

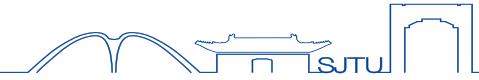


Talk 13:

Deep Neural Networks and Regularization



Outline

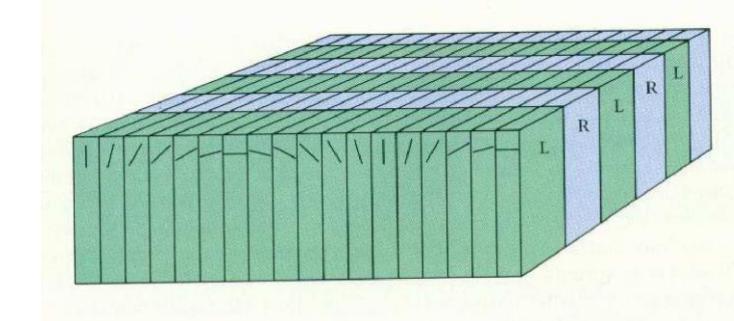
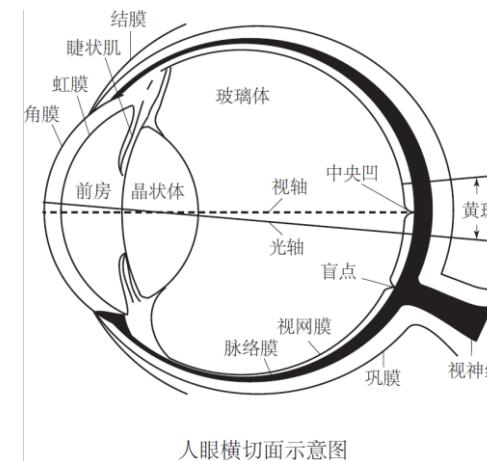
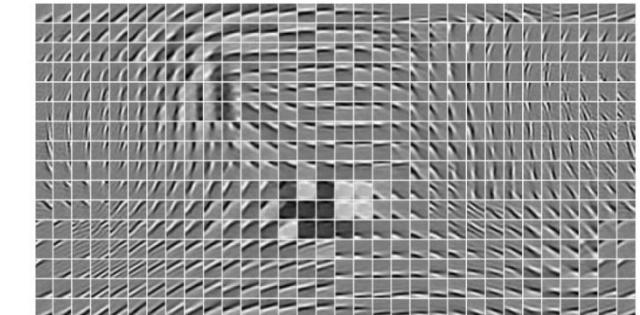
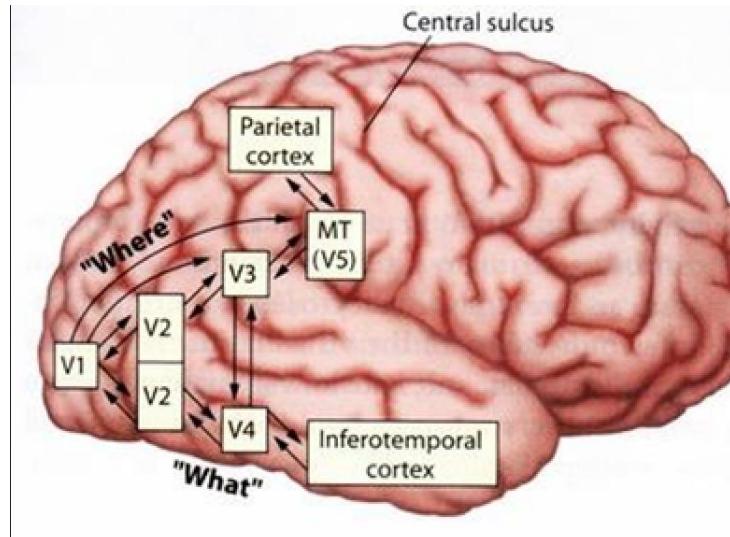


- Background
- Deep Convolutional Networks
 - Filter
 - Pooling
 - Dropout
- Disentangled Representation in DNN
- CNN for Computer Vision
- Perspectives and Future Directions

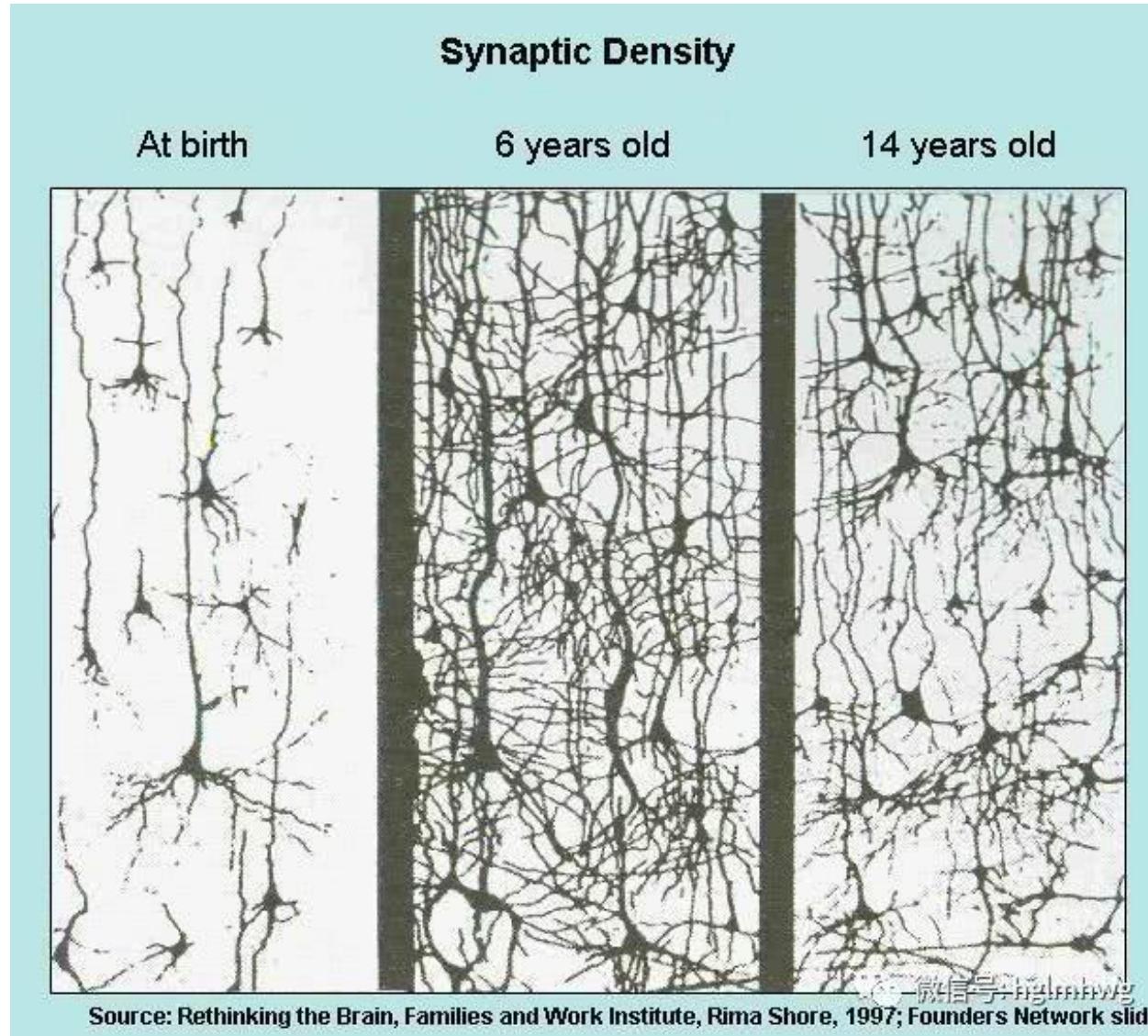
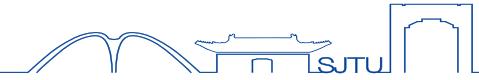
Cortical Structures



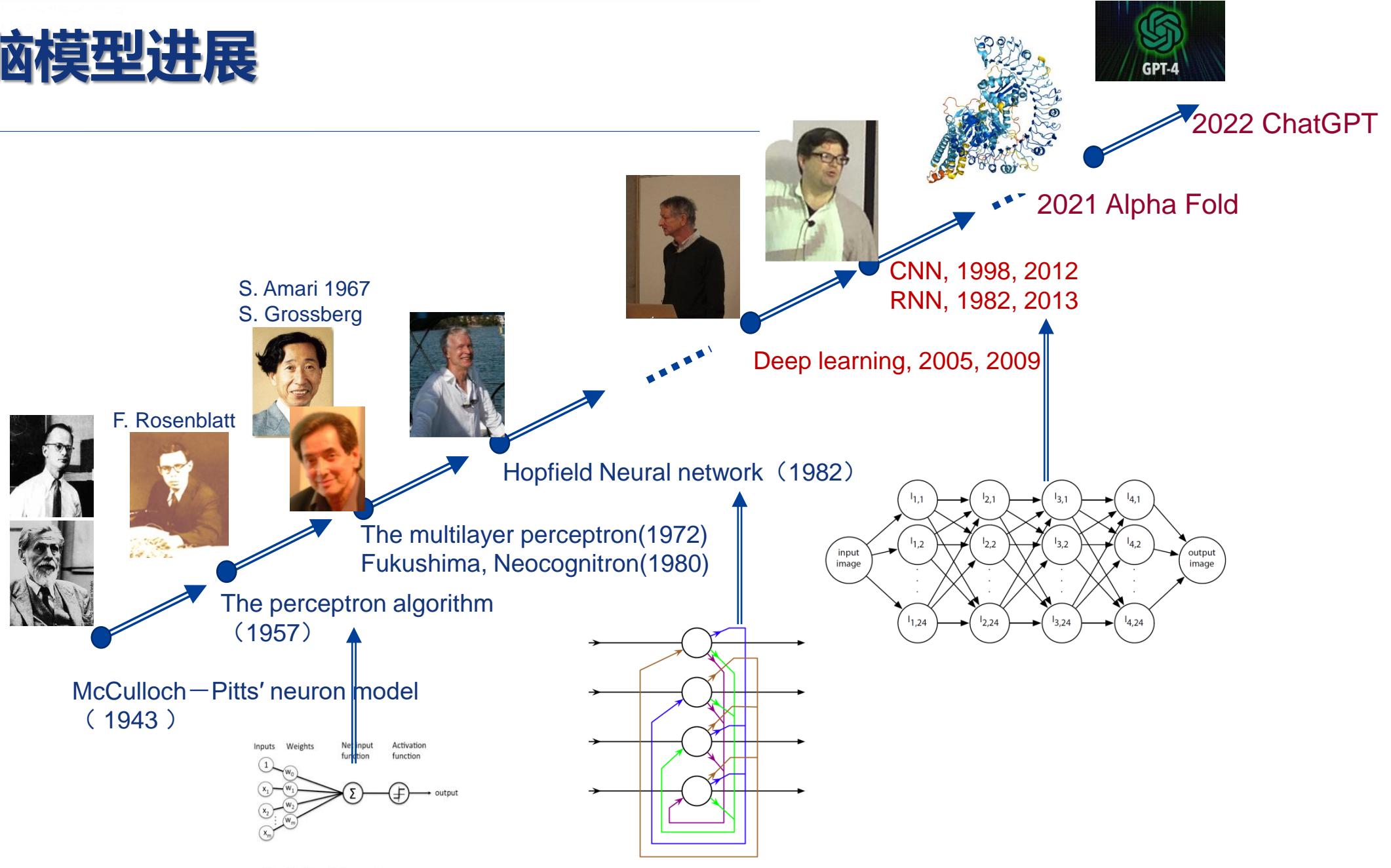
- Hierarchical Structures
- Topological Structures
 - Overcomplete
- Columnar Structures
- Connections
 - Feedforward, feedback, lateral
- Nonhomogeneity of Retina Cells
 - Computational implications



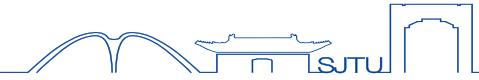
皮层突触修剪 (Pruning)



类脑模型进展



The New Challenge

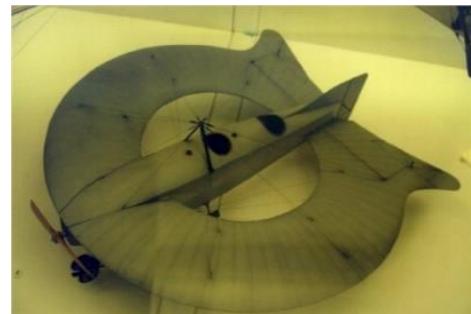


- How do we learn **invariant representations**?

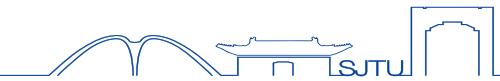
- ▶ From the image of an airplane, how do we extract a representation that is invariant to pose, illumination, background, clutter, object instance....
- ▶ How can a human (or a machine) learn those representations by just looking at the world?

- To learn visual categories from just **a few examples**?

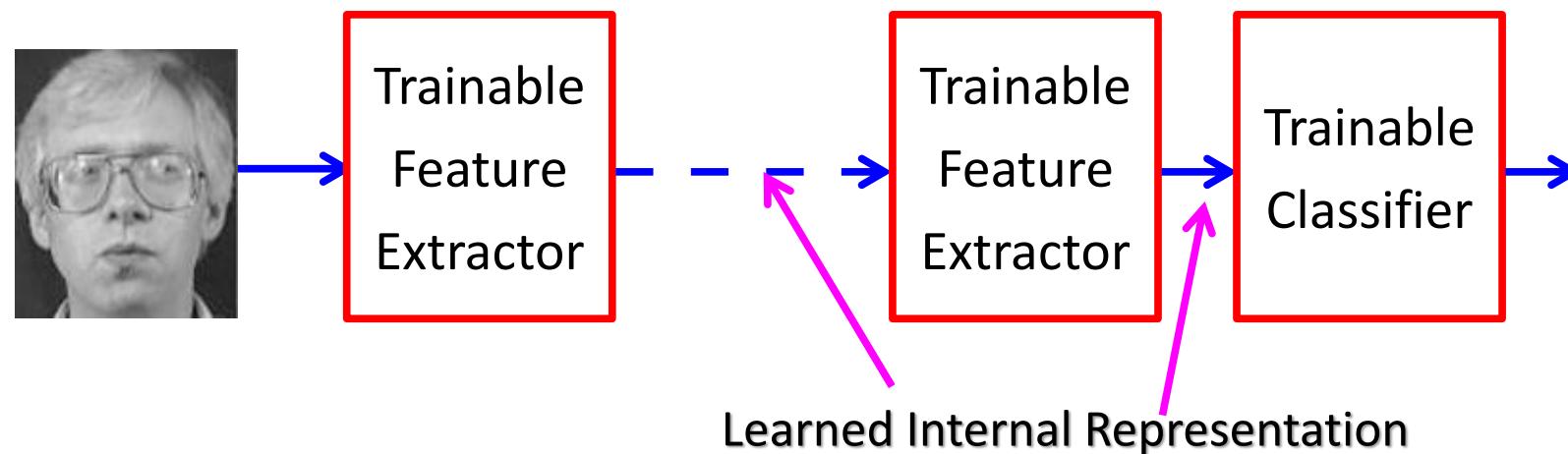
- ▶ I don't need to see many airplanes before I can recognize every airplane (even really weird ones)



Learning Hierarchical Representations



- Deep Learning: learning a hierarchy of internal representations
- Complexity: From low-level features to mid-level invariant representations, to object identities
- Invariance: Representations are increasingly invariant as we go up the layers
- Using multiple stages gets around the specificity/invariance dilemma



Do we really need deep architectures?



- We can approximate any function as close as we want with shallow architecture.
Why would we need deep ones?

$$y = \sum_{i=1}^P \alpha_i K(X, X^i) \quad y = F(W^1 \cdot F(W^0 \cdot X))$$

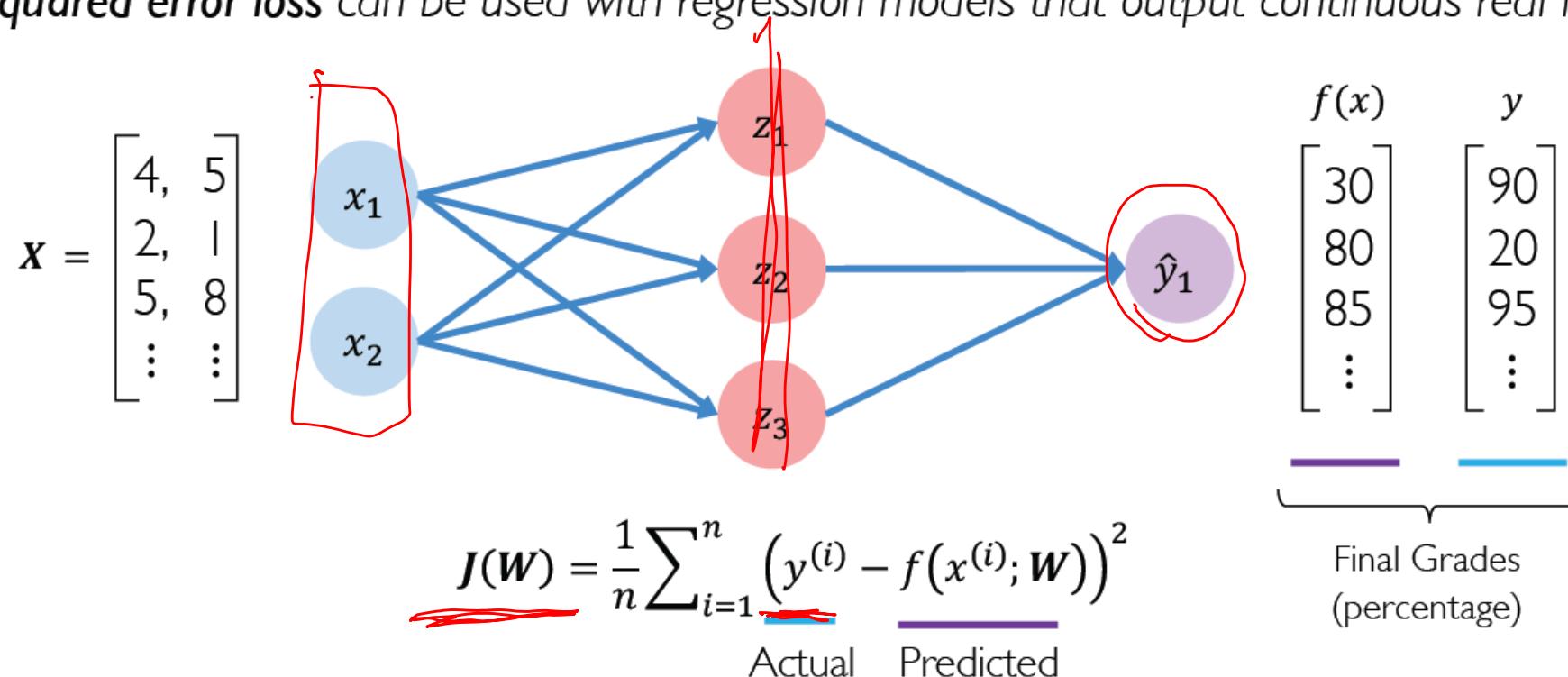
- ▶ kernel machines and 2-layer neural net are “universal”.
- Deep learning machines
$$y = F(W^K \cdot F(W^{K-1} \cdot F(\dots F(W^0 \cdot X) \dots)))$$
- Deep machines are more efficient for representing certain classes of functions, particularly those involved in visual recognition
 - Represent more complex functions with less “hardware”
 - Efficient parameterization of the class of functions that are useful for “AI” tasks.

Loss function

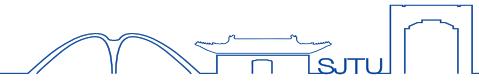


Mean Squared Error Loss

Mean squared error loss can be used with regression models that output continuous real numbers



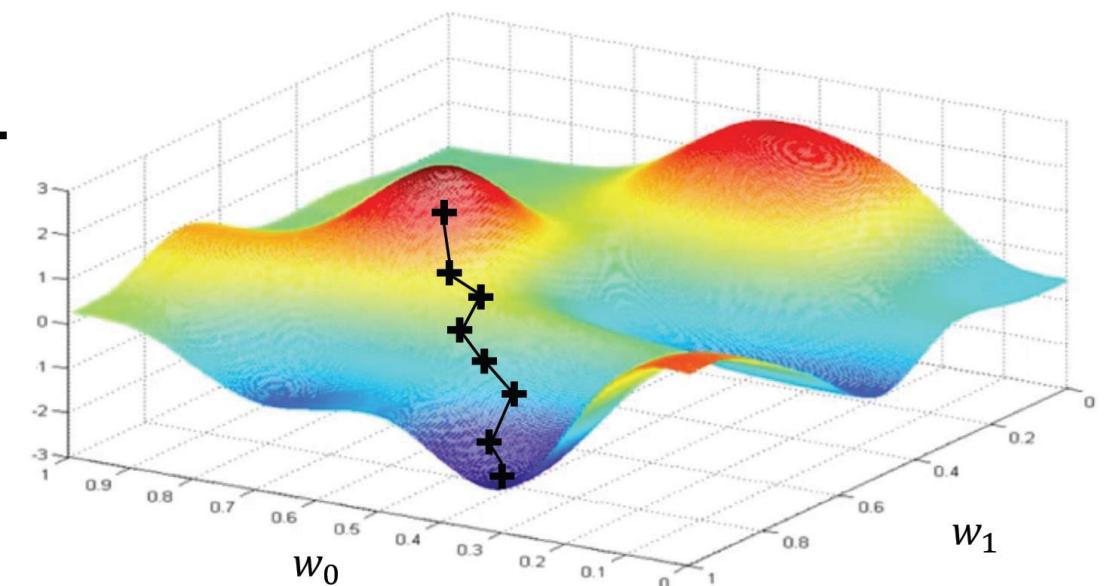
Learning Algorithm



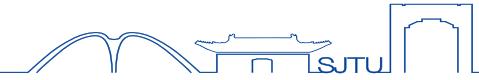
Algorithm

1. Initialize weights randomly $\sim \mathcal{N}(0, \sigma^2)$
2. Loop until convergence:
3. Compute gradient, $\frac{\partial J(\mathbf{W})}{\partial \mathbf{W}}$
4. Update weights, $\mathbf{W} \leftarrow \mathbf{W} - \eta \frac{\partial J(\mathbf{W})}{\partial \mathbf{W}}$
5. Return weights

Repeat until convergence

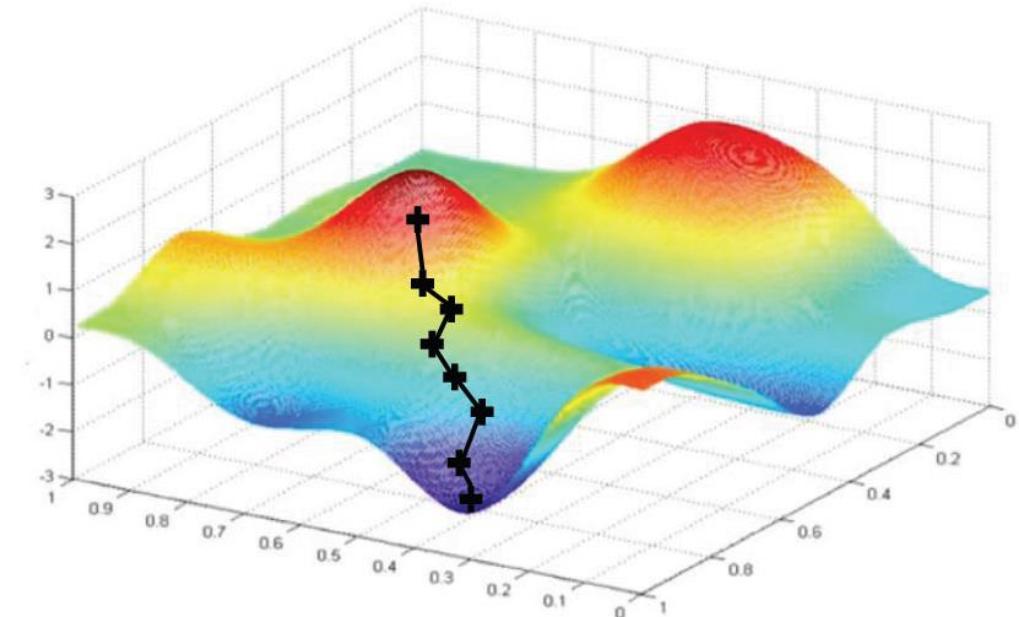


Stochastic Gradient Descent

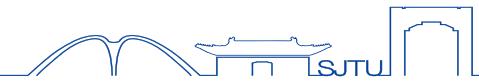


Algorithm

1. Initialize weights randomly $\sim \mathcal{N}(0, \sigma^2)$
2. Loop until convergence:
3. Pick single data point i
4. Compute gradient, $\frac{\partial J_i(\mathbf{W})}{\partial \mathbf{W}}$
5. Update weights, $\mathbf{W} \leftarrow \mathbf{W} - \eta \frac{\partial J(\mathbf{W})}{\partial \mathbf{W}}$
6. Return weights

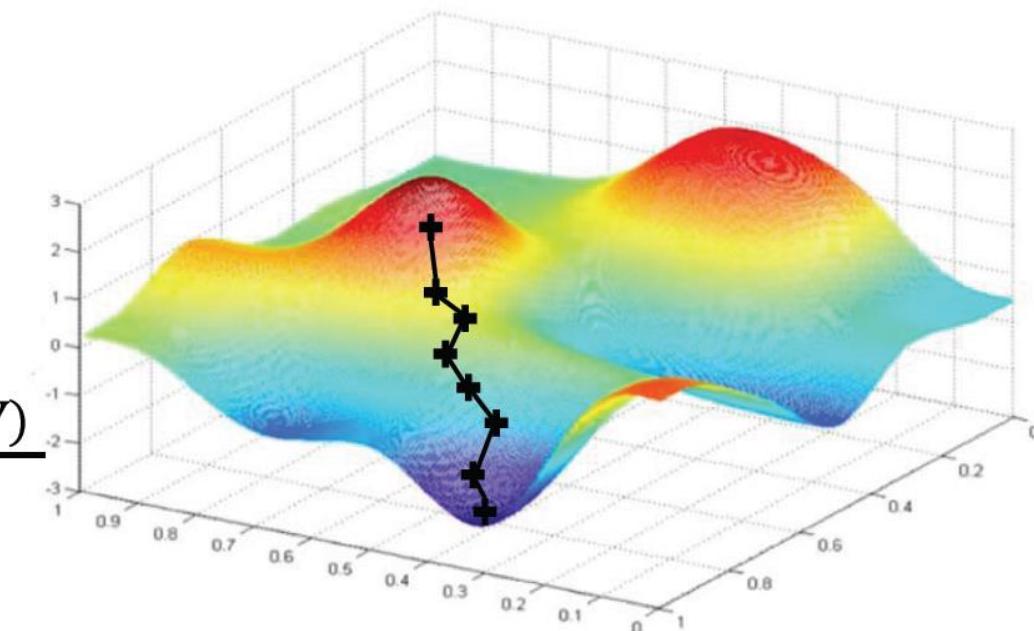


Mini-Batch



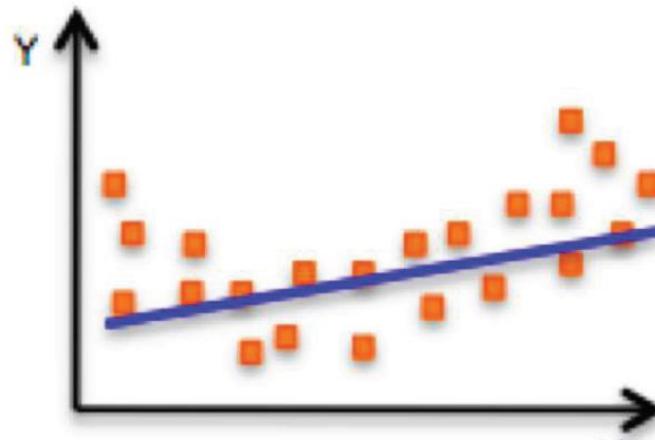
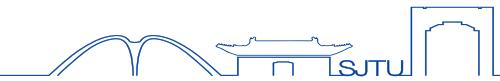
Algorithm

1. Initialize weights randomly $\sim \mathcal{N}(0, \sigma^2)$
2. Loop until convergence:
3. Pick batch of B data points
4. Compute gradient, $\frac{\partial J(\mathbf{W})}{\partial \mathbf{W}} = \frac{1}{B} \sum_{k=1}^B \frac{\partial J_k(\mathbf{W})}{\partial \mathbf{W}}$
5. Update weights, $\mathbf{W} \leftarrow \mathbf{W} - \eta \frac{\partial J(\mathbf{W})}{\partial \mathbf{W}}$
6. Return weights

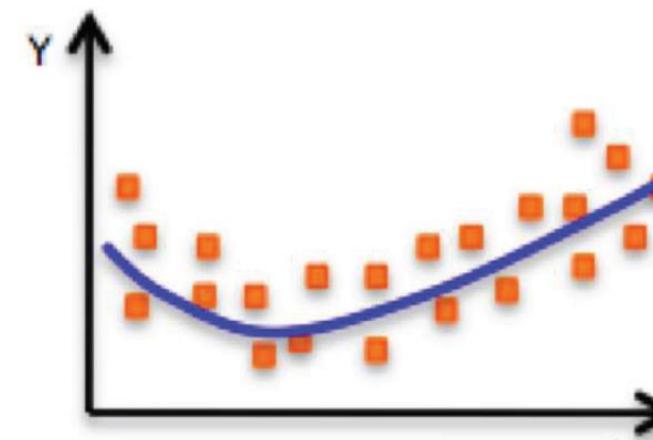




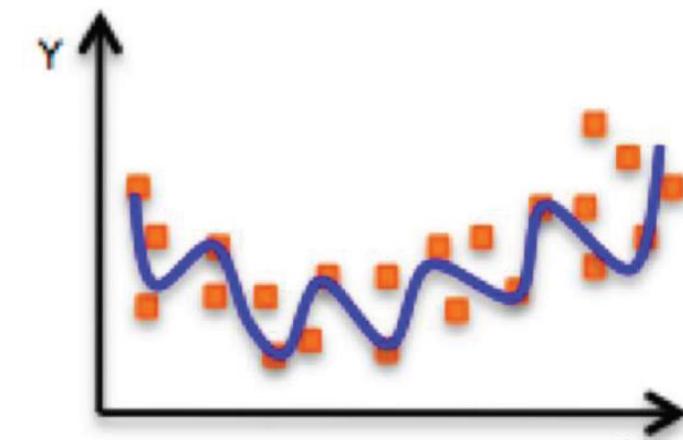
Overfitting Problem



Underfitting
Model does not have capacity
to fully learn the data



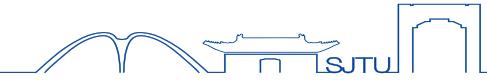
← **Ideal fit** →



Overfitting
Too complex, extra parameters,
does not generalize well

Regularization: Improve generalization of our model on unseen data

Dropout Regularization



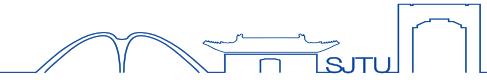
- **Dropout regularization** is used for reducing overfitting and improving the generalization of deep neural networks.
- Large weights in a neural network are a sign of a more complex network that has overfit the training data.
- **Probabilistically dropping out** nodes in the network is a simple and effective regularization method.
- A large network with more training and the use of a weight constraint are suggested when using dropout.

Problem With Overfitting



- Large neural nets trained on relatively small datasets can overfit the training data.
The statistical noise in the training data results in poor performance when the model is evaluated on new data, e.g. a test dataset
- One approach to reduce overfitting is to fit all possible different neural networks on the same dataset and to **average the predictions** from each model.
 - This is not feasible in practice, and can be approximated using a small collection of different models, called an ensemble.
 - The best way to “regularize” a fixed-sized model is to **average the predictions of all possible settings of the parameters**, weighting each setting by its posterior probability given the training data.

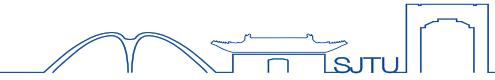
Randomly Drop Nodes



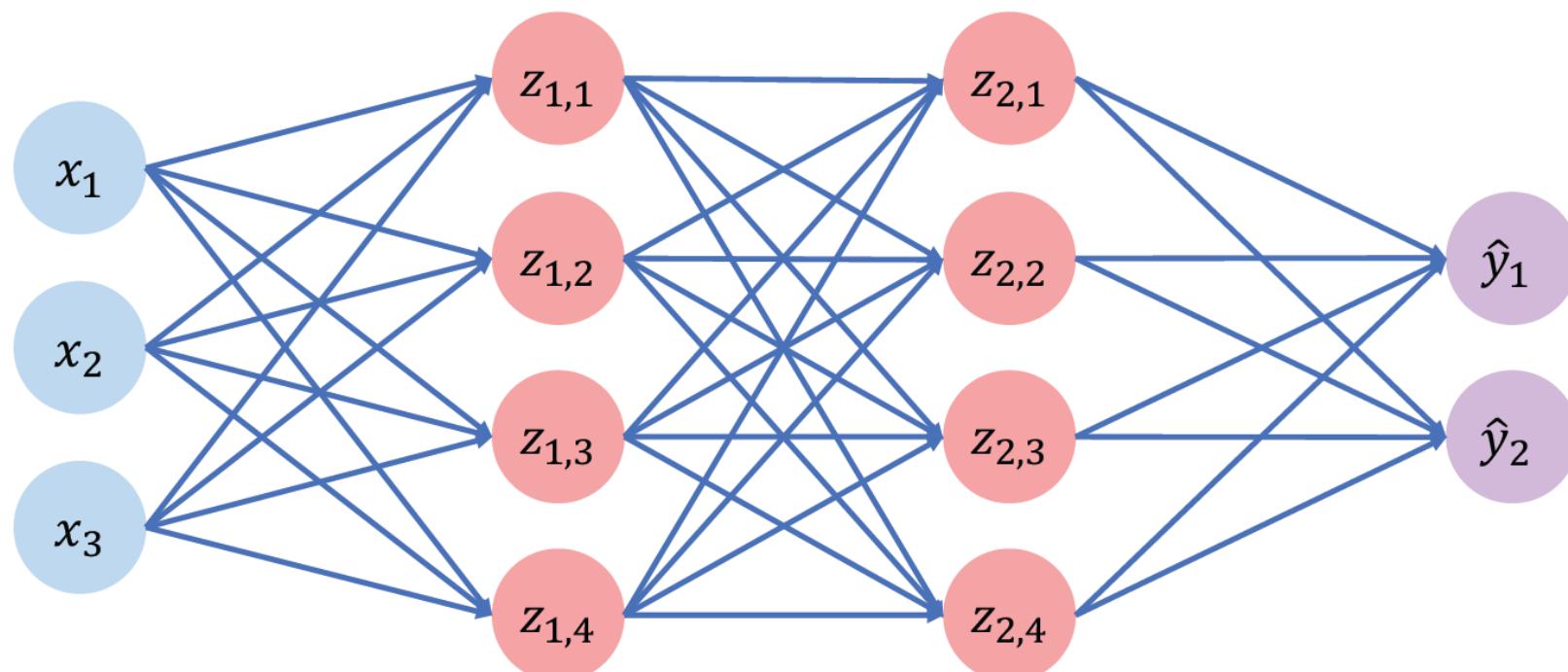
- Dropout is a regularization method that approximates training a large number of neural networks with different architectures in parallel.
 - During training, some number of layer outputs are randomly ignored or “dropped out.”
- By dropping a unit out, we mean temporarily removing it from the network, along with all its incoming and outgoing connections
 - Dropout has the effect of making the training process noisy, forcing nodes within a layer to probabilistically take on more or less responsibility for the inputs.



Regularization: Dropout

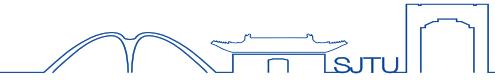


- During training, randomly set some activations to 0





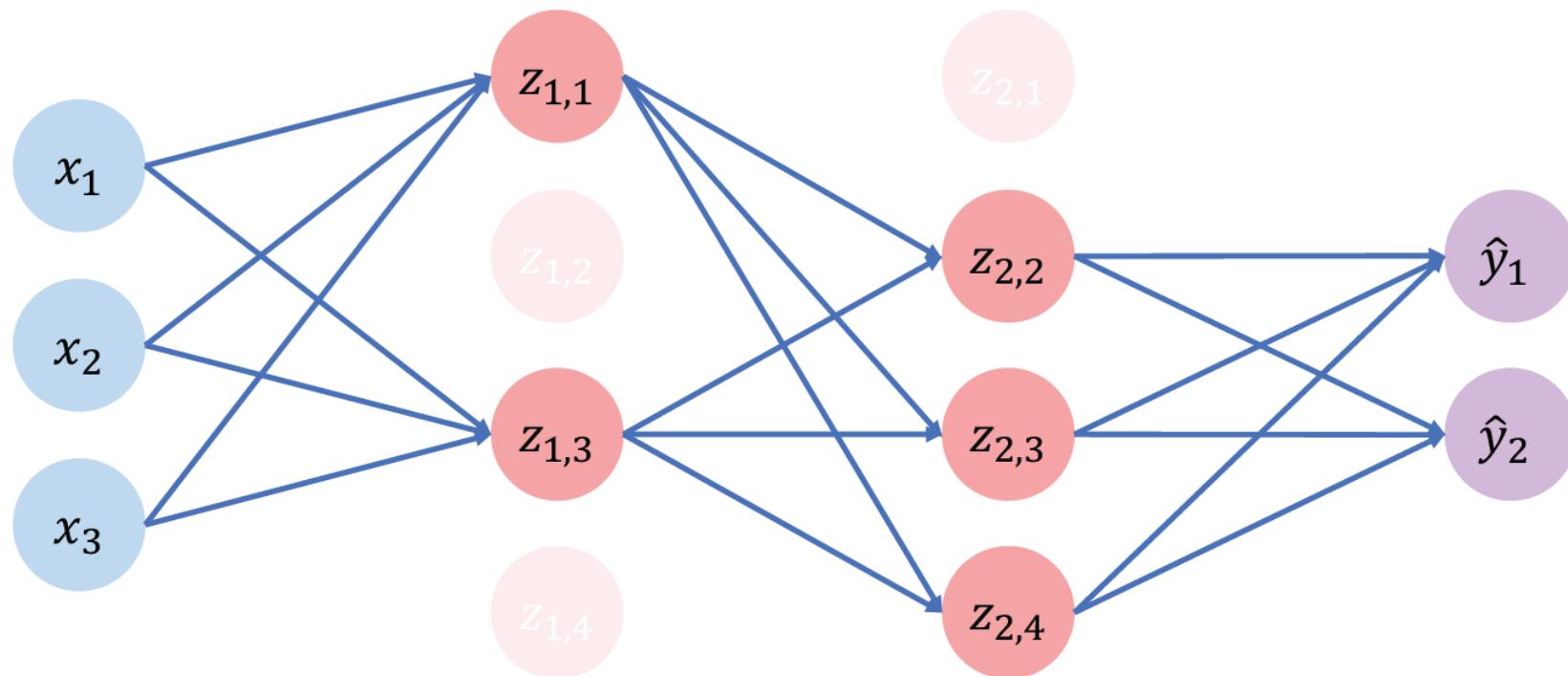
Regularization: Dropout



- During training, randomly set some activations to 0
 - Typically ‘drop’ 50% of activations in layer
 - Forces network to not rely on any 1 node

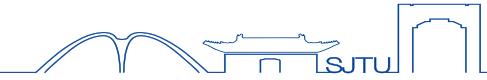


tf.keras.layers.Dropout (p=0.5)



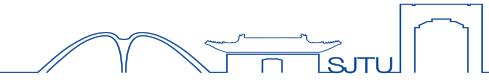


Properties of Dropout



- Dropout simulates **a sparse activation** from a given layer, which interestingly, in turn, encourages the network to actually learn a sparse representation as a side-effect.
 - An alternative to activity regularization for encouraging sparse representations in autoencoder models.
- The outputs of a layer under dropout are randomly subsampled, it has the effect of reducing the capacity or thinning the network during training.
 - As such, a wider network, e.g. more nodes, may be required when using dropout.

Tips for Using Dropout Regularization



▪ Dropout Rate

- The default interpretation of the dropout hyperparameter is the probability of training a given node in a layer
- 1.0 means no dropout, and 0.0 means no outputs from the layer.
- **A good value for dropout** in a hidden layer is **between 0.5 and 0.8**. Input layers use a larger dropout rate, such as of 0.8.

▪ Use a Larger Network

- Larger networks (more layers or nodes) to more easily overfit the training data.
- When using dropout regularization, it is possible to use larger networks with less risk of overfitting. A large network (more nodes per layer) may be required as dropout will probabilistically reduce the capacity of the network.



B-VAE –解耦表征

- Variational AutoEncoder

$$\max_{\phi, \theta} \mathbb{E}_{q_\phi(z|x)} [\log p_\theta(x|z)]$$

- Using KL divergence, we have

$$\log p_\theta(x|z) = D_{KL}(q_\phi(z|x) \| p(z)) + L(\theta, \phi; x, z)$$

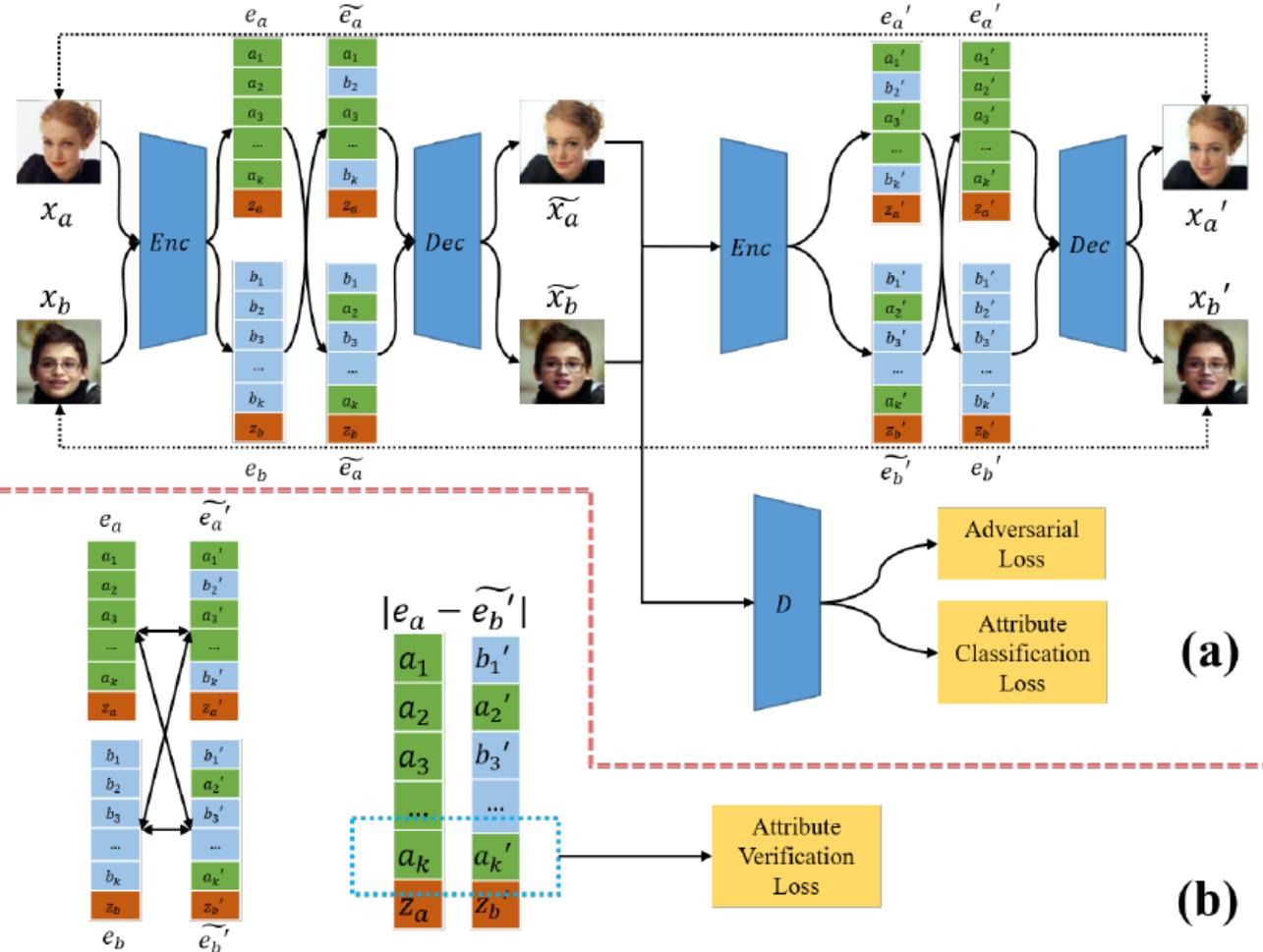
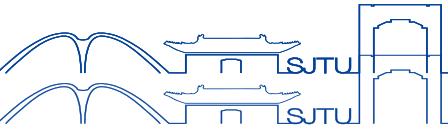
- Lower Bound maximization

$$\max_{\theta, \phi} L(\theta, \phi; x, z) = \log p_\theta(x|z) - D_{KL}(q_\phi(z|x) \| p(z))$$

- B-VAE

$$L(\theta, \phi; x, z, \beta) = \log p_\theta(x|z) - \beta D_{KL}(q_\phi(z|x) \| p(z))$$

对比解耦表征与属性转换



(a)

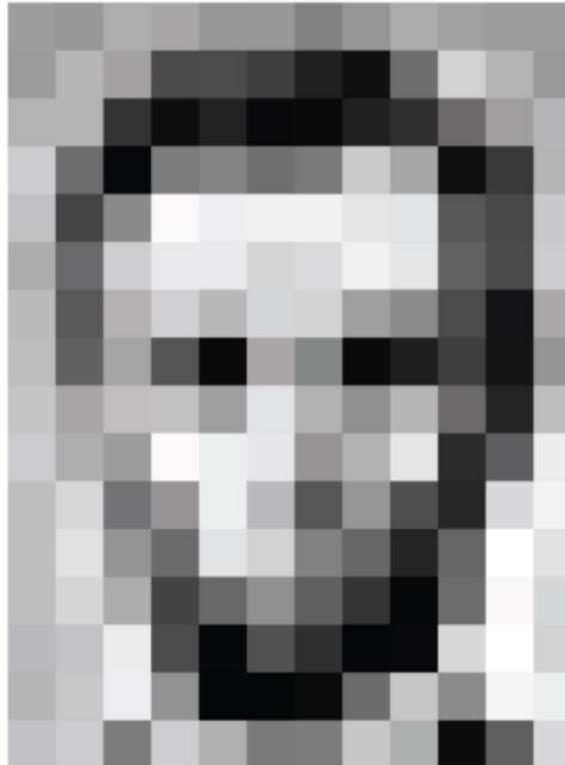
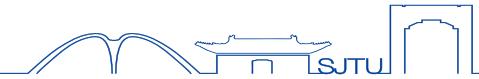
(b)



(c) CelebA: hair color and smiling



Deep Learning for Vision



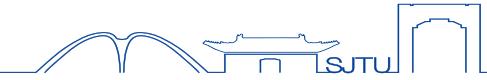
157	153	174	168	150	152	129	151	172	161	155	156
155	182	163	74	75	62	33	17	110	210	180	154
180	180	50	14	34	6	10	33	48	106	169	181
206	109	5	124	131	111	120	204	166	15	56	180
194	68	137	251	237	239	239	228	227	67	71	201
172	106	207	233	233	214	220	239	228	98	74	206
188	88	179	209	185	215	211	158	139	75	20	169
189	97	165	84	10	168	134	11	31	62	22	148
199	168	191	193	158	227	178	143	182	105	36	190
205	174	155	252	236	231	149	178	228	43	95	234
190	216	116	149	236	187	85	150	79	38	218	241
190	224	147	108	227	210	127	102	36	101	255	224
190	214	173	66	103	143	95	50	2	109	249	215
187	196	235	75	1	81	47	0	6	217	255	211
183	202	237	145	0	0	12	108	200	138	243	236
195	206	123	207	177	121	123	200	175	13	96	218

What the computer sees

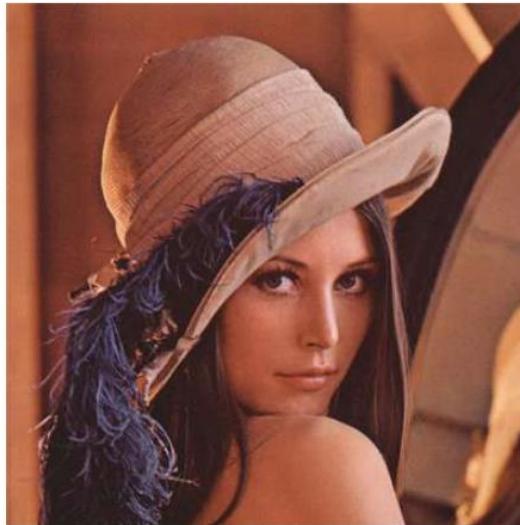
157	153	174	168	150	152	129	151	172	161	155	156
155	182	163	74	75	62	33	17	110	210	180	154
180	180	50	14	34	6	10	33	48	106	159	181
206	109	5	124	131	111	120	204	166	15	56	180
194	68	137	251	237	239	239	228	227	67	71	201
172	106	207	233	233	214	220	239	228	98	74	206
188	88	179	209	185	215	211	158	139	75	20	169
189	97	165	84	10	168	134	11	31	62	22	148
199	168	191	193	158	227	178	143	182	105	36	190
205	174	155	252	236	231	149	178	228	43	95	234
190	216	116	149	236	187	85	150	79	38	218	241
190	224	147	108	227	210	127	102	36	101	255	224
190	214	173	66	103	143	95	50	2	109	249	215
187	196	235	75	1	81	47	0	6	217	255	211
183	202	237	145	0	0	12	108	200	138	243	236
195	206	123	207	177	121	123	200	175	13	96	218

An image is just a matrix of numbers [0,255]!
i.e., 1080x1080x3 for an RGB image

High Level Feature Detection



Let's identify key features in each image category



Nose,
Eyes,
Mouth

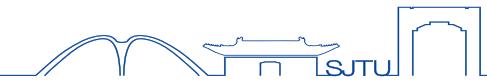


Wheels,
License Plate,
Headlights



Door,
Windows,
Steps

Manual Feature Extraction Domain



- Problems: ???

Domain knowledge

Define features

Detect features
to classify

Viewpoint variation



Scale variation



Deformation



Occlusion



Illumination conditions



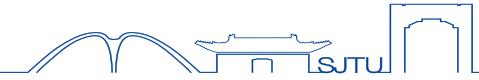
Background clutter



Intra-class variation

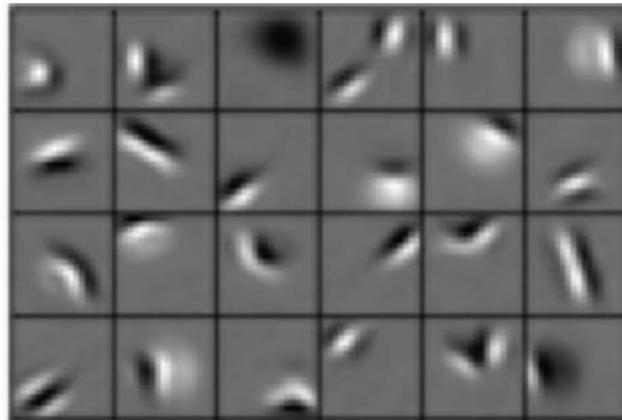


Feature Representation



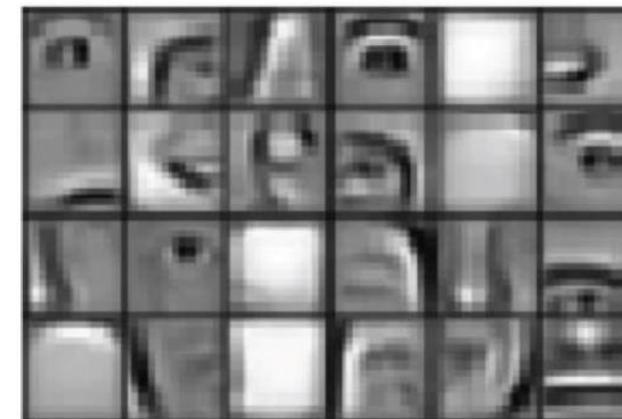
Can we learn a **hierarchy of features** directly from the data instead of hand engineering?

Low level features



Edges, dark spots

Mid level features



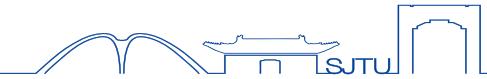
Eyes, ears, nose

High level features

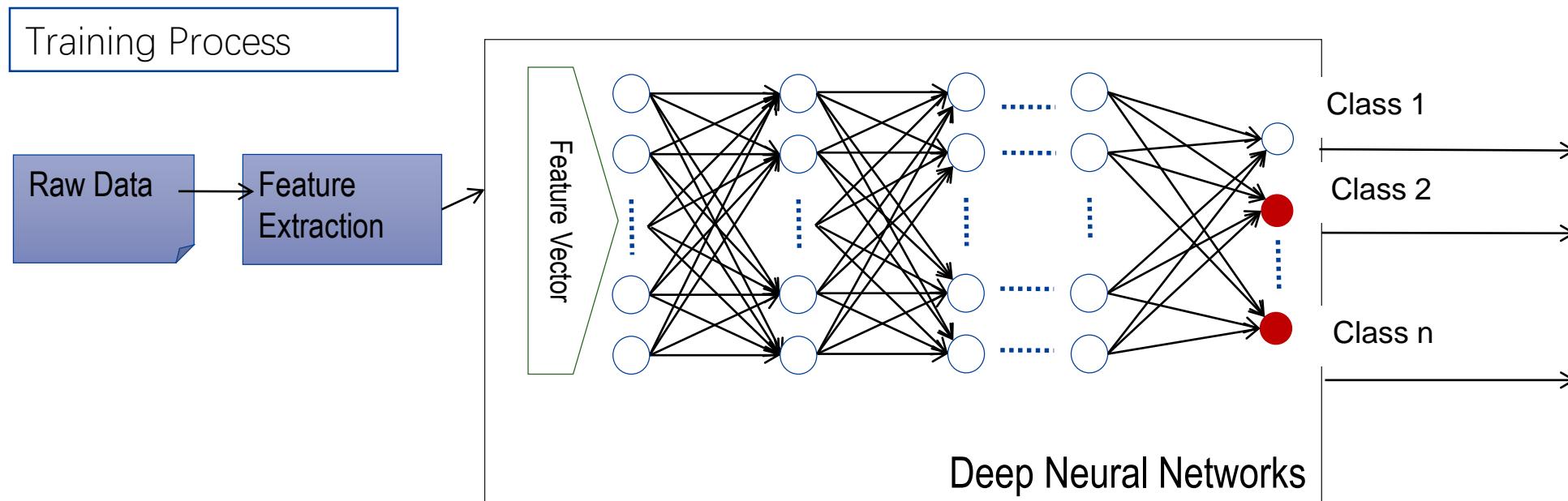


Facial structure

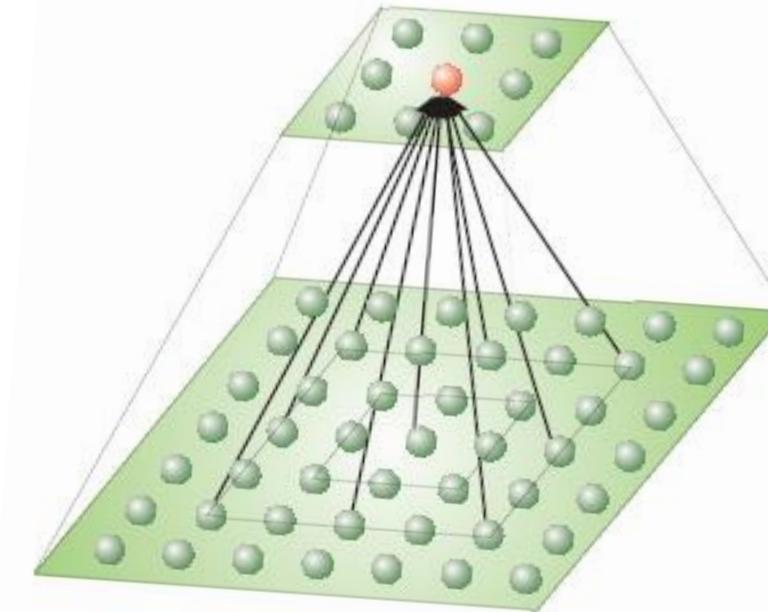
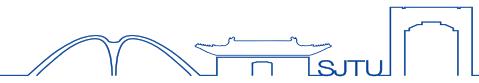
Feedforward Neural Network



- Image is represented in **2-dim** data structures.
- How to design Deep Neural Network for Image Feature Extraction



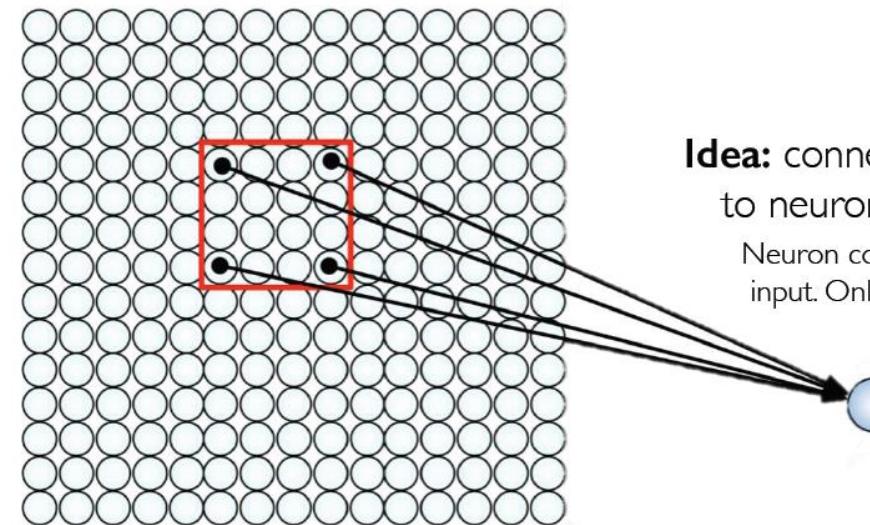
Cortical Networks – Spatial Convolution



Spatial Structure Revervation

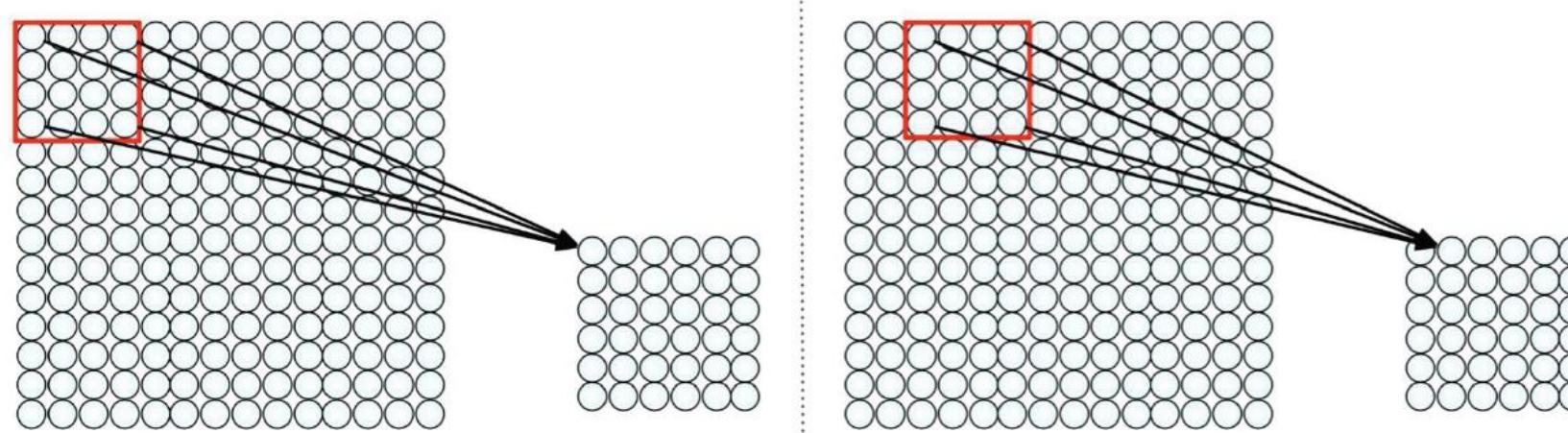


Input: 2D image.
Array of pixel values

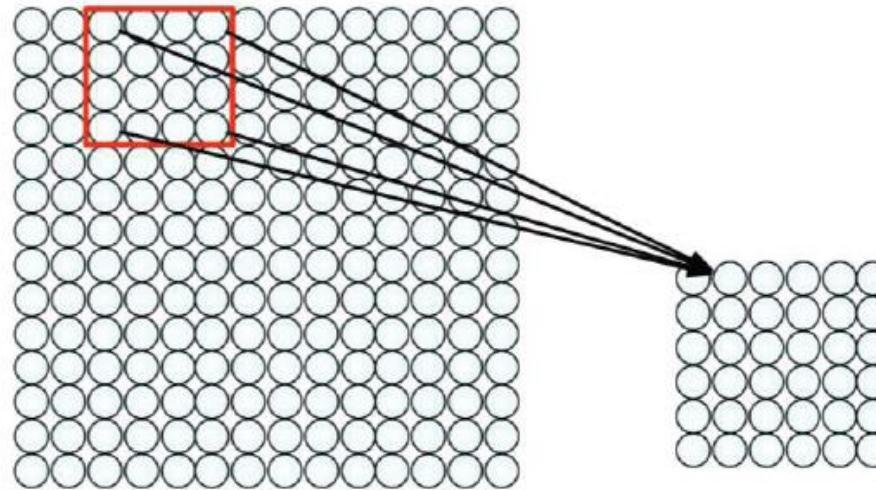


Idea: connect patches of input
to neurons in hidden layer.

Neuron connected to region of
input. Only “sees” these values.



Feature Extraction with Convolution



- Filter of size 4×4 : 16 different weights
- Apply this same filter to 4×4 patches in input
- Shift by 2 pixels for next patch

This “patchy” operation is **convolution**

- 1) Apply a set of weights – a filter – to extract **local features**
- 2) Use **multiple filters** to extract different features
- 3) **Spatially share** parameters of each filter

Feature Matching



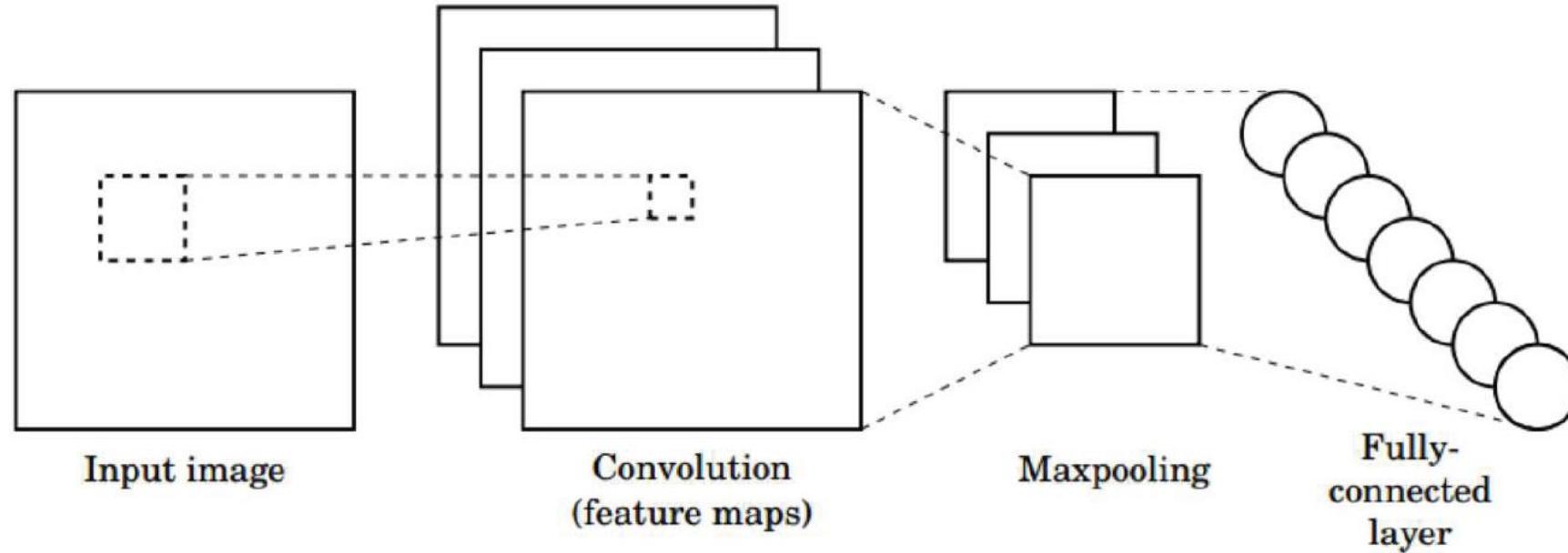
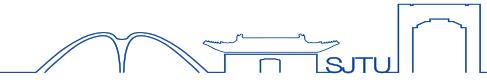
-1	-1	-1	-1	-1	-1	-1	-1	-1	-1
-1	1	-1	-1	-1	-1	-1	1	-1	-1
-1	-1	1	-1	-1	-1	1	-1	-1	-1
-1	-1	-1	1	-1	1	-1	-1	-1	-1
-1	-1	-1	-1	1	-1	-1	-1	-1	-1
-1	-1	-1	-1	1	-1	-1	-1	-1	-1
-1	-1	-1	1	-1	1	-1	-1	-1	-1
-1	-1	1	-1	-1	-1	1	-1	-1	-1
-1	1	-1	-1	-1	-1	-1	1	-1	-1
-1	-1	-1	-1	-1	-1	-1	-1	-1	-1



-1	-1	-1	-1	-1	-1	-1	-1	-1	-1
-1	-1	-1	-1	-1	-1	-1	1	-1	-1
-1	1	-1	-1	-1	-1	1	-1	-1	-1
-1	-1	1	1	-1	1	-1	-1	-1	-1
-1	-1	-1	-1	1	-1	1	-1	-1	-1
-1	-1	-1	-1	1	-1	1	-1	-1	-1
-1	-1	-1	1	-1	1	-1	1	-1	-1
-1	-1	1	-1	-1	-1	1	-1	-1	-1
-1	1	-1	-1	-1	-1	-1	1	-1	-1
-1	-1	-1	-1	-1	-1	-1	-1	-1	-1

Image is represented as matrix of pixel values... and computers are literal!
We want to be able to classify an X as an X even if it's shifted, shrunk, rotated, deformed.

CNN for Classification

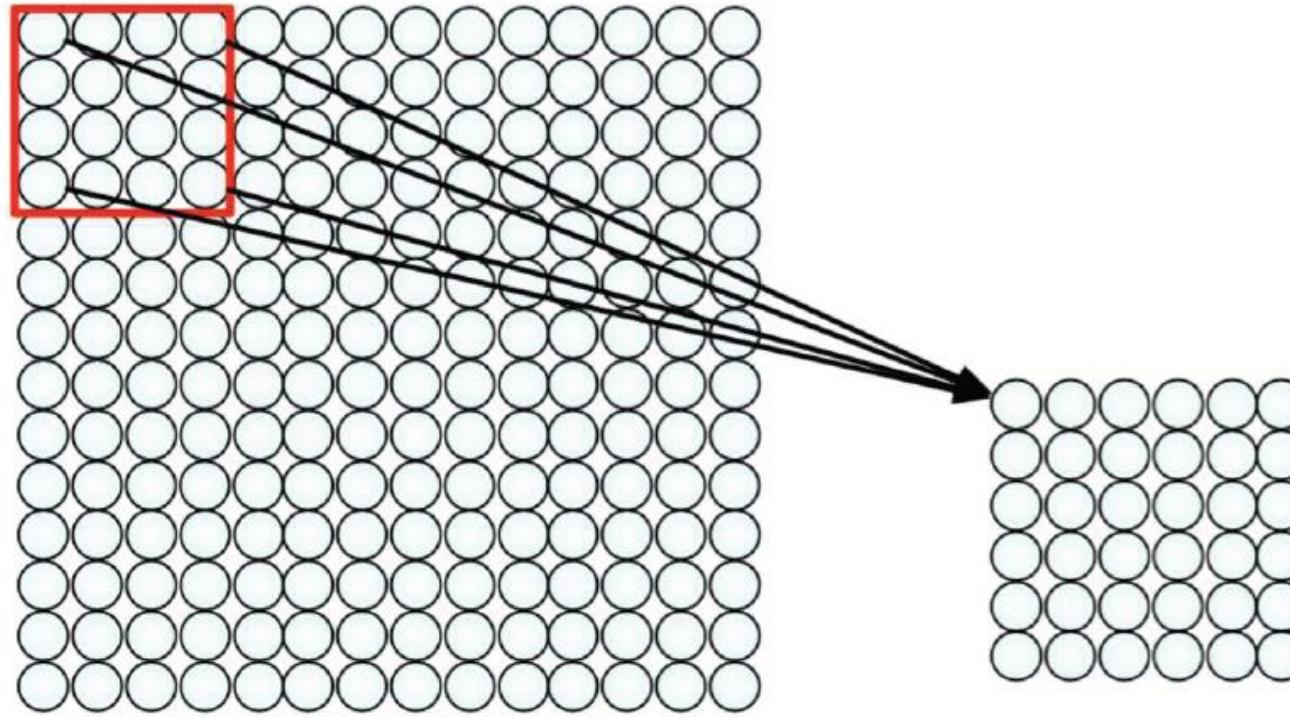


1. **Convolution:** Apply filters with learned weights to generate feature maps.
2. **Non-linearity:** Often ReLU.
3. **Pooling:** Downsampling operation on each feature map.

Train model with image data.

Learn weights of filters in convolutional layers.

Convolutional Layers: Local Connectivity



4x4 filter: matrix
of weights w_{ij}

$$\sum_{i=1}^4 \sum_{j=1}^4 w_{ij} x_{i+p,j+q} + b$$

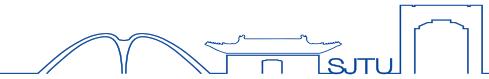
for neuron (p,q) in hidden layer

For a neuron in hidden layer:

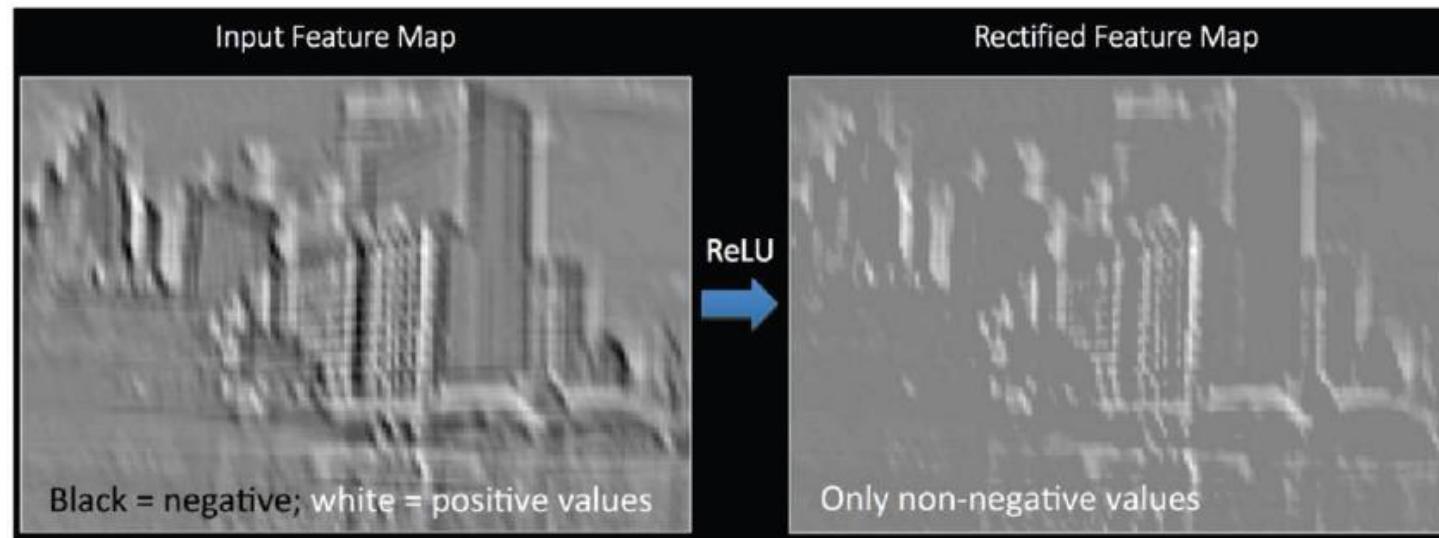
- Take inputs from patch
- Compute weighted sum
- Apply bias

- 1) applying a window of weights
- 2) computing linear combinations
- 3) activating with non-linear function

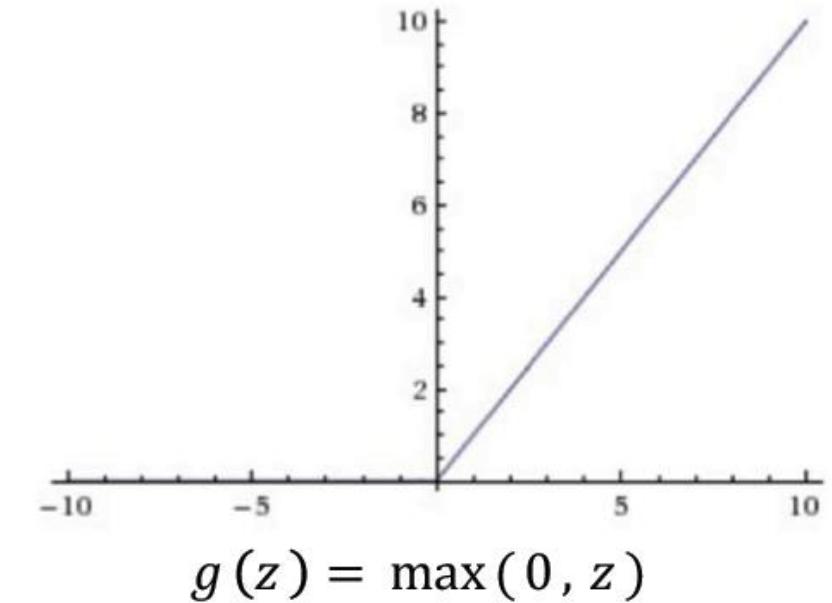
Non-Linearity



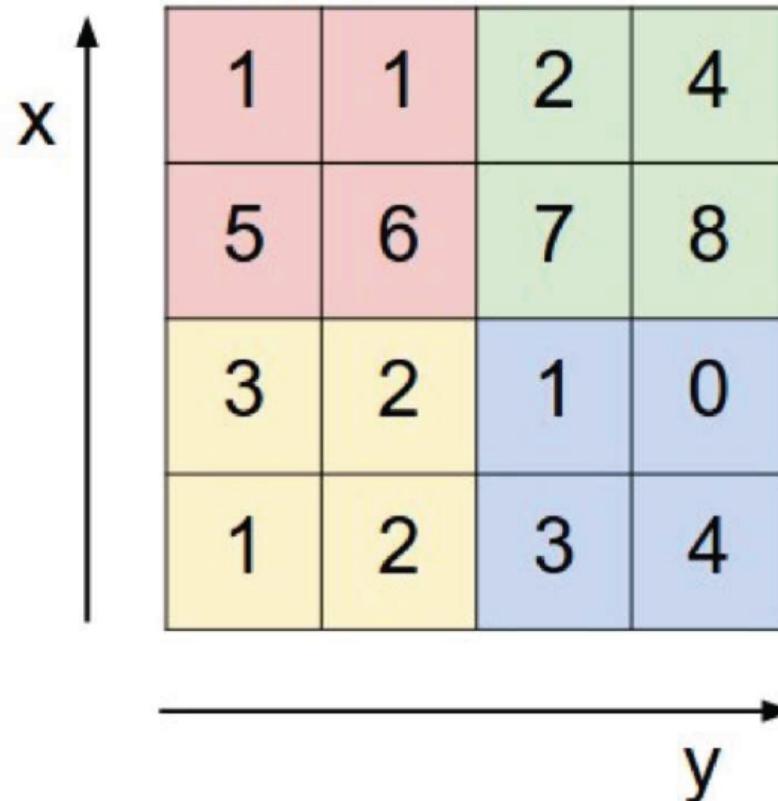
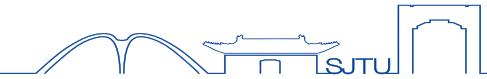
- Apply after every convolution operation (i.e., after convolutional layers)
- ReLU: pixel-by-pixel operation that replaces all negative values by zero. **Non-linear operation!**



Rectified Linear Unit (ReLU)



Pooling



max pool with 2x2 filters
and stride 2

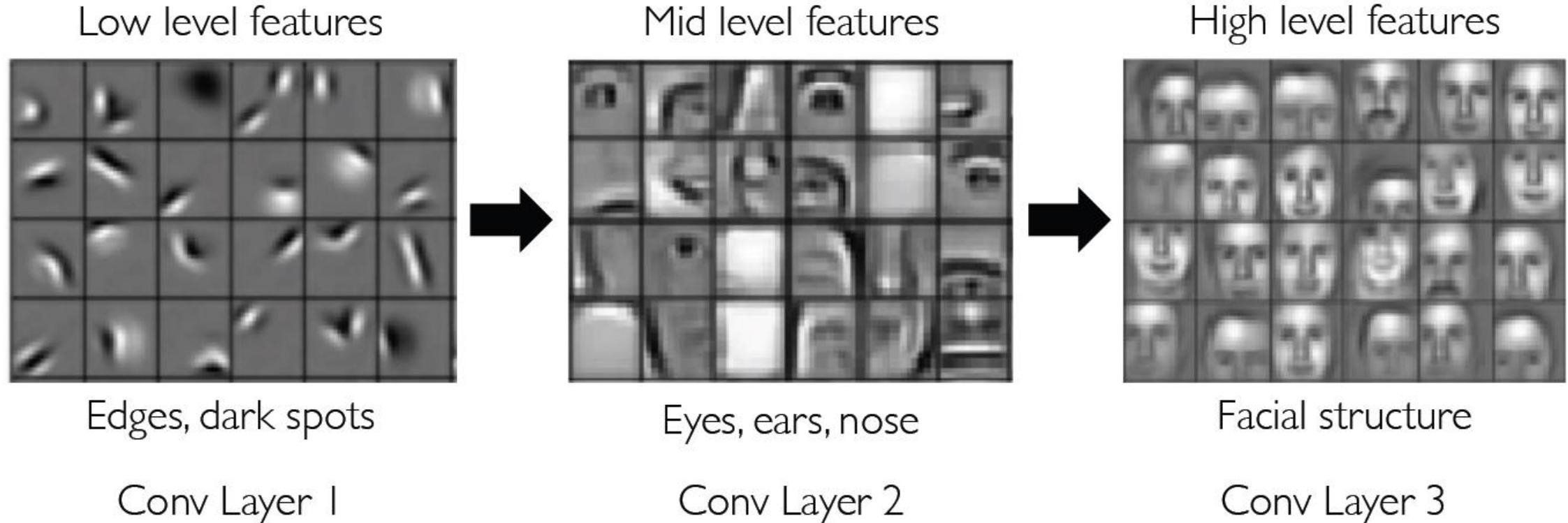
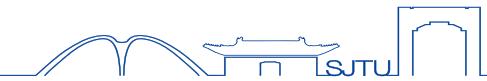


6	8
3	4

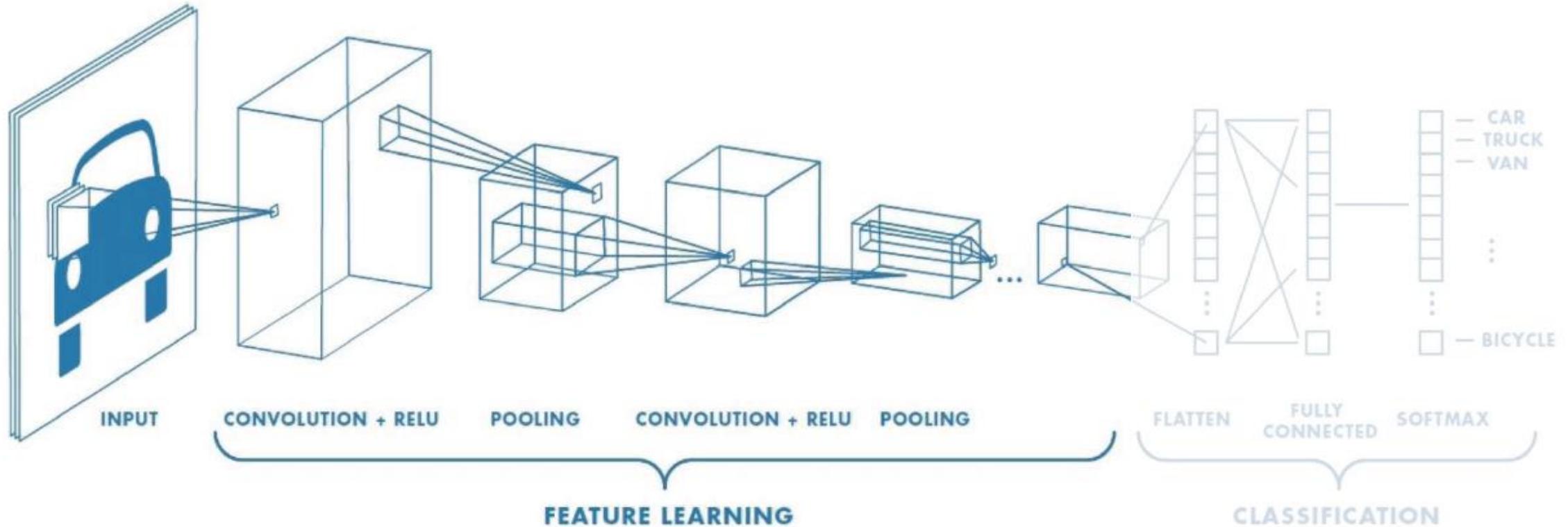
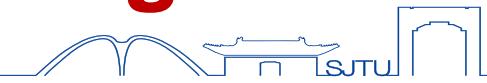
- 1) Reduced dimensionality
- 2) Spatial invariance

How else can we downsample and preserve spatial invariance?

Representation Learning in Deep CNNs

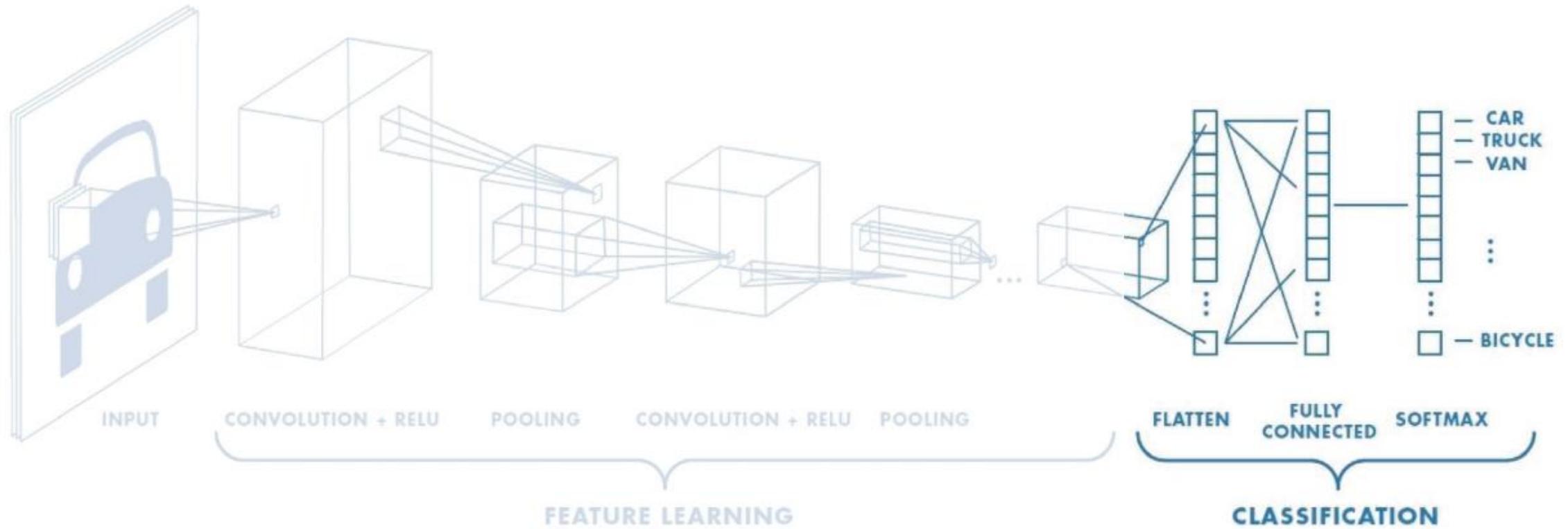


CNNs for Classification: Feature Learning



1. Learn features in input image through **convolution**
2. Introduce **non-linearity** through activation function (real-world data is non-linear!)
3. Reduce dimensionality and preserve spatial invariance with **pooling**

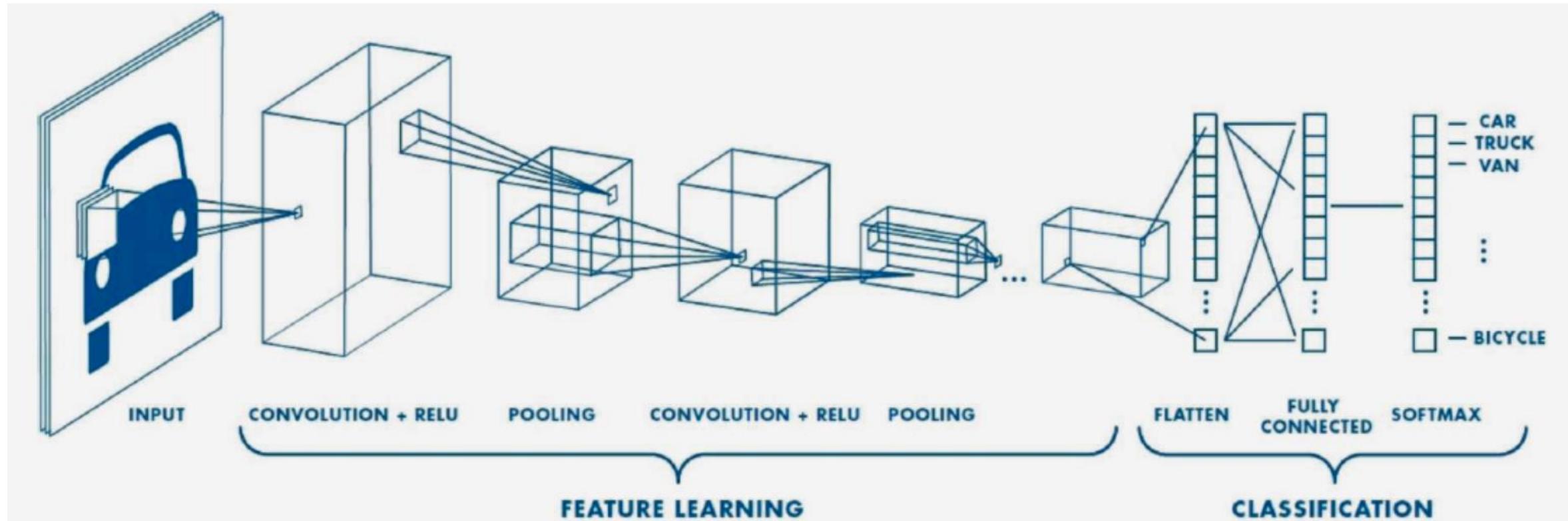
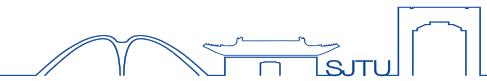
CNNs for Classification: Class Probabilities



- CONV and POOL layers output high-level features of input
- Fully connected layer uses these features for classifying input image
- Express output as **probability** of image belonging to a particular class

$$\text{softmax}(y_i) = \frac{e^{y_i}}{\sum_j e^{y_j}}$$

CNNs: Training with Backpropagation

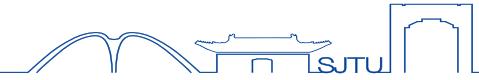


Learn weights for convolutional filters and fully connected layers

Backpropagation: cross-entropy loss

$$J(\theta) = \sum_i y^{(i)} \log(\hat{y}^{(i)})$$

ImageNet Dataset



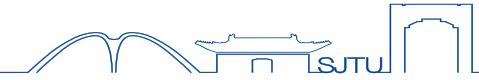
Dataset of over 14 million images across 21,841 categories

“Elongated crescent-shaped yellow fruit with soft sweet flesh”



1409 pictures of bananas.

ImageNet Challenge



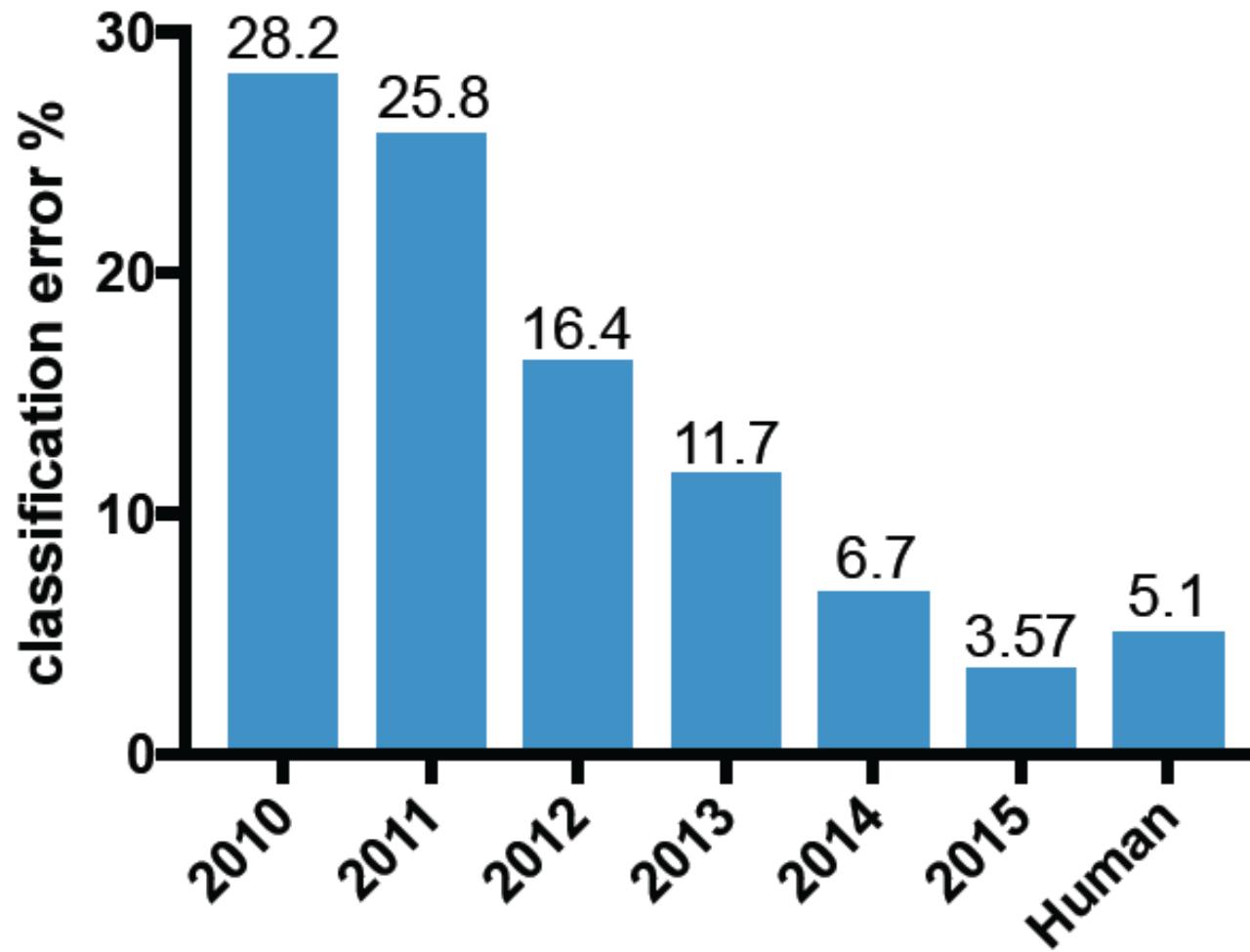
- Classification task: produce a list of object categories present in image. 1000 categories.
- “Top 5 error”: rate at which the model does not output correct label in top 5 predictions



ImageNet Large Scale Visual Recognition Challenges



ImageNet Challenge: Classification Task



2012: AlexNet. First CNN to win.

- 8 layers, 61 million parameters

2013: ZFNet

- 8 layers, more filters

2014: VGG

- 19 layers

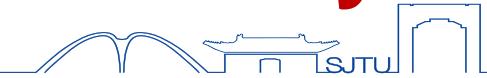
2014: GoogLeNet

- “Inception” modules
- 22 layers, 5 million parameters

2015: ResNet

- 152 layers

Deep Learning for Computer Vision: Summary



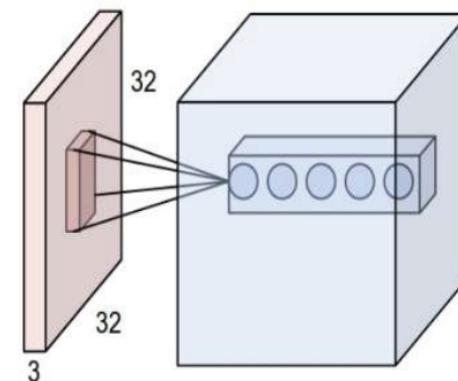
Foundations

- Why computer vision?
- Representing images
- Convolutions for feature extraction



CNNs

- CNN architecture
- Application to classification
- ImageNet



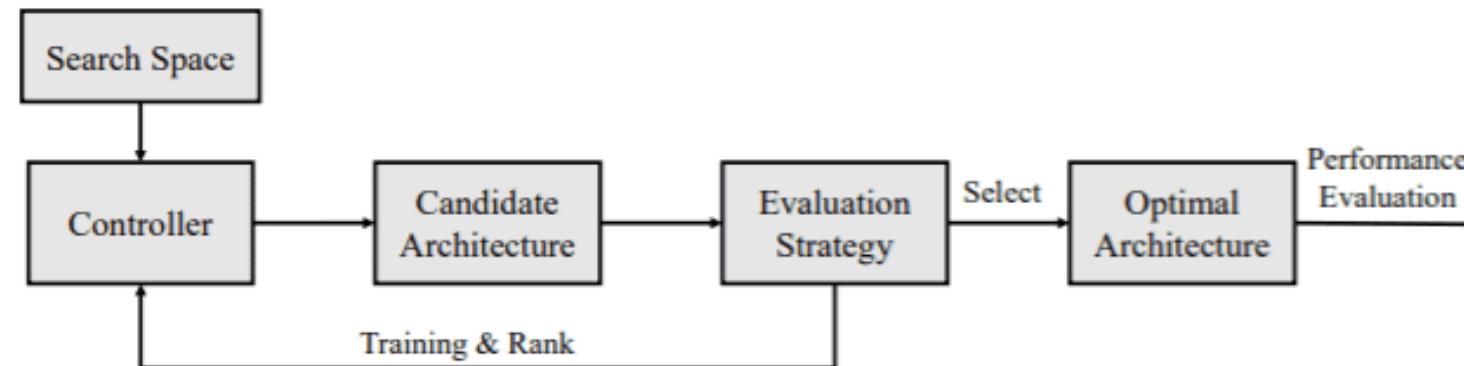
Applications

- Segmentation, object detection, image captioning
- Visualization



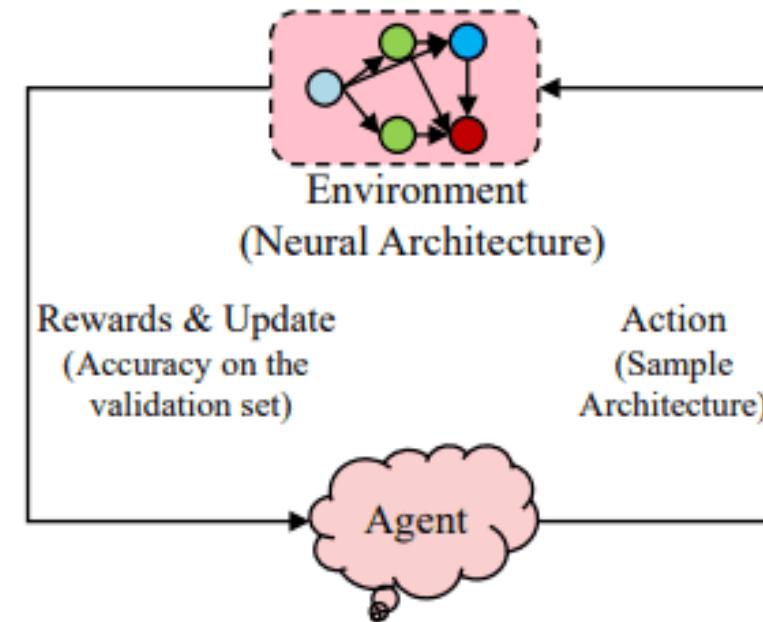
NAS algorithm (From Model selection to Search)

1. **Define search space:** a list of predefined operations (e.g.convolutional layers, recurrent, pooling, fully connected etc.) and their connections.
2. **Search Strategy:** a list of possible candidate architectures from the search space.
3. **Model Training:** The candidate architectures are trained and ranked based on their performance on the validation test.
4. **Model Ranking:** to re-adjust the search and obtain new candidates.
5. The process iterates until reaching a certain condition and provides the optimal architecture.
6. The optimal architecture is evaluated on the test set.



Search strategy

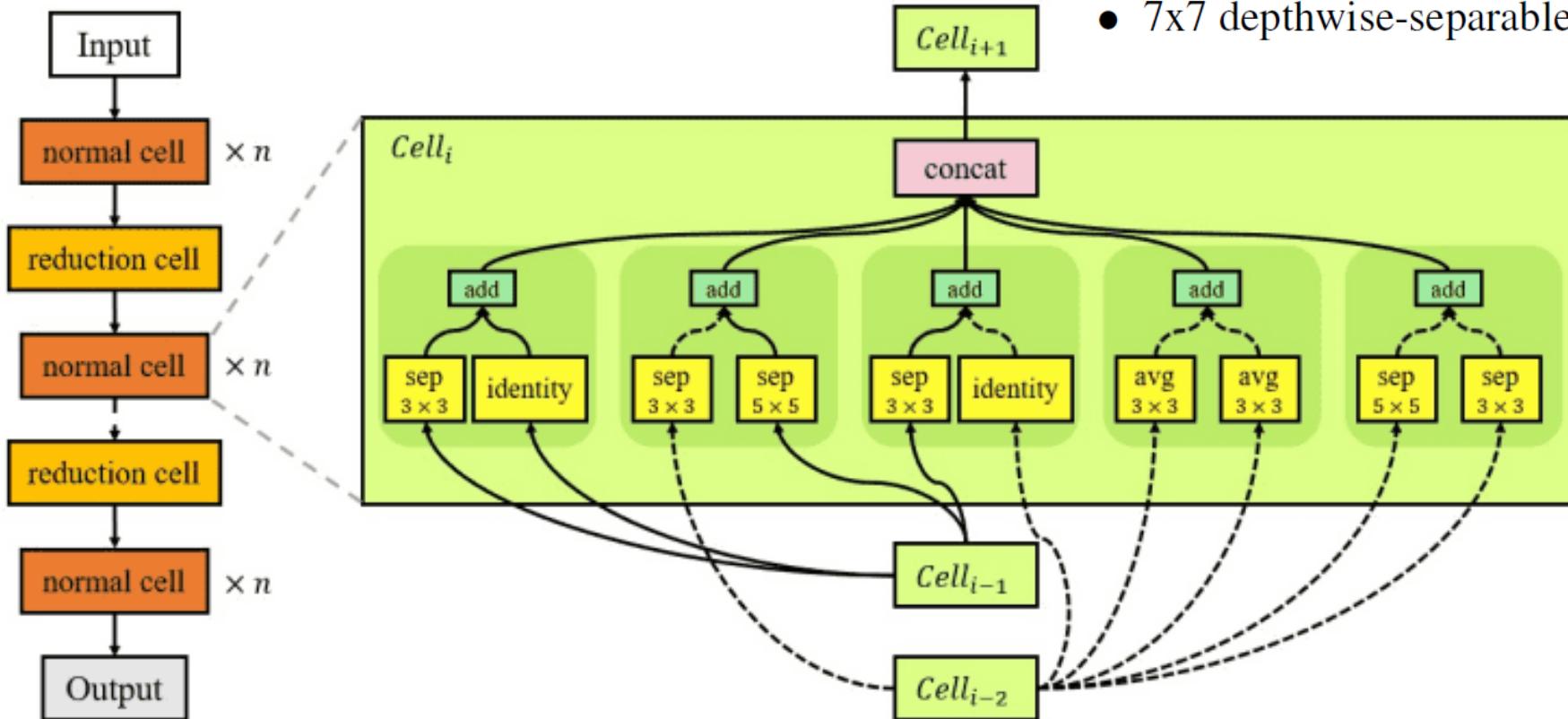
- **Search strategy:** the methodology used to search for the optimal architecture in the search space.
 1. Random search
 2. Reinforcement learning
 3. Evolutionary algorithms
 4. Sequential model-based optimization
 5. Gradient optimization
- Reinforcement learning



Modular search space

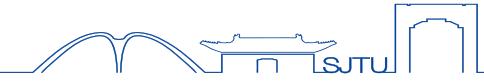
A normal cell that performs feature extraction and a reduction cell that downsamples the input

- identity
- 1x3 then 3x1 convolution
- 1x7 then 7x1 convolution
- 3x3 average pooling
- 5x5 max pooling
- 1x1 convolution
- 3x3 depthwise-separable conv
- 7x7 depthwise-separable conv



Summary

- Why CNN model is good for spatial feature extraction?
 - shared kernel, pooling, nonlinear transform
- How CNN model deals with the invariance of features?
- How CNN is regularized during training?
- How to implement the model selection for deep CNN model?



The End of Talk