Deep Learning and Drone Imagery

INI Project Practicum, Fall 2016

Team Ericsson Drone

Point of Contact
Per-Erik Brodin
Alvin Jude Hari Haran

Advisor
Jia Zhang

Students
Aditya Gautam
Raquel Rodriguez
Haoyang Yuan

Agenda

- Introduction
  - Motivation
  - Related Work
- System Design
  - Work Routine
  - Collecting Images
  - Model
- Implementation
- Demo
- Experiments/analysis
- Conclusions and future work

INTRODUCTION

The Big Picture - Motivation

How to monitor road conditions in real time?

Related Solutions

- Community driven information
- Human inspection
- Volunteer programs (California)

Proposed Solution

Use drones to monitor road conditions

Our Scope:

Use road images, that are publicly available, to train a deep learning model and test it against images that come from a drone.

Road Inspection via Drone

A service that can monitor road condition in an effective and smart way

Analysis and Model Learning

Project Goals

- **Prove** that publicly available images can be used to train Deep Learning for road inspection purpose
- **Deliver** the clean and well-tagged dataset
- **Train** a model that can classify and evaluate based on road images
- **Test** the model with real images gathered by the team

Agenda

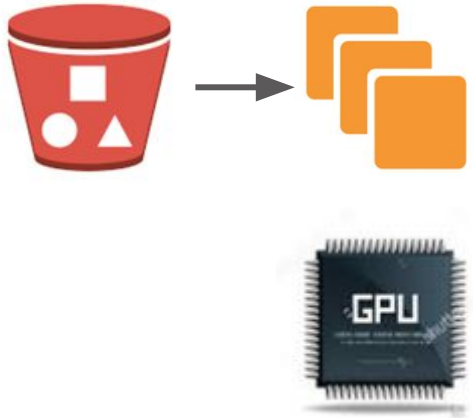
- Introduction
  - Motivation
  - Related Work
- System Design
  - Work Routine
  - Collecting Images
  - Model
- Implementation
- Demo
- Experiments/analysis
- Conclusions and future work

SYSTEM DESIGN

High Level Architecture:

Google Map Images
Openstreet Map API

GPU Supported Instances



Tensorflow Framework

Agenda

- Introduction
  - Motivation
  - Related Work
- System Design
  - Work Routine
  - Collecting Images
  - Model
- Implementation
- Demo
- Experiments/analysis
- Conclusions and future work

Collecting the Images

- Using OSM API to get road locations
- Connect Google Map API for images from locations
- Change the vertical and horizontal scope to get view
- Clean and label the images

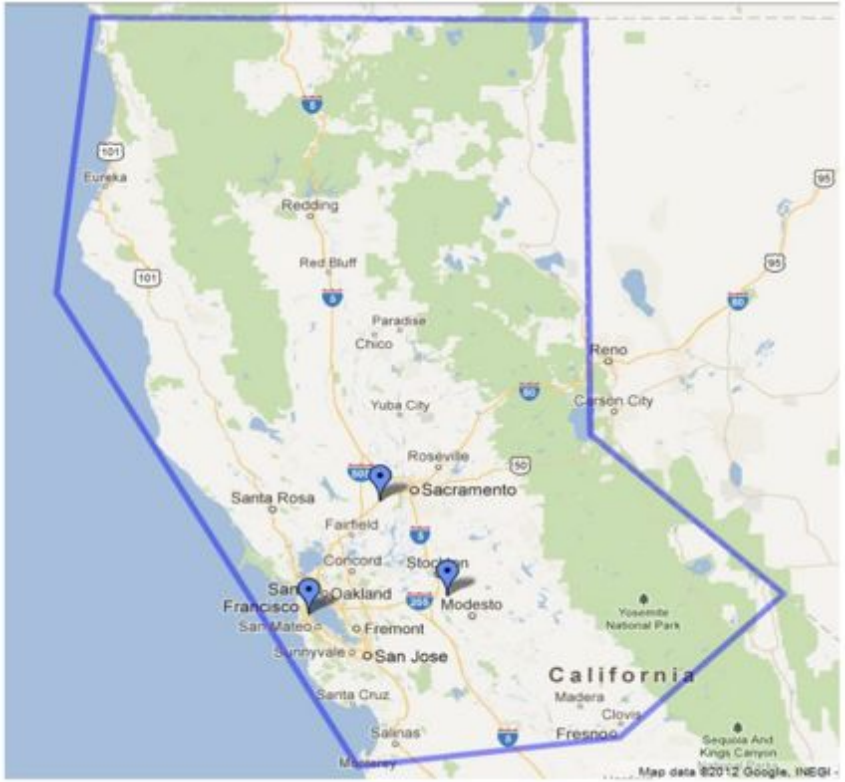
Images - Gather Locations

- Using “highway” to search the road we want from OSM API
- Use tag “motorway” to find the tagged locations

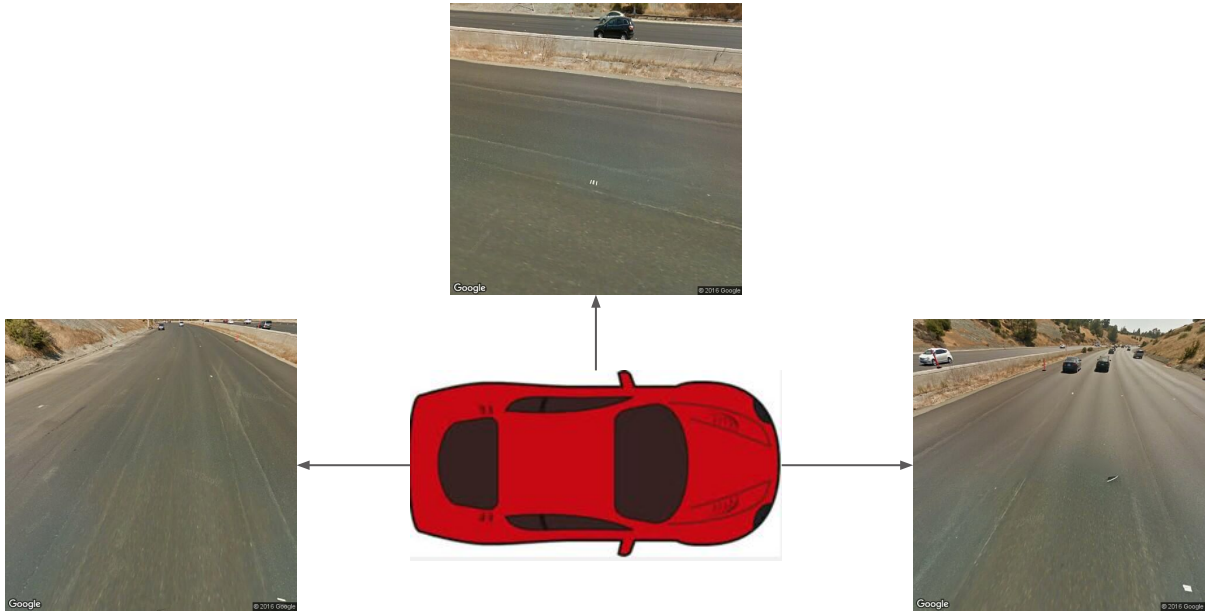
highway	motorway		A restricted access major divided highway, normally with 2 or more running lanes plus emergency hard shoulder. Equivalent to the Freeway, Autobahn, etc..		
highway	trunk		The most important roads in a country's system that aren't motorways. (Need not necessarily be a divided highway.)		
highway	primary		The next most important roads in a country's system. (Often link larger towns.)		
highway	secondary		The next most important roads in a country's system. (Often link towns.)		
highway	tertiary		The next most important roads in a country's system. (Often link smaller towns and villages)		

Images - Gather Locations

- Using Overpass API from OSM
- Download a set of coordinates among the set of **bounding box**



Images - Vertical Horizontal Scope



Images - Illumination Condition

Images - Image Noise

Images - Domain Knowledge

Images - Solution

- Similar Environment
  - close number of images from same cities
- Verification by team members

Images - Data Set

Boolean tagged Image data set

Batch	Cities	Number
1	Oklahoma City, Milwaukee, Bridgeport	~200 each category
2	Sacramento, Oklahoma City, Springfield, Boston	~800 each category
3	San diego (part 1)	~500 each category
4	San diego (part 2)	~500 each category

Agenda

- Introduction
  - Motivation
  - Related Work
- System Design
  - Work Routine
  - Collecting Images
  - Model
- Implementation
- Demo
- Experiments/analysis
- Conclusions and future work

Model - Our Aim:

To Learn cracks and potholes in the images

Model - How to Achieve it?

Problem is too complex for standard machine Learning algorithm. Highly nonlinear in nature.

Feature engineering is extremely difficult.

Solution ?

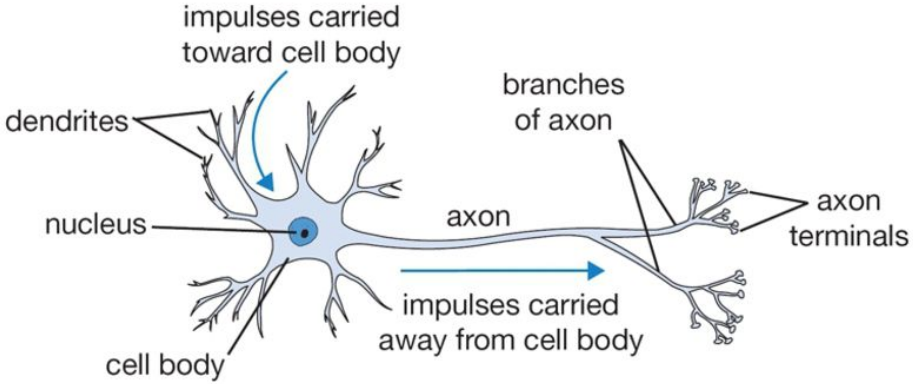
Deep Learning

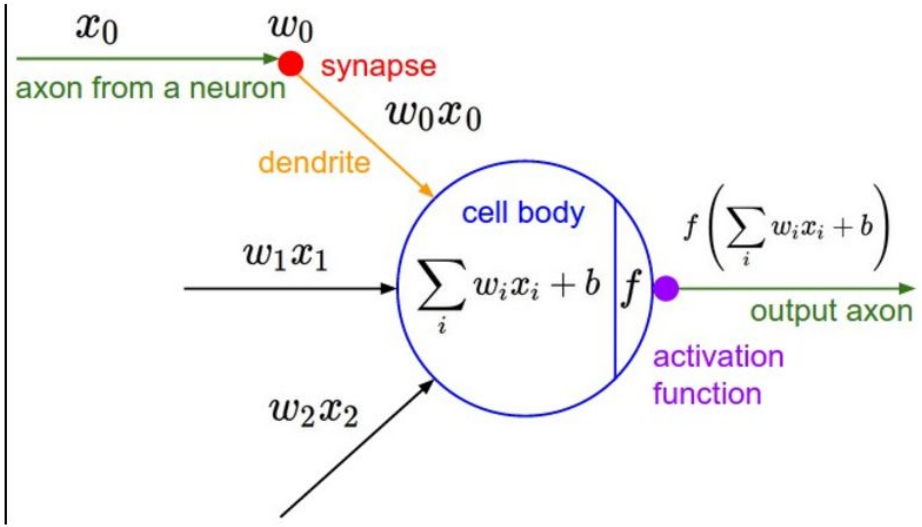
Model - Deep Learning

Traditional Machine Learning Flow

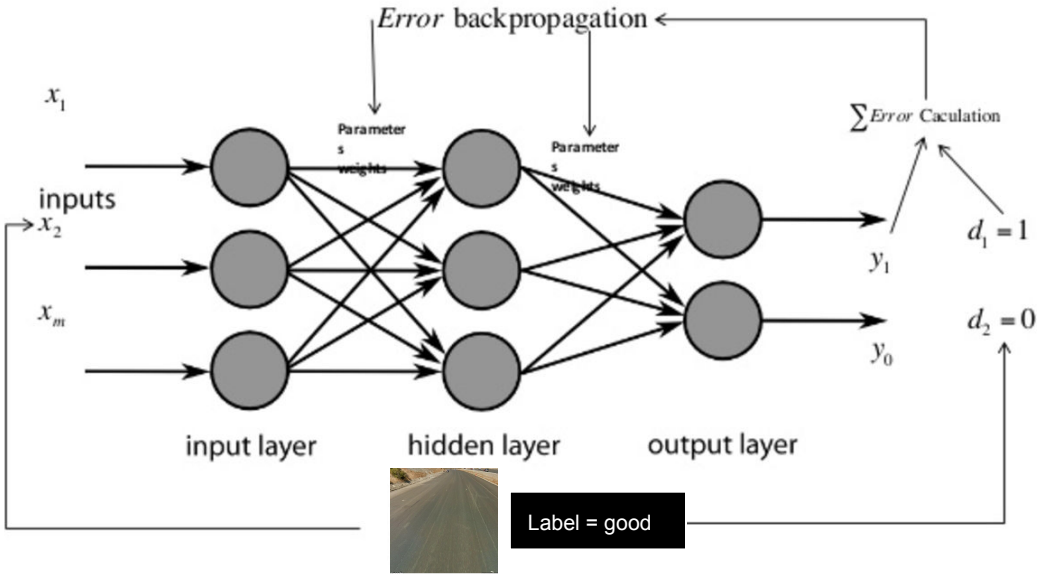
Deep Learning Flow

Model - Inspiration





Model - A Simple Neural Network



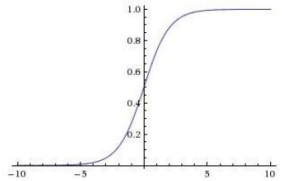
Source : http://www.slideshare.net/JunWang5/deep-learning-61493694

Model - Activation Functions

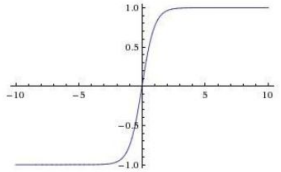
Activation Functions

Sigmoid

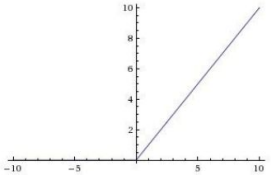
$$\sigma(x) = 1/(1 + e^{-x})$$



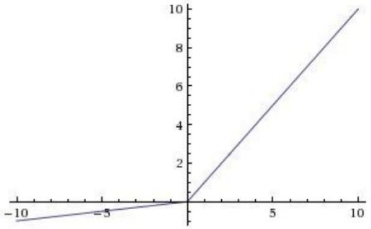
tanh tanh(x)



ReLU max(0,x)



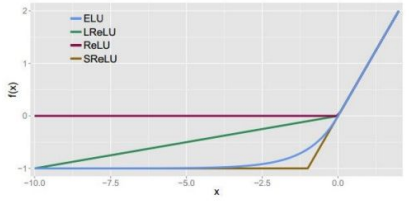
Leaky ReLU
max(0.1x, x)



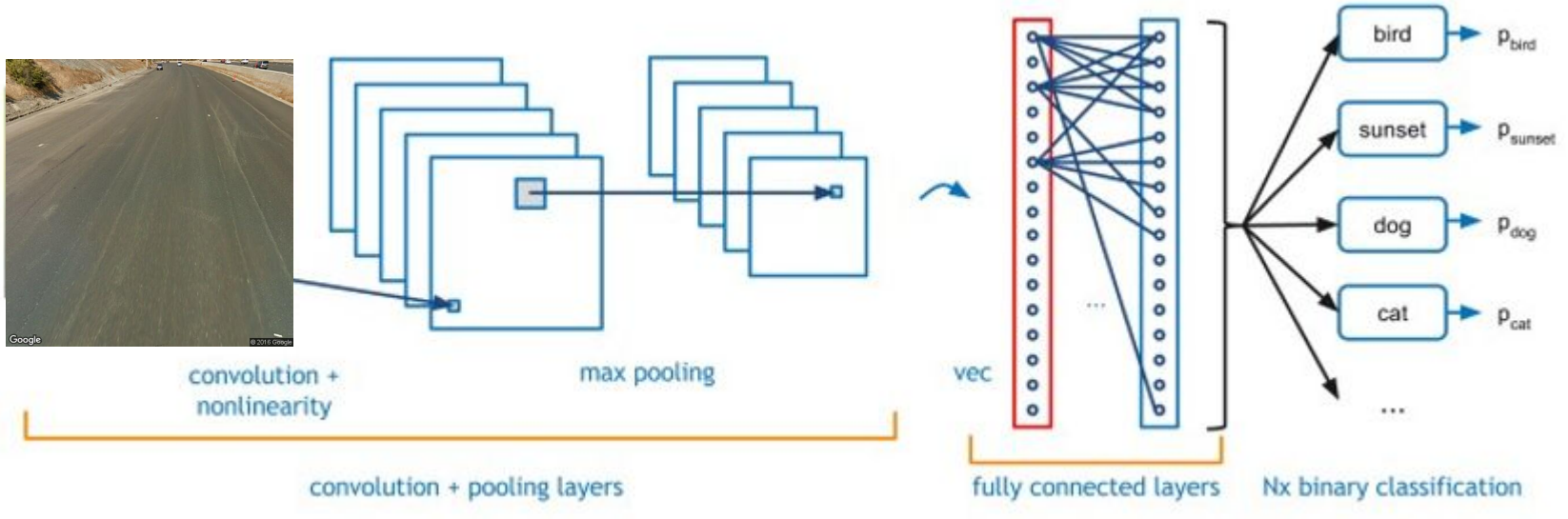
Maxout $\max(w_1^T x + b_1, w_2^T x + b_2)$

ELU

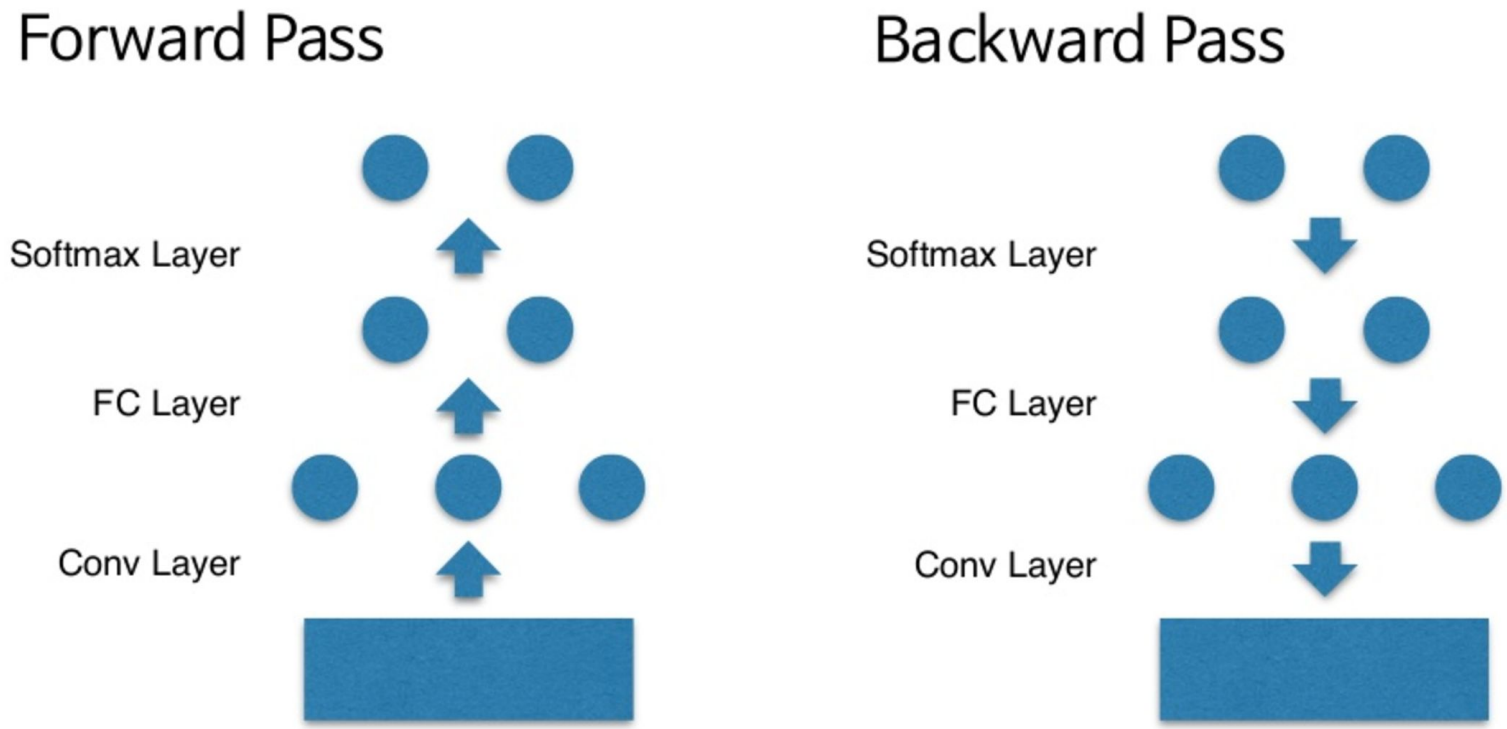
$$f(x) = \begin{cases} x & \text{if } x > 0 \\ \alpha\,(\exp(x) - 1) & \text{if } x \leq 0 \end{cases}$$



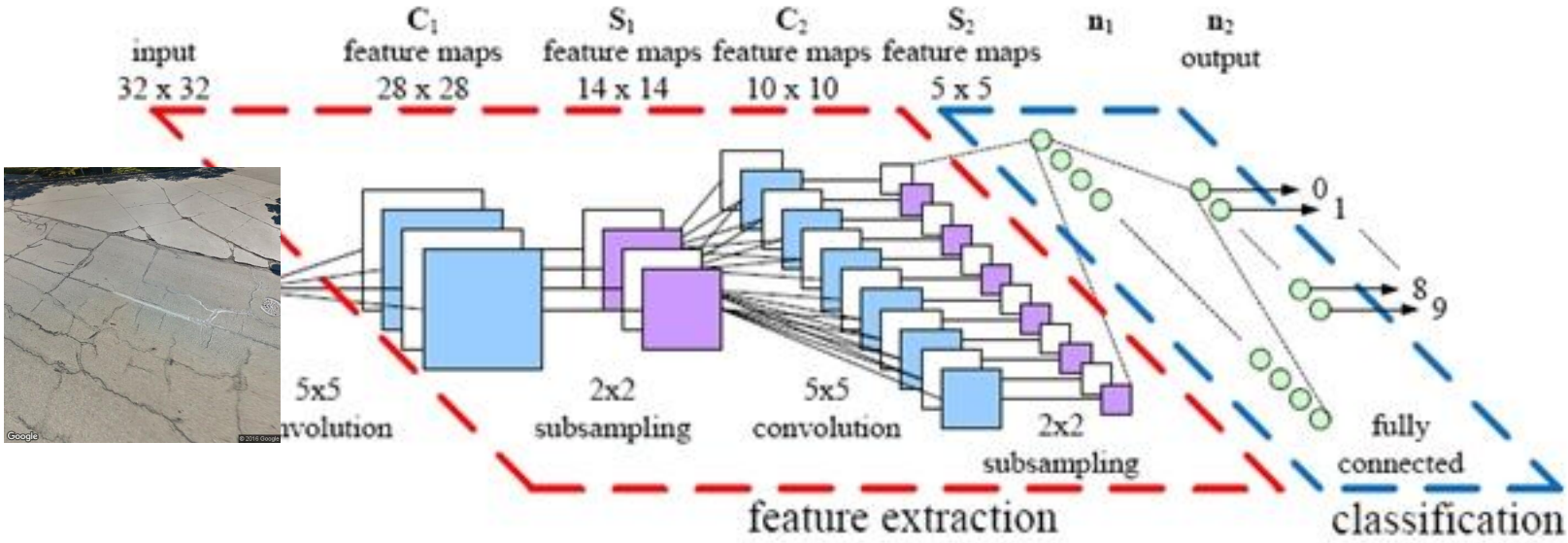
Model - Convolution Neural Network



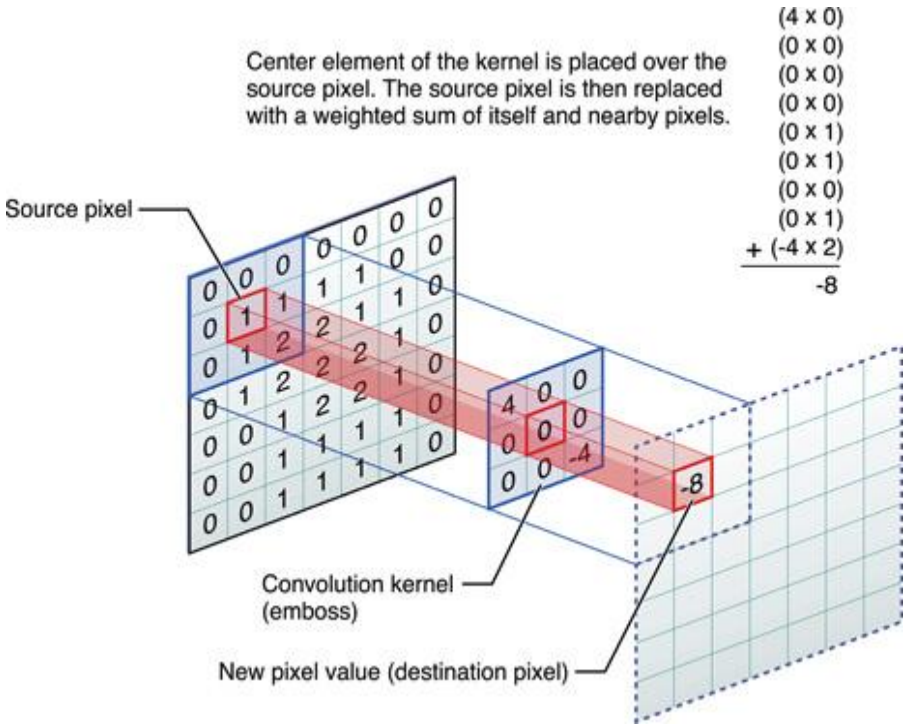
Model - Forward & Backward



Model - General CNN Architecture



Model - Convolution(Hidden Layer 1)

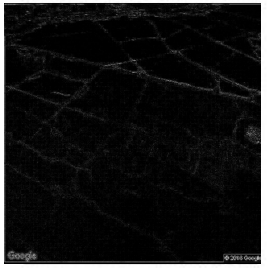


Input image

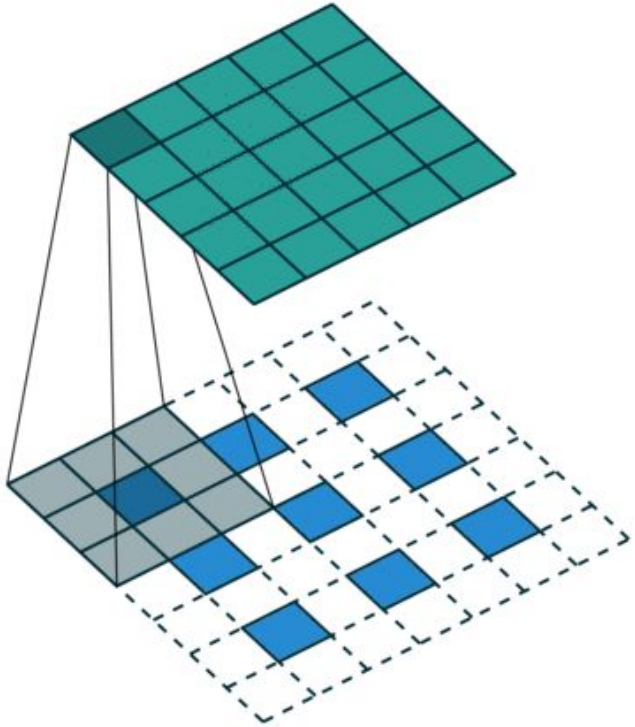
Convolution Kernel

$$\begin{bmatrix} -1 & -1 & -1 \\ -1 & 8 & -1 \\ -1 & -1 & -1 \end{bmatrix}$$

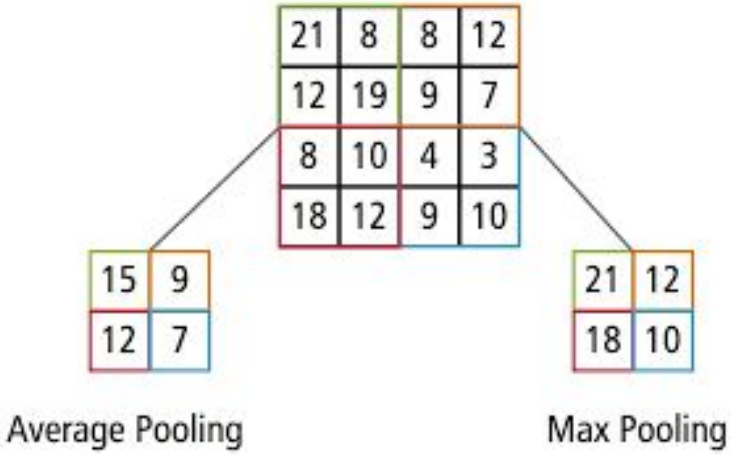
Feature map



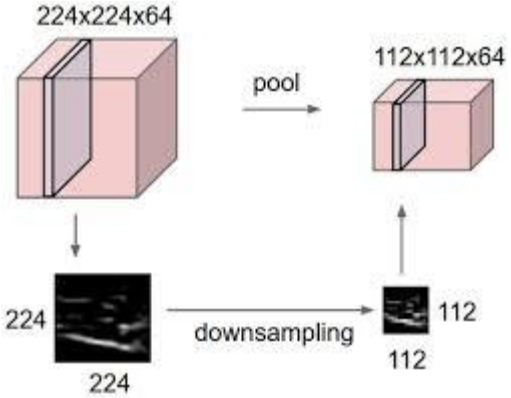
Model - Convolution(Hidden Layer 1)



Model - Pooling (Hidden Layer 2)

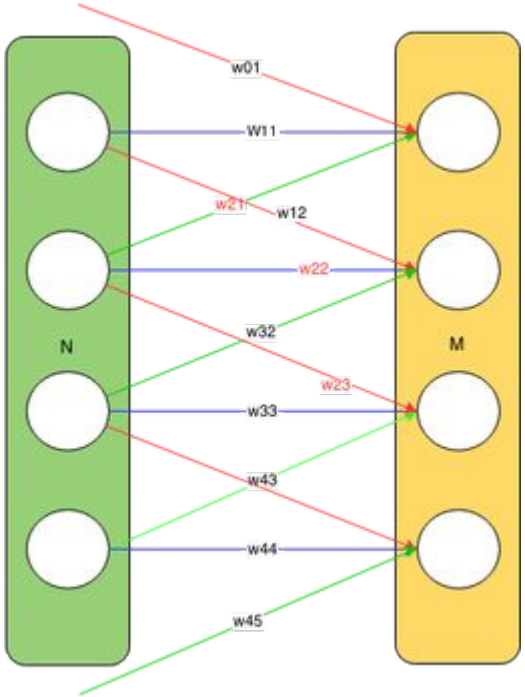


Maths



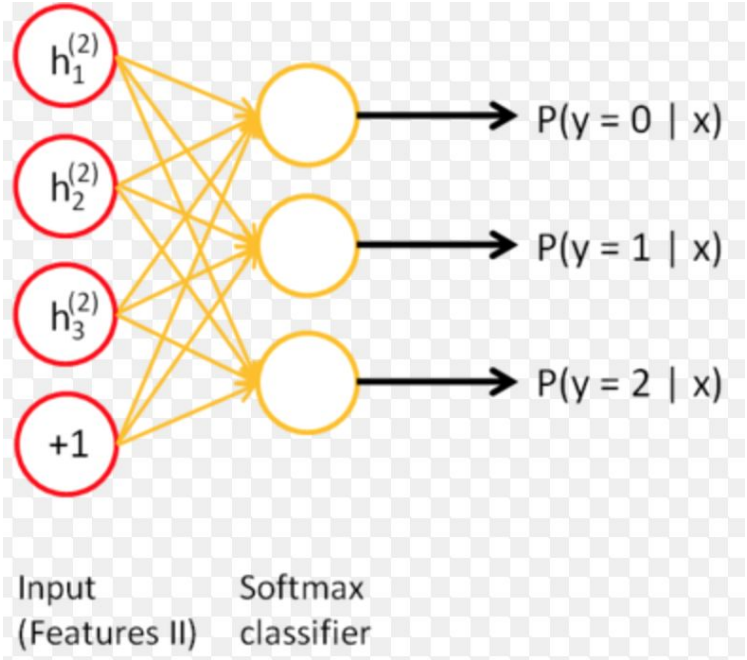
Result

Model - Fully Connected Layer



- Input from last Max pool layer.
- Do Multiplication and feed to softmax.
- Introduces non-linear in the network.
- Tells the softmax layer which features/neurons are best for which class

Model - Softmax Layer



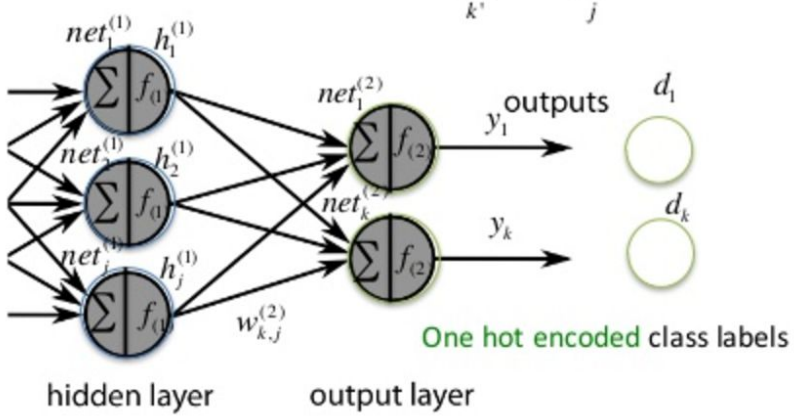
- Sum up the outputs of fully connected layer with multiplied weights.
- Normalize it to give probability of each class.
- Sum of all the output is 1.
- Calculate the loss function to be propagated backwards.

$$Y_i = \frac{e^{w_i^T h}}{\sum_i^k e^{w_i^T h}}$$

Model - Loss function

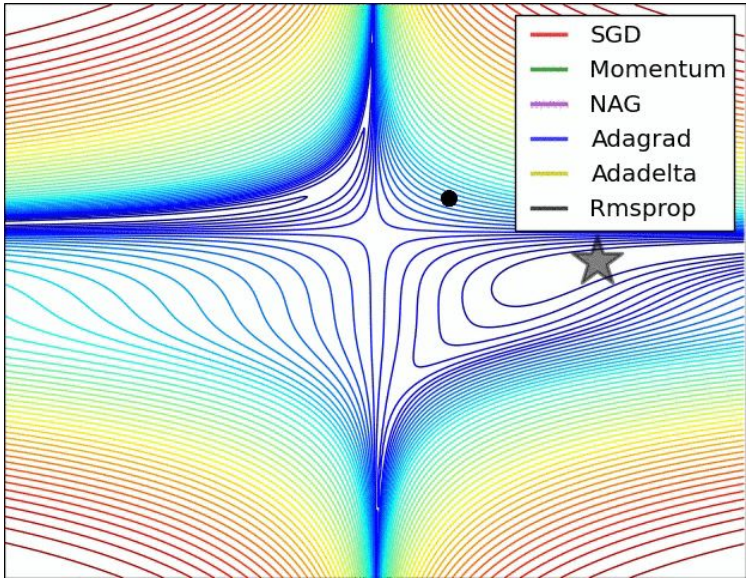
Cross entropy loss is calculated at softmax layer.

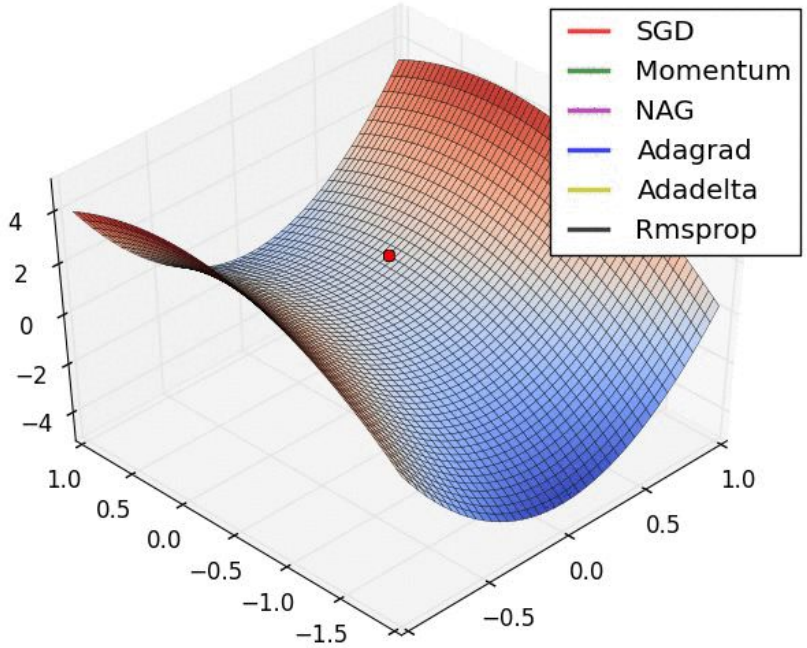
$$E(w) = -\sum_k (d_k \log y_k + (1 - d_k)\log(1 - y_k)) \quad \text{where} \quad y_k = \frac{\exp(\sum_j w_{k,j}^{(2)} h_j^{(1)})}{\sum_{k'} \exp(\sum_j w_{k',j}^{(2)} h_j^{(1)})}$$



Source : http://www.slideshare.net/JunWang5/deep-learning-61493694

Model - Optimization

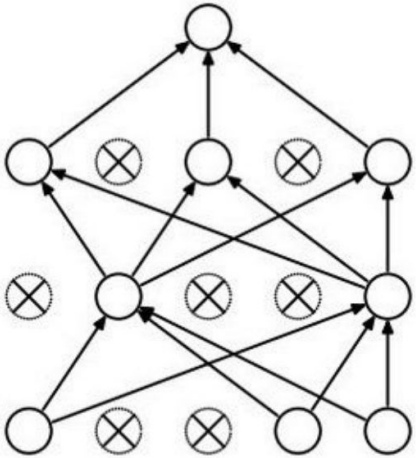




Source : http://sebastianruder.com/optimizing-gradient-descent/

Model - Dropouts/Regularization :

Dropout



Forces the network to have a redundant representation.

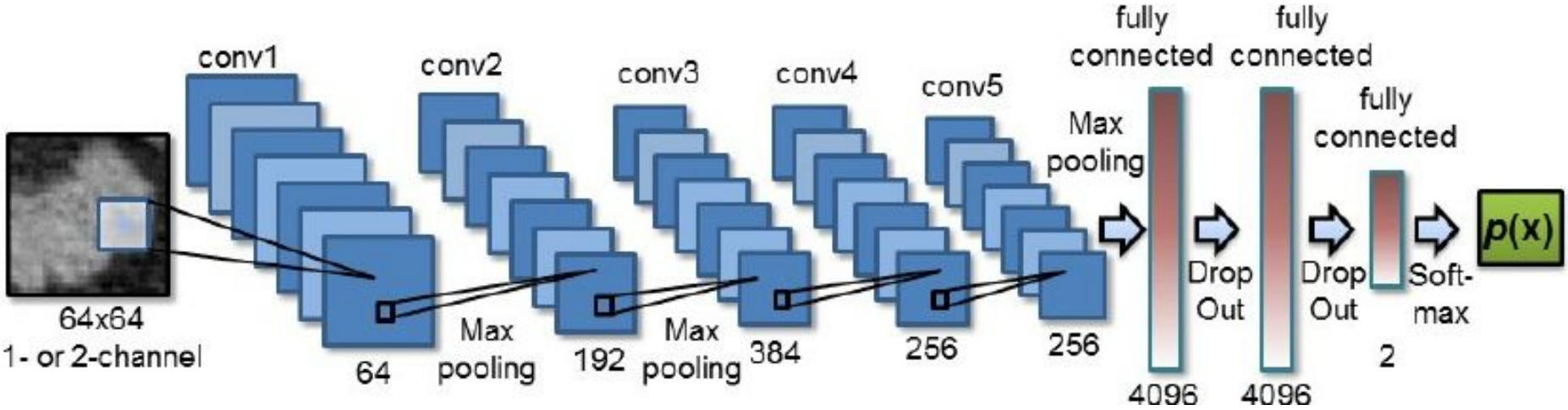
Model - Standard CNN classification

90% features are “different” as compared to 90% similar features in our case.
Cat vs Dog -> Easy problem **Good vs Bad Road -> Hard problem**

Model - What is CNN looking for ?

Cracks and potholes
(<10% of the features)

Model - Architecture 13 Layers CNN



Input size :
500*500*3

Filter Size :

Model - Regularization :

Adding images which are little off will generalize the errors and cause regularization by itself. Like adding images with shadows, house, cars etc. improves the testing accuracy.

Houses/Street side

Cars/trucks

Tree shows/concrete

Areas researched on :

- Number of hidden layers.
- Filter size, stride and other convolution related parameters
- Number of neurons at fully connected layer
- Loss function -> Softmax, 0/1 loss, logistic etc.
- Activation units -> tanh, Relu, sigmoid
- Batch size
- Learning rate
- Number of iterations (epoch)

Rationale :

To extracts the crack and focus on them for feature extraction as it looks like highly non linear problem i.e. extract potholes and image cracks on the roads as the key features and discard all the features.

- **Input image** : 500*500*3
- **10 Layers of Convolution and pooling :** Discarding all features and focusing on the key on needs more hidden layer.
- **Convolutional filter** of size 5*5 for better extraction of the cracks on the roads.
- **Pooling :** Max pooling
- **Fully connected layer :** One layer is doing a decent job.

Agenda

- Introduction
  - Motivation
  - Related Work
- System Design
  - Work Routine
  - Collecting Images
  - Model
- Implementation
- Demo
- Experiments/analysis
- Conclusions and future work

IMPLEMENTATION

Environment Setup

- Powerful Computing Resources - Cloud Service
- Google Cloud is free originally - working well for sample
- AWS GPU can support thousands of images
- Conclusion: **Using GPU from AWS**

Environment Setup - EC2 Setup

Setting up the instance:

Get Anaconda for Ubuntu

wget http://repo.continuum.io/archive/Anaconda2-4.2.0-Linux-x86_64.sh

Install Anaconda

bash Anaconda2-4.2.0-Linux-x86_64.sh

Install PIP

sudo apt-get install python-pip python-dev

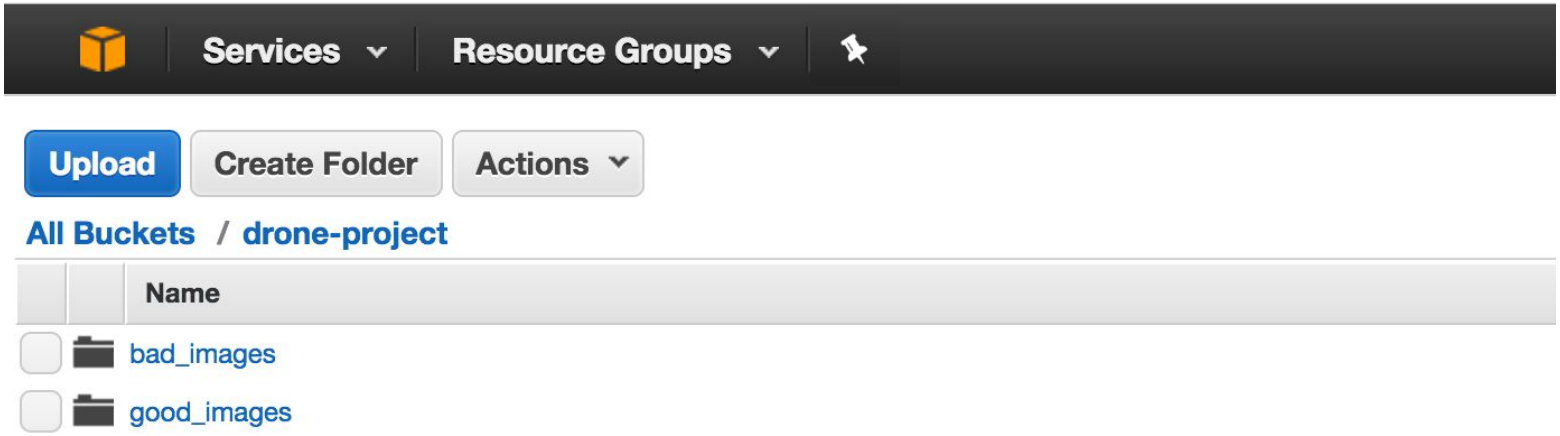
Install TENSORFLOW:

conda create -n tensorflow python=2.7

source activate tensorflow

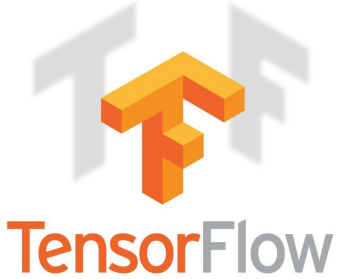
conda install -c conda-forge tensorflow

Environment Setup - AWS S3



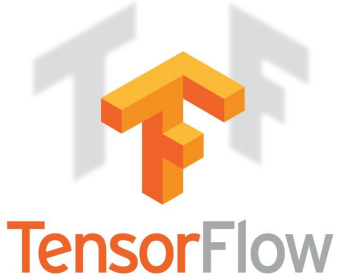
Deep Learning Frameworks

Tensor Flow



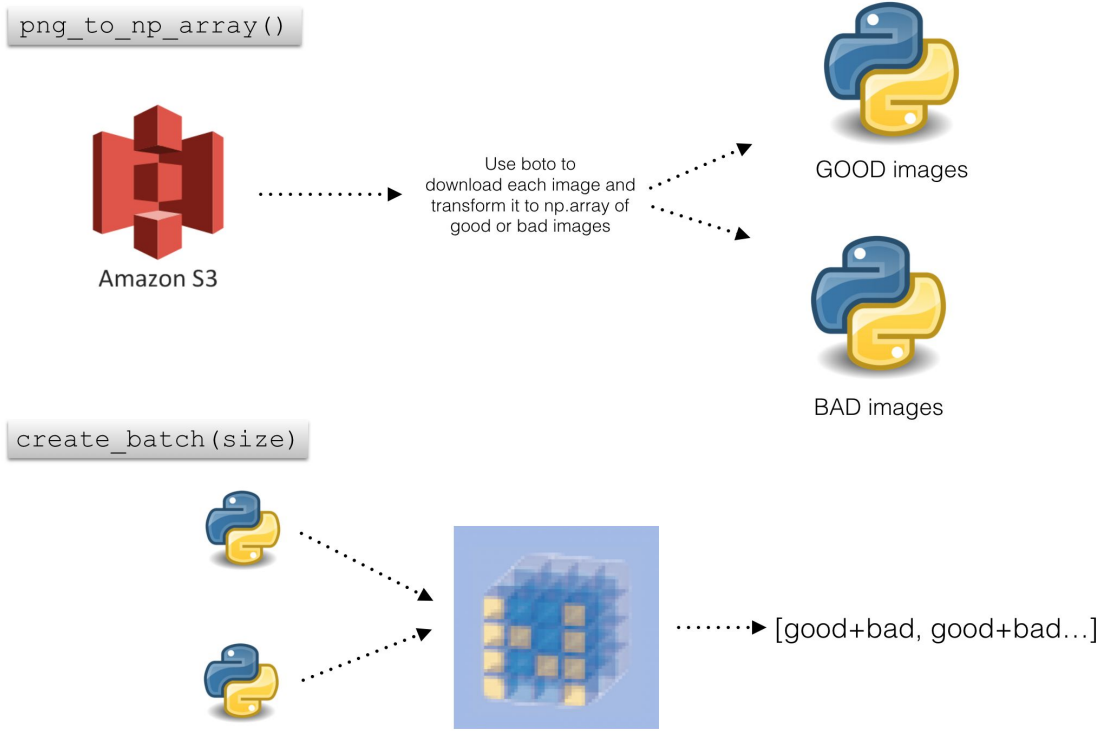
- Flexible Architecture
- Big Community - Github - Stack Overflow
- Easily supports GPU and CPU
- Easy to install and start implementing
- Good documentation and tutorials

Tensor Flow



- Represents computations as graphs
- Nodes in the graph are called ops (short for operations)
- Ops take zero or more Tensors, perform computations, and produce zero or more Tensors
- Tensors are typed-multidimensional arrays
- Programs are structured into a construction phase (assembles graph) and execution phase that uses a session to execute ops in the graph (training)

Batching of the Images



Model Implementation - Tensor Flow

Model creation and variable declaration

```
# Convolution Layer
conv1 = conv2d(x, weights['wc1'], biases['bc1'])
print("Conv_1 : ", conv1)
# Max Pooling (down-sampling)
conv1 = maxpool2d(conv1, k=2)
print("MaxPool_1 : ", conv1)
```

Define loss and optimizer + model evaluation

```
# Define loss and optimizer
cost = tf.reduce_mean(tf.nn.softmax_cross_entropy_with_logits(pred, y))
optimizer = tf.train.AdamOptimizer(learning_rate=learning_rate).minimize(cost)

# Evaluate model
correct_pred = tf.equal(tf.argmax(pred, 1), tf.argmax(y, 1))
accuracy = tf.reduce_mean(tf.cast(correct_pred, tf.float32))
```

Model Implementation - Tensor Flow

Session initialization and graph creation

```
# Launch the graph
with tf.Session() as sess:
    sess.run(init)
    step = 1
    result = create_batch(batch_size)  # this returns [[batch imgs], [batch labls], [batch imgs], [batch lbls]..

    # Keep training until reach max iterations
    while step < training_iters:
        for i in xrange(0, len(result) - 4, 2):
            batch_x = np.array(result[i])
            batch_y = np.array(result[i + 1])
            sess.run(optimizer, feed_dict={x: batch_x, y: batch_y})
            if step % 1 == 0:
                # Calculate batch loss and accuracy
                loss, acc = sess.run([cost, accuracy], feed_dict={x: batch_x, y: batch_y})
```

Agenda

- Introduction
  - Motivation
  - Related Work
- System Design
  - Work Routine
  - Collecting Images
  - Model
- Implementation
- Demo
- Experiments/analysis
- Conclusions and future work

DEMO

EXPERIMENT RESULTS

Accuracy Chart

Number of Images	Model Layers	Training Accuracy	Test Accuracy	Dropout
1000	5 (2 Con, 2 Pool, 1 fcc)	Random toss (50%)	Random toss	0%
2000	9 (4 Con, 4 Pool, 1 fcc)	76%	24 %	0%
4000	9 (4 Con, 4 Pool, 1 fcc)	98%	85%	0%
4000	9 (4 Con, 4 Pool, 1 fcc)	92%	**90%**	25%

Result Snippets 1

```
Iter 600, Minibatch Loss= 886388.187500, Training Accuracy= 0.74000
Iter 600, Minibatch Loss= 505352.000000, Training Accuracy= 0.81000
Iter 600, Minibatch Loss= 529201.375000, Training Accuracy= 0.78000
('step : ', 7)
Iter 700, Minibatch Loss= 700031.250000, Training Accuracy= 0.69000
Iter 700, Minibatch Loss= 1137422.375000, Training Accuracy= 0.62000
Iter 700, Minibatch Loss= 1130210.250000, Training Accuracy= 0.63000
Iter 700, Minibatch Loss= 539598.750000, Training Accuracy= 0.79000
Iter 700, Minibatch Loss= 1059106.625000, Training Accuracy= 0.58000
Iter 700, Minibatch Loss= 1422089.625000, Training Accuracy= 0.61000
Iter 700, Minibatch Loss= 582714.500000, Training Accuracy= 0.72000
Iter 700, Minibatch Loss= 512183.812500, Training Accuracy= 0.72000
Iter 700, Minibatch Loss= 320337.031250, Training Accuracy= 0.83000
Iter 700, Minibatch Loss= 705200.937500, Training Accuracy= 0.70000
Iter 700, Minibatch Loss= 748577.250000, Training Accuracy= 0.76000
Iter 700, Minibatch Loss= 326149.750000, Training Accuracy= 0.83000
Iter 700, Minibatch Loss= 202333.093750, Training Accuracy= 0.87000
Iter 700, Minibatch Loss= 226933.703125, Training Accuracy= 0.82000
Iter 700, Minibatch Loss= 613700.375000, Training Accuracy= 0.75000
('step : ', 8)
Iter 800, Minibatch Loss= 1072480.750000, Training Accuracy= 0.58000
Iter 800, Minibatch Loss= 614614.562500, Training Accuracy= 0.62000
Iter 800, Minibatch Loss= 577106.000000, Training Accuracy= 0.74000
Iter 800, Minibatch Loss= 1068604.000000, Training Accuracy= 0.66000
Iter 800, Minibatch Loss= 827507.375000, Training Accuracy= 0.65000
Iter 800, Minibatch Loss= 445344.562500, Training Accuracy= 0.76000
Iter 800, Minibatch Loss= 400837.843750, Training Accuracy= 0.76000
Iter 800, Minibatch Loss= 894869.625000, Training Accuracy= 0.61000
Iter 800, Minibatch Loss= 410922.625000, Training Accuracy= 0.78000
Iter 800, Minibatch Loss= 391415.281250, Training Accuracy= 0.76000
Iter 800, Minibatch Loss= 554107.437500, Training Accuracy= 0.79000
Iter 800, Minibatch Loss= 770382.875000, Training Accuracy= 0.69000
Iter 800, Minibatch Loss= 1188646.750000, Training Accuracy= 0.58000
Iter 800, Minibatch Loss= 852730.562500, Training Accuracy= 0.72000
Iter 800, Minibatch Loss= 490326.125000, Training Accuracy= 0.79000
('step : ', 9)
Iter 900, Minibatch Loss= 1008619.625000, Training Accuracy= 0.60000
Iter 900, Minibatch Loss= 1669628.500000, Training Accuracy= 0.52000
Iter 900, Minibatch Loss= 972983.625000, Training Accuracy= 0.67000
Iter 900, Minibatch Loss= 446808.031250, Training Accuracy= 0.73000
Iter 900, Minibatch Loss= 573520.812500, Training Accuracy= 0.65000
Iter 900, Minibatch Loss= 649114.812500, Training Accuracy= 0.70000
Iter 900, Minibatch Loss= 517182.125000, Training Accuracy= 0.80000
Iter 900, Minibatch Loss= 439234.812500, Training Accuracy= 0.80000
Iter 900, Minibatch Loss= 143228.281250, Training Accuracy= 0.88000
Iter 900, Minibatch Loss= 483415.000000, Training Accuracy= 0.72000
Iter 900, Minibatch Loss= 166460.218750, Training Accuracy= 0.82000
Iter 900, Minibatch Loss= 51386.378906, Training Accuracy= 0.95000
Iter 900, Minibatch Loss= 427765.125000, Training Accuracy= 0.68000
Iter 900, Minibatch Loss= 654111.187500, Training Accuracy= 0.74000
Iter 900, Minibatch Loss= 520737.843750, Training Accuracy= 0.74000
('step : ', 10)
Optimization Finished!
('Final Testing Accuracy:', 0.23999999)
```

Num Layers	13
Num Images	1000
Num Iterations	10
Dropout	0%
Training acc	~76%
Testing acc	24%

Result Snippets (Overfitting)

```
('epoch : ', 19)
Num iter = 100, Minibatch Loss= 20962.164062, Training Accuracy= 0.97000
Num iter = 500, Minibatch Loss= 0.000000, Training Accuracy= 1.00000
Num iter = 900, Minibatch Loss= 10528.570312, Training Accuracy= 0.98000
Num iter = 1300, Minibatch Loss= 42565.898438, Training Accuracy= 0.98000
Num iter = 1700, Minibatch Loss= 10790.144531, Training Accuracy= 0.99000
Num iter = 2100, Minibatch Loss= 9463.127930, Training Accuracy= 0.98000
Num iter = 2500, Minibatch Loss= 28591.904297, Training Accuracy= 0.99000
Num iter = 2900, Minibatch Loss= 0.000000, Training Accuracy= 1.00000
Num iter = 3300, Minibatch Loss= 10340.445312, Training Accuracy= 0.99000
Num iter = 3700, Minibatch Loss= 9410.150391, Training Accuracy= 0.99000
Num iter = 4100, Minibatch Loss= 109809.109375, Training Accuracy= 0.99000
Num iter = 4500, Minibatch Loss= 0.000000, Training Accuracy= 1.00000
Num iter = 4900, Minibatch Loss= 2309.745117, Training Accuracy= 0.98000
Num iter = 5300, Minibatch Loss= 5068.832520, Training Accuracy= 0.98000
Num iter = 5700, Minibatch Loss= 69293.382812, Training Accuracy= 0.98000
Num iter = 6100, Minibatch Loss= 0.000000, Training Accuracy= 1.00000
Num iter = 6500, Minibatch Loss= 7161.613281, Training Accuracy= 0.99000
Num iter = 6900, Minibatch Loss= 0.000000, Training Accuracy= 1.00000
Optimization Finished!
CNN Features : c1_do85_ss20_bs100_lr005_cpu
('Final Training Accuracy(3600 Images):', 0.98166667587227296)
('Final Testing Accuracy(300 Images):', 0.83599998950958254)
```

Num Layers	**13**
Num Images	**4000**
Num Iterations	**20**
Dropout	**0%**
Training acc	**98%**
Testing acc	**83.5%**

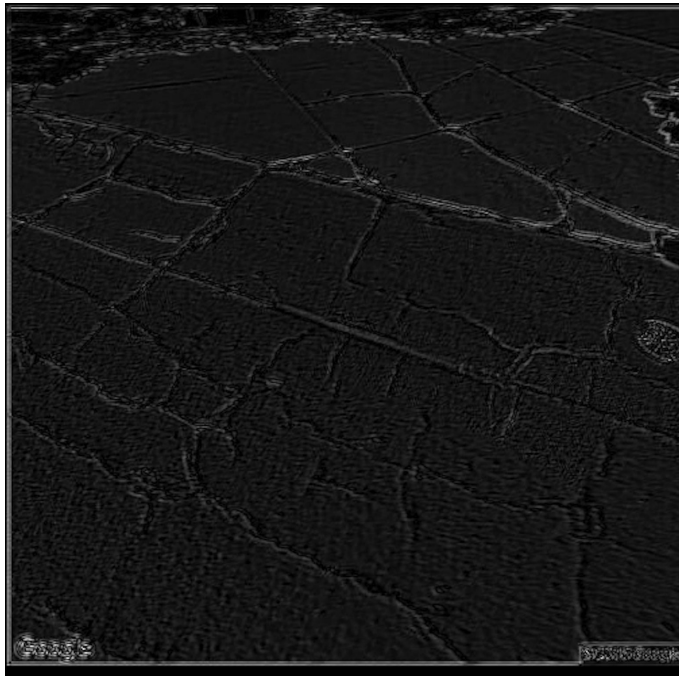
Result Snippets (With dropout)

```
Num iter = 4900, Minibatch Loss= 1437531.625000, Training Accuracy= 0.93000
Num iter = 5300, Minibatch Loss= 182129.406250, Training Accuracy= 0.94000
('epoch : ', 14)
Num iter = 100, Minibatch Loss= 1550717.500000, Training Accuracy= 0.95000
Num iter = 500, Minibatch Loss= 834193.250000, Training Accuracy= 0.95000
Num iter = 900, Minibatch Loss= 1036245.437500, Training Accuracy= 0.95000
Num iter = 1300, Minibatch Loss= 2379555.000000, Training Accuracy= 0.90000
Num iter = 1700, Minibatch Loss= 841424.000000, Training Accuracy= 0.95000
Num iter = 2100, Minibatch Loss= 1168264.500000, Training Accuracy= 0.92000
Num iter = 2500, Minibatch Loss= 1216655.375000, Training Accuracy= 0.94000
Num iter = 2900, Minibatch Loss= 1776212.750000, Training Accuracy= 0.94000
Num iter = 3300, Minibatch Loss= 375111.625000, Training Accuracy= 0.97000
Num iter = 3700, Minibatch Loss= 504157.125000, Training Accuracy= 0.96000
Num iter = 4100, Minibatch Loss= 1318682.000000, Training Accuracy= 0.94000
Num iter = 4500, Minibatch Loss= 719490.875000, Training Accuracy= 0.95000
Num iter = 4900, Minibatch Loss= 462484.812500, Training Accuracy= 0.95000
Num iter = 5300, Minibatch Loss= 353444.843750, Training Accuracy= 0.96000
Optimization Finished!
CNN Features : c1_do75_ss15_bs100_lr001_cpu result-6 shuffling/random
('Final Training Accuracy:', 0.92037037346098161)
('test_image_size = ', (50, 750000))
('test label size = ', (50, 2))
('Final Testing Accuracy:', 0.86000001)
('Final Testing Accuracy(2nd batch):', 0.89999998)
[[-1.59224701 -0.60946107  0.87576842  0.7827937   1.03983057]
 [-1.69337213 -0.62414539  1.06170225 -0.17529595  1.37775004]
 [ 2.26930618 -0.20784986  0.81450009 -1.6406312   1.94288754]
 [ 0.10350752 -0.06312151 -0.33681023 -0.30178145 -0.73511004]
 [ 0.33292654 -0.53923434 -0.36953798 -0.38226834 -0.02478076]]
0.890904
```

Num Layers	**13**
Num Images	**4000**
Num Iterations	**15**
Dropout	**25%**
Training acc	**92%**
Testing acc	**90%**

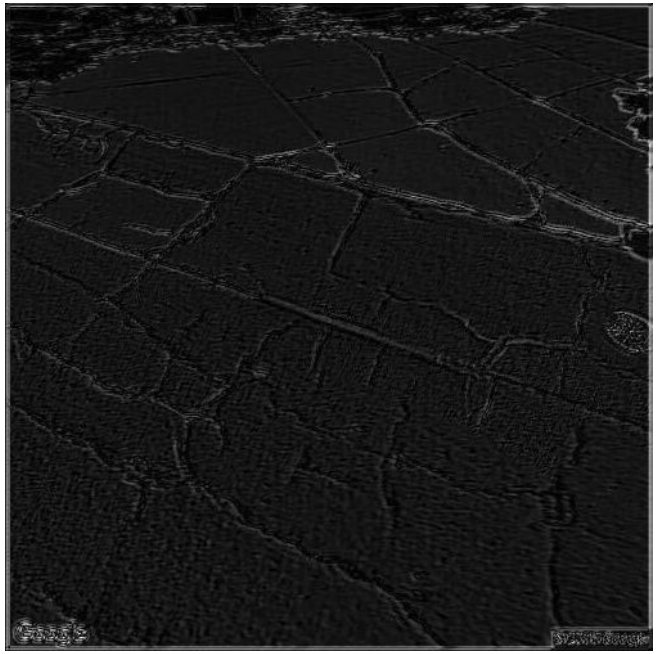
Image Visualization (Bad Image)

Original Image (Input bad Image)

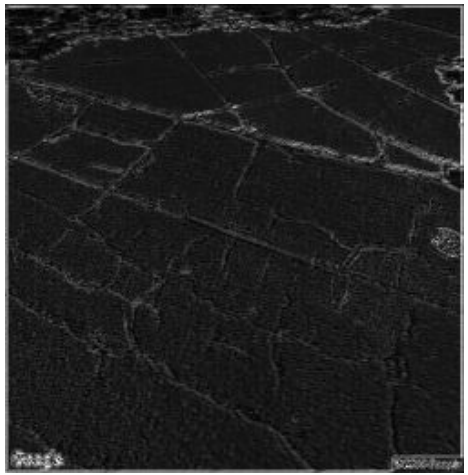


Filter result at Layer 1 (convolution 1)

Image Visualization (Bad Image)



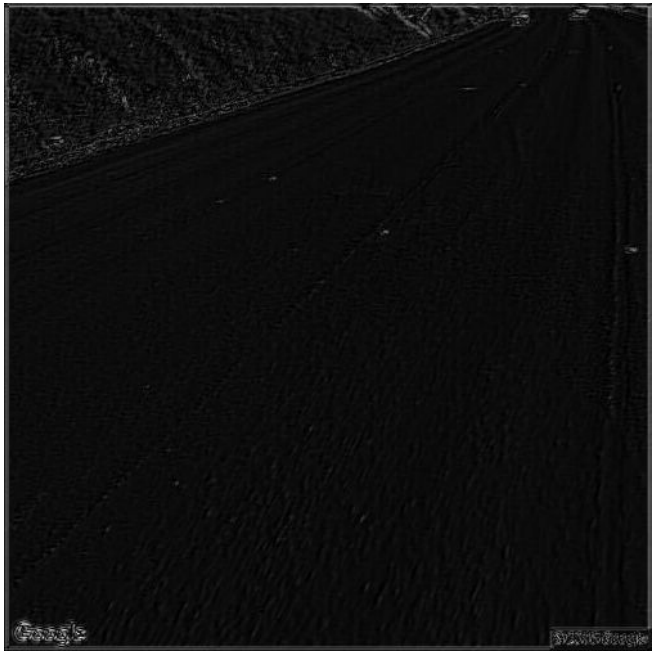
Conv 1 : 500*500



Max Pool 1 : 250*250

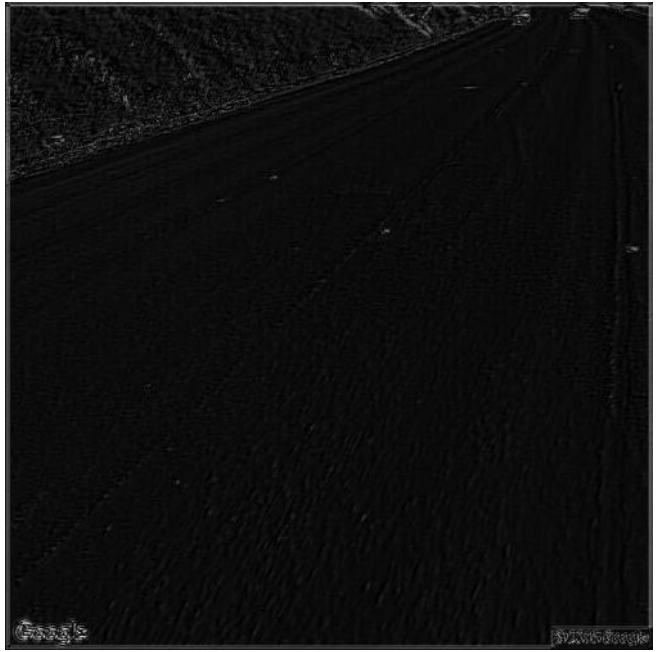
Image Visualization (Good Images)

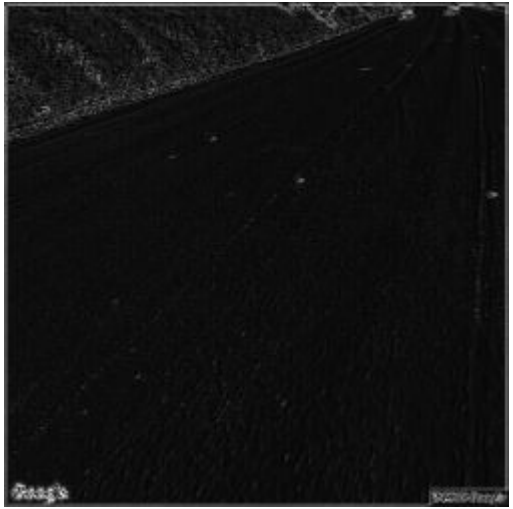
Original Image (Input bad Image)



Filter result at Layer 1 (convolution 1)

Image Visualization (Good Images)



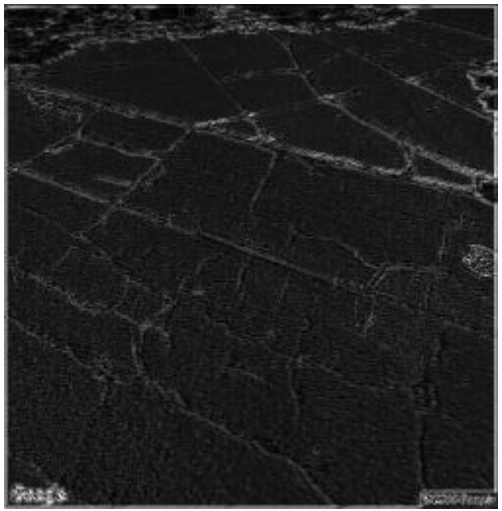


Conv 1 : 500*500

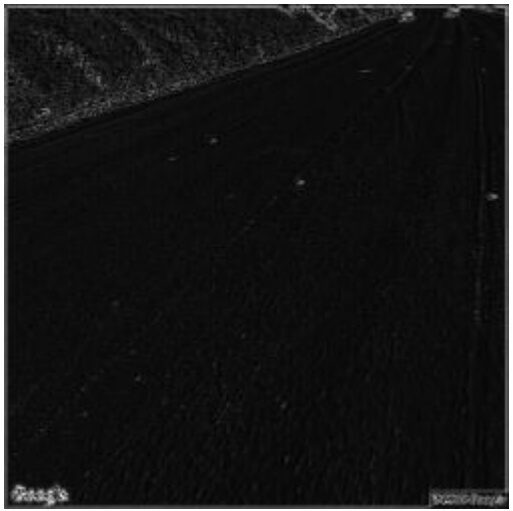
Max Pool 1 : 250*250

Image Visualization : Good vs Bad

Bad Image : Max Pool1

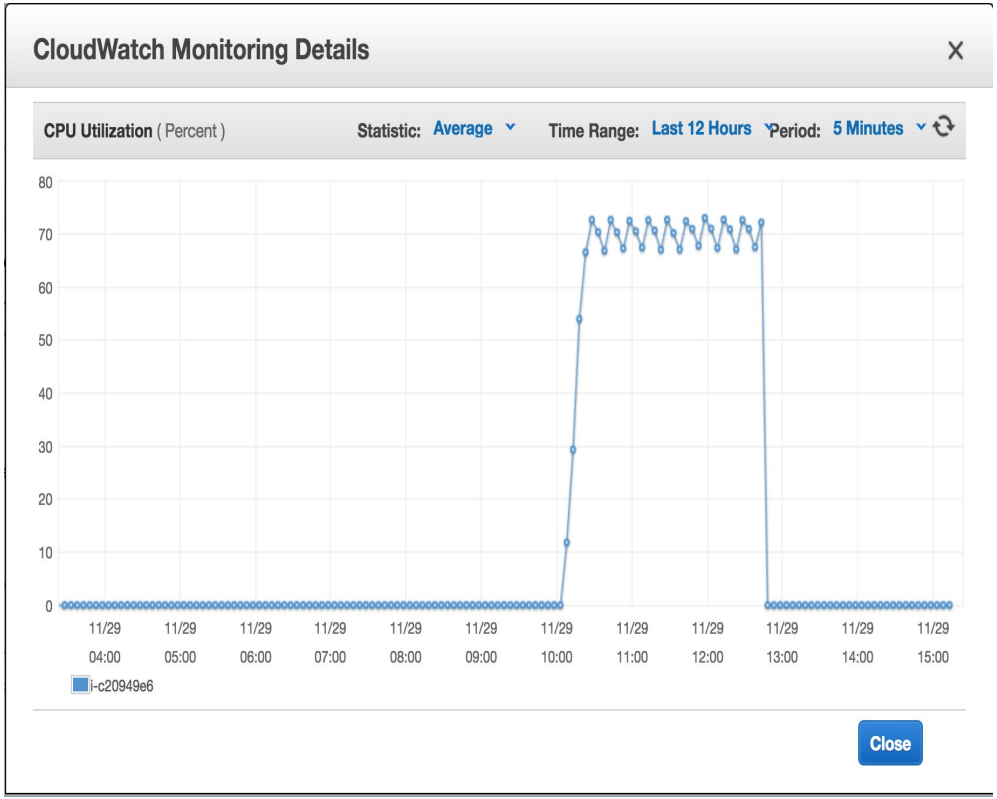


Good Image : Max Pool1



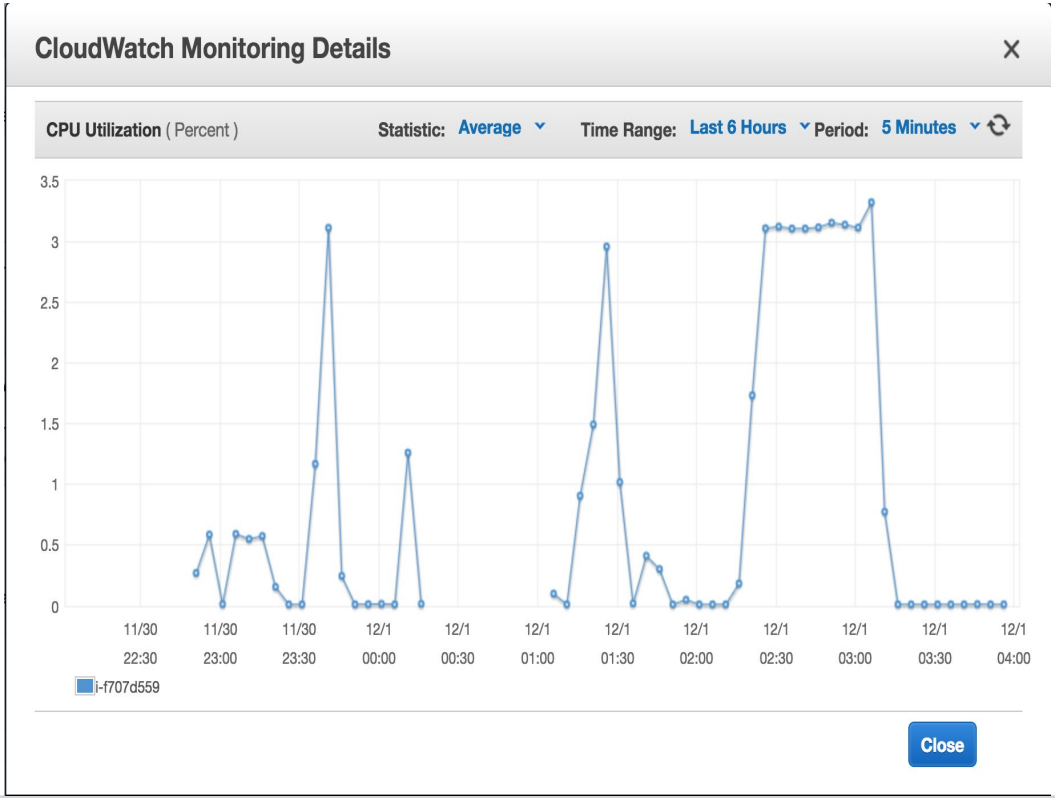
Neuron activation are high at deeper layer in Bad Image as compared to Good Images

AWS Metrics - CPU Utilization



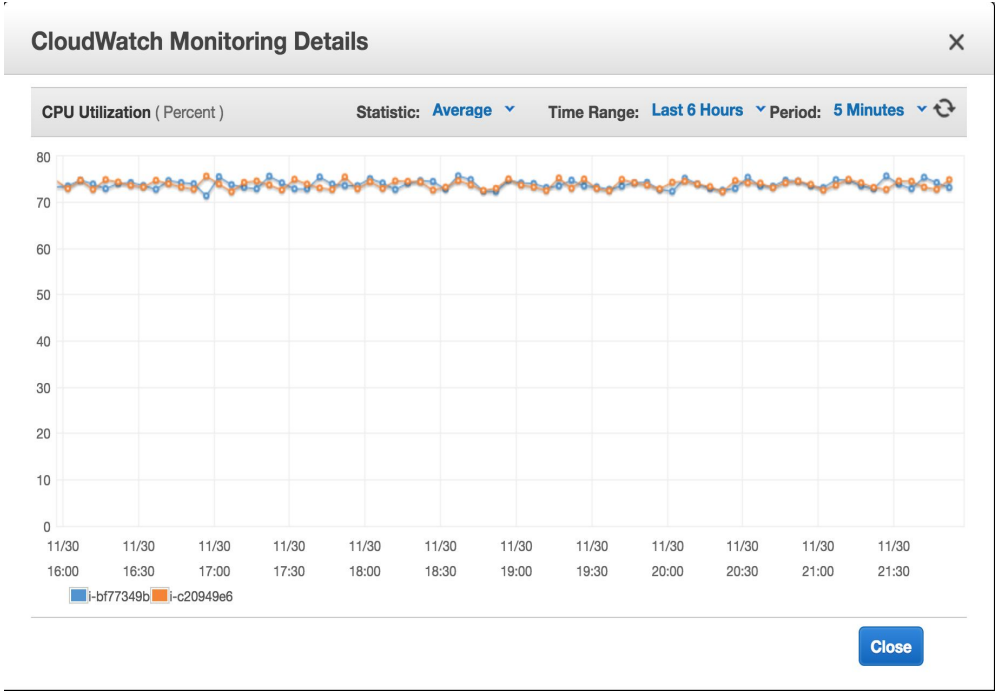
Number of Images	500 per category
CPU cores	optimized CPU 32 cores
Memory	60GB
Time	3 hours

AWS Metrics - GPU Utilization



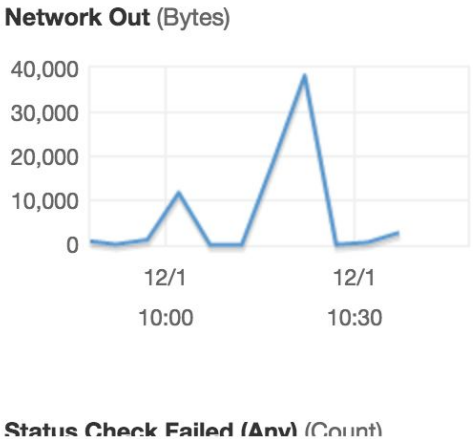
Number of Images	500 per category
GPU cores	32 cores
Memory	60GB
Time	45 minutes

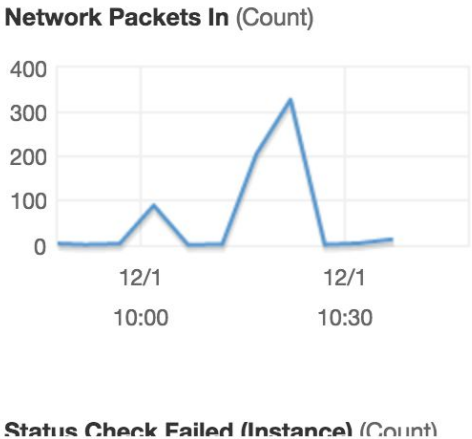
AWS Metrics - CPU Utilization 2

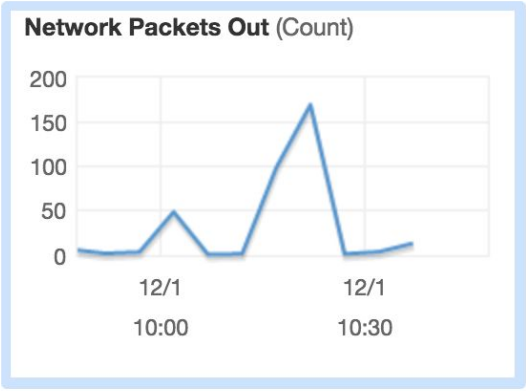


Number of Images	2000 per category
CPU cores	optimized CPU 32 cores
Memory	60GB
Time	6 hours

AWS Metrics - Network







Status Check Failed (Any) (Count)

Status Check Failed (Instance) (Count)

Status Check Failed (System) (Count)

CONCLUSION AND FUTURE

Analysis Conclusion

- Deep Learning Model is working well in images processing and prediction based on our data set
- It is highly possible that Deep Learning Model can be used in road inspection practice
- It is worthy of researching and testing to use Deep Learning Model to replace more infrastructure related work

Future Work

- Run and revise the model against real drone images
- Deal with more noisy images for real practice
- Evaluate the road condition in multiple grades or multiple categories

Applications :

- External proposal for image based processing

Agricultural Inspection

Railway Inspection

Market Potential

- Infrastructure drone market worth **$45 bn** - Pwc Dec. 2016
- **Fourteen** bills addressing drone use near critical infrastructure have been introduced in **nine** states so far in **2016**. - NCSL Sep. 2016
- Current drone applications are **not automatic or durable enough**
- **Reliable** and **distant** transmission is also necessary

Questions?

Links :

- TensorFlow : https://www.tensorflow.org/
- Dataset : https://console.cloud.google.com/storage/browser/ini_practicum_drones/?project=satellite-1475994749306
- Scripts https://github.com/raquerod/drone_imagery
- Sample Images https://cmu.box.com/s/kp1kfbpjfngnzp3iadt0wz0fxbevpelb
- Images and GIFs from google search (several sites)

Links :

- Special thanks to Stanford Cs231n course and Andrej Karpathy. Link : www.cs231n.github.io
- Pwc Press on drone and infrastructure
- NCSL drone and critical infrastructure