Instance Recognition

- visualize words: quantization, index, bag of word
- spacial verification: affine, ransac, hough
- Query/organize data by finding similarity btw scenes
  - instance means object vs category recognition

1. interest point detectors used like:
   1. harris (invariant to trans rotation illumination)
   2. or sift (rotation invariant, scale invariant)
2. regions that we find interesting -> extract as vector
3. lowest SSD (compare all and take closest within d)

Using Stereo disparity how far pixel moved in x and y -> define box that we can search in

Indexing local features boxes contribute feature v

- which building corresponds? look through features

-> don't compare each: **visual word**/descriptor

- exists as point in high dimmension space
- assumption **close** is more similar
- input query and find **closest**
- **Challenges**: Scale, efficiency in finding

Inverted file index: what page the things show up on

- find all images=page in which feature=thing shows

Visual Words

texton: cluster centers of filter responses over images

simple d/dx dim1; d/dy dim2

1. extract features -> point on dimmensional on plot
2. partition: reducing search (kmeans) center=vw
3. Inverted file index Word | Image
   - find word -> consider img

Issues: sampling where?, clustering, supervised?, universal vocab?, vocab size/num words?

- every pixel sampling = not informing (hard matching)
- little clusters means more searching

Summary via bag of words

Histogram over wordcounts (wordcount for vw)

- categorize each image with histogram of words
- rank frames by normalized scalar product of counts
  - could be weighted -> nearest neighbor search
- index of indiviudal words

Pro compactness, vector representations, results

- flexible to geometry/deformation/viewpoint bc hist

Cons geo, background, optimal formation clarity

- ignores geometry, must verify afterwards, encode fea
- background and foreground mixed (whole img bag)
- optimal vocabulary formation remains unclear

Retrieval quality, PR curve, optimize

precision = #relevant/ #returned //how many are rel

recall = #relevant/ #totatl relevant //of rel returned/tot

Avg percision: until we find all return find area under

vocab size:How well for each vocab size - sparsity

spatial verification: affine, RANSAC, hough

RANSAC more inliers = better match

- minimum subset of corr to estimate -> inliers
- uncertainty in img space
- must search all data points for inliers
- scales better in high dim

HOUGH each matched feature cast a vote on:

- location, scale, orientation of model obj
- uncertainty in parameter space
- linear complexity in corr & voting beyond 4D imp
- handle high outlier ratio
- needs to be invariant for base selection
- verify with enough votes
- gradient orientations all along edge of obj

1. collect all words within query region
2. inverted file index to find relevant frames
3. compare word counts
4. spacial verification

Object Recognition Intro

Category recognition

recognition via alignment

Pro effective when finding reliable features for clus

great results for matching specific instances

Cons Not suited for category recognitionnot diff view

- need to fit each object we are looking for

Classification now

Sematic segmentation: each pixel gets category

img classification: words that describe img

object detection: bounding boxes

instance segmentation: semantic + object

Task: training imgs -> class unknown

instance vs fine-grain recognition (living being)

category: mental image, similar percieved shape

Challenges: illumination, pose, clutter, occlusions, intra-class appearance, viewpoint, camera, robustness (objects covering), context (window)

supervised classfication labeled examples

measurements: mistakes, cost for mistake

risk of classifier: R(s) = Pr(4->9|using s)L(4->9)+Pr(9->4|using s)L(9->4) /// L is cost

- want to minimize risk:

Pr(4->9|using s)L(4->9)=Pr(9->4|using s)L(9->4)

ex: P(class is 9|x)L(9->4) + P(class is 4|x)L(4->4) /// L(4->4) is 0

$$\underbrace{P(skin|x)}_{\text{posterior}} = \frac{\overbrace{P(x|skin)}^{\text{likelihood}}\overbrace{P(skin)}^{\text{prior}}}{P(x)} \qquad P(skin|x) \propto P(x|skin)P(skin)$$

when we assume x is not chaging

P(skin) = skin/total pixels //don't predict too often

p(skin|x) > θ classifies as skin

Strategies

Generative Model training data to build representative probability model -> separately model class-conditional densities and priors

- i.e p(label) as p(skin) & p(x|label) as p(x|skin)

Discriminative: Directly construct a good decision boundary, model the posterior p(label|x)

model: y = mx+b or y = w^Tx

obj function:

loss function evaluates correctness: argmin on L2 norm or SSE

training/learning/fit minimizes this closed for

SSE: $w^* = (X^TX)^{-1}X^Ty$

inference(x) = $W^{Tx} = w_1x_1 + w_nx_n$

Inference (x): $\arg\max_k (\boldsymbol{Wx})_k$ (Take the class whose weight vector gives the highest score)

Training ($\mathbf{x}_i$,y_i):

$$\underbrace{\arg\min_{\boldsymbol{W}} \lambda\|\boldsymbol{W}\|_2^2}_{\text{Regularization}} + \underbrace{\sum_i}_{\text{Over all data points}} \sum_{j \neq y_i}^{n} \max(0, (\boldsymbol{Wx}_i)_j - (\boldsymbol{Wx}_i)_{y_i} + m)$$

Regularization

Over all data points

For every class j that's NOT the correct one (y_i)

Pay no penalty if prediction for clas is bigger than j by m ("margin").

Otherwise, pay proportional to the score of the wrong class.

first Wxi is incorrect class and second is based off correct, so we want prediction for correct to be big

Objective 2 softmax making probabilities

e^x -> sum used to normalize vector for out

Inference (x): $\arg\max_k (\boldsymbol{Wx})_k$ (Take the class whose weight vector gives the highest score)

$$\text{P(class j)}: \frac{\exp((Wx)_j)}{\sum_k \exp((Wx)_k)}$$

Training ($\mathbf{x}_i$,y_i):

$$\underbrace{\arg\min_{\boldsymbol{W}} \lambda\|\boldsymbol{W}\|_2^2}_{\text{Regularization}} + \underbrace{\sum_i^n}_{\text{Over all data points}} \underbrace{-\log\left(\frac{\exp((Wx)_{y_i})}{\sum_k \exp((Wx)_k)}\right)}_{\text{Pay penalty for negative log-likelihood of correct class}}$$

P(correct class)

p(corrrect) = 0 -> infinite penalty for neg log

p(correct) = 1 -> no penalty for neg log likelihood

Linear Models

Objective function: can be min/maxed as a goal

you fit/learn T using:

- pairs of data points (x data, y desired)
- just using set of datapoints x (need to define y)

x: feature vector/data point dead dim or feature is some aspect of this data

y: label/target each dimension represents some aspect of output data

maping to continuous, capture variance of data PCA

Bag of words gives us a better input

	supervised	unsupervised
d	classfication/categorization	clustering
c	regression (weight)	dim reduciton

N eqns > N vars -> overdetermined (will have errors)

N eqns = N vars -> exact solution

N eq < N vars -> undertetermined (infinite solns)

$$\rightarrow \sum_{i=1}^{k}(y_i - w^Tx_i)^2 \quad ||y - Xw||_2^2$$

Output: Temperature $y = \begin{bmatrix}33\\ \vdots \\ 83\end{bmatrix}$ **Inputs**: Latitude, 1 $X = \begin{bmatrix}42 & 1\\ \vdots & \vdots \\ 9 & 1\end{bmatrix}$ **Model/Weights**: Latitude, "Bias" $w = \begin{bmatrix}m\\ b\end{bmatrix}$

Separated Testing

overfitting: fitting datapoints too strictly poor on fin

- num datapoints = params = perfect fit

lat, high, snowfall -> nx4 x matrix

linear regression for predicting temperature

Rank deficiency more features than dim n -> no inverse -> infinite number of solutions -> regularization

Add **regularization** to objective that prefers some solutions

Before: $\arg\min_{\boldsymbol{w}} \|\boldsymbol{y} - \boldsymbol{Xw}\|_2^2$ → Loss

After: $\arg\min_{\boldsymbol{w}} \|\boldsymbol{y} - \boldsymbol{Xw}\|_2^2 + \lambda\|\boldsymbol{w}\|_2^2$

Loss, Trade-off, Regularization

Want model "smaller": pay a penalty for **w** with big norm

Intuitive Objective: accurate model (low loss) but not too complex (low regularization). λ controls how much of each.

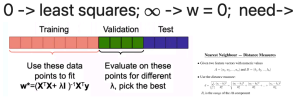
Take $\frac{\partial}{\partial \boldsymbol{w}}$ set to **0**, solve

$$\boldsymbol{w}^* = (\boldsymbol{X}^T\boldsymbol{X} + \lambda\boldsymbol{I})^{-1}\boldsymbol{X}^T\boldsymbol{y}$$

$\mathbf{X^TX+\lambda I}$ is full-rank (and thus invertible) for λ>0

λ is a hyperparameter which we have to provide

0 -> least squares; ∞ -> w = 0; need-> validation



Training (x_i,y_i): Memorize training set

Inference (x):

```
BestDist, prediction = Inf, None
for i in range(N):
    if dist(x_i, x) < BestDist:
        BestDist = dist(x_i, x)
        prediction = y_i
```

kNN regression algorithm

- Given:
  - training examples $\{x_i, y_i\}$
    - x_i ... attribute-value representation of examples
    - y_i ... real-valued target (profit, rating on YouTube, etc)
  - testing point x that we want to predict the target
- Algorithm:
  - compute distance $D(x,x_i)$ to every training example x_i
  - select k closest instances $x_{i1}...x_{ik}$ and their labels $y_{i1}...y_{ik}$
  - output the mean of $y_{i1}...y_{ik}$: $\hat{y} = f(x) = \frac{1}{k}\sum_{j=1}^{k} y_{i_j}$

as num data -> ∞ error rate is gauranteed to be at least x2 worse than optimal (since we not learning)

Linear model 1 weight per class

$W_{C \times F}$ classes x features

Geometric Intuition

close to line distance from boundary gets smaller

if wi is big -> values of xi are indicative

Multiclass SVM - minimize overfitting to objective

Gradient Decent

Add regularization to objective that prefers solution: want accurate model (low loss) but not too complex (low regularization)

Optimization

Goal: find the **w** minimizing some loss function L. $\arg\min_{\boldsymbol{w} \in R^N} L(\boldsymbol{w})$

$$L(\boldsymbol{W}) = \lambda\|\boldsymbol{W}\|_2^2 + \sum_{i=1}^{n} -\log\left(\frac{\exp((Wx)_{y_i})}{\sum_k \exp((Wx)_k)}\right)$$

Works for lots of different Ls:

$$L(\boldsymbol{w}) = \lambda\|\boldsymbol{w}\|_2^2 + \sum_{i=1}^{n}(y_i - \boldsymbol{w}^T\boldsymbol{x}_i)^2$$

$$L(\boldsymbol{w}) = C\|\boldsymbol{w}\|_2^2 + \sum_{i=1}^{n}\max(0, 1 - y_i\boldsymbol{w}^T\boldsymbol{x}_i)$$

Option num 1a grid search

evaluate loss fucntion by sampling space

partition space and pick one that minimized loss:

best, bestscore: none, inf

```
for dim1Value in dim1Values:
    ..
    for dimNvlaue in dimN vlaues
        w = [dim1value, ..., dimNvalue]
        if L(w) < bestScore
            best, bestScore = w,L(w)
```

scales poorly on high deminesions complex samples

perdim^numberofdims

Option 1b random search

learn with random points RANSAC style

```
for iter in range(numiterations):
    w = random(N,1) #sample
    score = L(w) #eval
        if score < bestScore
```

slow - throwing darts at high dim dart board can miss

P(all correct) = numgood^N

Option 1a&1b sma dim, space bound, obj can't be analyzed otherwise use gradient

Gradient arrows rep direction, len mag

rescale to unit length

Want: $\arg\min_{\boldsymbol{w}} L(\boldsymbol{w})$

What's the geometric interpretation of: $\nabla_{\boldsymbol{w}}L(\boldsymbol{w}) = \begin{bmatrix}\partial L/\partial \boldsymbol{x}_1\\ \vdots \\ \partial L/\partial \boldsymbol{x}_N\end{bmatrix}$

Which is bigger (for small positive α)?

$$L(\boldsymbol{w}) \overset{\leq ?}{\underset{>?}{}} L(\boldsymbol{w} + \alpha\nabla_{\boldsymbol{w}}L(\boldsymbol{w}))$$

w0 = *initialize*() #initialize

for iter in range(numIters):

g = $\nabla_{\boldsymbol{w}}L(\boldsymbol{w})$ #eval gradient

w = **w** + -*stepsize*(iter)***g** #update w

return **w**

Method: move in direction of negative gradient (alpha is the size of this step)

```
for iter in range(numIters):
    index = randit(0, #data)
    g = gradient(data[index], L)
    w = w + -stepsize(iter) * g #update 2
```

Computing Gradient

$$\nabla_{\boldsymbol{w}}L(\boldsymbol{w}) = \begin{bmatrix}\partial L(\boldsymbol{w})/\partial \boldsymbol{x}_1\\ \vdots \\ \partial L(\boldsymbol{w})/\partial \boldsymbol{x}_n\end{bmatrix}$$

How do you compute this?

$$\frac{\partial f(x)}{\partial x} = \lim_{\epsilon \to 0}\frac{f(x+\epsilon) - f(x)}{\epsilon}$$

In practice, use: $\frac{f(x+\epsilon) - f(x-\epsilon)}{2\epsilon}$

function evals per dimension -> 2

or use calculus - take derivative of L(w)

$$-\nabla_{\boldsymbol{w}}L(\boldsymbol{w}) = \underbrace{-2\lambda\boldsymbol{w}}_{\text{Push w towards 0}} + \overbrace{(2(y - \boldsymbol{w}^T\boldsymbol{x})\boldsymbol{x})}^{\alpha}$$

If y > w^Tx (too *low*): then w = w + αx for some α

Before: w^Tx. After: $(w + \alpha x)^{Tx} = w^{Tx} + \alpha x^{Tx}x$

abs(x) = $\frac{\partial}{\partial x}f(x) = \text{sign}(x) \quad x \neq 0$; $\frac{\partial}{\partial x}f(x) \in [-1,1] \quad x = 0$

Subset B $\quad -\nabla_{\boldsymbol{w}}L_B(\boldsymbol{w}) = -2\lambda\boldsymbol{w} + \sum_{i \in B}(2(y_i - \boldsymbol{w}^T\boldsymbol{x}_i)\boldsymbol{x}_i)$

```
for iter in range(numiters):
    g = gradient(data, L)
    w = w + -stepsize(iter)*g #update w
```

```
for iter in range(numIters):
    subset = choose_samples(#data, B)
    g = gradient(data[subset], L)
    w = w + -stepsize(iter) * g #update 2
```

Learning rate i.e. stepsize (lower it over time - don't overshoot later)

Solution avergate gradient: exponentially decaying weights: called momentum

Optimize

w on training set with SGD to max training acc

λ on random/grid search to max validation acc

optimizing λ ontraining sets it to 0

bias: error intrinsic to model

overfit: taking every datapoint too seriously

- high variance - removing one var changes

underfit: too little parameters to represent features

big weights = more complex model

fitting:

$$\arg\min_{\boldsymbol{w}} \|\boldsymbol{y} - \boldsymbol{Xw}\| + \lambda\|\boldsymbol{w}\|$$

$$\begin{bmatrix}x_1^F & \cdots & x_1^2 & x_1 & 1\\ \vdots & \ddots & \vdots & \vdots & \vdots \\ x_N^F & \cdots & x_N^2 & x_N & 1\end{bmatrix} \quad \begin{bmatrix}w_F\\ \vdots \\ w_0\end{bmatrix}$$

no reg: fits all datat points, reg: can't fit all

Neural Networks

1. use linear models for image classfication problems
2. use loss functions to express preference over different choices of weights
3. SGD to minimize loss function and train model
4. Add regularization to control overfitting

mointor learning error on new data comes from:

- bias: model is oversimplified and cannot fit data
- Variance: you don't have ability to estimate from limited data
- Inherent: the data is intrinsically difficult

Trade off btw bias and variance

Trade off btw bias and variance

small data: inflection closer to y axis -> overfits w/lil

big data: further on x -> overfits with bigger model

underfitting: poorly on training + validation with bias

sol: more features, more pow model, reduce regulariza

overfitting: well on training, bad validation, due to vari

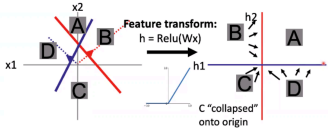
sol: more data, less powerful model, regularize more

generally want to overfit -> then cut down

input image: $x \in R^D$ category scores: $s \in R^c$

Linear Classifier: $s = Wx$, $W \in R^{C \times D}$

2-layer Neural Net: $s = W_2\max(0, W_1x)$, $W_1 \in \mathbb{R}^{H \times D}$, $W_2 \in \mathbb{R}^{C \times H}$



fully connected means: every element each layer con

width size of each layer, depth = num layers

deeper: more processing, less interpretable,

overfitting with low data, increase complexity

Backpropagation

want: $w_{t+1} = w_t - \alpha \nabla_w L(f(w))$ s.t. we can use SGD on complex L(f)

Chain rule: $\frac{\partial f}{\partial x} = \frac{\partial f}{\partial q}\frac{\partial q}{\partial r}\frac{\partial r}{\partial x}$

f(x) = (-x + 3)^2

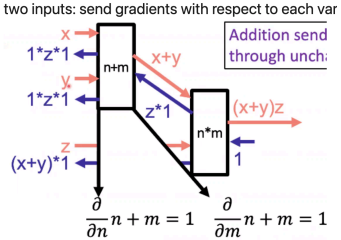
f = q^2, q = r +3, r = -x indiviudal funcitons -> partial

Backprop n -> f -> f(n) g*(df/dn) <- f <- g

- g=1 if no previous gradients

$\frac{\partial}{\partial n}(-n) = -1$

two inputs: send gradients with respect to each var



$\frac{\partial}{\partial n} n + m = 1$ $\frac{\partial}{\partial m} n + m = 1$

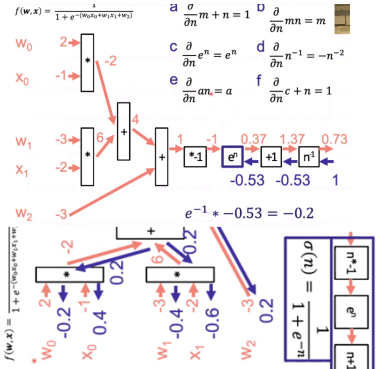
$\rightarrow 1 * z * 1$ $\rightarrow 1 * z * 1$

$f(w,x) = \frac{1}{1+e^{-(w_0x_0+w_1x_1+w_2)}}$

a $\frac{\partial}{\partial n} m + n = 1$ b $\frac{\partial}{\partial n} mn = m$

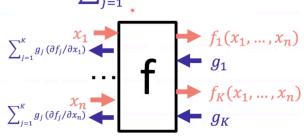
c $\frac{\partial}{\partial n} e^n = e^n$ d $\frac{\partial}{\partial n} n^{-1} = -n^{-2}$

e $\frac{\partial}{\partial n} an = a$ f $\frac{\partial}{\partial n} c + n = 1$

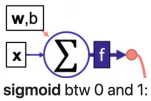


Gradients from different backwards sun

$\sum_{j=1}^{K} g_j (\partial f_j / \partial x_i)$



$= (1 - \sigma(n)\sigma(n))$



sigmoid btw 0 and 1: changes output range

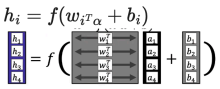
gradient are near 0 if neuron is high/low: can stop learning/dectivate a neuron

ReLU constant gradient, converges faster, neg -> 0 g

leaky relu x if x >= 0 .01x x < 0

Fully connected Network

$h_i = f(w_i^T \alpha + b_i)$



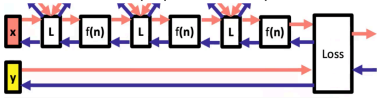
CovNets

forward pass: compute function

backward pass: compute derivate of all parts f

Each hidden layer acts as a hyperplane separating

- derivative of output/error with respect to all w




```
W = initializeWeights()
for i in range(numiterations):
    #sample a batch
    batch = random.subset(0, #datapoints, K)
    batchx, batchy = dataX[batch], datay
    #compute gradent with batch
    gradW = backprop(Loss(NN(batchx, W),
batchY)) #computes all gradients
    #update W with gradient step
    W += -stepsize * gradW*
```

Loss tells us how close we got to our desired input

L(n) = Wn + b # W^Tx + b # +/- dist to line

looking at absolute dist from line:

$max(w_{1^t x} + b, 0) + max(-w_{1^t x} + b, 0)$

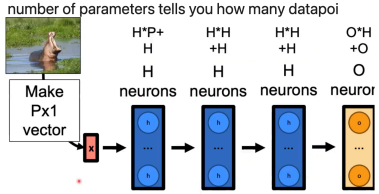
next layer: w_1 distance - w_2 distance > 0

Parameters

weigths 1x3 -> parameters 3 (bias is 3rd)

2 x | 4 h | 1 y -> weights 1x4 + 4x2 = 12 -> 12 + 5

number of parameters tells you how many datapoi



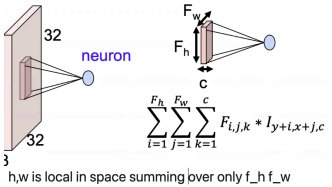
P: 285x350 picture **(terrible!)**, H: 1000, O: 3

-> reduction by convolutional NN

keep spatial resoltuion around

Fx1 transform ma-> HxWxF transform: convolution

Neuron is the same: weighted linear average



$\sum_{i=1}^{F_h}\sum_{j=1}^{F_w}\sum_{k=1}^{c} F_{i,j,k} * I_{y+i,x+j,c}$

h,w is local in space summing over only f_h f_w

while c is global: sum over all channels

(a) #input channels can be greater than one

(b) though these are call convolutional networks usually actually apply cross-correlation

Neural net:	**Convnet:**
series of matrix-multiplies parameterized by **W,b** + nonlinearity/activation	series of convolutions parameterized by **F,b** + nonlinearity/activation
Fit by gradient descent	Fit by gradient descent

Stride: how much are we moving the filter

Output Size $\frac{(N-F)}{S} + 1$

- add padding to allow for various strides

input: 32x32x3, 5x5 stride 1, neurons 5 28x28x5

num of parameters: (5x5)x3x5 = 380 SIZExCxN

Changing stride only changes output volumn

want spatiel resolutoin of activation/images

smaller; applied per channel

Max pool: take highest from window

Avg pool: take avg over window for out

Losing spatial specificity, can learn on fewer ex

too simple: very little we can learn form it

Covolutional NN 2

Sizing

binary classification which image is cat f=1

multiclassification: which of 1000 is it F=1000

Pose recognition: identify the x,y coord of 28 body joins of human F=56

Generic regression: at what gps cood img taken f2

HxWxC -> 1x1xF # can do avg pool HxW filter s=1

- if F == C

or HW convo with F filters

Alexnet each block is HxWxC

transform one vol to anothe rwith conv

Input	Conv 1	Conv 2	Conv 3	Conv 4	Conv 5	FC 6	FC 7	Output
227x227 3	55x55 96	27x27 256	13x13 384	13x13 384	13x13 256	1x1 4096	1x1 4096	1x1 1000

227x227x3 -> 55x55 96 <-decided

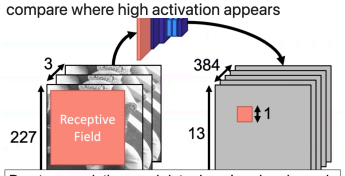
11x11 filter stride of 4 (227 - 11)/4 + 1 = 55

para 96 11x11 on 3 + 96

non-linearity applied to every element (sam dims)

conv 1 filters: 3 input dim R,G,B -> 96 filters with random weights (visual filters it found useful)

For later layers take slices differing btw images - compare where high activation appears



Due to convolution, each later layer's value depends on / "sees" only a fraction of the input image.

filter receptive field - when reach intermediate node what has been touched in the forward

Classic vs Deep recognition cut off last linear layer from perlearned model (useful for little data)

as a fixed feature extracter like we did with SIFT -> BOW -> classifier (classic steps don't talk)

VGG model 3x3 filters

3x3 followed by 3x3 filter -> filter with 5x5 receptive field

- very local, very few parameters

Receptive Field: 7x7 pixels

Parameters/channel: 49

Number of ReLUs: 1

Receptive Field: 7x7 pixels

Parameters/channel: 3x3x3=**27**

Number of ReLUs: **3**

Can add non-linearities in between each 3x3 filter

Input	Conv 1	Conv 2	Conv 3	Conv 4	Conv 5	FC 6	FC 7	Output
224x224 3	224x224 64	112x112 128	56x56 256	28x28 512	14x14 512	1x1 4096	1x1 4096	1x1 1000

backprop leverages chain rule (need to compute all local derivatives and multiply them)

- the deeper it is the multiplication gets closer and closer to 0 (d^(n-1)) is the vanishing gradient
- d >> 1 big -> exploding gradients

Localization

sol1 Batch Normalization in btw layers

normalizes activates into normal distribution

Mean(x) != Mean(Y) != 0 Mean(x) = Mean(Y) = 0

Var(x) != Var(y) != 1 Var(x) = Var(y) = 1

if values varry a lot -> gradients becom huge

too compact -> gradient too small

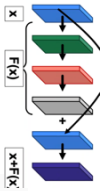
(want right img), subtract out emperical mean

gradient values stay closer to 1

Residual Learning

new building block: x + F(x)

skip connections: allowing model to decide what extra stuff it might want to add vs having to always pass through



Training CNN

```
1. download big dataset
2. initialize network weightrs randomly
3. for epoch in rang
    1. shuffle dataset
    2. for each minibatch in dataset
        1. put dat on gpu
        2. compute gradient
        3. update gradient with SGD
```

Need to start **w** somewhere

- AlexNet: weights ~ Normal(0,0.01), bias = 1
- "Xavier" initialization: Uniform$(\frac{-1}{\sqrt{n}}, \frac{1}{\sqrt{n}})$ where n is the number of neurons
- "Kaiming" initialization: Normal$(0, \sqrt{2/n})$

Trainig covnet

fewer data points that parameters = trouble

solution: need regularization/more data, agumentation that don't affect output

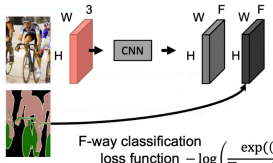
Fine tuning initialize the weights with one that has already been made from existing network

- few images: don't use fine tuning <10k
  - fixed feature extraction and linear cut off
- 10k<medium<100k try fine tuning
- 100k < consider trying from scratch

Localization vector output per pixel

HxW image into HxWxF (needed for semantic)

- don't distinguish between object instances!
- could be naming things



F-way classification loss function at every pixel: $-\log\left(\frac{\exp((Wx)_{y_i})}{\sum_k \exp((Wx)_k)}\right)$

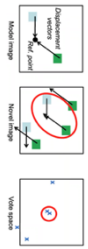
need HxW losses for the entire image instead of 1

- could make a depth map (regression)

Encoder-Decoder

Key idea: First **downsample** towards middle of network. Then **upsample** from middle.

- point in img -> sinusoid segment in hough



Algorithm

1. $H[d, (\theta)] = 0$$
2. for each edge point I[x,y]
   1. for theta = $[\theta_{min} to \theta_{max}]$
   2. theta = graident at x,y
   3. $xcos(\theta) + ysin(\theta) = d$
   4. $H[d, (\theta)] += 1$
3. find (d, theta) where H[] is max
4. detect line given by d

larger -> smaller is encoder

smaller -> larger is decoder

map 1x1xC -> 1x1xF need matrix that is CxF

- W(FxC matrix) + b (Fx1 vector)

with convolution filter of size 1x1 with F filters

$$b + \sum_{i=1}^{F_h}\sum_{j=1}^{F_w}\sum_{k=1}^{c} F_{i,j,k} * I_{y+i,x+j,c} \longrightarrow b + \sum_{k=1}^{c} F_k * I_c$$

linear layer -> convolutional (could have image that is x4 in size -> apply alexnet) less restricted by size

upsampling opposite downsampling by unpooling

convolution -> transpose convolutoin

- nearest neighbor pooling operation like reverse of max pool -> recover size that we previous had
- add padding around or padding in between each element

Accuracy intersection over union, avg over classes

visual words

find? kmeans clustering

- words are cluster centers

new image has features -> what would BoW be?

- BoW is histogram of each w `[count1, coun^>]`

Classifiers

k=1 means you just take the max

k=3 means you assign the majority

- if all are tie -> has no answer

binary affine: 2 weights, 1 bias

20 dim, 5 classes -> 20x5 + 5 params

SGD

algo: walk along negative gradient

params: step-size/learnign rate, batch size

- leanring rate too large (can miss minima)
- too small: get stuck on local

regulizers: momentum, weight-decay

Covnets

- conv vs linear layers: computing block sizes, para
  - conv allows us to go deeper by reducing par
- function to apply to parameters:
  - 100x100 input, 5 nodes -> 100^2 x5 + 5
  - filter size 5x5 5 filters -> 4x4x5 + 5
- conv layer stride 2 (100-4)/2 + 1
- Modeling tools: 3x3 conv, batch normalizatio, data aug, fine tuning
- sementic segmentation: unpooling transpose conv

Neural Nets

- linear/full connected blocks
- activation functions add non-linearity

df/dq was in between blocks