HISTOGRAM OF GRADIENTS (HOG)

Edges

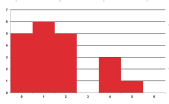
HOG: Human Detection

Navneet Dalal and Bill Triggs "Histograms of Oriented Gradients for Human Detection" CVPR05

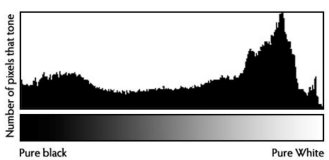
Revist Histogram

0	1	1	2	4
2	1	0	0	2
5	2	0	0	4
1	1	2	4	1

image



histogram

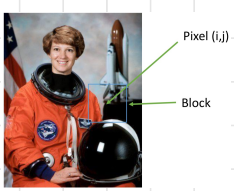


HOG: Oriented Gradients

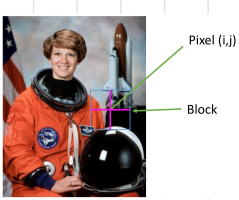
0) Given image I, Pixel location (i,j)
Compute the HOG feature for that pixel

steps

1. extract a block of some size



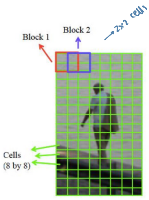
2. Divide block into sq grid of sub blocks (cells)



2x2 grid
so 4 cells

3. Compute orientation histogram of each cell
   - use sobel filter: $G_x = \begin{bmatrix} -1 & 0 & 1 \\ -2 & 0 & 2 \\ -1 & 0 & 1 \end{bmatrix}$, $G_y = \begin{bmatrix} 1 & 2 & 1 \\ 0 & 0 & 0 \\ -1 & -2 & -1 \end{bmatrix}$
   - $\theta = \tan^{-1}\left(\frac{G_y}{G_x}\right)$ → 9 bins 0°–180°
4. Concatenate the 4 histograms
   V = vector of concatonation of 4 histograms
5. Normalize V
   - There are 3 ways to normalize
     1. Divide v by its Euclidean norm
     2. Divide v by its L_1 norm (sum of all absolute values of v)
     3. Divide v by its Euclidean norm
        - then clip any value over 0.2 in the resulting vector
        - Then renormalize it by dividing again by its Euclidean

Histogram of Oriented Gradients



- Each block consists of 2x2 cells with size 8x8
- Quantize the gradient orientation into 9 bins (0-180)
  - The vote is the gradient magnitude

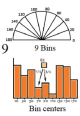
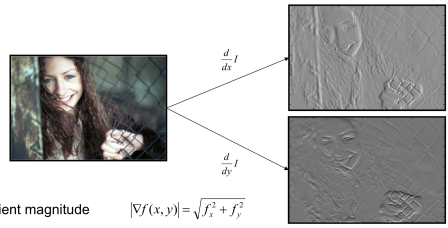


Image gradients



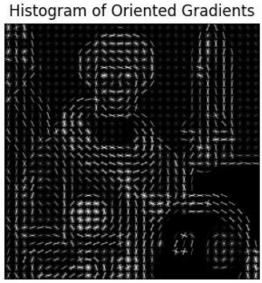
Gradient magnitude $|\nabla f(x,y)| = \sqrt{f_x^2 + f_y^2}$

Gradient direction $\theta = \tan^{-1}\frac{f_x}{f_y}$

HOG

Input image

Histogram of Oriented Gradients



Histograms of Oriented Gradients

- Parameters and design options:
  - Angles range from 0 to 180 or from 0 to 360 degrees?
    - In the Dalal & Triggs paper, a range of 0 to 180 degrees is used,
    - and HOGs are used for detection of pedestrians.
  - Number of orientation bins.
    - Usually 9 bins, each bin covering 20 degrees.
  - Cell size.
    - Cells of size 8x8 pixels are often used.
  - Block size.
    - Blocks of size 2x2 cells (16x16 pixels) are often used.
- Usually a HOG feature has 36 dimensions.
  - 4 cells * 9 orientation bins.

SCALE INVARIANT FEATURE TRANSFORM (SIFT)

SIFT

↳ Image content is transformed into local features coordinates

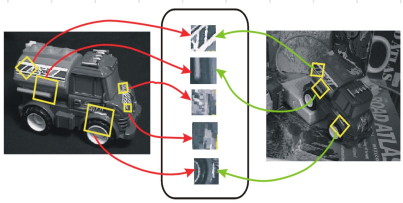
↳ Invariant local features

never changing

Invariant local Features

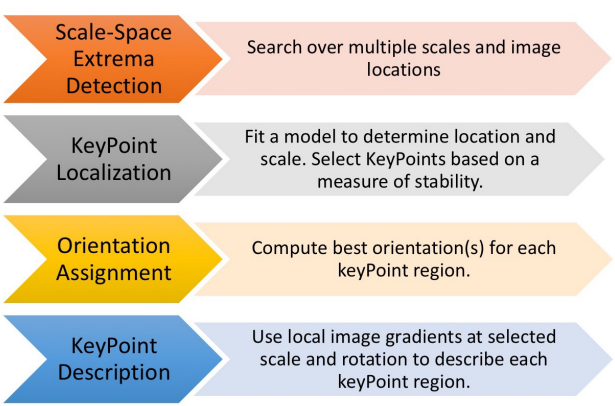
A good feature should be invariant to transformations:

Translation – Interest points should move consistently.
Rotation – Descriptors should stay the same even if the image rotates.
Scaling – Features should be detectable at multiple scales.
Lighting Changes – Interest points should be robust to brightness changes.

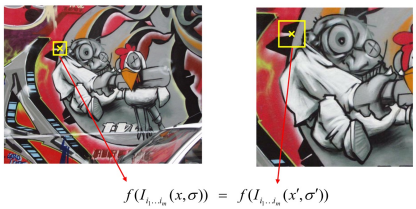


Features Descriptors

Overall Procedure at a High Level



Automatic scale selection

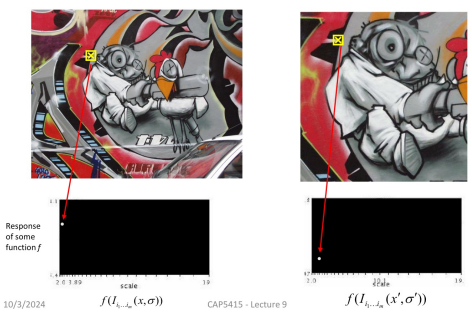


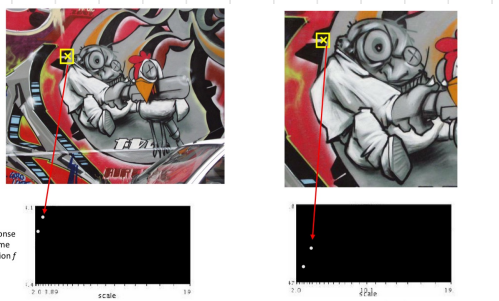
$$f(I_{i_1\dots i_m}(x,\sigma)) = f(I_{i_1\dots i_m}(x',\sigma'))$$

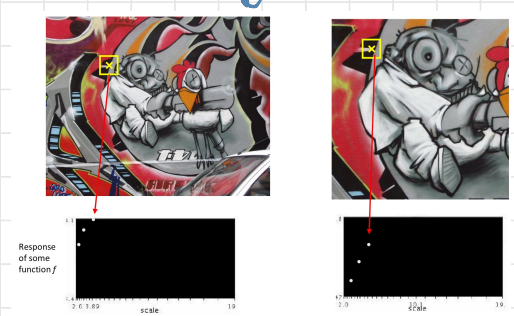
How to find patch sizes at which f response is equal?

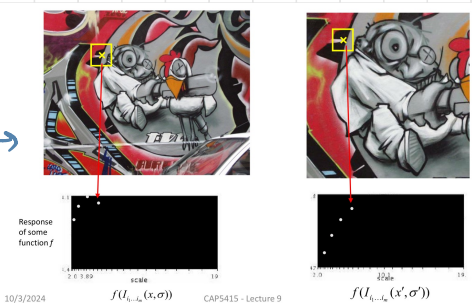
What is a good f ?

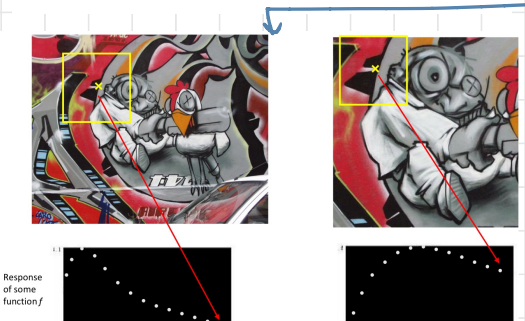
- Function responses for increasing scale (scale signature)

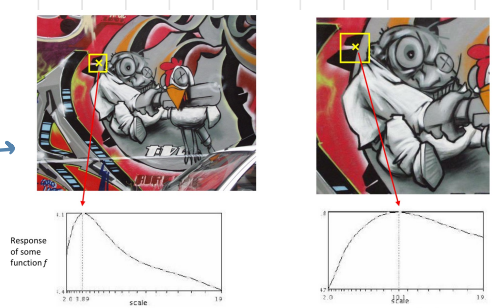




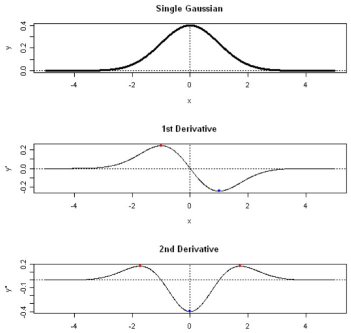




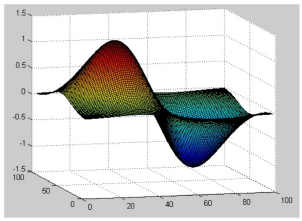




What Is A Useful Signature Function *f* ?



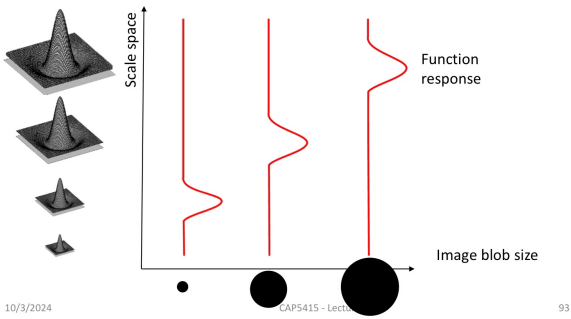
1st Derivative of Gaussian



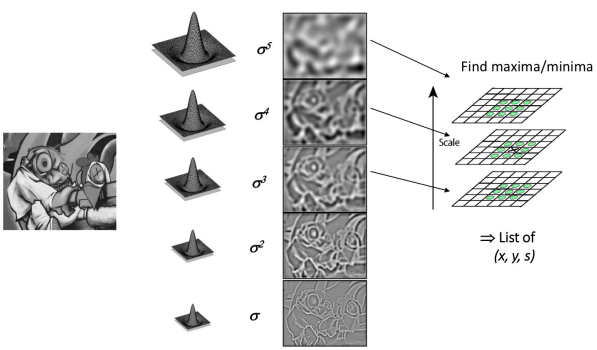
(Laplacian of Gaussian)

"Blob" detector is common for corners

- Laplacian (2nd derivative) of Gaussian (LoG)

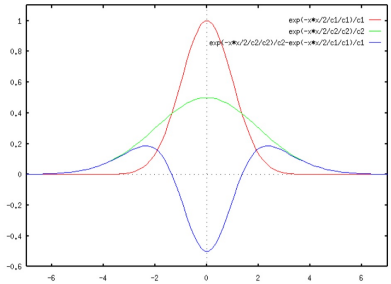


Find local maxima in position-scale space



Alternative kernel

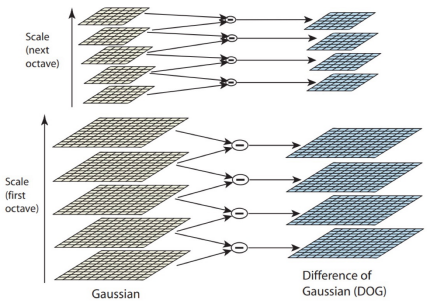
Approximate LoG with Difference-of-Gaussian (DoG).



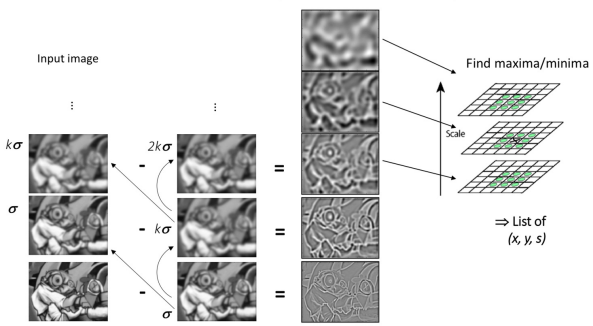
Approximate LoG with Difference-of-Gaussian (DoG).

1. Blur image with σ Gaussian kernel
2. Blur image with $k\sigma$ Gaussian kernel
3. Subtract 2. from 1.

Scale-space

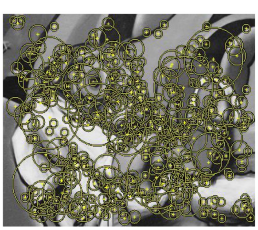


Find local maxima in position-scale space of DoG



Results: Difference-of-Gaussian

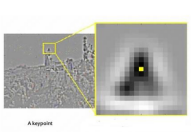
- Larger circles = larger scale
- Descriptors with maximal scale response

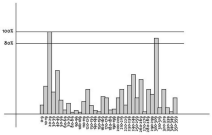


SIFT Orientation Estimation

↳ Compute gradient orientation histogram

↳ Select dominant orientation θ



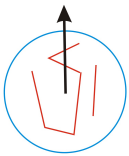


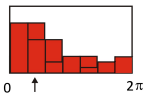
SIFT Orientation Normalization

↳ Compute gradient orientation histogram

↳ Select dominant orientation θ

↳ Normalize: rotate to fixed orientation



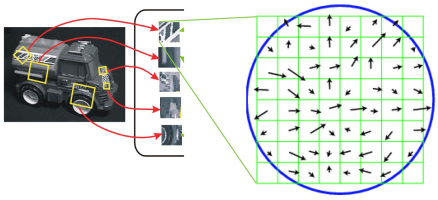


SIFT Descriptor formation

↳ Compute on local 16x16 window around detection

↳ Rotate and scale window according to discovered orientation θ and scale σ

↳ Compute gradients weighted by a Gaussian of variance half the window → for smooth fallout



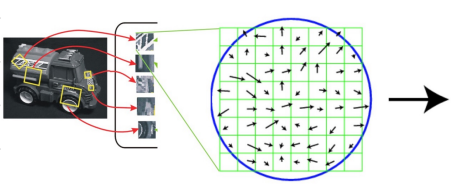
Actually 16x16, only showing 8x8

Image gradients

↳ 4x4 array of gradient orientation histogram weighted by gradient magnitude

↳ Bin into 8 orientations

8 x 4x4 array = 128 dimensions

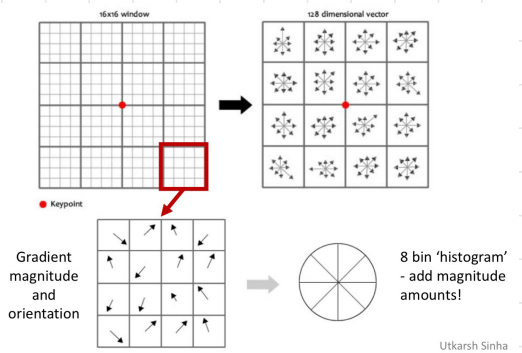


Showing only 2x2 here but is 4x4

Image gradients

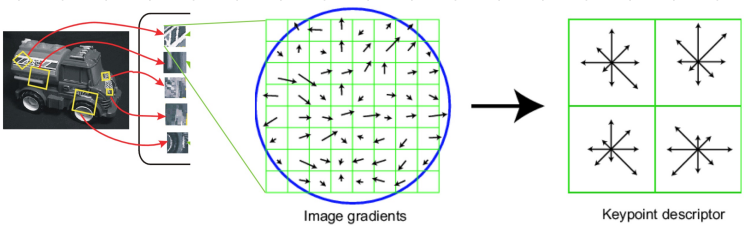
Keypoint descriptor

SIFT Descriptor Extraction



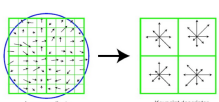
Reduce effect of illumination

- 128-dim vector normalize to 1
- Threshold gradient magnitudes
  - to avoid excessive influence of high gradients
  - after normalization, clamp gradients > 0.2
  - Renormalize



Review: Local Descriptors

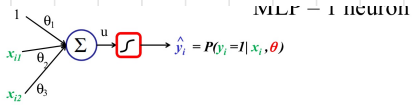
- Most features can be thought of as
  - templates,
  - histograms (counts),
  - or combinations
- The ideal descriptor should be
  - Robust and Distinctive
  - Compact and Efficient
- Most available descriptors focus on edge/gradient information
  - Capture texture information
  - Color rarely used



ishma hafeez notes

MULT LAYER PERCEPTRON

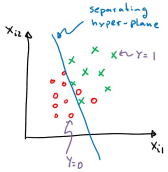
1. MLP - 1 layer 1 neuron



We are given the data $\{x_i, y_i\}_{i=1}^{n}$

eg:

	x_{i1}	x_{i2}	y_i
i=1	0.2	6	0
i=2	0.3	22	1
i=3	0.6	-0.6	1
i=4	-0.4	58	0
⋮			



$0 < \hat{y}_i < 1$

$u = \theta_1 + \theta_2 x_{i1} + \theta_3 x_{i2}$

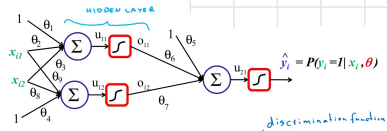
$$\hat{y}_i = \frac{1}{1+e^{-u}} = \frac{1}{1+e^{-\theta_1 - \theta_2 x_{i1} - \theta_3 x_{i2}}} = P(y_i = 1 | x_i, \theta)$$

$$P(y_i | x_i, \theta) = \hat{y}_i^{y_i} (1-\hat{y}_i)^{1-y_i} = \begin{cases} \hat{y}_i & \text{when } y_i = 1 \\ 1-\hat{y}_i & \text{otherwise} \end{cases}$$

For n independent observations (Bernoulli:)

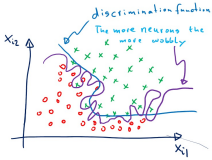
$$P(y | x, \theta) = \prod_{i=1}^{n} P(y_i | x_i, \theta)$$

2. MLP - 2 layers 3 neurons



Data:

	x_{i1}	x_{i2}	y_i
i=1	6	7	1
i=2	0.2	-5	0
i=3	-100	3.1	1
i=4	6	9	0
i=5	5	8	0



$u_{11} = \theta_1 + \theta_2 x_{i1} + \theta_3 x_{i2}$

$u_{12} = \theta_4 + \theta_5 x_{i1} + \theta_6 x_{i2}$

$o_{11} = \frac{1}{1+e^{-u_{11}}}$

$o_{12} = \frac{1}{1+e^{-u_{12}}}$

$\hat{y}_i = \frac{1}{1+e^{-u_{21}}}$

$u_{21} = \theta_7 + \theta_8 o_{11} + \theta_9 o_{12}$

$$P(y_i | x_i, \theta) = \hat{y}_i^{y_i} (1-\hat{y}_i)^{1-y_i}$$

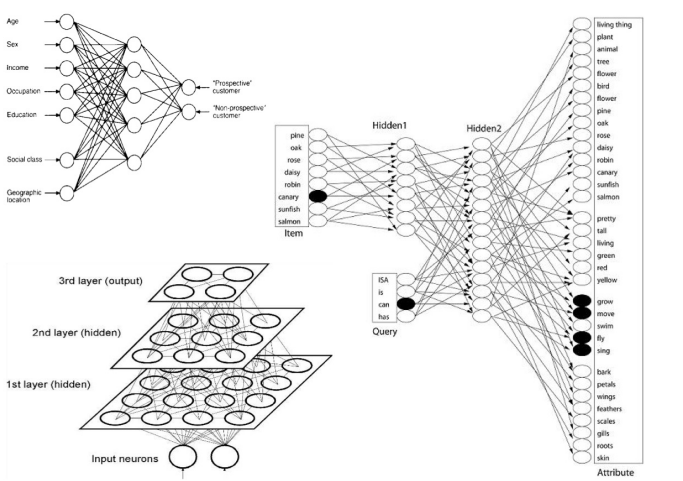
For n independent observations.

$$P(y | x, \theta) = \prod_{i=1}^{n} \hat{y}_i^{y_i} (1-\hat{y}_i)^{1-y_i} = \prod_{i=1}^{n} P(y_i | x_i, \theta)$$

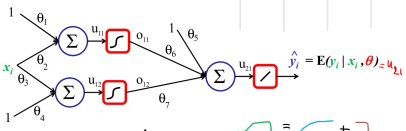
Cost:

$$C(\theta) = -\log P(y | x, \theta) = -\sum_{i=1}^{n} y_i \log \hat{y}_i + (1-y_i) \log (1-\hat{y}_i)$$

i.e minimize the cross-entropy error.

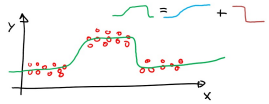


3. MLP - Regression

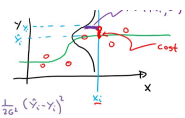


Data :

x_i	Y_i
0.2	0.6
0.9	0.4
-0.6	5
-0.3	6.2



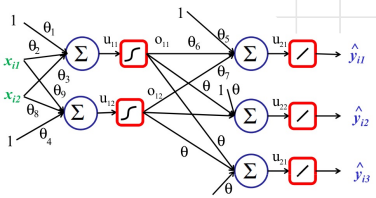
$\hat{Y}_i = \theta_5 + \frac{\theta_6}{1+e^{-\theta_1-\theta_2 x_i}} + \frac{\theta_7}{1+e^{-\theta_4-\theta_3 x_i}}$



$P(Y_i|x_i,\theta) = (2\pi\sigma^2)^{-1/2} e^{-\frac{1}{2\sigma^2}(\hat{Y}_i - Y_i)^2}$

$C(\theta) = \sum_{i=1}^{n} (Y_i - \hat{Y}_i)^2$ $\quad \hat{Y}_i = f(\theta, x_i)$

4. MLP - Multiclass



Data :

x_{i1}	x_{i2}	Y_{i1}	Y_{i2}	Y_{i3}	
0.2	0.3	0	1	0	class 2
-5	-6	1	0	0	class 1
-20	4	1	0	0	// 1
42	6.8	0	0	1	class 3

To get a probabilistic model, define: SOFTMAX

$$P(Y_i=(0,1,0)|x_i,\theta) = P(Y_i=2|x_i,\theta) = \frac{e^{\hat{y}_2}}{e^{\hat{y}_1}+e^{\hat{y}_2}+e^{\hat{y}_3}}$$

$\mathbb{I}_2(Y_i) = \begin{cases} 1 & Y_i = 2 \\ 0 & \text{O.W.} \end{cases}$

Then,

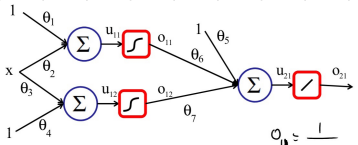
$$P(Y_i|x_i,\theta) = \left[\frac{e^{\hat{y}_{i1}}}{e^{\hat{y}_{i1}}+e^{\hat{y}_{i2}}+e^{\hat{y}_{i3}}}\right]^{\mathbb{I}_1(Y_i)} \left[\frac{e^{\hat{y}_{i2}}}{e^{\hat{y}_{i1}}+e^{\hat{y}_{i2}}+e^{\hat{y}_{i3}}}\right]^{\mathbb{I}_2(Y_i)} \left[\frac{e^{\hat{y}_{i3}}}{e^{\hat{y}_{i1}}+e^{\hat{y}_{i2}}+e^{\hat{y}_{i3}}}\right]^{\mathbb{I}_3(Y_i)}$$

$$= \begin{cases} e^{\hat{y}_{i1}}/\text{sum} & Y_i = 1 \\ e^{\hat{y}_{i2}}/\text{sum} & Y_i = 2 \\ e^{\hat{y}_{i3}}/\text{sum} & Y_i = 3 \end{cases}$$

Cost :

$$C(\theta) = -\log P(Y|X,\theta) = -\sum_{i=1}^{n} \sum_{j=1}^{3} \mathbb{I}_j(Y_i) \log \frac{e^{\hat{y}_{ij}}}{\text{sum}}$$

Backpropagation



$$\hat{Y} = o_{21} = u_{21} = \theta_5 + \theta_6 o_{11} + \theta_7 o_{12}$$

$$o_{11} = \frac{1}{1+e^{-u_{11}}}$$

$$o_{12} = \frac{1}{1+e^{-u_{12}}}$$

$$u_{11} = \theta_1 + \theta_2 x$$

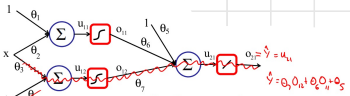
$$u_{12} = \theta_4 + \theta_3 x$$

$$\hat{Y} = o_{21} = \theta_5 + \theta_6 o_{11} + \theta_7 o_{12}$$

$$E(\theta) = \left(Y_i - \hat{Y}_i(x_i,\theta)\right)^2$$

$$\frac{\partial E(\theta)}{\partial \theta_j} = -2\left(Y_i - \hat{Y}_i(x_i,\theta)\right)\frac{\partial \hat{Y}_i(x_i,\theta)}{\partial \theta_j}$$

$$\frac{\partial \hat{Y}_i}{\partial \theta_5} = 1 \qquad \frac{\partial \hat{Y}_i}{\partial \theta_6} = o_{11} \qquad \frac{\partial \hat{Y}_i}{\partial \theta_7} = o_{12}$$



$$\hat{Y} = \theta_5 o_{11} + \theta_6 o_{11} + \theta_7$$

$$u_{12} = \theta_4 x + \theta_3 1$$

$$\frac{\partial \hat{Y}}{\partial \theta_3} = \frac{\partial \hat{Y}}{\partial o_{12}} \frac{\partial o_{12}}{\partial u_{12}} \frac{\partial u_{12}}{\partial \theta_3}$$

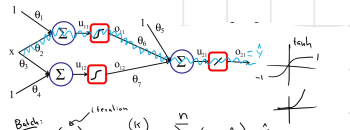
$$= \theta_7 o_{12}[1-o_{12}] x$$

Tanh

$$\frac{\partial o}{\partial u_{12}} = o_{12}(1-o_{12})$$

i.e.

$$\frac{\partial}{\partial u}\frac{1}{1+e^{-u}} = \left(\frac{1}{1+e^{-u}}\right)\left(1-\frac{1}{1+e^{-u}}\right)$$



Batch:

$$\theta_j^{(k+1)} = \theta_j^{(k)} + \sum_{i=1}^{n} (Y_i - \hat{Y}_i)\frac{\partial \hat{Y}_i}{\partial \theta_j}$$

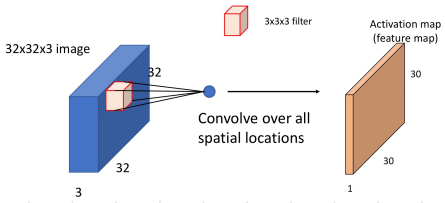
(k+1) iteration

Online:

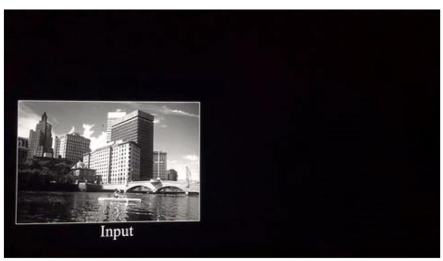
$$\theta_2^{i} = \theta_2^{i-1} + \eta (Y_i - \hat{Y}_i)\theta_6^{i-1}(o_{1i}(1-o_{1i}))x_i$$

TRAINING NEURAL NETWORKS

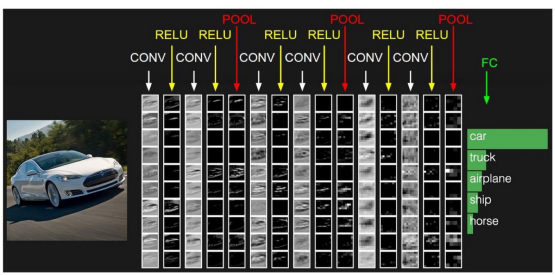
Network Parameters - Recap



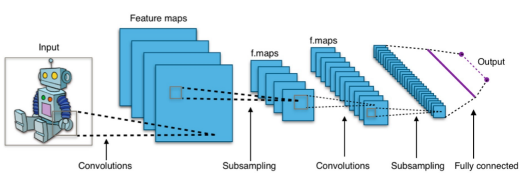
Convolution - Intuition



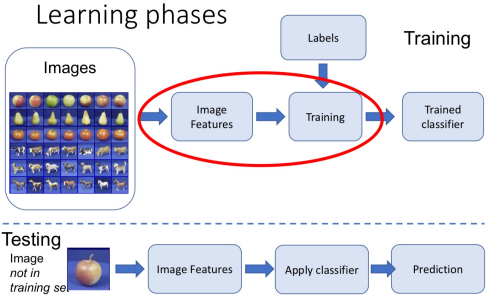
Visualizing CNN



General CNN architecture - recap



Learning phases



Loss Function

↳ tells how good the network performs in terms of prediction

↳ Network training (Optimization)

↳ find the best network parameters → to minimize the loss

$$L(W) = \frac{1}{N}\sum_{i=1}^{N} L_i(f(x_i, W), y_i)$$

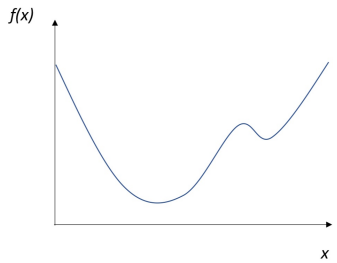
input, Ground truth, Loss function, Network parameters, network

Loss function measures how 'good' our network is at classifying the training examples wrt. the parameters of the model (the perceptron weights).

Network Training → Minimize cost

↳ Gradient Descent

↳ way to minimize a cost function



Loss Function Types

1. Cross Entropy → for multiclass classification

↳ measures diff b/w 2 probability distributions

True Distribution (actual labels), Predicted Distribution

$$\mathcal{L} = -\frac{1}{N}\sum_{i=1}^{N}\left[y_i \log(\hat{y}_i) + (1-y_i)\log(1-\hat{y}_i)\right]$$

y_i: actual, $\hat{y}_i$: predicted

2. Mean Squared Error (MSE)

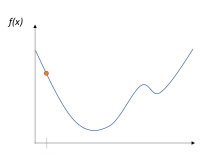
↳ used in regression tasks

↳ measures the avg of squared diff b/w actual and predicted values

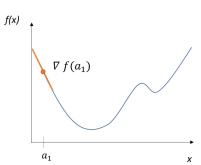
$$\mathcal{L} = \frac{1}{N}\sum_{i=1}^{N}(y_i - \hat{y}_i)^2$$

General Approach

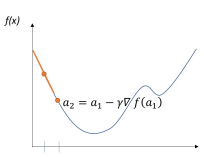
Pick random starting point.



Compute gradient at point (analytically or by finite differences)

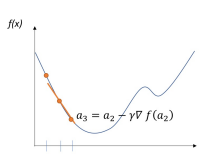


Move along parameter space in direction of negative gradient



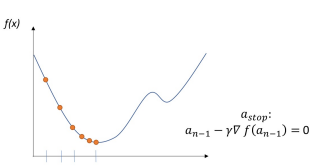
γ = amount to move = *learning rate*

Move along parameter space in direction of negative gradient.



γ = amount to move = *learning rate*

Stop when we don't move any more.

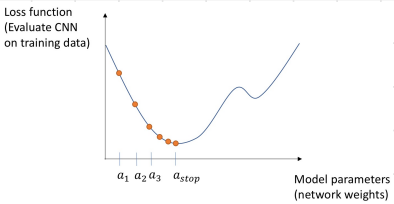


Gradient Descent

- Optimizer for functions
- Guaranteed to find optimum for convex functions
  - non-convex ⟶ find local optimum
  - most vision problems aren't convex
- Works for multi-variate functions
  - need to compute matrix of partial derivatives

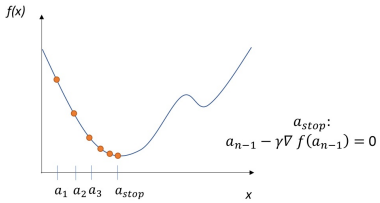
Train CNN with Gradient Descent

- x^i, y^i = n training examples
- $f(x)$ = feed forward network
- L(x, y; θ) = some *loss function*



General approach - recap

Stop when we don't move any more.

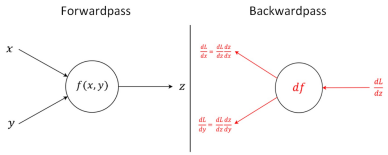


Differentiability

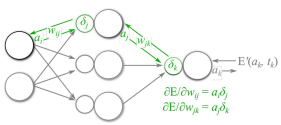
- Loss function
- Activation function
- Convolution
- Pooling
- ...

Backpropagation - Chain Rule

- Chain rule $\frac{\partial L}{\partial x} = \frac{\partial L}{\partial z}\frac{\partial z}{\partial x}$



- Add gate:
  - gradient distributor
- Max gate:
  - gradient router
- Mul gate:
  - gradient switcher

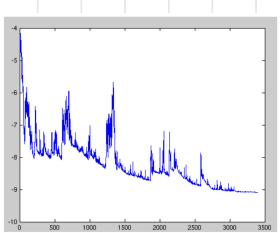


Stochastic Gradient Descent

↳ Dataset can be too large
  ↳ can not apply gradient descent w.r.t all data points
↳ Randomly sample a data point
  ↳ Perform gradient descent per sample and iterate
  ↳ Picking a subset of points: "mini batch" → batchsize

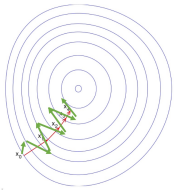
↳ Randomly initialize starting W and Pick learning rate γ
  While not at minimum:
    ↳ Shuffle training set
    ↳ for each data point i=1....n → maybe as mini-batch } → Epoch
      ↳ Gradient Descent

↳ loss will not always decrease (locally)
↳ as training data point is random
↳ still converges over time



Gradient Descent Oscillations

Slow to converge to the (local) optimum



Momentum

↳ Adjust the gradient by a weighted sum of the prev amount plus the current amount.

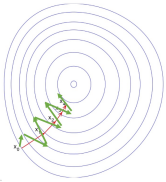
↳ without momentum: $\theta_{t+1} = \theta_t - \gamma \frac{dL}{d\theta}$

↳ with momentum (new α parameter)

$\theta_{t+1} = \theta_t - \gamma(\alpha \left[\frac{dL}{d\theta}\right]_{t-1} + \left[\frac{dL}{d\theta}\right]_t)$

Lowering the learning rate

-Takes longer to get to the optimum



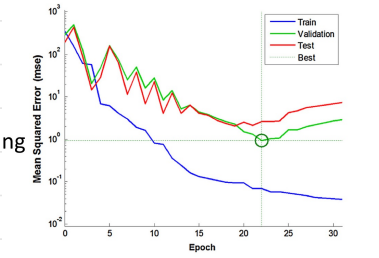
Problem of fitting

↳ Too many parameters = overfitting

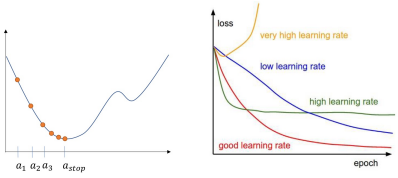
↳ Not enough parameters = underfitting

↳ more data = less chance to overfit

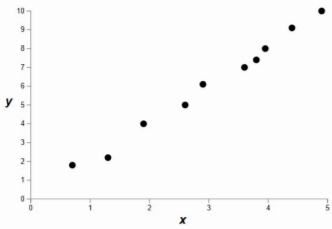
↳ How do we know what is required



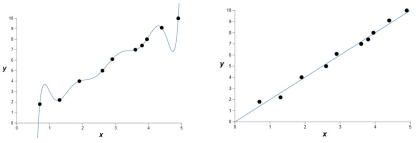
Learning rate



Data fitting problem



Which is better?

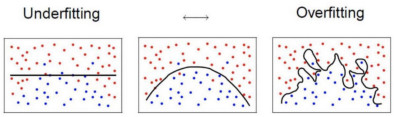


9th order polynomial

1st order polynomial

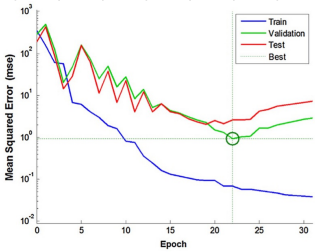
[Nielson]

Data fitting problem



- Early stopping
- Regularization
- Dropout
- ...

1. Early stopping



2. Regularization

↳ Attempt to guide solution to not overfit
↳ But still give freedom with many parameters
↳ so basically

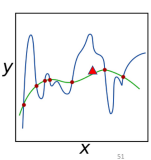
Penalise the use of parameters
to prefer small weights

↳ Add a cost to having high weights

$$L(w) = \frac{1}{N}\sum_{i=1}^{N} L_i(f(x_i, w), y_i) + \lambda R(w)$$

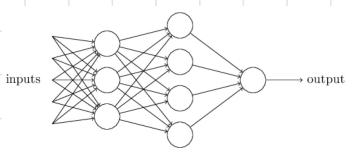
(Loss; Weight Decay; Regularization)

- In common use,
  L1 Norm – $R(W) = \sum_i \sum_j |Wij|$
  L2 Norm – $R(W) = \sum_i \sum_j W^2_{ij}$
  Elastic net – $R(W) = \sum_i \sum_j \beta W^2_{ij} + |Wij|$



Regularization: Ensemble

↳ Networks start with random weights
at every train = slightly different outcome
↳ Random weights are used cuz
if weights are all equal,
response across filters will
be equivalent
↳ so network doesnt train

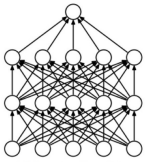


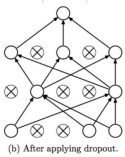
$$w \cdot x \equiv \sum_j w_j x_j$$

Why not train 5 different networks with random starts and vote on their outcome?
- Works fine!
- Helps generalization because error due to overfitting is averaged; reduces variance.

3. Dropout

↳ Stochastically switch neurons off
↳ each neuron is set to 0 → with probability p
↳ hidden units cannot co-adapt
to each other
↳ Units are useful independently
↳ Hyperparameter
↳ p is usually set to 0.5


(a) Standard Neural Net

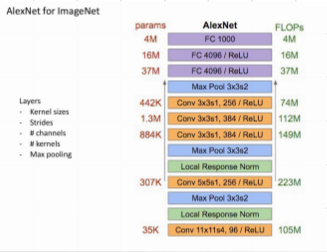

(b) After applying dropout.

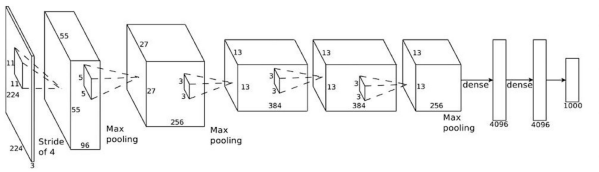
Training steps

- Define network
- Loss function
- Initialize network parameters
- Get training data
    - Prepare batches
- Feedforward one batch
    - Compute loss
    - Backpropagate gradients
    - Update network parameters
    - Repeat

AlexNet - Training

Parameters:
- First use of ReLU
- Dropout 0.5
- Batch size 128
- Optimizer SGD
- Momentum 0.9
- Learning rate 1e-2
- Decay – lr reduced by 10 manually when val accuracy plateaus
- L2 weight decay 5e-4





Conv 1	Conv 2	Conv 3	Conv 4	Conv 4
11 x 11 x 3	5 x 5 x 48	3 x 3 x 256	3 x 3 x 192	3 x 3 x 192
Stride 4	Stride 1	Stride 1	Stride 1	Stride 1
96 filters	256 filters	384 filters	384 filters	256 filters

Residual Networks

- Deep networks performs worse
    - As we add more layers
- Problem
    - Vanishing gradients
- It models
    - H(x) = F(x) + x
- Skip connections
    - Help in backpropagation

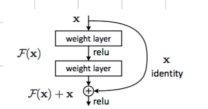
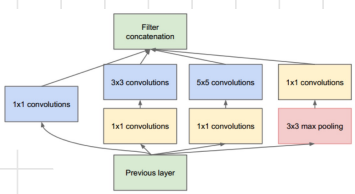


Figure 2. Residual learning: a building block.

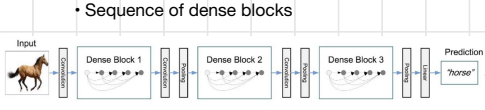
GoogleNet - Inception

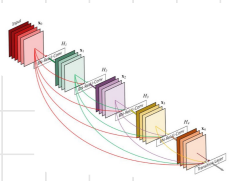
- ResNet is about going deeper
- Inception is about going wider
- Focused on computational efficiency
- The network learns
    - Which features are useful



DenseNet

- Densely Connected Convolutional Network
    - Similar to ResNet
    - Concat features instead of summation
    - A layer is passed all previous maps
    - Sequence of dense blocks





Huang, Gao, et al. "Densely connected convolutional networks." (2016).

CNN AND LEARNING RATE

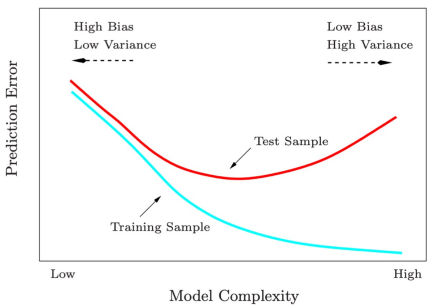
Bias

error caused because the model lacks the ability to represent the (complex) concept

Variance

error caused because the learning algorithm overreacts to small changes (noise) in the training data

Total Loss = Bias + Variance (+ noise)



Universality Theorem

↳ Any continuous function f

$f: R^N \rightarrow R^M$

can be realized by a network

with one hidden layer → given enough hidden neurons

Universality is Not Enough

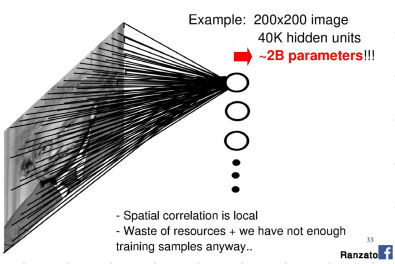
- Neural network has very high capacity (millions of parameters)
- By our basic knowledge of bias-variance tradeoff , so many parameters should imply very low bias, and very high variance. The test loss may not be small.
- Many efforts of deep learning are about mitigating overfitting!

Address Overfitting for NN

- Use larger training data set
- Design better network architecture

convolutional NN (CNN)

Images as input to neural networks



CNN = a multi-layer neural network with

- Local connectivity:
  - Neurons in a layer are only connected to a small region of the layer before it
- Share weight parameters across spatial positions:
  - Learning shift-invariant filter kernels

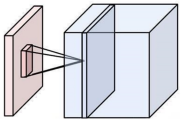
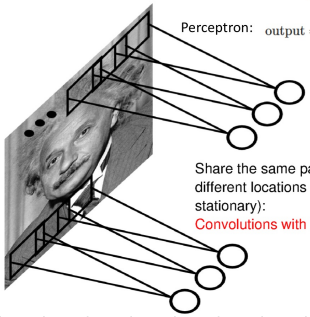


Image credit: A. Karpathy

Convolutional Layer



Perceptron: $\text{output} = \begin{cases} 0 & \text{if } w \cdot x + b \leq 0 \\ 1 & \text{if } w \cdot x + b > 0 \end{cases}$

$w \cdot x \equiv \sum_j w_j x_j$

This is convolution!

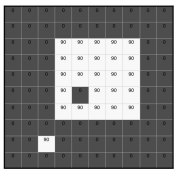
Share the same parameters across different locations (assuming input is stationary):
Convolutions with learned kernels

Ranzato

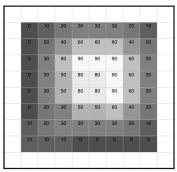
Recap: Image filtering

$f[\cdot,\cdot]$:

$I[.,.]$



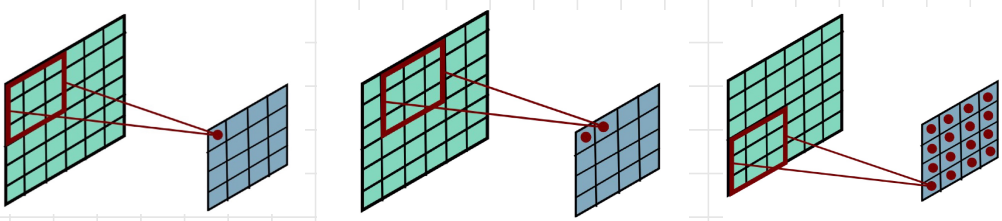
$h[.,.]$



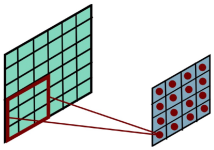
$$h[m,n] = \sum_{k,l} f[k,l]\, I[m+k, n+l]$$

Credit: S. Seitz

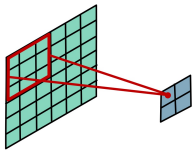
Convolutional Layer



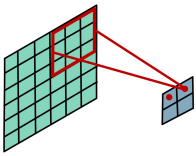
Stride = 3



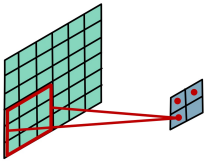
Stride = 3



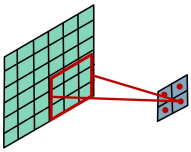
Stride = 3



Stride = 3

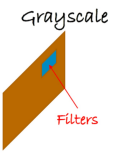


Stride = 3



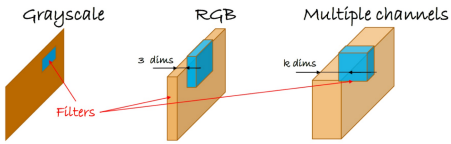
2D spatial features

- If images are 2-D, parameters should also be organized in 2-D
  - That way they can learn the local correlations between input variables
  - That way they can "exploit" the spatial nature of images

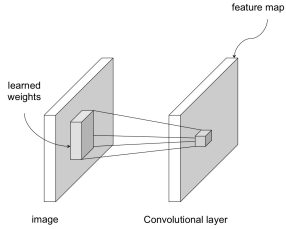


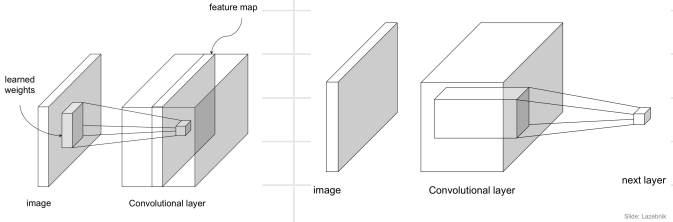
K-D spatial features

- Similarly, if images are k-D, parameters should also be k-D

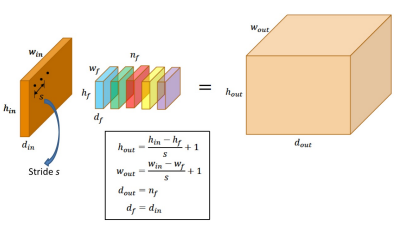


Dimensions of convolution

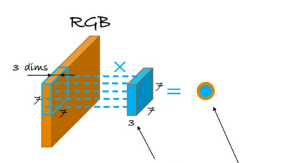


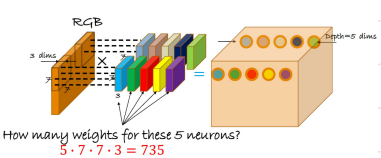


Dimensions of convolution



Number of weights





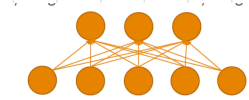
- Question: Spatial structure?
  - Answer: Convolutional filters
- Question: Huge input dimensionalities?
  - Answer: Parameters are shared between filters
- Question: Local variances?
  - Answer: Pooling

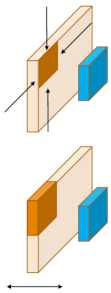
Local connectivity

- The weight connections are surface-wise local!
  - Local connectivit

The weights connections are depth-wise global

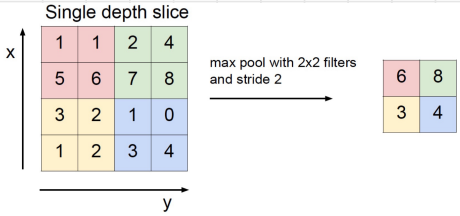
- For standard neurons no local connectivity
  - Everything is connected to everything

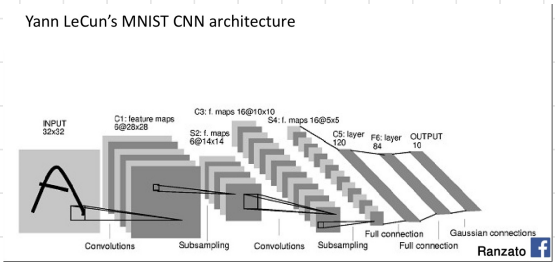




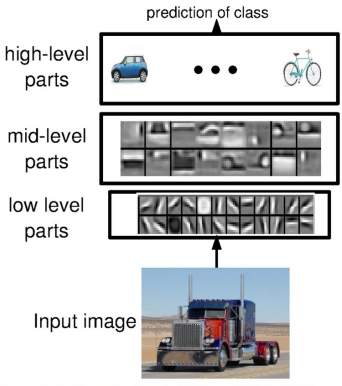
Pooling operations

- Aggregate multiple values into a single value
- Invariance to small transformations
  - Keep only most important information for next layer
- Reduces the size of the next layer
  - Fewer parameters, faster computations
- Observe larger receptive field in next layer
  - Hierarchically extract more abstract features





Interpretation



- distributed representations
- feature sharing
- compositionality

Lee et al. "Convolutional DBN's ..." ICML 2009

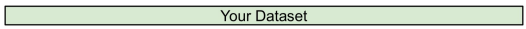
Ranzato

LEARNING NN

1. Setting Hyperparameters

Idea #1: Choose hyperparameters that work best on the data

BAD: big network always works perfectly on training data

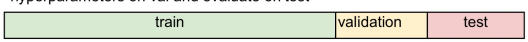


Idea #2: Split data into **train** and **test**, choose hyperparameters that work best on test data

BAD: No idea how algorithm will perform on new data

Idea #3: Split data into **train**, **val**, and **test**; choose hyperparameters on val and evaluate on test

Better!



2. Select Optimizer

1. Stochastic gradient descent

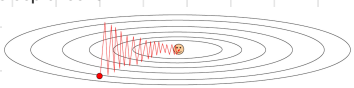
Gradient from entire training set:

$$\nabla C = \frac{1}{n}\sum_n \nabla C_n$$

- For large training data, gradient computation takes a long time
  - Leads to "slow learning"
- Instead, consider a mini-batch with m samples
- If sample size is large enough, properties approximate the dataset

$$\frac{\sum_{j=1}^{m} \nabla C_{n_j}}{m} \approx \frac{\sum_n \nabla C_n}{n} = \nabla C$$

Q) What if the loss function has a local minima or saddle point?
A) Zero gradient, gradient descent gets stuck Stochastic gradient descent
Q) What if loss changes quickly in one direction and slowly in another?
Q) What does gradient descent do?
A) Very slow progress along shallow dimension, jitter along steep direction

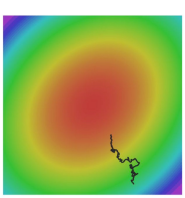


Loss function has high condition number: ratio of largest to smallest singular value of the Hessian matrix is large

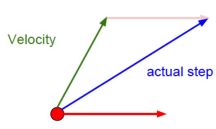
Our gradients come from minibatches so they can be noisy!

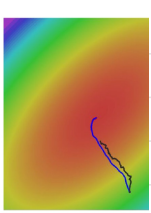
$$L(W) = \frac{1}{N}\sum_{i=1}^{N} L_i(x_i, y_i, W)$$

$$\nabla_W L(W) = \frac{1}{N}\sum_{i=1}^{N} \nabla_W L_i(x_i, y_i, W)$$



Momentum update:





SGD

$$x_{t+1} = x_t - \alpha \nabla f(x_t)$$

SGD+Momentum

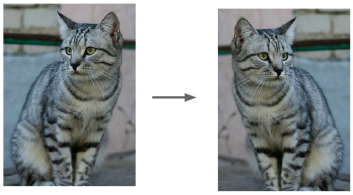
$$v_{t+1} = \rho v_t + \nabla f(x_t)$$
$$x_{t+1} = x_t - \alpha v_{t+1}$$

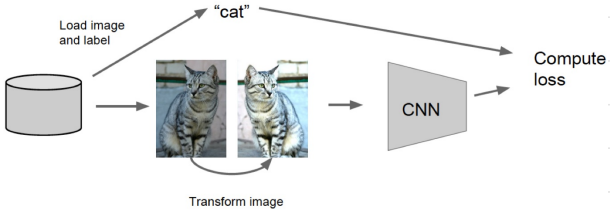
Build up velocity as a running mean of grad

Many variations of using momentum
- In PyTorch, you can manually specify the momentum of SGD
- Or, you can use other optimization algorithms with "adaptive" momentum, e.g., ADAM
- ADAM: Adaptive Moment Estimation
- Empirically, ADAM usually converges faster, but SGD gives local minima with better generalizability

3. Data Augmentation

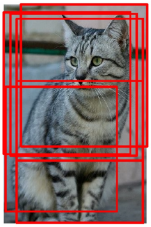
Horizontal flips





Random crops and scales

1. Pick random L in range [256, 480]
2. Resize training image, short side = L
3. Sample random 224 x 224 patch



Color jitter

Simple: Randomize contrast and brightness

More Complex:

1. Apply PCA to all [R, G, B] pixels in training set
2. Sample a "color offset" along principal component directions
3. Add offset to all pixels of a training image

Can do a lot more: rotation, shear, non-rigid, motion blur, lens distortions, ….

Exam

- Linear algebra, such as
  - rank, null space, range, invertible, eigen decomposition, SVD, pseudo inverse, basic matrix calculus
- Optimization:
  - Least square, low-rank approximation, statistical interpretation of PCA
- Image formation
  - diffuse/specular reflection, Lambertian lighting equation
- Filtering
  - Linear filter, filter vs convolution, properties of filters, filterbank, usage of filters, median filter
- Statistics:
  - Bias, variance, bias-variance tradeoff, overfitting, underfitting
- Neural network
  - Linear classifier, softmax, why linear classifier is insufficient, activation function, feed-forward pass, universality theorem, what does back-propagation do, stochastic gradient descent, concepts in neural networks, why CNN, concepts in CNN, how to set hyperparameter, moment in SGD, data augmentation