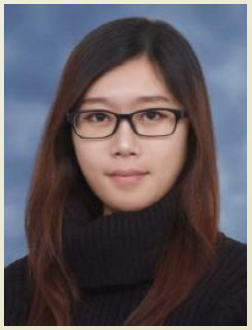
Convolution Neural Network

Il-Chul Moon, Wonsung Lee, Sungrae Park, Su-Jin Shin, Kyungwoo Song, Weonyoung Joo, JunKeon Park, YoonYeong Kim, Joonho Jang

Dept. of Industrial and Systems Engineering
KAIST
icmoon@kaist.ac.kr


Wonsung Lee


Sungrae Park


Su-Jin Shin


Kyungwoo Song


Weonyoung Joo


JunKeon Park

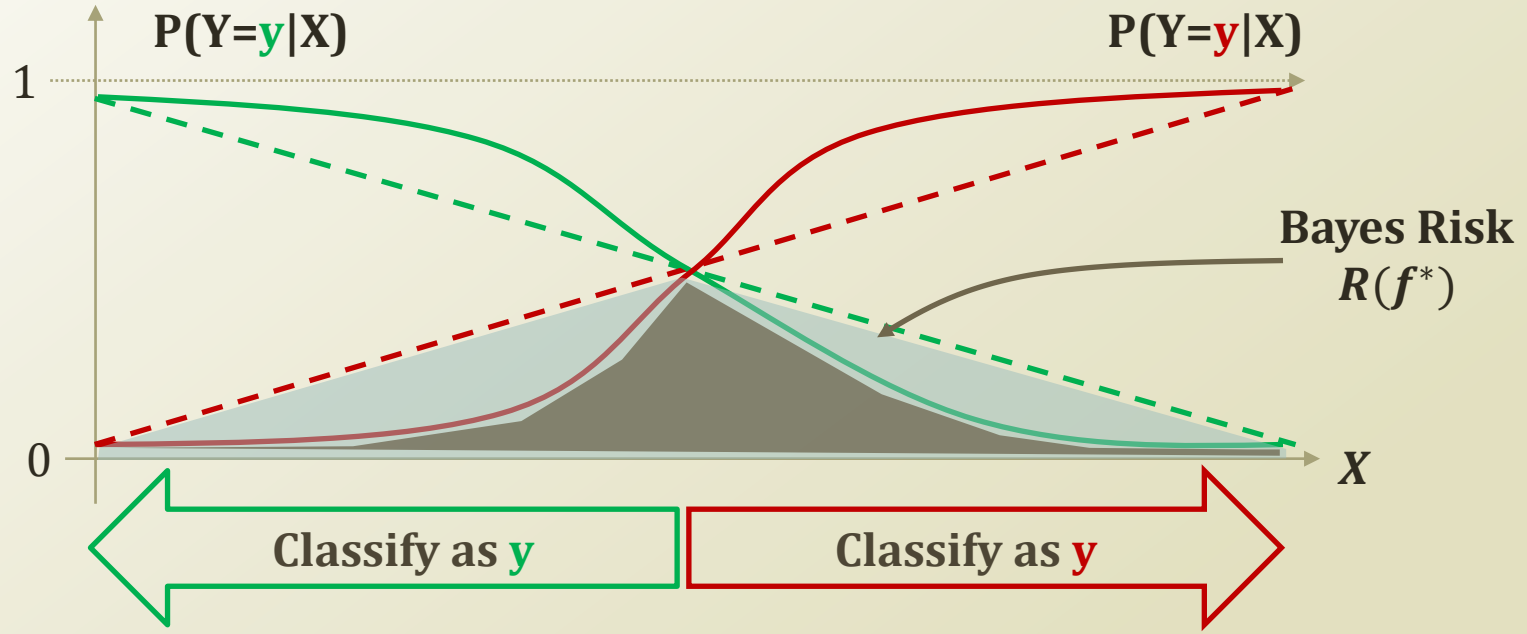

YoonYeong Kim


Joonho Jang

ACKNOWLEDGEMENTS

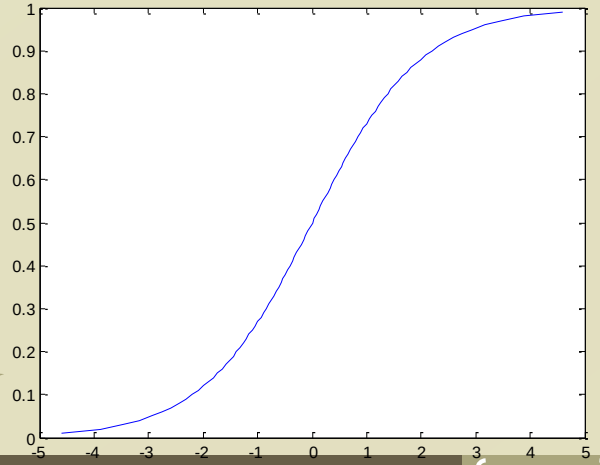
SIMPLE FEED-FORWARD NETWORKS

Optimal Classification and Bayes Risk

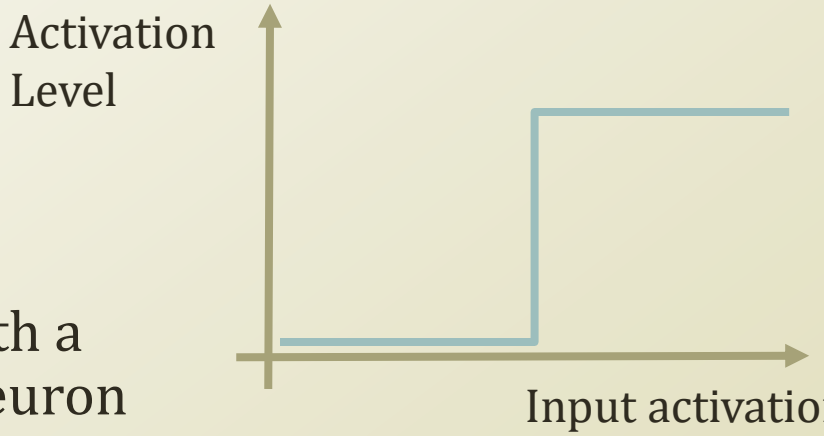


- Linear function vs. Non-linear function of P(Y|X)
  - Which is better?
- Problems of linear function
  - Range
  - Risk optimization
- Which function to use?
  - Need S-curve!

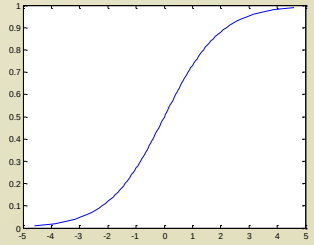
S-curve
a.k.a. Sigmoid
function



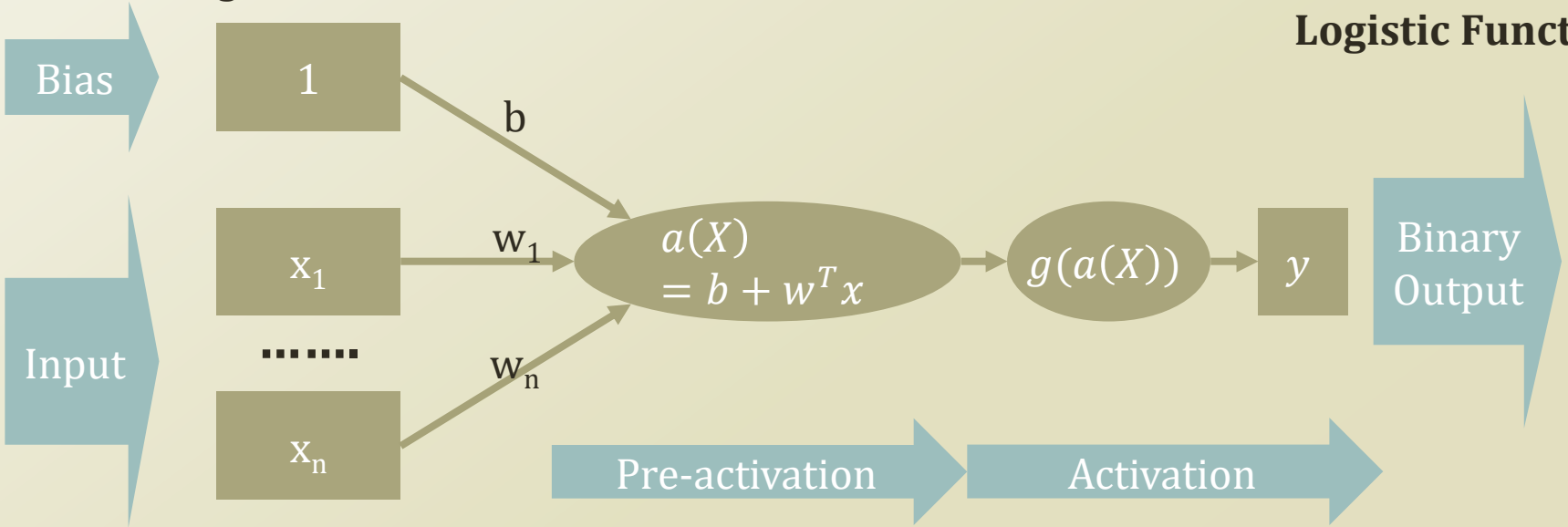
Perceptron



- Perceptron is a binary classifier with a similar structure resembling the neuron
- A part of mechanism of perceptron is already covered
  - Do you recall logistic regression?
    - $\mu(x) = \frac{1}{1+e^{-\dot{\theta}^T x}} = g\big(a(x)\big), a(x) = \dot{\theta}^T x, g(y) = \frac{1}{1+e^{-y}}$
    - Logistic function becomes the smooth activation function

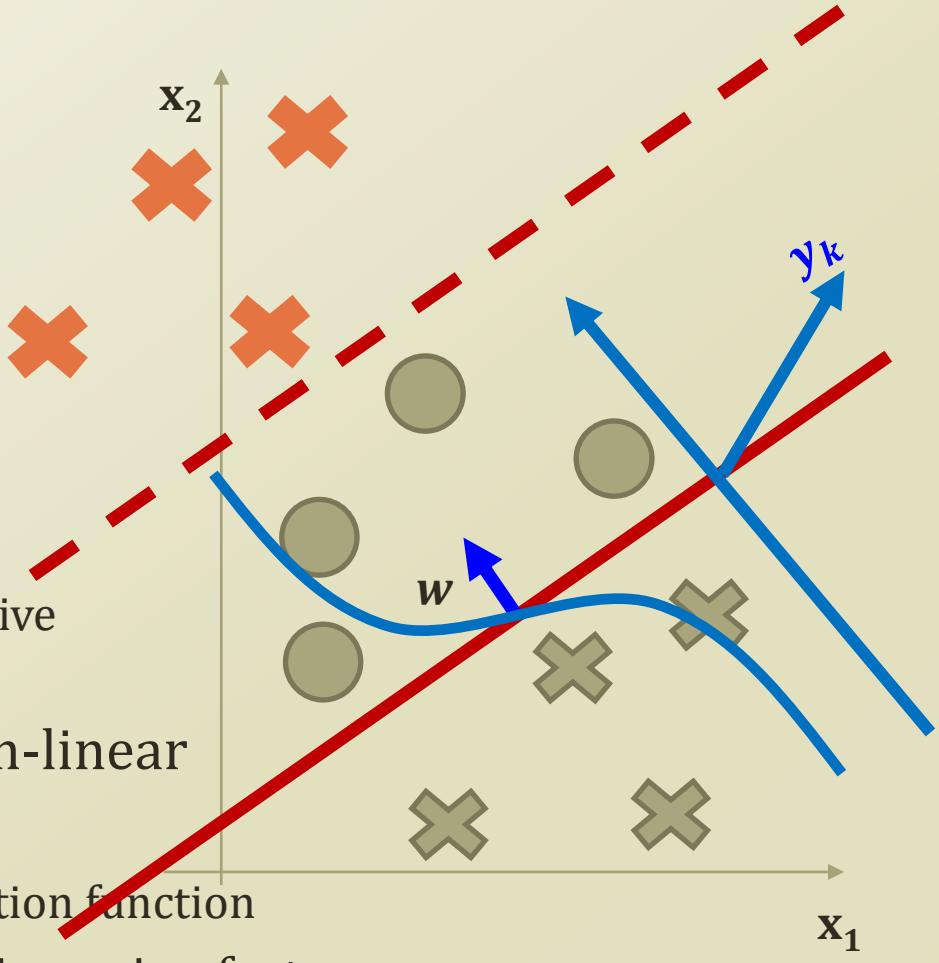


Logistic Function

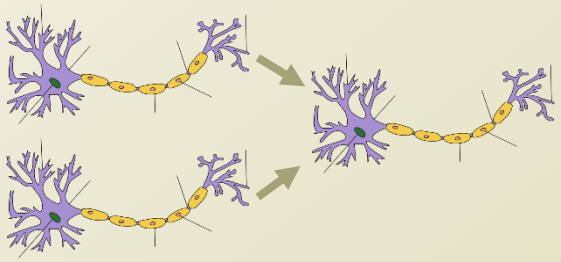


Problem of Linear Decision Boundary

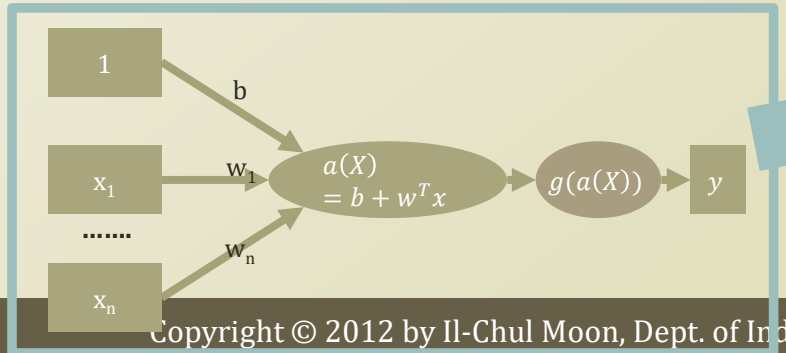
- Consider the sandwich case
  - The orange X marks cannot be classified as the negative by the model limitation
- Many real world applications have non-linear cases
  - Any cases with peak as positive
    - Then, the before and the after around the peak are negative → Non-linear boundary
- Ways to make the boundary non-linear
  - Make the model more complex
    - Change the neuron input activation function
  - Apply the model to the higher dimension features

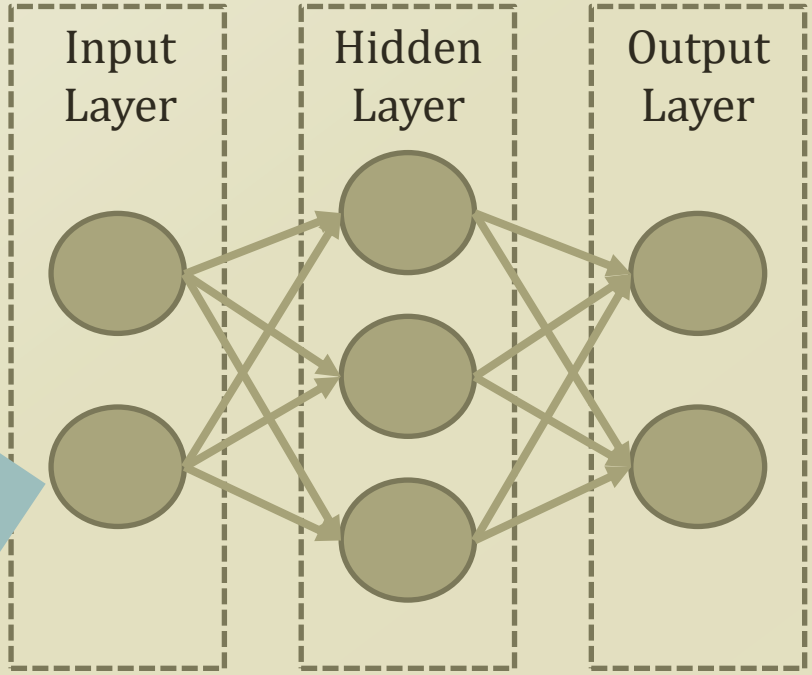


Artificial Neural Network



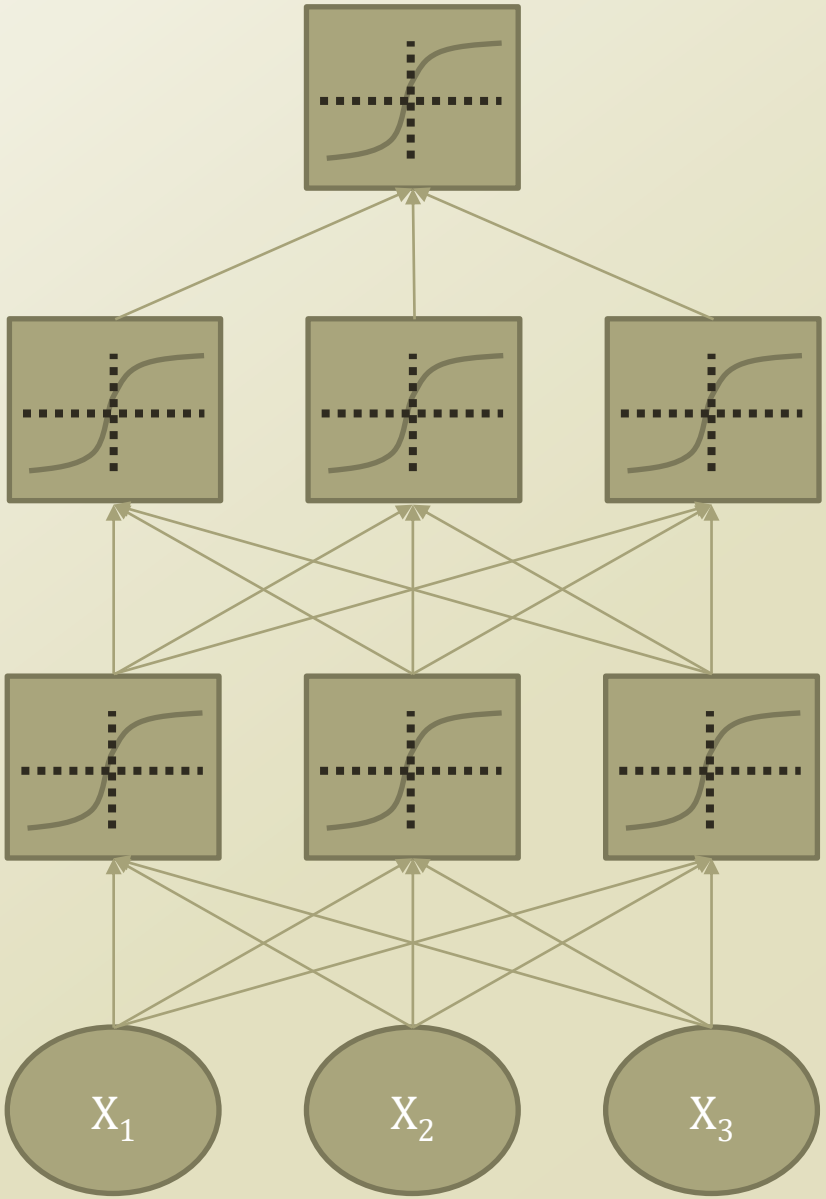
- To overcome the linear decision boundary of a single perceptron, we use multiple perceptrons structured in a certain form.
- Artificial neural network
  - A function approximation method with a collection of interconnected perceptrons
  - Structured by stacking layers
    - Input layer
    - Hidden layer
    - Output layer
  - Can create a multiple output





Multiple Layers of Neural Network

- Considering ***L*** hidden layers
  - Neuron input activation function for the ***k-th*** layer
    - $a^k(x) = b^k + W^k h^{(k-1)}(x)$
  - Neuron output activation function for the ***k-th*** layer
    - $h^k(x) = g\left(a^k(x)\right)$
  - Output layer activation
    - $h^{L+1}(x) = o\left(a^{L+1}(x)\right) = f(x)$
- Each function is available by choices
  - Sigmoid functions
  - Softmax functions



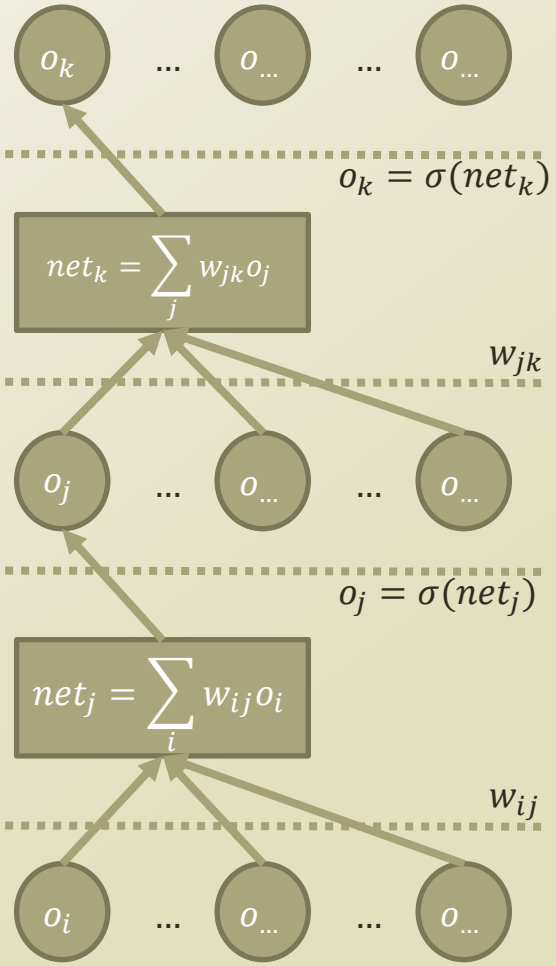
Backpropagation

- Backpropagation
  - Do
    - For training examples, x

      // forward pass of the neural net.
      - $o_k = f(x; w)$
      - $E = \frac{1}{2}(\sum_k (t_k - o_k)^2)$

      // backward pass of the neural net.
      - Calculate $\delta_k = (t_k - o_k) o_k (1 - o_k)$
      - For the backward-pass from the top to the bottom
        - Calculate $\delta_j = o_j(1 - o_j) \sum_k \delta_k w_{jk}$

      // weight update
      - Update w_{jk} with $w_{jk}^{t+1} \leftarrow w_{jk}^t + \eta \delta_k o_j$
      - For the backward-pass from the top to the bottom
        - Update $w_{ij}^{t+1} \leftarrow w_{ij}^t + \eta o_i \delta_j$
  - Until converges



Problem of Artificial Neural Network

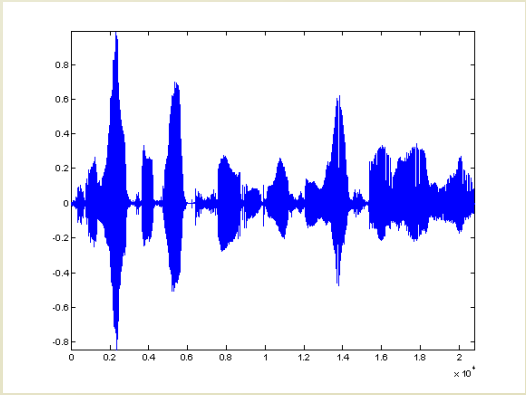
- Problem of gradient method
  - Vanilla version of gradient method
  - Random initialization?
    → Potentially very long learning process
    → Risk of local optima and saddle points
- Problem of back-propagation
  - What-if many layers?
    - Multiplication of many gradients → Getting close to zero
  - Computation time
    - A single logistic regression → Multiple logistic regression with interactions
    - Long training time
- Problem of structure
  - Is feasible and meaningful to fully connect the neurons?

LeCun, Yann, Yoshua Bengio, and Geoffrey Hinton. "Deep learning." *Nature* 521.7553 (2015): 436-444.

CONVOLUTIONAL NEURAL NETWORKS

Neural Networks for Various Domains

- Previous structure : Fully connected networks
- Recent advances in neural networks
  - Computer vision
  - Language models
- Application domain influences network structures
  - Convolutional structure utilizing localized connections
  - Recurrent structure utilizing chain connections



Speech Recognition

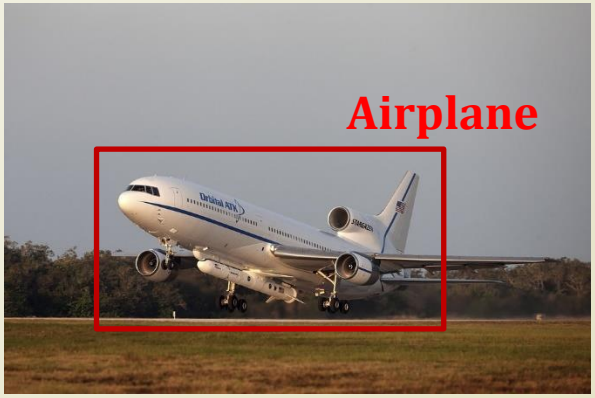
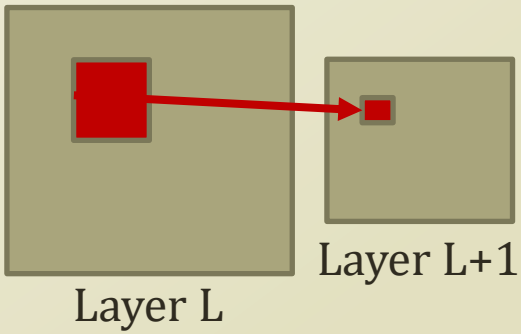
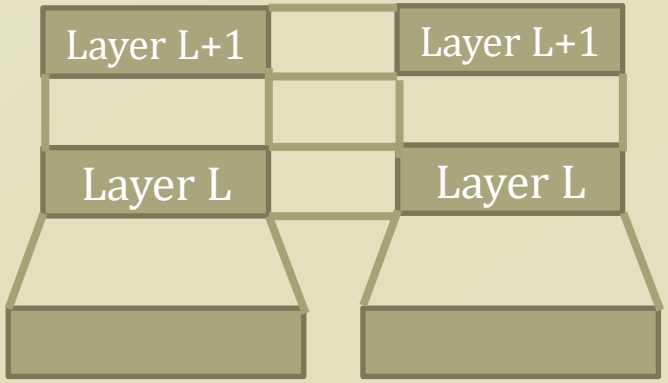


Image Classification
Object Recognition



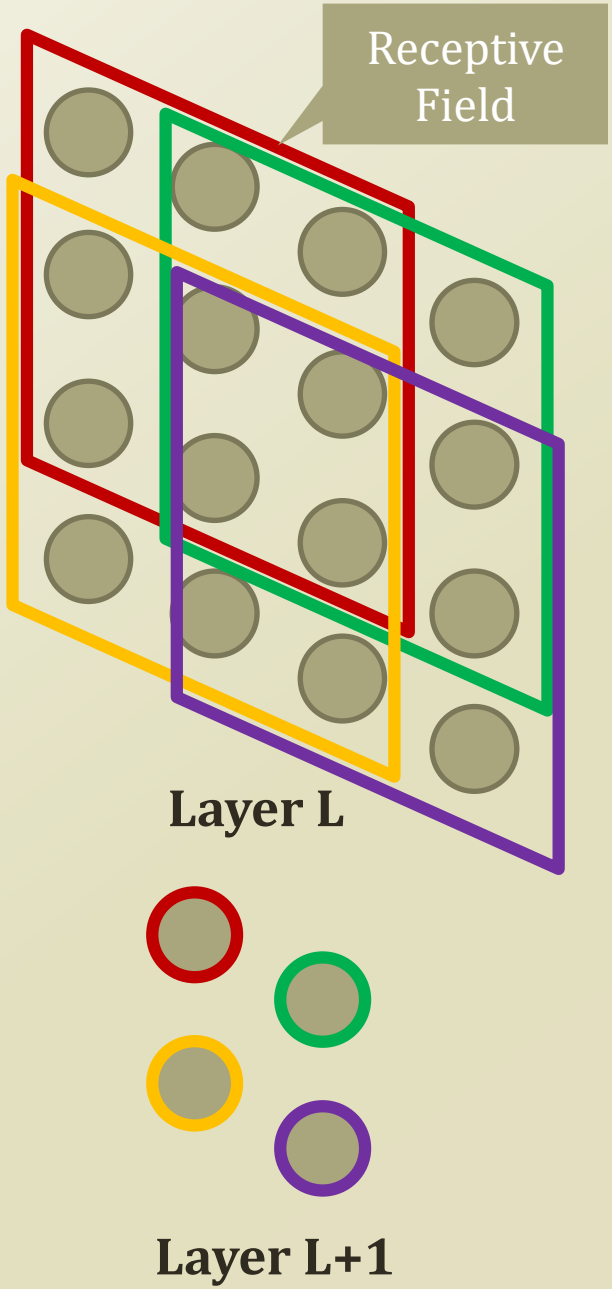


Example : Computer Vision

- Vision problem characteristics
  - High dimensional inputs
    - 150 X 100 pixels = 15000 pixels
    - 15000 pixels X 3 RGB channels = 45000 real value inputs
  - 2D topology of pixels for images and 3D topology of pixels for video
  - Need to handle invariance
    - Rotation, Translation, Scale…
- Convolutional neural networks mitigates these characteristics
  - Local connectivity
  - Convolution
  - Pooling

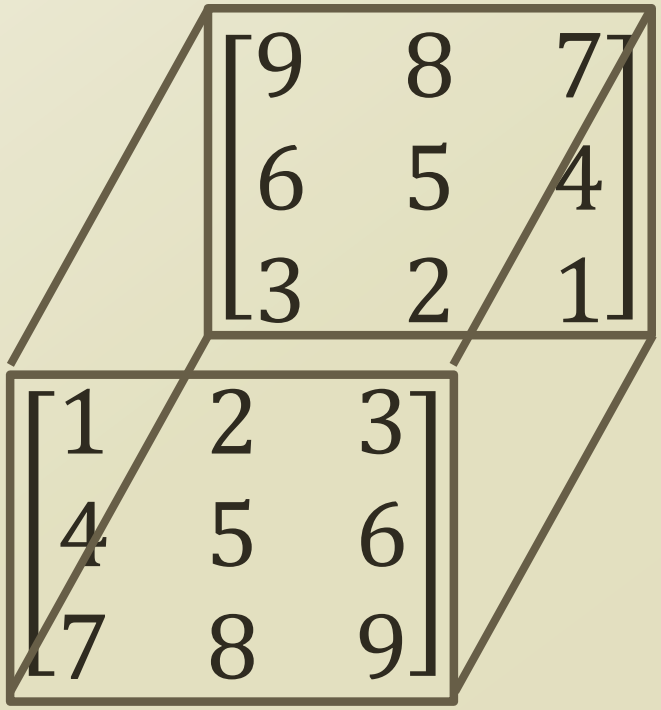
Local Connectivity

- Use a local connectivity of hidden units
  - Each hidden unit is connected to only to a sub-region of the input images
  - Connected to all channels : 1 channel of grayscale and 3 RGB channel of color images
- Advantages of local connectivity
  - # of a fully connected layer >>> # of a sparsely connected Layer
    - More number of parameters to optimize
      - More data, more difficulties in learning
  - How to sparsely connect?
    - Utilizing the 2D topologies : Receptive field



Vector, Matrix, Tensor…

- Tensor is a geometric object that describe linear relations between geometric vectors, scalars, and other tensors.
  - Relations : dot product, cross product…
    - Dot product : $\mathbf{a} \cdot \mathbf{b} = a_1 b_1 + \cdots + a_n b_n$
    - Cross product : $\mathbf{a} \times \mathbf{b} = \hat{\boldsymbol{n}} |\mathbf{a}| |\mathbf{b}| \sin\theta$
  - Scalar : 0-dimensional tensor
    - 0, 1, 2….
  - Vector : 1-dimensional tensor
    - <0,1>, <1,2> …
  - Matrix : 2-dimensional tensor
    - $\begin{bmatrix} 1 & 2 \\ 3 & 4 \end{bmatrix}$….
  - Tensor : Multi-dimensional tensor (so called)
  - Dimension of a tensor → rank
- Can be viewed as a multi-dimensional array
  - Array can be viewed as a storage of a tensor

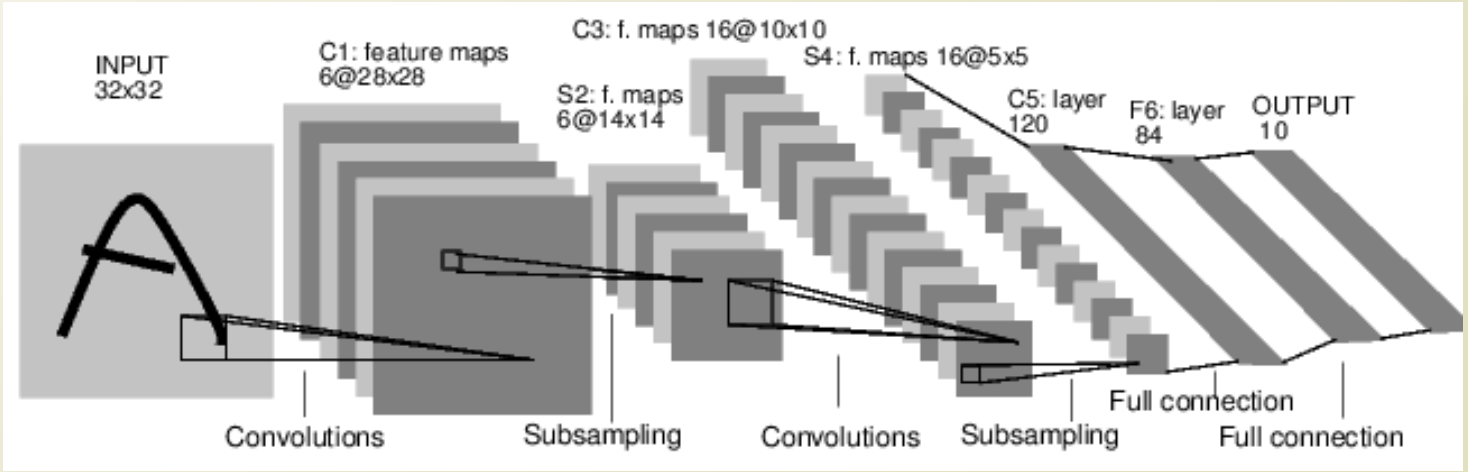


3-dimensional Tensor

Convolutional Neural Network

- Convolutional Neural Network (CNN)
  - Designed to recognize visual patterns directly from pixel images with minimal preprocessing
  - Recognize patterns with extreme variability, and with robustness to distortions and simple geometric transformations
  - Three main types of layers
    - Convolutional (C) / Pooling (S) / Fully-Connected (F) Layer

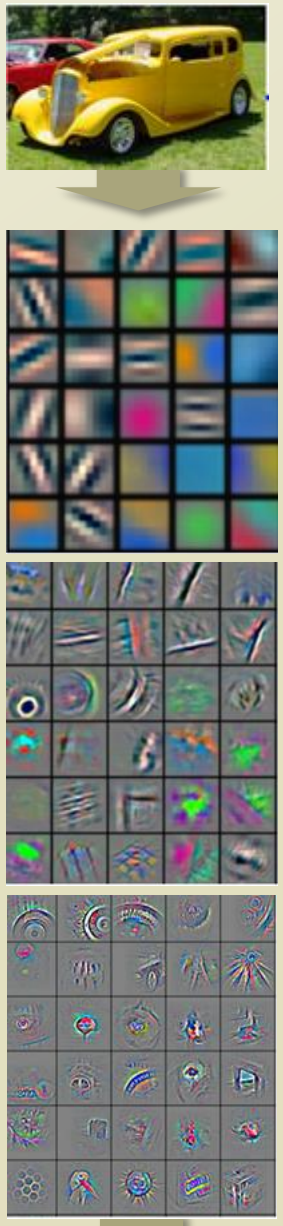
- Example Architecture: LeNet [12]



  - [INPUT] → [CONV – POOL]*2 → [FC]*2
  - Developed for digits/hand-written recognition

Hierarchical feature learning

lower

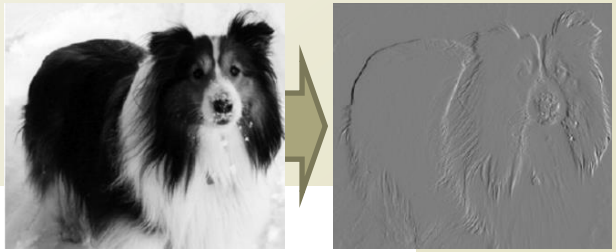


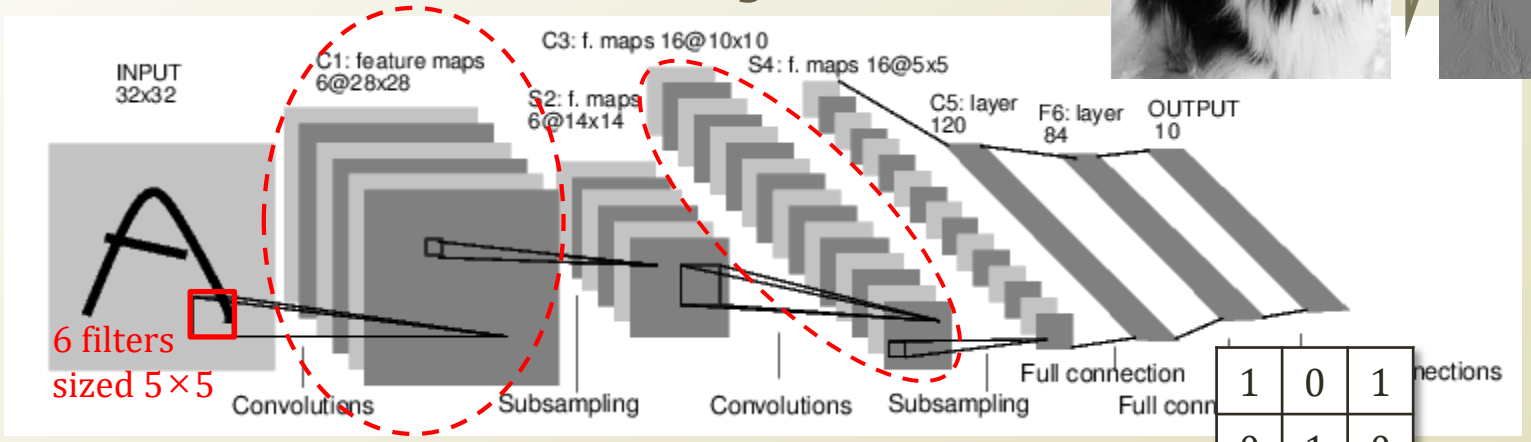
higher

*Figure from [13]

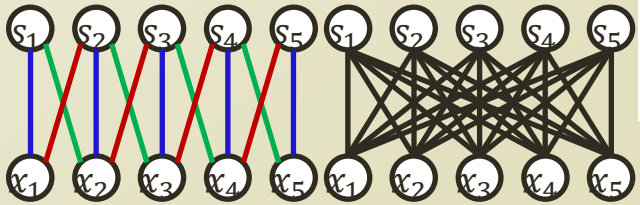
Car

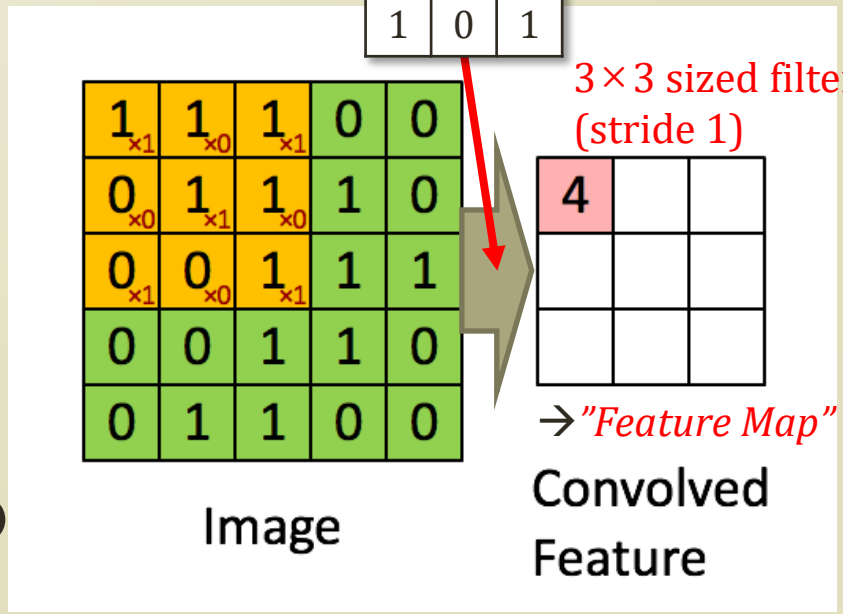
Convolutional Layer





- Convolution Operation
  - $s(t) = (x * w)(t) = \int x(a)w(t-a)da$
    - x: data, w: filter/kernel, $s(t)$: feature map
- Convolutional Layer
  - Core building block of CNN
  - Parameters: a set of learnable filters
  - Local connectivity & Shared/Tied weights
    - Model complexity ↓
    - Efficient



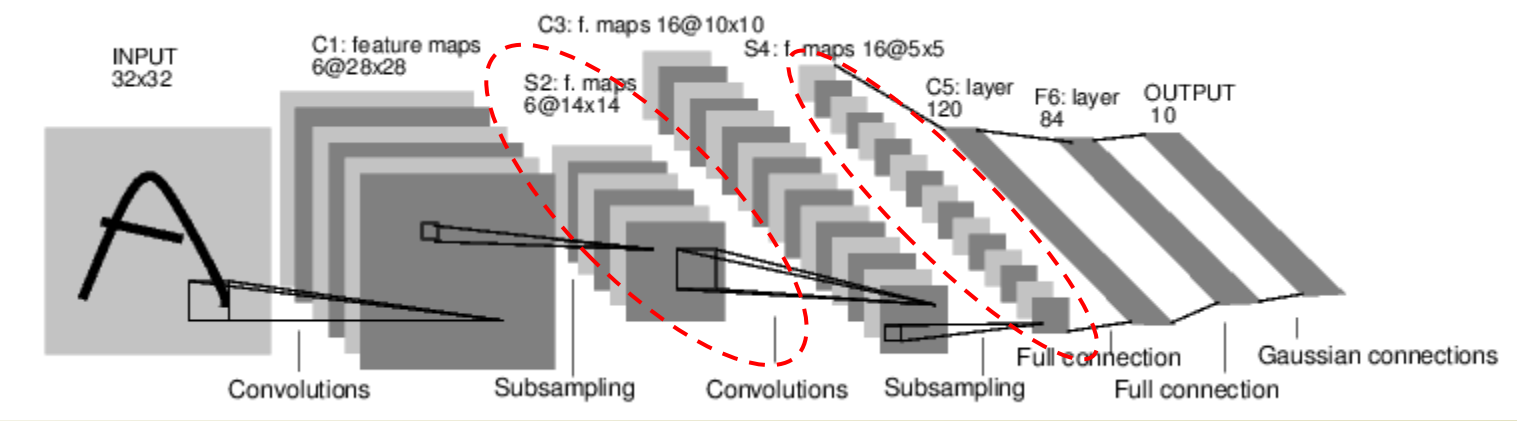


Receptive fields

Convolution Layer

- Each feature map forms a 2D grid of features
  - Can be computed with a discrete convolution (*) of a kernel matrix k_{ij}
    - Which is the hidden weights matrix W_{ij} with its rows and columns flipped
    - The kernel can be matched to the receptive field dimension
- Mathematical definition on convolution
  - L-th layer has the neuron with $n_1 \times n_2$
  - Kernel matrix, $K \in R^{(2h_1+1)\times(2h_2+1)}$
  - $(I * K)_{r,s} := \sum_{u=-h_1}^{h_1} \sum_{v=-h_2}^{h_2} K_{u,v} I_{r+u,s+v}$
    - There are exception cases when we compute the convolution at the edge of the images with the negative indexes
      - Limit the convolution range to avoid the negative indexes
      - Padding values, usually zero (so called zero padding) to the edge of the layer
- Convolution layer
  - $(Y_i^{(l)})_{r,s} = \sigma\left(\left(B_i^{(l)}\right)_{r,s} + \sum_{j=1}^{m_1^{(l-1)}} \sum_{u=-h_1^{(l)}}^{h_1^{(l)}} \sum_{v=-h_2^{(l)}}^{h_2^{(l)}} \left(K_{i,j}^{(l)}\right)_{u,v} \left(Y_j^{(l-1)}\right)_{r+u,s+v}\right)$
    - $m_2^{(l)} = \frac{m_2^{(l-1)} - 2h_1^{(l)}}{s_1^{(l)}+1}, m_3^{(l)} = \frac{m_3^{(l-1)} - 2h_2^{(l)}}{s_2^{(l)}+1}$
  - l : layer index
  - r, s : 2D neuron index at layer l
  - $m_1^{(l-1)} \times m_2^{(l-1)} \times m_3^{(l-1)}$: Dimension of layer (*l-1)* by Depth X Width X Height
  - $s_1^{(l)}, s_2^{(l)}$ *:* Stride (skipping factor) of the kernel matrix on the width and the height dim.

Pooling Layer



- Common to periodically insert a pooling layer in-between successive convolutional layers
- Progressively reduce the spatial size of the representation → #params. & computation ↓
- Types
  - **Max** / Average / L2-Norm / Stochastic pooling
- Common form
  - 2×2 filters applied with a stride of 3
    - Called overlapping pooling
  - 2×2 filters applied with a stride of 2
    - Discard 75% activation

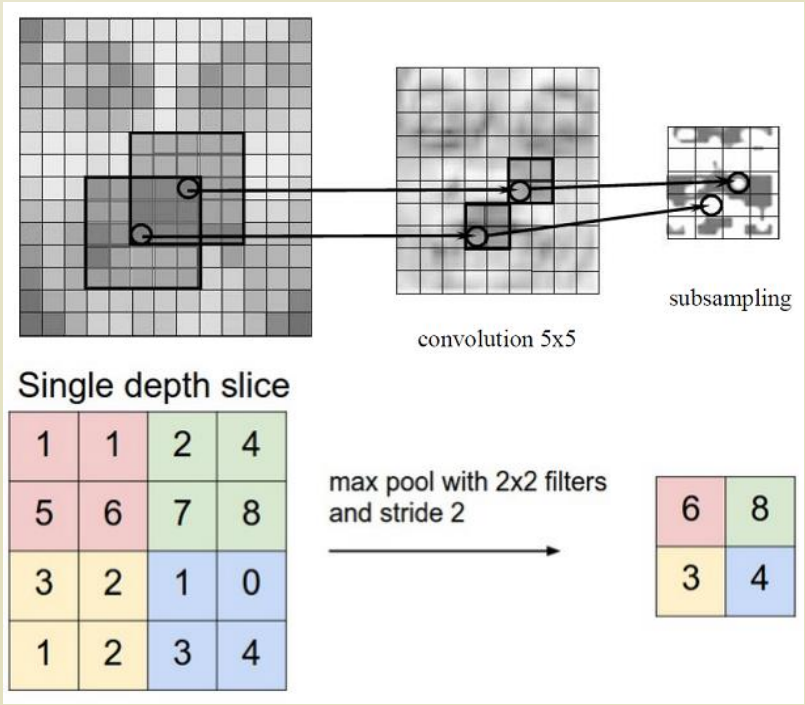
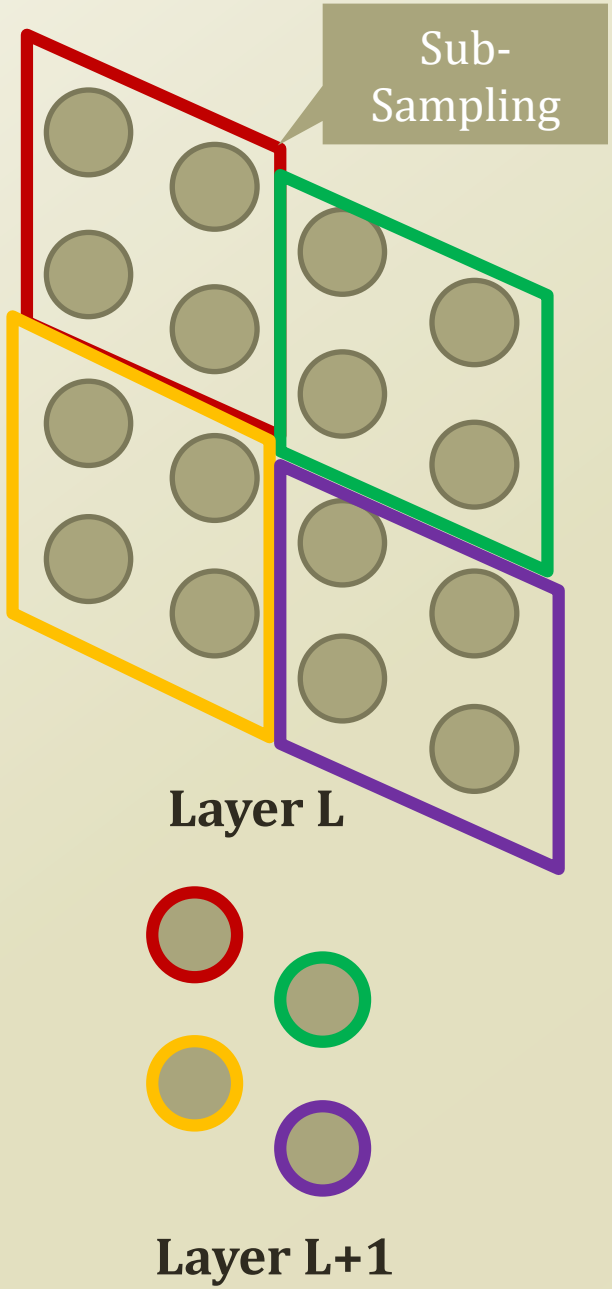


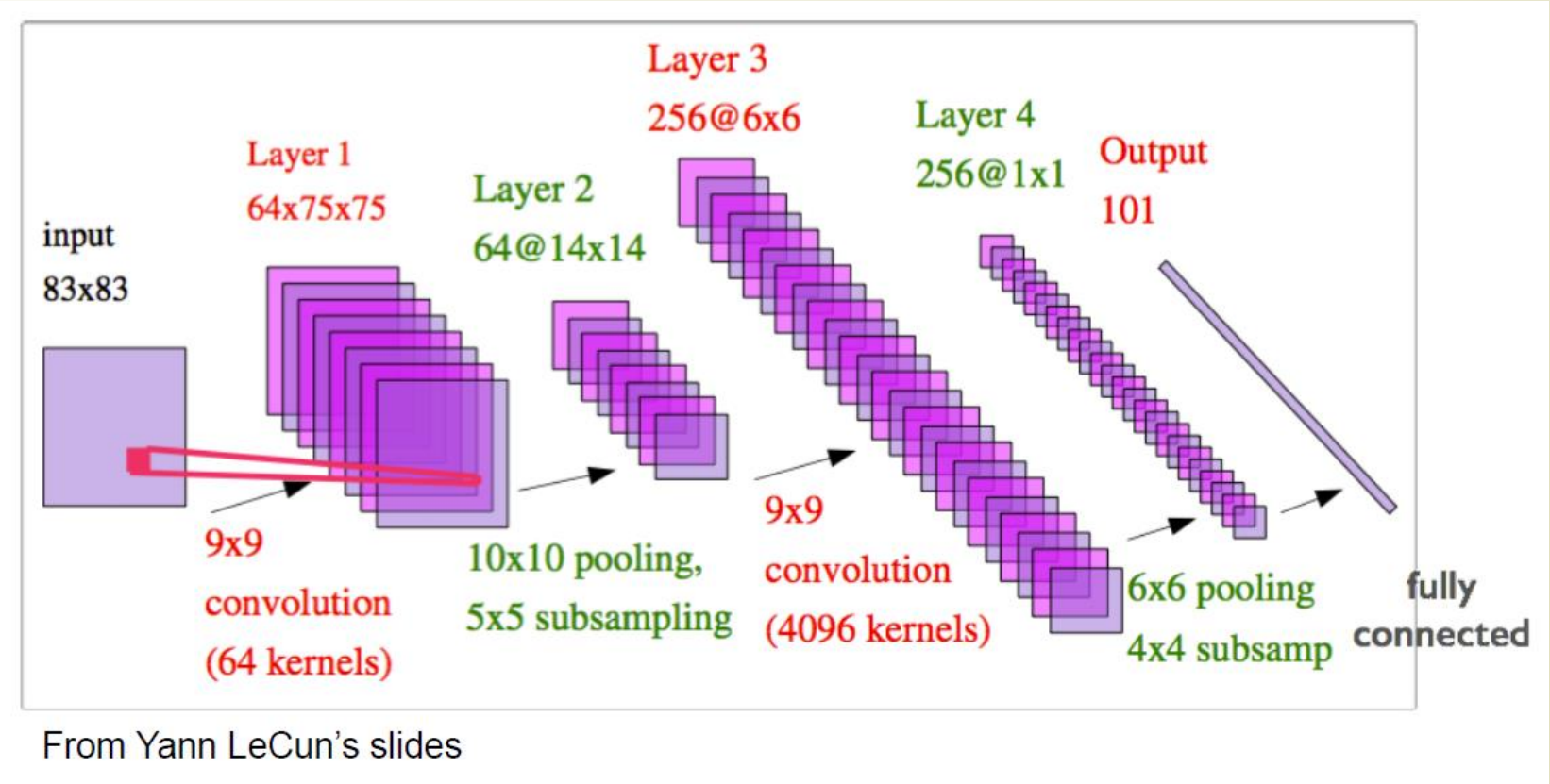
Figure adopted from Fei-Fei Li & Andrej Karpathy, CS231n: Convolutional Neural Networks for Visual Recognition, Winter 2015

Pooling Layer

- Pool hidden units in the same neighborhood
  - Introduce invariance to local translations
- Definition of various pooling
  - $(Y_i^{(l)})_{r,s} = f\left(\left(Y_i^{(l-1)}\right)_{r-h_1,s-h_2}, \ldots, \left(Y_i^{(l-1)}\right)_{r+h_1,s+h_2}\right)$
  - Max pooling : $f(Y_i) = \max(Y_i)$
  - Average pooling : $f(Y_i) = \frac{1}{h_1 \times h_2} \sum Y_i$



Example of Convolution Neural Network

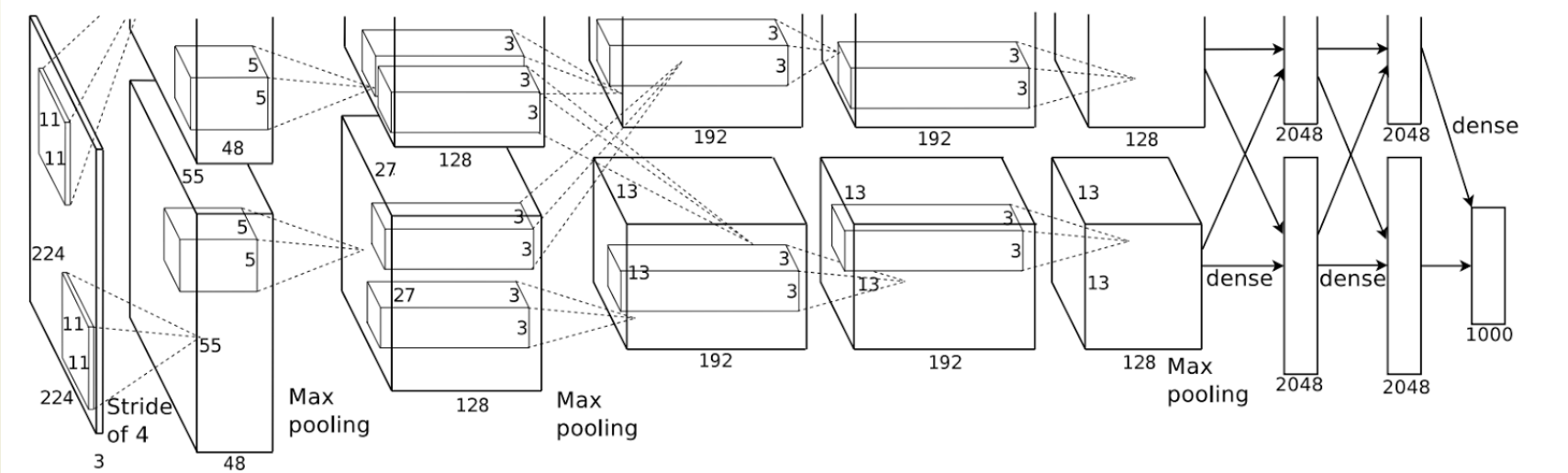


From Yann LeCun's slides

- Should be able to read the structure presented as the above diagram
- Convolution layer, pooling layer, fully connected layer
- Require to use diverse kernels and carefully calibrated hyperparameters
- Still learnable with the back-propagation with some techniques

Existing Developed CNN Architectures

- Various CNN architectures
  - LeNet, AlexNet (TorontoNet), ZFNet, VGGNet, GoogleNet
- AlexNet (TorontoNet) – First prize of ILSVRC 2012 challenge



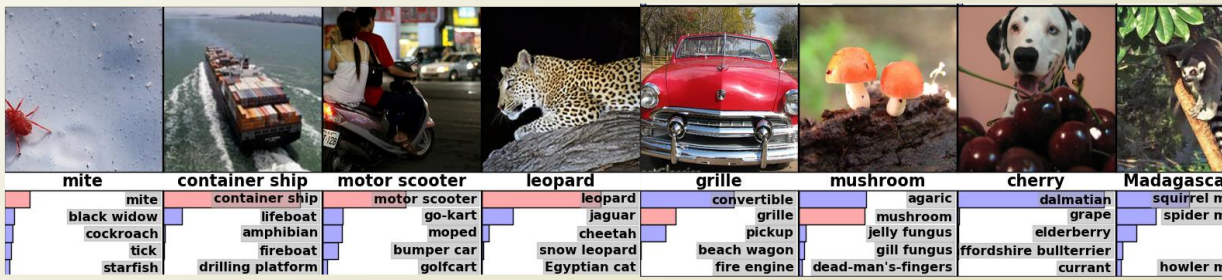
  - Caffe implementation (https://github.com/BVLC/caffe/tree/master/models/bvlc_alexnet)
  - Main Features
    - 7 hidden layers & about 650,000 units & about 60,000,000 parameters
    - Dropout, ReLU instead of sigmoid
    - Parallelization on many GPUs (50x speedup over CPU)
      - Trained on two GPUs for a week

IMAGE ANALYSIS ACHIEVEMENTS

Images….

- Various Image Tasks
  - Image classification (Object recognition)

[1]



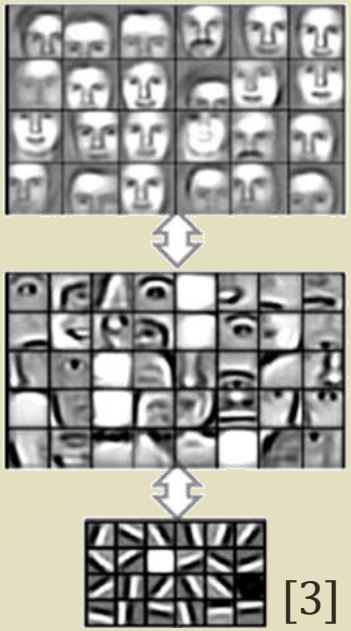
  - Image segmentation / Object detection

[2]

- Approaches for Recognition: "Shallow" vs. "Deep" Architectures
  - Shallow

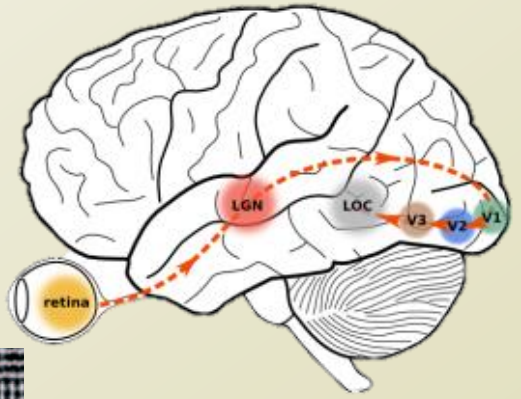
    - Features are not learned
  - Deep

    - Learn hierarchical features
    - Each layer extracts features from the output of previous layer

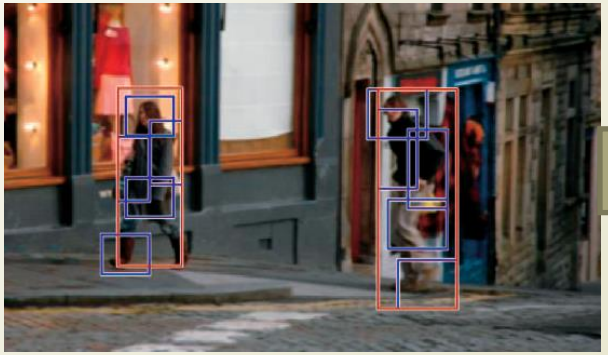


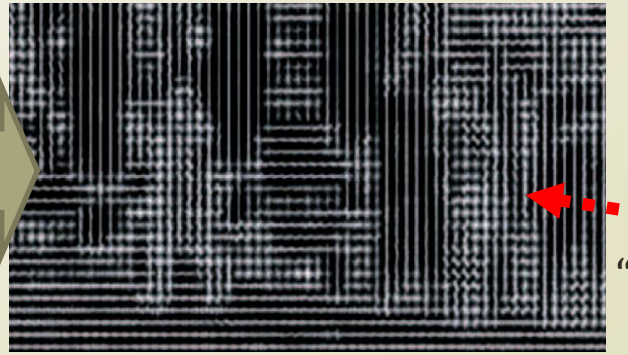
[3]

Object Detection Task

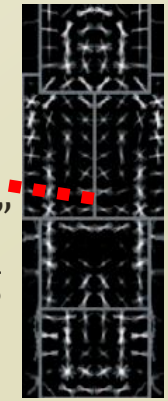


- Based on "Shallow" Architectures
  - **DPM**, UVA, Regionlets
    - **Histogram of Oriented Gradient (HOG)** / Scale Invariant Feature Transform (SIFT) - based

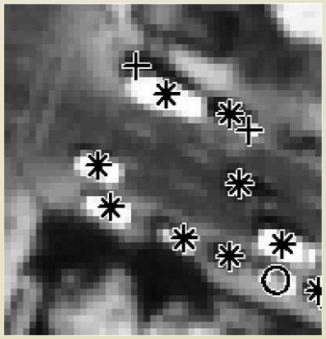




"*template*" matching



  - Small gains obtained in PASCAL VOC object detection
- Based on "Deep" Architectures
  - Neural networks have been used typically on constrained object categories; face/pedestrian/vehicle detection



Region-Based CNN

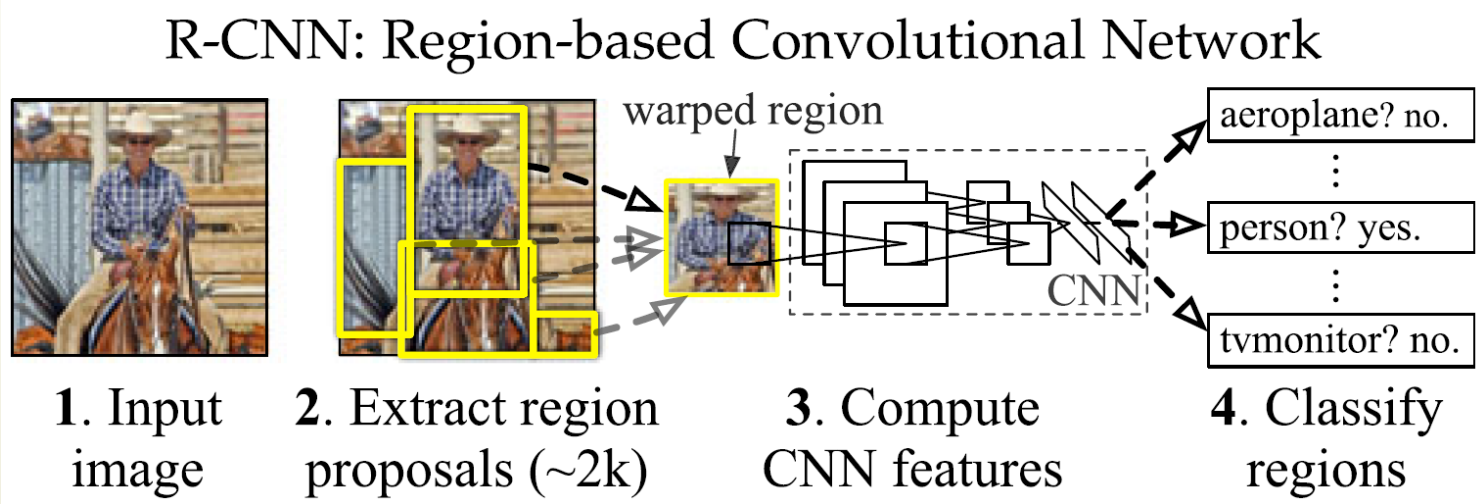
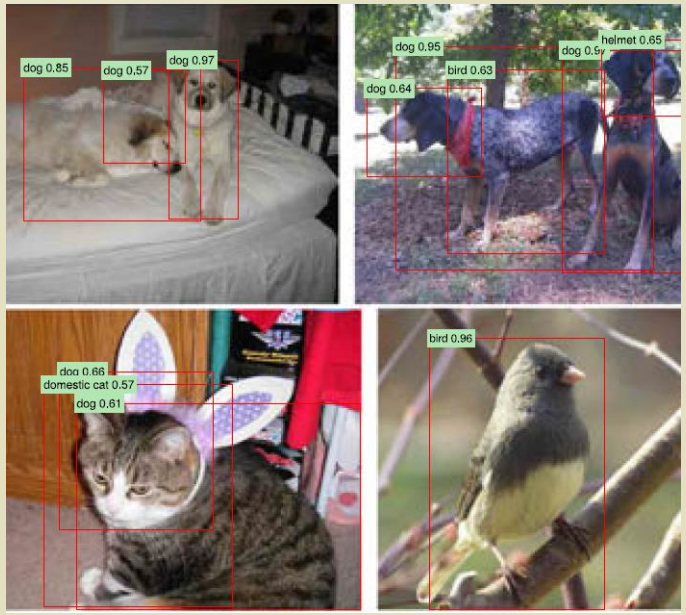
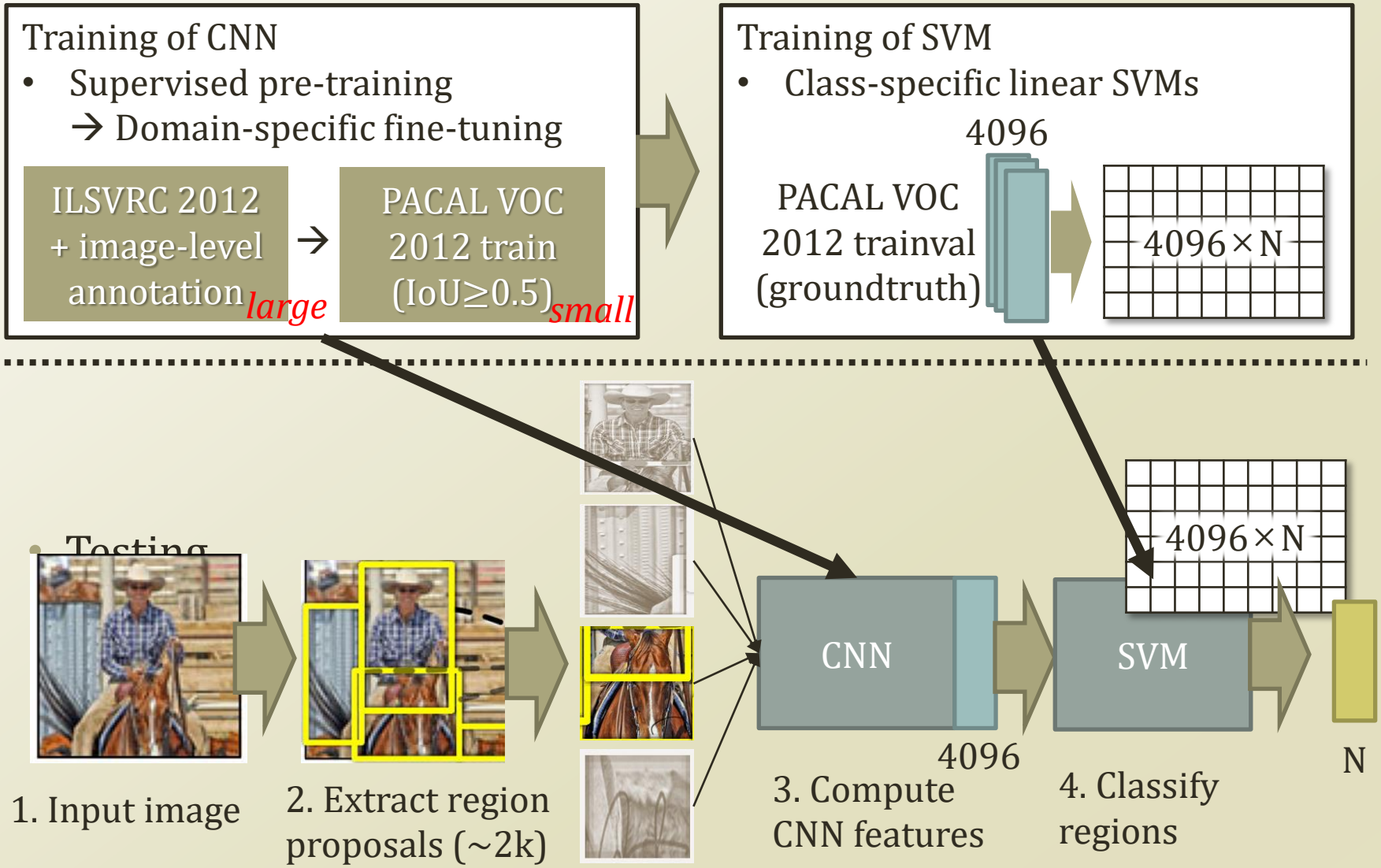


Figure: Object Detection System Overview

- Main Contributions
  1. Simple and scalable object detection
  2. Region proposals + CNN
  3. Training for scarce labeled data
     - Supervised pre-training
     - Domain-specific fine-tuning

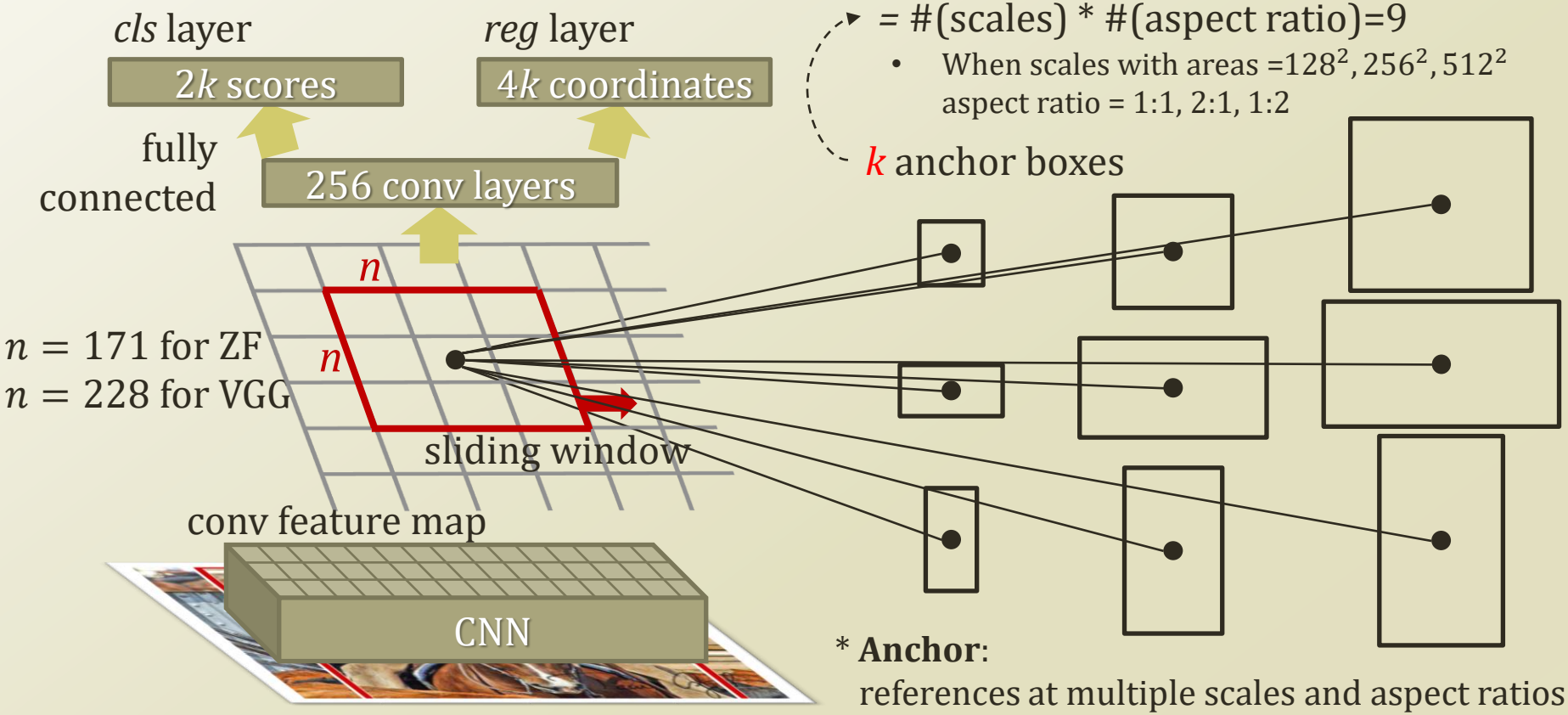


Structure of Region-Based CNN



Region Proposal Network

- A fully convolutional network that simultaneously predicts object bounds and objectness scores at each position
  - Input: an image
  - Output: a set of (square) object proposals, each with an objectness score



Framework of Mask R-CNN

- Mask R-CNN
  - Region proposals by Region Proposal Network (RPN)
    - Alternative external method; e.g., selective search
  - Object detection by Fast R-CNN

  ] **Faster R-CNN**

  - +) Semantic Segmentation by Fully Convolutional Network (FCN)
  - RoIPool → RoIAlign

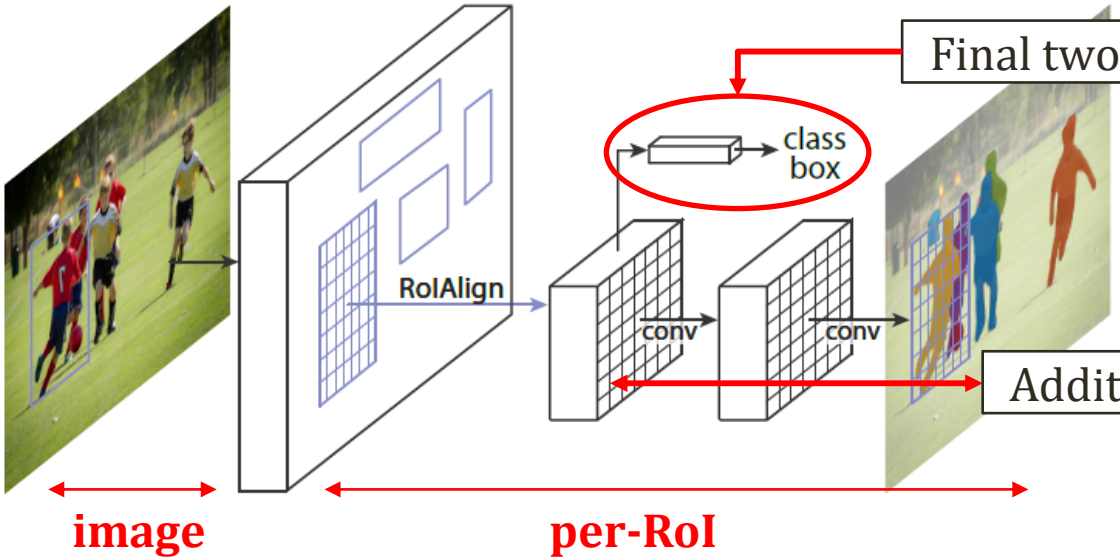


Figure 1. The **Mask R-CNN** framework for instance segmentation.

#Branch of sub-network per RoI
: 2 (Faster R-CNN)
→ 3 (Mask R-CNN)

→ *"Mask Branch"*

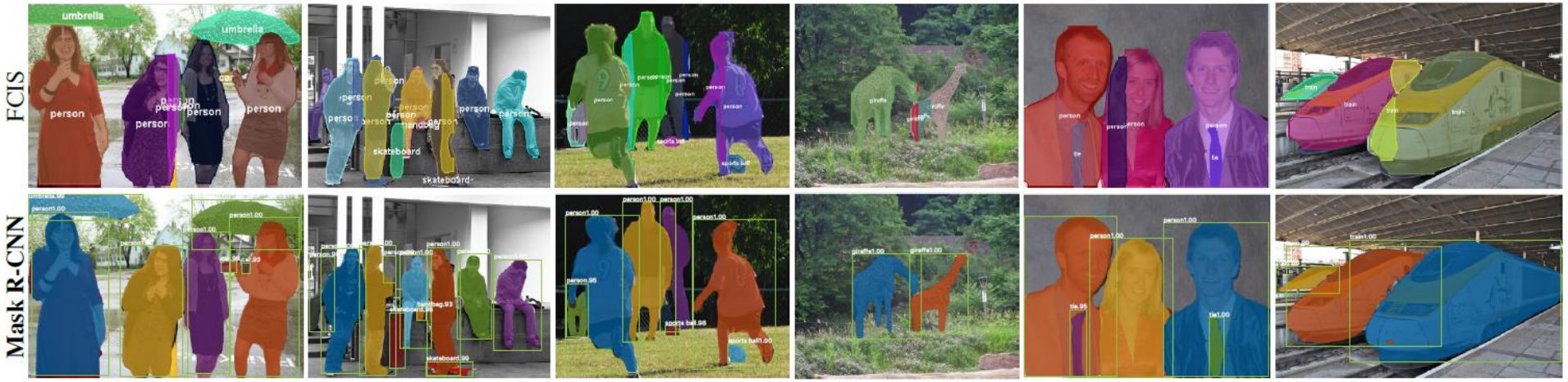


Figure 5. FCIS+++ [26] (top) *vs.* Mask R-CNN (bottom, ResNet-101-FPN). FCIS exhibits systematic artifacts on overlapping objects.

net-depth-features	AP	AP_{50}	AP_{75}
ResNet-50-C4	30.3	51.2	31.5
ResNet-101-C4	32.7	54.2	34.3
ResNet-50-FPN	33.6	55.2	35.3
ResNet-101-FPN	35.4	57.3	37.5
ResNeXt-101-FPN	**36.7**	**59.5**	**38.9**

(a) **Backbone Architecture**: Better backbones bring expected gains: deeper networks do better, FPN outperforms C4 features, and ResNeXt improves on ResNet.

	AP	AP_{50}	AP_{75}
softmax	24.8	44.1	25.1
sigmoid	**30.3**	**51.2**	**31.5**
	+5.5	*+7.1*	*+6.4*

(b) **Multinomial *vs.* Independent Masks** (ResNet-50-C4): *Decoupling* via per-class binary masks (sigmoid) gives large gains over multinomial masks (softmax).

	align?	bilinear?	agg.	AP	AP_{50}	AP_{75}
RoIPool [12]			max	26.9	48.8	26.4
RoIWarp [10]		✓	max	27.2	49.2	27.1
		✓	ave	27.1	48.9	27.1
RoIAlign	✓	✓	max	**30.2**	**51.0**	**31.8**
	✓	✓	ave	**30.3**	**51.2**	**31.5**

(c) **RoIAlign** (ResNet-50-C4): Mask results with various RoI layers. Our RoIAlign layer improves AP by ~3 points and AP_{75} by ~5 points. Using proper alignment is the only factor that contributes to the large gap between RoI layers.

	AP	AP_{50}	AP_{75}	AP^{bb}	AP^{bb}_{50}	AP^{bb}_{75}
RoIPool	23.6	46.5	21.6	28.2	52.7	26.9
RoIAlign	**30.9**	**51.8**	**32.1**	**34.0**	**55.3**	**36.4**
	+7.3	*+ 5.3*	*+10.5*	*+5.8*	*+2.6*	*+9.5*

(d) **RoIAlign** (ResNet-50-**C5**, *stride 32*): Mask-level and box-level AP using *large-stride* features. Misalignments are more severe than with stride-16 features (Table 2c), resulting in massive accuracy gaps.

	mask branch	AP	AP_{50}	AP_{75}
MLP	fc: 1024→1024→80·28^2	31.5	53.7	32.8
MLP	fc: 1024→1024→1024→80·28^2	31.5	54.0	32.6
FCN	conv: 256→256→256→256→256→80	**33.6**	**55.2**	**35.3**

(e) **Mask Branch** (ResNet-50-FPN): Fully convolutional networks (FCN) *vs.* multi-layer perceptrons (MLP, fully-connected) for mask prediction. FCNs improve results as they take advantage of explicitly encoding spatial layout.

Table 2. **Ablations** for Mask R-CNN. We train on `trainval35k`, test on `minival`, and report *mask* AP unless otherwise noted.

Image+Domain

- Human Pose Estimation
  - Human pose
    → 17 keypoints
    → 17 masks
- Only a single pixel is labeled as foreground

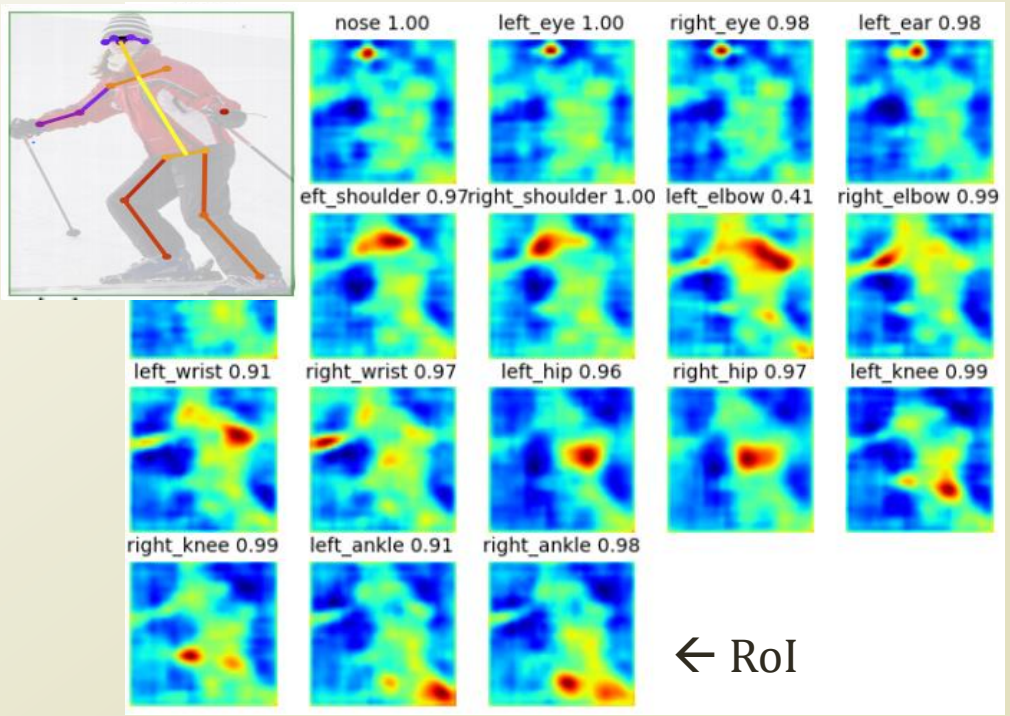
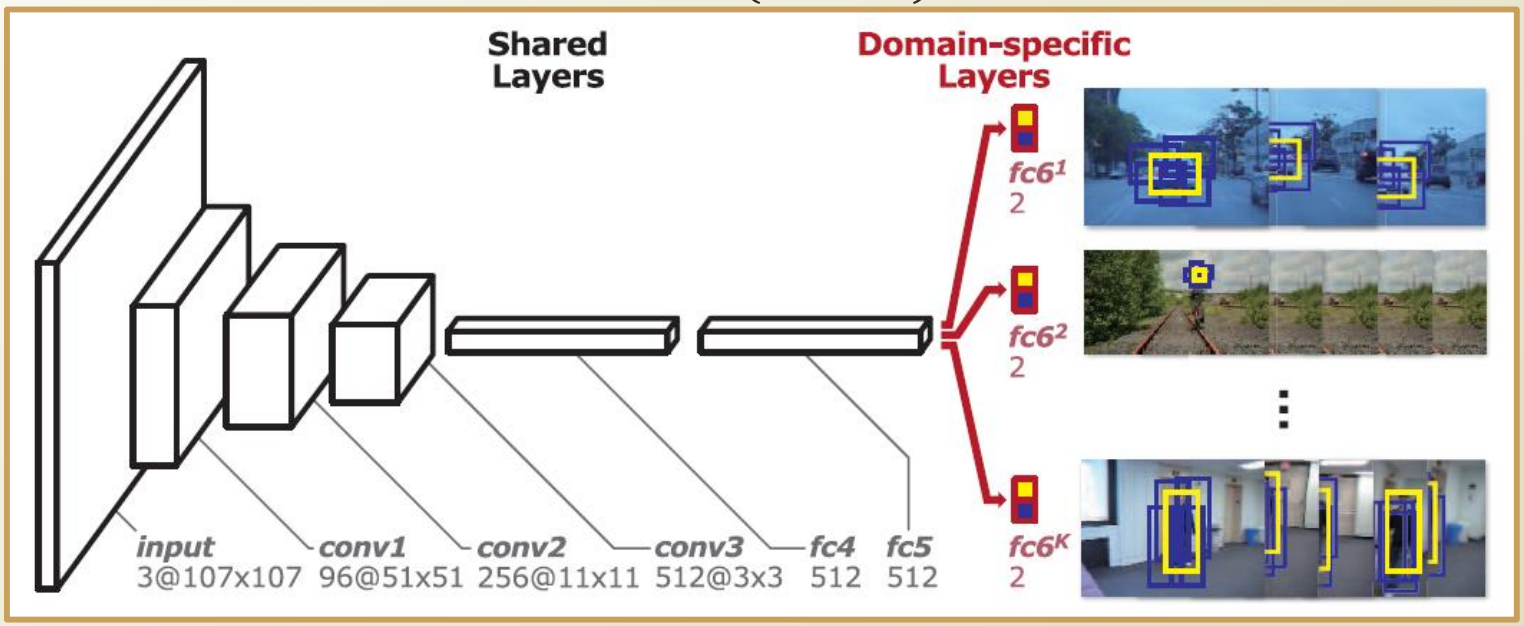


Figure 6. Keypoint detection results on COCO `test` using Mask R-CNN (ResNet-50-FPN), with person segmentation masks predicted from the same model. This model has a keypoint AP of 63.1 and runs at 5 fps.

Multi-Domain Network (MDNet)

- Pre-train Multi-Domain Network (MDNet)

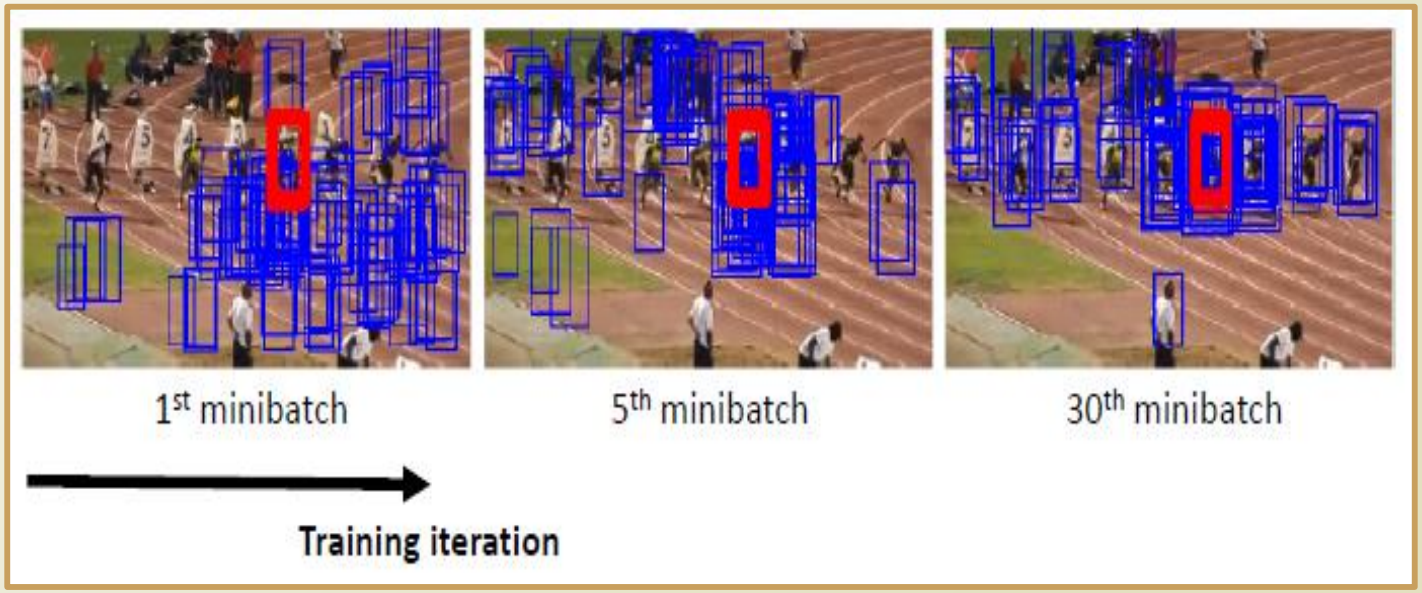


  - Convolutional layers
    - Identical to the corresponding parts of VGG_M network
  - Fully-connected layers
    - 512 output units and are combined with ReLUs and dropouts
  - K branches
    - Binary classification layer with softmax cross-entropy loss
    - Distinguish target and background

Online Tracking with MDNet

- Real-time online tracking of moving objects
  - Hard Mini-batch Mining

Red boxes : positive samples
Blue boxes : negative samples

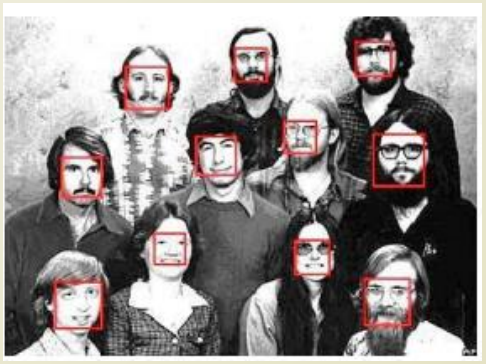


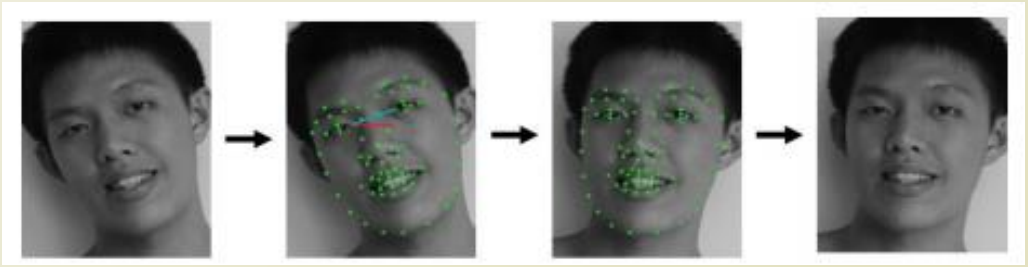
  - As learning proceeds and the network becomes more discriminative, the classification in a mini-batch becomes more challenging
  - This approach identifies critical negative examples (false positives) effectively

Human-level Face Verification

- Step1. Face Detection
- Step2. Face Alignment
- Step3. Feature Extraction
- Step4. Face Classification

Step1





Step2

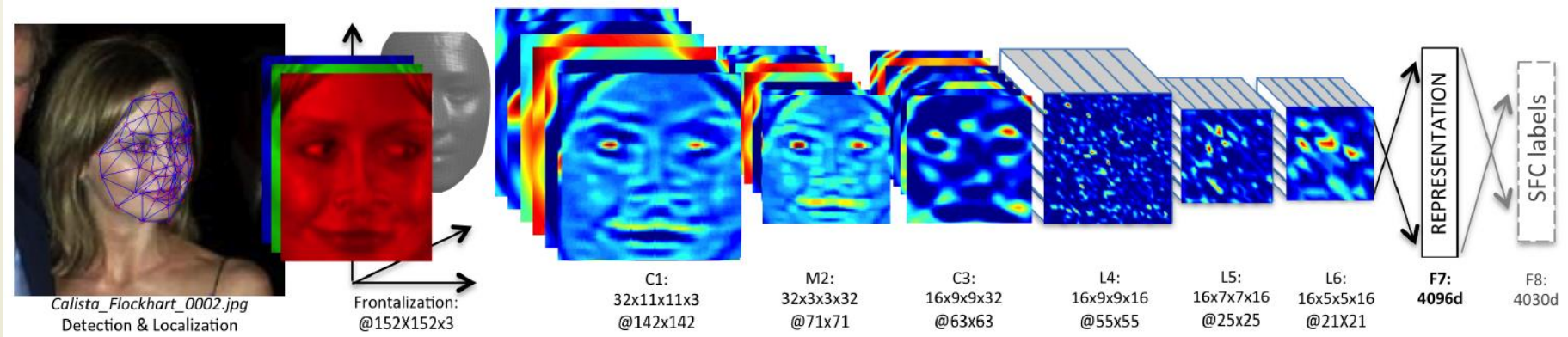


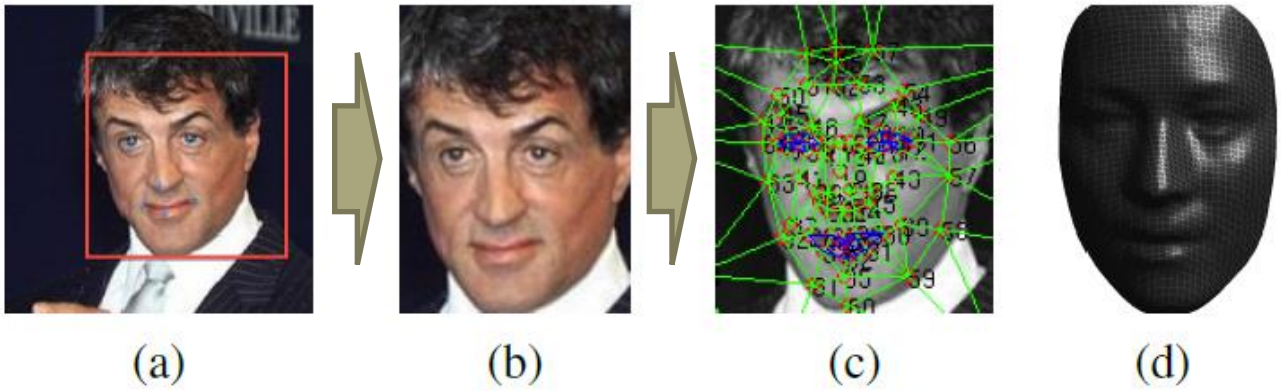
Figure 2. **Outline of the *DeepFace* architecture**. A front-end of a single convolution-pooling-convolution filtering on the rectified input, followed by three locally-connected layers and two fully-connected layers. Colors illustrate feature maps produced at each layer. The net includes more than 120 million parameters, where more than 95% come from the local and fully connected layers.

Preprocessing Faces

/w 6 initial fiducial points

- Step1. Face Detection
- Step2. Face Alignment

Reference 3D fiducial point location



warping (frontalization)

Final Alignment

- 2D Alignment

: $x_{anchor} = (S * R * T)x_{source}$

: Scaling & Rotation & Translate

- 3D Alignment

: take the average of the 3D scans from the USF Human-ID database

: piecewise affine transformation

Neural Network Structure for Face Verification

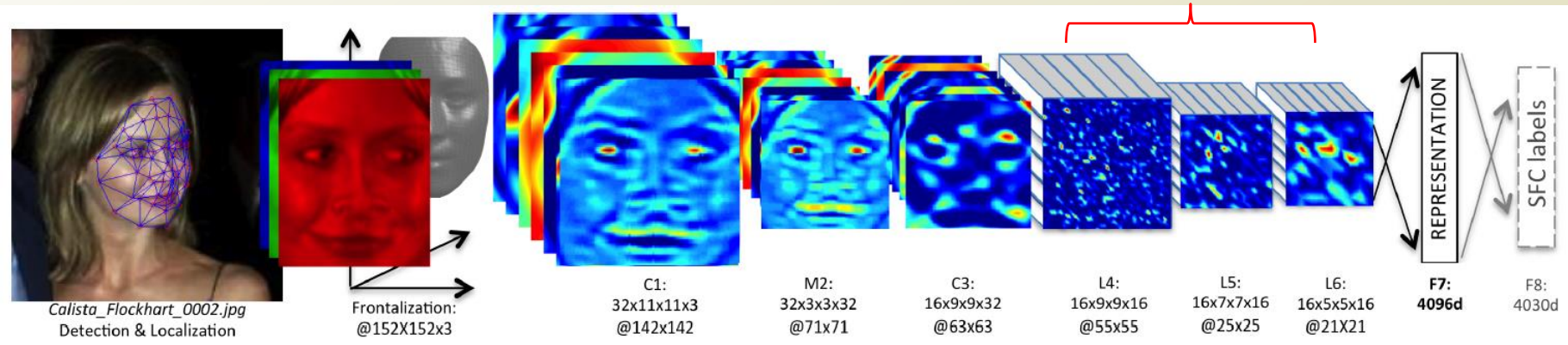
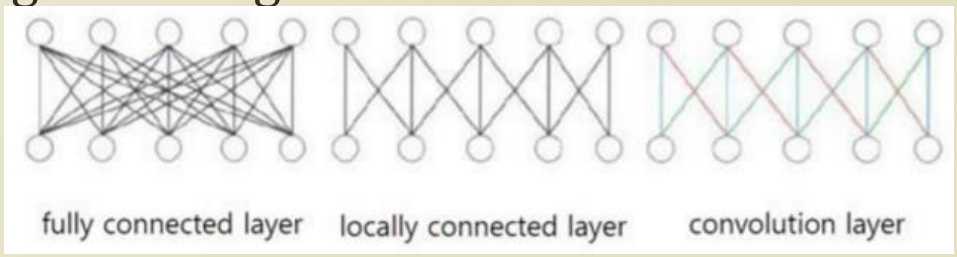


Figure 2. **Outline of the *DeepFace* architecture**. A front-end of a single convolution-pooling-convolution filtering on the rectified input, followed by three locally-connected layers and two fully-connected layers. Colors illustrate feature maps produced at each layer. The net includes more than 120 million parameters, where more than 95% come from the local and fully connected layers.

- Step3. Feature Extraction & Step4. Face Classification
  - Locally connected layer
    - No weight sharing unlike the standard convolutional layer



Face Verification Results

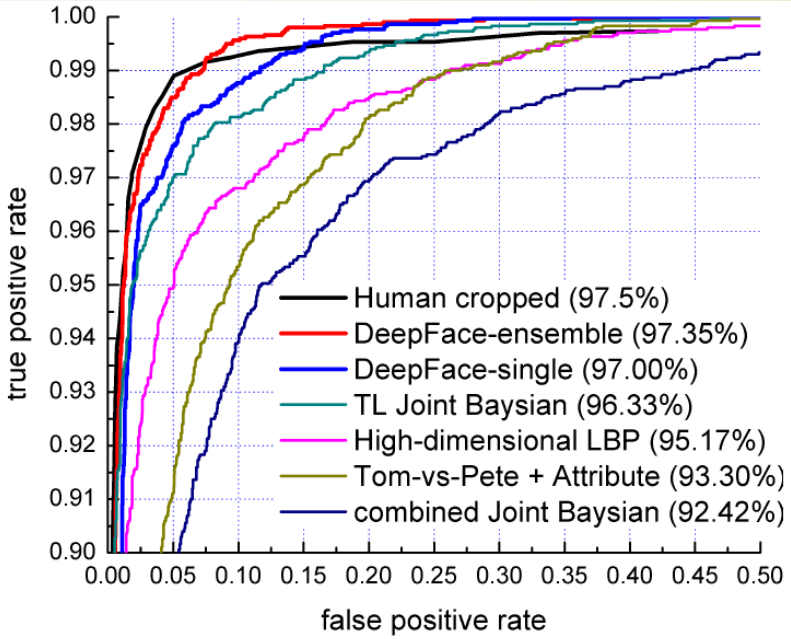


Figure 3. The ROC curves on the *LFW* dataset. Best viewed in color.

Method	Accuracy ± SE	Protocol
Joint Bayesian [6]	0.9242 ±0.0108	restricted
Tom-vs-Pete [4]	0.9330 ±0.0128	restricted
High-dim LBP [7]	0.9517 ±0.0113	restricted
TL Joint Bayesian [5]	0.9633 ±0.0108	restricted
DeepFace-single	**0.9592** ±0.0029	unsupervised
DeepFace-single	**0.9700** ±0.0028	restricted
DeepFace-ensemble	**0.9715** ±0.0027	restricted
DeepFace-ensemble	**0.9735** ±0.0025	unrestricted
Human, cropped	0.9753	

Table 3. Comparison with the state-of-the-art on the *LFW* dataset.

TECHNIQUES IN NEURAL NETWORKS

Many Techniques in Neural Networks

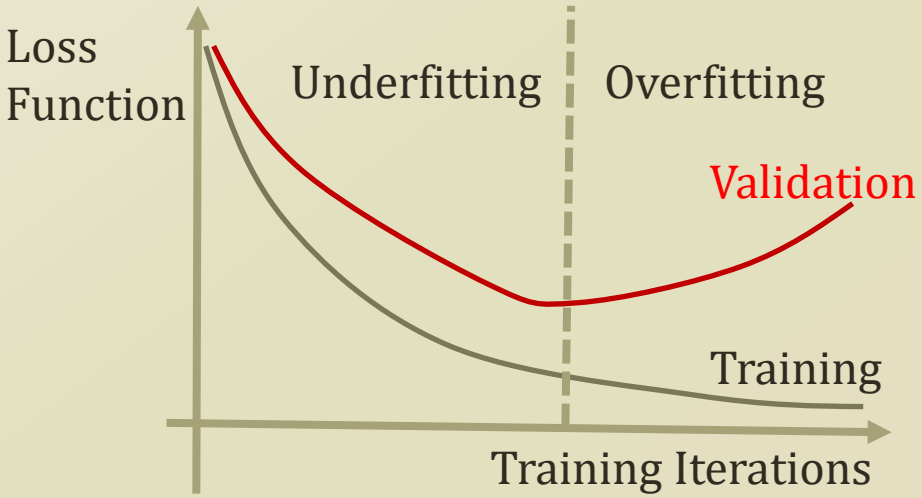
- Neural network requires diverse techniques
  - Training process to facilitate the learning
    - Batch normalization
    - Early stopping
    - Adversarial training
    - Momentum in optimization, adaptive learning rate, gradient checking …
  - Structure to facilitate the learning
    - Attention network
    - Highway network
    - Dropout….
  - Some are originated from experiments, and others are motivated by theory
    - A few found to be useful by experiments, and later, they are revealed to be meaningful, theoretically
- Inference is limited to the stochastic gradient descent

Model Selection

- Training processes
  - Given a dataset, D
  - Divide D into three datasets
    - Training set : D^{train}
      - To infer the parameter of the model to train
    - Validation set : D^{valid}
      - To search the hyper-parameter
        - Hidden layer size, learning rate, number of iterations
    - Test set : D^{test}
      - Actual testing
- In the real world,
  - Train the model with the training and the validation set
    - Deployment domain requires adaptations
      - Training : development phase
      - Validation : deployment phase
  - Test the model with the testing set
    - Evaluate the actual performance after the deployment

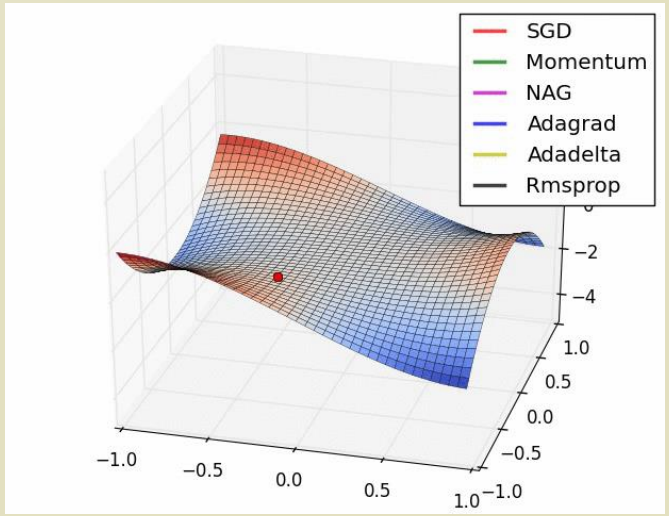
Early Stopping

- Neural network model is naturally complex with many parameters
  - Very easy to overfit
  - Stop the fitting before the model overfits
- Training : the model parameter inference
- Validation : early stopping point check



Momentum and Adaptive Learning Rate

- Gradient Descent : $\theta_{t+1} = \theta_t - \eta \nabla_\theta l\left(f\left(x^{(t)}\right), y^{(t)}; \theta_t\right)$
- Momentum
  - Exponential average of the previous gradients
    - Move along the direction to overcome the plateau
    - β becomes the decay factor
    - $\theta_{t+1} = \theta_t - \eta \hat{\nabla}_\theta^{(t)}$
    - $\hat{\nabla}_\theta^{(t)} = \nabla_\theta l\left(f\left(x^{(t)}\right), y^{(t)}; \theta_t\right) + \beta \hat{\nabla}_\theta^{(t-1)}$
- Adaptive Learning Rate
  - Adagrad : learning rate scaled by the square root of the cumulative sum of squared gradients
    - More movements on the smaller movements, less movements on the larger movements
    - $\theta_{t+1} = \theta_t - \frac{\eta}{\sqrt{G_t + \epsilon}} \nabla_\theta l\left(f\left(x^{(t)}\right), y^{(t)}; \theta_t\right)$
    - $G_t = G_{t-1} + \left(\nabla_\theta l\left(f\left(x^{(t)}\right), y^{(t)}; \theta_t\right)\right)^2$
  - RMSProp : Exponential moving average of the squared gradients
    - $\theta_{t+1} = \theta_t - \frac{\eta}{\sqrt{G_t + \epsilon}} \nabla_\theta l\left(f\left(x^{(t)}\right), y^{(t)}; \theta_t\right)$
    - $G_t = \gamma G_{t-1} + (1-\gamma)\left(\nabla_\theta l\left(f\left(x^{(t)}\right), y^{(t)}; \theta_t\right)\right)^2$
  - Adam : RMSProp + Momentum
    - $\theta_{t+1} = \theta_t - \frac{\eta}{\sqrt{G_t + \epsilon}} \hat{\nabla}_\theta^{(t)}$
    - $\hat{\nabla}_\theta^{(t)} = (1-\beta_1)\nabla_\theta l\left(f\left(x^{(t)}\right), y^{(t)}; \theta_t\right) + \beta_1 \hat{\nabla}_\theta^{(t-1)}$
    - $G_t = \gamma G_{t-1} + (1-\gamma)\left(\nabla_\theta l\left(f\left(x^{(t)}\right), y^{(t)}; \theta_t\right)\right)^2$



Mini-Batch and Gradient Checking

- Backpropagation algorithm
  - Single instance gradient calculation
    - $w \leftarrow w - \eta \nabla_{\boldsymbol{w}} E\left[l\left(f\left(\mathbf{x}^{(i)}; \boldsymbol{w}\right), y^{(i)}\right)\right]$
    - With a single instance, not much to calculate as expectation
  - Full instance gradient calculation
    - $w \leftarrow w - \eta \nabla_{\boldsymbol{w}} E[l(f(\boldsymbol{x}; \boldsymbol{w}), \boldsymbol{y})]$
    - Normalize the summation with the size of the dataset
  - Should be mini-batch
    - $w \leftarrow w - \eta \nabla_{\boldsymbol{w}} E\left[l\left(f\left(\mathbf{x}^{(i;i+n)}; \boldsymbol{w}\right), y^{(i;i+n)}\right)\right]$
    - Normalize the summation with the size of the mini-batch
- Gradient checking
  - When you implement the propagation algorithm, compare the finite-difference approximation of the gradient
    - $\frac{\partial f(x)}{\partial x} \approx \frac{f(x+\epsilon) - f(x-\epsilon)}{2\epsilon}$

Batch Normalization

- Layer type output normalization with mini-batch data
  - BatchNormalization$(x_{1\ldots m}; \gamma, \beta)$
    - $\mu_B \leftarrow \frac{1}{m}\sum_{i=1}^{m} x_i$
    - $\sigma_B^2 \leftarrow \frac{1}{m}\sum_{i=1}^{m}(x_i - \mu_B)^2$
    - $\hat{x}_i \leftarrow \frac{x_i - \mu_B}{\sqrt{\sigma_B^2 + \epsilon}}$
    - $y_i \leftarrow \gamma \hat{x}_i + \beta \equiv BN_{\gamma,\beta}(x_i)$
- Batch normalization parameter, γ and β, is trainable through stochastic gradient descent, as well
  - Can use chain rule
- How to use it?
  - Training : After the regression, and before the activation
  - Testing : remember the mean of γs and βs from mini-batches
- Why use it?
  - Scale and shift normalization

Adversarial Training

- Instances in datasets
  - Some instances work well with the trained model
  - Some instances do not work well with the trained model
    - Some instances should not be accommodated because it could be noise
    - Some instances should be accommodated because it is a meaningful case
- Find an (or artificially generate) instance that disrupt the current trained model
  - "Training on adversarial example"
  - $w^T \tilde{x} = w^T x + w^T \eta, ||\eta||_\infty < \epsilon$
    - $\tilde{x}$: adversarial example, $w^T \eta$: adversarial perturbation
    - $\eta^* = max_\eta w^T \eta = sign(w)$
    - Can make many infinitesimal changes, "Accidental steganography"
- Given $y \in \{-1,1\}, P(y = 1) = \sigma(w^T x + b), g(z) = \log(1 + \exp(z))$
  - Training objective
    - $min_w E_{x,y \sim P_{data}} g(-y(w^T x + b))$
  - Adding the adversarial training
    - $min_w E_{x,y \sim P_{data}} g(-y(w^T x + b - \epsilon ||w||_1))$
      - $-\left|\left|w\right|\right|_1 = -w^T sign(w)$, because of the minimization objective
  - Similar to the L1 regularization, but it is different
    - Not directly applied as a penalty that does not go away.
    - Good fitting of *w* can eliminate the adversarial learning term
    - L1 regularization → controlling the sensitivity
    - Adversarial training → robust weight vector

Dropout

- Challenge of the neural network
  - Complex model to learn a simple principle
  - Limit the complexity, stochastically
    - Remove a set of randomly selected neurons for a certain mini-batch
- Rational
  - Hidden units cannot co-adapt to other units that were active in the mini-batch gradient descent
  - Hidden units must be more generally useful
- By defining a mask, $r^{(l)}, l = 1 \dots L$
  - Neural network without dropout
    - $z_i^{(l+1)} = w_i^{(l+1)} y^{(l)} + b_i^{(l+1)}, y_i^{(l+1)} = f(z_i^{(l+1)})$
  - Neural network with dropout
    - $r_j^{(l)} \sim Bernoulli(p)$
    - $\tilde{y}^{(l)} = r^{(l)} * y^{(l)}$
    - $z_i^{(l+1)} = w_i^{(l+1)} \tilde{y}^{(l)} + b_i^{(l+1)}, y_i^{(l+1)} = f(z_i^{(l+1)})$
- Marginalizing the dropout in the linear regression
  - $min_w E_{R\sim Bernoulli(p)} \left[\left|\left| y - (R * X)w \right|\right|^2 \right] = min_w \left[\left|\left| y - pXw \right|\right|^2 + p(1-p) \left|\left| \Gamma w \right|\right|^2 \right]$
    - $\Gamma = (diag(X^T X))^{1/2}$
  - $= min_w \left[\left|\left| y - X\tilde{w} \right|\right|^2 + \frac{1-p}{p} \left|\left| \Gamma \tilde{w} \right|\right|^2 \right]$
    - $\tilde{w} = wp$