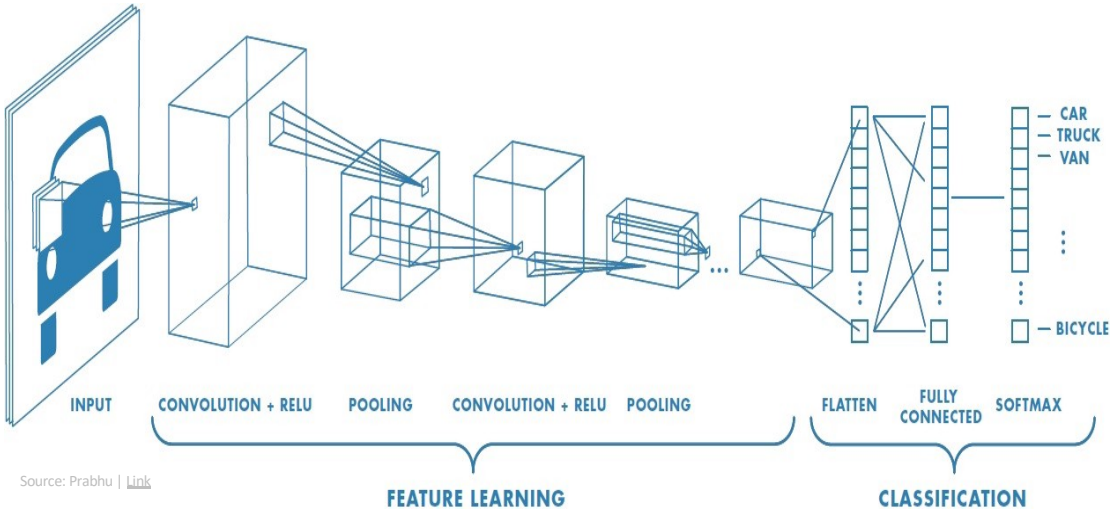
Deep Learning
Neural Networks

Dr. Ratna Babu Chinnam
Industrial & Systems Engineering
Wayne State University

“Deep Learning” Neural Networks

- **What**: Networks with lot of hidden layers ($\gg 1$) that extract complex features and can learn complex patterns and associations
- **Example**: Image Processing
  - 1st layer might detect edges
  - 2nd layer might group edges to detect longer contours
  - Deeper layers might group contours to detect even more complex features



Source: Prabhu | Link

- **Why**: k-layer network can represent compactly functions (with # of hidden units polynomial in # of inputs) that a $(k-1)$-layer network cannot represent without an exponentially large # of hidden units
  - Cortical computations (brain) also have multiple layers of processing

Primary Source: http://ufldl.stanford.edu

Difficulty Training Deep Architectures

WAYNE STATE UNIVERSITY

- Until recently we had little success training deep architectures
- Standard gradient descent methods do not usually work:
  - *Availability of Data*: Difficult to get enough labeled data to fit parameters of a complex model
  - *Local Optima*: Deep networks are vulnerable to bad local optima
  - *Diffusion of Gradients*: Gradients propagated backwards rapidly diminish in magnitude as depth of network increases
    - Called “diffusion of gradients” or “vanishing gradients” problem

Greedy Layer-wise Training

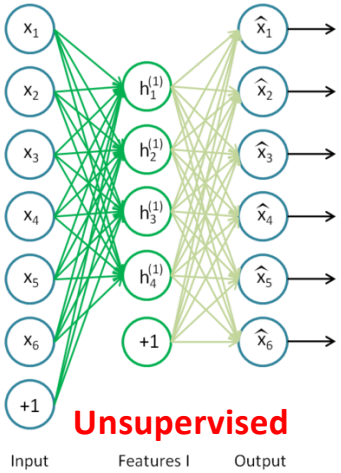
WAYNE STATE UNIVERSITY

- **Key Idea: Train layers of the network one at a time**
  - First train a network with 1 hidden layer, then train network with 2 hidden layers, and so on
  - At each step, we take old network with $k - 1$ hidden layers, and add an additional k-th hidden layer
    - Takes as input the output of hidden layer $k - 1$ that is just trained
- **Training: Can be supervised but more frequently it is unsupervised**
  - As in an autoencoder
- **Weights from greedy training are "fine-tuned"** in final deep network
  - Trained further together to optimize labeled training set error
- Success attributed to several factors:
  - **Availability of Unlabeled Data**: While labeled data can be expensive to obtain, unlabeled data can be cheap and plentiful.
  - **Better Local Optima**: After training on unlabeled data, weights are now starting at a better location in parameter space

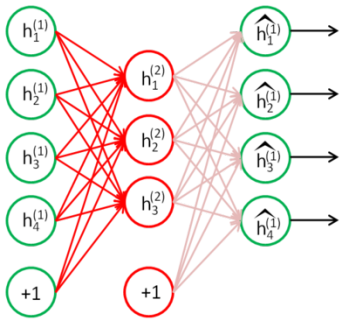
Stacked Autoencoders: Greedy Layer-wise Training

- **Stacked Autoencoder**: Network consisting of multiple layers of sparse autoencoders in which outputs of each layer is wired to inputs of successive layer

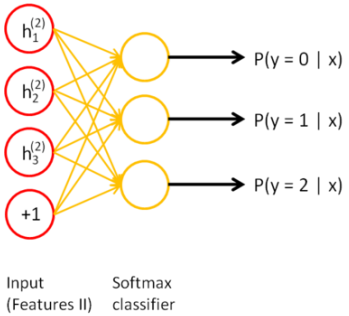
Example with 2 Stacked Autoencoders for Classification



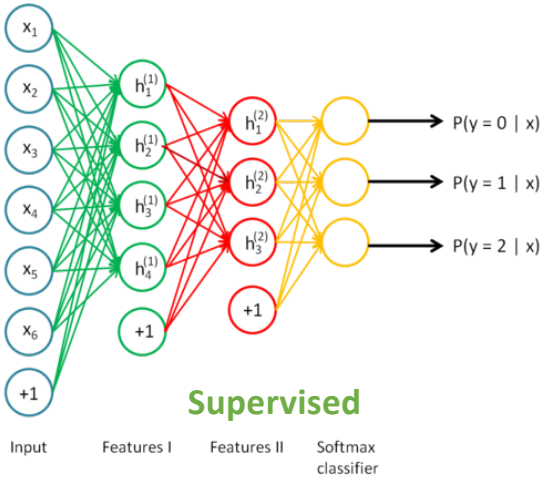
First, train a sparse autoencoder on raw inputs $x^{(k)}$ to learn "primary features" $h^{(1)(k)}$



Use "primary features" as "raw input" to another sparse autoencoder to learn secondary features $h^{(2)(k)}$



Treat secondary features as "raw input" to classifier



Combine all layers to form a "stacked" autoencoder and a final classifier layer for "fine tuning"

Autoencoders & Sparsity

WAYNE STATE UNIVERSITY

- Can we get hidden units to "specialize"?
  - Impose a "sparsity" constraint on hidden units
  - For any input, only few hidden units can be "active"
- Enforce constraint $\hat{p}_j \leq \rho$, where $\rho \approx 0$
  - Hidden unit's activations must mostly be near 0
- Add extra penalty term to optimization objective penalizing $\hat{p}_j$ deviating significantly from ρ
- Kullback-Leibler (KL) divergence measure allows backpropagation:

$$\sum_{j=1}^{s_2} \mathrm{KL}(\rho||\hat{\rho}_j) = \sum_{j=1}^{s_2} \rho \log \frac{\rho}{\hat{\rho}_j} + (1-\rho) \log \frac{1-\rho}{1-\hat{\rho}_j}.$$

- Need batch training to compute $\hat{p}_j$

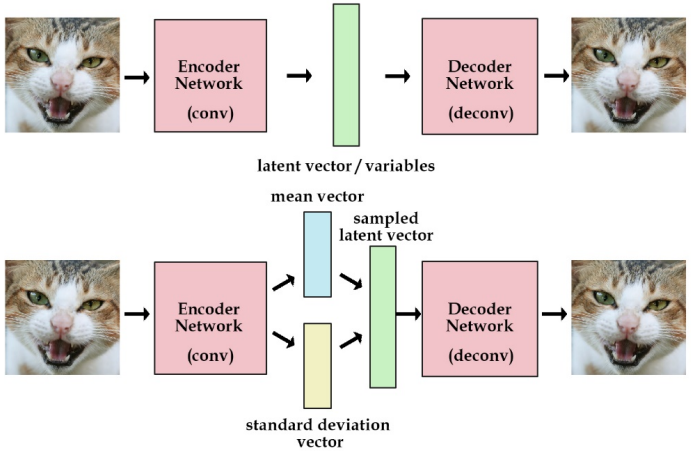
Constrain Avg. Activation of Hidden Unit *j*

$$\hat{\rho}_j = \frac{1}{m} \sum_{i=1}^{m} \left[a_j^{(2)}(x^{(i)}) \right]$$

Note: KL divergence measures how different two different distributions are

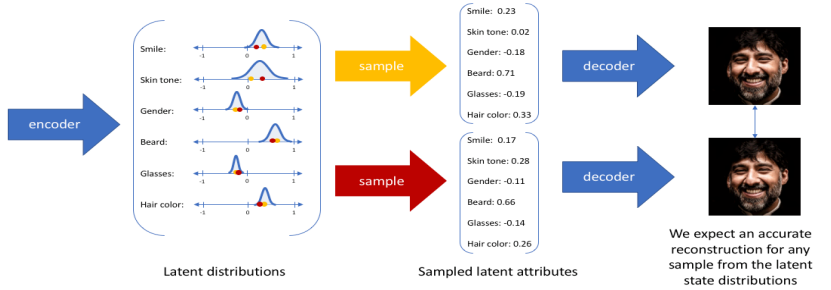
Variational Autoencoders (VAE)

- **Image Autoencoder Example:**
  - Takes original images and “encodes” them into “code” vectors
  - Decoder layers then "decode" vectors back to original images



- If we save code vector of an image, we can reconstruct it later by passing it through the decoder!
  - Cannot create “new” images; codes available only for “memorized” images
  - If you interpolate or disturb learnt code vectors, results are unpredictable
- **We need an effective “generative” model instead!**

Variational Autoencoders …

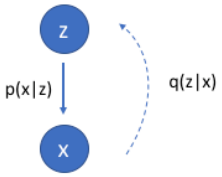


- **Solution**: Add a constraint on encoding that forces it to generate latent vectors that roughly follow a unit Gaussian
  - Generating new images is now easy: Simply sample a latent vector from unit gaussian and pass it into the decoder
- **Tradeoff**: Generative Loss (MSE image reconstruction accuracy) + Latent Loss (KL divergence measures how well latent variables match unit Gaussian)
- **Reparameterization Trick**: To optimize the KL divergence, instead of encoder generating the sample latent vector itself, it will generate a vector of means and a vector of standard deviations
  - Different means and standard deviations for different classes of images
- **Generalization**: In addition to allowing us to generate random latent variables, constraint also improves generalization of network!

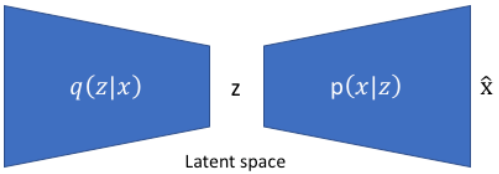
Variational Autoencoders: Details & MNIST Example

WAYNE STATE UNIVERSITY

Graphical Model



We'd like to use our observations to understand the hidden variable.

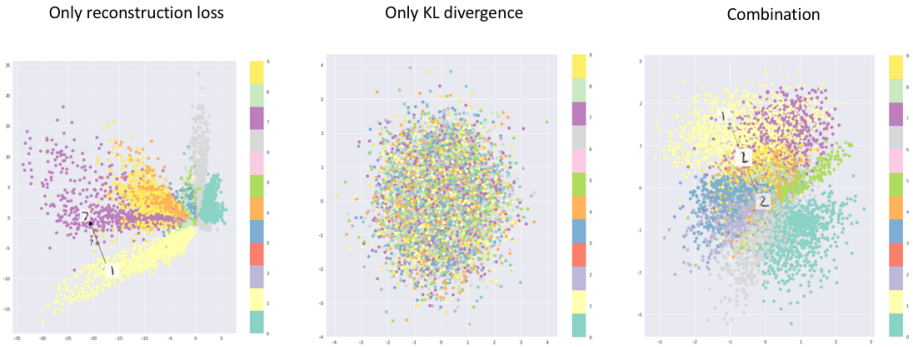


Latent space representation.

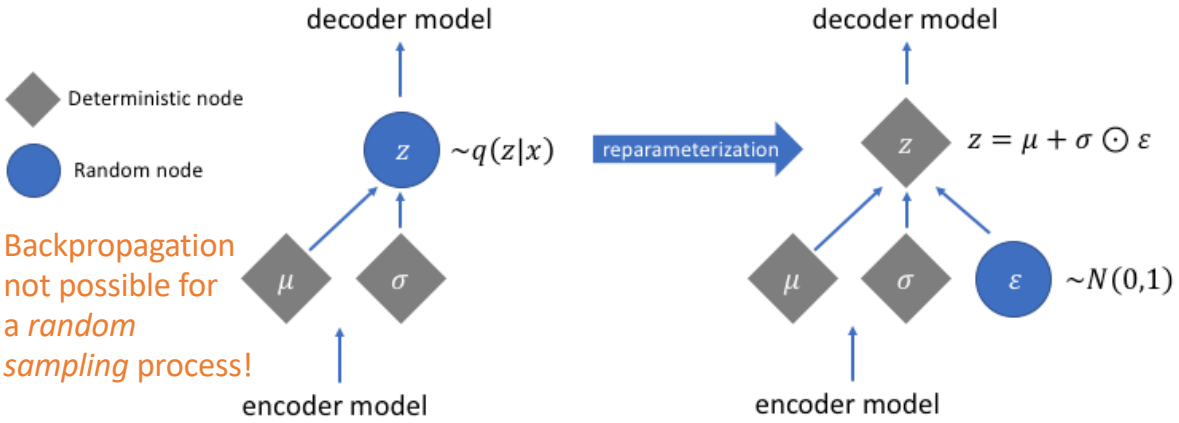
Neural network mapping x to z.

Neural network mapping z to x.

Visualizing 2-D Latent Space

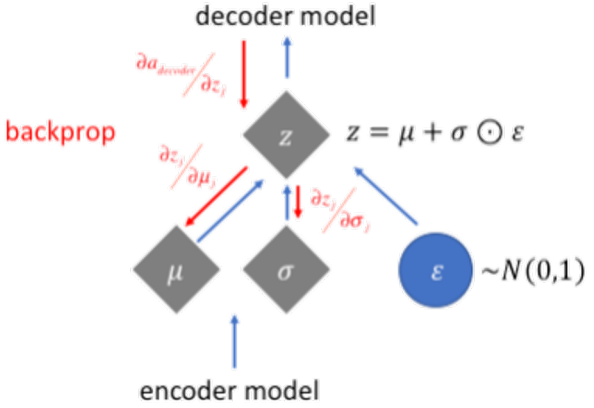


Reparameterization



Backpropagation not possible for a *random sampling* process!

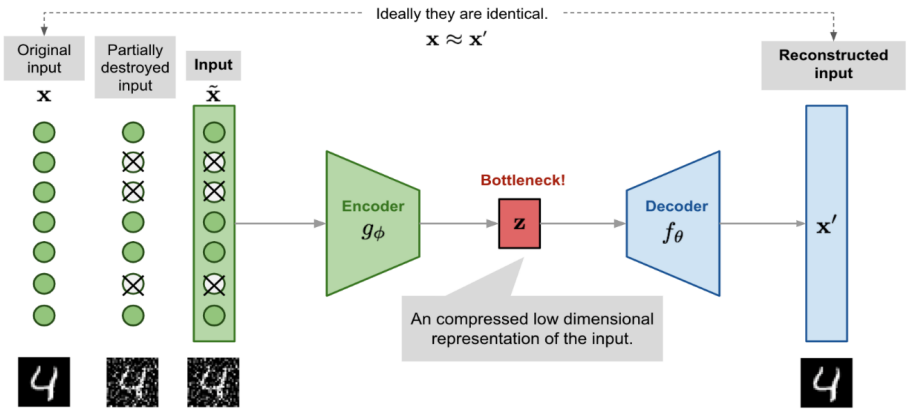
Backpropagation



Source: Jeremy Jordan | Link

Variational Autoencoders: Variants

- **Excellent Blog by Lilian Weng (Nov 2018): "From Autoencoder to Beta-VAE" | Link**



Denoising Autoencoder

To avoid overfitting and improve robustness, in **Denoising Autoencoder** (Vincent et al. 2008) input is partially corrupted by adding noises to or masking some values of the input vector in a stochastic manner.

k-Sparse Autoencoder

Sparsity is enforced by only keeping the top k highest activations in the latent layer with linear activation function (Makhzani and Frey, 2013).

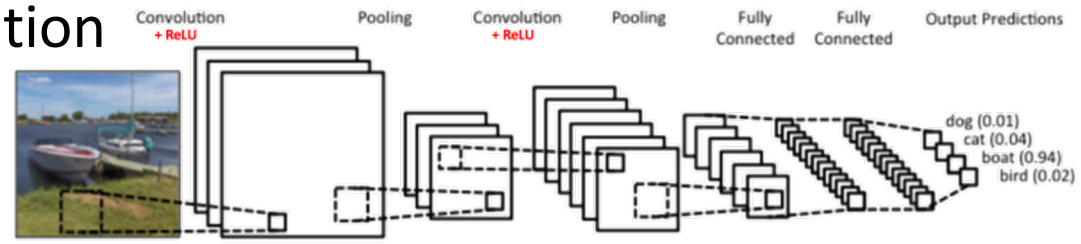
Beta-VAE

If each variable in the inferred latent representation z is only sensitive to one single generative factor and relatively invariant to others, we say representation is disentangled or factorized. Benefit is *good interpretability* and generalization to a variety of tasks.

β-VAE (Higgins et al., 2017) allows discovery of disentangled latent factors.

Convolutional Neural Networks (CNNs)

- **CNNs**: Apply a series of convolution "**filters**" to extract features, for pattern recognition
- Contains three components:



  - **Convolutional** layers: Apply convolution filters to input (or 1-D or *n*-D vectors)
    - For each subregion, layer performs a set of mathematical operations to produce a single value in the output feature map
    - Typically, also apply a ReLU activation function to output to introduce nonlinearities
  - **Pooling** layers: Down-sample data extracted by convolutional layers to reduce dimensionality of feature map
    - A commonly used pooling algorithm is "max" pooling, which extracts subregions of feature map (e.g., 3x3-pixel tiles) and keeps their maximum value
  - **Dense** (fully connected) layers: Perform pattern recognition on features extracted by the pooling layers
    - In a dense layer, every node in layer is connected to every node in the preceding layer

Source: Blog by Ujjwal Karn- https://ujjwalkarn.me/2016/08/11/intuitive-explanation-CNNs/

Convolutional Neural Networks (CNNs) …

- **Image: A Matrix of Pixel Values**
  - 3 Matrices for Color RGB Images
  - Channel is a component of an image
  - Grayscale image has just one channel

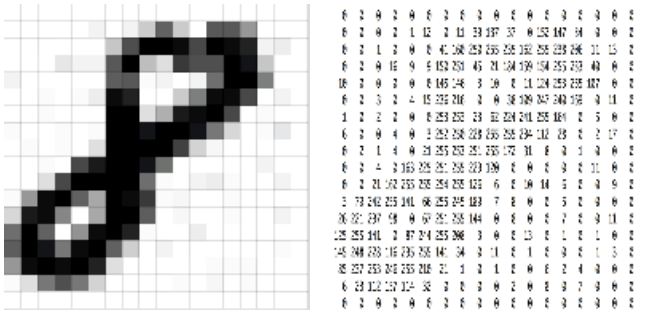
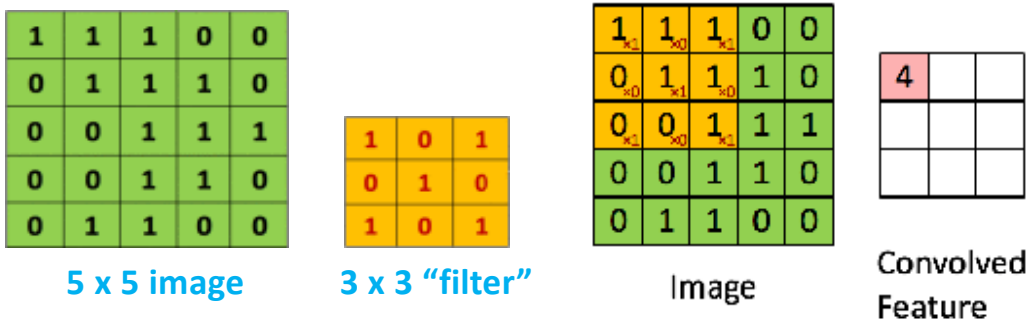


Image is a matrix of pixel values

- **Convolution Step**
  - Preserves spatial relationship between pixels by learning image features using small squares of input data
  - Consider a 5x5 image whose pixel values are only 0 and 1



5 x 5 image | 3 x 3 "filter" | Image | Convolved Feature

Convolution operation. Output is called "Feature Map"!

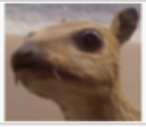
QUESTIONS:
What should be the size of the filter?
How to set the filter weights? How many filters?

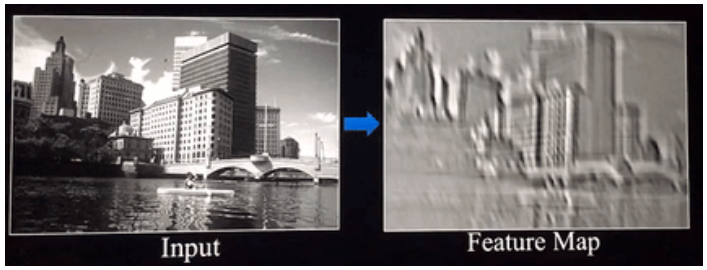
Convolutional Neural Networks (CNNs) …

Different Type of Filters

Learning Involves Optimizing Filter!

How Many Filters Do we Need?

Operation	Filter	Convolved Image
Identity	$\begin{bmatrix} 0 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 0 \end{bmatrix}$	
Edge detection	$\begin{bmatrix} 1 & 0 & -1 \\ 0 & 0 & 0 \\ -1 & 0 & 1 \end{bmatrix}$	
	$\begin{bmatrix} 0 & 1 & 0 \\ 1 & -4 & 1 \\ 0 & 1 & 0 \end{bmatrix}$	
	$\begin{bmatrix} -1 & -1 & -1 \\ -1 & 8 & -1 \\ -1 & -1 & -1 \end{bmatrix}$	
Sharpen	$\begin{bmatrix} 0 & -1 & 0 \\ -1 & 5 & -1 \\ 0 & -1 & 0 \end{bmatrix}$	
Box blur (normalized)	$\frac{1}{9}\begin{bmatrix} 1 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & 1 \end{bmatrix}$	
Gaussian blur (approximation)	$\frac{1}{16}\begin{bmatrix} 1 & 2 & 1 \\ 2 & 4 & 2 \\ 1 & 2 & 1 \end{bmatrix}$	



Example Convolution Operation

Feature Map having depth of 3 (since 3 filters have been used)

Feature Map with "Depth=3"

Convolution Operation

Each filter "learns" on its own to extract some special unique feature!

Filters are generally small; dictate number of learning parameters!

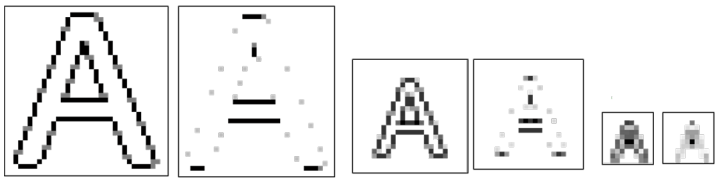
Convolutional Neural Networks (CNNs) …

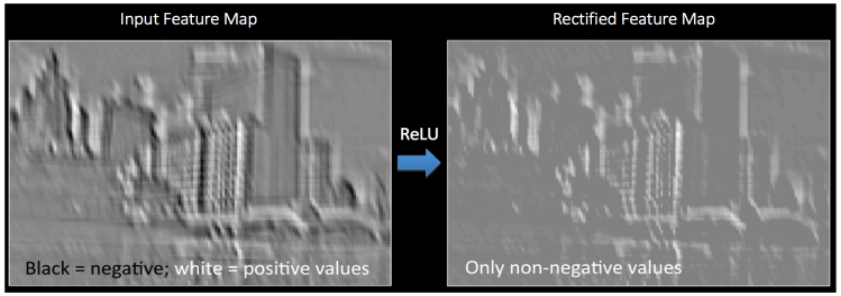
WAYNE STATE UNIVERSITY

- **Content of "feature map" is controlled by three parameters**
- **Depth**: Corresponds to # of filters used for convolution
  - For boat image, we used 3 distinct filters, thus producing 3 different feature maps; 'depth' of feature map would be 3
- **Stride**: Number of pixels by which we slide our filter matrix
  - When stride is 1, we move filters one pixel at a time
  - Having a larger stride will produce smaller feature maps
- **Zero-padding**:
  - Sometimes, it is convenient to pad input matrix with zeros around border, so that we can apply the filter to bordering elements of input image matrix
  - Allows us to control the size of feature maps

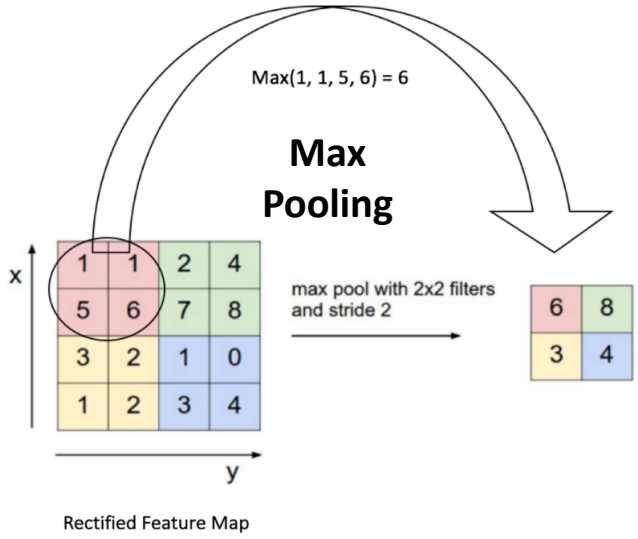
Convolutional Neural Networks (CNNs) …

- **Introducing Non-Linearity**
  - Typically used after every Convolution operation
  - Example: ReLU
- **Pooling Step**
  - Reduces dimensionality of feature map but retains important information
  - Different types: **Max**, **Average**, **Sum** …





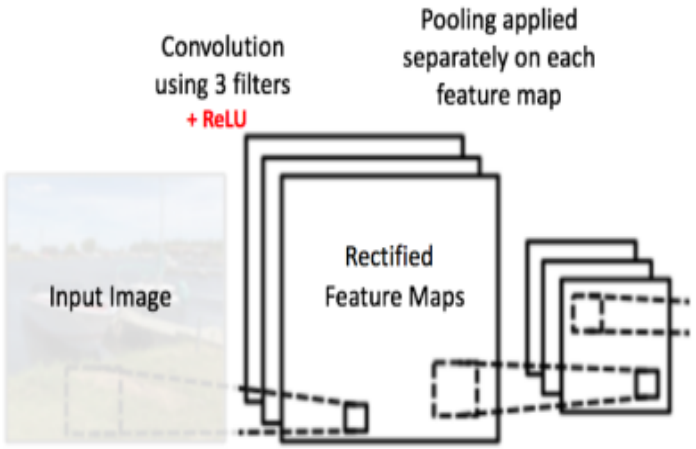
ReLU Operation



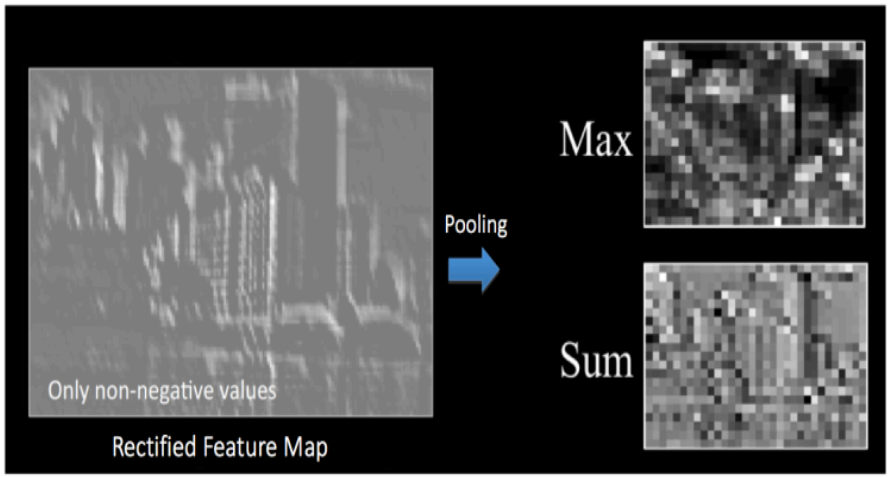
Rectified Feature Map

Convolutional Neural Networks (CNNs) …

- Pooling progressively reduces size of input representation
  - Makes input representations smaller and more manageable
  - Reduces the number of parameters and computations in the network
  - Makes network invariant to small transformations, distortions and translations in input image



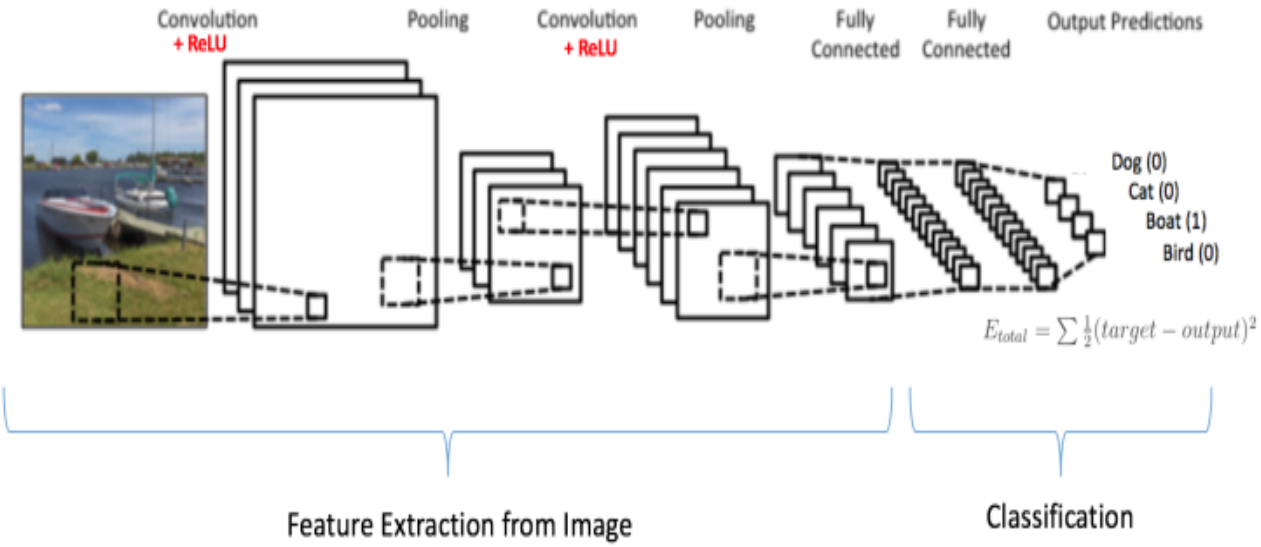
Pooling applied to Rectified Feature Maps



Pooling

CNNs: Putting It All Together & Training

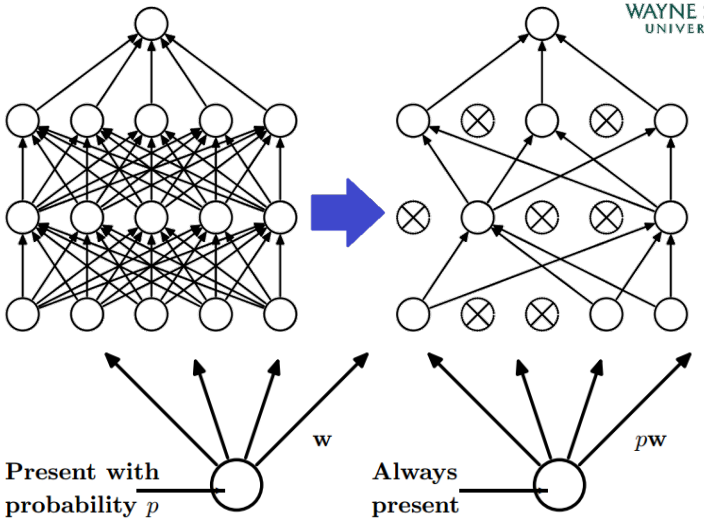
- Convolution + Pooling layers act as Feature Extractors from the input image while Fully Connected layer acts as a classifier



Implementing CNNs using Google's TensorFlow: MNIST Tutorial
Variational Autoencoders ("Generative" Models): Tutorial by Jaan Altosaar

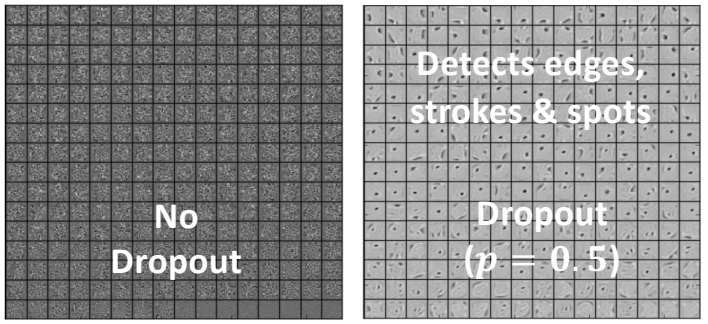
CNNs: "Dropout"

- **Neural networks are prone to overfitting**
- **Inspiration from biology**: Each neuron should work with a random sample of other units
  - At training (each iteration): Each unit is retained with a probability p
  - At test: Network is used as a whole; Weights are scaled-down by a factor of p
  - In practice, dropout trains 2^n networks (n units).
- **Why apply dropout on units and not arcs?**
  - Improves generalization
- **Slows training**: By 2~3 times
- **Excels when amount of data is average-large**
  - When data is big enough, does not help much
- **Achieves better results than older regularization methods** (e.g., weight decay)



$p = 0.5$ Sees to make the charm!

MNIST, one hidden layer, 256 ReLUs



Evolution of CNN Architectures for Images

WAYNE STATE UNIVERSITY

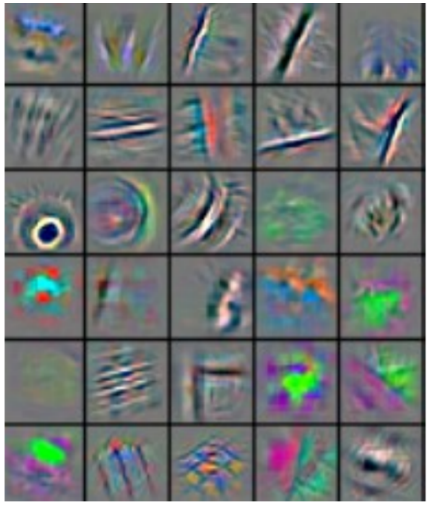
- **LeNet (1990s):** First CNN
- **AlexNet (2012):** Alex Krizhevsky (and others) released AlexNet, **deeper and much wider** version of LeNet, and won by a large margin the difficult ImageNet Large Scale Visual Recognition Challenge (ILSVRC) in 2012
- **GoogLeNet (2014):** ILSVRC 2014 winner from Google. Main contribution was development of an ***Inception Module*** that dramatically reduced number of parameters in network (4M, compared to AlexNet with 60M)
- **ResNets (2015):** ***Residual Network*** developed by Kaiming He (and others) was winner of ILSVRC 2015
- **DenseNet (August 2016):** Gao Huang (and others) proposed ***Densely Connected*** network that has each layer directly connected to every other layer in a feed-forward fashion. Obtained significant improvements over previous architectures on five highly competitive benchmark tasks.

CNNs for Images: Learn Low, Medium & High Level Features

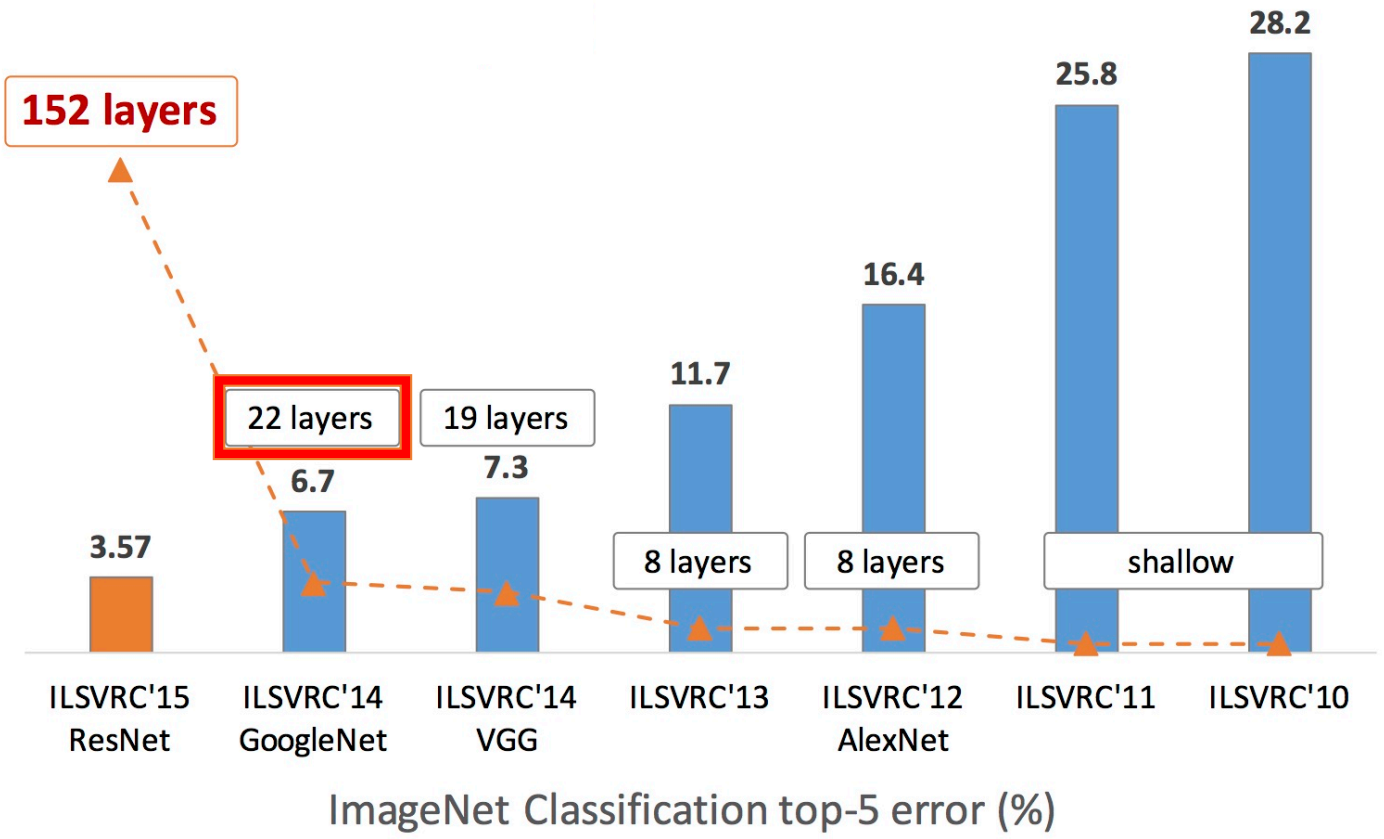
WAYNE STATE UNIVERSITY

IMAGENET: Link

1.2 Million Training Images
1,000 Classes



CNNs for Images: Increasing Depth



Kaiming He, Xiangyu Zhang, Shaoqing Ren, & Jian Sun. "Deep Residual Learning for Image Recognition". CVPR

CNNs for Images: GoogleNet

Layers: 144
Cross-layer Connections: 170

1x1 DAGNetwork

Property	Value
Layers	*144x1 Layer*
Connections	*170x2 table*

WAYNE STATE UNIVERSITY

Layers: 144 Input: 224x224 Color Image (3 Channels)

1	*1x1 ImageInputLayer*	41	*1x1 Convolution2DLayer*	81	*1x1 ReLULayer*	121	*1x1 ReLULayer*
2	*1x1 Convolution2DLayer*	42	*1x1 ReLULayer*	82	*1x1 DepthConcatenationLayer*	122	*1x1 MaxPooling2DLayer*
3	*1x1 ReLULayer*	43	*1x1 Convolution2DLayer*	83	*1x1 Convolution2DLayer*	123	*1x1 Convolution2DLayer*
4	*1x1 MaxPooling2DLayer*	44	*1x1 ReLULayer*	84	*1x1 ReLULayer*	124	*1x1 ReLULayer*
5	*1x1 CrossChannelNormalizationLayer*	45	*1x1 Convolution2DLayer*	85	*1x1 Convolution2DLayer*	125	*1x1 DepthConcatenationLayer*
6	*1x1 Convolution2DLayer*	46	*1x1 ReLULayer*	86	*1x1 ReLULayer*	126	*1x1 Convolution2DLayer*
7	*1x1 ReLULayer*	47	*1x1 Convolution2DLayer*	87	*1x1 Convolution2DLayer*	127	*1x1 ReLULayer*
8	*1x1 Convolution2DLayer*	48	*1x1 ReLULayer*	88	*1x1 ReLULayer*	128	*1x1 Convolution2DLayer*
9	*1x1 ReLULayer*	49	*1x1 Convolution2DLayer*	89	*1x1 Convolution2DLayer*	129	*1x1 ReLULayer*
10	*1x1 CrossChannelNormalizationLayer*	50	*1x1 ReLULayer*	90	*1x1 ReLULayer*	130	*1x1 Convolution2DLayer*
11	*1x1 MaxPooling2DLayer*	51	*1x1 MaxPooling2DLayer*	91	*1x1 Convolution2DLayer*	131	*1x1 ReLULayer*
12	*1x1 Convolution2DLayer*	52	*1x1 Convolution2DLayer*	92	*1x1 ReLULayer*	132	*1x1 Convolution2DLayer*
13	*1x1 ReLULayer*	53	*1x1 ReLULayer*	93	*1x1 MaxPooling2DLayer*	133	*1x1 ReLULayer*
14	*1x1 Convolution2DLayer*	54	*1x1 DepthConcatenationLayer*	94	*1x1 Convolution2DLayer*	134	*1x1 Convolution2DLayer*
15	*1x1 ReLULayer*	55	*1x1 Convolution2DLayer*	95	*1x1 ReLULayer*	135	*1x1 ReLULayer*
16	*1x1 Convolution2DLayer*	56	*1x1 ReLULayer*	96	*1x1 DepthConcatenationLayer*	136	*1x1 MaxPooling2DLayer*
17	*1x1 ReLULayer*	57	*1x1 Convolution2DLayer*	97	*1x1 Convolution2DLayer*	137	*1x1 Convolution2DLayer*
18	*1x1 Convolution2DLayer*	58	*1x1 ReLULayer*	98	*1x1 ReLULayer*	138	*1x1 ReLULayer*
19	*1x1 ReLULayer*	59	*1x1 Convolution2DLayer*	99	*1x1 Convolution2DLayer*	139	*1x1 DepthConcatenationLayer*
20	*1x1 Convolution2DLayer*	60	*1x1 ReLULayer*	100	*1x1 ReLULayer*	140	*1x1 AveragePooling2DLayer*
21	*1x1 ReLULayer*	61	*1x1 Convolution2DLayer*	101	*1x1 Convolution2DLayer*	141	*1x1 DropoutLayer*
22	*1x1 MaxPooling2DLayer*	62	*1x1 ReLULayer*	102	*1x1 ReLULayer*	142	*1x1 FullyConnectedLayer*
23	*1x1 Convolution2DLayer*	63	*1x1 Convolution2DLayer*	103	*1x1 Convolution2DLayer*	143	*1x1 SoftmaxLayer*
24	*1x1 ReLULayer*	64	*1x1 ReLULayer*	104	*1x1 ReLULayer*	144	*1x1 ClassificationOutputLayer*
25	*1x1 DepthConcatenationLayer*	65	*1x1 MaxPooling2DLayer*	105	*1x1 Convolution2DLayer*		
26	*1x1 Convolution2DLayer*	66	*1x1 Convolution2DLayer*	106	*1x1 ReLULayer*		
27	*1x1 ReLULayer*	67	*1x1 ReLULayer*	107	*1x1 MaxPooling2DLayer*		
28	*1x1 Convolution2DLayer*	68	*1x1 DepthConcatenationLayer*	108	*1x1 Convolution2DLayer*		
29	*1x1 ReLULayer*	69	*1x1 Convolution2DLayer*	109	*1x1 ReLULayer*		
30	*1x1 Convolution2DLayer*	70	*1x1 ReLULayer*	110	*1x1 DepthConcatenationLayer*		
31	*1x1 ReLULayer*	71	*1x1 Convolution2DLayer*	111	*1x1 MaxPooling2DLayer*		
32	*1x1 Convolution2DLayer*	72	*1x1 ReLULayer*	112	*1x1 Convolution2DLayer*		
33	*1x1 ReLULayer*	73	*1x1 Convolution2DLayer*	113	*1x1 ReLULayer*		
34	*1x1 Convolution2DLayer*	74	*1x1 ReLULayer*	114	*1x1 Convolution2DLayer*		
35	*1x1 ReLULayer*	75	*1x1 Convolution2DLayer*	115	*1x1 ReLULayer*		
36	*1x1 MaxPooling2DLayer*	76	*1x1 ReLULayer*	116	*1x1 Convolution2DLayer*		
37	*1x1 Convolution2DLayer*	77	*1x1 Convolution2DLayer*	117	*1x1 ReLULayer*		
38	*1x1 ReLULayer*	78	*1x1 ReLULayer*	118	*1x1 Convolution2DLayer*		
39	*1x1 DepthConcatenationLayer*	79	*1x1 MaxPooling2DLayer*	119	*1x1 ReLULayer*		
40	*1x1 MaxPooling2DLayer*	80	*1x1 Convolution2DLayer*	120	*1x1 Convolution2DLayer*		

Details for Select Layers:

netCNN.Layers(1, 1)

Property	Value
Name	'data'
InputSize	[224,224,3]
DataAugmentation	'none'
Normalization	'zerocenter'
AverageImage	*224x224x3 single*
NumInputs	0
InputNames	*0x0 cell*
NumOutputs	1
OutputNames	*1x1 cell*

netCNN.Layers(2, 1)

Property	Value
Name	'conv1-7x7_s2'
FilterSize	[7,7]
NumChannels	3
NumFilters	64
Stride	[2,2]
DilationFactor	[1,1]
PaddingMode	'manual'
PaddingSize	[3,3,3,3]
Weights	*4-D single*
Bias	*1x1x64 single*
WeightsInitializer	'glorot'
WeightLearnRateFactor	1
WeightL2Factor	1
BiasInitializer	'zeros'
BiasLearnRateFactor	2
BiasL2Factor	0
NumInputs	1
InputNames	*1x1 cell*
NumOutputs	1
OutputNames	*1x1 cell*

netCNN.Layers(137, 1)

Property	Value
Name	'inception_5b-pool_proj'
FilterSize	[1,1]
NumChannels	832
NumFilters	128
Stride	[1,1]
DilationFactor	[1,1]
PaddingMode	'manual'
PaddingSize	[0,0,0,0]
Weights	*4-D single*
Bias	*1x1x128 single*
WeightsInitializer	'glorot'
WeightLearnRateFactor	1
WeightL2Factor	1
BiasInitializer	'zeros'
BiasLearnRateFactor	2
BiasL2Factor	0
NumInputs	1
InputNames	*1x1 cell*
NumOutputs	1
OutputNames	*1x1 cell*

netCNN.Layers(142, 1)

Property	Value
Name	'loss3-classifier'
InputSize	1024
OutputSize	1000
Weights	*1000x1024 single*
Bias	*1000x1 single*
WeightsInitializer	'glorot'
WeightLearnRateFactor	1
WeightL2Factor	1
BiasInitializer	'zeros'
BiasLearnRateFactor	2
BiasL2Factor	0
NumInputs	1
InputNames	*1x1 cell*
NumOutputs	1
OutputNames	*1x1 cell*

netCNN.Layers(144, 1)

Property	Value
Name	'output'
Classes	*1000x1 categorical*
OutputSize	1000
LossFunction	'crossentropyex'
NumInputs	1
InputNames	*1x1 cell*
NumOutputs	0
OutputNames	*0x0 cell*

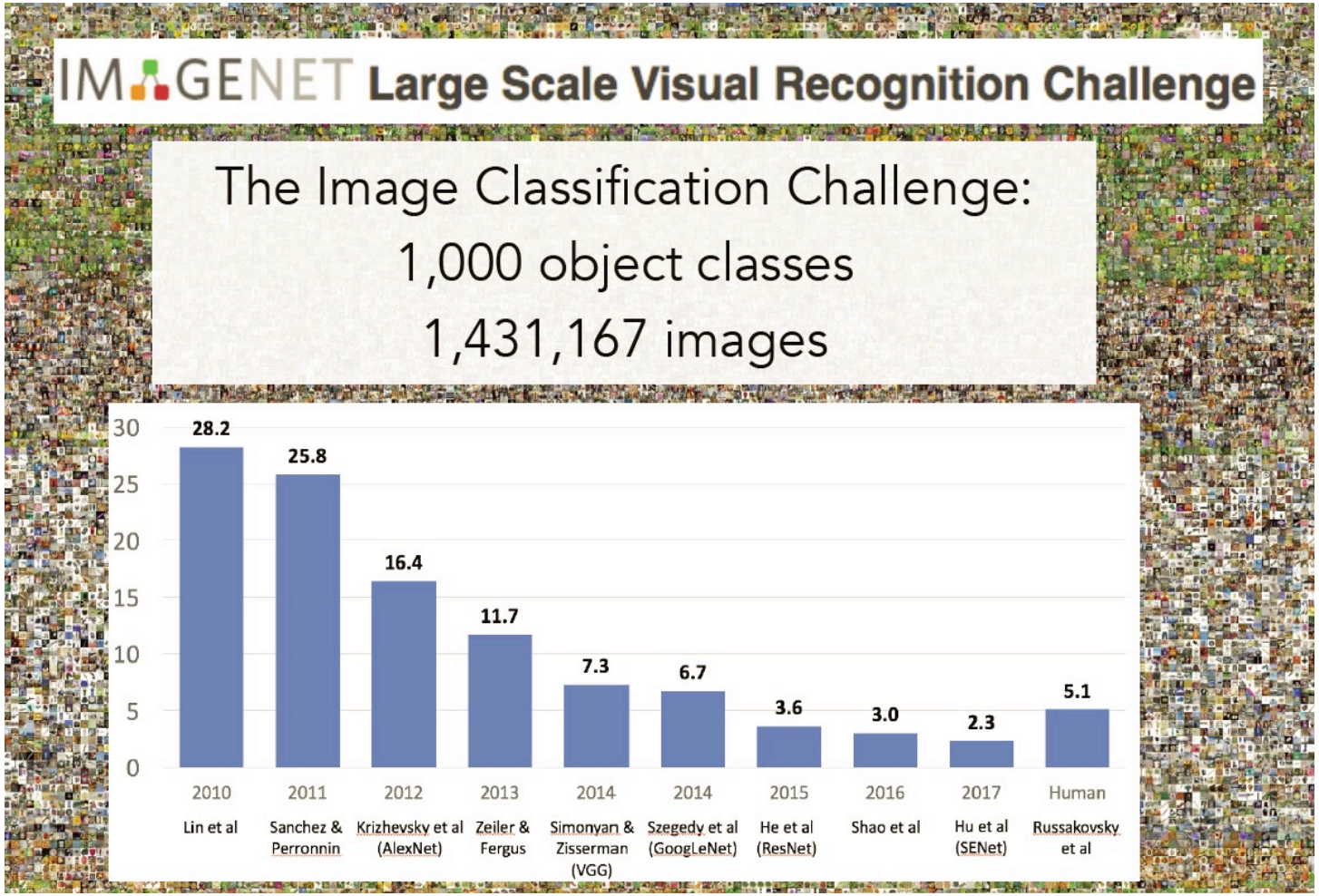
Connections:

netCNN.Connections

	1 Source	2 Destination
1	'data'	'conv1-7x7_s2'
2	'conv1-7x7_s2'	'conv1-relu_7x7'
3	'conv1-relu_7x7'	'pool1-3x3_s2'
4	'pool1-3x3_s2'	'pool1-norm1'
5	'pool1-norm1'	'conv2-3x3_reduce'
6	'conv2-3x3_reduce'	'conv2-relu_3x3_reduce'
7	'conv2-relu_3x3_reduce'	'conv2-3x3'
8	'conv2-3x3'	'conv2-relu_3x3'
9	'conv2-relu_3x3'	'conv2-norm2'
10	'conv2-norm2'	'pool2-3x3_s2'
11	'pool2-3x3_s2'	'inception_3a-1x1'
12	'pool2-3x3_s2'	'inception_3a-3x3_reduce'
13	'pool2-3x3_s2'	'inception_3a-5x5_reduce'
14	'pool2-3x3_s2'	'inception_3a-pool'
15	'inception_3a-1x1'	'inception_3a-relu_1x1'
16	'inception_3a-relu_1x1'	'inception_3a-output/in1'
17	'inception_3a-3x3_reduce'	'inception_3a-relu_3x3_reduce'
18	'inception_3a-relu_3x3_reduce'	'inception_3a-3x3'
19	'inception_3a-3x3'	'inception_3a-relu_3x3'
20	'inception_3a-relu_3x3'	'inception_3a-output/in2'
21	'inception_3a-5x5_reduce'	'inception_3a-relu_5x5_reduce'
22	'inception_3a-relu_5x5_reduce'	'inception_3a-5x5'
23	'inception_3a-5x5'	'inception_3a-relu_5x5'
24	'inception_3a-relu_5x5'	'inception_3a-output/in3'
25	'inception_3a-pool'	'inception_3a-pool_proj'
26	'inception_3a-pool_proj'	'inception_3a-relu_pool_proj'
27	'inception_3a-relu_pool_proj'	'inception_3a-output/in4'
28	'inception_3a-output'	'inception_3b-1x1'
29	'inception_3a-output'	'inception_3b-3x3_reduce'
30	'inception_3a-output'	'inception_3b-5x5_reduce'
31	'inception_3a-output'	'inception_3b-pool'

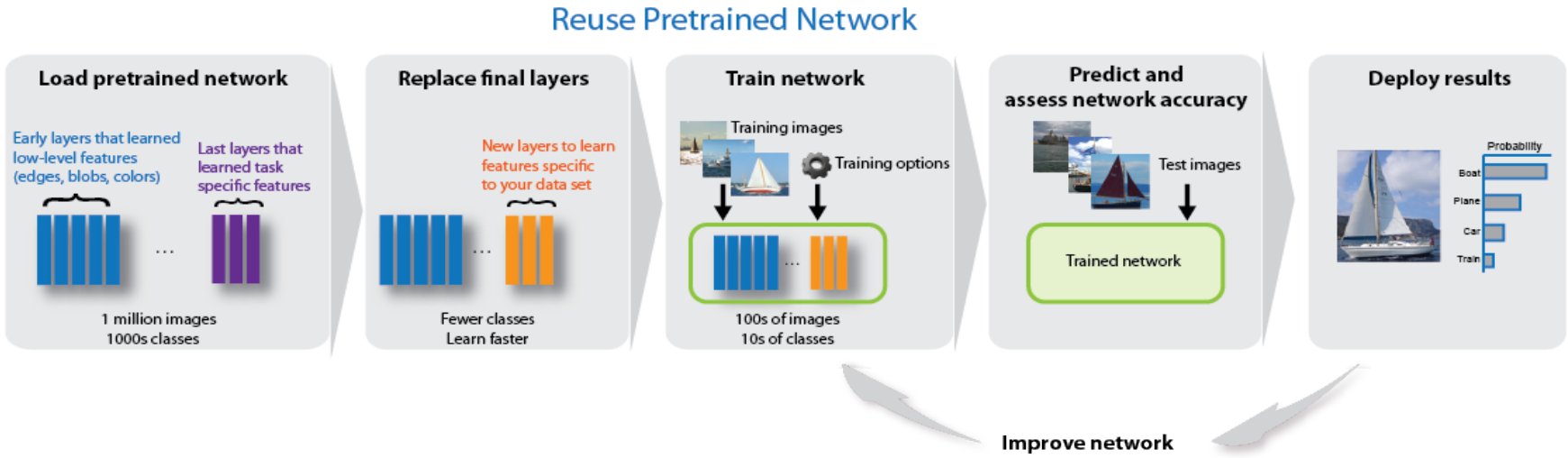
Source: Matlab 2019a (GoogleNet)

Machine Surpasses Humans (Image Classification)



Transfer Learning

- **Transfer Learning**: Take a pretrained network and use it as a starting point to learn a new task
  - Fine-tuning a network with transfer learning is usually much faster and easier than training a network from scratch
  - Can quickly transfer learned features to a new task using a smaller number of training images

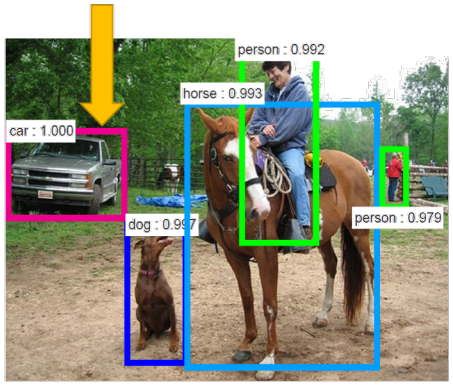


Need Hierarchical Strategies for the Real-World

WAYNE STATE UNIVERSITY

Localization
Where?

Recognition
What?



Example: Convolutional Feature Maps for Multi-Object Recognition/Location

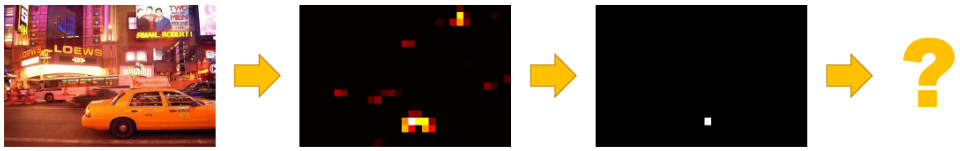


image — a feature map — keep one response (e.g., the strongest)

Feature Map: Visualizing "One Response"

system	time
R-CNN	~50s
Fast R-CNN	~2s
Faster R-CNN	198ms

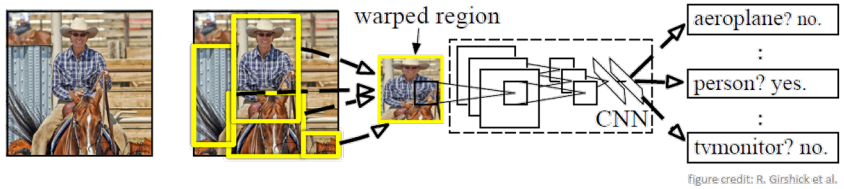
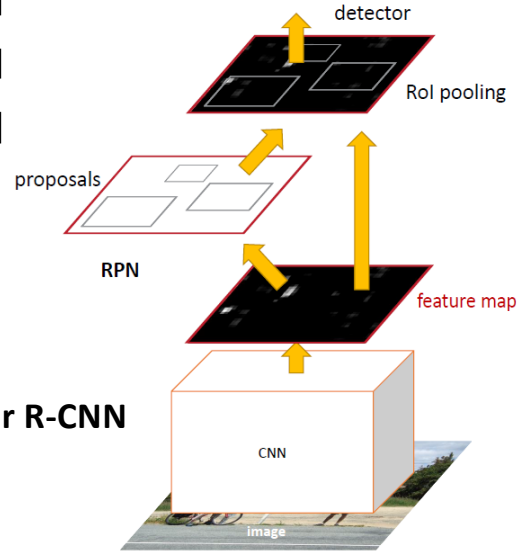


figure credit: R. Girshick et al.

input image — region proposals ~2,000 — 1 CNN for each region — classify regions

"Region-Based" CNN (R-CNN) Pipeline



Faster R-CNN

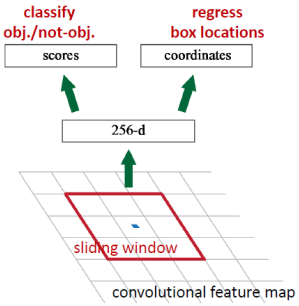
Strategy

Convolutional: *sliding-window operations*

Feature: *encoding "what" (and implicitly encoding "where")*

Map: *explicitly encoding "where"*

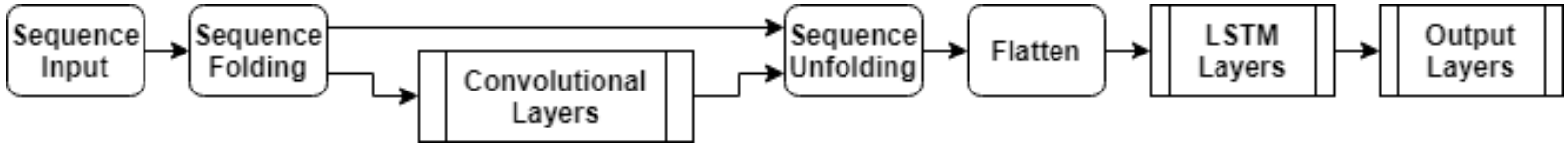
Region "Proposal Network" (RPN)



Source: He, K. (2015) Convolutional Feature Maps – ICCV 2015 | Link

Matlab: Classify Videos Using Deep Learning | Link

- **Create a network for video classification by combining a pretrained image classification model and an LSTM network**
  - Convert video frames to sequences of feature vectors using a pretrained convolutional neural network to extract features from each frame
  - Train an LSTM network on sequences to predict video labels
  - Assemble network that classifies videos directly by combining layers from both



- To input image sequences to network, use a sequence input layer
- To apply convolutional operations to each frame of the videos independently, use a sequence folding layer followed by convolutional layers
- To restore sequence structure and reshape output to vector sequences, use a sequence unfolding layer and a flatten layer
- To classify resulting vector sequences, include LSTM layers followed by output layers

Matlab: Classify Videos Using Deep Learning | Link

WAYNE STATE UNIVERSITY

Load Pretrained GoogLeNet CNN and Data

```
netCNN = googlenet;
```

- Download HBMD51 dataset from HMDB: Large human motion database and extract RAR file into a folder "hmdb51_org"
  - Dataset contains about 2 GB of video data for 7000 clips over 51 classes, such as "drink", "run", and "shake_hands"
- After extracting RAR files, use supporting function hmdb51Files to get file names and labels of videos

```
dataFolder = "hmdb51_org";
[files,labels] = hmdb51Files(dataFolder);
```

- Read first video using readVideo helper function and view size
  - Video is a H-by-W-by-C-by-S array, where H, W, C, and S are the height, width, number of channels, and number of frames

```
idx = 1;
filename = files(idx);
video = readVideo(filename);
size(video)
ans = 1×4: 240   320     3   409
```

- View corresponding label.

```
labels(idx)
ans = categorical: brush_hair
```

Convert Frames to Feature Vectors

- Use CNN as feature extractor
  - Feature vectors are output of activations function on last pooling layer of GoogLeNet network ("pool5-7x7_s1")
- To read video data and resize it to match input size of GoogLeNet network, use readVideo and centerCrop helper functions (takes time)
  - After converting to sequences, save sequences in a MAT-file in tempdir folder

```
inputSize = netCNN.Layers(1).InputSize(1:2);
layerName = "pool5-7x7_s1";
tempFile = fullfile(tempdir,"hmdb51_org.mat")
if exist(tempFile,'file')
    load(tempFile,"sequences")
else
    numFiles = numel(files);
    sequences = cell(numFiles,1);
    for i = 1:numFiles
        fprintf("Reading file %d of %d...\n", i, numFiles)
        video = readVideo(files(i));
        video = centerCrop(video,inputSize);
        sequences{i,1} =
activations(netCNN,video,layerName,'OutputAs','columns');
    end
    save(tempFile,"sequences","-v7.3");
end
```

- View sizes of first few sequences

```
sequences(1:2)
ans = 10×1 cell array: {1024×409 single} {1024×395 single}
```

Matlab: Classify Videos Using Deep Learning | Link

WAYNE STATE UNIVERSITY

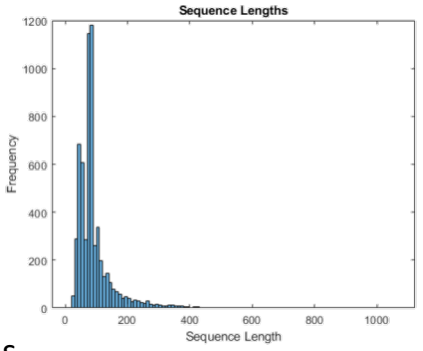
Prepare Training Data

- Create Training and Validation Partitions: 90% to training and 10% to validation

```
numObservations = numel(sequences);
idx = randperm(numObservations);
N = floor(0.9 * numObservations);
idxTrain = idx(1:N);
sequencesTrain = sequences(idxTrain);
labelsTrain = labels(idxTrain);
idxValidation = idx(N+1:end);
sequencesValidation = sequences(idxValidation);
labelsValidation = labels(idxValidation);
Remove Long Sequences to Avoid Too Much Padding
numObservationsTrain = numel(sequencesTrain);
sequenceLengths =
zeros(1,numObservationsTrain);
for i = 1:numObservationsTrain
    sequence = sequencesTrain{i};
    sequenceLengths(i) = size(sequence,2);
end
```

- Plot length of sequences

```
figure
histogram(sequenceLengths)
title("Sequence Lengths")
xlabel("Sequence Length")
ylabel("Frequency")
```



- Few sequences have > 400 frames
- To improve classification accuracy, remove long sequences.

```
maxLength = 400;
idx = sequenceLengths > maxLength;
sequencesTrain(idx) = [];
labelsTrain(idx) = [];
```

Matlab: Classify Videos Using Deep Learning | Link

WAYNE STATE UNIVERSITY

Create LSTM Network

- A sequence input layer with an input size corresponding to feature dimension of feature vectors
- A BiLSTM layer with 2000 hidden units with a dropout layer afterwards.
- To output only one label for each sequence by setting the 'OutputMode' option of the BiLSTM layer to 'last'
- A fully connected layer with an output size corresponding to number of classes, a softmax layer, and a classification layer

```
numFeatures = size(sequencesTrain{1},1);
numClasses = numel(categories(labelsTrain));
layers = [
    sequenceInputLayer(numFeatures,'Name','sequence')
    bilstmLayer(2000,'OutputMode','last','Name','bilstm')
    dropoutLayer(0.5,'Name','drop')
    fullyConnectedLayer(numClasses,'Name','fc')
    softmaxLayer('Name','softmax')
    classificationLayer('Name','classification')];
```

Specify Training Options

- Set a mini-batch size 16, initial learning rate of 0.0001, and a gradient threshold of 2 (to prevent gradients from exploding)
- Truncate sequences in each mini-batch to have same length as shortest sequence
- Shuffle data every epoch; Validate network once per epoch
- Display training progress in a plot and suppress verbose output

```
miniBatchSize = 16;
numObservations = numel(sequencesTrain);
numIterationsPerEpoch = floor(numObservations / miniBatchSize);
options = trainingOptions('adam', ...
    'MiniBatchSize',miniBatchSize, ...
    'InitialLearnRate',1e-4, ...
    'GradientThreshold',2, ...
    'Shuffle','every-epoch', ...

'ValidationData',{sequencesValidation,labelsValidation},
...
    'ValidationFrequency',numIterationsPerEpoch, ...
    'Plots','training-progress', ...
    'Verbose',false);
```

Matlab: Classify Videos Using Deep Learning | Link

WAYNE STATE UNIVERSITY

Train LSTM Network

```
[netLSTM,info] =
trainNetwork(sequencesTrain,
labelsTrain,layers,options);
```

- Calculate classification accuracy of network on validation set using same mini-batch size

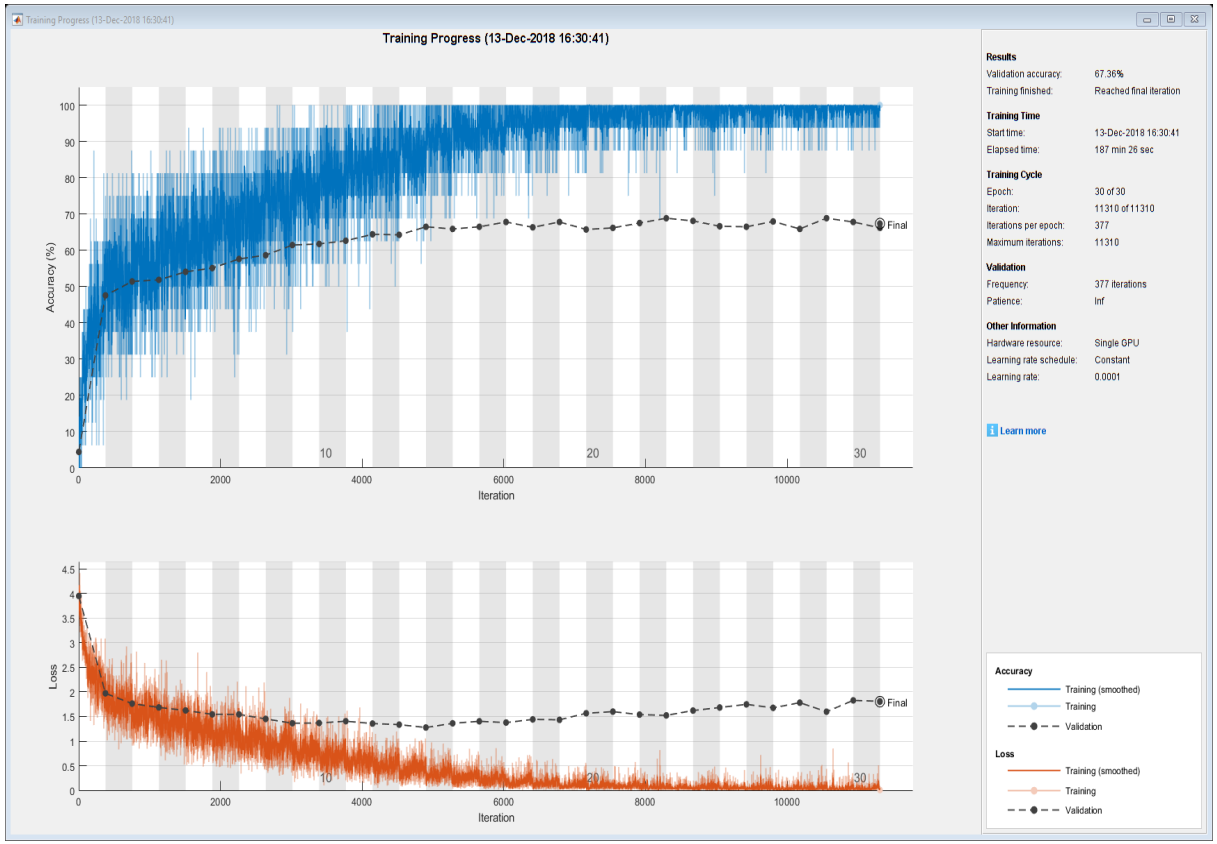
```
YPred = classify(netLSTM,
sequencesValidation,
'MiniBatchSize',miniBatchSize);

YValidation = labelsValidation;

accuracy = mean(YPred ==
YValidation)

accuracy = 0.6647
```

- Training accuracy close to 100% but validation accuracy of 66.47%



Matlab: Classify Videos Using Deep Learning | Link

WAYNE STATE UNIVERSITY

Assemble Video Classification Network

- **Add Convolutional Layers**
  - First, create a layer graph of GoogLeNet network

```
cnnLayers = layerGraph(netCNN);
```

  - Remove input layer ("data") and layers after pooling layer used for activations ("pool5-drop_7x7_s1", "loss3-classifier", "prob", and "output")

```
layerNames = ["data" "pool5-drop_7x7_s1" "loss3-classifier" "prob" "output"];
cnnLayers = removeLayers(cnnLayers,layerNames);
```

- **Add Sequence Input Layer**
  - Create a sequence input layer that accepts image sequences containing images of same input size as GoogLeNet network. To normalize images using same average image as GoogLeNet network, set 'Normalization' option of sequence input layer to 'zerocenter' and 'Mean' option to average image of input layer of GoogLeNet

```
inputSize = netCNN.Layers(1).InputSize(1:2);
averageImage = netCNN.Layers(1).AverageImage;
inputLayer = sequenceInputLayer([inputSize 3], ...
    'Normalization','zerocenter', ...
    'Mean',averageImage, ...
    'Name','input');
```

  - Add sequence input layer to layer graph. To apply convolutional layers to images of sequences independently, remove sequence structure of image sequences by including a sequence folding layer between sequence input layer and convolutional layers. Connect output of sequence folding layer to input of first convolutional layer ("conv1-7x7_s2").

```
layers = [
    inputLayer
    sequenceFoldingLayer('Name','fold')];
lgraph = addLayers(cnnLayers,layers);
lgraph = connectLayers(lgraph,"fold/out","conv1-7x7_s2");
```

- **Add LSTM Layers**
  - Add LSTM layers to layer graph by removing sequence input layer of LSTM network. To restore sequence structure removed by sequence folding layer, include a sequence unfolding layer after convolution layers. LSTM layers expect sequences of vectors. To reshape output of sequence unfolding layer to vector sequences, include a flatten layer after sequence unfolding layer.
  - Take layers from LSTM network and remove sequence input layer.

```
lstmLayers = netLSTM.Layers;
lstmLayers(1) = [];
```

  - Add sequence folding layer, flatten layer, and LSTM layers to layer graph. Connect last convolutional layer ("pool5-7x7_s1") to input of sequence unfolding layer ("unfold/in").

```
layers = [
    sequenceUnfoldingLayer('Name','unfold')
    flattenLayer('Name','flatten')
    lstmLayers];
lgraph = addLayers(lgraph,layers);
lgraph = connectLayers(lgraph,"pool5-7x7_s1","unfold/in");
```

  - To enable unfolding layer to restore sequence structure, connect "miniBatchSize" output of sequence folding layer to corresponding input of sequence unfolding layer.

```
lgraph = connectLayers(lgraph,"fold/miniBatchSize","unfold/miniBatchSize");
```

- **Assemble Network**
  - Check that network is valid using analyzeNetwork function.

```
analyzeNetwork(lgraph)
```

  - Assemble the network so that it is ready for prediction using the assembleNetwork function.

```
net = assembleNetwork(lgraph)
net =
  DAGNetwork with properties:
         Layers: [148×1 nnet.cnn.layer.Layer]
    Connections: [175×2 table]
```

Matlab: Classify Videos Using Deep Learning | Link

WAYNE STATE UNIVERSITY

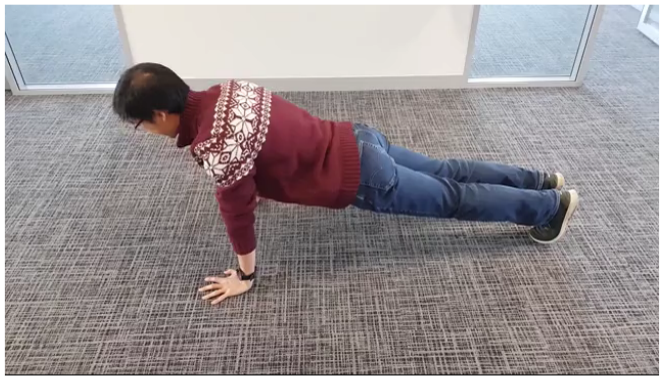
Classify Using New Data

- Read and center-crop video "pushup.mp4" using same steps

```
filename = "pushup.mp4";
video = readVideo(filename);
```

- To view video, use implay function (requires Image Processing Toolbox)
- This function expects data in range [0,1], so you must first divide data by 255. Alternatively, loop over individual frames and use imshow function.

```
numFrames = size(video,4);
figure
for i = 1:numFrames
    frame = video(:,:,:,i);
    imshow(frame/255);
    drawnow
end
```

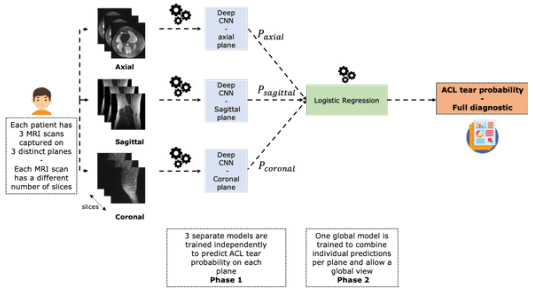


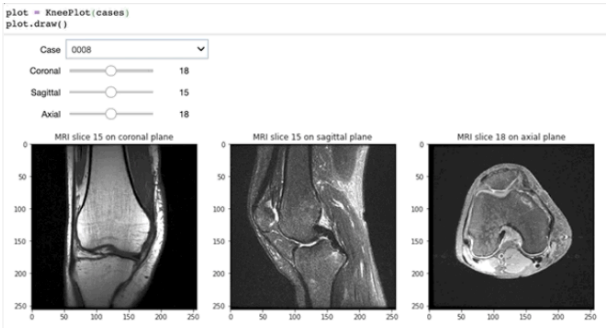
- Classify video using assembled network
- Classify function expects a cell array containing input videos, so you must input a 1-by-1 cell array containing video

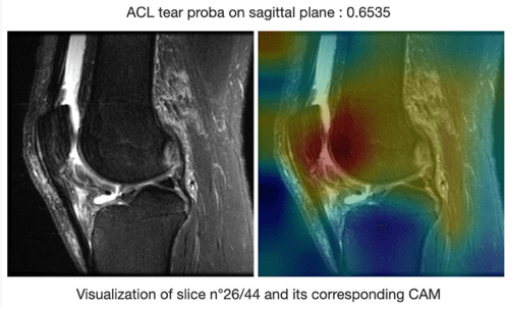
```
video = centerCrop(video,inputSize);
YPred = classify(net,{video})
YPred = categorical : pushup
```

Python & Tensor Flow/PyTorch Examples for DL

- Google's Tensor Flow Examples for Images:
  - Basic CNN - CIFARD10 Data Set: Link
  - CNN (Avoid Overfitting: Data Augmentation & Dropout) - Cats vs Dogs Dataset : Link
  - Transfer learning with CNN - Cats vs Dogs Dataset: Link
- Automate Diagnosis of Knee Injuries using MRI – PyTorch (Ahmed Besbes):
  - Overview of Problem & MRNet Dataset: Link
  - Building an ACL Tear Classifier: Link
  - Interpret Models' Predictions: Link
  - GitHub: Repo





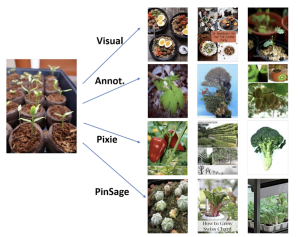


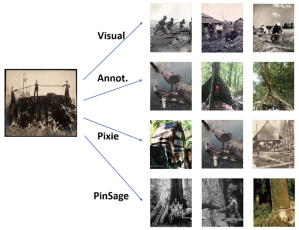
Graph Neural Networks (GNNs)

- Some datasets involve data inputs as graphs
  - Social network data, traffic networks, retail store networks, chemical molecule structures …
- **Graph Machine Learning** deals with graph data
  - Users: Uber Eats, Facebook, Pinterest, Amazon …
  - Applications: Recommendation systems, Node Classification, Edge Classification, …
- **Graphs**: Consist of ***nodes***, that may have associated feature vectors, and ***edges***, and may have attached feature vectors
  - Different kinds: Undirected and directed graphs; Field of mathematics aptly named "graph theory"
- **Challenge**: Need non-Euclidean preprocessing to present graph to NN and other ML models
  - Nodes have no particular "order"; Same graph can appear in different ways
  - Whereas most ML datasets have Euclidean structure (e.g., even a pixel in an image or a word in a sentence)

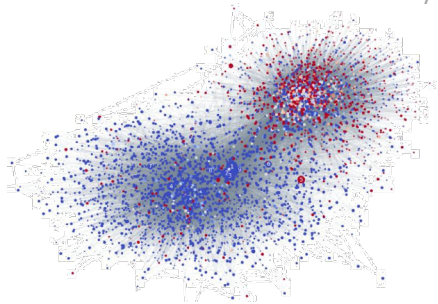
Primary Source: Aleksa Gordic | LINK

Food discovery with Uber Eats: Using Graph Learning to Power Recommendations





PinSage of Pinterest: A new graph convolutional neural network for web-scale recommender systems



Twitter: Reliable users colored in blue and unreliable ones (prone to spreading fake news) in red.

Graph Neural Networks (GNNs) …

- **Graph Embedding Methods**: Convert graphs into "embedding vectors"
  - DeepWalk — it's reduced to Word2Vec if you do the following: sample "sentences" from the graph by doing random walks.
  - Node2Vec — literally the same idea as DeepWalk with the additional control of how you'll sample from your graph (BFS/DFS).
  - Planetoid — a semi-supervised setting. Aside from random walks (unsupervised), you can also leverage the labels (supervised).
- **Graph Neural Networks**: Generalize "convolution" concept to graphs
  - "Spectral" Methods: Resort to frequency domain for convolution.
    - Computationally expensive; Not as popular
  - "Spatial" (message passing) Methods: Are not strict convolutions, but we still informally call them convolutions.
  - Deeper Overview: Watch this high-level overview of GNNs by Microsoft Research and by Xavier Bresson
  - Few Blogs: Tutorial on GNNs | A Tale of Two Convolutions; | Some Important GNNs
- **Python Packages**:
  - DGL: Easy deep learning on graphs using PyTorch, TensorFlow or MXNet
  - Spektral: Library for graph deep learning, based on the Keras API and TensorFlow 2
  - PyTorch Geometric: Geometric deep learning extension library for PyTorch

Primary Source: Aleksa Gordic | LINK