No Coding Farmer

Tao Hu
Department of Computer Science
Peking University
No.5 Yiheyuan Road Haidian District, Beijing, P.R.China
taohu@pku.edu.cn

Abstract

Some Miscellaneous Summary.

Author Info: Taohu ,Peking University

Contents

1 Expectation Maximization Introduction **5**

1.1 EM Induction 5

1.2 EM convergence proof 5

1.3 Different Writing Style of EM Algorithm 5

2 EM applications **6**

2.1 Gaussian Mix Model 6

2.2 Hidden Markov Model 6

2.3 Naive Bayesian 7

2.4 other papers 7

3 VAE **7**

4 ADMM **7**

5 Key steps you must know when building a DL Framework **7**

5.1 Convolution 7

5.2 Loss Function 8

5.2.1 Loss function example 9

5.3 WitchCraft 10

5.3.1 Awesome Maxout 10

5.3.2 Softmax 10

5.3.3 ResNet Pre Activation[4] 10

5.4 How to back Propagation 10

6 R-PCA **11**

6.1 Solve RPCA by ADMM 11

6.2 Adaptive Penalty for ADMM 11

7 SFM **12**

8 Mainfold Learning **12**

8.1 Laplace Matrix 12

8.2 Normalized Cut 12

8.3 Linear Dimension Reduction 12

8.3.1 PCA 12

8.4 NonLinear Dimension Reduction 13

9 DCT **13**

10 KL-divergence Application **17**

11 GAN **17**
11.1 Why is maxD then minG . . . 18
11.2 WGAN . . . 18

12 Reinforcement Learning **20**
12.1 Model-based Method . . . 23
12.1.1 Policy Iteration . . . 23
12.1.2 Value Iteration . . . 24
12.2 Model-Free Method . . . 25
12.2.1 on-policy TD . . . 25
12.2.2 on-policy MCMC . . . 25
12.2.3 off-policy method . . . 25
12.2.4 Comparisons: DP, MC, TD . . . 25

13 RNN **28**
13.1 BPTT . . . 28
13.2 LSTM . . . 28
13.3 Multidimension RNN . . . 30

14 Algorithm **30**

15 Determinantal Point Processes **30**

16 Typical Basic Networks **31**
16.1 VGGNet . . . 31
16.2 Inception Series . . . 31
16.2.1 Inception v1:GoogleNet . . . 31
16.2.2 Inception v2,v3 . . . 32
16.2.3 Inception v4 . . . 33
16.3 ResNext . . . 33
16.4 Xception . . . 34
16.5 ResNet . . . 36
16.6 DenseNet . . . 39

17 Network Compression **40**
17.1 MobileNet . . . 40
17.2 ShuffleNet . . . 43

18 Detection **44**
18.1 SPPNet . . . 44
18.2 Faster RCNN . . . 44
18.3 SSD . . . 45

18.4 YOLO 45
18.5 Mask-RCNN 45
18.6 NMS 45

19 Segmentation **46**
19.1 Deformable Convolution Network 46

20 Large Kernel Matters **46**

21 Skeleton **47**
21.1 HourGlass 47

1 Expectation Maximization Introduction

1.1 EM Induction

$L(\theta) = \sum_{i=1}^{M} logp(X;\theta) = \sum_{i=1}^{m} log \sum_z p(X, Z;\theta)$
let θ_i be some distribution over z's ($\sum_z \theta_i(z) = 1, \theta_i(z) \geq 0$)
$\sum_i logp(X^{(i)};\theta)$
$= \sum_i log \sum_{Z^{(i)}} \theta_i(Z^{(i)}) \frac{p(X^{(i)},Z^{(i)};\theta)}{\theta_i(Z^{(i)})}$
$\geq \sum_i \sum_{Z^{(i)}} \theta_i(Z^i) log \frac{P(X^{(i))},Z^{(i)};\theta)}{\theta_i(Z^{(i)})} (f(x) = logx \quad is \quad concave.)$
let $\frac{P(X^{(i))},Z^{(i)};\theta)}{\theta_i(Z^{(i))}} = C$
the equality can be only reached when $\frac{P(X^{(i))},Z^{(i)};\theta)}{\theta_i(Z^{(i))}}$ is a constant.
we can get: $\sum_i \frac{p(X^{(i)},Z^{(i)};\theta)}{C} = 1$ namely: $\sum_i p(X^{(i)}, Z^{(i)};\theta) = C$
further induction: $\theta_i(Z^{(i)}) = \frac{p(X^{(i)},Z^{(i)};\theta)}{\sum_i p(X^{(i)},Z^{(i)};\theta)} = p(Z^{(i)}|X^{(i)};\theta)$
so the procedure of EM algorithm is:

Repeat Until Convergence:

- E-step: for each i,get $Q_i(Z^{(i)}) = p(Z^{(i)}|X^{(i)};\theta)$
- M-step: $\theta := argmax_\theta \sum_i \sum_{Z^i} Q_i(Z^{(i)}) log \frac{p(X^{(i)},Z^{(i)}),\theta)}{Q_i(Z^{(i)})}$

1.2 EM convergence proof

let $l(\theta^{(t)}) = \sum_i \sum_{Z^{(i)}} Q_i^{(t)}(Z^{(i)}) log \frac{p(X^{(i)},Z^{(i)},\theta)}{Q_i^{(t)}(Z^{(i)})}$
then,we have the following inequality:
$l(\theta^{(t+1)})$
$\geq \sum_i \sum_{Z^{(i)}} Q_i^{(t)}(Z^{(i)}) log \frac{p(X^{(i)},Z^{(i)},\theta^{(t+1)})}{Q_i^{(t)}(Z^{(i)})}$
$\geq \sum_i \sum_{Z^{(i)}} Q_i^{(t)}(Z^{(i)}) log \frac{p(X^{(i)},Z^{(i)},\theta^{(t)})}{Q_i^{(t)}(Z^{(i)})}$
$\geq l(\theta^{(t)})$
the first inequality is because :$l(\theta) \geq \sum_i \sum_{Z^{(i)}} Q_i^{(t)}(Z^{(i)}) log \frac{p(X^{(i)},Z^{(i)},\theta)}{Q_i^{(t)}(Z^{(i)})} \forall \theta, Q_i$
the second inequality is because of the maximum of the M-step.

Hence, EM causes the likelihood to converge monotonically.

1.3 Different Writing Style of EM Algorithm

There are many writing style of EM algorithm. here I just mention the book <Statistics Learning Method> by LiHang who is very famous in China.

EM algorithm from LiHang(Li-version):

Algorithm 1 EM from LIHang

Require: observation X,hidden variable Z,joint distribution $P(X, Z|\theta)$,conditional distribution $P(Z|Y, \theta)$

while Not convergence **do**

E-Step: let $\theta^{(i)}$ is the i-th estimate of θ,

$Q(\theta, \theta^{(i)}) = E_z[logP(X, Z|\theta)|X, \theta^{(i)}] = \sum_Z logP(X, Z|\theta)P(Z|X, \theta^{(i)})$

M-step: $\theta^{(i+1)} = argmax_\theta Q(\theta, \theta^{(i)})$

end while

output model parameter θ

it seems that Li-version is different from the above version. however, they are the same. because:

- the above version just consider every data, so that it include subscript i. however Li-version only consider one data.
- the above version can be transformed to Li-version.

  $\sum_Z Q(Z)log\frac{P(X,Z:\theta)}{Q(Z)}$
  $= \sum_Z P(Z|X;\theta^{(t)})log\frac{p(X,Z;\theta)}{p(Z|X;\theta^{(t)})}$
  $= \sum_Z P(Z|X;\theta^t)logP(X,Z;\theta) - \sum_Z P(Z|X;\theta^{(t)})logP(Z|X;\theta^{(t)})$

  as the variable is θ,so $\sum_Z P(Z|X;\theta^{(t)})logP(Z|X;\theta^{(t)}$ can be removed.
- $Q(\theta, \theta^{(i)}) = \sum_Z logP(X, Z|\theta)P(Z|X;\theta^{(i)})$ can be also written as $Q(\theta, \theta^{(i)}) = \sum_Z logP(X, Z|\theta)P(Z, X;\theta^{(i)})$,because X is a observation.

2 EM applications

2.1 Gaussian Mix Model

GMM can be solved by EM. **notice here we use the expectation of EM:**

$Q(\theta, \theta^{(i)})$
$= E_\gamma[logP(y, \gamma|\theta)|y, \theta^{(i)}]$
$= E[\sum_{k=1}^K[n_k log\alpha_k + \sum_{j=1}^N \gamma_{jk}[log\frac{1}{\sqrt{2\pi}} - log\sigma_k - \frac{1}{2\sigma_k^2}(y_j - \mu_k)^2]]]$
$= \sum_{k=1}^K[(E\gamma_{jk})log\alpha_k + \sum_{j=1}^N(E\gamma_{jk})[log\frac{1}{\sqrt{2\pi}} - log\sigma_k - \frac{1}{2\sigma_k^2}(y_j - \mu_k)^2]]$

here $(E\gamma_{jk})$ can be easily calculated.

$\hat{\mu_k}, \hat{\sigma_k^2}$ can be acquired by derivation.

$\hat{\alpha_k}$ can be acquired by the derivation on the Lagrangian($\sum_i^K \alpha_k = 1$).

2.2 Hidden Markov Model

HMM Learning Method is also called Baum-Welch algorithm.the target is learning $\lambda = (A, B, \pi)$.

Q function is:

$Q(\lambda, \bar{\lambda}) = \sum_I logP(O, I|\lambda)P(O, I|\bar{\lambda})$

$P(O, I, \lambda) = \pi_{i1}b_{i1}(o_1)a_{i1i2}b_{i2}(o_2)...a_{i_{T-1}i_T}b_{i_T}(o_T)$

so the Q function can also be written as:

$$Q(\lambda, \bar{\lambda}) =$$
$$\sum_I log\pi_{i1} P(O, I|\bar{\lambda}) + \sum_I (\sum_{t=1}^{T-1} loga_{i,i+1}) P(O, I|\bar{\lambda}) + \sum_I (\sum_{t=1}^{T} logb_{it}(o_t)) P(O, I|\bar{\lambda})$$

note here: I is not only one state. it includes state length from 1 to T,which all start from i_1

so we can solve the maximum of Q function by derivation on the Lagrangian polynomial (because exists these limitations: $\sum_{i=1}^{N} \pi_i = 1, \sum_{j=1}^{N} a_{ij} = 1, \sum_{i=1}^{M} b_i = 1$)

2.3 Naive Bayesian

2.4 other papers

We can use softmax to model transition probability, normal distribution to model emission probability.

it's a good example in Car that Knows Before You Do: Anticipating Maneuvers via Learning Temporal Driving Models, the AIO-HMM can be more complicated,which can be enriched by the graphic model by M.I Jordon.

3 VAE

here is a complete VAE tutorial [2]

$$max \quad logP(x)$$
$$\text{lhs} = log \int P(x, z) dz$$
$$= log \int P(x/z) p(z) dz$$
$$= \log \int \frac{P(x/z)}{q(z/x)} q(z/x) p(z) dz$$
$$= logE_{q(z/x)}[\frac{p(x/z)}{q(z/x)} p(z)]$$
$$\text{jenson's inequality,we can know: } \geq E_{q(z/x)}[log \frac{p(x/z)}{q(z/x)} p(z)]$$
$$= E_{q(z/x)}[logp(x/z)] + E_{q(z/x)}[log \frac{p(z)}{q(z/x)}]$$
$$= E_{q(z/x)}[logp(x/z)] - E_{q(z/x)}[log \frac{q(z/x)}{p(z)}]$$
$$= E_{q(z/x)}[logp(x/z)] - KL(q(z/x)||p(z))$$

4 ADMM

5 Key steps you must know when building a DL Framework

5.1 Convolution

convert the convolution to matrix multiplication like the fully-connected network.

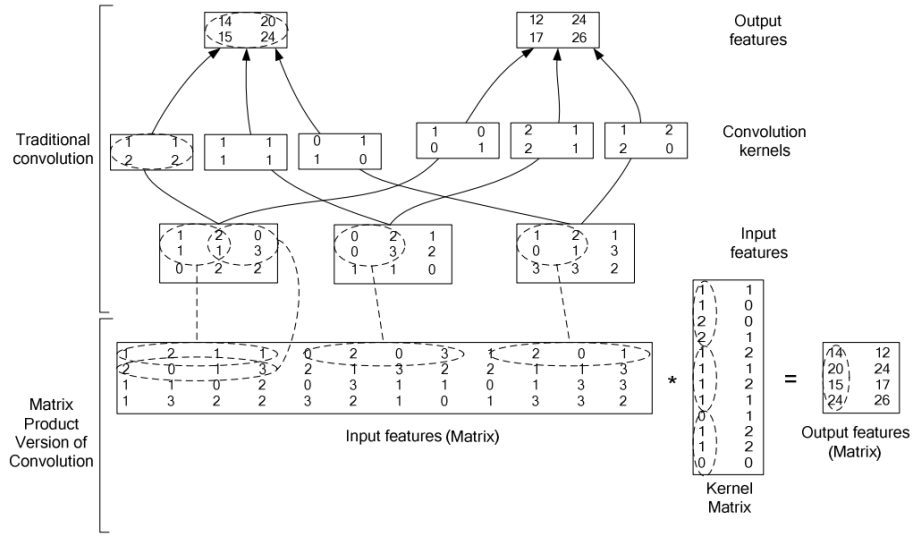


Figure 2. Example convolution operations in a convolutional layer (biases, sub-sampling, and non-linearity omitted). The top figure presents the traditional convolution operations, while the bottom figure presents the matrix version.

figure is from [1].
Let's notate:
H:image height
W:image width
in: input image number
out: output image number
K: convolution kernel size

the matrix product version of convolution,the dimension of two matrix is:
(H*W) *(in*K*K)
(in*K*K) * out

the operation above is like matlab function: img2col
im2col(A,[m n],block_type),where block_type="sliding".

GEMM(GEneral Matrix to Matrix Multiplication) is at the heart of deep learning.`https://petewarden.com/2015/04/20/why-gemm-is-at-the-heart-of-deep-learning/`

5.2 Loss Function

`https://stats.stackexchange.com/questions/222585/what-are-the-impacts-of-choosing-different-loss`

Hinge Loss:
$f(y,t) = (1-yt)_+$

Log Loss:
$f(y,t) = ln(1+e^{-yt})$

Square Loss:
$f(y,t) = (1-yt)^2$
square loss is sensitive to outliers.

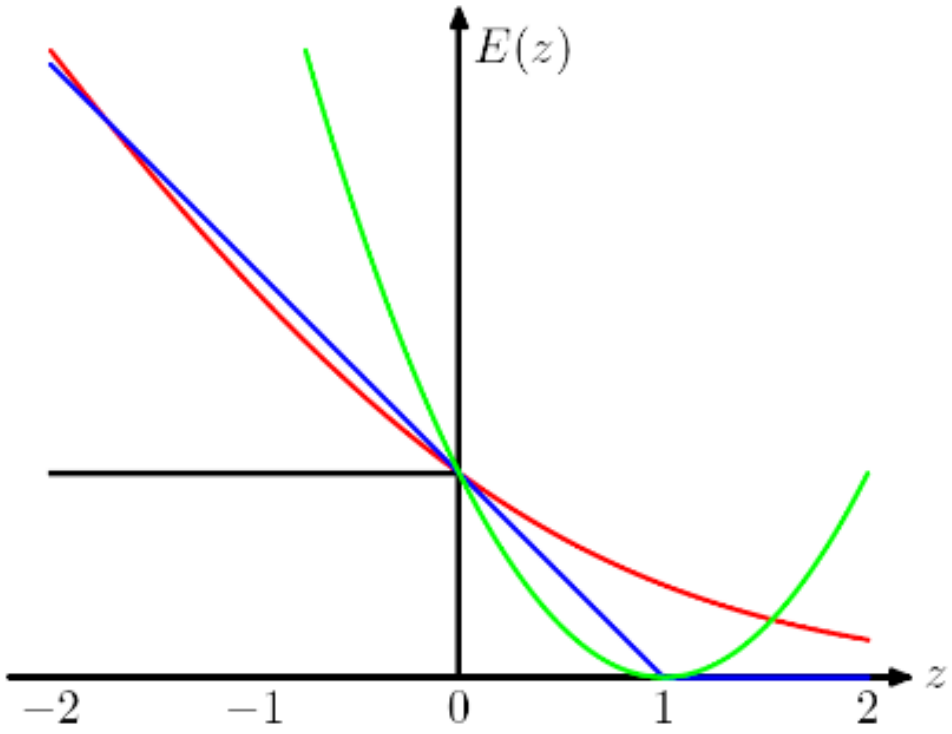
Following plot is coming from Chris Bishop's PRML book. The Hinge Loss is plotted in blue, the Log Loss in red, the Square Loss in green and the 0/1 error in black.

Cross Entropy:
`https://jamesmccaffrey.wordpress.com/2013/11/05/why-you-should-use-cross-entropy-error-instead-o`

$\sum_i^n \sum_k^K -y_{true}^k log(y_{predict}^{(k)})$

when multi classification, only exists one $y_{true}^i = 1$ with remaining $y_{true}^j = 0, i \neq j$.

Interpretation of Cross Entropy:
(1). encoding length:if a signal exist with probability p. then we only need $log(\frac{1}{p})$ bits to encode it.
(2). KL-divergence. $E_{x\sim p}[-log(Q(x))] - E_{x\sim p}[-logP(x)] = E_{x\sim p}[log\frac{P(x)}{Q(x)}] = KL(P||Q)$
Here the empirical distribution for each data point simply assigns probability 1 to the class of that data point, and 0 to all other classes.namely p only equals 1 once. **notice here P is empirical distribution from network, it's known. what we only optimize is Q!**



5.2.1 Loss function example

Center Loss[8]

combine center loss with softmax loss, let the classification not only separable but also discriminable.

$$-\sum_{i=1}^m log\frac{e^{W_{y_i}^T x_i + b_{y_i}}}{\sum_{j=1}^n e^{W_{y_j}^T x_i + b_{y_j}}} + \frac{\lambda}{2}\sum_{i=1}^m ||x_i - c_{y_i}||_2^2$$

just assume the last layer of the neural network is full connected network. before the neural network, the data have already become linear separable. so we can have the concept of "center". x_i is the former layer output of the last layer!

this is a softmax loss function,which actually is a cross entropy loss function:

$$-\sum_{i=1}^m log\frac{e^{W_{y_i}^T x_i + b_{y_i}}}{\sum_{j=1}^n e^{W_{y_j}^T x_i + b_{y_j}}}$$

5.3 WitchCraft

5.3.1 Awesome Maxout

MaxOut is Max pooling over channels
Maxout before softmax to boost quality. FC to 4* nr_class -> Maxout(4) -> Softmax or 2*nr_class

5.3.2 Softmax

$$y_i = \frac{exp(x_i)}{\sum_j exp(x_j)}$$

beta is reciprocal of temperature

$$y_i = \frac{exp(-\beta x_i)}{\sum_j exp(-\beta x_j)}$$

1, 2, -3 => 0.4, 0.6, 0.1
10, 20, -30 ==> 0, 1, 0

5.3.3 ResNet Pre Activation[4]

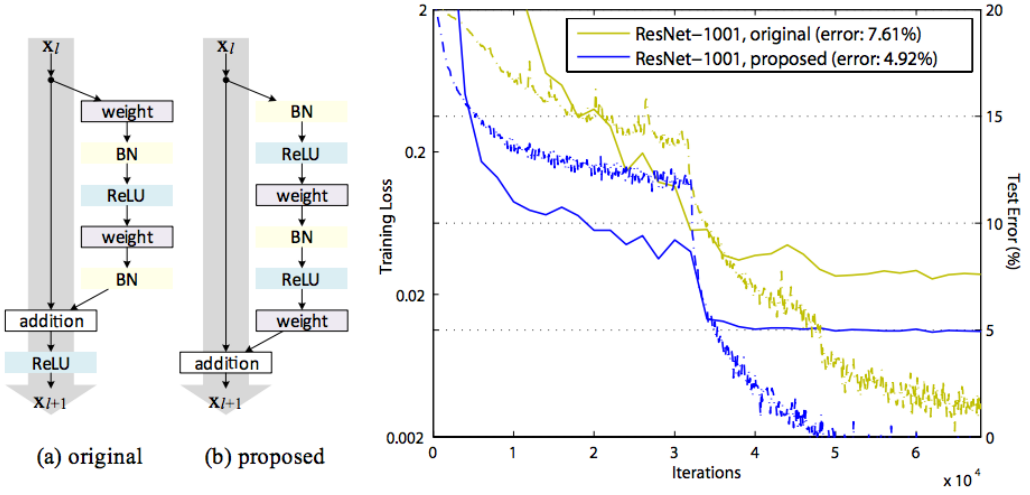


Figure 1. Left: (a) original Residual Unit in [1]; (b) proposed Residual Unit. The grey arrows indicate the easiest paths for the information to propagate, corresponding to the additive term "$\mathbf{x}_l$" in Eqn.(4) (forward propagation) and the additive term "1" in Eqn.(5) (backward propagation). **Right**: training curves on CIFAR-10 of **1001-layer** ResNets. Solid lines denote test error (y-axis on the right), and dashed lines denote training loss (y-axis on the left). The proposed unit makes ResNet-1001 easier to train.

let the scale to be very little at beginning, then restore it intermediately. the training will much more faster!!

5.4 How to back Propagation

how to deal with zero padding in resnet(happened when downsampling)

notice: one solution is to do zero padding so that "upsampling" the feature map to the larger one. another solution is projection shortcut.

forward-passing: just do zero padding.
backward-passing: in fact just no change. because the variable is convolution, the feature map is a constant!!!

how to deal with 2*2 pooling

TODO

6 R-PCA

RPCA problem:
$min_{A,E}\ ||A||_* + \lambda||E||_1$
S.t D=A+E

RPCA dual problem:

Augmented Lagrangian is :
$A_t(A, E; \Lambda) = min_{A,E} L(A, E; \Lambda)$
$= min_{A,E}||A||_* + \Lambda||E||_1 + < \Lambda, D - A - E >$
$= min_A||A||_* - < \Lambda, A > +$
$min_E \lambda||E||_1 - < \Lambda, E > + < \Lambda, D >$

both of the sub-problem is conjugate function,according to the property of conjugate function :

$A_t(A, E; \Lambda) =< \Lambda, D >$
$S.t \quad ||\Lambda||_2 \leq 1, ||\Lambda||_\infty \leq \lambda$

so the dual problem is:
$max_\Lambda \quad < \Lambda, D >$
$S.t \quad ||\Lambda||_2 \leq 1, ||\Lambda||_\infty \leq \lambda$

6.1 Solve RPCA by ADMM

the ADMM sub-problem is:
A-sub-problem:
$A_{k+1} = argmin_A||A||_* + \frac{\beta}{2}||D - A - E_k + \Lambda_k/\beta||_F^2$

E-sub-problem:
$E_{k+1} = argmin_E \lambda||E||_1 + \frac{\beta}{2}||D - A_{k+1} - E + \Lambda_k||_F^2$

E-sub-problem has closed-form solution as follows:
$E_{k+1} = S_{\lambda\beta^{-1}}(D - A_k + \Lambda_k/\beta)$.

$S_\epsilon = sgn(x)max(|x| - \epsilon, 0)$,which is the same form as shrinkage.

A-sub-problem has a closed-form solution offered by Singular Value Thresholding(SVT):suppose that the SVD of $W = D - E_k + \Lambda_k/\beta_k$ is $W = U\Sigma V^T$,then the optimal solution is $A = US_{\beta^{-1}}(\Sigma)V^T$.

6.2 Adaptive Penalty for ADMM

Lin et al.[6] suggest updating the penalty parameter β as follows:
$\beta_{k+1} = min(\beta_{max}, \rho\beta_k)$
where ρ_{max} is an upper bound of $\{\beta_k\}$. the value of ρ is defined as:

$$\rho = \begin{cases} \rho_0 & if \frac{\beta_k max(\sqrt{\eta_A}||x_{k+1} - x_k||_2, \sqrt{\eta_B}||y_{k+1} - y_k||_2)}{||c||_2} < \epsilon_2 \\ 1 & otherwise \end{cases}$$

where η_A, η_B is linearized Taylor second-order factor.

7 SFM

https://www.robots.ox.ac.uk/~vgg/hzbook/hzbook2/HZepipolar.pdf

- $\mathtt{F}$ is a rank 2 homogeneous matrix with 7 degrees of freedom.
- **Point correspondence**: If $\mathbf{x}$ and $\mathbf{x}'$ are corresponding image points, then $\mathbf{x}'^{\mathsf{T}}\mathtt{F}\mathbf{x} = 0$.
- **Epipolar lines**:
  - $\mathbf{l}' = \mathtt{F}\mathbf{x}$ is the epipolar line corresponding to $\mathbf{x}$.
  - $\mathbf{l} = \mathtt{F}^{\mathsf{T}}\mathbf{x}'$ is the epipolar line corresponding to $\mathbf{x}'$.
- **Epipoles**:
  - $\mathtt{F}\mathbf{e} = \mathbf{0}$.
  - $\mathtt{F}^{\mathsf{T}}\mathbf{e}' = \mathbf{0}$.
- **Computation from camera matrices** $\mathtt{P}, \mathtt{P}'$:
  - General cameras,
    $\mathtt{F} = [\mathbf{e}']_\times \mathtt{P}'\mathtt{P}^+$, where $\mathtt{P}^+$ is the pseudo-inverse of $\mathtt{P}$, and $\mathbf{e}' = \mathtt{P}'\mathbf{C}$, with $\mathtt{P}\mathbf{C} = \mathbf{0}$.
  - Canonical cameras, $\mathtt{P} = [\mathtt{I} \mid \mathbf{0}]$, $\mathtt{P}' = [\mathtt{M} \mid \mathbf{m}]$,
    $\mathtt{F} = [\mathbf{e}']_\times \mathtt{M} = \mathtt{M}^{-\mathsf{T}}[\mathbf{e}]_\times$, where $\mathbf{e}' = \mathbf{m}$ and $\mathbf{e} = \mathtt{M}^{-1}\mathbf{m}$.
  - Cameras not at infinity $\mathtt{P} = \mathtt{K}[\mathtt{I} \mid \mathbf{0}]$, $\mathtt{P}' = \mathtt{K}'[\mathtt{R} \mid \mathbf{t}]$,
    $\mathtt{F} = \mathtt{K}'^{-\mathsf{T}}[\mathbf{t}]_\times \mathtt{R}\mathtt{K}^{-1} = [\mathtt{K}'\mathbf{t}]_\times \mathtt{K}'\mathtt{R}\mathtt{K}^{-1} = \mathtt{K}'^{-\mathsf{T}}\mathtt{R}\mathtt{K}^{\mathsf{T}}[\mathtt{K}\mathtt{R}^{\mathsf{T}}\mathbf{t}]_\times$.

Figure 1: Summary of Fundamental matrix properties

8 Mainfold Learning

http://www.cad.zju.edu.cn/reports/%C1%F7%D0%CE%D1%A7%CF%B0.pdf

8.1 Laplace Matrix

8.2 Normalized Cut

8.3 Linear Dimension Reduction

PCA,CDMS,RP

8.3.1 PCA

let $X = (X_1, X_2, X_3, ..., X_p)^T$, the covariance matrix is:

$$\Sigma = (\sigma_{ij})_{p*p} = E[(X - E(x))(X - E(X))^T]$$

which is a p-order semi-positive definite matrix. let $l_i = (l_{i1}, l_{i2}, l_{i3}, ..., l_{ip})^T$ be a column vector. consider the linear transformation:

$$Y = L^T X$$

$Y = (Y_1, Y_2, ..., Y_p)^T, L = (l_1, l_2, ..., l_p)$. we have:

$$Var(Y_i) = Var(l_i^T X) = l_i^T \Sigma l_i$$

$$Cov(Y_i, Y_j) = Cov(l_i X, l_j X) = l_i^T \Sigma l_j$$

the above formulation is easy to understand, where Y_i, X is variable!l_i is a constant!
just recall the one-dimension circumstance: variance(3X) = 9 variance(X),covariance(2X,3X) = 6 variance(X).

the final optimization problem is:

$$\begin{aligned} max_{l_1}\ & Var(Y_1) = l_1^T \Sigma l_i \\ \text{S.t}\ & l_1^T l_1 = 1 \end{aligned}$$

the optimization of l_2:

$$\begin{aligned} max_{l_2}\ & Var(Y_2) = l_2^T \Sigma l_i \\ \text{S.t}\ & l_2^T l_2 = 1 \\ & l_1 \Sigma l_2 = 0 \end{aligned}$$

the optimization of l_3:

$$\begin{aligned} max_{l_3}\ & Var(Y_3) = l_3^T \Sigma l_i \\ \text{S.t}\ & l_3^T l_3 = 1 \\ & l_1 \Sigma l_3 = 0 \\ & l_2 \Sigma l_3 = 0 \end{aligned}$$

8.4 NonLinear Dimension Reduction

KPCA,ISOMAP,LLE,LSTA,LPCA, LE(Laplacian EigenMaps) , Diffusion Maps, MVU

9 DCT

`http://eeweb.poly.edu/~yao/EE3414/ImageCoding_DCT.pdf`

1D Unitary Transform

Consider the N – point signal $s(n)$ as an N - dimensional vector

$$\mathbf{s} = \begin{bmatrix} s_0 \\ s_1 \\ \\ s_{N-1} \end{bmatrix}$$

The inverse transform says that $\mathbf{s}$ can be represented as the sum of N basis vectors

$\mathbf{s} = t_0\mathbf{u}_0 + t_1\mathbf{u}_1 + \ldots + t_{N-1}\mathbf{u}_{N-1}$

where $\mathbf{u}_k$ corresponds to the k - th transform kernel :

$$\mathbf{u}_k = \begin{bmatrix} u_{k,0} \\ u_{k,1} \\ \ldots \\ u_{k,N-1} \end{bmatrix}$$

The forward transform says that the expansion coefficient t_k can be determined by the inner product of $\mathbf{s}$ with $\mathbf{u}_k$:

$$t_k = (\mathbf{u}_k, \mathbf{s}) = \sum_{n=0}^{N-1} u_{k;n}^* s_n$$

1D Discrete Cosine Transform

- Can be considered "real" version of DFT
  - Basis vectors contain only co-sinusoidal patterns

DFT

$$u_{k,n} = \frac{1}{\sqrt{N}} \exp\left(j\frac{2\pi k}{N}n\right) = \frac{1}{\sqrt{N}}\left(\cos\left(\frac{2\pi k}{N}n\right) + j\sin\left(\frac{2\pi k}{N}n\right)\right)$$

$$\mathbf{u}_k = \frac{1}{\sqrt{N}}\begin{bmatrix} \cos\left(\frac{2\pi k}{N}0\right) \\ \cos\left(\frac{2\pi k}{N}1\right) \\ \dots \\ \cos\left(\frac{2\pi k}{N}(N-1)\right) \end{bmatrix} + j\frac{1}{\sqrt{N}}\begin{bmatrix} \sin\left(\frac{2\pi k}{N}0\right) \\ \sin\left(\frac{2\pi k}{N}1\right) \\ \dots \\ \sin\left(\frac{2\pi k}{N}(N-1)\right) \end{bmatrix}$$

DCT

$$u_{k,n} = \alpha(k)\cos\left(\frac{\pi k}{2N}(2n+1)\right)$$

$$\alpha(0) = \sqrt{\frac{1}{N}}, \alpha(k) = \sqrt{\frac{2}{N}}, k = 1,2,\dots,N-1$$

$$\mathbf{u}_k = \alpha(k)\begin{bmatrix} \cos\left(\frac{\pi k}{2N}1\right) \\ \cos\left(\frac{\pi k}{2N}3\right) \\ \dots \\ \cos\left(\frac{\pi k}{2N}(2N+1)\right) \end{bmatrix}$$

Example: 4-point DCT

Using $u_{k,n} = \alpha(\mathrm{k})\cos\left(\frac{\mathrm{k}\pi}{2*4}(2n+1)\right), \alpha(0) = \sqrt{\frac{1}{4}} = \frac{1}{2}, \alpha(\mathrm{k}) = \sqrt{\frac{2}{4}} = \sqrt{\frac{1}{2}}, \mathrm{k} \neq 0,$

1D DCT basis are: $\mathbf{u}_0 = \frac{1}{2}\begin{bmatrix}1\\1\\1\\1\end{bmatrix}; \mathbf{u}_1 = \frac{1}{\sqrt{2}}\begin{bmatrix} \cos\left(\frac{\pi}{8}\right) \\ \cos\left(\frac{3\pi}{8}\right) \\ \cos\left(\frac{5\pi}{8}\right) \\ \cos\left(\frac{7\pi}{8}\right) \end{bmatrix} = \frac{1}{\sqrt{2}}\begin{bmatrix}0.9239\\0.3827\\-0.3827\\-0.9239\end{bmatrix}; \mathbf{u}_2 = \frac{1}{\sqrt{2}}\begin{bmatrix} \cos\left(\frac{\pi}{4}\right) \\ \cos\left(\frac{3\pi}{4}\right) \\ \cos\left(\frac{5\pi}{4}\right) \\ \cos\left(\frac{7\pi}{4}\right) \end{bmatrix} = \frac{1}{2}\begin{bmatrix}1\\-1\\-1\\1\end{bmatrix}; \mathbf{u}_3 = \frac{1}{\sqrt{2}}\begin{bmatrix} \cos\left(\frac{3\pi}{8}\right) \\ \cos\left(\frac{9\pi}{8}\right) \\ \cos\left(\frac{15\pi}{8}\right) \\ \cos\left(\frac{21\pi}{8}\right) \end{bmatrix} = \frac{1}{\sqrt{2}}\begin{bmatrix}0.3827\\-0.9239\\0.9239\\-0.3827\end{bmatrix}$

For $\mathbf{s} = \begin{bmatrix}2\\4\\5\\3\end{bmatrix}$, determine the transform coefficients t_k. Also determine the reconstructed vector from all coefficients and two largest coeffcieints.

2D Separable Transform

Consider the $M \times N$ – point image $\mathbf{S}$ as a $M \times N$ - dimensional array (matrix)

$$\mathbf{S} = \begin{bmatrix} S_{0,0} & S_{0,1} & \dots & S_{0,N-1} \\ S_{1,0} & S_{1,1} & \dots & S_{1,N-1} \\ \dots & \dots & \dots & \dots \\ S_{M-1,0} & S_{M-1,1} & \dots & S_{M-1,N-1} \end{bmatrix}$$

The inverse transform says that $\mathbf{s}$ can be represented as the sum of $M \times N$ basis images

$$\mathbf{S} = T_{0,0}\mathbf{U}_{\mathbf{0},0} + T_{0,1}\mathbf{U}_{0,1} + \dots + T_{M-1,N-1}\mathbf{U}_{M-1,N-1}$$

where $\mathbf{U}_{k,l}$ corresponds to the (k,l) - th transform kernel :

$$\mathbf{U}_{k,l} = \mathbf{u}_k(\mathbf{u}_l)^T = \begin{bmatrix} u_{k,0} \\ u_{k,1} \\ \dots \\ u_{k,N-1} \end{bmatrix} \begin{bmatrix} u^*_{l,0} & u^*_{l,1} & \dots & u^*_{l,N-1} \end{bmatrix} = \begin{bmatrix} u_{k,0}u^*_{l,0} & u_{k,0}u^*_{l,1} & \dots & u_{k,0}u^*_{l,N-1} \\ u_{k,1}u^*_{l,0} & u_{k,1}u^*_{l,1} & \dots & u_{k,1}u^*_{l,N-1} \\ \dots & \dots & \dots & \dots \\ u_{k,N-1}u^*_{l,0} & u_{k,N-1}u^*_{l,1} & \dots & u_{k,N-1}u^*_{l,N-1} \end{bmatrix}$$

The forward transform says that the expansion coefficient S_k can be determined by the inner product of $\mathbf{S}$ and $\mathbf{U}_{k,l}$:

$$T_{k,l} = (\mathbf{U}_{k,l}, \mathbf{S}) = \sum_{m=0}^{M-1}\sum_{n=0}^{N-1} U^*_{k,l;m,n} S_{m,n}$$

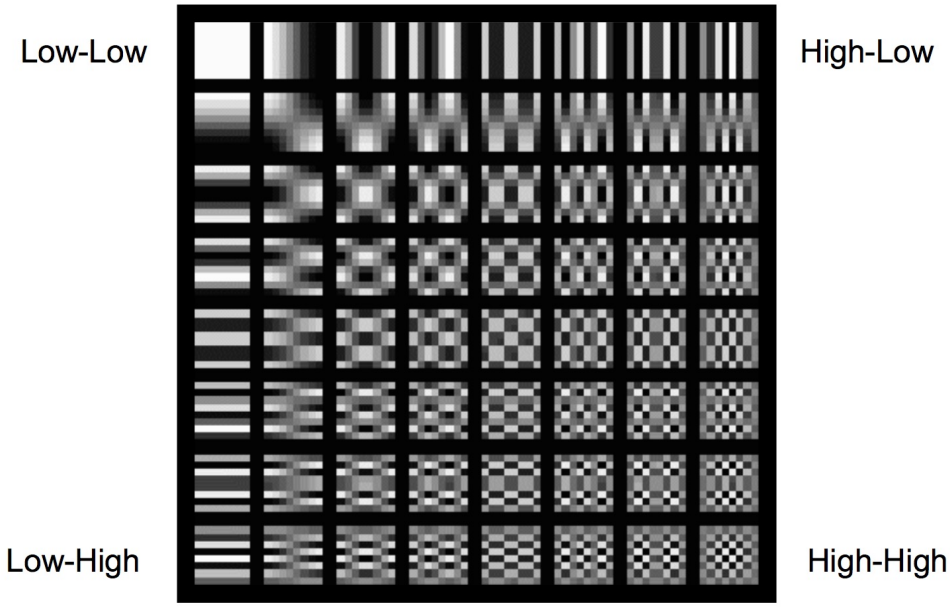
- Basis image = outer product of 1D DCT basis vector

$$\mathbf{u}_{k;N} = \alpha(k) \begin{bmatrix} \cos\left(\frac{\pi k}{2N}1\right) \\ \cos\left(\frac{\pi k}{2N}3\right) \\ \dots \\ \cos\left(\frac{\pi k}{2N}(2N+1)\right) \end{bmatrix}, \quad \alpha(0) = \sqrt{\frac{1}{N}}, \alpha(k) = \sqrt{\frac{2}{N}}, k = 1,2,\dots,N-1$$

$$\mathbf{U}_{k,l;M,N} = \mathbf{u}_{k;M}(\mathbf{u}_{l;N})^T$$

$$= \alpha(k)\alpha(l) \begin{bmatrix} \cos\left(\frac{k\pi}{2M}1\right)\cos\left(\frac{l\pi}{2N}1\right) & \cos\left(\frac{k\pi}{2M}1\right)\cos\left(\frac{l\pi}{2N}3\right) & \dots & \cos\left(\frac{k\pi}{2M}1\right)\cos\left(\frac{l\pi}{2N}(2N+1)\right) \\ \cos\left(\frac{k\pi}{2M}3\right)\cos\left(\frac{l\pi}{2N}1\right) & \cos\left(\frac{k\pi}{2M}3\right)\cos\left(\frac{l\pi}{2N}3\right) & \dots & \cos\left(\frac{k\pi}{2M}3\right)\cos\left(\frac{l\pi}{2N}(2N+1)\right) \\ \dots & \dots & \dots & \\ \cos\left(\frac{k\pi}{2M}(2M+1)\right)\cos\left(\frac{l\pi}{2N}1\right) & \cos\left(\frac{k\pi}{2M}(2M+1)\right)\cos\left(\frac{l\pi}{2N}3\right) & \dots & \cos\left(\frac{k\pi}{2M}(2M+1)\right)\cos\left(\frac{l\pi}{2N}(2N+1)\right) \end{bmatrix}$$

Basis Images of 8x8 DCT



Example: 4x4 DCT

Using $u_{k,n} = \alpha(\mathrm{k})\cos\left(\frac{\mathrm{k}\pi}{2*4}(2n+1)\right), \alpha(0) = \sqrt{\frac{1}{4}} = \frac{1}{2}, \alpha(\mathrm{k}) = \sqrt{\frac{2}{4}} = \sqrt{\frac{1}{2}}, \mathrm{k} \neq 0,$

1D DCT basis are: $\mathbf{u}_0 = \frac{1}{2}\begin{bmatrix}1\\1\\1\\1\end{bmatrix}; \mathbf{u}_1 = \frac{1}{\sqrt{2}}\begin{bmatrix}\cos\left(\frac{\pi}{8}\right)\\ \cos\left(\frac{3\pi}{8}\right)\\ \cos\left(\frac{5\pi}{8}\right)\\ \cos\left(\frac{7\pi}{8}\right)\end{bmatrix} = \frac{1}{\sqrt{2}}\begin{bmatrix}0.9239\\0.3827\\-0.3827\\-0.9239\end{bmatrix}; \mathbf{u}_2 = \frac{1}{\sqrt{2}}\begin{bmatrix}\cos\left(\frac{\pi}{4}\right)\\ \cos\left(\frac{3\pi}{4}\right)\\ \cos\left(\frac{5\pi}{4}\right)\\ \cos\left(\frac{7\pi}{4}\right)\end{bmatrix} = \frac{1}{2}\begin{bmatrix}1\\-1\\-1\\1\end{bmatrix}; \mathbf{u}_3 = \frac{1}{\sqrt{2}}\begin{bmatrix}\cos\left(\frac{3\pi}{8}\right)\\ \cos\left(\frac{9\pi}{8}\right)\\ \cos\left(\frac{15\pi}{8}\right)\\ \cos\left(\frac{21\pi}{8}\right)\end{bmatrix} = \frac{1}{\sqrt{2}}\begin{bmatrix}0.3827\\-0.9239\\0.9239\\-0.3827\end{bmatrix}$

using $\mathbf{U}_{kl} = \mathbf{u}_k(\mathbf{u}_l)^T$ yields:

$\mathbf{U}_{0,0} = \frac{1}{4}\begin{bmatrix}1&1&1&1\\1&1&1&1\\1&1&1&1\\1&1&1&1\end{bmatrix}$, $\mathbf{U}_{0,2} = \frac{1}{4}\begin{bmatrix}1&-1&-1&1\\1&-1&-1&1\\1&-1&-1&1\\1&-1&-1&1\end{bmatrix}$, $\mathbf{U}_{2,0} = \frac{1}{4}\begin{bmatrix}1&1&1&1\\-1&-1&-1&-1\\-1&-1&-1&-1\\1&1&1&1\end{bmatrix}$, $\mathbf{U}_{2,2} = \frac{1}{4}\begin{bmatrix}1&-1&-1&1\\-1&1&1&-1\\-1&1&1&-1\\1&-1&-1&1\end{bmatrix}$, ...

What Should You Know

- How to perform 2D DCT: forward and inverse transform
  - Manual calculation for small sizes, using inner product notation
  - Using Matlab: dct2, idct2
- Why DCT is good for image coding
  - Real transform, easier than DFT
  - Most high frequency coefficients are nearly zero and can be ignored
  - Different coefficients can be quantized with different accuracy based on human sensitivity
- How to quantize DCT coefficients
  - Varying stepsizes for different DCT coefficients based on visual sensitivity to different frequencies
  - A quantization matrix specifies the default quantization stepsize for each coefficient
  - The matrix can be scaled using a user chosen parameter (QP) to obtain different trade-offs between quality and size

10 KL-divergence Application

LINE: Large-scale Information Network Embedding `https://arxiv.org/pdf/1503.03578.pdf`

11 GAN

optimization problem:
$min_G max_D V(D,G) = E_{x\sim p_{data}}[logD(x)] + E_{z\sim p_z(z)}[log(1-D(G(z)))]$

notation:
p_g:generator's distribution
p_{data}: training data's distribution
$p_g(x)$: the probability that x comes from generator
$p_{data}(x)$: the probability that x comes from data
$p_z(z)$: a input noise variable
$G(z;\theta_g)$: generator network,input is z,parameter is θ_g
$D(x;\theta_d)$: discriminator network, input is x, parameter is θ_d
$D^*_G(x)$: optimal discriminator for any given generator G

K-L divergence:
$KL(p_1||p_2) = E_{x\sim p_1} log\frac{p_1}{p_2} = \int_x p_1(x) log(\frac{p_1(x)}{p_2(x)})dx$

JS divergence:
$JS(p_1||p_2) = \frac{1}{2}KL(p_1||\frac{p_1+p_2}{2}) + \frac{1}{2}KL(p_2||\frac{p_1+p_2}{2})$

a important property of JS divergence:

> the Jensen–Shannon divergence between two distributions is always non-negative and zero only when they are equal.

Firstly, let's image the generator is given!!!
just let G(z) = x, z$\sim p_z(Z)$, x is the G function operating on z, so x$\sim p_g$
therefore, $E_{z\sim p_z(z)}[log(1-D(G(z)))]$ can be converted to $E_{x\sim p_g(z)}[log(1-D(x))]$

given any generator G, we should maximize the quantity V(G,D):
$V(G,D) = \int_x [p_{data}(x) log D(x) + p_g(x)(1-D(x))]dx$
according to the definition calculus, we can calculate the optimum inner the calculus, which induces the problem to a log(y) + b log(1-y), the optimal value is : $\frac{a}{a+b}$. thus, the optimal discriminator is
$D^*_G(x) = \frac{p_{data}(x)}{p_{data}(x)+p_g(x)}$

when D(x) get optimal, we can convert C(G) as:
$C(G) = -log(4) + log\frac{p_{data}}{\frac{p_{data}+p_d}{2}} + log\frac{p_d}{\frac{p_{data}+p_d}{2}}$
$= -log(4) + KL(p_{data}||p_d) + KL(p_d||p_{data})$
$= -log(4) + JS(p_{data}||p_d)$

Since the Jensen–Shannon divergence between two distributions is always non-negative and zero only when they are equal. we can show that $C(G) = -log(4)$ is the global minimum of C(G) and that the only solution is $p_g = p_{data}$, i.e., the generative model perfectly replicating the data generating process.

11.1 Why is maxD then minG

reference in Boyed's Book 5.4.3

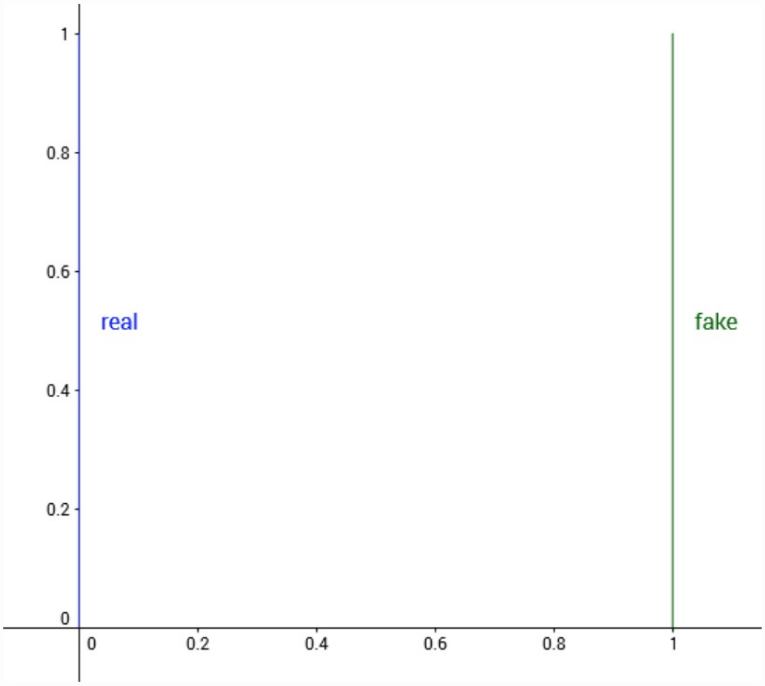
11.2 WGAN

`http://www.alexirpan.com/2017/02/22/wasserstein-gan.html`

Earth Mover (EM) or Wasserstein distance:
$W(P_r, P_g) = inf_{r\in\Pi(P_r,P_g)} E_{(x,y)\sim r}[||x-y||]$

Wasserstein distance advantage:
even two distribution have no overlap, Wasserstein distance can also describe the mutual distance .

Consider probability distributions defined over $\mathbb{R}^2$. Let the true data distribution be $(0, y)$, with y sampled uniformly from $U[0, 1]$. Consider the family of distributions P_θ, where $P_\theta = (\theta, y)$, with y also sampled from $U[0, 1]$.



Real and fake distribution when $\theta = 1$

We'd like our optimization algorithm to learn to move θ to 0, As $\theta \to 0$, the distance $d(P_0, P_\theta)$ should decrease. But for many common distance functions, this doesn't happen.

- Total variation: For any $\theta \neq 0$, let $A = \{(0, y) : y \in [0, 1]\}$. This gives

$$\delta(P_0, P_\theta) = \begin{cases} 1 & \text{if } \theta \neq 0\,, \\ 0 & \text{if } \theta = 0\,. \end{cases}$$

- KL divergence and reverse KL divergence: Recall that the KL divergence $KL(P\|Q)$ is $+\infty$ if there is any point (x, y) where $P(x, y) > 0$ and $Q(x, y) = 0$. For $KL(P_0\|P_\theta)$, this is true at $(\theta, 0.5)$. For $KL(P_\theta\|P_0)$, this is true at $(0, 0.5)$.

$$KL(P_0\|P_\theta) = KL(P_\theta\|P_0) = \begin{cases} +\infty & \text{if } \theta \neq 0\,, \\ 0 & \text{if } \theta = 0\,, \end{cases}$$

- Jenson-Shannon divergence: Consider the mixture $M = P_0/2 + P_\theta/2$, and now look at just one of the KL terms.

$$KL(P_0\|M) = \int_{(x,y)} P_0(x, y) \log \frac{P_0(x, y)}{M(x, y)}\, dy\, dx$$

For any x, y where $P_0(x, y) \neq 0$, $M(x, y) = \frac{1}{2}P_0(x, y)$, so this integral works out to $\log 2$. The same is true of $KL(P_\theta\|M)$, so the JS divergence is

$$JS(P_0, P_\theta) = \begin{cases} \log 2 & \text{if } \theta \neq 0\,, \\ 0 & \text{if } \theta = 0\,, \end{cases}$$

- Earth Mover distance: Because the two distributions are just translations of one another, the best way transport plan moves mass in a straight line from $(0, y)$ to (θ, y). This gives $W(P_0, P_\theta) = |\theta|$

This example shows that there exist sequences of distributions that don't converge under the JS, KL, reverse KL, or TV divergence, but which do converge under the EM distance.

This example also shows that for the JS, KL, reverse KL, and TV divergence, there are cases where the gradient is always 0. This is especially damning from an optimization perspective - any approach that works by taking the gradient $\nabla_\theta d(P_0, P_\theta)$ will fail in these cases.

Admittedly, this is a contrived example because the supports are disjoint, but the paper points out that when the supports are low dimensional manifolds in high dimensional space, it's very easy for the intersection to be measure zero, which is enough to give similarly bad results.

The calculation of Original Wasserstein distance is very difficult, A result from Kantorovich-Rubinstein duality shows Wasserstein is equivalent to:
$W(P_r, P_\theta) = sup_{||f||_L \leq K} E_{x \sim P_r}[f(x)] - E_{x \sim P_\theta}[f(x)]$
which format is very similar to the original GAN.

12 Reinforcement Learning

Markov Decision Process (**MDP**) is tuple$(\mathcal{S}, \mathcal{A}, \mathcal{P}, r, \gamma)$:

r is a reward function,$r(s, a, s^{'})$

γ is a discount factor

$\mathcal{P}$ is the transition probability distribution:
probability from state s with action a to state $s^{'}$: $P(s^{'}|s, a)$

$\mathcal{S}$ is a finite set of states.

$\mathcal{A}$ is a finite set of actions.

Markov Property:
$P(s_{t+1}|s_t) = P(s_{t+1}|s_1, ..., s_t)$

Stochastic Policy:
$\pi(a|s) = P(a_s = a|s_t = s), \pi(a|s)$ here is a probability,we can often see $\pi(s)$, which returns a, means under policy π and state s, you should better take action a.

Value function:
$G_t = r_{t+1} + \gamma r_{t+2} + \gamma^2 r_{t+3} + ... = \sum_{k=0}^{\infty} \gamma^k r_{t+k+1}$

State-value function:
$V_\pi(s) = E_\pi(G_t|s_t = s)$

Action-value function:
$Q_\pi(s, a) = E_\pi(G_t|s_t = s, a_t = a)$

Bellman Equation:
$V_\pi(s) = E_\pi(r_{t+1} + \gamma r_{t+2} + \gamma^2 r_{t+3} + ...|s_t = s)$
$= E_\pi(r_{t+1} + \gamma(r_{t+2} + \gamma r_{t+3} + ...)|s_t = s)$
$= E_\pi(r_{t+1} + \gamma G_{t+1}|s_t = s)$
$= E_\pi(r_{t+1} + \gamma V_\pi(s_{t+1})|s_t = s)$

For state-value function, Bellman Equation can be written as:
$V_\pi(s) = E_\pi(r_{t+1} + \gamma V_\pi(s_{t+1})|s_t = s)$
$= \sum_{\alpha \in A} \pi(a|s) \sum_{s' \in S} P(s^{'}|s, a)[r(s, a, s^{'}) + \gamma V_\pi(s^{'})]$
$r(s, a, s^{'})$ is same with r_{t+1} to some extent.

For action-value function, Bellman Equation can be written as:
$Q_\pi(s, a) = E_\pi(r(s, a, s^{'}) + \gamma Q_\pi(s^{'}, a^{'})|s, a)$
$= \sum_{s' \in S} P(s^{'}|s, a)[r(s, a, s^{'}) + \gamma \sum_{a' \in A} \pi(a^{'}, s^{'}) Q_\pi(s^{'}, a^{'}))]$

Normally, we just assume the $\pi(a|s), p(s^{'}|s, a), r(s, a, s^{'})$ are known(namely the MDP is known). so we can solve the linear equation above. however, when data become huge, it is not feasible to solve the Bellman Equation directly.

optimal value function:

the optimal state-value function $V_*(s)$ is :
$V_*(s) = max_\pi V_\pi(s)$

the optimal action-value function $Q_*(s, a)$ is:
$Q_*(s, a) = max_\pi Q_\pi(s, a)$

$Q^*(s, a) = max_\pi Q^\pi(s, a)$
following important properties:

$Q^*(s, a) = E[r_{t+1} + \gamma V^*(s_{t+1})|s_t = s, a_t = a]$

$V^*(s) = max_{a \in \mathcal{A}} Q^*(s,a)$
$Q^*(s,a) = \sum_{s' \in \mathcal{S}} p(s'|s,a)[r(s,a,s') + \gamma V^*(s')]$

The goal for any MDP is finding the optimal value function.

Or equivalent an optimal policy π^* for any policy $\pi, V_{\pi^*}(s) \geq V_\pi(s), \forall s \in S$

how do we take action after we obtain the optimal $V^*(s)$? if we known the $p(s'|s,a), r(s,a,s')$, then we can get: $Q^*(s,a) = \sum_{s' \in S} p(s'|s,a)[R(s,a,s') + \gamma V^*(s')]$. after $Q^*(s,a)$ is acquired,it's more trivial to take action a when it's at state s.

Relation between Q and V Functions

reference: `http://www.cs.cmu.edu/~mgormley/courses/10601-s17/slides/lecture26-ri.pdf`

Q from V:
$Q^\pi(s,a) = \sum_{s' \in S} P(s'|s,a)[R(s,a,s') + \gamma V^\pi(s')]$

V from Q:
$V^\pi(s) = \sum_{a \in \mathcal{A}} \pi(a|s) Q^\pi(s,a)$

V and Q
$V^\pi(s) = \sum_{a \in \mathcal{A}} \pi(a|s) \sum_{s' \in \mathcal{S}} p(s'|s,a)[r(s,a,s') + \gamma V^\pi(s')]$

Q and V
$Q^\pi(s,a) = \sum_{s' \in \mathcal{S}} p(s'|s,a)[r(s,a,s') + \gamma \sum_{a' \in \mathcal{A}} \pi(a'|s') Q^\pi(s',a')]$

more complicated..
$V^\pi(s) = \sum_{a \in \mathcal{A}} \pi(a|s) \sum_{s' \in \mathcal{S}} p(s'|s,a)[r(s,a,s') + \gamma \sum_{a' \in \mathcal{A}} \pi(a'|s') Q^\pi(s',a')]$

mainly can classified into two main approaches:

Model based approaches:

First we will discuss methods that need to know the model:
$P(s' \mid s, a)$ and $R(s, a, s')$.

- ❑ **Policy Iteration**
- ❑ **Value Iteration**

Model-free approaches:

Then we will discuss "model-free" methods that do NOT need to know the model: $P(s' \mid s, a)$ and $R(s, a, s')$.

- ❑ **Monte Carlo Method**
- ❑ **TD Learning**

34

Figure 2: Structure of RL

12.1 Model-based Method

12.1.1 Policy Iteration

One drawback of policy iteration is that each iteration involves policy evaluation.

1. Initialization

$V(s) \in \mathbb{R}$ and $\pi(s) \in \mathcal{A}(s)$ for all $s \in \mathcal{S}$.
$\pi(s)$ is a deterministic policy.
$\delta > 0$ is a small threshold parameter.

2. Policy Evaluation

repeat
  $\Delta \leftarrow 0$
  for all $s \in \mathcal{S}$ **do**:
    $v \leftarrow V(s)$
    $a \leftarrow \pi(s)$
    $V(s) \leftarrow \sum_{s' \in \mathcal{S}} P(s'|s, a)[R(s, a, s') + \gamma V(s')]$
    $\Delta \leftarrow \max(\Delta, |v - V(s)|)$
  end for
until $\Delta < \delta$

3. Policy Improvement

$policyStable \leftarrow true$

for all $s \in \mathcal{S}$ **do**:

$b \leftarrow \pi(s)$

$\pi(s) \leftarrow \arg\max_{a \in \mathcal{A}(s)} \sum_{s' \in \mathcal{S}} P(s'|s,a)[R(s,a,s') + \gamma V(s')]$

if $b \neq \pi(s)$ **then**

$policyStable \leftarrow false$

end if

end for

if $policyStable$ **then**

$STOP$

else

Go to 2 (Policy Evaluation)

end if

Policy Improvement just improve the policy $\pi(s)$, then when we back to policy evaluation, the next action a is determined by the policy $\pi(s)$ which was updated in policy improvement. thus the relationship of policy improvement and policy evaluation is founded.

12.1.2 Value Iteration

Value Iteration

Main idea:

Use the Bellman equation of V^* instead of V^π

The greedy operator:

$$[T^*V](s) := \max_{a \in \mathcal{A}(s)} \sum_{s' \in \mathcal{S}} P(s'|s,a)[R(s,a,s') + \gamma V(s')]$$

V^* is the solution of $V = T^*(V)$ fixpoint iteration.

The value iteration update:

$k = 0$ and $V_0(s) \in \mathbb{R}$ for all $s \in \mathcal{S}$

repeat

for all $s \in \mathcal{S}$ **do**:

$V_{k+1}(s) \leftarrow \max_{a \in \mathcal{A}(s)} \sum_{s' \in \mathcal{S}} P(s'|s,a)[R(s,a,s') + \gamma V_k(s')]$

end for

$k \leftarrow k + 1$

until $V_k(\cdot)$ converged

37

1, value iteration converges to the true solution of Bellman optimal equations
2, Learn optimal value function directly, unlike policy iteration, there is no explicit policy.

12.2 Model-Free Method

12.2.1 on-policy TD

TD0

TD(n)

TDλ

12.2.2 on-policy MCMC

12.2.3 off-policy method

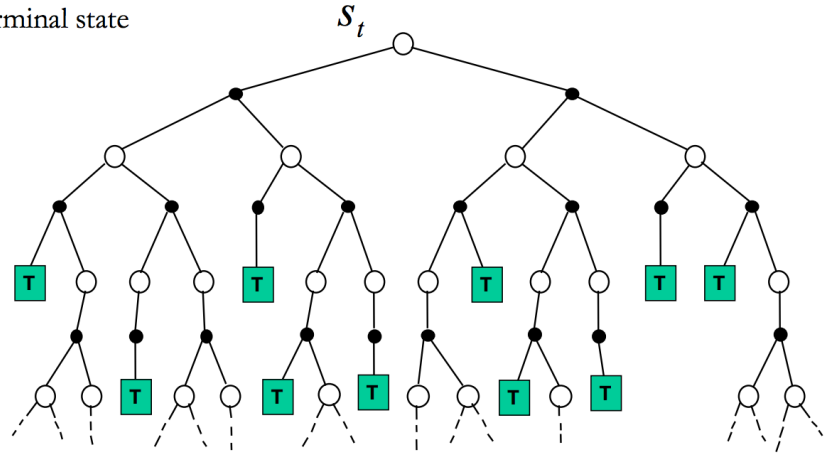
12.2.4 Comparisons: DP, MC, TD

Comparisons: DP, MC, TD

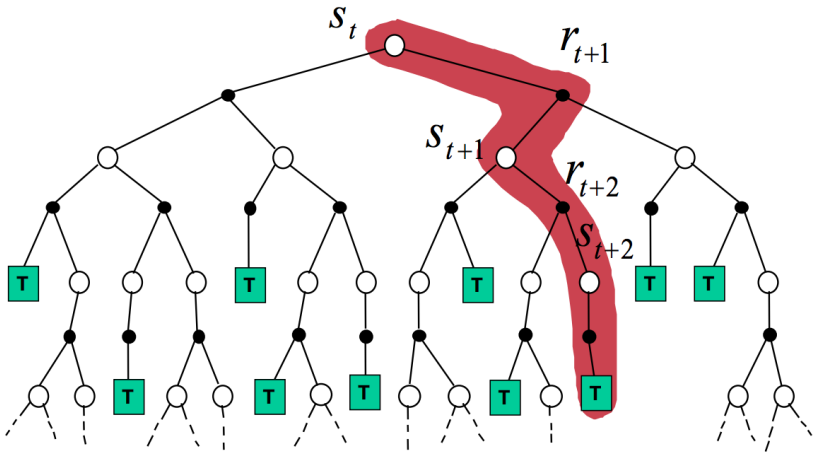
- They all estimate V^{π}
- DP: $V_k(s_t) \approx E_{\pi}\left(r_{t+1} + \gamma V_{k-1}(s_{t+1}) \mid s_t\right)$
  - Estimate comes from the Bellman equation
  - It needs to know the model
- TD: $V_k(s_t) \approx \left(r_{t+1} + \gamma V_{k-1}(s_{t+1})\right)$
  - Expectation is approximated with random samples
  - Doesn't need to wait for the end of the episodes.
- MC: $V_k(s_t) \approx R_t(s_t)$
  - Expectation is approximated with random samples
  - It needs to wait for the end of the episodes

MDP Backup Diagrams

- White circle: state
- Black circle: action
- T: terminal state

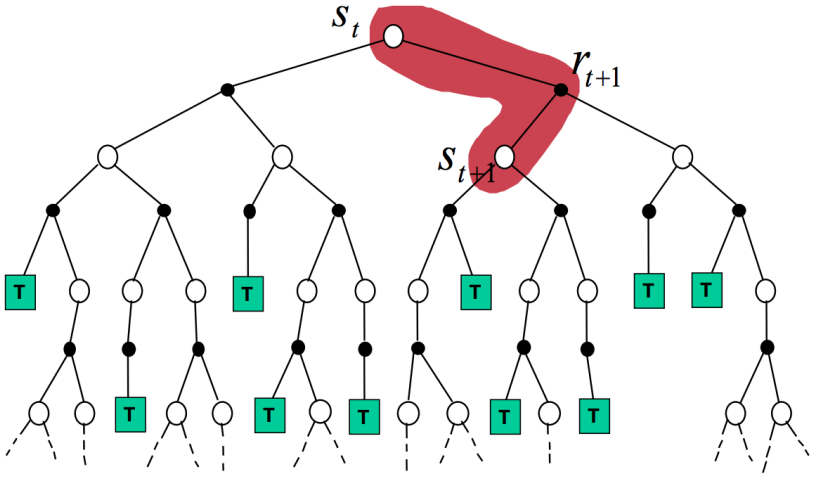


Monte Carlo Backup Diagram



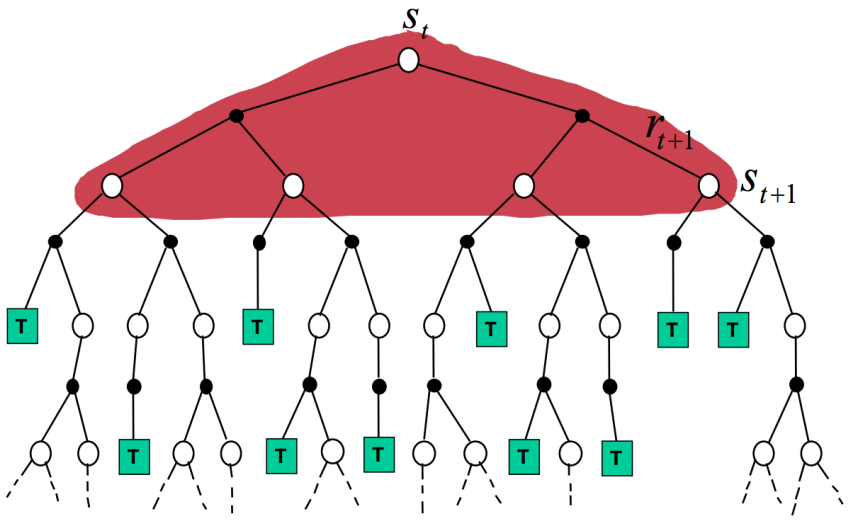
MC estimate: $V_k(s_t) := V_{k-1}(s_t) + \alpha_k \cdot (R_k(s_t) - V_{k-1}(s_t))$

Temporal Differences Backup Diagram



TD estimate: $V_k(s_t) := V_{k-1}(s_t) + \alpha_k \cdot \big((r_{t+1} + \gamma V_{k-1}(s_{t+1})) - V_{k-1}(s_t)\big)$

Dynamic Programming Backup Diagram



$$V^\pi(s) = \sum_{a \in \mathcal{A}} \pi(a|s) \sum_{s' \in \mathcal{S}} P(s'|s,a)[R(s,a,s') + \gamma V^\pi(s')]$$

50

13 RNN

standard RNN:
$a^{(t)} = b + Wh^{(t-1)} + Ux^{(t)}$
$h^{(t)} = tanh(a^{(t)})$
$o^{(t)} = c + Vh^{(t)}$
$y^{(t)} = softmax(o^{(t)})$

13.1 BPTT

13.2 LSTM

`http://colah.github.io/posts/2015-08-Understanding-LSTMs/`

standard LSTM:
$f_t = \sigma_g(W_f x_t + U_f h_{t-1} + b_f)$
$i_t = \sigma_g(W_i x_t + U_i h_{t-1} + b_i)$
$o_t = \sigma_g(W_o x_t + U_o h_{t-1} + b_o)$
$c_t = f_t \circ c_{t-1} + i_{t-1} \circ \sigma_c(W_c x_t + U_c h_{t-1} + b_c)$
$h_t = o_t \circ \sigma_h(c_t)$

f means forget gate, i means input gate, o means output gate h means hidden.

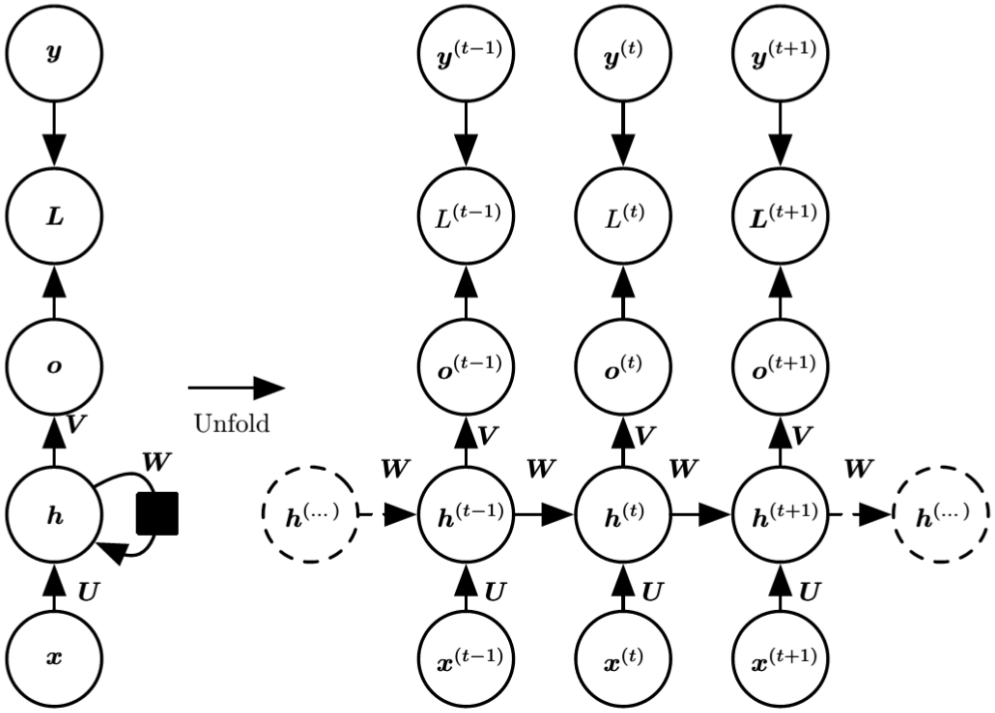
another equivalent writing style of LSTM is as follows, this writing style is equivalent to the original version,because $[x_t^T, h_{t-1}^T]^T$ means concat in row,and the W means W and U of original version, then utilize the Block Matrix Multiplication. most importantly,this style is more convenient to implement in the mainstream deep learning framework such as tensorflow,caffe etc.
$i_t = sigmoid(W_i[x_t^T, h_{t-1}^T]^T)$
$f_t = sigmoid(W_f[x_t^T, h_{t-1}^T]^T)$
$o_t = sigmoid(W_o[x_t^T, h_{t-1}^T]^T)$
$c_t = i_t \circ tanh(W_c[x_t^T, h_{t-1}^T]^T) + f_t \circ c_{t-1}$
$h_t = o_t \circ tanh(c_t)$

LSTM variants:

PeepHole:
$f_t = \sigma(W_f[c_{t-1}, h_{t-1}, x_t] + b_f)$
$i_t = \sigma(W_i[c_{t-1}, h_{t-1}, x_t] + b_i)$
$o_t = \sigma(W_o[c_t, h_{t-1}, x_t] + b_o)$

Gated Recurrent Unit (GRU):
$z_t = \sigma_g(W_z x_t + U_z h_{t-1} + b_z)$
$r_t = \sigma_g(W_r x_t + U_r h_{t-1} + b_r)$
$h_t = z_t \circ h_{t-1} + (1 - z_t) \circ \sigma_h(W_h x_t + U_h(r_t \circ h_{t-1}) + b_h)$

z_t : update gate
r_t : reset gate
h_t : output gate



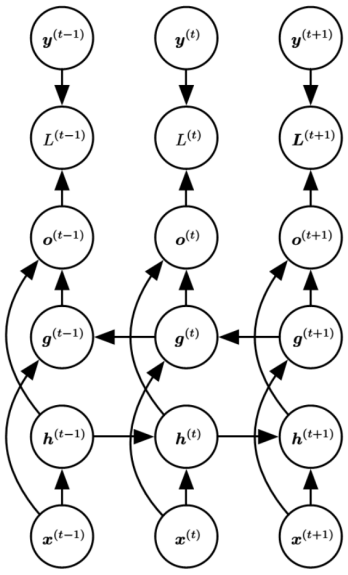
`https://cs224d.stanford.edu/lecture_notes/LectureNotes4.pdf`,the two RNNs can be imagined parallelized. the following is the structure of Bidirectional LSTM.

- In many applications, however, we want to output a prediction that may depend on the whole input sequence.

For example, in speech recognition, the correct interpretation of the current sound as a phoneme may depend on the next few phonemes because of co-articulation and may even depend on the next few words because of the linguistic dependencies between nearby words:

if there are two interpretations of the current word that are both acoustically plausible, we may have to look far into the future (and the past) to disambiguate them.

This is also true of handwriting recognition and many other sequence-to-sequence learning tasks



and several BiLSTM can also concatenated into a big LSTM network. (stacked LSTM)

13.3 Multidimension RNN

MDRNN[3] the most fundamental structure used for multi dimension data(image, video, fMRI). in this paper Multi direction MDRNN also was proposed by extending the conception of BiRNN. the forward pass MDRNN is as follows:

for $x_1 = 0$ to $X_1 - 1$ **do**
  for $x_2 = 0$ to $X_2 - 1$ **do**
    ...
    for $x_n = 0$ to $X_n - 1$ **do**
      initialize $a \leftarrow \sum_j in_j^{\mathbf{x}} w_{kj}$
      for $i = 1$ to n **do**
        if $x_i > 0$ **then**
          $a \leftarrow a + \sum_j h_j^{(x_1,...,x_i-1,...,x_n)} w_{kj}$
      $h_k^{\mathbf{x}} \leftarrow tanh(a)$

Algorithm 1: MDRNN Forward Pass

14 Algorithm

Longest Ordered Subsequence:

h(i) = max(h(j)) +1,where j<i and h(j) < h(i).
this is o(n) solution.
we can reach a o(nlog(n)) solution via setting up a array to save previous information.(this method is very tricky.).

Bipartite graph: Hungarian algorithm: `http://blog.csdn.net/hurmishine/article/details/52749670`

15 Determinantal Point Processes

the following introduction is copied from [7].

Determinantal Point Processes (DPPs) are discrete probability models over the subsets of a ground set of N items. They provide an elegant model to assign probabilities to an exponentially large sample, while permitting tractable (polynomial time) sampling and marginalization. They are often used to provide models that balance "diversity" and quality, characteristics valuable to numerous problems in machine learning and related areas.

suppose we have a ground set of N items $\mathcal{Y} = \{1, ..., N\}$, A discrete DPP over $\mathcal{Y}$ is a probability measure $\mathcal{P}$ on$2^{\mathcal{Y}}$ parametrized by a positive definite matrix K (the marginal kernel) such that $0 \leq K \leq I$, so that for any $Y \in \mathcal{Y}$ drawn from $\mathcal{P}$, the measure satisfies:

$$\forall A \subseteq \mathcal{Y}, P(A \subseteq Y) = det(K_A)$$

where K_A is the submatrix of K indexed by elements in A. if a DPP with marginal kernel K assigns nonzero probability to the empty set,the DPP can alternatively be parametrized by a positive definite matrix L (the DPP kernel) so that:

$$P(Y) = \frac{det(L_Y)}{L+I}$$

a brief manipulation([5] Eq.15) shows that when the inverse exist,$L = K(I - K)^{-1}$.

a detailed introduction pdf can be found here`https://jmhldotorg.files.wordpress.com/2014/02/slidesrcc-dpps.pdf`.

a application of DPP is video summarization:[9].

16 Typical Basic Networks

In the bottleneck-like units (like ResNet, ResNeXt or ShuffleNet) bottleneck ratio implies the ratio of bottleneck channels to output channels. For example, bottleneck ratio = 1 : 4 means the output feature map is 4 times the width of the bottleneck feature map.

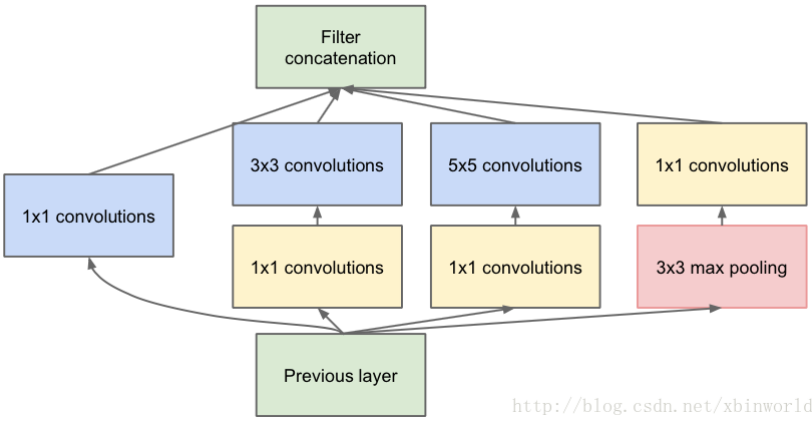
several mainstream:

- VGGNet
- inception series: googlenet,inecptionv2,v3,ResNext
- Resnet
- Xception(Depthwise Convolution)
- ResNext
- DenseNet

16.1 VGGNet

16.2 Inception Series

16.2.1 Inception v1:GoogleNet



use different 1*1, 3*3, 5*5 kernel to realize diverse reception field, the two 1*1 convolution is to decrease the feature map number so that decrease calculation.

16.2.2 Inception v2,v3

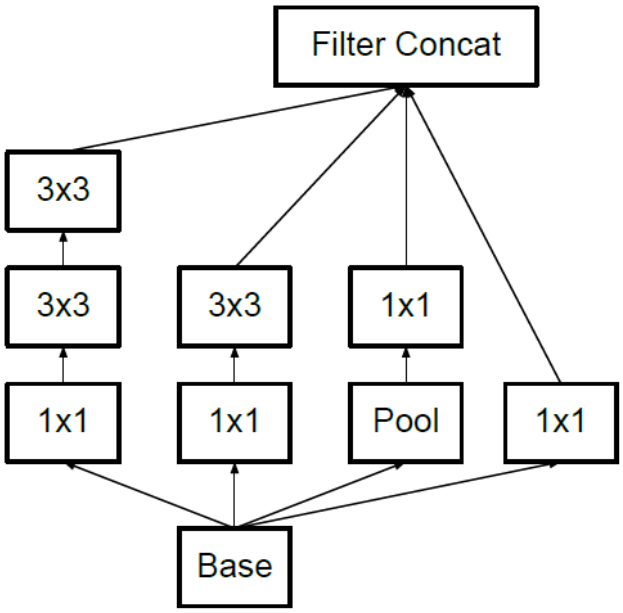


Figure 5. Inception modules where each 5×5 convolution is replaced by two 3×3 convolution, as suggested by principle 3 of Section 2.

turn 5*5 into two 3*3 convolution to save calculation overload.

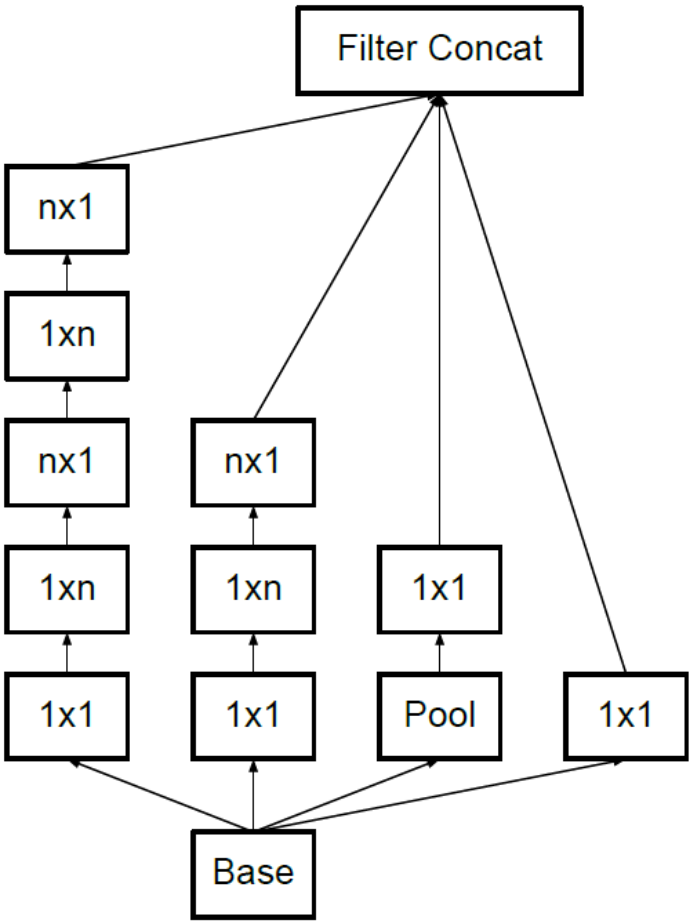


Figure 6. Inception modules after the factorization of the $n \times n$ convolutions. In our proposed architecture, we chose $n = 7$ for the 17×17 grid. (The filter sizes are picked using principle 3)

16.2.3 Inception v4

too complicated, ignore temporarily.

16.3 ResNext

Aggregated Residual Transformations for Deep Neural Networks

`http://www.cnblogs.com/lillylin/p/6799173.html`

- introduce a concept of cardinality.
- same parameter, but better effect!
- use a trick like the resnet: first 1*1 convolution to decrease the feature map number, then apply 3*3 convolution to do normal conv operation, finally use 1*1 convolution again to recover the feature map number.

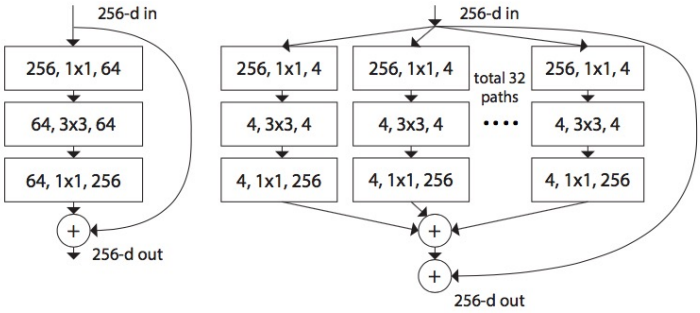


Figure 1. **Left**: A block of ResNet [13]. **Right**: A block of ResNeXt with cardinality = 32, with roughly the same complexity. A layer is shown as (# in channels, filter size, # out channels).

stage	output	ResNet-50	**ResNeXt-50 (32×4d)**
conv1	112×112	7×7, 64, stride 2	7×7, 64, stride 2
conv2	56×56	3×3 max pool, stride 2	3×3 max pool, stride 2
		$\begin{bmatrix} 1\times1, 64 \\ 3\times3, 64 \\ 1\times1, 256 \end{bmatrix}\times3$	$\begin{bmatrix} 1\times1, 128 \\ 3\times3, 128, C{=}32 \\ 1\times1, 256 \end{bmatrix}\times3$
conv3	28×28	$\begin{bmatrix} 1\times1, 128 \\ 3\times3, 128 \\ 1\times1, 512 \end{bmatrix}\times4$	$\begin{bmatrix} 1\times1, 256 \\ 3\times3, 256, C{=}32 \\ 1\times1, 512 \end{bmatrix}\times4$
conv4	14×14	$\begin{bmatrix} 1\times1, 256 \\ 3\times3, 256 \\ 1\times1, 1024 \end{bmatrix}\times6$	$\begin{bmatrix} 1\times1, 512 \\ 3\times3, 512, C{=}32 \\ 1\times1, 1024 \end{bmatrix}\times6$
conv5	7×7	$\begin{bmatrix} 1\times1, 512 \\ 3\times3, 512 \\ 1\times1, 2048 \end{bmatrix}\times3$	$\begin{bmatrix} 1\times1, 1024 \\ 3\times3, 1024, C{=}32 \\ 1\times1, 2048 \end{bmatrix}\times3$
	1×1	global average pool 1000-d fc, softmax	global average pool 1000-d fc, softmax
# params.		**25.5**×10^6	**25.0**×10^6
FLOPs		**4.1**×10^9	**4.2**×10^9

Table 1. (**Left**) ResNet-50. (**Right**) ResNeXt-50 with a 32×4d template (using the reformulation in Fig. 3(c)). Inside the brackets are the shape of a residual block, and outside the brackets is the number of stacked blocks on a stage. "C=32" suggests grouped convolutions [23] with 32 groups. *The numbers of parameters and FLOPs are similar between these two models.*

notice the conventional setting: resnet bottleneck ratio = 1:2, resnext bottleneck ratio=1:4

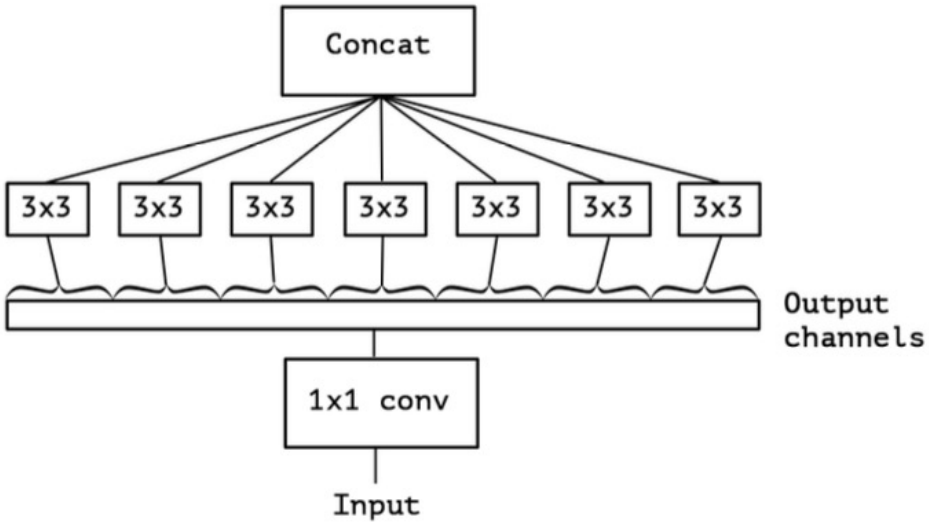
16.4 Xception

Xception: Deep Learning with Depthwise Separable Convolutions.

`http://blog.csdn.net/kangroger/article/details/69929915`

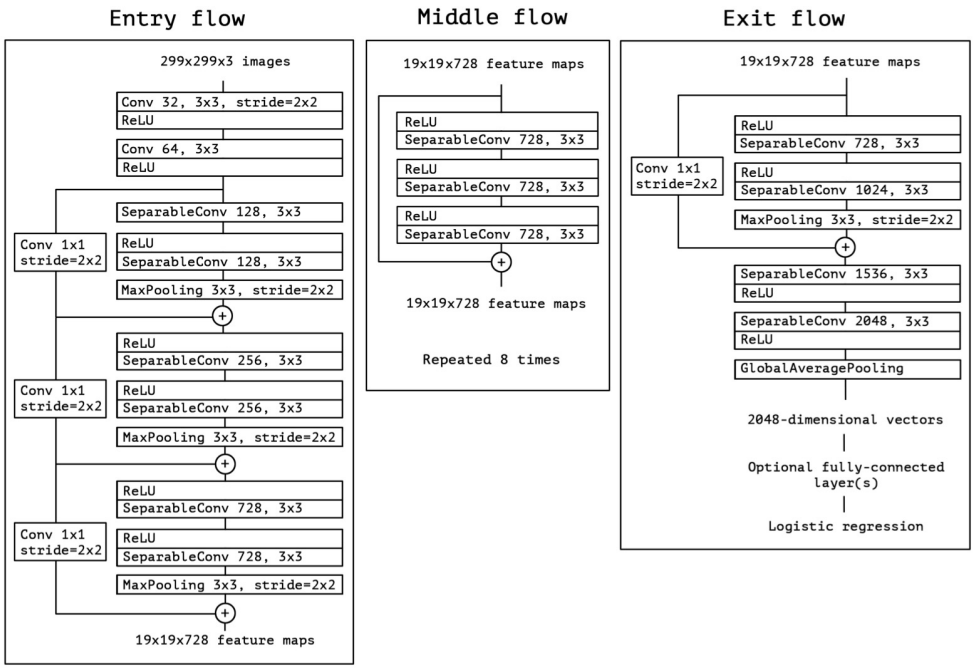
a caffe network overview is here:
http://ethereon.github.io/netscope/#/gist/b898efc9749bfdb87d09432e2239e526

Figure 4. An "extreme" version of our Inception module, with one spatial convolution per output channel of the 1x1 convolution.



Two minor differences between and "extreme" version of an Inception module and a depthwise separable convolution would be:

- The order of the operations: depthwise separable convolutions as usually implemented (e.g. in TensorFlow) perform first channel-wise spatial convolution and then perform 1x1 convolution, whereas Inception performs the 1x1 convolution first.
- The presence or absence of a non-linearity after the first operation. In Inception, both operations are followed by a ReLU non-linearity, however depthwise



in keras, it is called Depthwise separable 2D convolution.

Separable convolutions consist in first performing a depthwise spatial convolution (which acts on each input channel separately) followed by a pointwise convolution which mixes together the resulting output channels.

Intuitively, separable convolutions can be understood as a way to factorize a convolution kernel into two smaller kernels, or as an extreme version of an Inception block.

16.5 ResNet

notices:

- W_2 in residual block have no ReLu
- in resnet18/34, the residual block only have W_1, W_2, in resnet50/101/152, the residual block have W_1, W_2, W_3
- two ways to realize downsampling residual unit.
- 7*7,64,stride=2 and 7*7 max pool,stride=2 to downsample 4X at the begining of the network.
- resnet4 is the heaviest part of the whole network.
- finally GAP and FC is used to generation 1000 softmax.

$$y = \mathcal{F}(x, \{W_i\}) + W_s x$$
$$\mathcal{F} = W_2 \sigma(W_1 x)$$

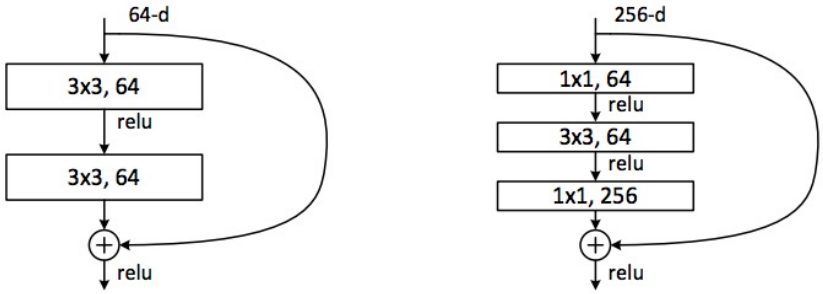


Figure 5. A deeper residual function $\mathcal{F}$ for ImageNet. Left: a building block (on 56×56 feature maps) as in Fig. 3 for ResNet-34. Right: a "bottleneck" building block for ResNet-50/101/152.

the left one is a typical resnet block, however, a more reasonable block is like the right one: first 1*1 convolution is to decrease feature map number, second 1*1 convolution is to increase(recover) feature map number.

layer name	output size	18-layer	34-layer	50-layer	101-layer	152-layer
conv1	112×112	7×7, 64, stride 2				
conv2_x	56×56	3×3 max pool, stride 2				
		$\begin{bmatrix} 3\times3, 64 \\ 3\times3, 64 \end{bmatrix}\times2$	$\begin{bmatrix} 3\times3, 64 \\ 3\times3, 64 \end{bmatrix}\times3$	$\begin{bmatrix} 1\times1, 64 \\ 3\times3, 64 \\ 1\times1, 256 \end{bmatrix}\times3$	$\begin{bmatrix} 1\times1, 64 \\ 3\times3, 64 \\ 1\times1, 256 \end{bmatrix}\times3$	$\begin{bmatrix} 1\times1, 64 \\ 3\times3, 64 \\ 1\times1, 256 \end{bmatrix}\times3$
conv3_x	28×28	$\begin{bmatrix} 3\times3, 128 \\ 3\times3, 128 \end{bmatrix}\times2$	$\begin{bmatrix} 3\times3, 128 \\ 3\times3, 128 \end{bmatrix}\times4$	$\begin{bmatrix} 1\times1, 128 \\ 3\times3, 128 \\ 1\times1, 512 \end{bmatrix}\times4$	$\begin{bmatrix} 1\times1, 128 \\ 3\times3, 128 \\ 1\times1, 512 \end{bmatrix}\times4$	$\begin{bmatrix} 1\times1, 128 \\ 3\times3, 128 \\ 1\times1, 512 \end{bmatrix}\times8$
conv4_x	14×14	$\begin{bmatrix} 3\times3, 256 \\ 3\times3, 256 \end{bmatrix}\times2$	$\begin{bmatrix} 3\times3, 256 \\ 3\times3, 256 \end{bmatrix}\times6$	$\begin{bmatrix} 1\times1, 256 \\ 3\times3, 256 \\ 1\times1, 1024 \end{bmatrix}\times6$	$\begin{bmatrix} 1\times1, 256 \\ 3\times3, 256 \\ 1\times1, 1024 \end{bmatrix}\times23$	$\begin{bmatrix} 1\times1, 256 \\ 3\times3, 256 \\ 1\times1, 1024 \end{bmatrix}\times36$
conv5_x	7×7	$\begin{bmatrix} 3\times3, 512 \\ 3\times3, 512 \end{bmatrix}\times2$	$\begin{bmatrix} 3\times3, 512 \\ 3\times3, 512 \end{bmatrix}\times3$	$\begin{bmatrix} 1\times1, 512 \\ 3\times3, 512 \\ 1\times1, 2048 \end{bmatrix}\times3$	$\begin{bmatrix} 1\times1, 512 \\ 3\times3, 512 \\ 1\times1, 2048 \end{bmatrix}\times3$	$\begin{bmatrix} 1\times1, 512 \\ 3\times3, 512 \\ 1\times1, 2048 \end{bmatrix}\times3$
	1×1	average pool, 1000-d fc, softmax				
FLOPs		1.8×10^9	3.6×10^9	3.8×10^9	7.6×10^9	11.3×10^9

Table 1. Architectures for ImageNet. Building blocks are shown in brackets (see also Fig. 5), with the numbers of blocks stacked. Downsampling is performed by conv3_1, conv4_1, and conv5_1 with a stride of 2.

Q: how to downsampling in resnet?

- The shortcut still performs identity mapping, with extra zero entries padded for increasing dimensions. This option introduces no extra parameter

- The projection shortcut(1X1 convolution with stride=2) is used to match dimensions(feature map size)

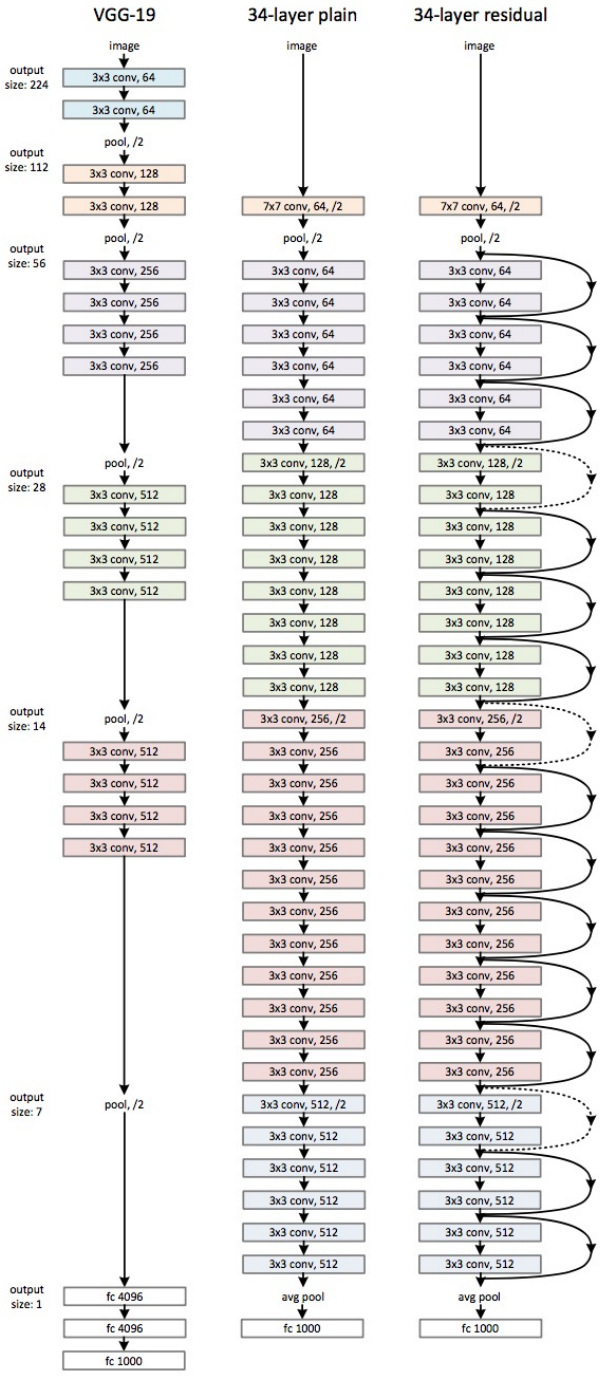


Figure 3. Example network architectures for ImageNet. **Left**: the VGG-19 model [40] (19.6 billion FLOPs) as a reference. **Middle**: a plain network with 34 parameter layers (3.6 billion FLOPs). **Right**: a residual network with 34 parameter layers (3.6 billion FLOPs). The dotted shortcuts increase dimensions. **Table 1** shows more details and other variants.

16.6 DenseNet

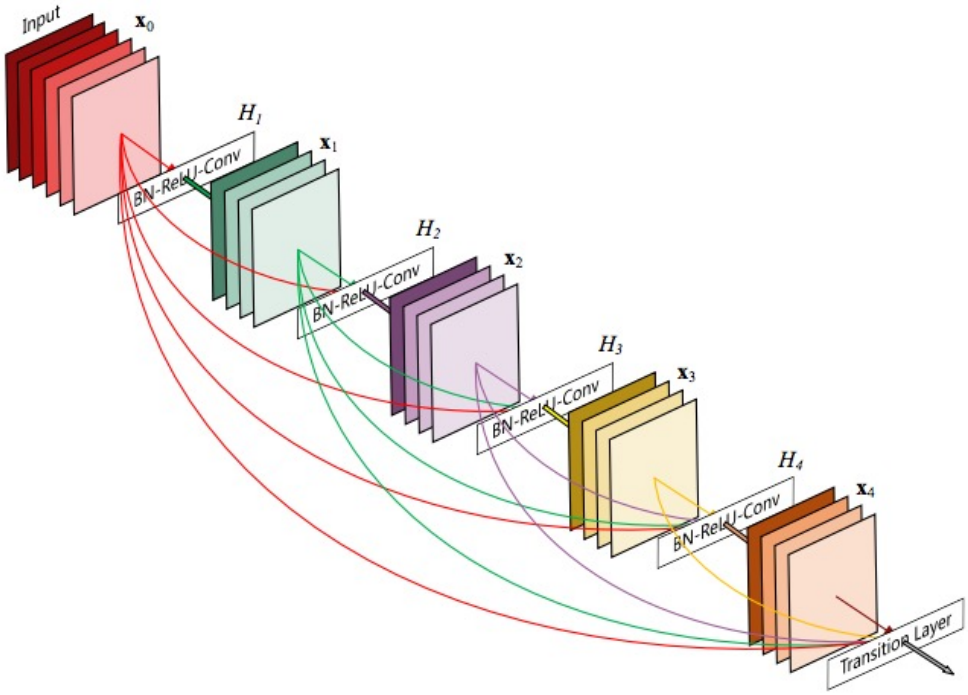


Figure 1. A 5-layer dense block with a growth rate of $k = 4$. Each layer takes all preceding feature-maps as input.

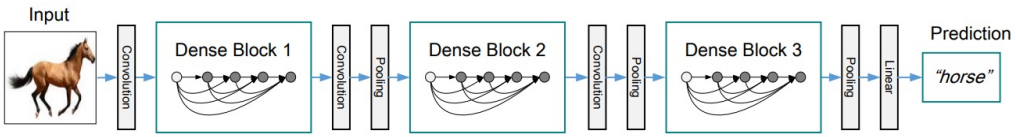


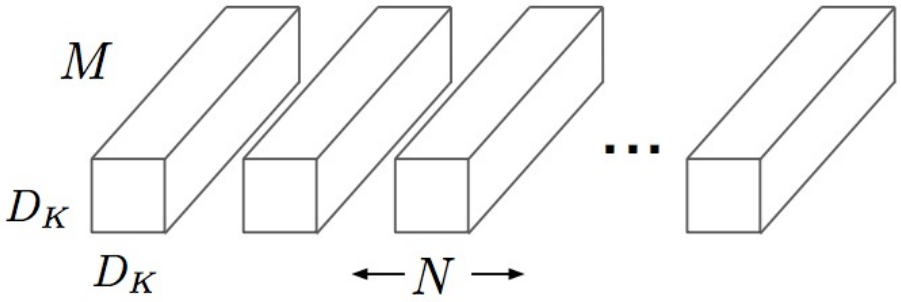
Figure 2. A deep DenseNet with three dense blocks. The layers between two adjacent blocks are referred to as transition layers and change feature map sizes via convolution and pooling.

Layers	Output Size	DenseNet-121($k = 32$)	DenseNet-169($k = 32$)	DenseNet-201($k = 32$)	DenseNet-161($k = 48$)
Convolution	112 × 112	7 × 7 conv, stride 2			
Pooling	56 × 56	3 × 3 max pool, stride 2			
Dense Block (1)	56 × 56	$\begin{bmatrix} 1 \times 1 \text{ conv} \\ 3 \times 3 \text{ conv} \end{bmatrix} \times 6$	$\begin{bmatrix} 1 \times 1 \text{ conv} \\ 3 \times 3 \text{ conv} \end{bmatrix} \times 6$	$\begin{bmatrix} 1 \times 1 \text{ conv} \\ 3 \times 3 \text{ conv} \end{bmatrix} \times 6$	$\begin{bmatrix} 1 \times 1 \text{ conv} \\ 3 \times 3 \text{ conv} \end{bmatrix} \times 6$
Transition Layer (1)	56 × 56	1 × 1 conv			
	28 × 28	2 × 2 average pool, stride 2			
Dense Block (2)	28 × 28	$\begin{bmatrix} 1 \times 1 \text{ conv} \\ 3 \times 3 \text{ conv} \end{bmatrix} \times 12$	$\begin{bmatrix} 1 \times 1 \text{ conv} \\ 3 \times 3 \text{ conv} \end{bmatrix} \times 12$	$\begin{bmatrix} 1 \times 1 \text{ conv} \\ 3 \times 3 \text{ conv} \end{bmatrix} \times 12$	$\begin{bmatrix} 1 \times 1 \text{ conv} \\ 3 \times 3 \text{ conv} \end{bmatrix} \times 12$
Transition Layer (2)	28 × 28	1 × 1 conv			
	14 × 14	2 × 2 average pool, stride 2			
Dense Block (3)	14 × 14	$\begin{bmatrix} 1 \times 1 \text{ conv} \\ 3 \times 3 \text{ conv} \end{bmatrix} \times 24$	$\begin{bmatrix} 1 \times 1 \text{ conv} \\ 3 \times 3 \text{ conv} \end{bmatrix} \times 32$	$\begin{bmatrix} 1 \times 1 \text{ conv} \\ 3 \times 3 \text{ conv} \end{bmatrix} \times 48$	$\begin{bmatrix} 1 \times 1 \text{ conv} \\ 3 \times 3 \text{ conv} \end{bmatrix} \times 36$
Transition Layer (3)	14 × 14	1 × 1 conv			
	7 × 7	2 × 2 average pool, stride 2			
Dense Block (4)	7 × 7	$\begin{bmatrix} 1 \times 1 \text{ conv} \\ 3 \times 3 \text{ conv} \end{bmatrix} \times 16$	$\begin{bmatrix} 1 \times 1 \text{ conv} \\ 3 \times 3 \text{ conv} \end{bmatrix} \times 32$	$\begin{bmatrix} 1 \times 1 \text{ conv} \\ 3 \times 3 \text{ conv} \end{bmatrix} \times 32$	$\begin{bmatrix} 1 \times 1 \text{ conv} \\ 3 \times 3 \text{ conv} \end{bmatrix} \times 24$
Classification Layer	1 × 1	7 × 7 global average pool			
		1000D fully-connected, softmax			

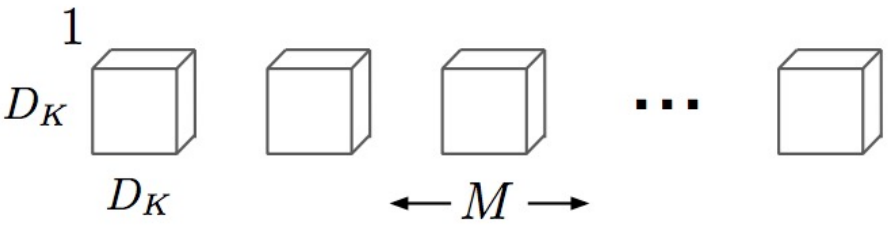
Table 1. DenseNet architectures for ImageNet. The growth rate for the first 3 networks is $k = 32$, and $k = 48$ for DenseNet-161. Note that each "conv" layer shown in the table corresponds the sequence BN-ReLU-Conv.

17 Network Compression

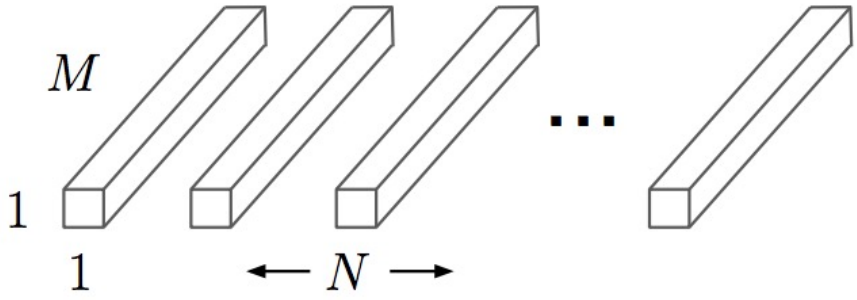
17.1 MobileNet



(a) Standard Convolution Filters



(b) Depthwise Convolutional Filters



(c) 1×1 Convolutional Filters called Pointwise Convolution in the context of Depthwise Separable Convolution

Figure 2. The standard convolutional filters in (a) are replaced by two layers: depthwise convolution in (b) and pointwise convolution in (c) to build a depthwise separable filter.

summary:

- factorize a standard convolution into a depthwise convolution and a 1X1 convolution called a pointwise convolution.

- the model structure puts nearly all of the computation into dense 1X1 convolutions. This can be implemented with highly optimized general matrix multiply (GEMM) functions. Often convolutions are implemented by a GEMM but require an initial reordering in memory called im2col in order to map it to a GEMM. For instance, this approach is used in the popular Caffe package . 1X1 convolutions do not require this reordering in memory and can be implemented directly with GEMM which is one of the most optimized numerical linear algebra algorithms.

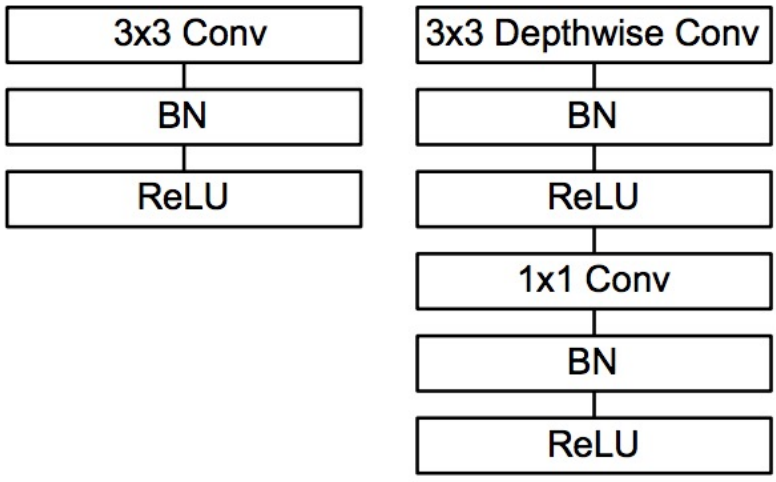


Figure 3. Left: Standard convolutional layer with batchnorm and ReLU. Right: Depthwise Separable convolutions with Depthwise and Pointwise layers followed by batchnorm and ReLU.

Table 1. MobileNet Body Architecture

Type / Stride	Filter Shape	Input Size
Conv / s2	$3 \times 3 \times 3 \times 32$	$224 \times 224 \times 3$
Conv dw / s1	$3 \times 3 \times 32$ dw	$112 \times 112 \times 32$
Conv / s1	$1 \times 1 \times 32 \times 64$	$112 \times 112 \times 32$
Conv dw / s2	$3 \times 3 \times 64$ dw	$112 \times 112 \times 64$
Conv / s1	$1 \times 1 \times 64 \times 128$	$56 \times 56 \times 64$
Conv dw / s1	$3 \times 3 \times 128$ dw	$56 \times 56 \times 128$
Conv / s1	$1 \times 1 \times 128 \times 128$	$56 \times 56 \times 128$
Conv dw / s2	$3 \times 3 \times 128$ dw	$56 \times 56 \times 128$
Conv / s1	$1 \times 1 \times 128 \times 256$	$28 \times 28 \times 128$
Conv dw / s1	$3 \times 3 \times 256$ dw	$28 \times 28 \times 256$
Conv / s1	$1 \times 1 \times 256 \times 256$	$28 \times 28 \times 256$
Conv dw / s2	$3 \times 3 \times 256$ dw	$28 \times 28 \times 256$
Conv / s1	$1 \times 1 \times 256 \times 512$	$14 \times 14 \times 256$
$5\times$ Conv dw / s1	$3 \times 3 \times 512$ dw	$14 \times 14 \times 512$
$5\times$ Conv / s1	$1 \times 1 \times 512 \times 512$	$14 \times 14 \times 512$
Conv dw / s2	$3 \times 3 \times 512$ dw	$14 \times 14 \times 512$
Conv / s1	$1 \times 1 \times 512 \times 1024$	$7 \times 7 \times 512$
Conv dw / s2	$3 \times 3 \times 1024$ dw	$7 \times 7 \times 1024$
Conv / s1	$1 \times 1 \times 1024 \times 1024$	$7 \times 7 \times 1024$
Avg Pool / s1	Pool 7×7	$7 \times 7 \times 1024$
FC / s1	1024×1000	$1 \times 1 \times 1024$
Softmax / s1	Classifier	$1 \times 1 \times 1000$

Table 2. Resource Per Layer Type

Type	Mult-Adds	Parameters
Conv 1×1	94.86%	74.59%
Conv DW 3×3	3.06%	1.06%
Conv 3×3	1.19%	0.02%
Fully Connected	0.18%	24.33%

the mobilenet structure is some like VGG, which don't have shortcut layer like resnet.

17.2 ShuffleNet

The core idea of ShuffleNet lies in pointwise group convolution and channel shuffle operation reference

For example, given the input size c*h*w and the bottleneck channels m, ResNet unit requires hw(2cm + 9m2) FLOPs and ResNeXt has hw(2cm + 9m2/g) FLOPs, while ShuffleNet unit requires only hw(2cm/g + 9m) FLOPs, where g means the number of groups for convolutions.

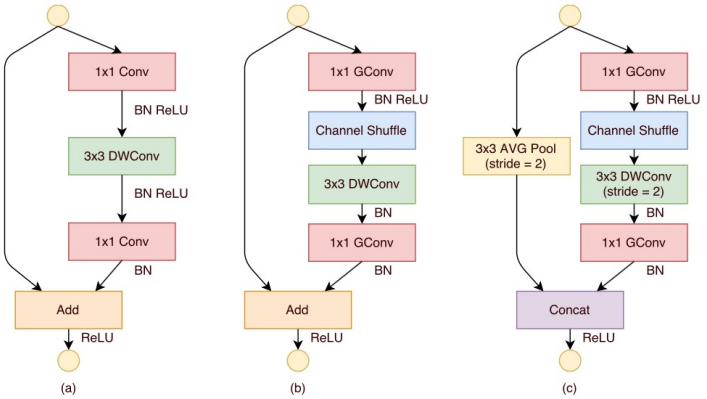


Figure 2: ShuffleNet Units. a) bottleneck unit [9] with depthwise convolution (DWConv) [3, 12]; b) ShuffleNet unit with pointwise group convolution (GConv) and channel shuffle; c) ShuffleNet unit with stride = 2.

18 Detection

18.1 SPPNet

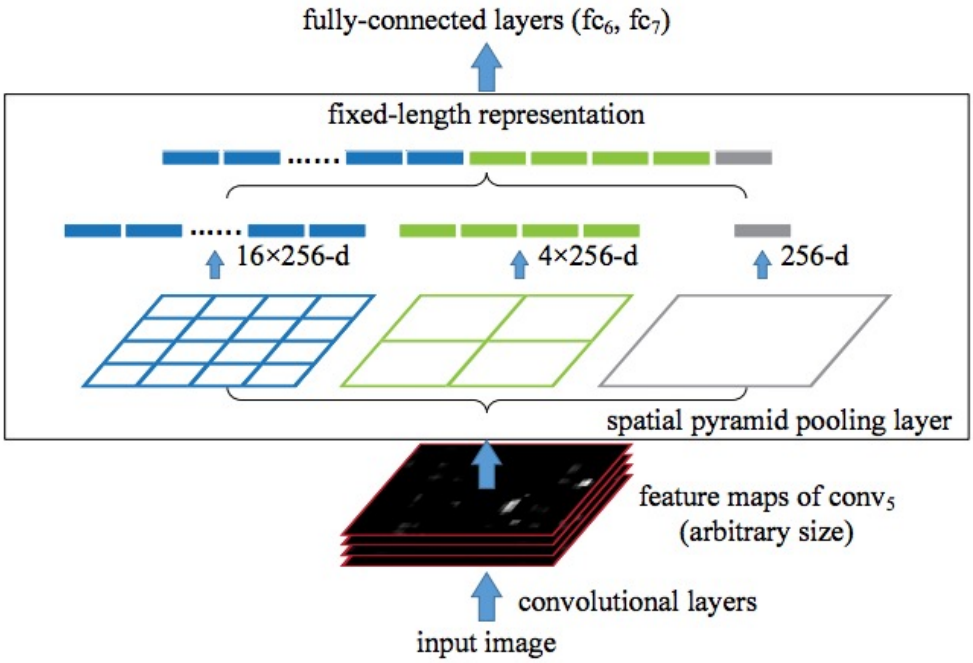


Figure 3: A network structure with a **spatial pyramid pooling layer**. Here 256 is the filter number of the $conv_5$ layer, and $conv_5$ is the last convolutional layer.

Our method extracts window-wise features from regions of the feature maps, while R-CNN extracts directly from image regions. In previous works, the Deformable Part Model (DPM) [23] extracts features from windows in HOG [24] feature maps, and the Selective Search (SS) method [20] extracts from windows in encoded SIFT feature maps. The Overfeat detection method [5] also extracts from windows of deep convolutional feature maps, but needs to predefine the window size. On the contrary, our method enables feature extraction in arbitrary windows from the deep convolutional feature maps.

The resulting SPP-net shows outstanding accuracy in classification/detection tasks and greatly accelerates DNN-based detection. Our studies also show that many time-proven techniques/insights in computer vision can still play important roles in deep-networks-based recognition.

18.2 Faster RCNN

Faster R-CNN: Towards Real-Time Object Detection with Region Proposal Networks

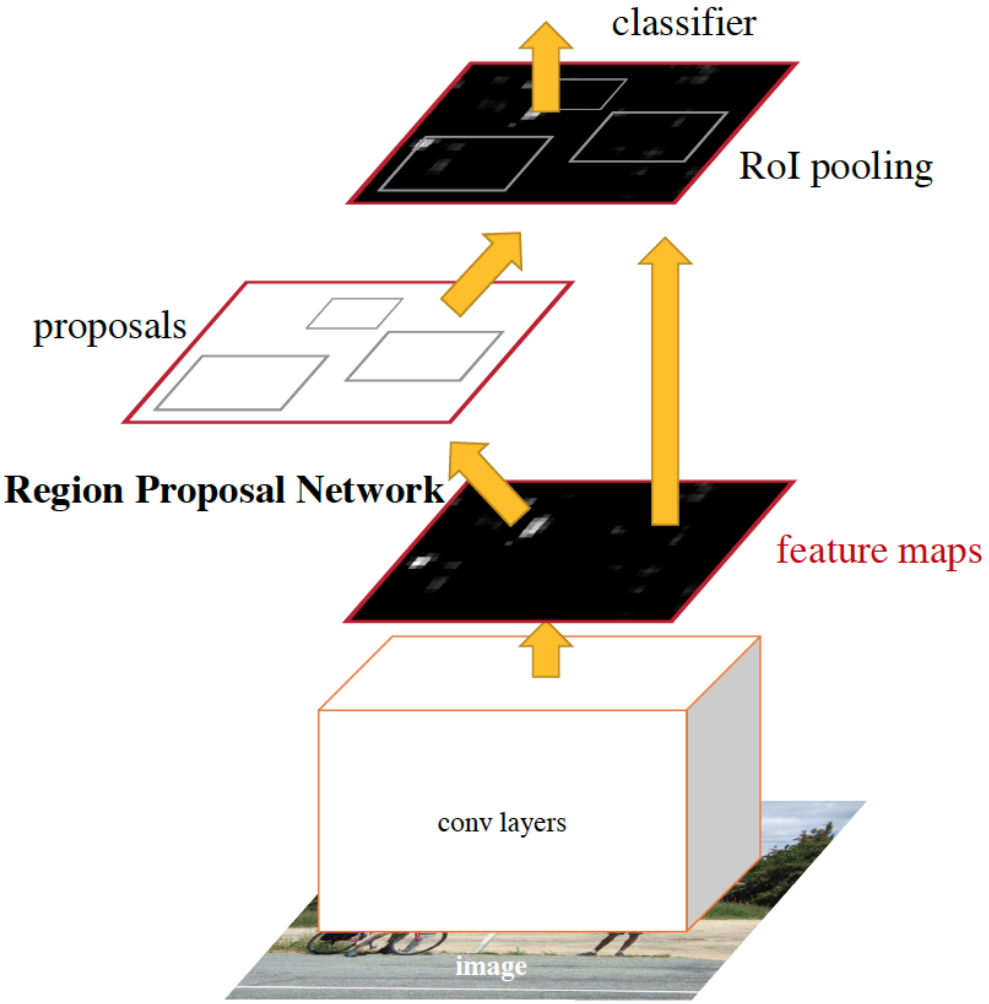


Figure 2: Faster R-CNN is a single, unified network for object detection. The RPN module serves as the 'attention' of this unified network.

18.3 SSD

18.4 YOLO

18.5 Mask-RCNN

18.6 NMS

take the detection task as an example.

- (1).order all boundary descend by score, choose the highest score and its corresponding boundary.
- (2). iterate all other boundary, if the IoU between current boundary and highest-score boundary is greater than a threshold, then delete the boundary.
- (3). choose a highest score from the remaining boundaries, and repeat from (1).

19 Segmentation

19.1 Deformable Convolution Network

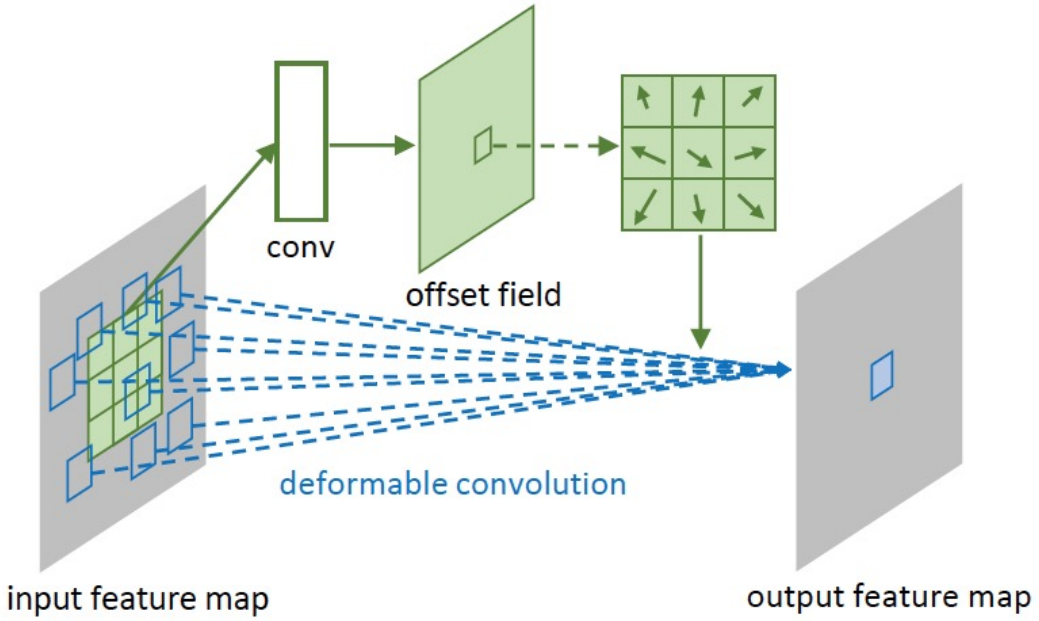


Figure 2: Illustration of 3×3 deformable convolution.

when do convolution, first get 3*3 conv kernel value, then get 3*3*2 kernel offset(may be fractional! need bilinear sampling).

$R = \{(-1,-1),(-1,0)...\}$
normal convolution operation:
$y(p_0) = \sum_{p_n \in R} w(p_n)x(p_0 + p_n)$
deformable convolution operation:
$y(p_0) = \sum_{p_n \in R} w(p_n)x(p_0 + p_n + \Delta p_n)$
we need to notice here x(p), the p may be fractional. so we need bilinear sampling.
$x(p) = \sum_q G(q,p)x(q)$
here $G(q,p) = g(q_x,p_x)g(q_y,p_y)$
$g(a,b) = max(0, 1-|a-b|)$

20 Large Kernel Matters

Semantic segmentation can be considered as a per-pixel classification problem. There are two challenges in this task: 1) classification: an object associated to a specific semantic concept should be marked correctly; 2) localization: the classification label for a pixel must be aligned to the appropriate coordinates in output score map. A well-designed segmentation model should deal with the two issues simultaneously. However, these two tasks are naturally contradictory. For the classification task, the models are required to be invariant to various transformations like translation and rotation. But for the localization task, models should be transformation-sensitive, i.e., precisely locate every pixel for each semantic category. The conventional semantic segmentation algorithms mainly target for the localization issue, as shown in Figure 1 B.

1*k and k*1 equals k*k, but can save more computation!

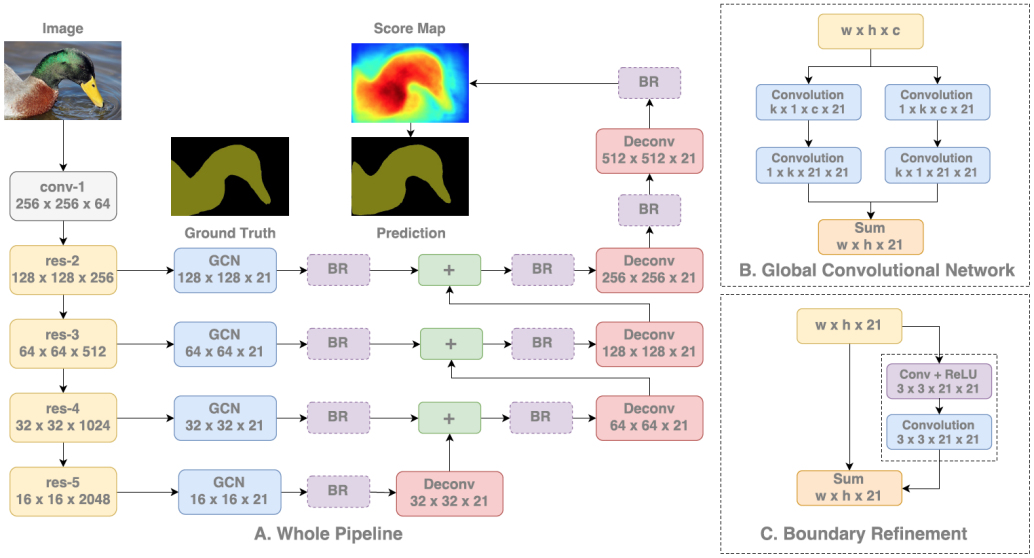


Figure 2. An overview of the whole pipeline in (A). The details of Global Convolutional Network (GCN) and Boundary Refinement (BR) block are illustrated in (B) and (C), respectively.

21 Skeleton

21.1 HourGlass

notice:

- first branch off, then downsampling.
- downsampling via poolind2D
- upsampling via nearest neighbor upsampling.
- the middle of Fig 3. is wrong, it only have one residual module, not three residual module.
- the topography is symmetric.
- each of the block in Fig.3 is a residual module in Fig.4 Left.
- the beginning of hourglass, we downsampling 4X to save memory. during each stage of hourglass, we use one 1*1 conv to generate ground truth predict. the other two 1*1 conv is to recover the standard feature map numbers, so that they can do elementwise add.

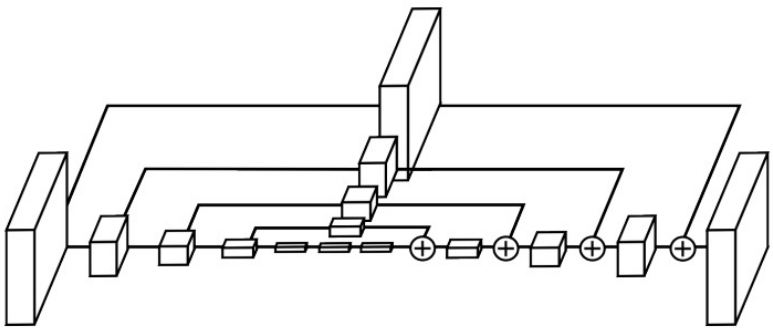


Fig. 3. An illustration of a single "hourglass" module. Each box in the figure corresponds to a residual module as seen in Figure 4. The number of features is consistent across the whole hourglass.

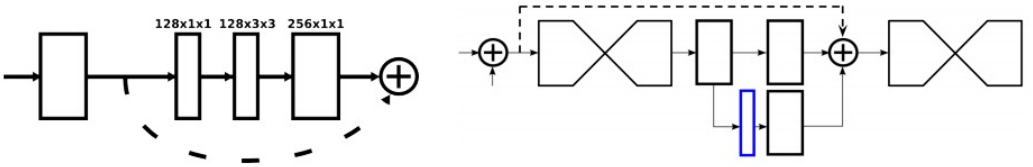


Fig. 4. Left: Residual Module [14] that we use throughout our network. **Right:** Illustration of the intermediate supervision process. The network splits and produces a set of heatmaps (outlined in blue) where a loss can be applied. A 1x1 convolution remaps the heatmaps to match the number of channels of the intermediate features. These are added together along with the features from the preceding hourglass.

Acknowledgments

References

[1] Kumar Chellapilla, Sidd Puri, and Patrice Simard. High performance convolutional neural networks for document processing. In *Tenth International Workshop on Frontiers in Handwriting Recognition*. Suvisoft, 2006.

[2] Carl Doersch. Tutorial on variational autoencoders. *arXiv preprint arXiv:1606.05908*, 2016.

[3] Alex Graves, Santiago Fernandez, and Juergen Schmidhuber. Multi-dimensional recurrent neural networks, 2007.

[4] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Identity mappings in deep residual networks. In *European Conference on Computer Vision*, pages 630–645. Springer, 2016.

[5] Alex Kulesza and Ben Taskar. Determinantal point processes for machine learning. 2012.

[6] Zhouchen Lin, Risheng Liu, and Zhixun Su. Linearized alternating direction method with adaptive penalty for low-rank representation. In J. Shawe-Taylor, R. S. Zemel, P. L. Bartlett, F. Pereira, and K. Q. Weinberger, editors, *Advances in Neural Information Processing Systems 24*, pages 612–620. Curran Associates, Inc., 2011.

[7] Zelda Mariet and Suvrit Sra. Kronecker determinantal point processes, 2016.

[8] Yandong Wen, Kaipeng Zhang, Zhifeng Li, and Yu Qiao. A discriminative feature learning approach for deep face recognition. In *European Conference on Computer Vision*, pages 499–515. Springer, 2016.

[9] Ke Zhang, Wei-Lun Chao, Fei Sha, and Kristen Grauman. Video summarization with long short-term memory, 2016.