



# **AirBnb Data- Uncovering a “host” of information**

**-Deepshikha Adhikari**



# Format

Motivation

Data Insights

Findings

Supervised V/S Unsupervised

Challenges



# Motivation

Uncovering “host” of informations on hosts!

How to differentiate best hosts?



# Data Insights

Resource: InsideAirbnb

Initially, 80, 000 rows with 83 features.

Selected subset of data related with NY listings

Data-Processing/Cleaning reduced to 16652 rows with 13 variables

Change categorical data into numeric variable



# Sample Data

1	date_diff	host_total_li	number_of_i	price	minimum_ni	review_score	host_is_supe	diff_date
2	417.000000	1	2	80	3	100	0	417
3	410.000000	1	4	150	7	100	0	410
4	396.428571	1	11	114	4	98	0	396.43
5	393.428571	2	48.5	40	1.5	97	0	393.43
6	392.714286	2	74.5	240	3	91.5	0	392.71
7	391.714286	1	16	170	5	99	0	391.71
8	390.285714	3	105	54	2	91.5	0	390.29
9	390.142857	1	1	145	3	60	0	390.14
10	390.142857	7	1.33	275	1	96.67	0	390.14
11	389.714286	2	67	62	2	95	1	389.71
12	387.428571	10	4	125	12.67	95.33	0	387.43
13	387.428571	1	194	150	1	86	0	387.43



# More on Data

**Numerical** :diff\_date , host\_total\_listings\_count,  
number\_of\_reviews, price, minimum\_nights, review\_scores\_rating

**Categorical**= host\_is\_superhost, host\_has\_profile\_pic,  
instant\_bookable, cancellation\_policy, room\_type,  
host\_response\_rate

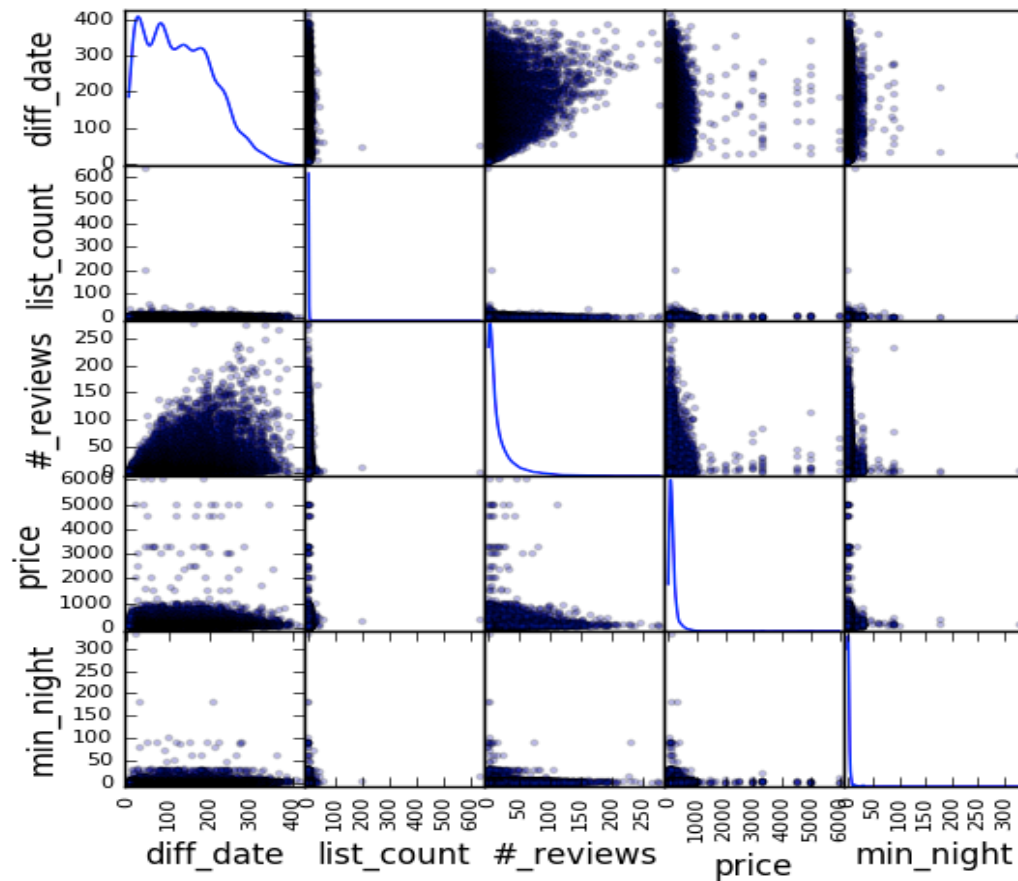
# Findings

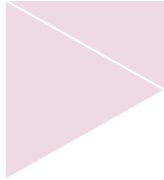
## 1. Correlation Matrix



# Findings

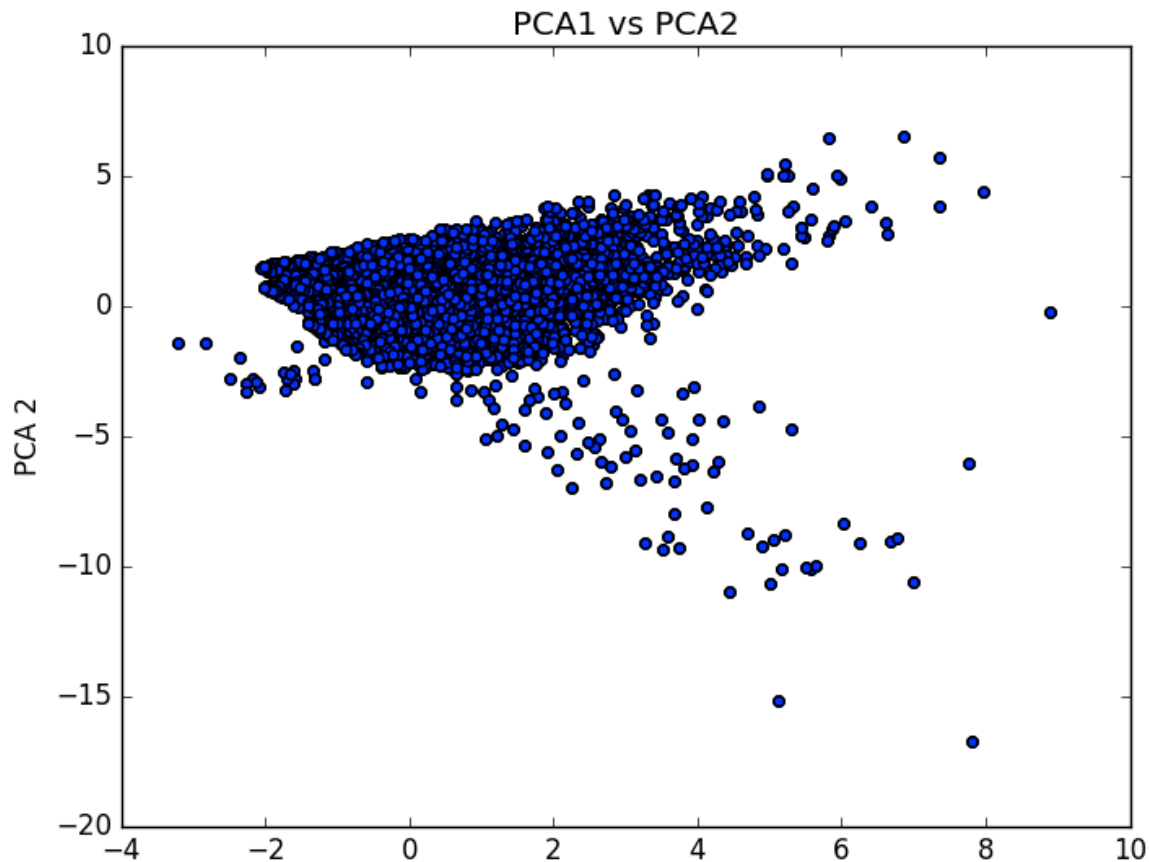
## 2. Scatter Matrix





# Findings

## 3. PCA1 vs PCA2





# Supervised vs Unsupervised

Supervised Machine Learning method

Understand what makes someone a superhost?

Binary Classification



# Challenges

Data Cleaning

Missing Data

Mixed Data (Categorical and Numerical)



# Future Work

Carry out K-fold cross validation logistic regression

Carry out Multiple Correspondence Analysis

Get rid of the outliers to see

NLP to get more variables





**Thank you!**

---

---

# Image Segmentation Using DeconvNet to distinguish different breeds of dogs

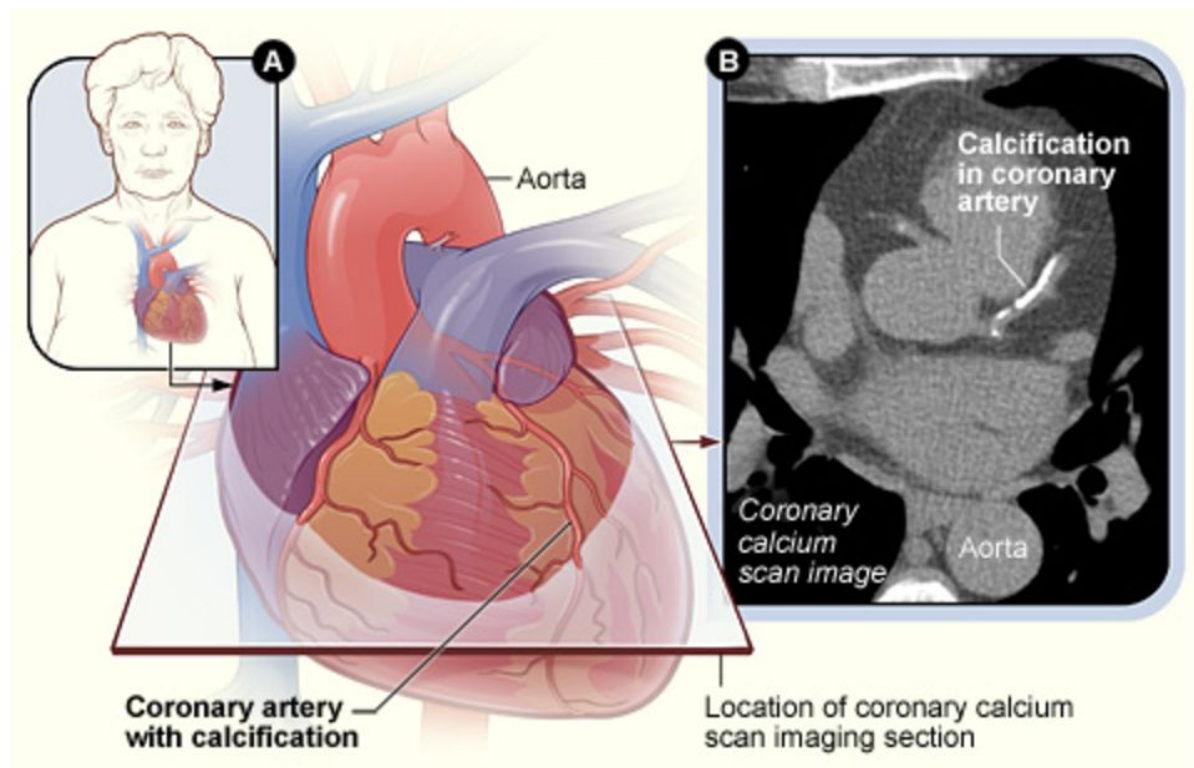
— Amelia Yeoh, CSC 390 —

Dec 6, 2016

---

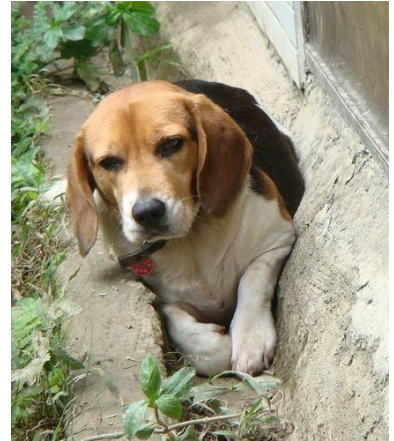
---

# Motivation



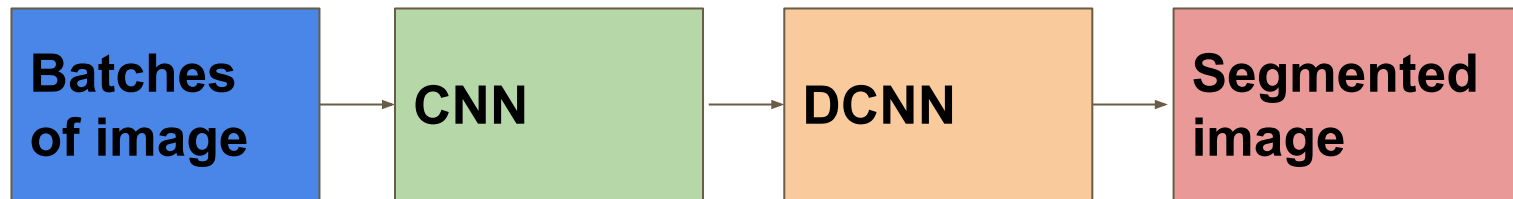
# Data

- Stanford Dog Dataset
- Subset of the ImageNet dataset
- Contains 20 580 images (120 dog breeds, ~150 images per class)
- Train\_data (12000x12000), test\_data (8580x12000)



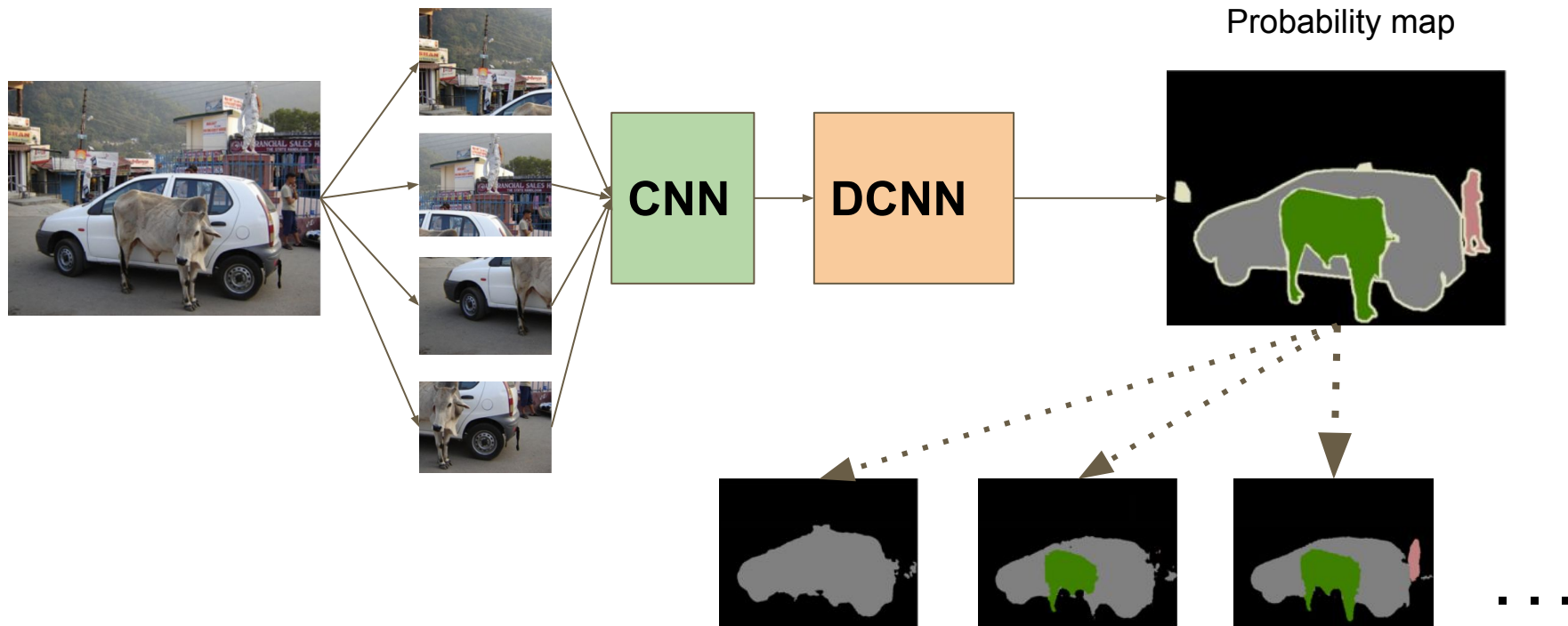
# Architecture: Instance-wise Segmentation

BIG PICTURE

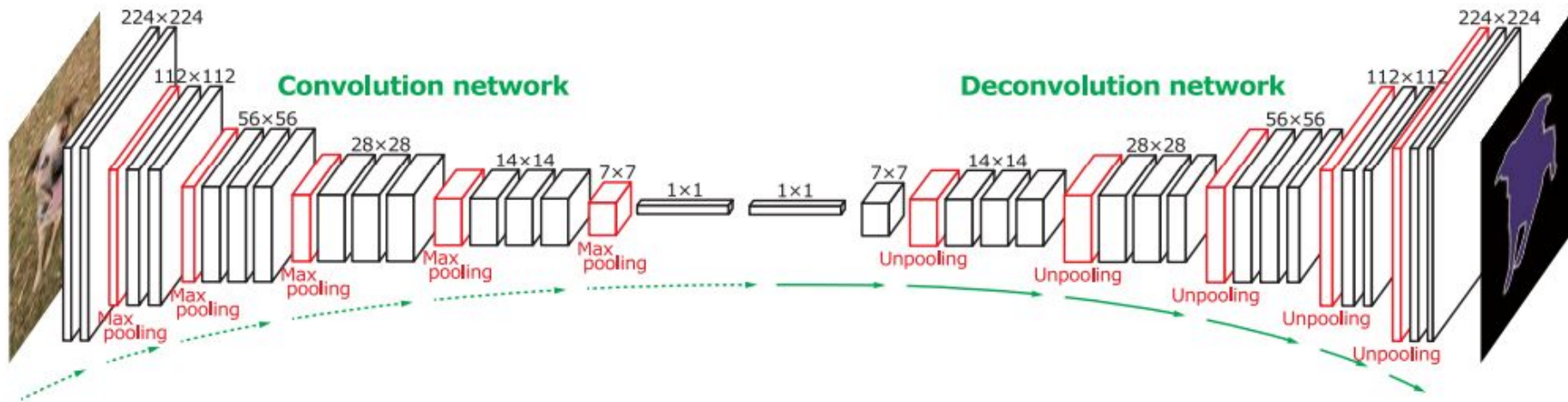


# Architecture: Instance-wise Segmentation

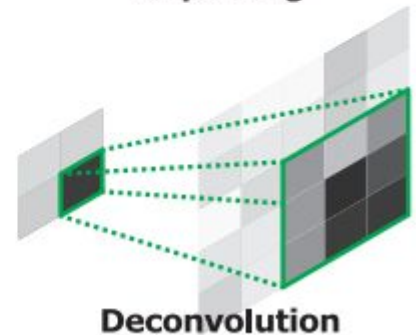
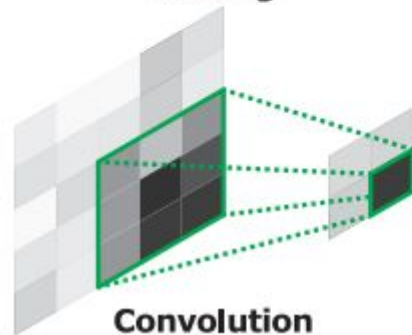
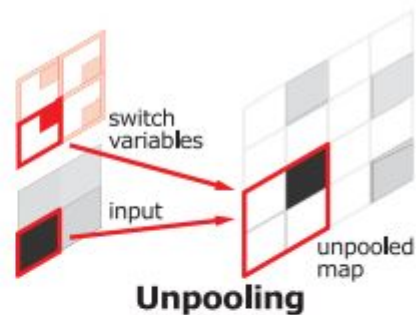
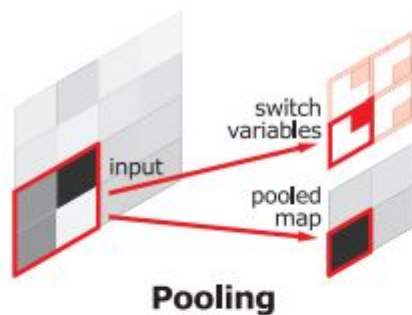
BIG PICTURE



# Architecture



# Architecture





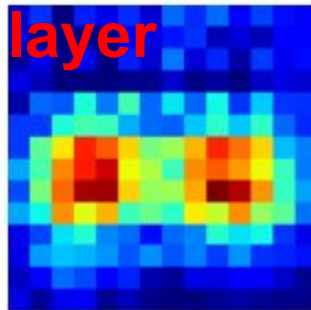
# Expected Results and Outcomes

## 14x14 deconv

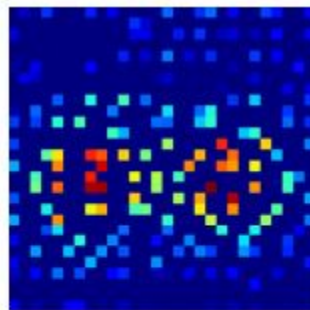
layer



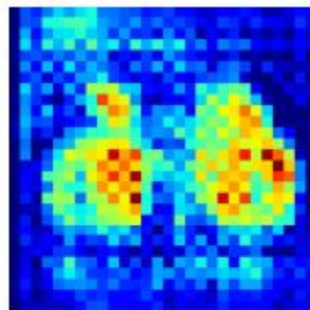
(a)



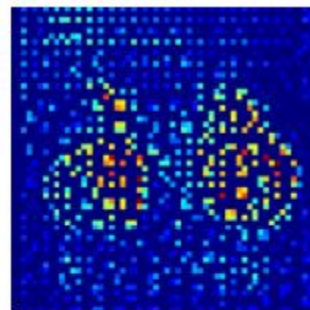
(b)



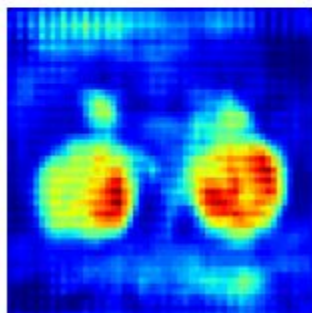
(c)



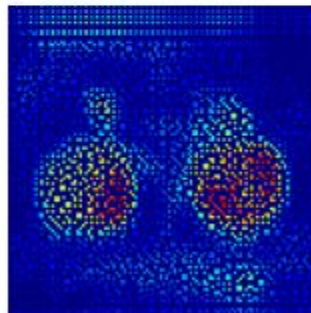
(d)



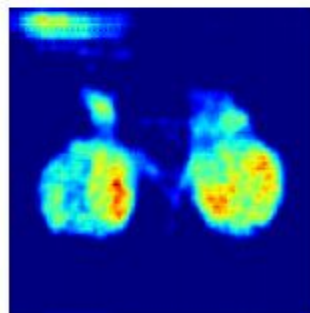
(e)



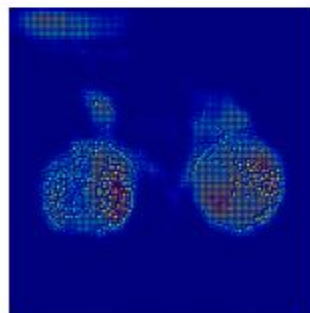
(f)



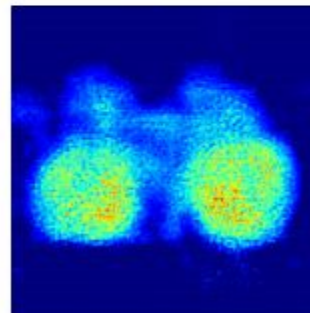
(g)



(h)

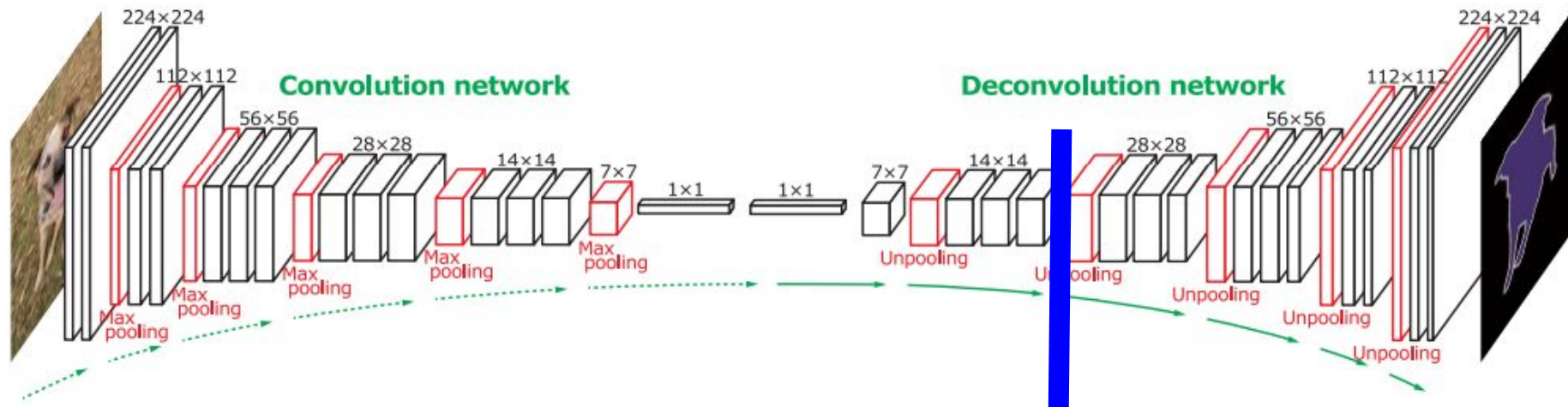


(i)



(j)

# Expected Results and Outcomes

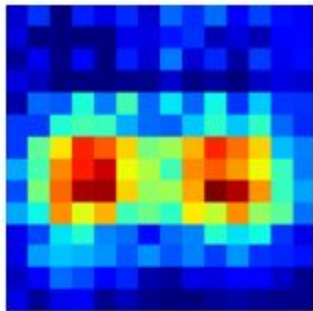


# Expected Results and Outcomes

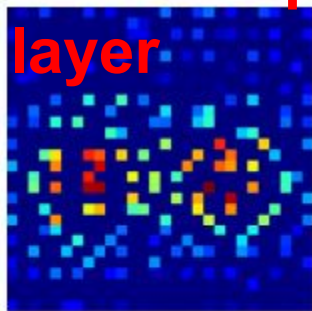
28x28 unpooling  
layer



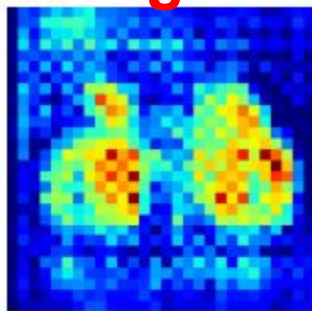
(a)



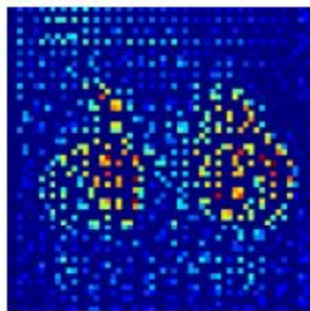
(b)



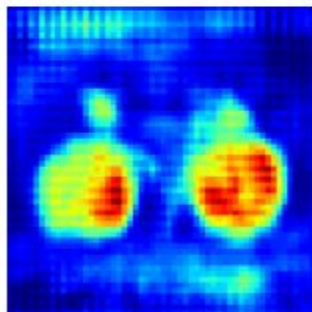
(c)



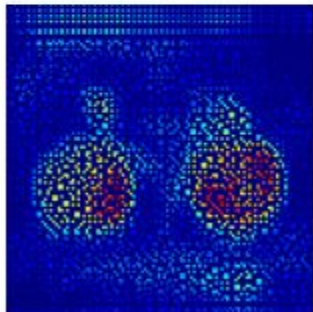
(d)



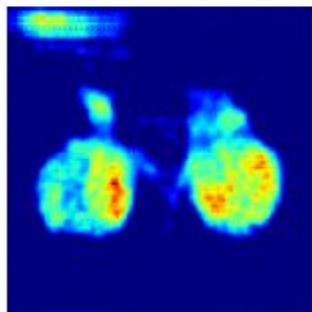
(e)



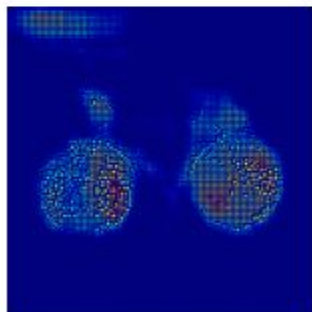
(f)



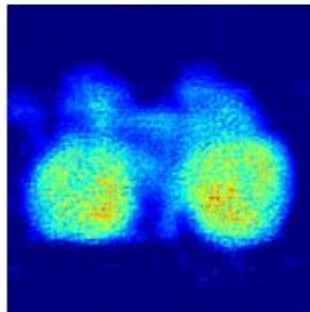
(g)



(h)

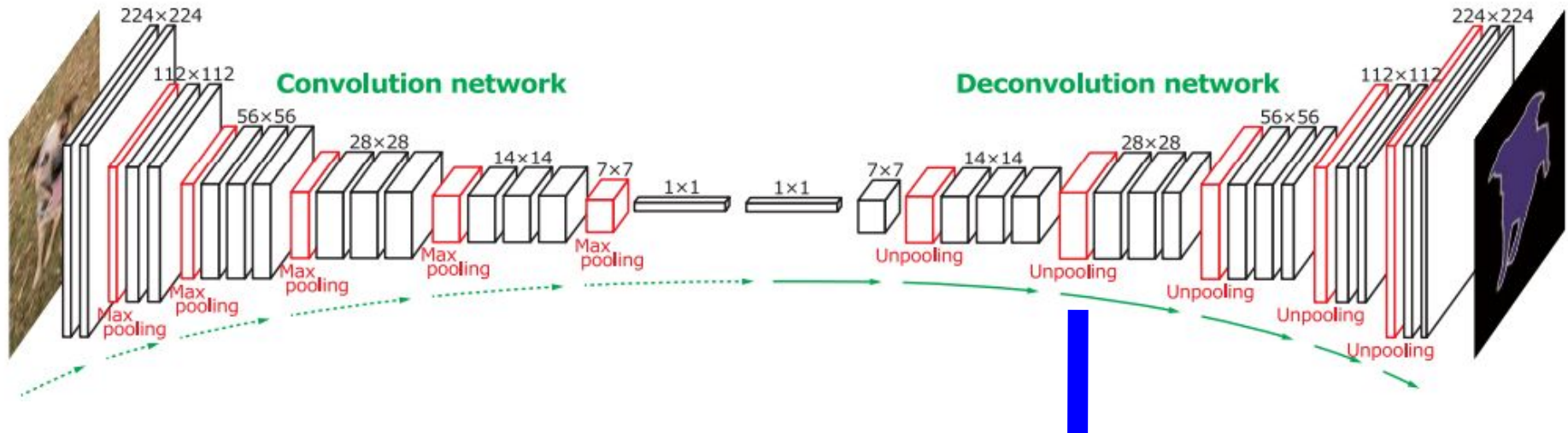


(i)



(j)

# Expected Results and Outcomes



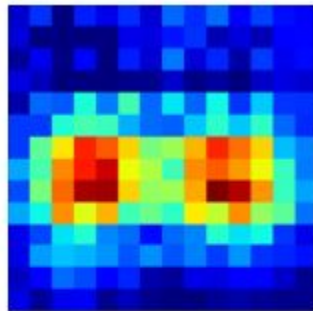


# Expected Results and Outcomes

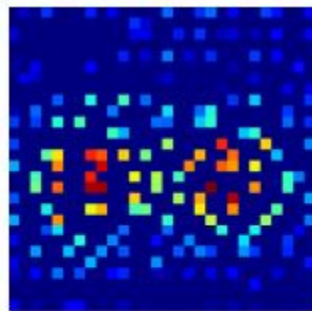
28x28 deconv  
layer



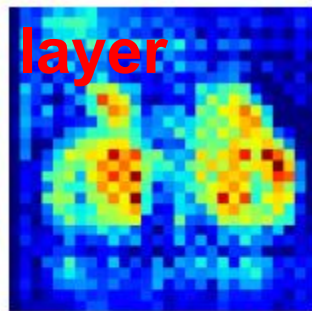
(a)



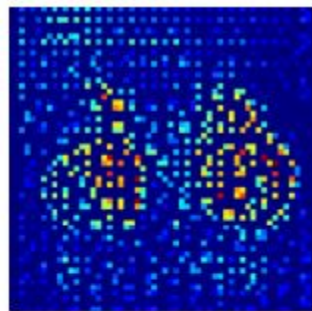
(b)



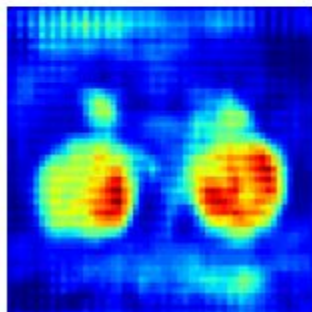
(c)



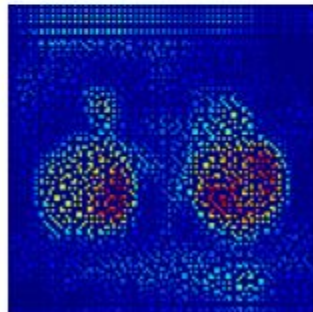
(d)



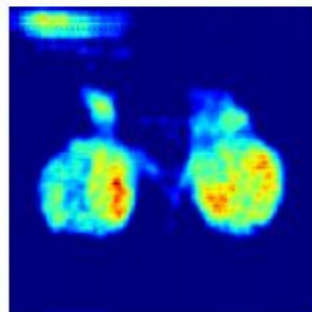
(e)



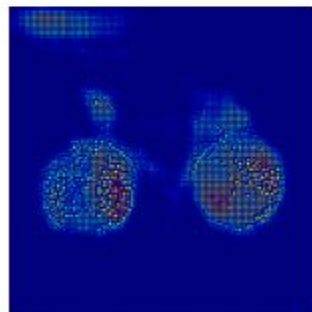
(f)



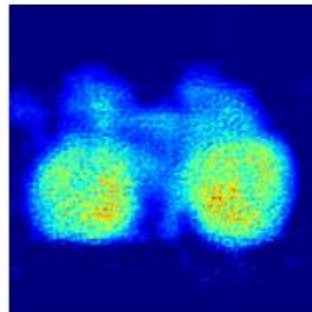
(g)



(h)

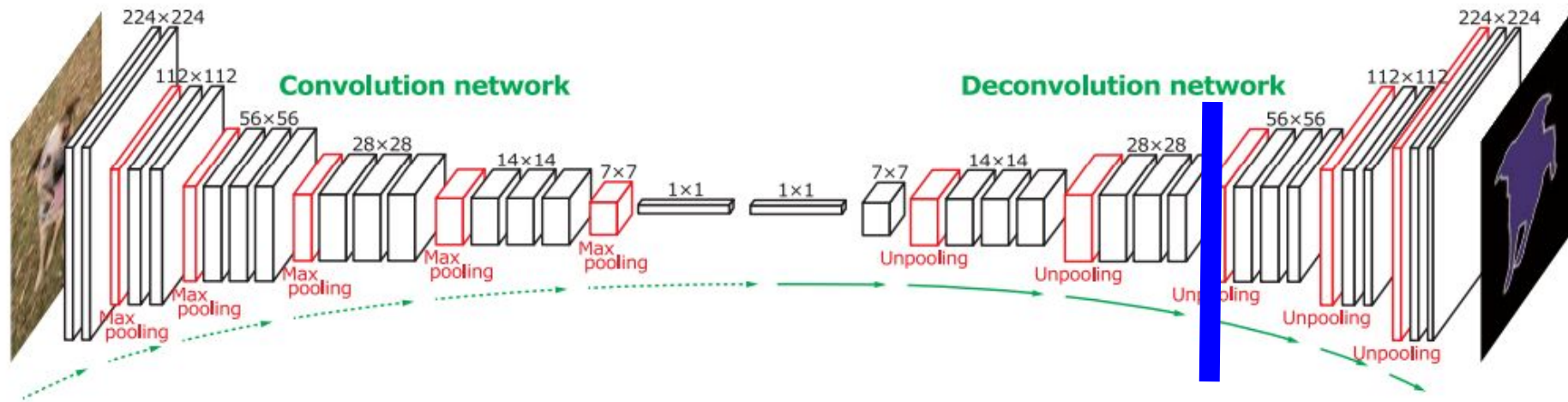


(i)



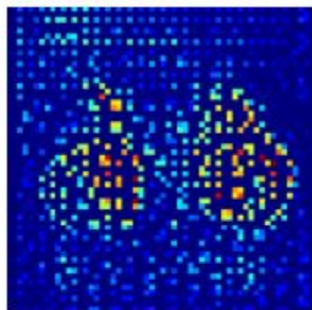
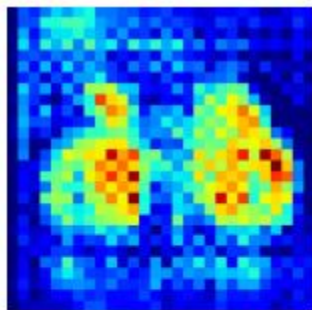
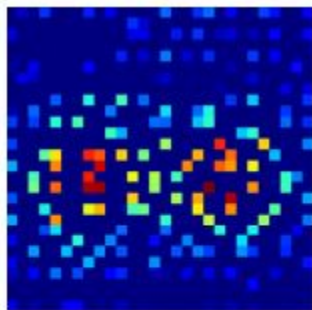
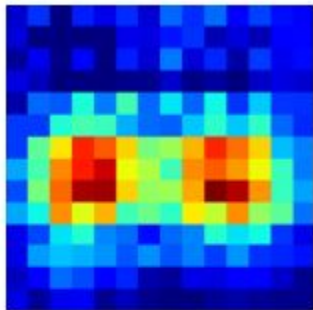
(j)

# Expected Results and Outcomes



# Expected Results and Outcomes

56x56 unpooling



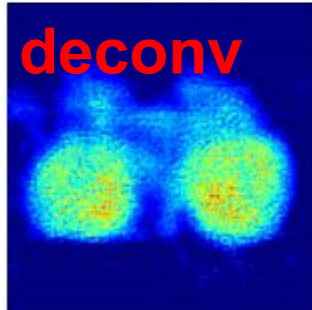
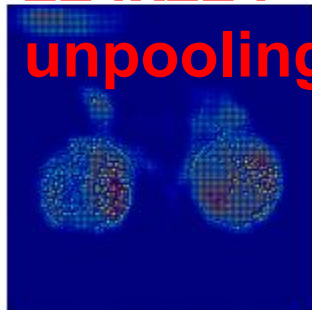
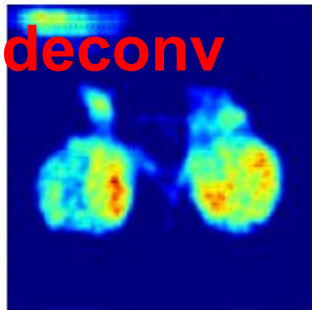
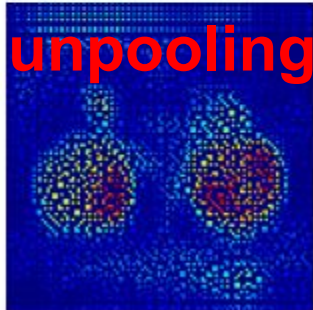
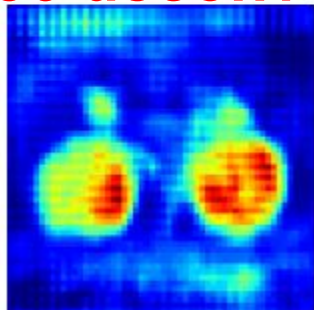
56x56 deconv

112x112

112x112

224x224

224x224



(f)

(g)

(h)

(i)

(j)

unpooling

deconv

unpooling

deconv

# Expected Outcomes and Challenges

< 6 days in a single Nvidia GTX Titan X GPU with 12G memory.

Filters in lower layers - overall shape of an object (location, shape, region)

Filters in higher layers - capture class-specific fine details (ignored in CNN)

Segment nose / whole dog image

Dog data set labels are not by pixel



# Future Work

Getting the model to run using VOC 2012 Challenge dataset

Apply Dog Dataset to VGG-16 layer Model.

Perform accuracy test on 30, 60 and 100 images per class.

**... and even further**

**Thank you!**

Q&A





# NYC UBER PICKUPS DATA ANALYSIS

yvaine liu  
December 6 2016

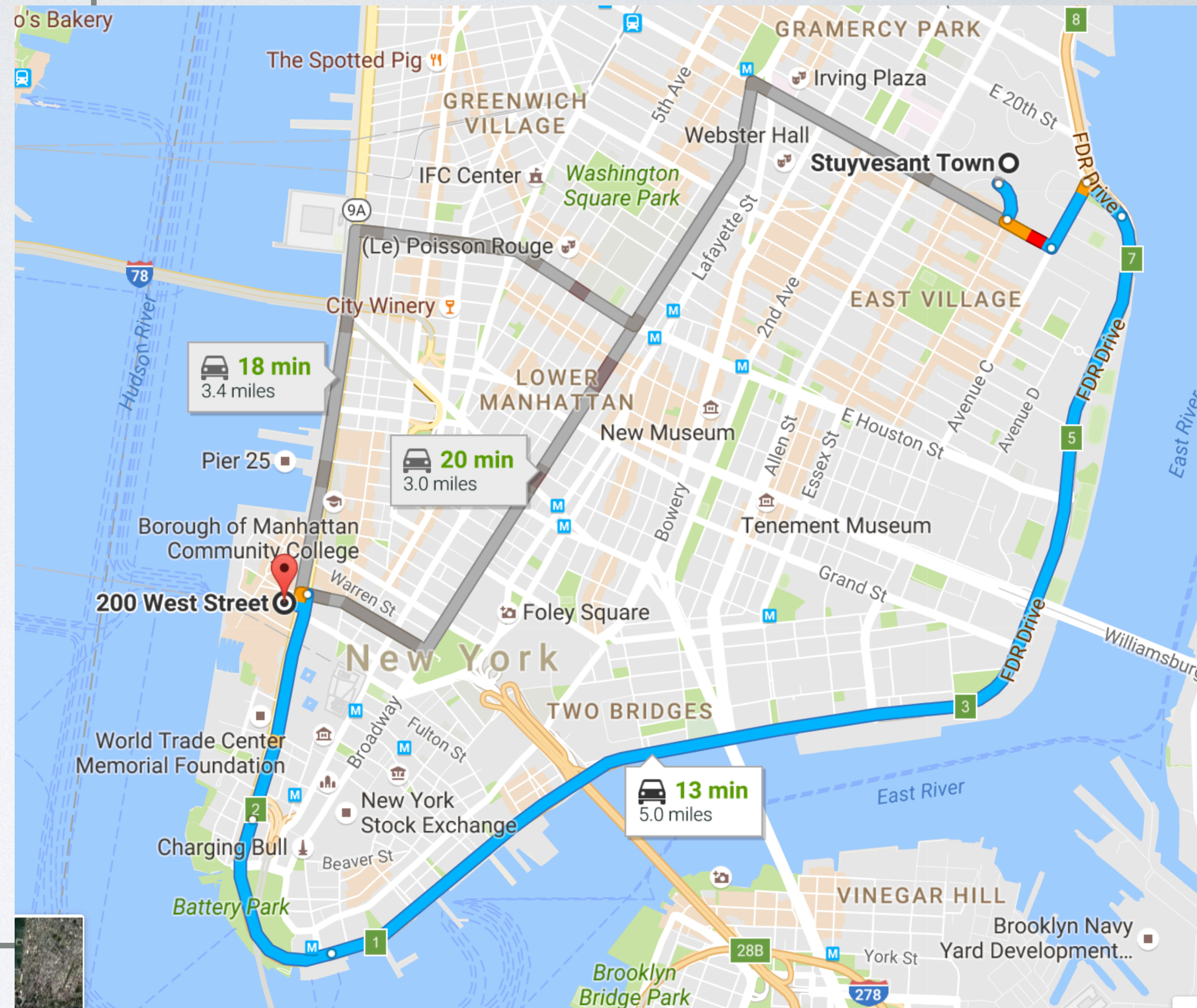


# MOTIVATION

- New technologies and business models are changing our lives dramatically
- In what ways has Uber reshaped our commuting habits? What patterns can we learn from existing data?
- my summer Uber experience in Manhattan:



- From: East 20 st
- To: 200 west st
- 75%-150% surge pricing at 8:30 am
- avg wait 5-8 minutes





# MOTIVATION

- What can be implied from it?
  - surge pricing → **time** influences
  - different waiting times → density at different **locations**
- How can the results be useful?
  - insight on time and location patterns
  - good base results for more in-depth research



# DATA

|———— p = 3 ———|

- Uber pickups in NYC from April to September 2014
- NYC Taxi & Limousine Commission
- Over a million data points → taking a subset
- 4 features: Time, Latitude, Longitude, Base

```
"Date/Time", "Lat", "Lon", "Base"
"4/1/2014 0:11:00", 40.769, -73.9549, "B02512"
"4/1/2014 0:17:00", 40.7267, -74.0345, "B02512"
"4/1/2014 0:21:00", 40.7316, -73.9873, "B02512"
"4/1/2014 0:28:00", 40.7588, -73.9776, "B02512"
"4/1/2014 0:33:00", 40.7594, -73.9722, "B02512"
"4/1/2014 0:33:00", 40.7383, -74.0403, "B02512"
"4/1/2014 0:39:00", 40.7223, -73.9887, "B02512"
"4/1/2014 0:45:00", 40.762, -73.979, "B02512"
"4/1/2014 0:55:00", 40.7524, -73.996, "B02512"
"4/1/2014 1:01:00", 40.7575, -73.9846, "B02512"
"4/1/2014 1:19:00", 40.7256, -73.9869, "B02512"
"4/1/2014 1:48:00", 40.7591, -73.9684, "B02512"
"4/1/2014 1:49:00", 40.7271, -73.9803, "B02512"
"4/1/2014 2:11:00", 40.6463, -73.7896, "B02512"
"4/1/2014 2:25:00", 40.7564, -73.9167, "B02512"
"4/1/2014 2:31:00", 40.7666, -73.9531, "B02512"
```



# METHODS

- Data very discrete → clustering for underlying patterns, completely unsupervised
- Great geographical features → good to visualize, won't lose much information in 2D
- Time feature: how to represent? how to cluster? → 3 clusters:
  - weekdays
  - weekends
  - national vacations +1 and -1 day



# METHODS

- `sklearn.cluster.AgglomerativeClustering` + **PCA**
- A type of hierarchical clustering, very similar to UPGMA, bottom-up approach with more variation and parameters; no edit distance matrices
- 3 parameters worth mentioning:
  - `n_clusters`
  - `affinity` = 'euclidean', 'l1', 'l2', 'Manhattan', 'cosine'...
  - `linkage` = 'ward', 'complete', 'average' - what criterion of distance to use. The algorithm will merge the pairs of cluster that minimize this criterion.



# METHODS

- (continued) linkage options:
  - ✓ • ‘ward’: minimizes the sum of squared differences within all clusters (wc-ss)
  - ‘complete’: minimizes the maximum distance between observations of pairs of clusters
  - ‘average’: minimizes the *average* of the distances between all observations of pairs of clusters



# WHAT TO EXPECT

- hierarchical clustering of data points, by both location and time
- visualization of cluster locations in 2D dimension
- A comparison (some alignment) with the actual map of NYC

- In what region do people use Uber the most in NYC? During what times?
- visual explanation of the clusters
- better understanding the results within the context of specific NYC regions



# FUTURE DIRECTIONS

- More analysis on drop-off data too
- Compare with taxi data: answer the question “Is Uber taking away jobs of traditional taxi drivers?”
- More detailed analysis on:
  - *time*: congestion problem
  - *customers/drivers*: people’s lifestyle, economic research on labor



“Questions?”

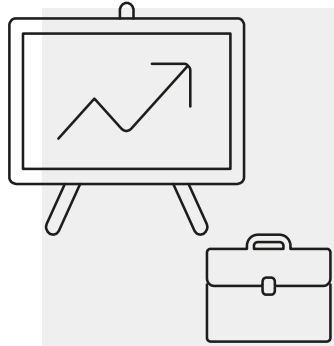




# HMMs & Stock Market Analysis

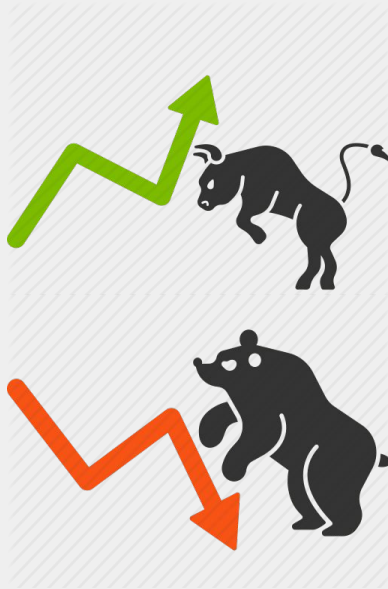


Isha Raut  
CSC 390



# Stock Market Analysis

- *Unpredictable to randomwalk hypothesis*
- *Many existing time series approaches*
- *Recently popularity of HMMs, ANNs ,  
Bayesian networks, deep learning*



*DJIA is interesting for ...*

- *investors with a portfolio stock*
- *indicating the economy*
- *To start stock market analysis*

# 1. Dow Jones Industrial Average

*Why should you even care?*



# Daily data from 2000-2016

*Pulled from Yahoo finance API*

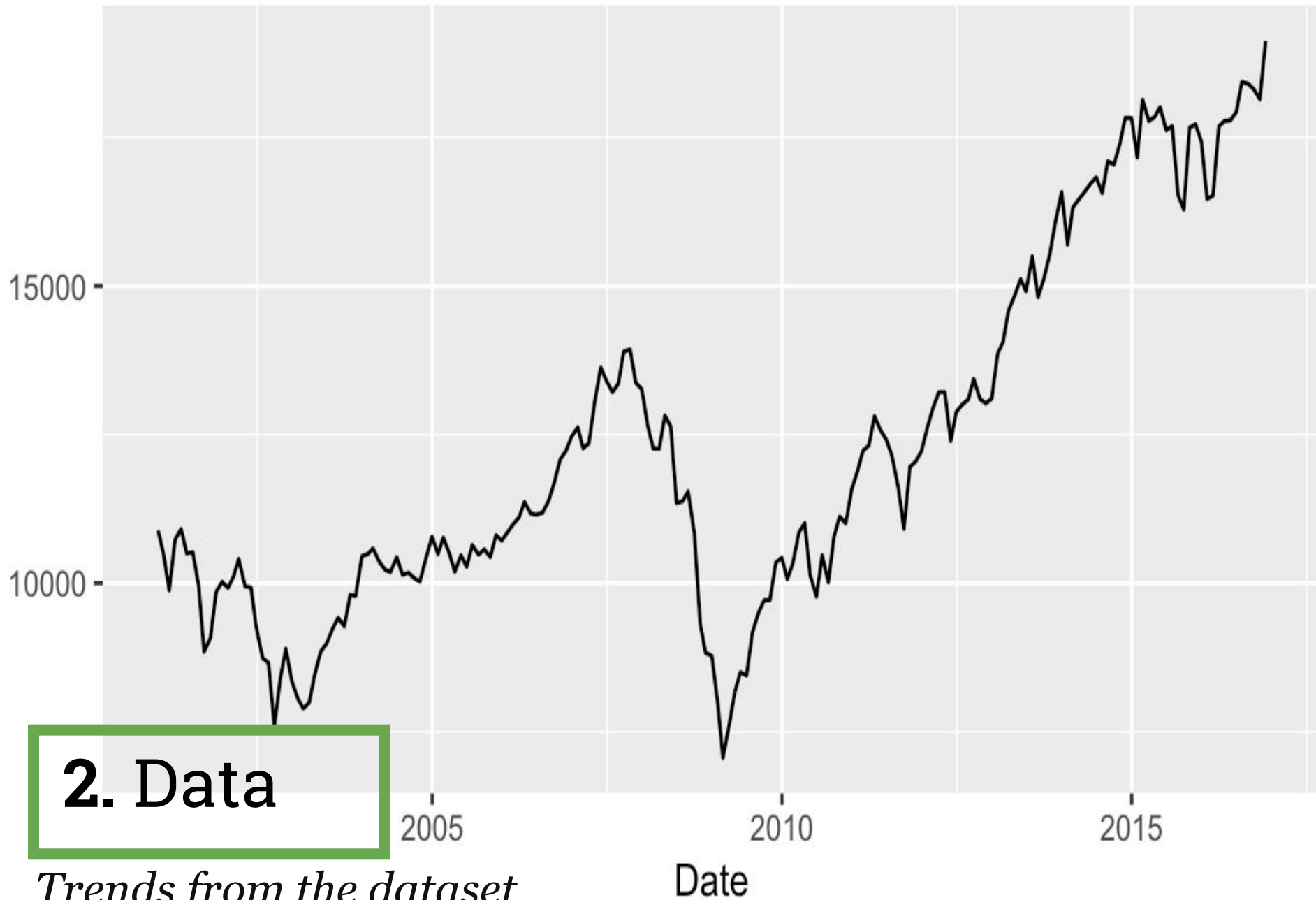
## 4,025 data points

*With Open Close , high , low and volume attributes*

## 2. Data

*About the dataset*

# Dow Jones Close over time



# #GOALS



## Application

Apply Hidden Markov model to time series



## Prediction

Build a predictive model to give close or return value



## Evaluation

Evaluate and compare to other methods

- *Implement Hidden Markov model and find hidden states*
- *With states find most probable value change for prediction*
- *Tools : Python(Hmm Learn)  
R(RHmm , depmixS4)*

### 3. Methods

*How am I doing this project?*

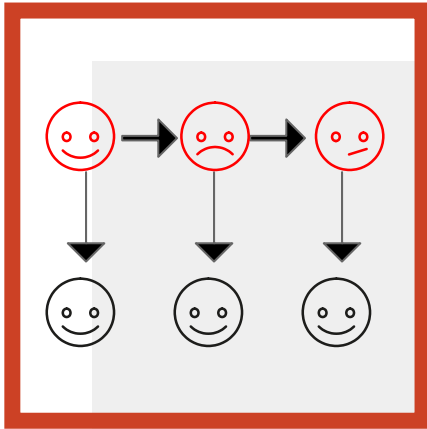
# Road Map

```
graph LR; A[Initialize & Construct Model] --> B{Obtain states and trends}; B --> C[Use model results to predict change]
```

Initialize &  
Construct  
Model

Obtain  
states and  
trends

Use model  
results to  
predict  
change



# Hidden Markov Model

- *Can model temporal data*
- *Strong stats foundation*
- *Handles new data robustly*
- *Computationally efficient*
- *Able to predict similar patterns*

# Recap : HMM

Decoding:  
**Viterbi**

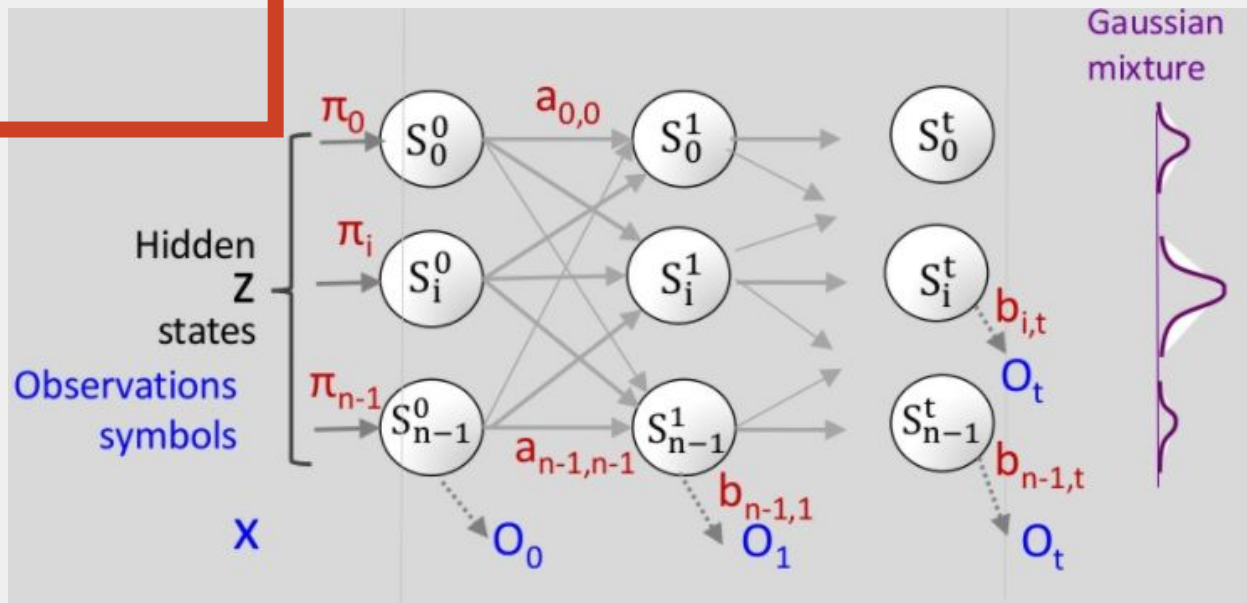
Training:  
**EM - Baum Welch**

- **Z** hidden states  
( $N=2, 3$  or  $5$  : bullish , bearish , no change)
- **M** observation symbols per state
- Transition matrix **A** =  $\{a_{ij}\}$  represents the transition from state  $i$  to state  $j$
- Observation emission matrix **B** where  $b_{ij} = P(O_t = j | S_t = i)$  is the prob of observing  $O$  at state  $j$
- Initial state distribution  **$\pi$**  =  $\{\pi_i\}$

$$\lambda = (A, B, \pi)$$

# Gaussian HMM

Image from patricknicolas.blogspot.com



## G-HMM vs HMM

Instead of Multinomial ,  
Gaussian Model used

This is because the  
observations are  
continuous

## Key Components

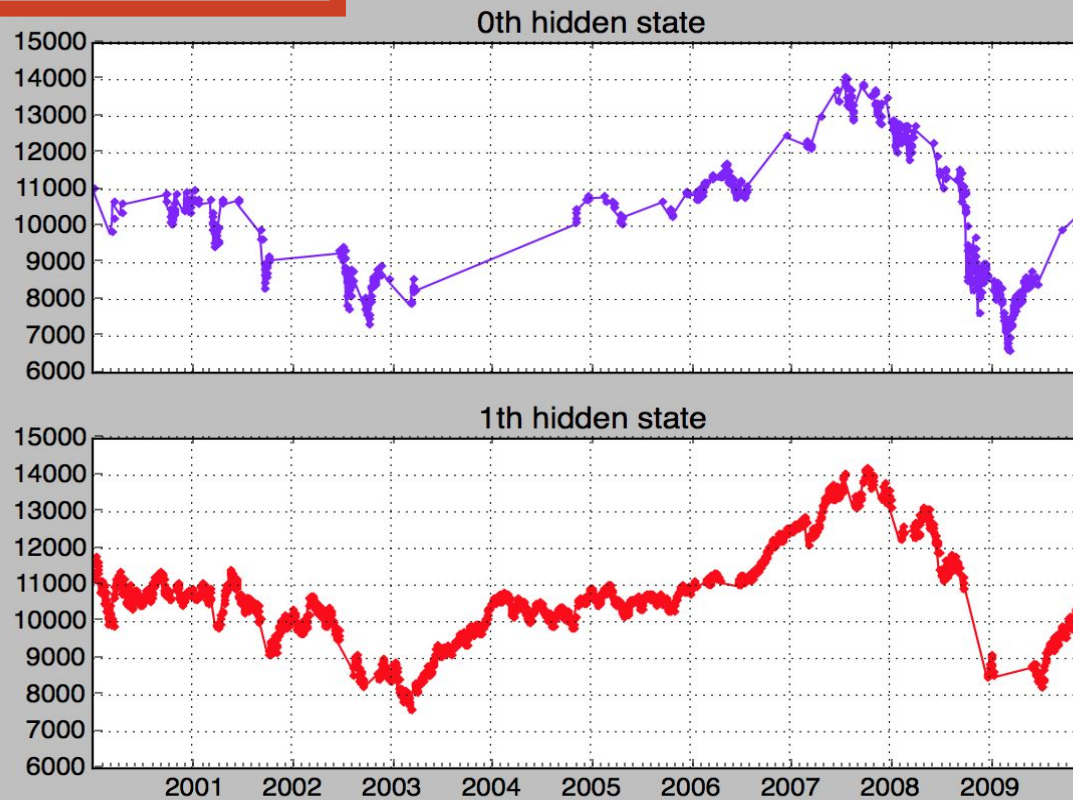
$Z$  = market behaviour  
of  $n$  hidden states

$X$  = observation of  
index

$\lambda (A_{ij}, B_{ik}, \pi_i)$

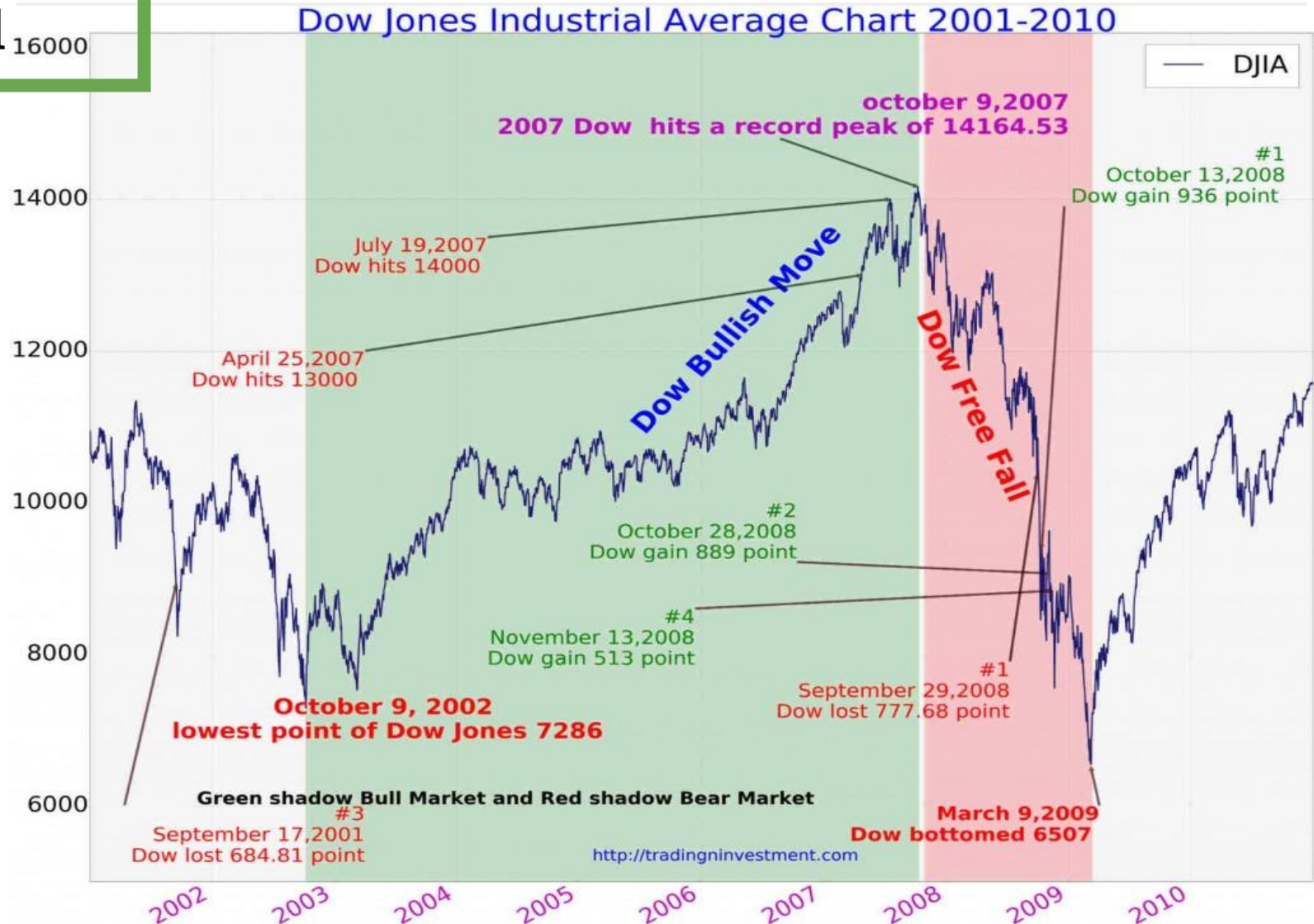


# Hidden States 2000-2010

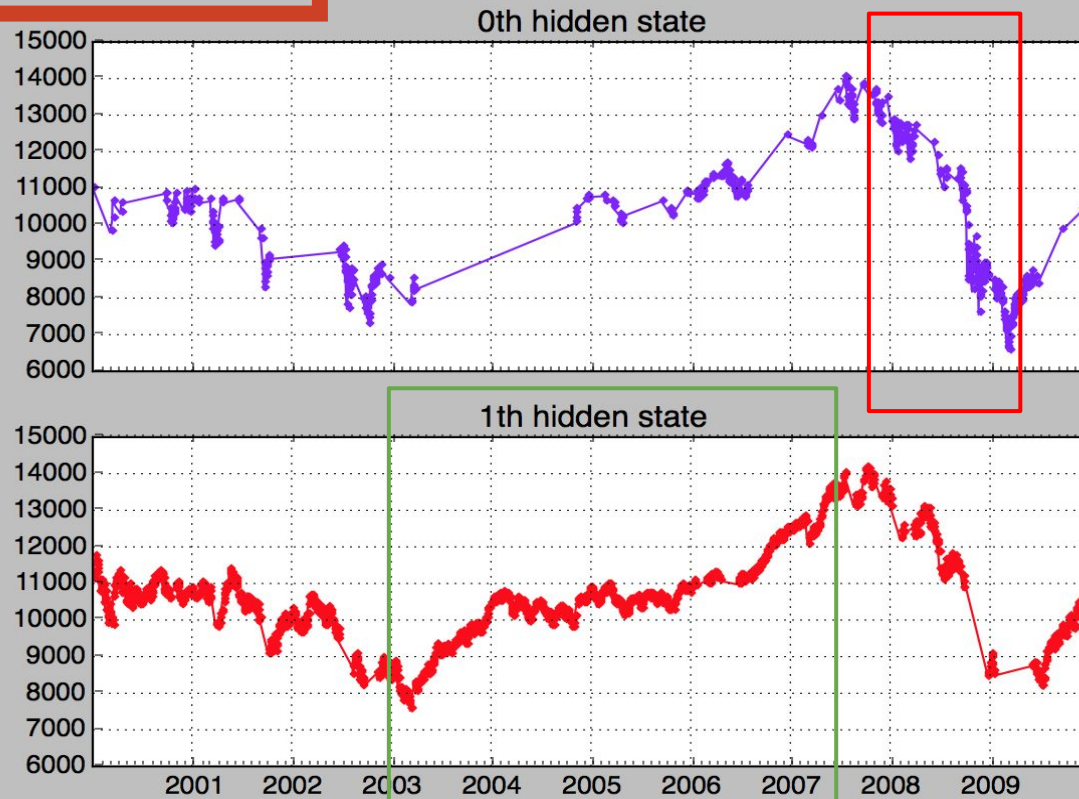


# Actual Trend

Image and analysis from tradinginvestment.com



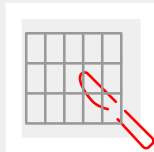
# Hidden States 2000-2010



Next step: Try more states?

# Roadblocks & Future steps

## Training



How to properly split the data for time series?

Randomly or through K fold CV?

How to initialise so the fit from EM isn't stuck in local optima?

## Evaluation



How to evaluate the model?

Compare to other methods, which methods?

## Prediction



Is predicting returns or open/close adds more to the story ?

How to get a result that is useful or effective shows something interesting?

# *Thanks!*

*Any questions?  
suggestions?*





# Classifying Images with CNN

## An Intro to TensorFlow

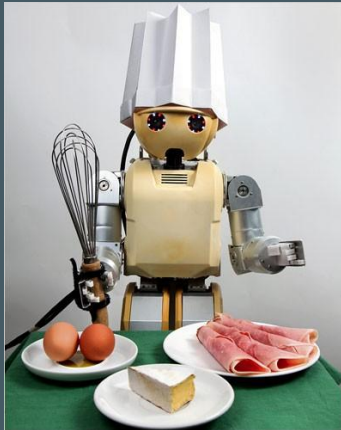
**Presenter: Alice Yang**

Why  TensorFlow™ ?

---

# Why TensorFlow™ ?

---



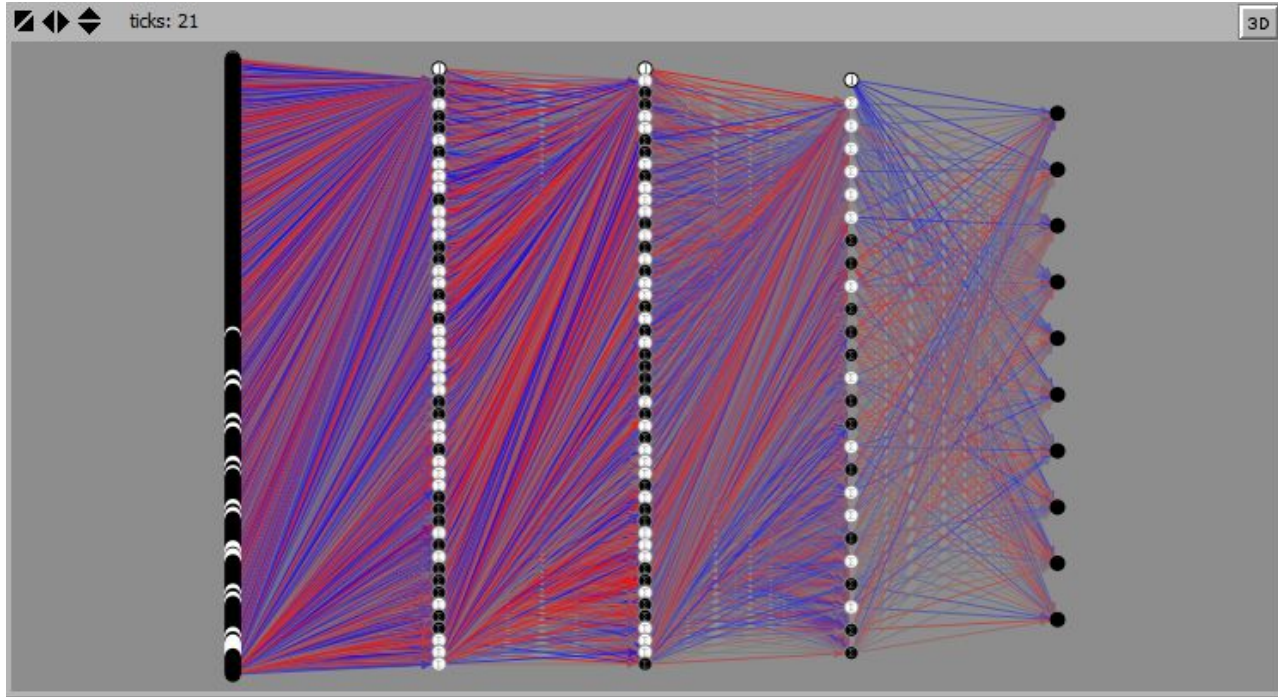


# Why TensorFlow™ ?

---



# What is TensorFlow™

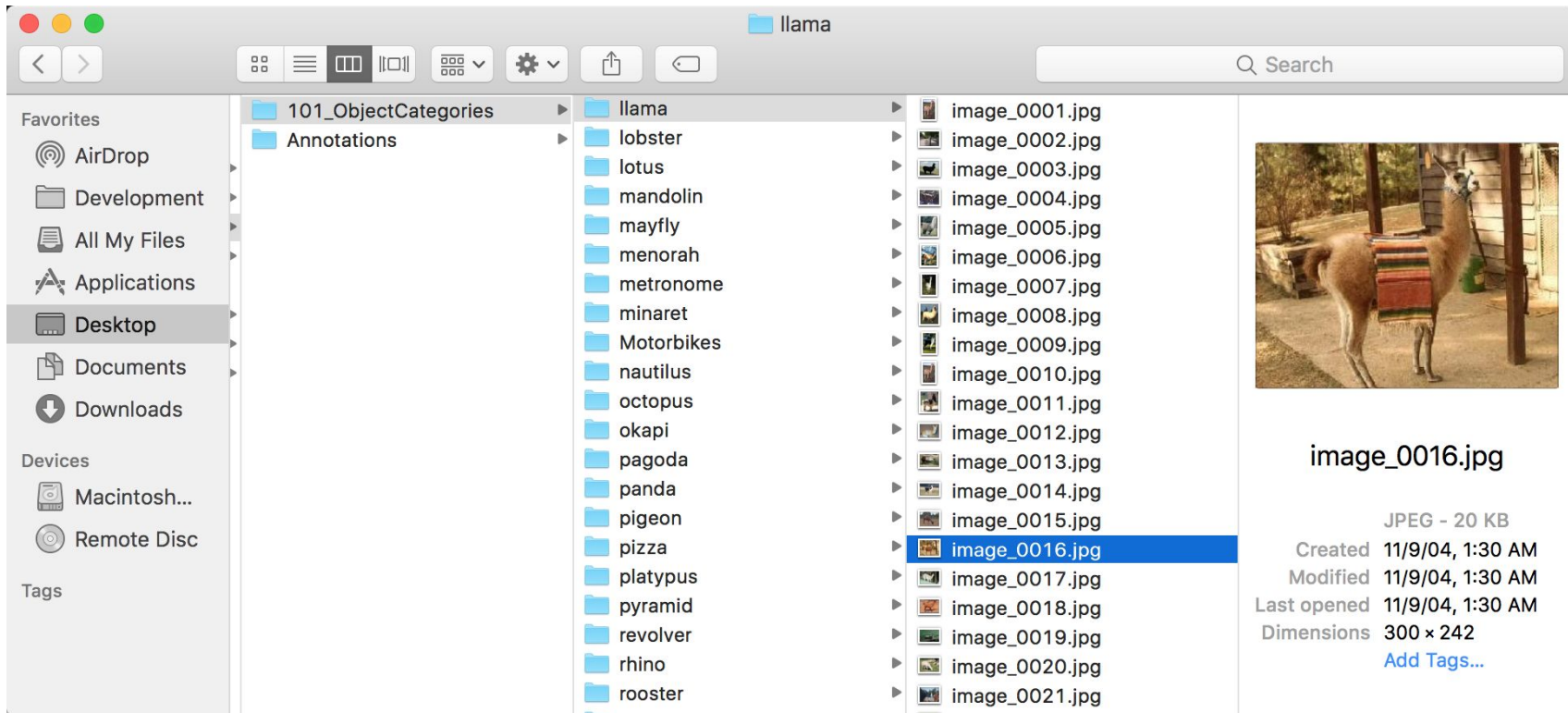




# CalTech-101

9,146 images

300 x 200 pixels



# Classify



Llama (78 pictures)



Giraffe (70 pictures)

---

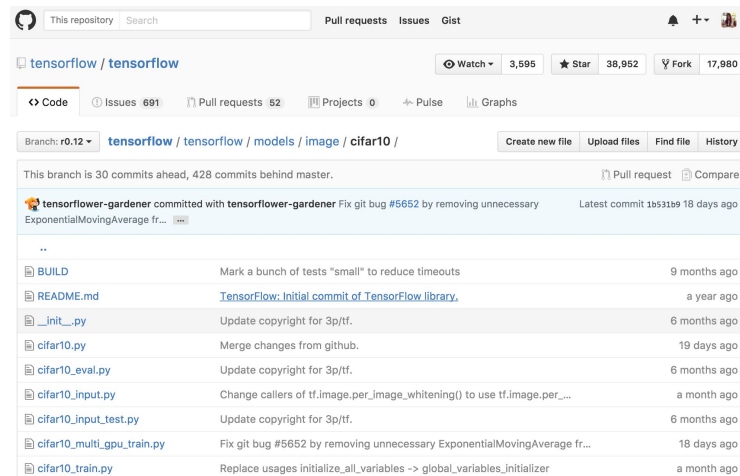


# Methods

## Code Organization

The code for this tutorial resides in [tensorflow/models/image/cifar10/](#).

File	Purpose
<a href="#">cifar10_input.py</a>	Reads the native CIFAR-10 binary file format.
<a href="#">cifar10.py</a>	Builds the CIFAR-10 model.
<a href="#">cifar10_train.py</a>	Trains a CIFAR-10 model on a CPU or GPU.
<a href="#">cifar10_multi_gpu_train.py</a>	Trains a CIFAR-10 model on multiple GPUs.
<a href="#">cifar10_eval.py</a>	Evaluates the predictive performance of a CIFAR-10 model.



How to classify RGB 32x32 pixel images across 10 categories:  
airplane, automobile, bird, cat, deer, dog, frog, horse, ship, and truck.

# Linear Regression in TensorFlow

```
1 import numpy as np
2 import tensorflow as tf
3
4 # Create fake data for  $y = Wx + b$  where  $W = 2$ ,  $b = 0$ 
5 x_data = np.array([[1, 2, 3, 4, 5, 6, 7, 8, 9, 10]])
6 y_data = np.array([[2, 4, 6, 8, 10, 12, 14, 16, 18, 20]])
7
8 # Model linear regression  $y = Wx + b$ 
9 x = tf.placeholder(tf.float32, [None, 1])
10 W = tf.Variable(tf.zeros([1, 1]))
11 b = tf.Variable(tf.zeros([1]))
12 product = tf.matmul(x, W)
13 y = product + b
14 y_ = tf.placeholder(tf.float32, [None, 1])
15
16 # Cost function  $\sum((y_-y)**2)$ 
17 cost = tf.reduce_mean(tf.square(y_-y))
18
19 # Training using Gradient Descent to minimize cost
20 train_step = tf.train.GradientDescentOptimizer(0.0000001).minimize(cost)
21
22 sess = tf.Session()
23 init = tf.initialize_all_variables()
24 sess.run(init)
25 steps = 1000
26 for i in range(steps):
27     # Train
28     feed = {x: x_data, y_: y_data}
29     sess.run(train_step, feed_dict=feed)
30
31     print("W: %f" % sess.run(W))
32     print("b: %f" % sess.run(b))
```

```
1 import numpy as np
2 from sklearn import linear_model
3
4 # Create fake data for  $y = Wx + b$  where  $W = 2$ ,  $b = 0$ 
5 x_data = np.array([[1, 2, 3, 4, 5, 6, 7, 8, 9, 10]])
6 y_data = np.array([[2, 4, 6, 8, 10, 12, 14, 16, 18, 20]])
7
8 # Train the linear regression
9 regression = linear_model.LinearRegression()
10 regression.fit(x_data, y_data)
11
12 # Predict
13 prediction = regression.predict(np.array([[12]]))
14 print(prediction)
```

# TensorObjects

- **Placeholder:** container vector that holds actual data values that are fed into the model when performing gradient descent
- **Variable:** coefficients that we are trying to find the values of



# TensorObjects

- **Placeholder:** container vector that holds actual data values that are fed into the model when performing gradient descent
- **Variable:** coefficients that we are trying to find the values of

```
8      # Model linear regression  $y = Wx + b$ 
9      x = tf.placeholder(tf.float32, [None, 1])
10     W = tf.Variable(tf.zeros([1, 1]))
11     b = tf.Variable(tf.zeros([1]))
12     product = tf.matmul(x, W)
13     y = product + b
14     y_ = tf.placeholder(tf.float32, [None, 1])
```

# TensorObjects

- **Cost Function:** objective function to minimize, ex. Sum of squares
- **Gradient Descent**

# TensorObjects

- Cost Function
- Gradient Descent

```
16     # Cost function sum((y_-y)**2)  
17     cost = tf.reduce_mean(tf.square(y_-y))  
18  
19     # Training using Gradient Descent to minimize cost  
20     train_step = tf.train.GradientDescentOptimizer(0.0000001).minimize(cost)
```

# Sessions

```
22     sess = tf.Session()  
23     init = tf.initialize_all_variables()  
24     sess.run(init)
```

# Actual Training

```
26     for i in range(steps):
27         # Train
28         feed = {x: x_data, y_: y_data}
29         sess.run(train_step, feed_dict=feed)
30
31         print("W: %f" % sess.run(W))
32         print("b: %f" % sess.run(b))
```

# Linear Regression in TensorFlow

```
1 import numpy as np
2 import tensorflow as tf
3
4 # Create fake data for  $y = Wx + b$  where  $W = 2$ ,  $b = 0$ 
5 x_data = np.array([[1, 2, 3, 4, 5, 6, 7, 8, 9, 10]])
6 y_data = np.array([[2, 4, 6, 8, 10, 12, 14, 16, 18, 20]])
7
8 # Model linear regression  $y = Wx + b$ 
9 x = tf.placeholder(tf.float32, [None, 1])
10 W = tf.Variable(tf.zeros([1, 1]))
11 b = tf.Variable(tf.zeros([1]))
12 product = tf.matmul(x, W)
13 y = product + b
14 y_ = tf.placeholder(tf.float32, [None, 1])
15
16 # Cost function  $\sum((y_-y)**2)$ 
17 cost = tf.reduce_mean(tf.square(y_-y))
18
19 # Training using Gradient Descent to minimize cost
20 train_step = tf.train.GradientDescentOptimizer(0.0000001).minimize(cost)
21
22 sess = tf.Session()
23 init = tf.initialize_all_variables()
24 sess.run(init)
25 steps = 1000
26 for i in range(steps):
27     # Train
28     feed = {x: x_data, y_: y_data}
29     sess.run(train_step, feed_dict=feed)
30
31     print("W: %f" % sess.run(W))
32     print("b: %f" % sess.run(b))
```

```
1 import numpy as np
2 from sklearn import linear_model
3
4 # Create fake data for  $y = Wx + b$  where  $W = 2$ ,  $b = 0$ 
5 x_data = np.array([[1, 2, 3, 4, 5, 6, 7, 8, 9, 10]])
6 y_data = np.array([[2, 4, 6, 8, 10, 12, 14, 16, 18, 20]])
7
8 # Train the linear regression
9 regression = linear_model.LinearRegression()
10 regression.fit(x_data, y_data)
11
12 # Predict
13 prediction = regression.predict(np.array([[12]]))
14 print(prediction)
```

# Multiple Regression in TensorFlow?

```
1 import numpy as np
2 import tensorflow as tf
3
4 # Create fake data for  $y = Wx + b$  where  $W = 2$ ,  $b = 0$ 
5 x_data = np.array([[1, 2, 3, 4, 5, 6, 7, 8, 9, 10]])
6 y_data = np.array([[2, 4, 6, 8, 10, 12, 14, 16, 18, 20]])
7
8 # Model linear regression  $y = Wx + b$ 
9 x = tf.placeholder(tf.float32, [None, 1])
10 W = tf.Variable(tf.zeros([1, 1]))
11 b = tf.Variable(tf.zeros([1]))
12 product = tf.matmul(x, W)
13 y = product + b
14 y_ = tf.placeholder(tf.float32, [None, 1])
15
16 # Cost function  $\sum((y_-y)**2)$ 
17 cost = tf.reduce_mean(tf.square(y_-y))
18
19 # Training using Gradient Descent to minimize cost
20 train_step = tf.train.GradientDescentOptimizer(0.0000001).minimize(cost)
21
22 sess = tf.Session()
23 init = tf.initialize_all_variables()
24 sess.run(init)
25 steps = 1000
26 for i in range(steps):
27     # Train
28     feed = {x: x_data, y_: y_data}
29     sess.run(train_step, feed_dict=feed)
30
31     print("W: %f" % sess.run(W))
32     print("b: %f" % sess.run(b))
```

```
1 import numpy as np
2 from sklearn import linear_model
3
4 # Create fake data for  $y = Wx + b$  where  $W = 2$ ,  $b = 0$ 
5 x_data = np.array([[1, 2, 3, 4, 5, 6, 7, 8, 9, 10]])
6 y_data = np.array([[2, 4, 6, 8, 10, 12, 14, 16, 18, 20]])
7
8 # Train the linear regression
9 regression = linear_model.LinearRegression()
10 regression.fit(x_data, y_data)
11
12 # Predict
13 prediction = regression.predict(np.array([[12]]))
14 print(prediction)
```

# Converting Images to Correct Input

```
# Make a queue of file names including all the JPEG images files in the  
# relative image directory.  
filename_queue = tf.train.string_input_producer(  
    tf.train.match_filenames_once("./llamas/*.jpg"))  
  
# Read an entire image file which is required since they're JPEGs, if the image  
# are too large they could be split in advance to smaller files or use the  
# Fixed reader to split up the file.  
image_reader = tf.WholeFileReader()  
  
# Read a whole file from the queue, the first returned value in the tuple is  
# the filename which we are ignoring.  
_, image_file = image_reader.read(filename_queue)  
  
# Decode the image as a JPEG file, this will turn it into a Tensor which we can  
# then use in training.  
image = tf.image.decode_jpeg(image_file)
```



# Results

- # Correctly Classified llamas
- # Correctly Classified Giraffes

# Implmenting + Future Work

- Convert jpgs to tensor objects
  - Cleaning images
  - Converting images
- Run sample CNN code on my data
  - Modify dimensionality of layers
- Modify parameters
  - Add/Remove layers
  - Different combinations of filtering, pooling, activations