Talk 13:

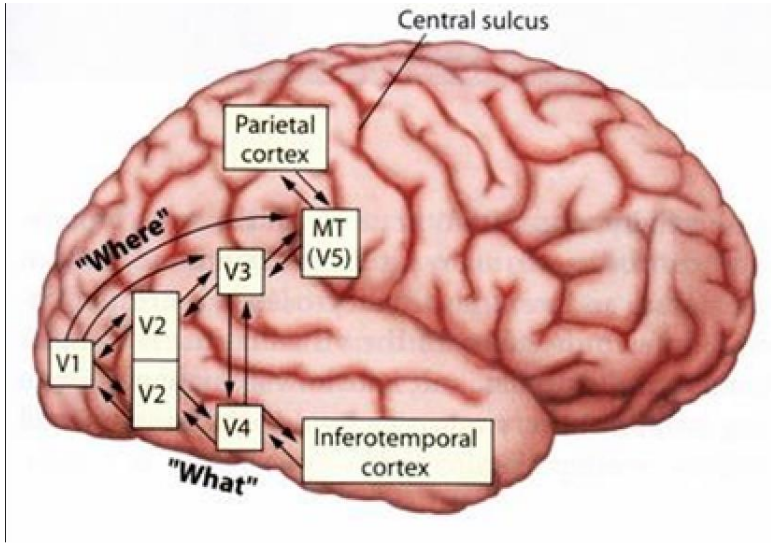
Deep Neural Networks and Regularization

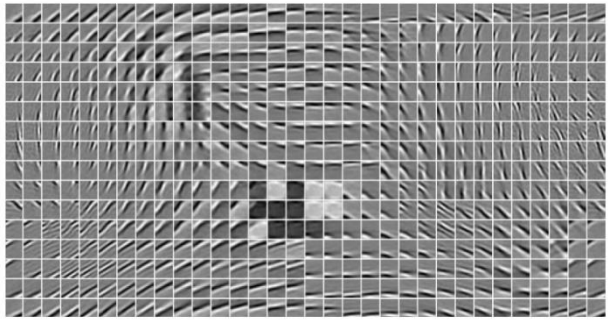
Contents

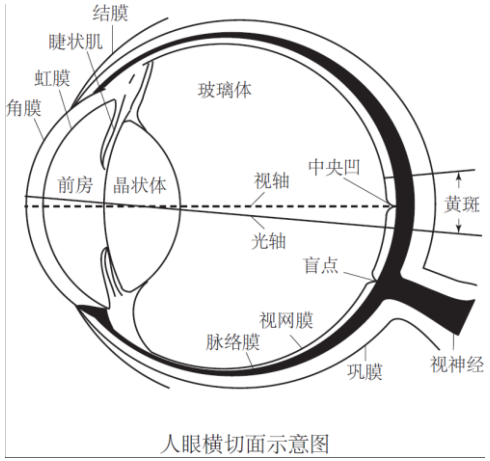
- Background
- Deep Neural Networks
  - Regularization - Dropout
- Disentangled Representation in DNN
- CNN for Computer Vision
  - Convolutional Networks
    - Filter; Pooling
- Perspectives and Future Directions

Cortical Structures

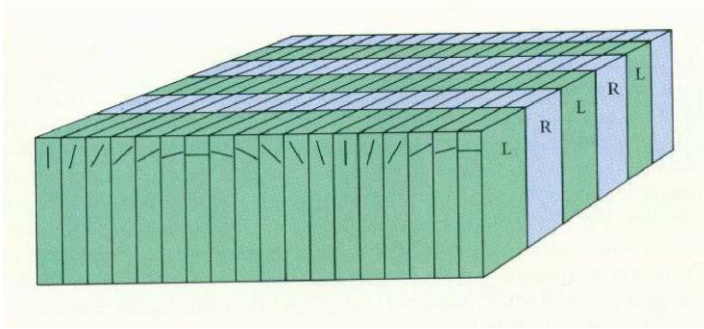
- Hierarchical Structures
- Topological Structures
  - Overcomplete
- Columnar Structures
- Connections
  - Feedforward, feedback, lateral
- Nonhomogeneity of Retina Cells
  - Computational implications



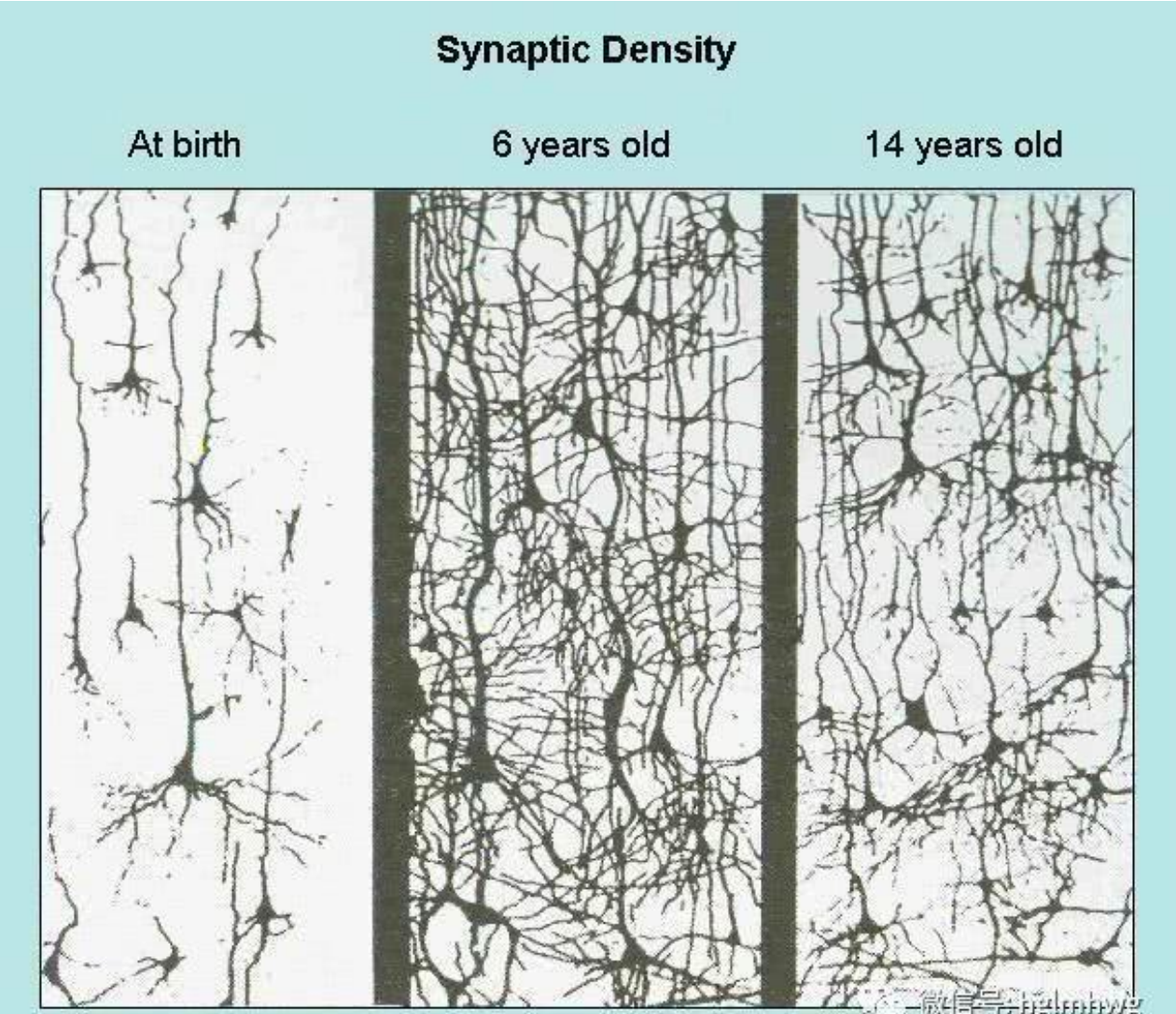




人眼横切面示意图



Background - Pruning

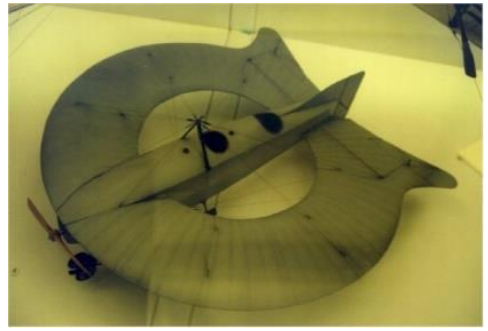


The New Challenge

- 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)

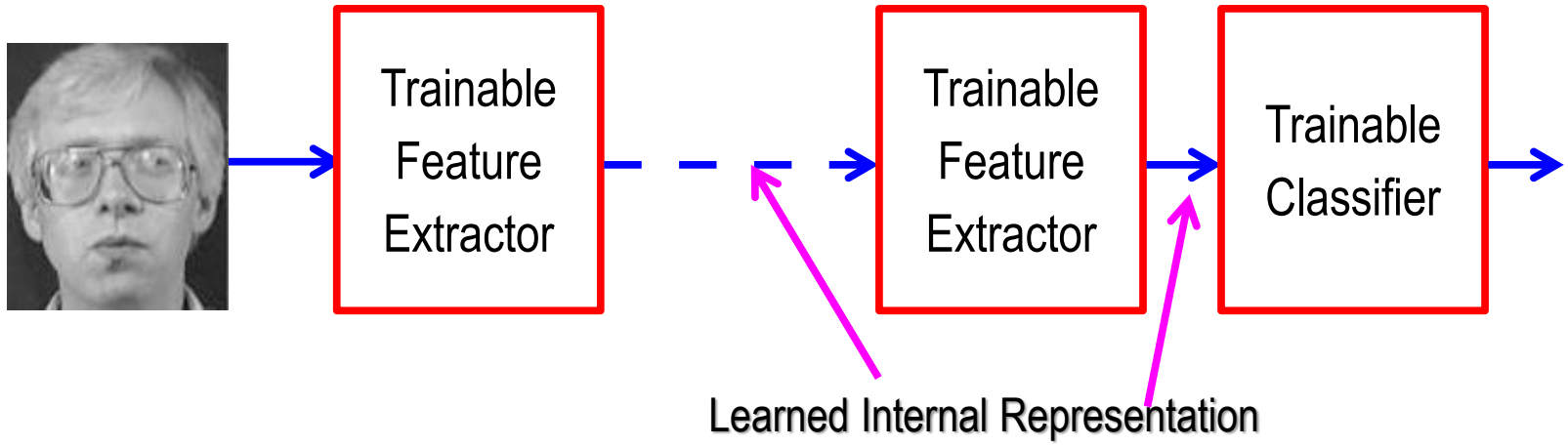






Learning Hierarchical Representations

- **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



Why deep architectures?

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

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

- **kernel machines and 2-layer neural net are "universal".**

- **Deep learning machines**

$$y = F(W^K.F(W^{K-1}.F(....F(W^0.X)...)))$$

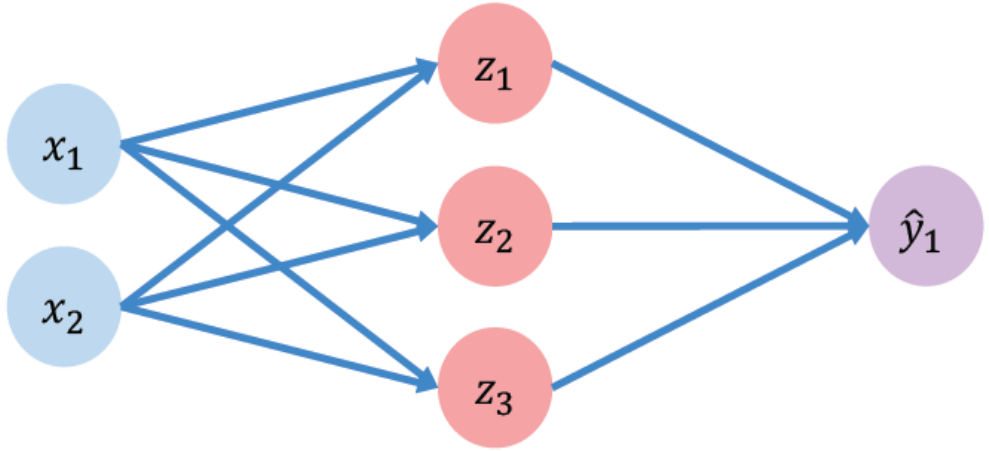
- **Deep machines are more efficient for representing certain classes of functions, particularly those involved in visual recognition**
  - Represent more complex functions with less "hardware"
  - Efficient parameterization of the class of functions that are useful for "AI" tasks.

Loss function

Mean Squared Error Loss

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

$$\boldsymbol{X} = \begin{bmatrix} 4, & 5 \\ 2, & 1 \\ 5, & 8 \\ \vdots & \vdots \end{bmatrix}$$



$$J(\boldsymbol{W}) = \frac{1}{n}\sum_{i=1}^{n}\left(y^{(i)} - f\left(x^{(i)}; \boldsymbol{W}\right)\right)^2$$

Actual Predicted

$f(x)$ y

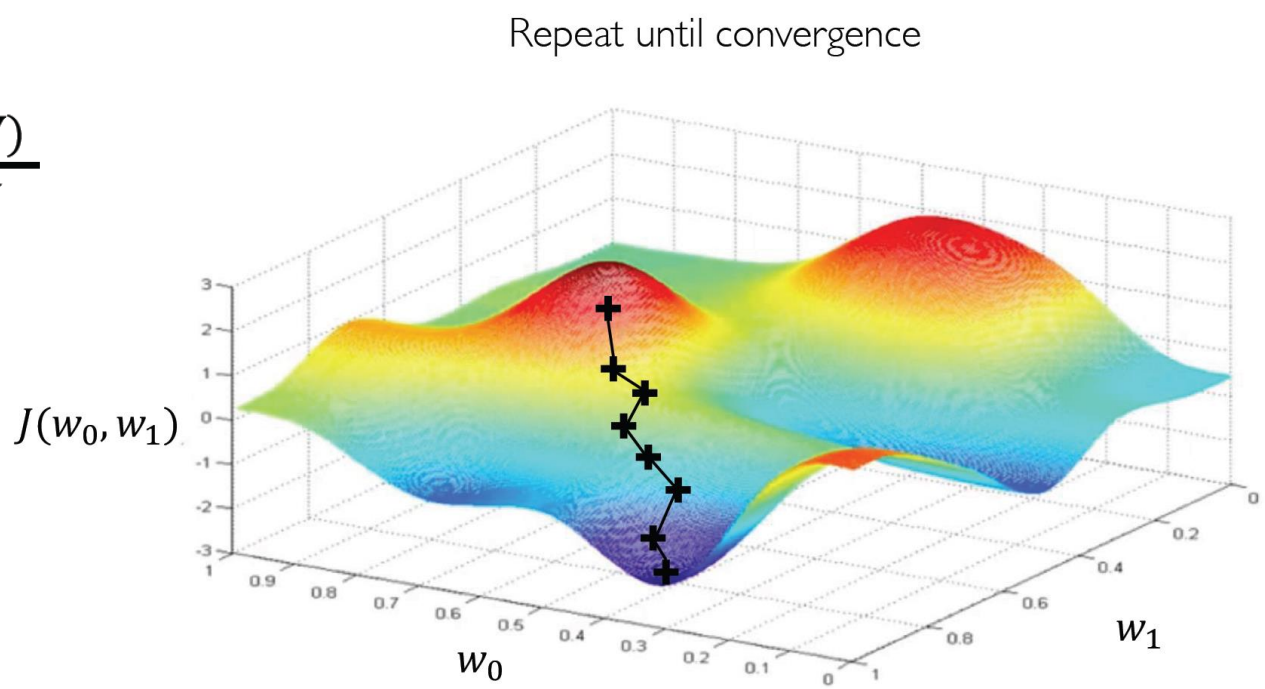
$$\begin{bmatrix} 30 \\ 80 \\ 85 \\ \vdots \end{bmatrix} \quad \begin{bmatrix} 90 \\ 20 \\ 95 \\ \vdots \end{bmatrix}$$

Final Grades (percentage)

Learning Algorithm

Algorithm

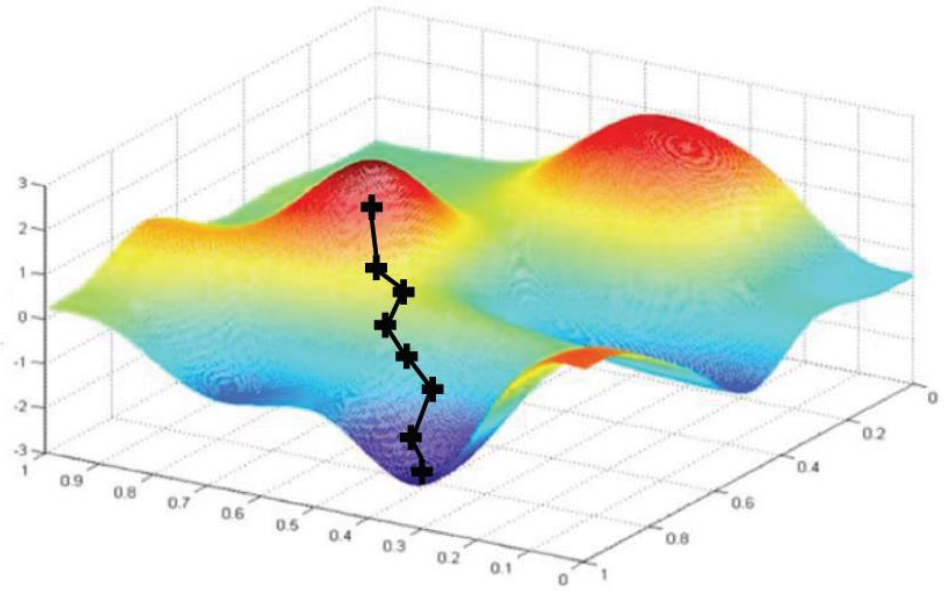
1. Initialize weights randomly $\sim \mathcal{N}(0, \sigma^2)$
2. Loop until convergence:
3. Compute gradient, $\frac{\partial J(\boldsymbol{W})}{\partial \boldsymbol{W}}$
4. Update weights, $\boldsymbol{W} \leftarrow \boldsymbol{W} - \eta \frac{\partial J(\boldsymbol{W})}{\partial \boldsymbol{W}}$
5.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



Stochastic Gradient Descent

Algorithm

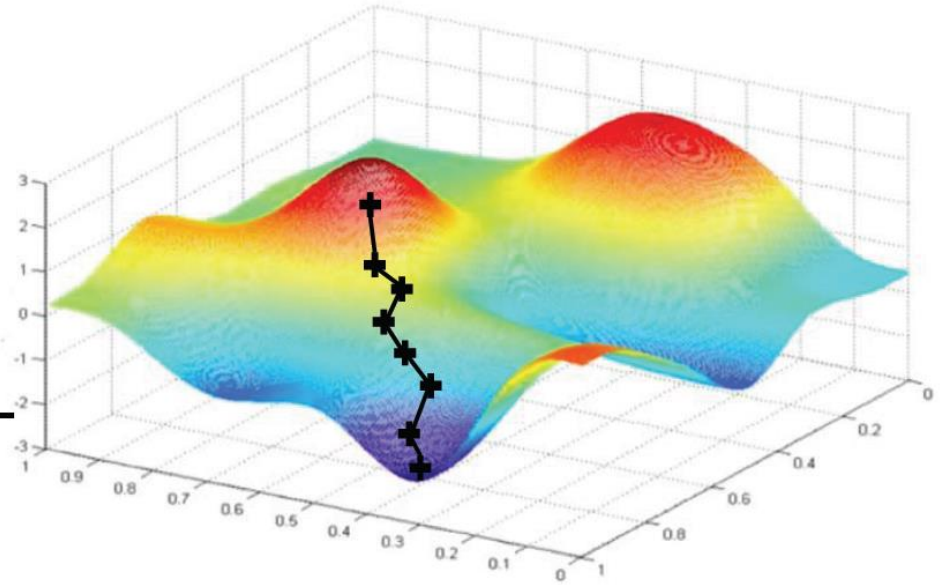
1. Initialize weights randomly $\sim \mathcal{N}(0, \sigma^2)$
2. Loop until convergence:
3. Pick single data point $\boldsymbol{i}$
4. Compute gradient, $\frac{\partial J_i(\boldsymbol{W})}{\partial \boldsymbol{W}}$
5. Update weights, $\boldsymbol{W} \leftarrow \boldsymbol{W} - \eta \frac{\partial J(\boldsymbol{W})}{\partial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



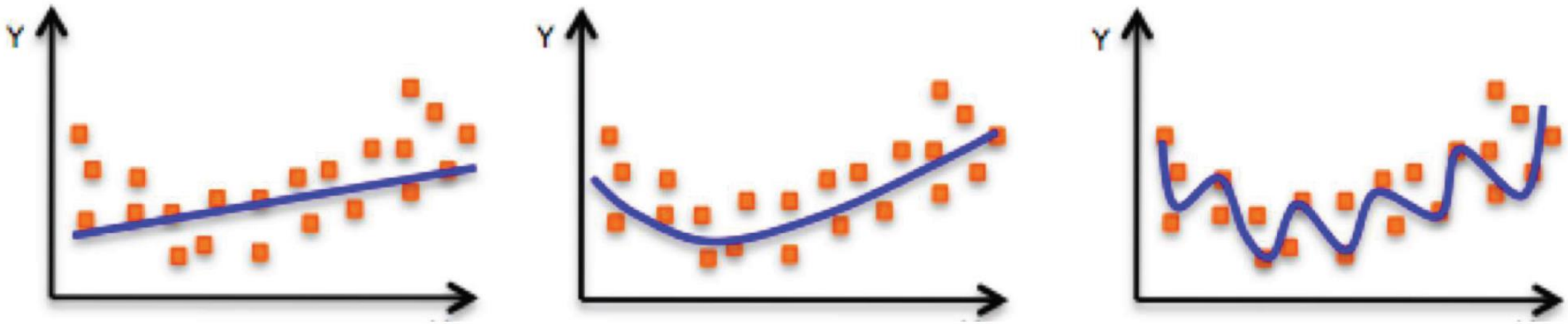
Mini-Batch

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, $\frac{\partial J(\boldsymbol{W})}{\partial \boldsymbol{W}} = \frac{1}{B} \sum_{k=1}^{B} \frac{\partial J_k(\boldsymbol{W})}{\partial \boldsymbol{W}}$
5. Update weights, $\boldsymbol{W} \leftarrow \boldsymbol{W} - \eta \frac{\partial J(\boldsymbol{W})}{\partial \boldsymbol{W}}$
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*

Outline

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

Dropout Regularization

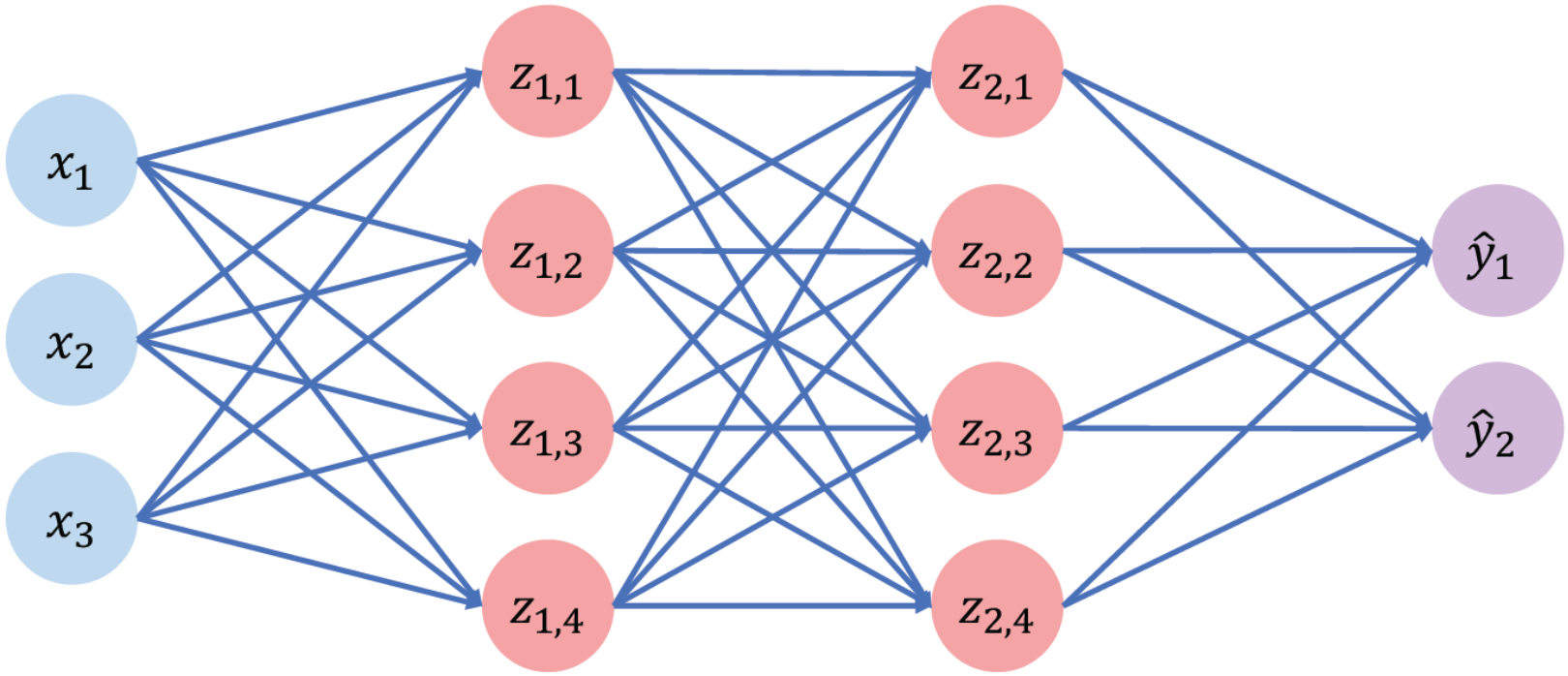
- **Dropout regularization** is used for reducing overfitting and improving the generalization of deep neural networks.
- Large weights in a neural network are a sign of a more complex network that has overfit the training data.
- **Probabilistically dropping out** nodes in the network is a simple and effective regularization method.
- **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

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

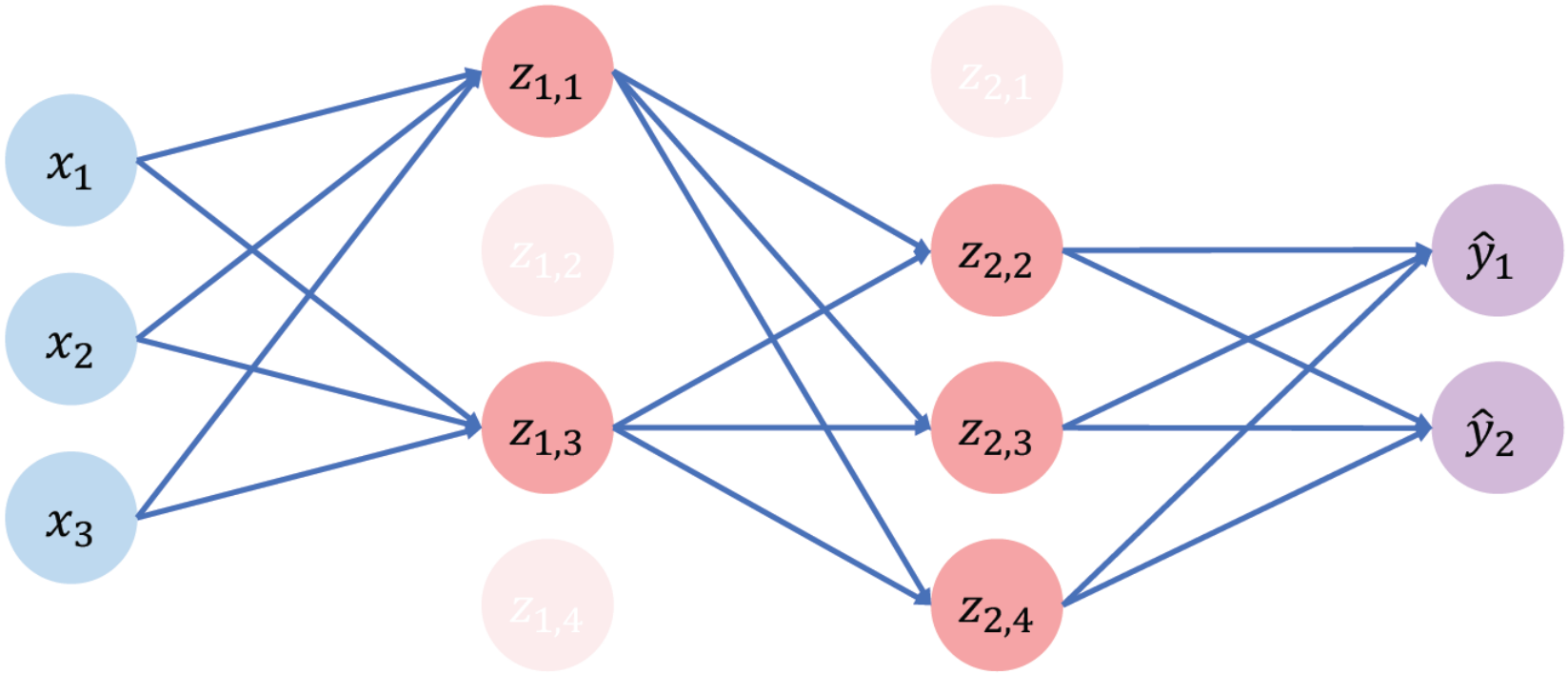
- During training, randomly set some activations to 0



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, encouraging 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.

Outline

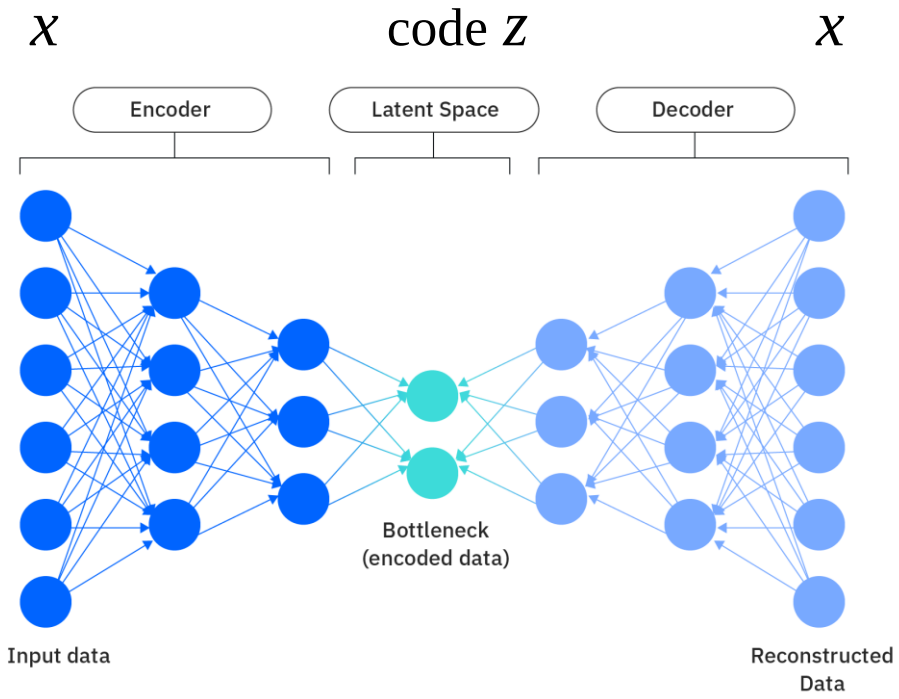
- Background
- Deep Neural Networks
  - Regularization - Dropout
- Disentangled Representation in DNN
- CNN for Computer Vision
  - Convolutional Networks
    - Filter; Pooling
- Perspectives and Future Directions

AutoEncoder

- Autoencoder (AE)
- Input x; Encoder: z

Encoder $f : X \to Z; \quad z = f(x, \theta);$

Decoder $g : Z \to X; \quad \hat{x} = g(z, \varphi);$



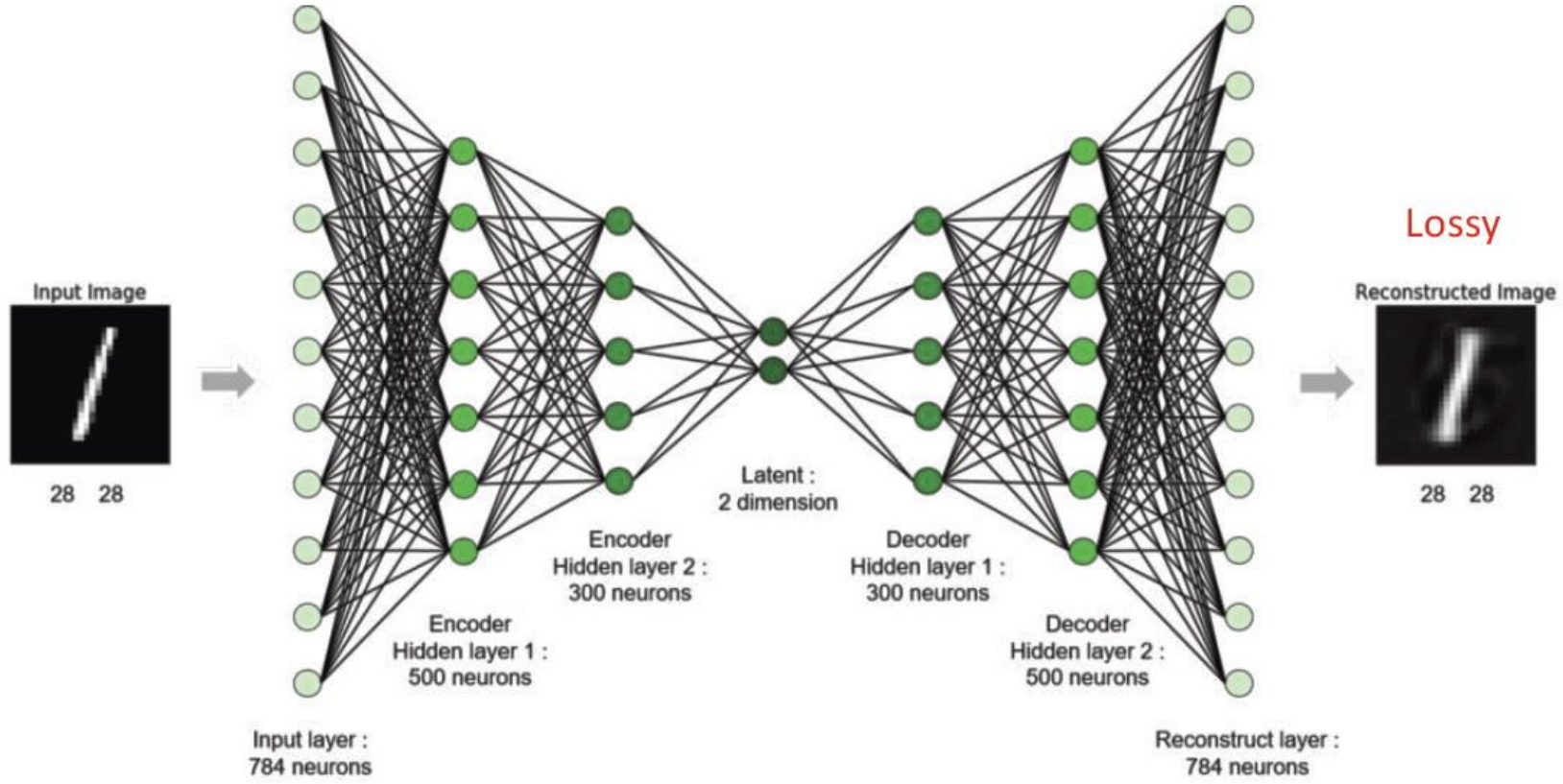
Risk function(Cost Function)

$$f, g = \arg\min_{f,g} \left\| x - g[f(x)] \right\|^2$$

AutoEncoder is a natrual extension of PCA to Nonlinear Cases

AutoEncoder for Vector Embedding

- **Example:**
  - Compress MNIST (28x28x1) to the latent code with only 2 variables

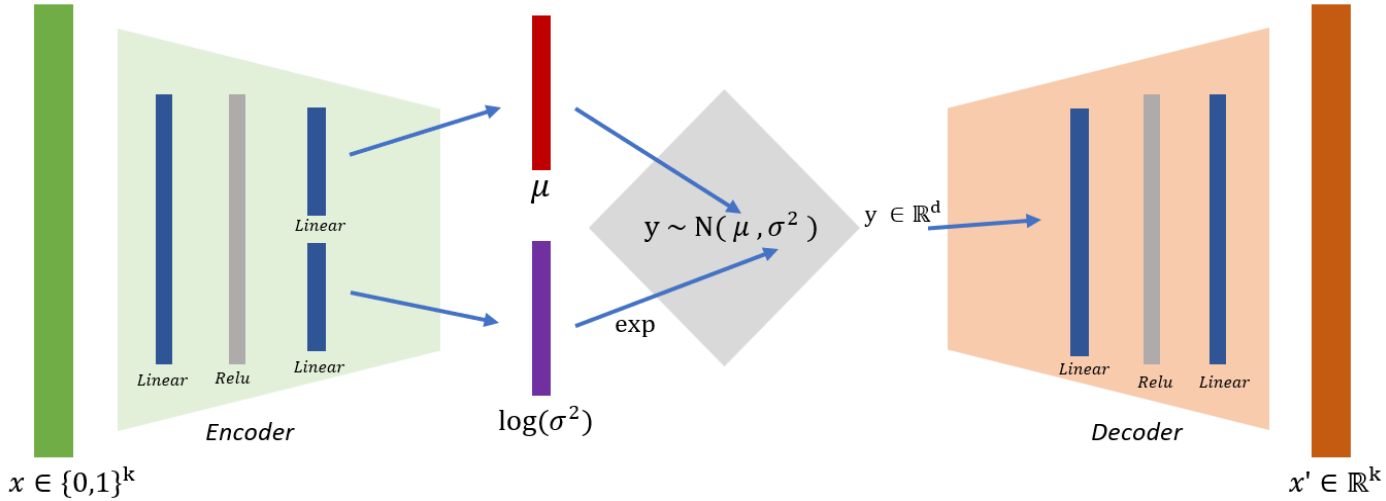


Variational AutoEncoder (VAE)

- Main Components in VAE
  - Probabilistic formulation of Encoder and Decoder
  - Learning Criterion for Model Learning
    - Reconstruction loss
    - Kullback-Leibler (KL) divergence
  - Evidence lower bound (ELBO)
  - The reparameterization trick

Variational Autoencoder (VAE)

- VAE =AutoEncoders + Variational inference
  - to learn a probability distribution over the input data, which can then be used to generate new data similar to the training data.



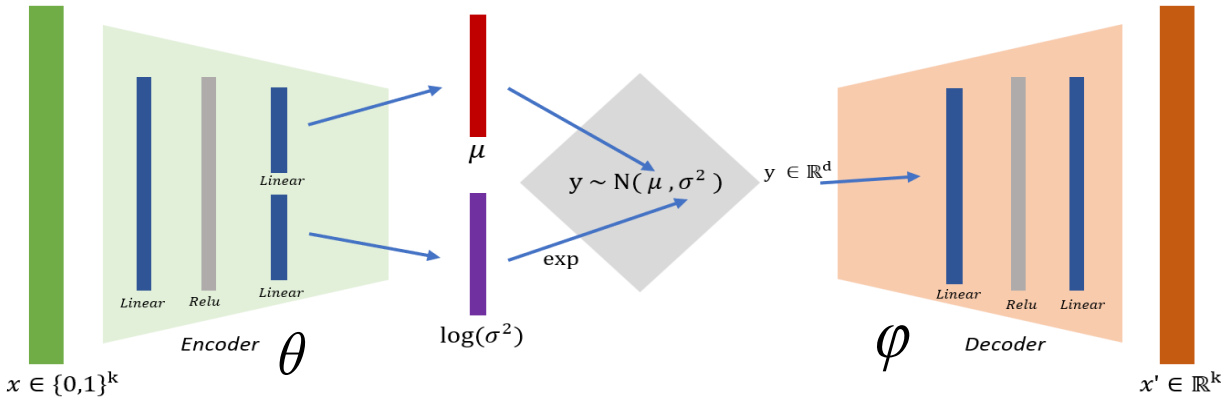
VAES are *probabilistic* models. VAEs encode latent variables of training data not as a fixed discrete value z, but as a continuous range of possibilities expressed as a probability distribution $p(z)$.

Mathematical Formulation of the Encoder

- $\mathbf{X}$: the input data point (e.g., an image, a sequence of text, etc.).
- The encoder network $p(z \mid X, \theta)$ maps the input X to a latent variable z.
  - Here, θ represents the parameters of the encoder network.
- The encoder network outputs the mean $\mu_\theta(x)$ and the covariance matrix $\sigma_\theta(x)$ or more commonly, the log - variance $\log \sigma_\theta^2(x)$ of a Gaussian distribution.
- The latent variable z is then sampled from this distribution:

$$z = \mu_\theta(x) + \varepsilon \, \sigma_\theta(x)$$

$$\varepsilon \sim N(0, I)$$



Mathematical Formulation of the Decoder

- The decoder network $q(x|z,\varphi)$ maps the latent variable z back to the original data space.
  - Here, φ represents the parameters of the decoder network.
- The decoder is usually modeled as a conditional probability distribution. For example, if the data is continuous (like images), it might be a Gaussian distribution

$$q(x|z,\varphi) = N(z \mid \mu_\varphi, \Sigma_\varphi)$$

where $\mu_\varphi, \Sigma_\varphi$ are the mean and covariance of the distribution predicted by the decoder. If the data is discrete (like text), it might be a categorical distribution.

Loss Function

- Reconstruction Error: This measures how well the decoder can reconstruct the original input from the latent variable.

$$L_{rec} = \sum_{i=1}^{N} \left(x_i - G_{\varphi}(F_{\theta}(x_i)) \right)^2$$

- Regularization: KL - Divergence :

$$L_{reg} = KL(p(z \mid x, \theta) \| p(z)) = E_{p(z|x,\theta)} \left[\frac{p(z \mid x, \theta)}{p(z)} \right]$$

  - KL term encourages the $p(z \mid x, \theta)$ close to a prior distribution $p(z)$ (usually a standard normal distribution *N(0, I)* . The KL - divergence is given by:

$$L_{reg} = KL\left[N(\mu_1, \Sigma_1) \| N(\mu_2, \Sigma_2) \right] = \frac{1}{2} \left[\log \frac{|\Sigma_1|}{|\Sigma_2|} - n + tr(\Sigma_2^{-1}\Sigma_1) + (\mu_2 - \mu_1)^T \Sigma_2^{-1} (\mu_2 - \mu_1) \right]$$

- **The total loss of the VAE:**

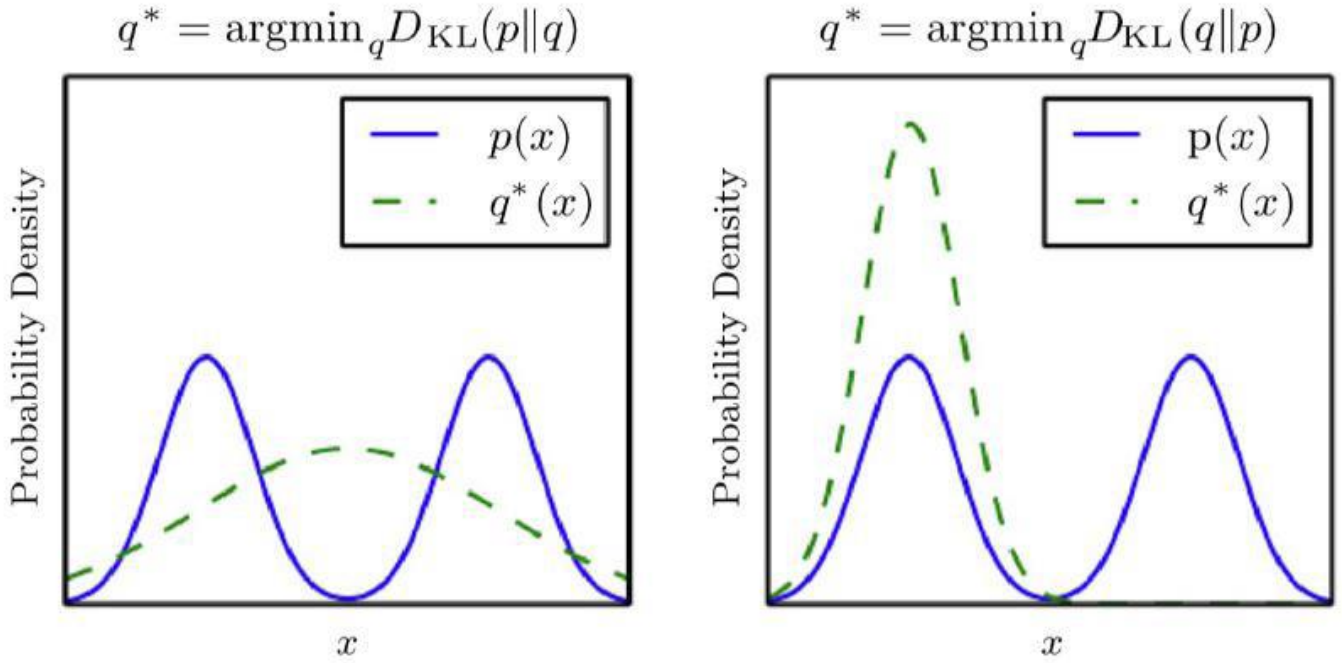
$$L_{total} = L_{rec} + L_{reg} = \sum_{i=1}^{N} \left\| x_i - G_{\varphi}(F_{\theta}(x_i)) \right\|^2 + KL(p(z \mid x, \theta) \| p(z))$$

KL(Q||P) or *KL(P||Q)*

- Some applications require an approximation that usually places high probability anywhere that the true distribution places high probability: left one

- VAE requires an approximation

that rarely places high probability anywhere that the true distribution places low probability: right one



$$D_{\text{KL}}(P\|Q) = \mathbb{E}_{\text{x}\sim P}\left[\log \frac{P(x)}{Q(x)}\right] = \mathbb{E}_{\text{x}\sim P}\left[\log P(x) - \log Q(x)\right].$$

Variational Inference

- Maximizing the variational lower bound

$$\log p(x) \geq F(\theta, \varphi) = E_{q(z)}\left[\log p(x \mid z)\right] - D_{KL}(q \mid p)$$

- The relation with cost function for VAE

$$\log p(x \mid z, \varphi) = -\frac{1}{2\sigma^2}\left\|x - G_{\varphi}(z)\right\|^2 + const.$$

- Miximizing the variational lower bound is equivalent to minimizing the total Loss function

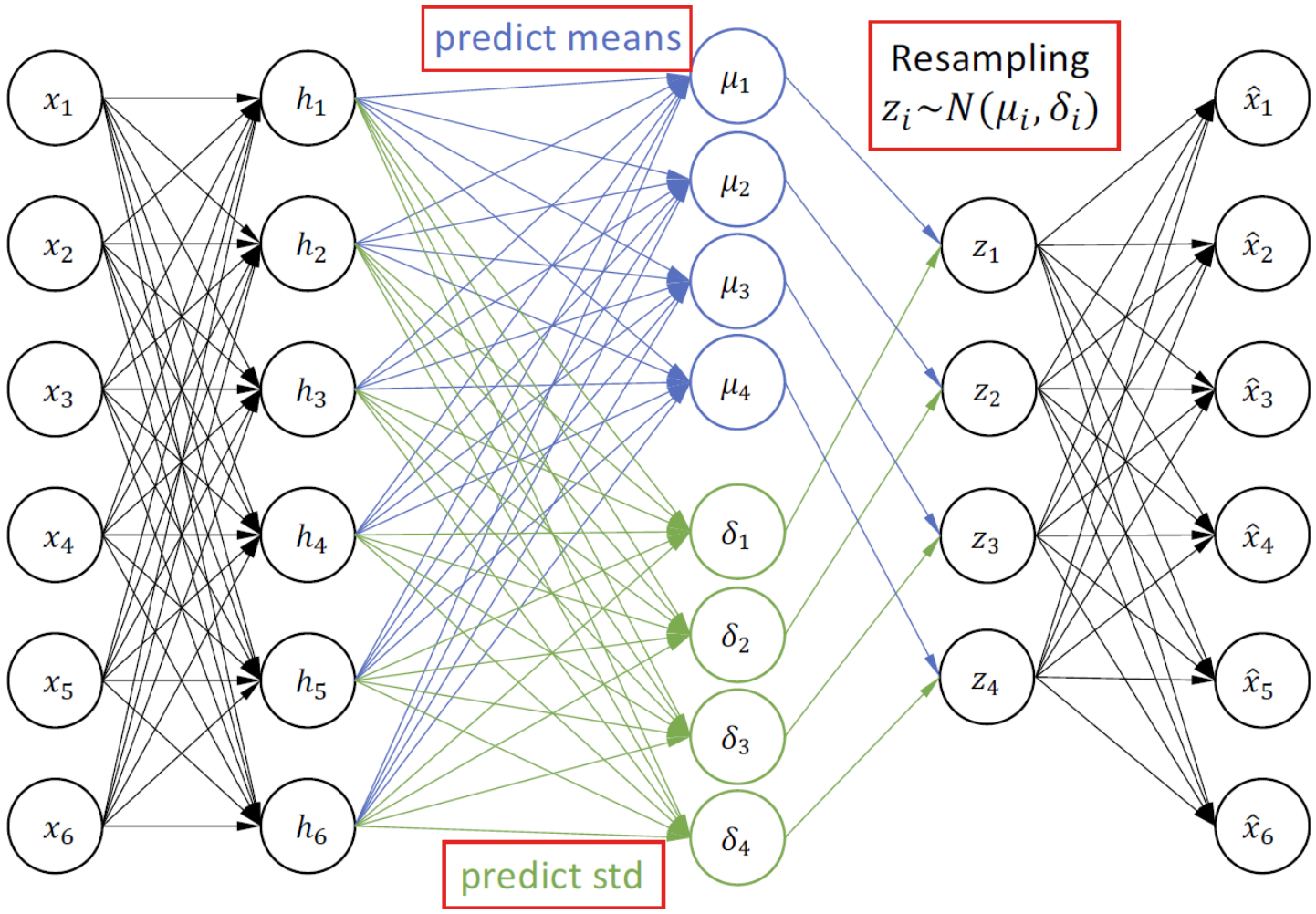
$$\max_{\theta,\varphi} F(\theta, \varphi) \sim \min_{\theta,\varphi} L_{total}$$

Reparameterisation Trick

1. Encode the input
2. Predict means
3. Predict standard derivations
4. Use the predicted means and stan
5. Reconstruct the input

$$z = \mu + \varepsilon \delta$$

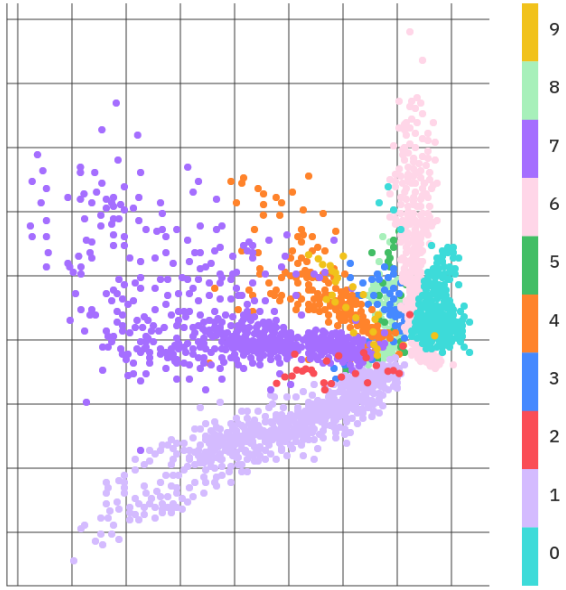
$$\varepsilon \sim N(0, I)$$



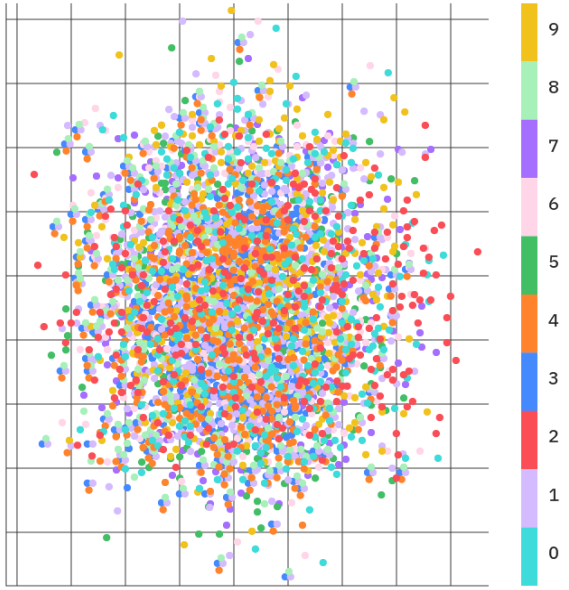
Experin

- Examples latent spa

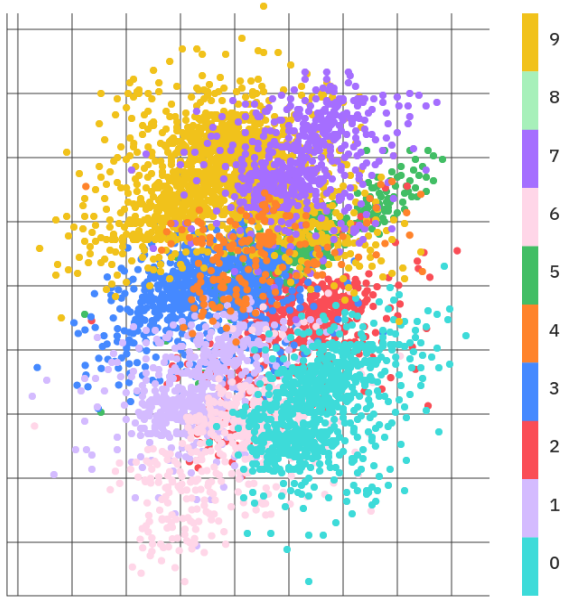
Only reconstruction loss



Only KL divergence



Reconstruction loss and KL divergence



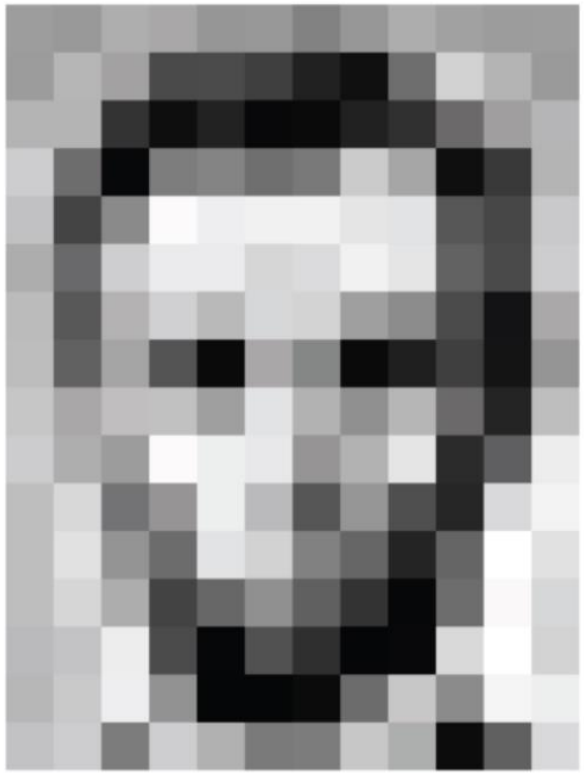
Advantages of VAE over AE

- Generative Capability
  - VAE: VAEs have a generative property. New data points can be generated by sampling from this latent distribution and passing the samples through the decoder.
- Regularization through KL - Divergence
  - VAE: The KL - divergence term in the VAE loss function acts as a regularization mechanism. It forces the learned posterior distribution of the latent variables to be close to a prior distribution (usually a standard normal distribution).
- Uncertainty Estimation
  - VAE: VAEs can provide an estimate of the uncertainty associated with the generated or reconstructed data.
  - For example, in a task such as image in - painting or data imputation, the VAE can give an indication of how confident it is about the reconstructed or generated values.

Outline

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

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	159	181
206	109	5	124	131	111	120	204	166	15	56	180
194	68	137	251	237	239	239	228	227	87	71	201
172	105	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	106	36	190
205	174	155	252	236	231	149	178	228	43	95	234
190	216	116	149	236	187	86	150	79	38	218	241
190	224	147	108	227	210	127	102	36	101	255	224
190	214	173	66	103	143	96	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	87	71	201
172	105	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	106	36	190
205	174	155	252	236	231	149	178	228	43	95	234
190	216	116	149	236	187	86	150	79	38	218	241
190	224	147	108	227	210	127	102	36	101	255	224
190	214	173	66	103	143	96	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

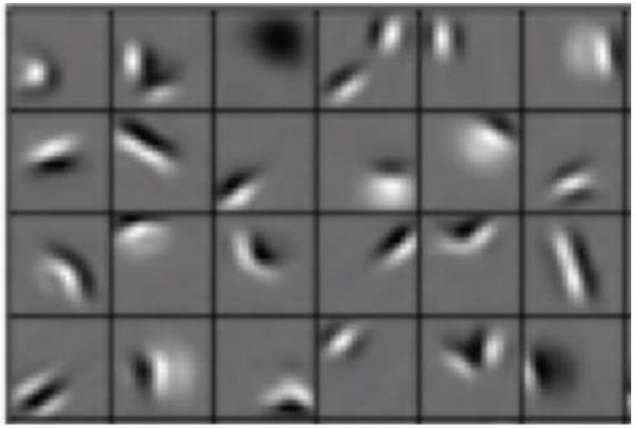
Manual Feature Extraction Domain

- Problems: ???

Feature Representation

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

Low level features



Edges, dark spots

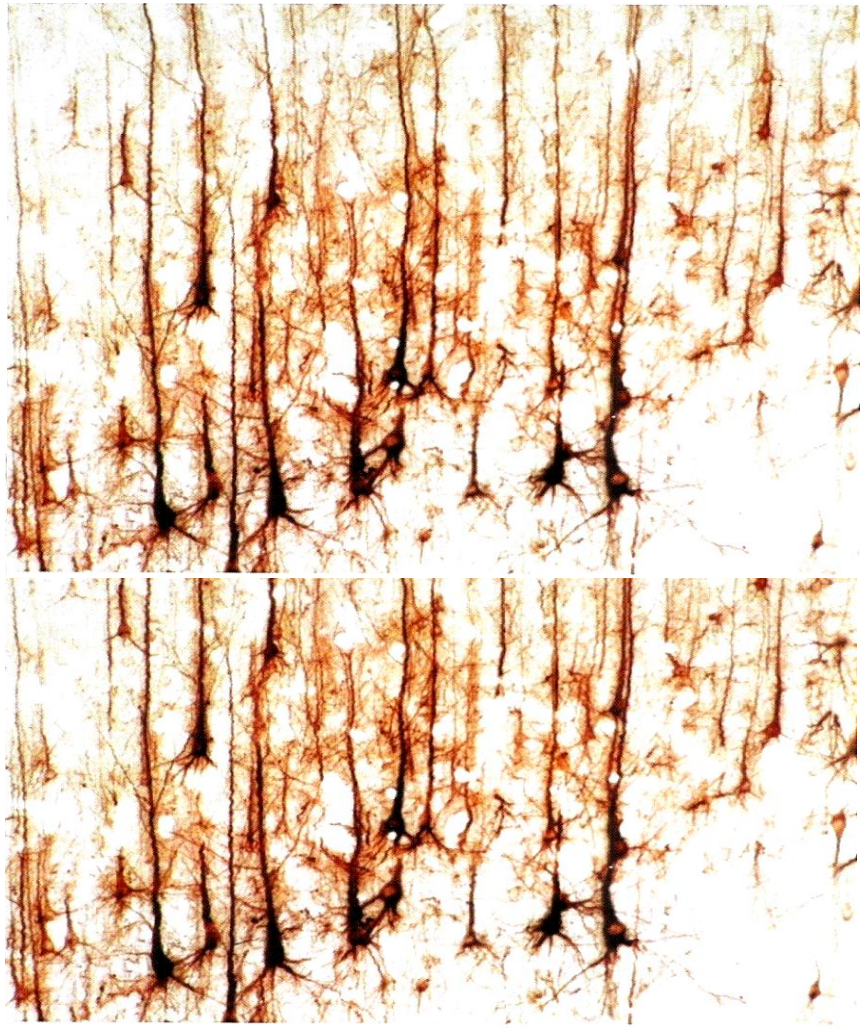
Mid level features

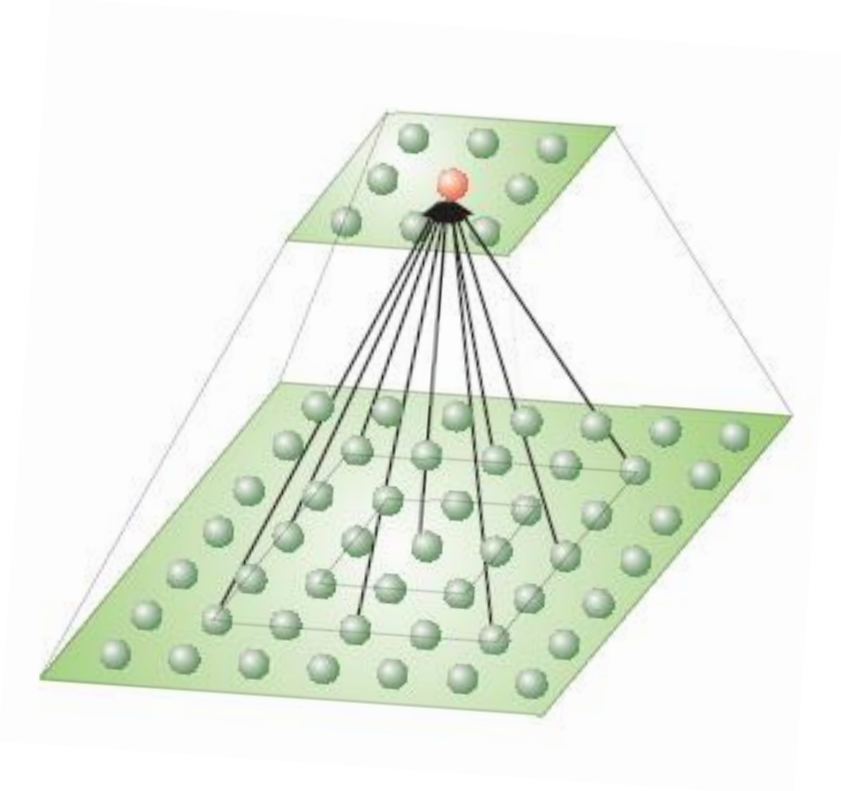
Eyes, ears, nose

High level features

Facial structure

Cortical Networks– Spatial Convolution





Feature Matching

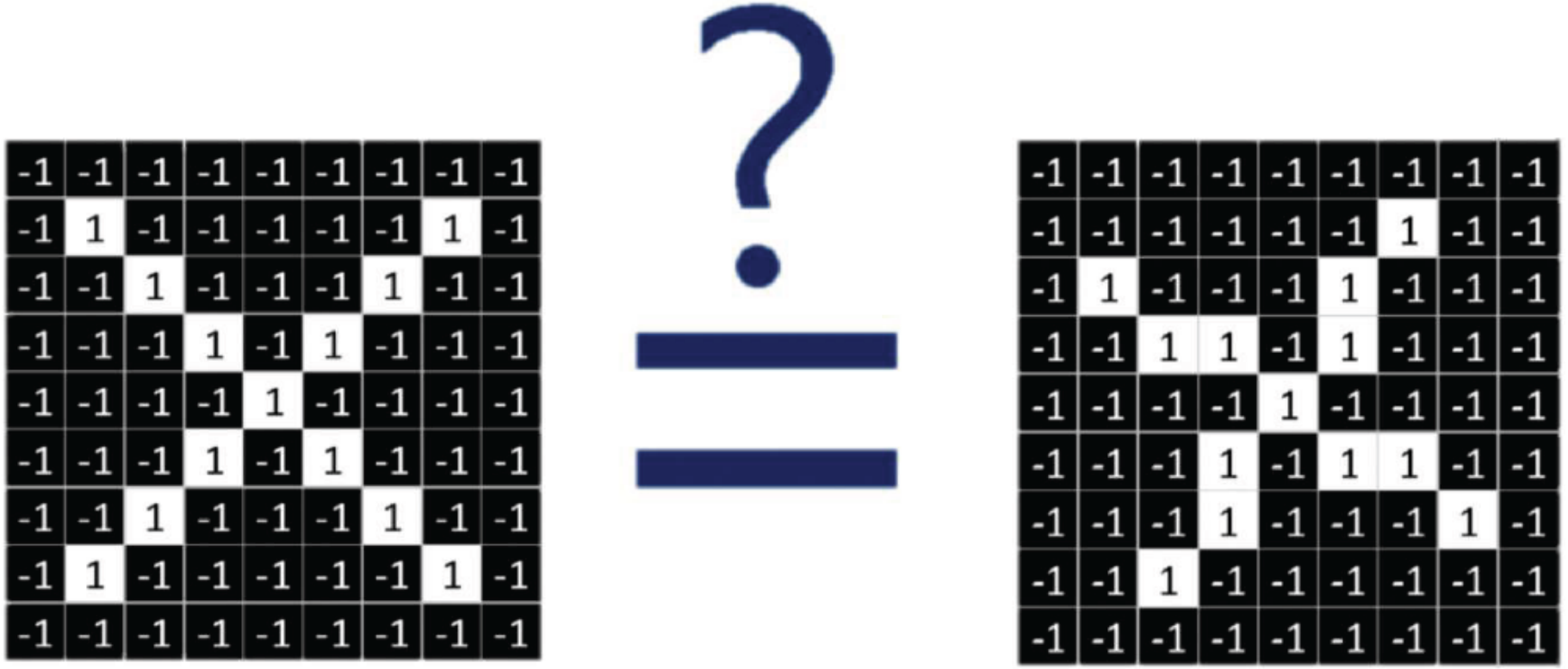
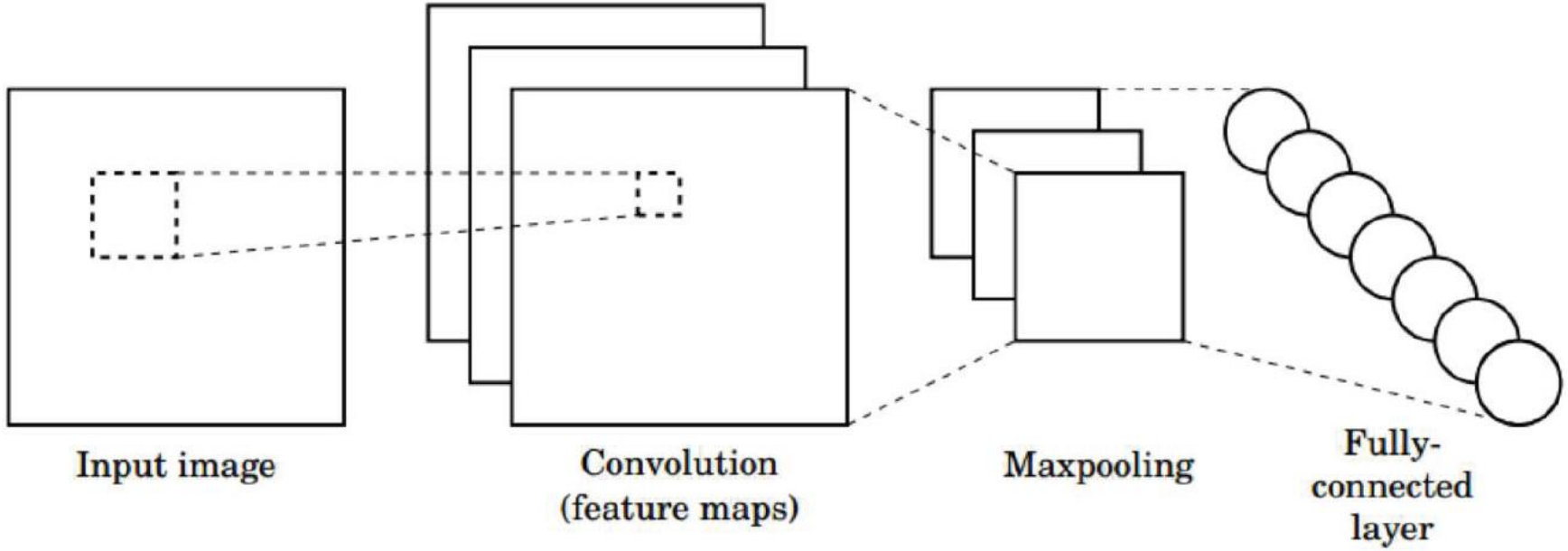


Image is represented as matrix of pixel values… and computers are literal!
We want to be able to classify an X as an X even if it's shifted, shrunk, rotated, deformed.

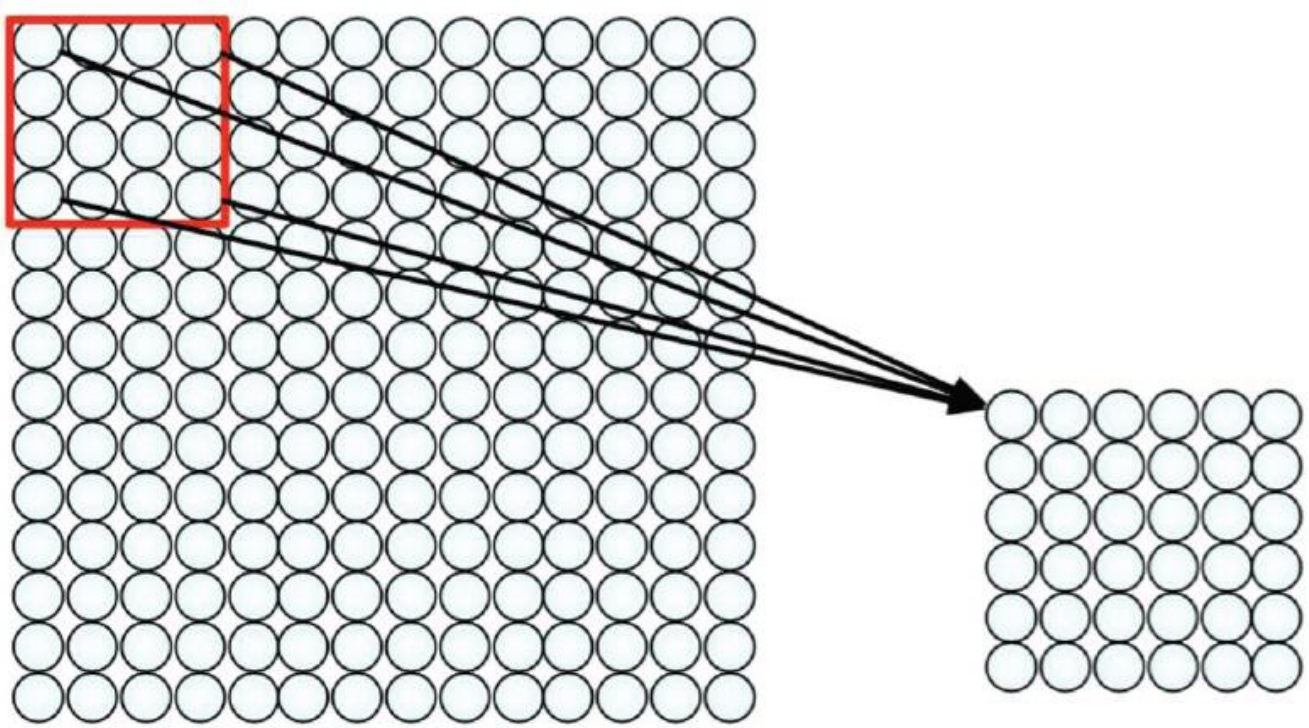
CNN for Classification



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

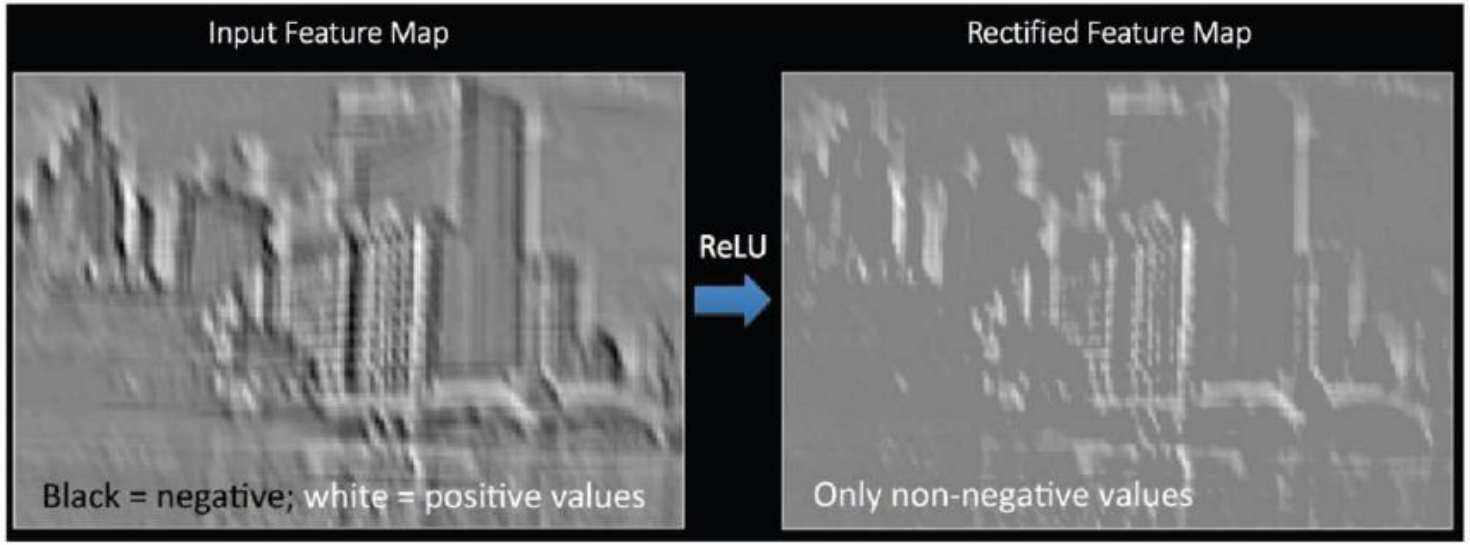
For a neuron in hidden layer:

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

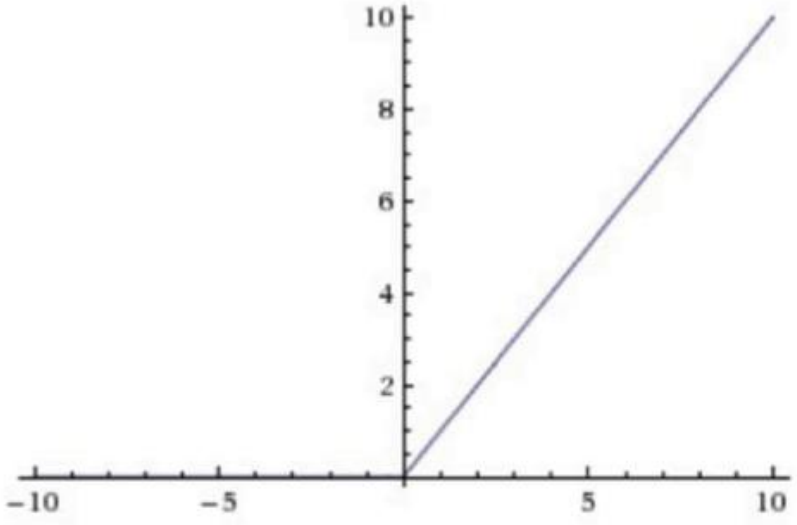
1) applying a window of weights
2) computing linear combinations
3) activating with non-linear function

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!**

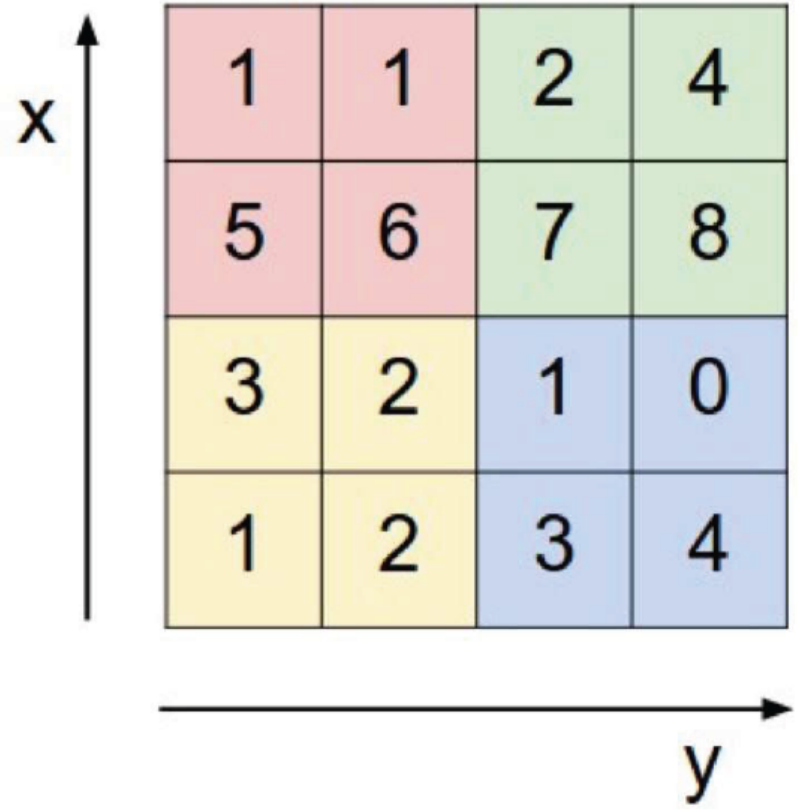


Rectified Linear Unit (ReLU)



$$g(z) = \max(0, z)$$

Pooling



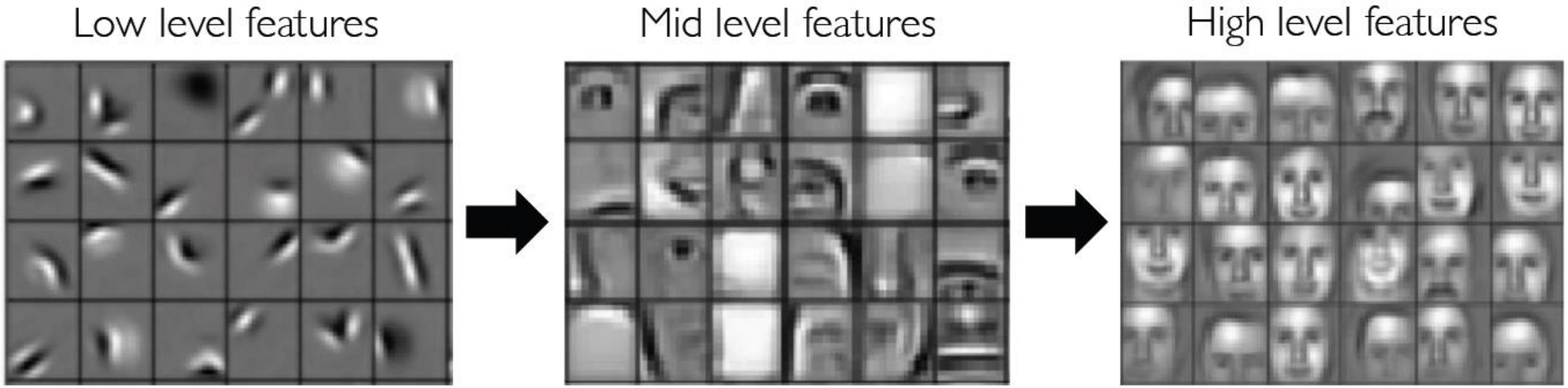
max pool with 2x2 filters
and stride 2

6	8
3	4

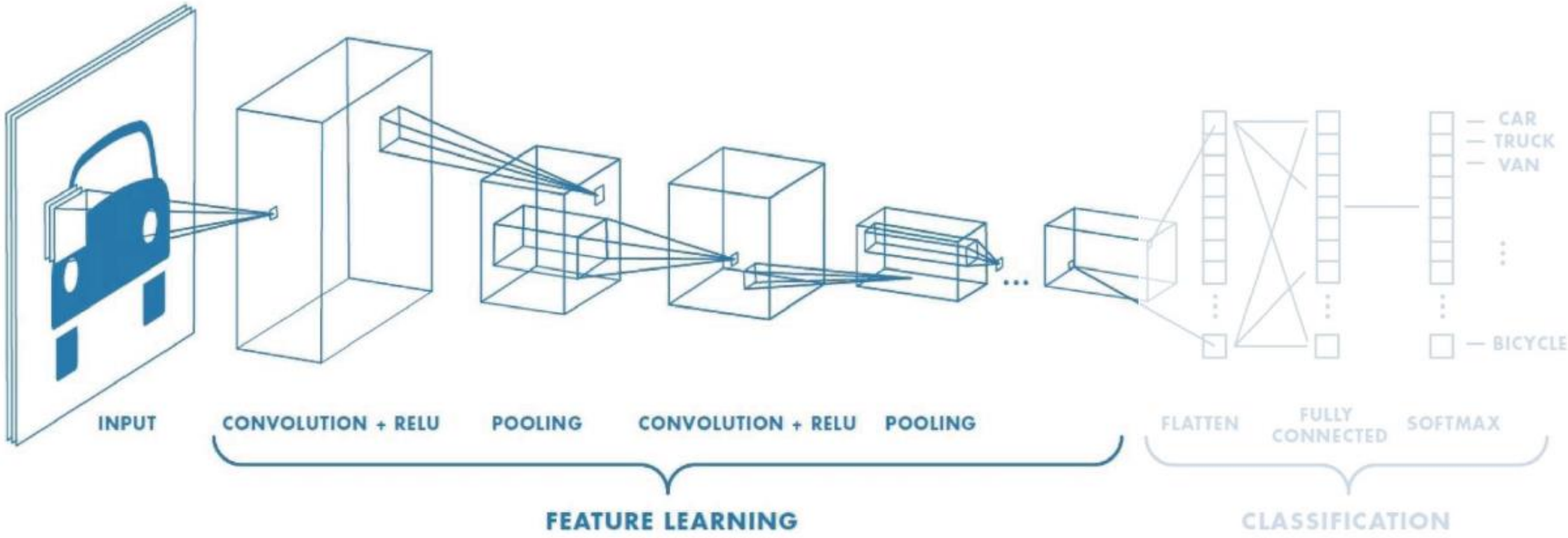
1) Reduced dimensionality
2) Spatial invariance

How else can we downsample and preserve spatial invariance?

Representation Learning in Deep CNNs

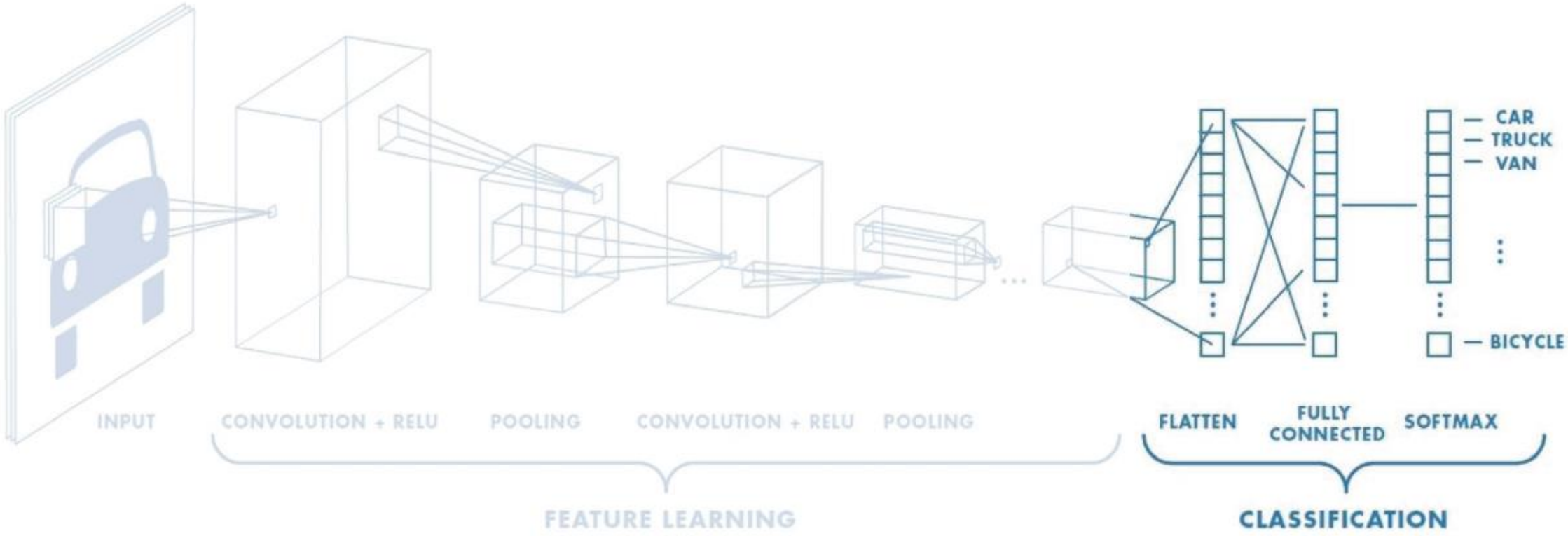


CNNs for Classification: Feature Learning



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**

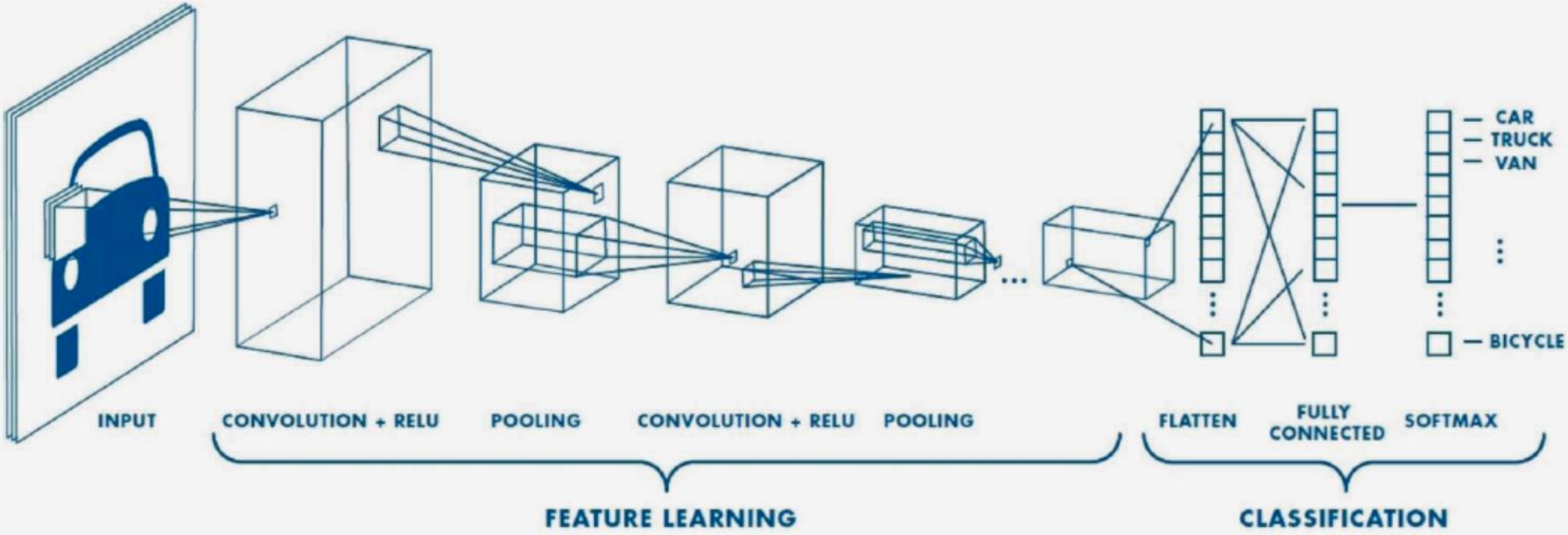
CNNs for Classification: Class Probabilities



- 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

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

ImageNet Dataset

Dataset of over 14 million images across 21,841 categories

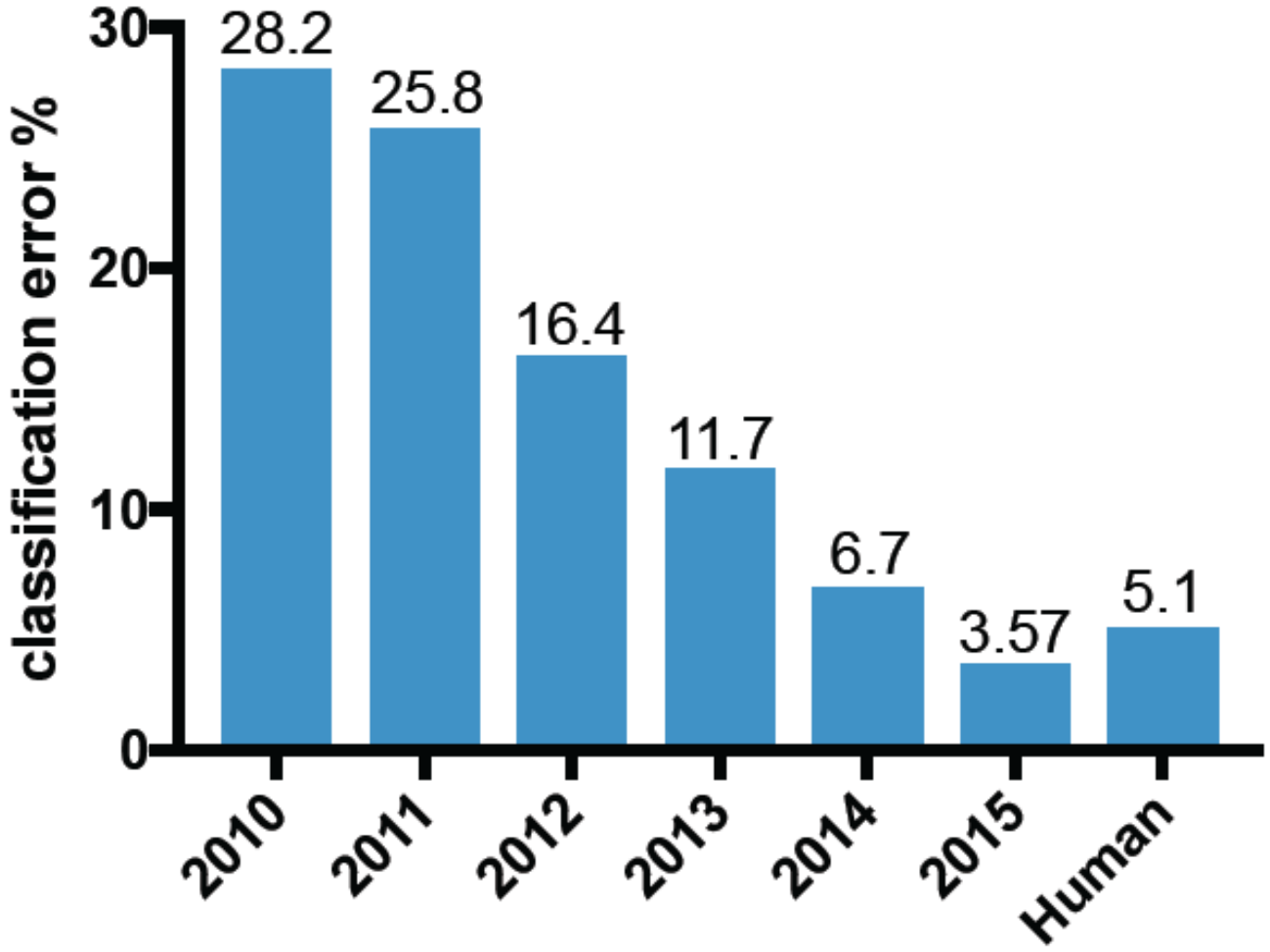
"Elongated crescent-shaped yellow fruit with soft sweet flesh"

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 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, 5million parameters

2015: ResNet
- 152 layers

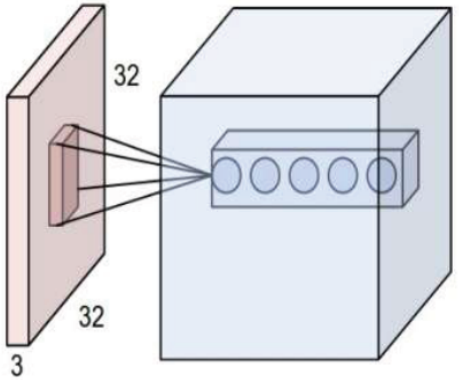
Deep Learning for Computer Vision: Summary

Foundations

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

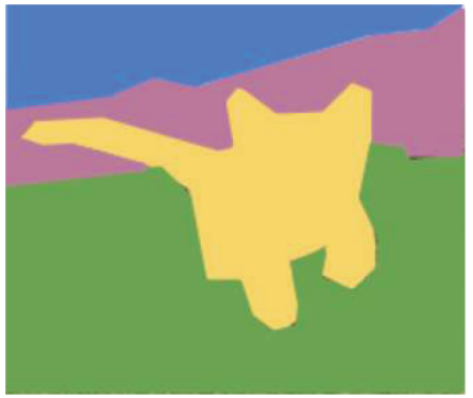
CNNs

- CNN architecture
- Application to classification
- ImageNet



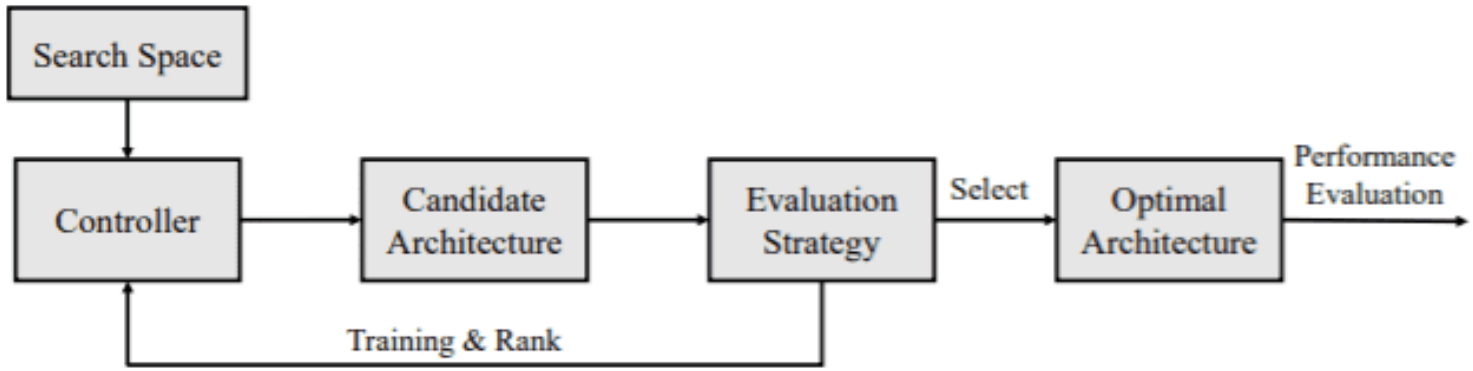
Applications

- Segmentation, object detection, image captioning
- Visualization



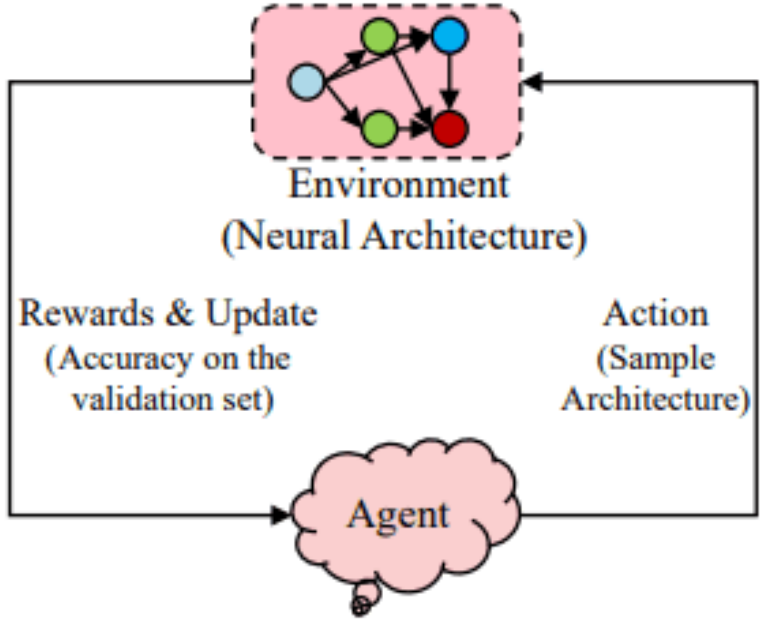
NAS algorithm (From Model selection to Search)

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

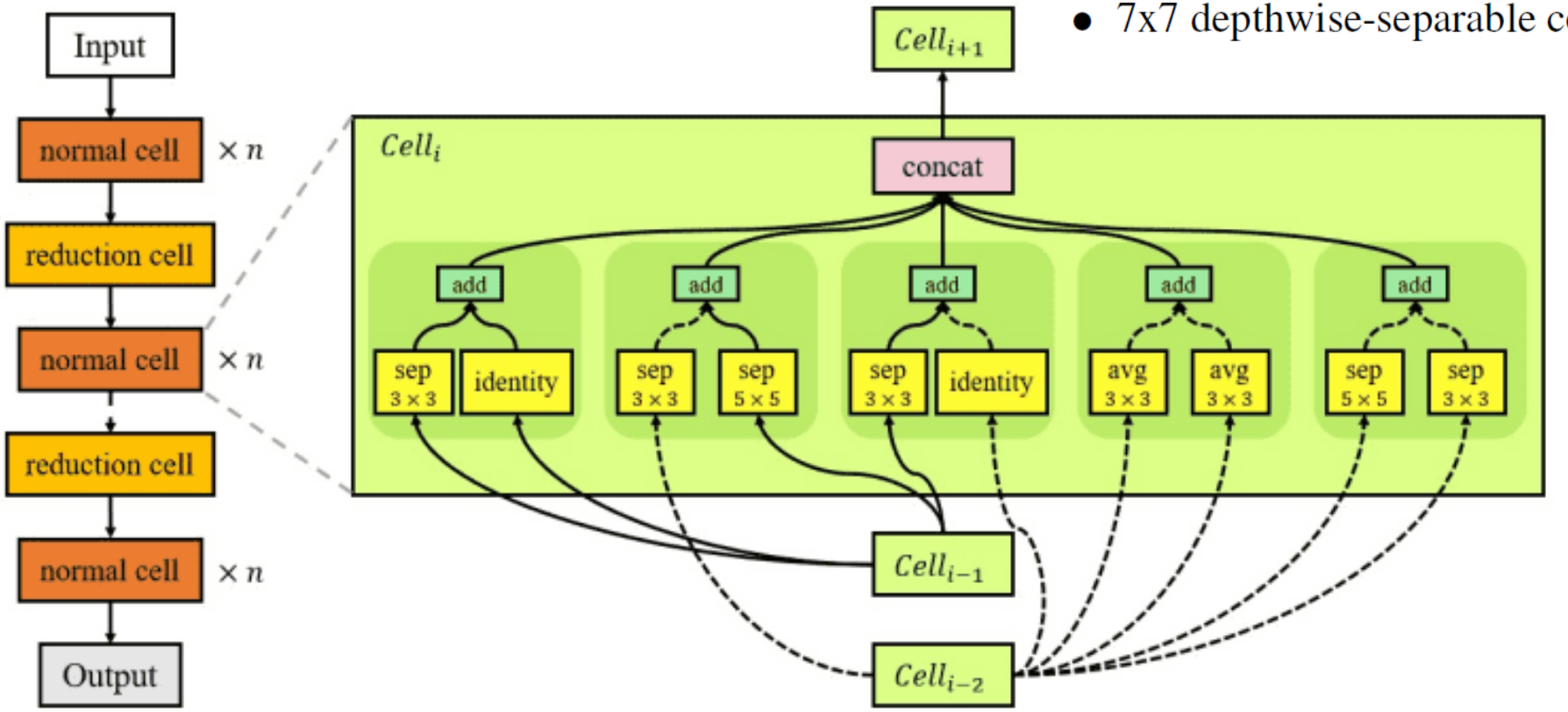
- **Search strategy:** the methodology used to search for the optimal architecture in the search space.
  1. Random search
  2. Reinforcement learning
  3. Evolutionary algorithms
  4. Sequential model-based optimization
  5. Gradient optimization
- Reinforcement learning



Modular search space

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

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



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?
  - Dropout; Disentangled Regularization
- How to implement the model selection for deep CNN model?

The End of Talk