



Pivotal

Josh Plotkin's Job Talk

February 6, 2017

Overview

- About me
- Problem Statement
- Demo
- Convolutional Neural Nets
- Transfer Learning
- Visualizing ConvNet for women's boots
- Model selection
- Adding intelligence to image serving
- “Pivotalizing”
- Demo again
- Repo: https://github.com/joshplotkin/convnet_transfer_learning

About me

- Pivotal DE since March '15, on Prasad's pod
- Hoping to become a Pivotal DS
- Graduated May '16 with MS in CS from CU (majority of courses in ML and stats, particularly Bayesian stats)
- Have lived in NYC since 2002
- Poker

Problem Statement

- How can we leverage a Convolutional Neural Network to engineer image features?
- How can we do this without 2 weeks and GPUs?
- What else can we do with these features?
 - Image similarities on boots
 - Recommend boots based on small amounts of training data (likes/dislikes)
- How can we apply this to our jobs and for our customers, possibly using Pivotal software?



Deep Learning!



Demo

- Group project at Columbia for *Advanced Machine Learning* course, Spring '16 with Dave Greenfield, Erin McMahon, Priya Venkat
- Source of data:
 - Amazon
 - 4096-length feature vectors extracted by Caffe
 - <http://jmcauley.ucsd.edu/data/amazon/>
 - python — download selected images from amazon
 - python — extract female boots based on descriptions and metadata
- Source paper: <http://cseweb.ucsd.edu/~jmcauley/pdfs/sigir15.pdf>
- ~2 week project, ~5 min presentation
- I've added TensorFlow parts, visualizations, and deck

Convolutional Neural Nets (aka ConvNet or CNN)

- Use cases?
- How do they work?
- Parallelism
- Hyper-parameters
- The “magic”
- Transfer learning

Convolutional Neural Nets

Use cases

- Image processing
- Video processing
- Learning games, e.g. AlphaGo (combined with search algorithm)
- Semantic analysis, topic modeling
 - Compare to Recurrent Neural Networks

more on CNNs for NLP: https://www.microsoft.com/en-us/research/wp-content/uploads/2014/10/604_Paper.pdf
https://www.microsoft.com/en-us/research/wp-content/uploads/2016/02/cikm2014_cdssm_final.pdf

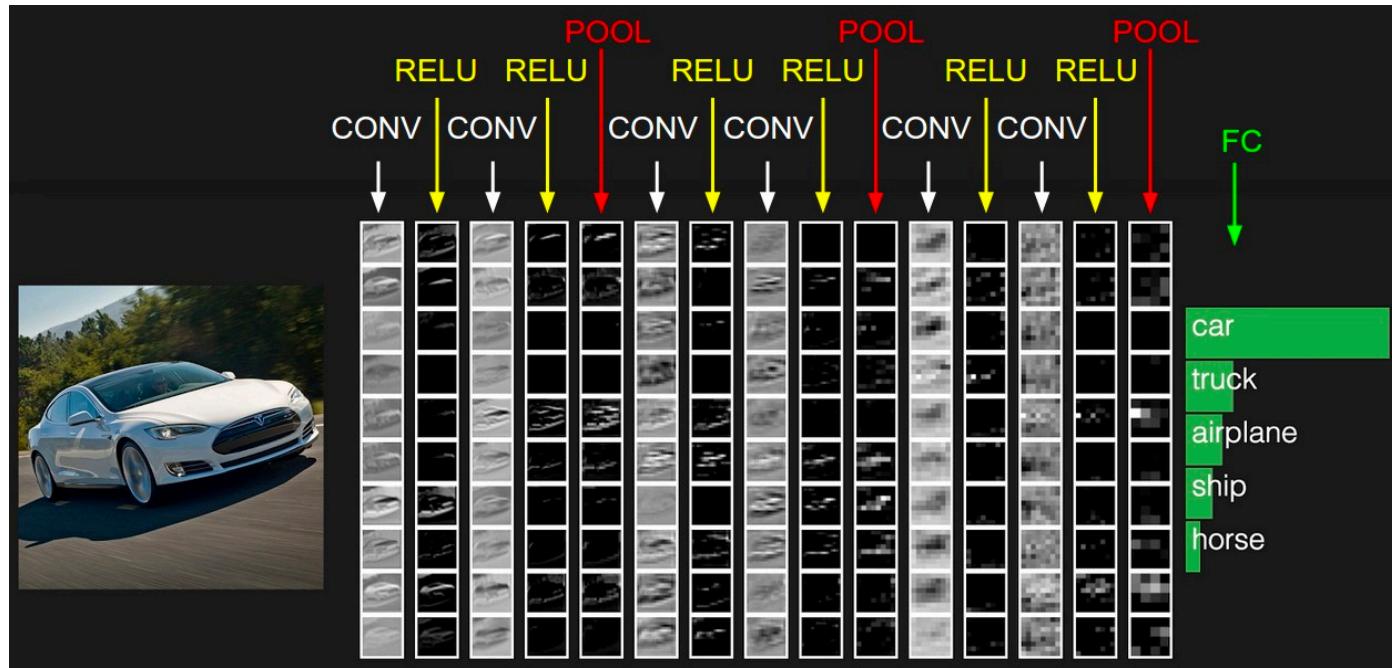
Convolutional Neural Nets

Layers

- Convolution
- ReLU
- Pooling
- Fully connected
- Note: all layers 3D (depth is filters, or RGB at the start)

Convolutional Neural Nets

Layers



source: <http://cs231n.github.io/convolutional-networks/>

Convolutional Neural Nets

Convolution

- Filters
 - learned features
 - provide activation function
- Early filters pick up edges; later filters pick up more complex features
- For each learned filter
 - For each patch in a sliding window
 - Elementwise multiplication
 - Sum
- Read more about image kernels: <http://setosa.io/ev/image-kernels/>

	$\begin{bmatrix} 1 & 0 & -1 \\ 0 & 0 & 0 \\ -1 & 0 & 1 \end{bmatrix}$	
Edge detection	$\begin{bmatrix} 0 & 1 & 0 \\ 1 & -4 & 1 \\ 0 & 1 & 0 \end{bmatrix}$	
	$\begin{bmatrix} -1 & -1 & -1 \\ -1 & 8 & -1 \\ -1 & -1 & -1 \end{bmatrix}$	
Sharpen	$\begin{bmatrix} 0 & -1 & 0 \\ -1 & 5 & -1 \\ 0 & -1 & 0 \end{bmatrix}$	

Convolutional Neural Nets

Convolution — Toy example

- This single filter/feature/kernel is: $[[1,0,1], [0,1,0], [1,0,1]]$
- This CONV layer will have many such filters
- Stride=1
- No padding

1	1	1	0	0
0	1	1	1	0
0	0	1	1	1
0	0	1	1	0
0	1	1	0	0

Image

4		

Convolved Feature



1	1	1	0	0
0	1	1	1	0
0	0	1	1	1
0	0	1	1	0
0	1	1	0	0

Image

4	3	

Convolved Feature



1	1	1	0	0
0	1	1	1	0
0	0	1	1	1
0	0	1	1	0
0	1	1	0	0

Image

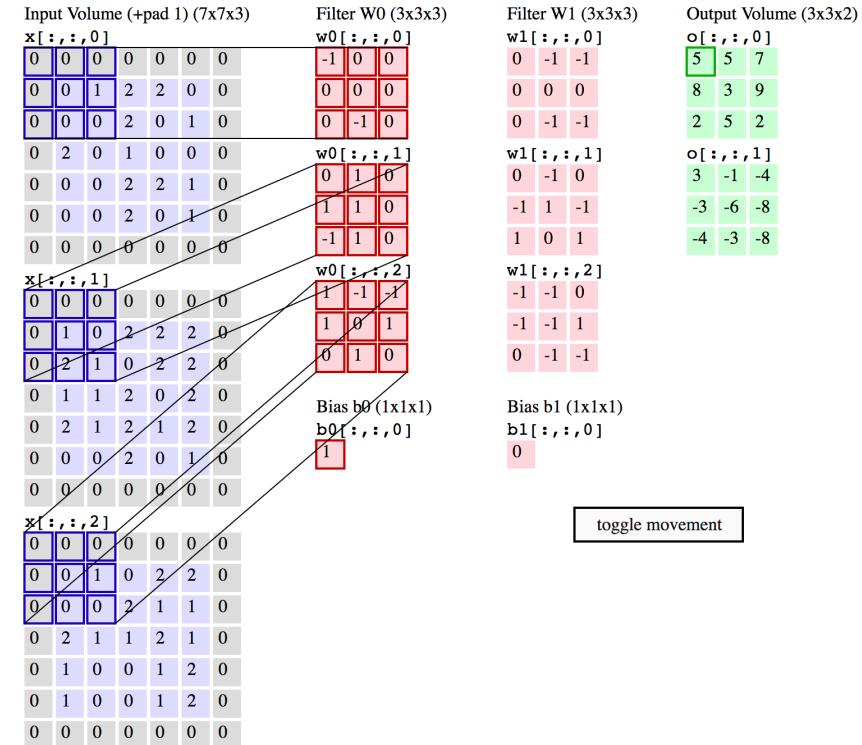
4	3	4
2	4	3
2	3	4

Convolved Feature

Convolutional Neural Nets

Convolution — More realistic example

- 3x3x3 filter (RGB, or from 3 filters)
- Zero-Padding in gray (not part of input)
- Stride=2 (yields 3x3)
- Green matrices are neurons
- Calculations on boxed areas
 - Top=0, Middle=3, Bottom=1
 - Bias=1
 - Total=5



Convolutional Neural Nets

Convolution — Learned Filters/Features/Activations



- From an early layer (picking up edges)
- General features — apply to all parts of image

source: <http://cs231n.github.io/convolutional-networks/>

Convolutional Neural Nets

Convolution Filters — 2nd layer and 3rd layer of CONV

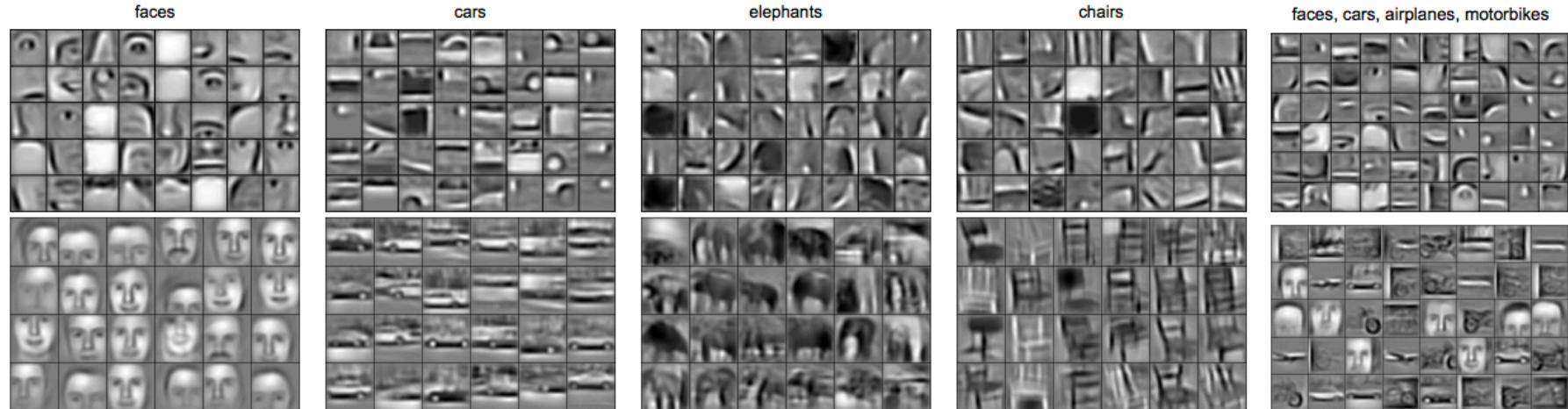
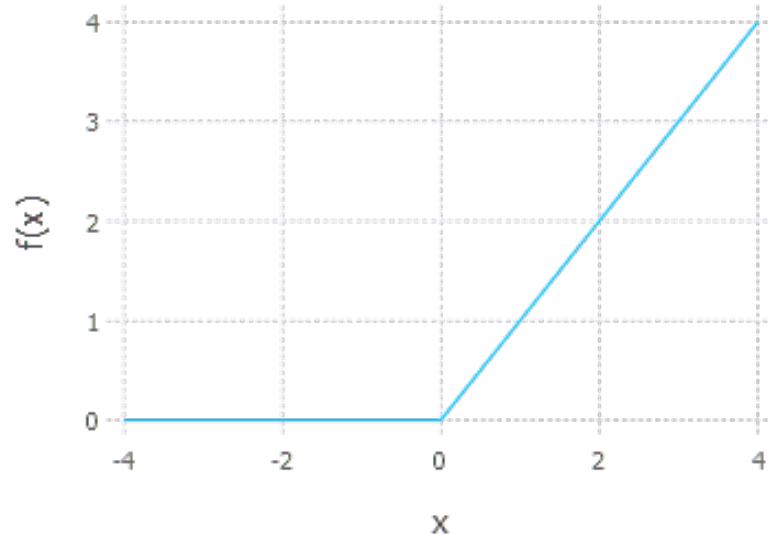


Figure 3. Columns 1-4: the second layer bases (top) and the third layer bases (bottom) learned from specific object categories. Column 5: the second layer bases (top) and the third layer bases (bottom) learned from a mixture of four object categories (faces, cars, airplanes, motorbikes).

Convolutional Neural Nets

ReLU — Rectified Linear Units

- Compared to sigmoid
 - Faster training
 - Similar results

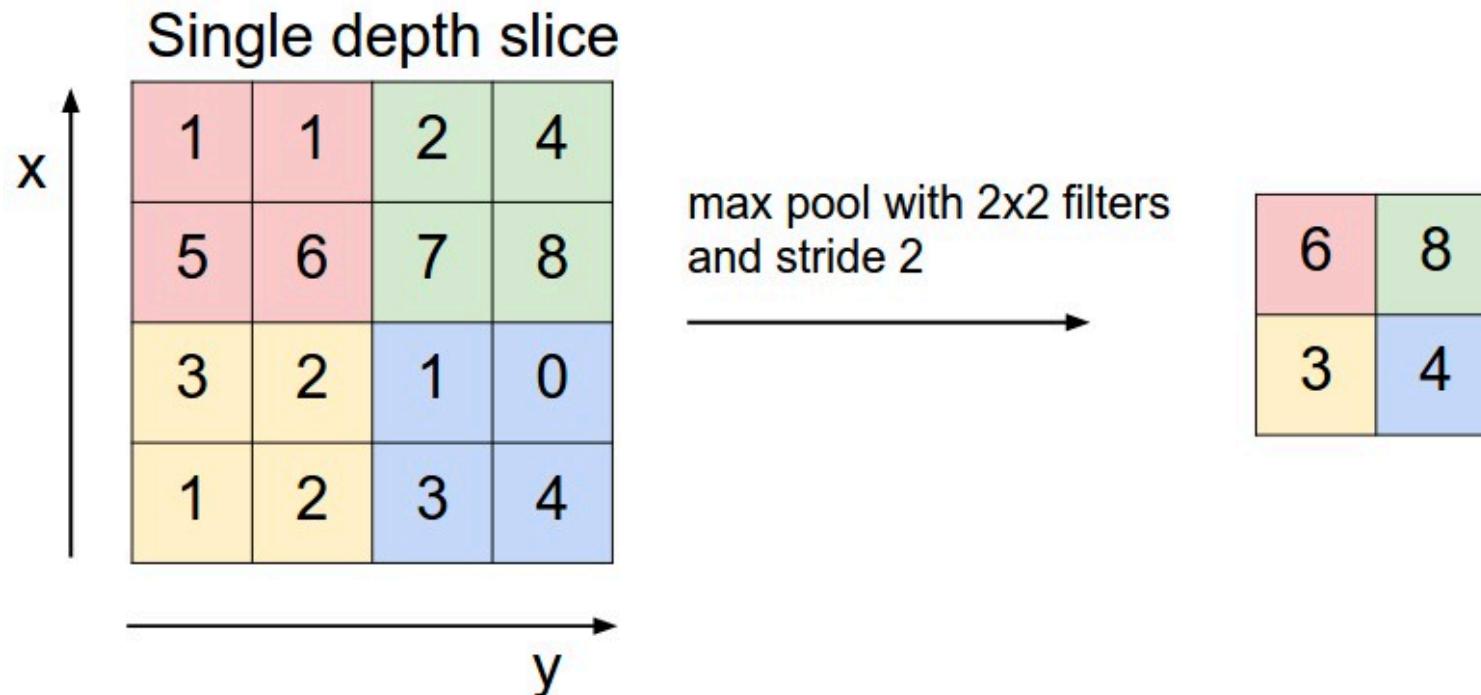


$$f(x) = \max(0, x),$$

source: [https://en.wikipedia.org/wiki/Rectifier_\(neural_networks\)](https://en.wikipedia.org/wiki/Rectifier_(neural_networks))

Convolutional Neural Nets

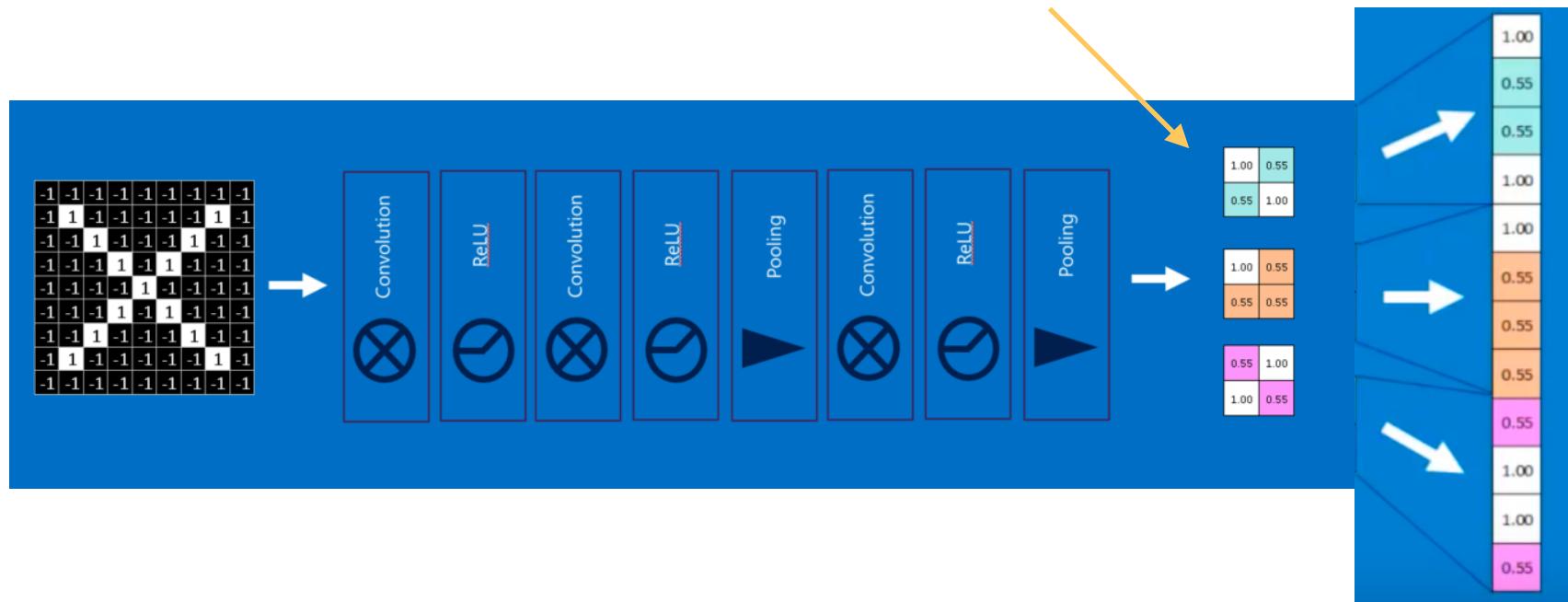
Pooling



Convolutional Neural Nets

Stacking the layers

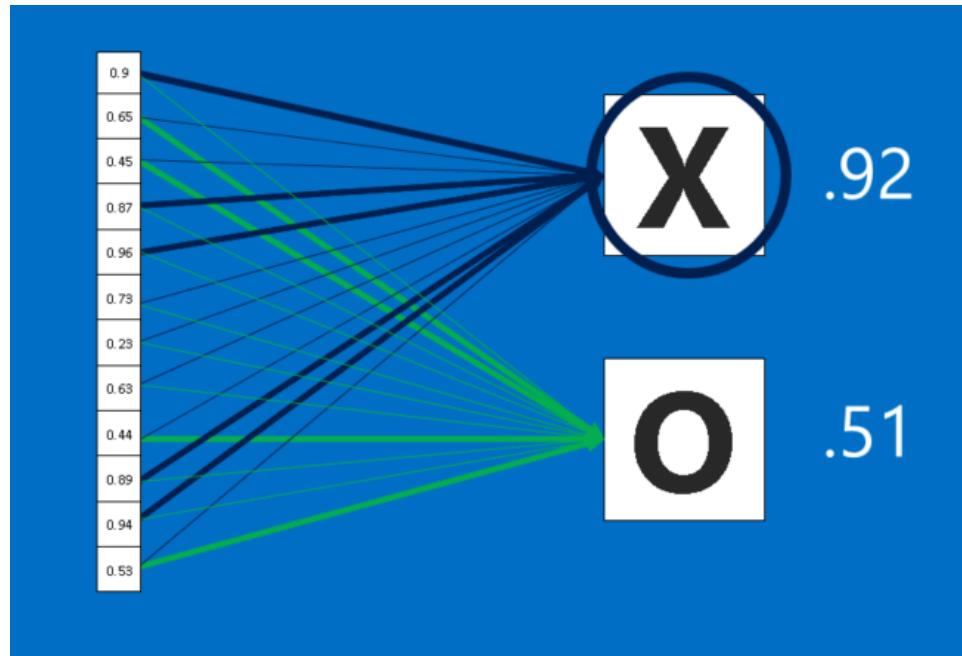
3 filters in convolution



Convolutional Neural Nets

Fully Connected Layers — Scoring

- Perform dot product
 - feature vector
 - learned weights
- Pass through function
 - sigmoid
 - softmax
 - none (linear regression)
- Or extract feature vector
 - Any supervised model
 - e.g. SVM



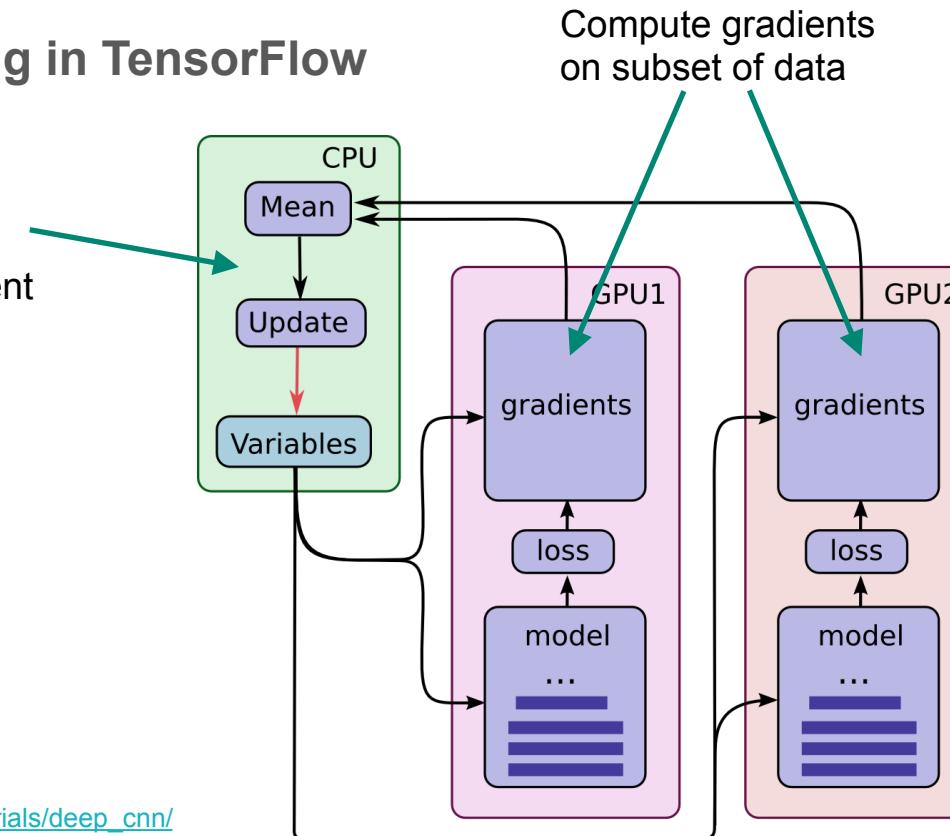
source: http://brohrer.github.io/how_convolutional_neural_networks_work.html

Convolutional Neural Nets

Parallelized Training in TensorFlow

- Backprop
- Average gradients
- Perform Gradient Descent update step:

$$\theta = \theta - \alpha \nabla_{\theta} J(\theta; x^{(i)}, y^{(i)})$$



source: https://www.tensorflow.org/tutorials/deep_cnn/

source: <http://ufldl.stanford.edu/tutorial/supervised/OptimizationStochasticGradientDescent/>

code: https://github.com/tensorflow/models/blob/master/inception/inception_train.py

backprop for CNNs: <http://www.jefkine.com/general/2016/09/05/backpropagation-in-convolutional-neural-networks/>

Convolutional Neural Nets

Parameters and Hyperparameters

- Hyperparameters
 - Size of filter (kernel size)
 - Number of filters
 - Size of filter stride
 - Pooling (size and stride)
 - Padding
 - How to stack the layers
 - Dropout proportion
 - (S)GD learning rate and SGD batch size
- Parameters
 - Pixel values in filter (kernel)
 - Weights for FC layer(s)

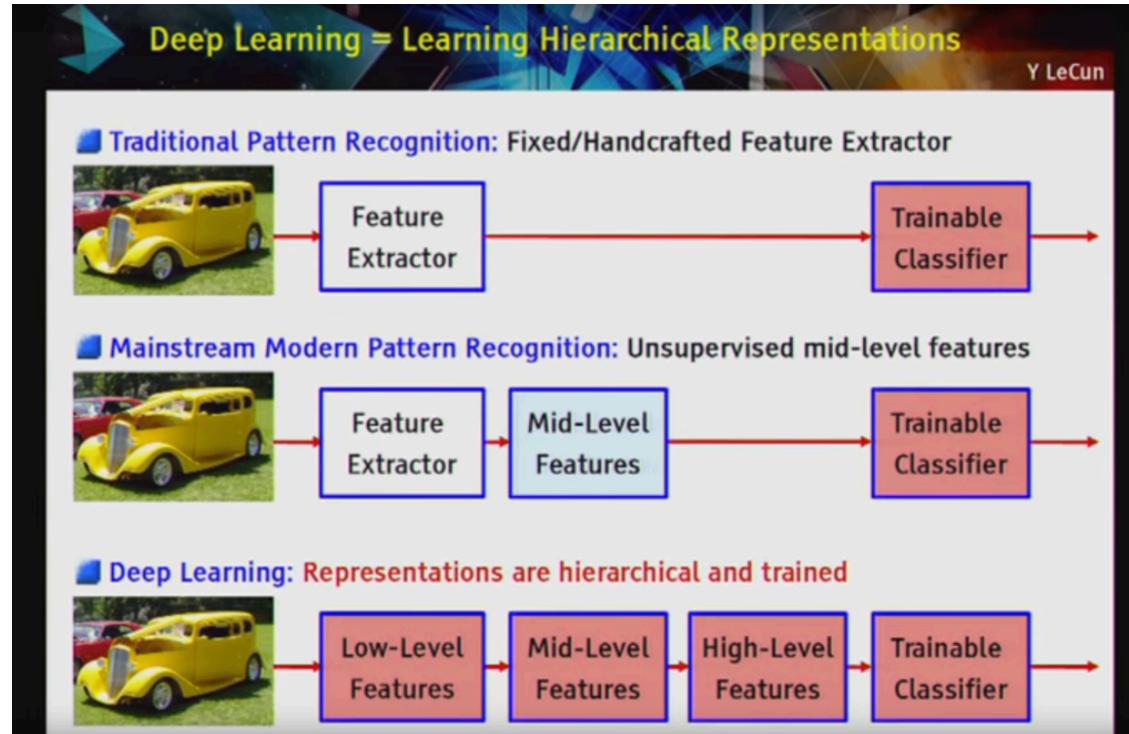
Convolutional Neural Nets

The Magic

- Invariant to location in image (no need to crop)
 - Trainable
 - Secondary benefit of generalized features, and pooling)
- Only one set of filters needs to be learned for each CONV layer
 - Simultaneously scale it down from e.g. $300 \times 300 \times 3 = 270k$ pixels to 4096 or 2048 learned features, and project them into a higher space
- Familiar backprop+gradient descent to solve
- Train in parallel (usually on GPU)
- Transfer learning
- Many frameworks for many languages

Convolutional Neural Nets

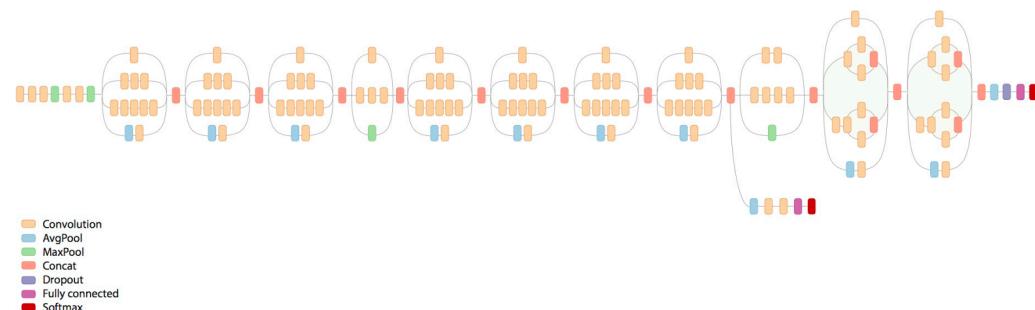
Summary



Convolutional Neural Nets

Transfer Learning

- Start with a pre-trained model
 - Train on ImageNet corpus (1M images, 1K categories)
 - Google: “We can train a model from scratch to its best performance on a desktop with 8 NVIDIA Tesla K40s in about 2 weeks.” <https://research.googleblog.com/2016/03/train-your-own-image-classifier-with.html>
 - GoogLeNet won 2014 ImageNet
- Hold filters fixed
- Provide new images, with class labels
- Re-train final layer(s)
- Ensembles in recent years

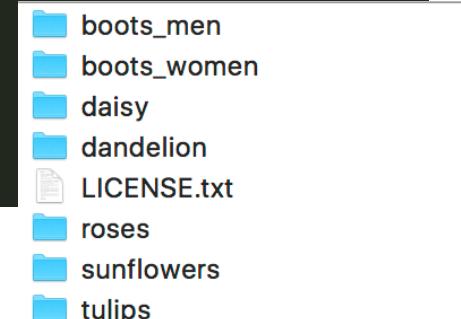


TensorFlow

Transfer Learning

- Tutorial on retraining the Inception model: https://www.tensorflow.org/how_tos/image_retraining/
- Directories hold labels (silly example)
- Training on ~12K images took me <1 hour
- TF performs cross validation
- output_graph.pb is serialized model

```
bazel build -c opt --copt=-mavx tensorflow/examples/image_retraining:retrain
python tensorflow/examples/image_retraining/retrain.py \
    --image_dir /tmp/pics \
    --output_graph /tmp/output_graph.pb \
    --output_labels /tmp/output_labels.txt \
    --bottleneck_dir /tmp/bottleneck
```



TensorFlow

Using Trained Model

- Starter code here: [https://github.com/eldor4do/TensorFlow-Examples/
blob/master/retraining-example.py](https://github.com/eldor4do/TensorFlow-Examples/blob/master/retraining-example.py)

```
def create_graph():
    """Creates a graph from saved GraphDef file and returns a saver."""
    # Creates graph from saved graph_def.pb.
    with tf.gfile.FastGFile(modelFullPath, 'rb') as f:
        graph_def = tf.GraphDef()
        graph_def.ParseFromString(f.read())
        _ = tf.import_graph_def(graph_def, name='')
```

Load graph

```
# make image category prediction
softmax_tensor = sess.graph.get_tensor_by_name('final_result:0')
predictions = sess.run(softmax_tensor, {'DecodeJpeg/contents:0': image_data})
predictions = np.squeeze(predictions)
```

Get prediction

```
# extract features
feature_tensor = sess.graph.get_tensor_by_name('pool_3:0')
features = sess.run(feature_tensor, {'DecodeJpeg/contents:0': image_data})
features = np.squeeze(features)
```

Extract features

TensorFlow

Using Trained Model — scoring unseen samples



```
boots women (score = 0.75998)
boots men (score = 0.23904)
daisy (score = 0.00030)
tulips (score = 0.00026)
roses (score = 0.00020)
[Finished in 6.2s]
```

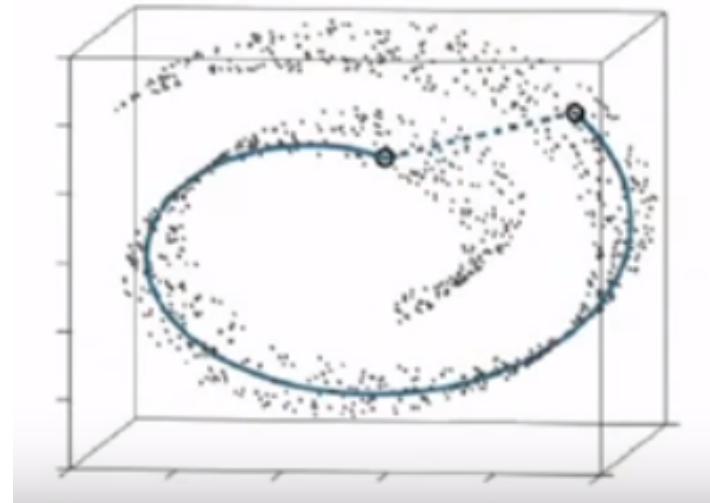


```
boots men (score = 0.87363)
boots women (score = 0.12485)
tulips (score = 0.00044)
daisy (score = 0.00033)
roses (score = 0.00032)
[Finished in 6.8s]
```

Visualizing ConvNet Features

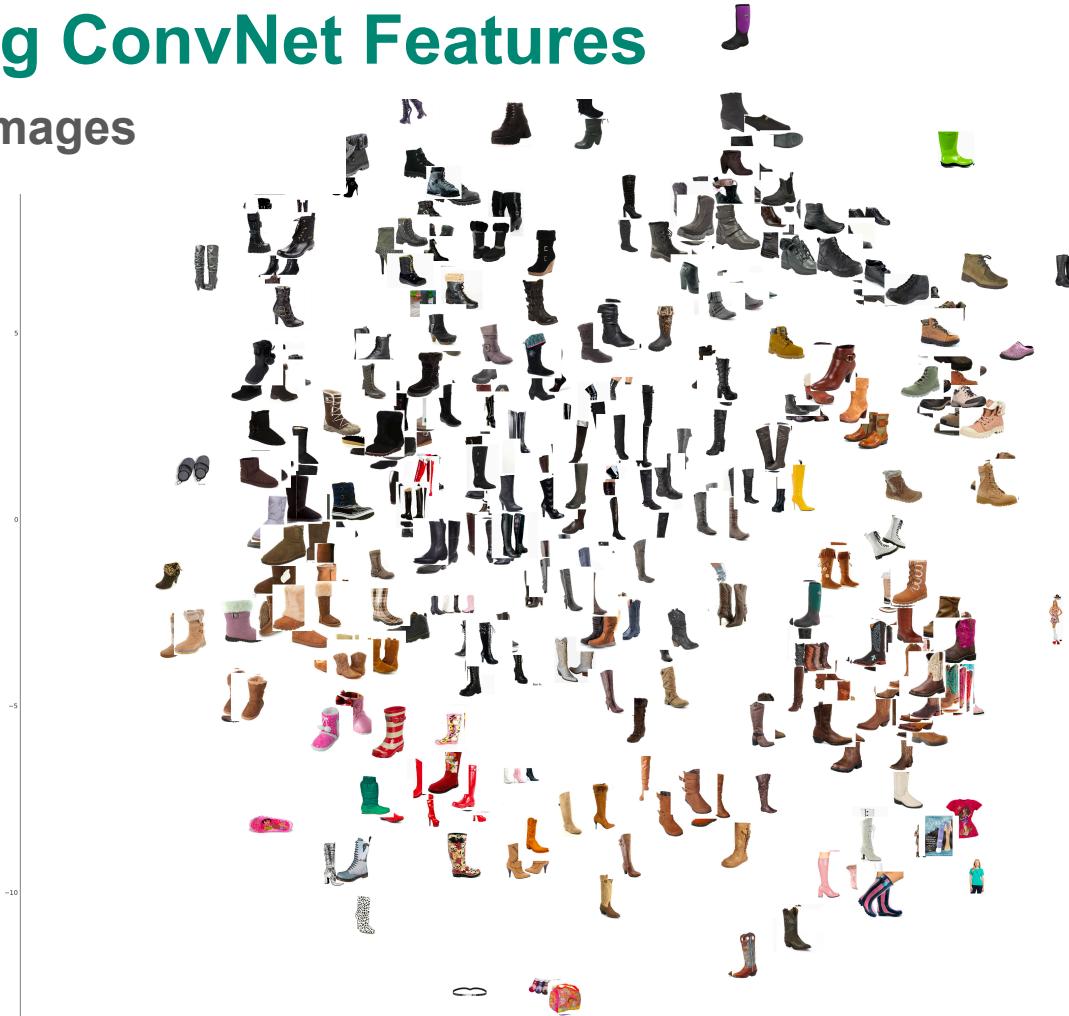
t-SNE

- t-Distributed Stochastic Neighborhood Embedding
- Winner of [Kaggle Data Viz Challenge](#)
- Dimensionality reduction technique
 - Non-linear (manifold learning) unlike PCA and LDA
 - What is the (approximate) distribution in 2-D space that is most similar to this distribution in 4096-D space?
- Minimizes KL-divergence
 - Maps points to probability distribution
 - Probability of being a given distance away
 - Gradient descent
 - from `sklearn.manifold import TSNE`



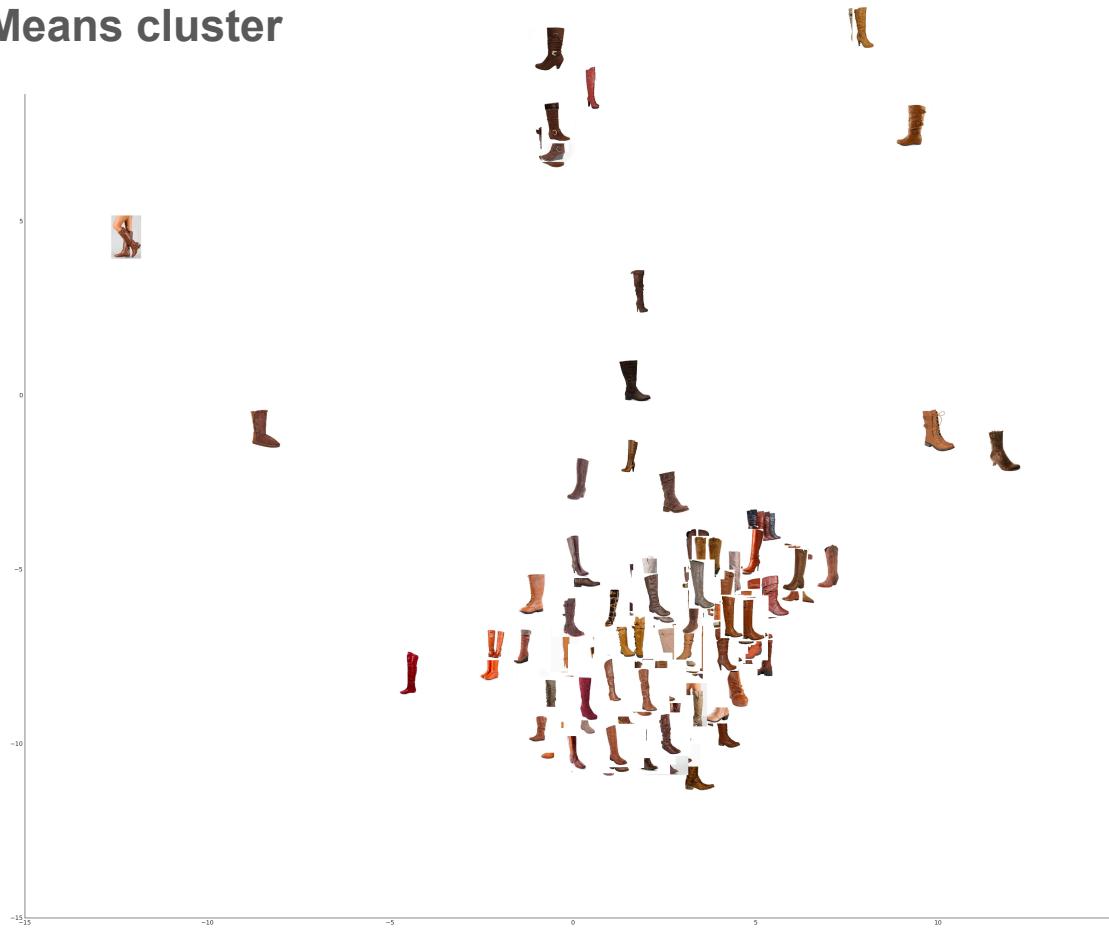
Visualizing ConvNet Features

t-SNE on all images



Visualizing ConvNet Features

t-SNE on one K-Means cluster



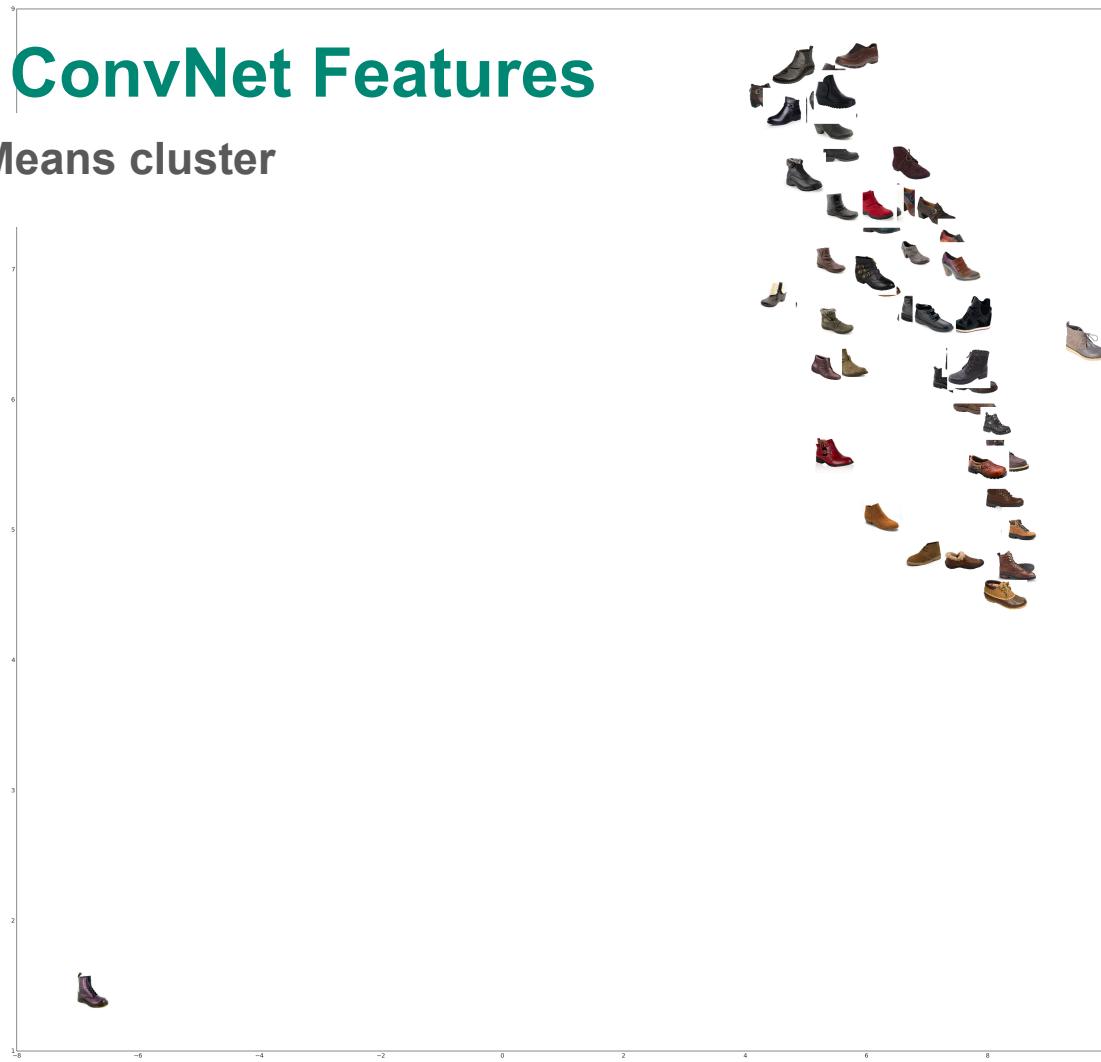
Visualizing ConvNet Features

t-SNE on one K-Means cluster



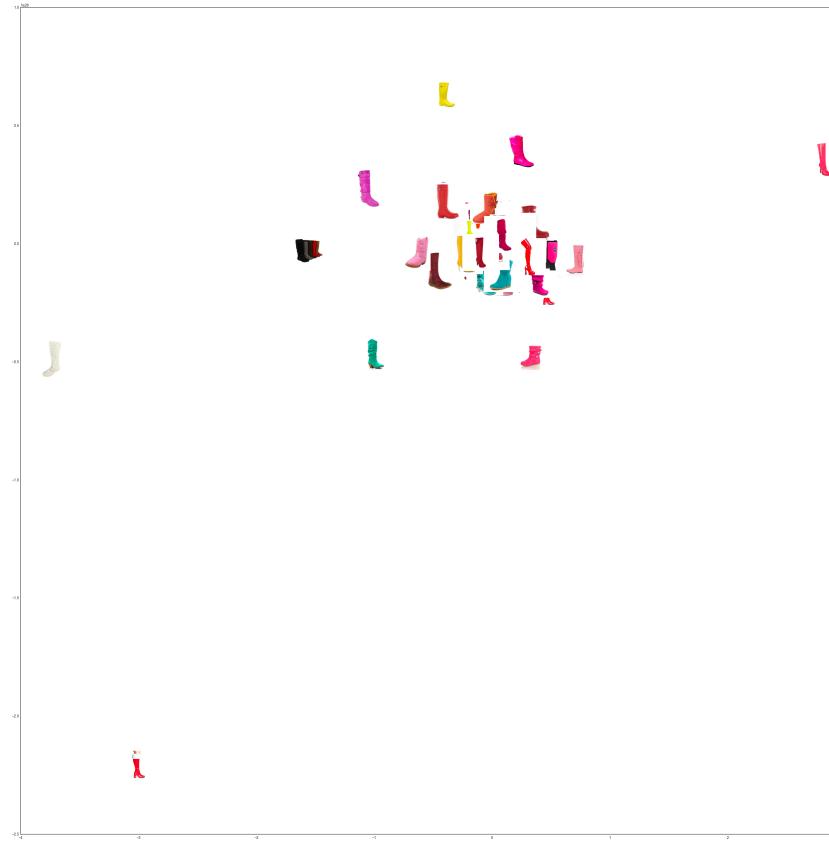
Visualizing ConvNet Features

t-SNE on one K-Means cluster



Visualizing ConvNet Features

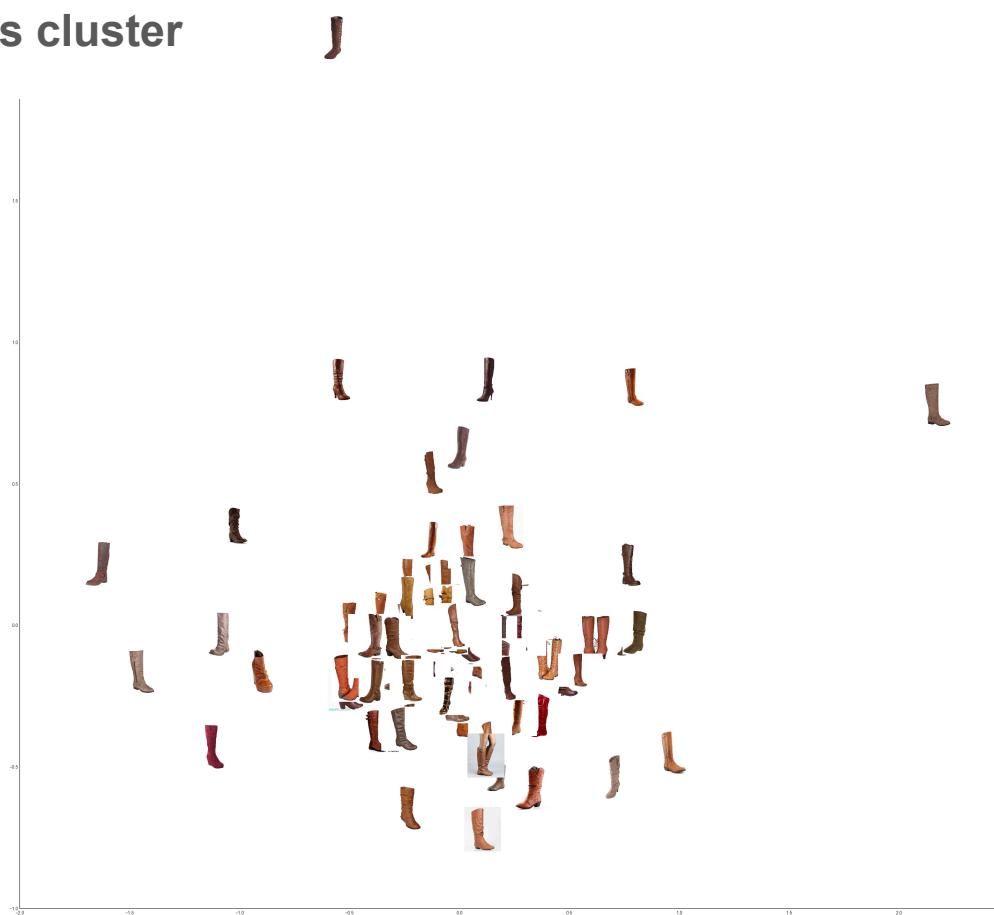
t-SNE on one K-Means cluster



Visualizing ConvNet Features

t-SNE on one K-Means cluster

J



Visualizing ConvNet Features

t-SNE on one K-Means cluster



Image Similarity

Approaches

- sklearn Nearest Neighbors
- sklearn pairwise cosine similarity
- Spotify's Annoy library for Approximate Nearest Neighbors
- All give similar results but using Annoy

Image Similarity



#1



#2



#3



#4



Image Similarity



Image Similarity



Image Similarity



Image Similarity



Model Selection

Training set

- Had friends/family use app for 5-10 mins as training samples
- Challenge 1: Amazon's boot selection is not ideal (few positive samples)
- Challenge 2: Different models/hyperparameters for different sample sizes

Model Selection

Methodology for classification

- Iterate over:
 - Training set sizes (20, 40, 60, 80)
 - Unbalanced/Oversampled
 - Replicated samples
 - Split into k-folds then balance within folds
 - Require each fold get a positive example
 - Models: XGB, RF, NB, SVM, Logistic regression (ENet w/ SGD updates)
 - Grid search over hyperparameters (5-fold CV)
- Mean performance over samples: AUC and speed
- Chose sklearn SGD classifier
 - Elastic net logistic regression
 - Hinge loss (SVM)
 - Online classifier — spread training time more evenly

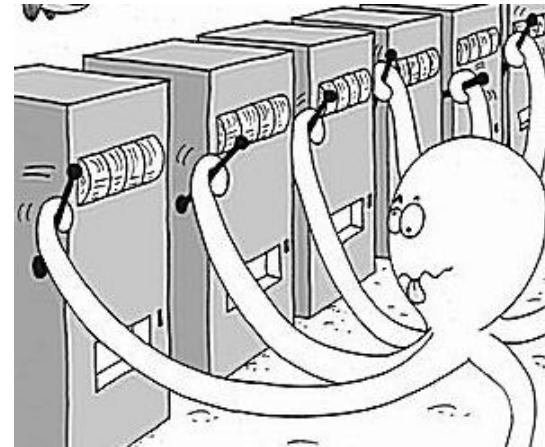
Model Selection

Future Ideas

- Implement batch training step when recommendation is requested
 - Allow to learn best model to use for different situations (class balance, sample size)
 - Use L1 subset selection from logistic regression to re-classify with fewer features
 - Ensemble of models to vote
 - Nearest neighbors when #likes < n
- Find better boots, or approach as anomaly detection problem
- Use aggregate likes/dislikes to understand trends and styles over time
- Hierarchical model — do inference on local and global user preference
- Incorporate metadata (reviews, similar items, Amazon recommendations)
- Generate more data (duh)
- Consider scalability

Adding Intelligence to Image Serving

- Treat it as multi-armed bandits problem
- Probability of a “like” in a given cluster
 - Clusters are independent
 - Modeled as binomial
 - Beta(1,1) (uniform) prior
 - After each up/down vote, posterior is updated
 - See Excel sheet
- Normalize probabilities, and sample from distribution
- Display boot with highest predicted probability
- Future improvements
 - Informative prior from aggregate user data
 - Hierarchical clustering



“Pivotalizing”

- PL/Python: parallelize TensorFlow feature extraction (no GPU needed)
... blog post?
- GemFire/PCC: caching user results, boot features, and nearest neighbors
- PCF: scaling app and back-end software
- GPDB/HAWQ: storing user interactions for further model selection

References

- <http://cs231n.github.io/convolutional-networks/>
- https://www.tensorflow.org/versions/master/tutorials/image_recognition/
- <https://github.com/tensorflow/models/tree/master/inception>
- <https://github.com/eldor4do/TensorFlow-Examples/blob/master/retraining-example.py>
- http://brohrer.github.io/how_convolutional_neural_networks_work.html
- <https://arxiv.org/pdf/1310.1531v1.pdf>
- <https://www.youtube.com/watch?v=RJVL80Gg3IA>
- <http://jmcauley.ucsd.edu/data/amazon/>
- <http://web.eecs.umich.edu/~honglak/icml09-ConvolutionalDeepBeliefNetworks.pdf>
- <http://colah.github.io/posts/2014-07-Conv-Nets-Modular/>