it can be seen here, the number of images for the `wallet` and `Mouse` class is much more than the `Nail Cutter` . This will be explained in detail under the [Discussion](#) section.

Size of images

The images were captured after setting **image aspect ratio to 1:1** so that the images are square in shape. This is because the network that we will be training on accepts `256x256` images . If the images are square, they can be easily resized to 256 pixels maintaining the aspect ratio. Had the aspect ratio been different from 1:1, it would lead to *the unnecessary cropping or squashing* of the image data causing loss of important data.

Also, each such square image had the file size of `7 - 13` kb .

Image Color Type

Real World objects are in color and **the grayscale conversions of them will conflict with each other leading to 2 objects looking the same**. Thus, the RGB images of the objects was captured and a network that can handle the RGB(*GoogLeNet*) was used to successfully classify them.





Image Collection Method

The smartphone camera was used for the collection of the images because:

1. The same camera would be used for the test of the classification accuracy.
2. Lending to the ubiquitous nature of the smartphone cameras, the trained network **will have better probability of successfully classifying the objects** in a different condition. This is because atleast the testing and training data capture device stayed the same, i.e, smartphone camera.

It should be duly noted that the **environment conditions** in which the classification will need to successfully work **should overlap** with the environment in which the training image would be captured.

For example, if the classification needs to work *in a lighted room with diffused sunlight from the window* then the data acquisition should be done in the *same environment*.









Mouse	Wallet	Nail Cutter	Nothing
			
© Shivang Tripathi	© Shivang Tripathi	© Shivang Tripathi	© Shivang Tripathi

As evident, the images:

1. Have an aspect ratio of 1:1
2. Diffused light that lead to shiny surface

The test conditions will be similar to the above features.

In the supplied data, the images were also square.

Supplied Data			
			
Candy Box	Bottle	Candy Box	Candy Box
			
Candy Box	Candy Box © Shivang Tripathi	Bottle	Bottle

Results

Model for classifying supplied dataset

Used GoogLeNet with SGD with the supplied data that achieved :

1. The accuracy of 75.4098%
2. Inference time of 5.11062 ms using TensorRT

+

Terminal 1

×

Terminal 2

×

Do not run while you are processing data or training a model.

Please enter the Job ID: 20180911-183700-b8ed

Calculating average inference time over 10 samples...

deploy: /opt/DIGITS/digits/jobs/20180911-183700-b8ed/deploy.prototxt

model: /opt/DIGITS/digits/jobs/20180911-183700-b8ed/snapshot_iter_711.caffemodel

output: softmax

iterations: 5

avgRuns: 10

Input "data": 3x224x224

Output "softmax": 3x1x1

name=data, bindingIndex=0, buffers.size()=2

name=softmax, bindingIndex=1, buffers.size()=2

Average over 10 runs is 5.56579 ms.

Average over 10 runs is 5.54944 ms.

Average over 10 runs is 5.62501 ms.

Average over 10 runs is 5.52087 ms.

Average over 10 runs is 5.11062 ms.

Calculating model accuracy...

% Total	% Received	% Xferd	Average Speed	Time	Time	Time	Current
			Dload Upload	Total	Spent	Left	Speed
100 14662	100 12346	100 2316	211 39	0:00:59	0:00:58	0:00:01	3089

Your model accuacy is 75.4098360656 %

© Shivang Tripathi

The accuracy and inference time of the network trained on the supplied data

The caffe model trained on the supplied data can be found [here](#).


The five millisecond inference time is good enough to be deployed for real time classification for use in the robotics workbench.

Candy Box

GoogLeNet Workbench classifier

Image Classification

Model



Predictions

Candy Box

100.0%

Nothing

0.0%

Bottle

0.0%


© Shivang Tripathi

Bottle

GoogLeNet Workbench classifier

Image Classification

Model



Predictions

Bottle

99.86%

Candy Box

0.14%

Nothing

0.0%

© Shivang Tripathi

It should be noted that when training the GoogLeNet with the ADAM, the accuracy dropped to 67.213%.


```
Instructions.txt x
1 Welcome to DIGITS!

+ Terminal 1 x Terminal 2 x Terminal 3 x

Do not run while you are processing data or training a model.

Please enter the Job ID: 20180911-185252-aa03

Calculating average inference time over 10 samples...
deploy: /opt/DIGITS/digits/jobs/20180911-185252-aa03/deploy.prototxt
model: /opt/DIGITS/digits/jobs/20180911-185252-aa03/snapshot_iter_1185.caffemodel
output: softmax
iterations: 5
avgRuns: 10
Input "data": 3x224x224
Output "softmax": 3x1x1
name=data, bindingIndex=0, buffers.size()=2
name=softmax, bindingIndex=1, buffers.size()=2
Average over 10 runs is 5.52408 ms.
Average over 10 runs is 5.51425 ms.
Average over 10 runs is 5.36674 ms.
Average over 10 runs is 5.05502 ms.
Average over 10 runs is 5.04445 ms.

Calculating model accuracy...

  % Total    % Received % Xferd  Average Speed   Time    Time       Time  Current
   100  14839   100  12523   100   2316    212     39  0:00:59  0:00:58  0:00:01  2713

Your model accuracy is 67.2131147541 %
root@6e9b2837904c: /home/workspace#
```

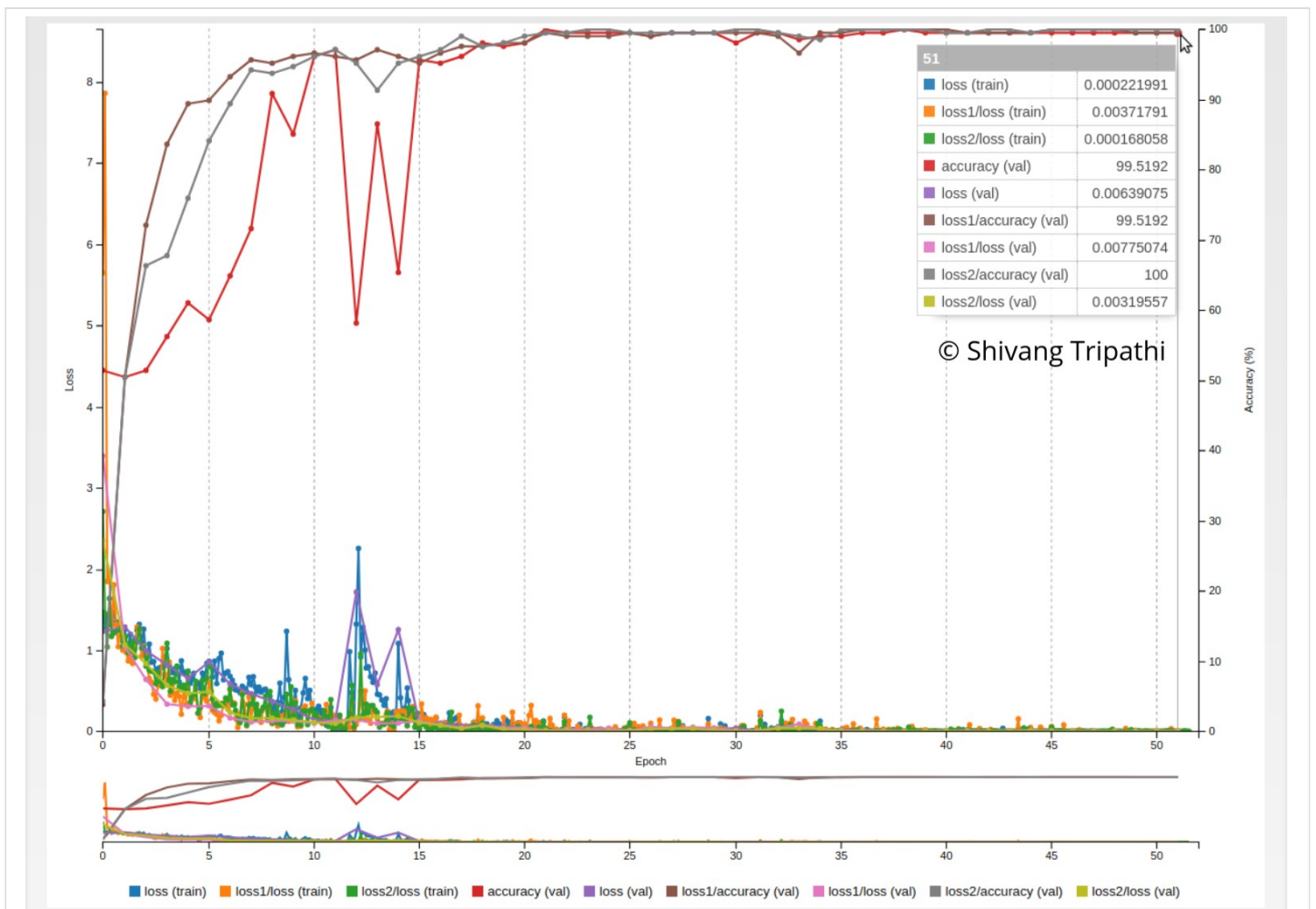
© Shivang Tripathi

The accuracy of the network decreased when trained with ADAM instead of SGD

Model for classifying self acquired dataset

In the custom model, the following important features can be seen as a function of epochs:

- Validation Loss
- Validation Accuracy
- Training Loss
- Training Accuracy



Loss and Accuracy of Train and Validation data with with SGD

The custom model was also tested on separate images the result which the network was able to successfully classify with good accuracy.

Separately Tested on Different Images



Predictions

mouse	99.93%
nailCutter	0.07%
wallet	0.0%
nothing	0.0%

© Shivang Tripathi

Separately Tested on Different Images



Predictions

nailCutter	98.35%
mouse	1.02%
wallet	0.62%
nothing	0.02%

© Shivang Tripathi



Predictions

nothing	99.99%
nailCutter	0.01%
mouse	0.0%
wallet	0.0%

© Shivang Tripathi



Predictions

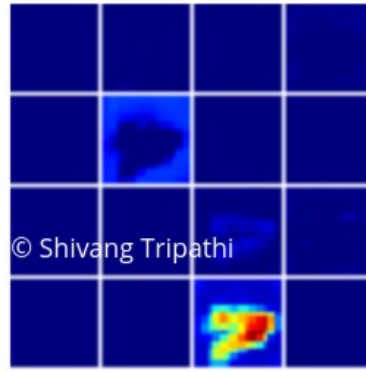
wallet	99.99%
nailCutter	0.01%
mouse	0.0%
nothing	0.0%

© Shivang Tripathi

As evident from the separate tested images, the finetuned GoogLeNet was able to successfully classify the objects in the problem set with good accuracy.

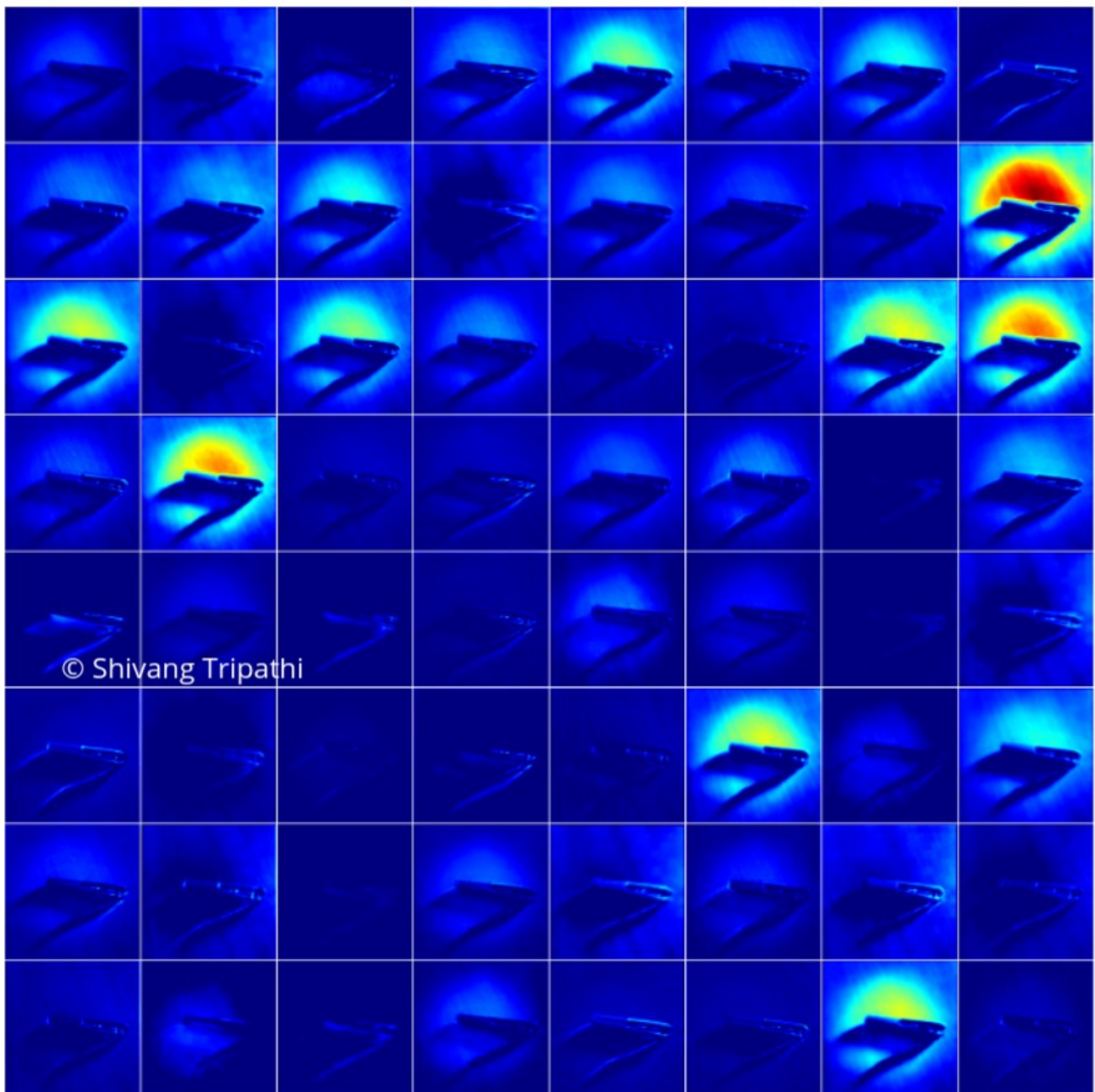
Visualizations

As an advantage of using **Nvidia DIGITS**, the network can be easily visualized. The weights and activations from the pooling layers, convolutional layers, Inception layer can be easily visualized.



Input image of nail cutter


Inception layer(*near final layers*) Output Activations



Output Activation from Convolutional layer near input layer

Discussion

It was interesting to see the classification accuracy improve over the epochs as the network was finetuned by SGD.



Predictions

mouse	88.78%
wallet	10.94%
nothing	0.15%
nailCutter	0.13%

© Shivang Tripathi

performance of a snapshotted earlier epoch less tuned network having a hard time classifying

When training was started with only 25 images in each class to test the effect of less data on the network, the network was able to classify the **Nothing** class but could not differentiate between the wallet and Mouse . This was because the objects wallet and Mouse were almost identical in terms of:

- Color
- Surface shine

This is the reason more data had to be collected for these 2 classes to As it can be seen from the earlier epochs snapshot of the model, there is more problem classifying between wallet and Mouse than the nailCutter . As the epochs increased, this problem was minimized as the network got better at classification by learning the underlying patterns in the image.

Accuracy vs Inference Time

As the accuracy increase,

1. The number parameters required increases leading to a bigger sized network.
2. More parameters mean more time to train the network.
3. Bigger network will be difficult to deploy on the embedded system which comes with less compute power.

But the higher accuracy means better classification. This can be a dependable and reliable solution solution that can handles wider spectrum of data.

To reduce the inference time, the network needs to be smaller, i.e, less number of network parameters.

1. This narrows the learning capability of the network.

Therefore, the trade-off between accuracy and inference time needs to be made. This should be highly dependent on the task at hand.

If a general solution is needed that can infer for the wide range of data inputs with the network running with good compute power on a work station, accuracy can be increased.

If a very task specific solution is needed, that needs to work fast on the embedded system with limited compute power in real time, the inference time should be reduced to be in order of milliseconds.

Conclusion

The project good results in the supplied as well as the custom self acquired data. This was essentially a result of:

- Quality data collection of the objects.
- Smooth and fast development of ideas using the Nvidia DIGITS workflow.
- And most importantly, **transfer learning** which enabled one with limited data and compute power to build a good classification network.

This inference model can be deployed on a small robotic arm that can manipulate everyday objects on the table. The problems with this approach is that:

1. Since no depth information is available, it will be difficult to manipulate the table objects as the trajectory calculations depend on the final pose to be reached.
2. Finer manipulations will require the computation the grasping pose that is still a open area of research.

Future Work

In the future, rather than using a classification network, a detection network can be used to output the object coordinates in the image. Although this would require each image in the dataset to be annotated with a bounding box.

Also, it will be interesting to see the performance of such networks with the point cloud data to LIDARs, stereographic and RGB-D camera.