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 Optimization

1.1 Super-Convergence: Very Fast Training of Residual Networks Using Large Learning Rates

`https://www.zhihu.com/question/64697394/answer/223304511`

- Use Hessian-free optimization method to estimate optimal learning rate(upper bound and lower bound).
- Cyclic learning rate can induce to super-convergence.
- large learning rate can realize regularization.

2 Expectation Maximization Introduction

2.1 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: $\theta := argmax_\theta \sum_i \sum_{Z^i} Q_i(Z^{(i)}) log \frac{p(X^{(i)},Z^{(i)}),\theta)}{Q_i(Z^{(i)})}$

2.2 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.

2.3 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.

3 EM applications

3.1 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($\sum_i^K \alpha_k = 1$).

3.2 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: $\sum_{i=1}^{N} \pi_i = 1, \sum_{j=1}^{N} a_{ij} = 1, \sum_{i=1}^{M} b_i = 1$)

3.3 Naive Bayesian

3.4 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.

4 VAE

here is a complete VAE tutorial [9]

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

Another point of view:

$$D[Q(z)||P(z|X)] = E_{z\sim Q}[logQ(z) - logP(z|X)]$$
apply Bayes rule to P(z|X):
$$D[Q(z)||P(z|X)] = E_{z\sim Q}[logQ(z) - logP(X|z) - logP(z)] + logP(X)$$
rearrange:
$$logP(X) - D[Q(z)||P(z|X)] = E_{z\sim Q}[logP(X|z)] - D[Q(z)||P(z)]$$
Note that X is fixed, and Q can be any distribution, not just a distribution which does a good job mapping X to the z's that can produce X. change Q(z) to Q(z|X)
$$logP(X) - D[Q(z|X)||P(z|X)] = E_{z\sim Q}[logP(X|z)] - D[Q(z|X)||P(z)]$$

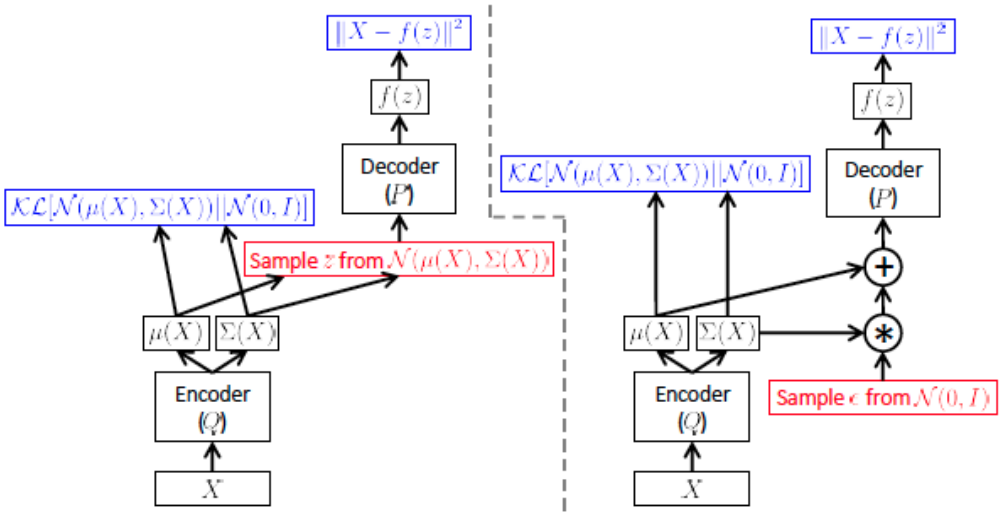


Figure 4: A training-time variational autoencoder implemented as a feed-forward neural network, where $P(X|z)$ is Gaussian. Left is without the "reparameterization trick", and right is with it. Red shows sampling operations that are non-differentiable. Blue shows loss layers. The feedforward behavior of these networks is identical, but backpropagation can be applied only to the right network.

5 ADMM

6 Key steps you must know when building a DL Framework

6.1 Convolution

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

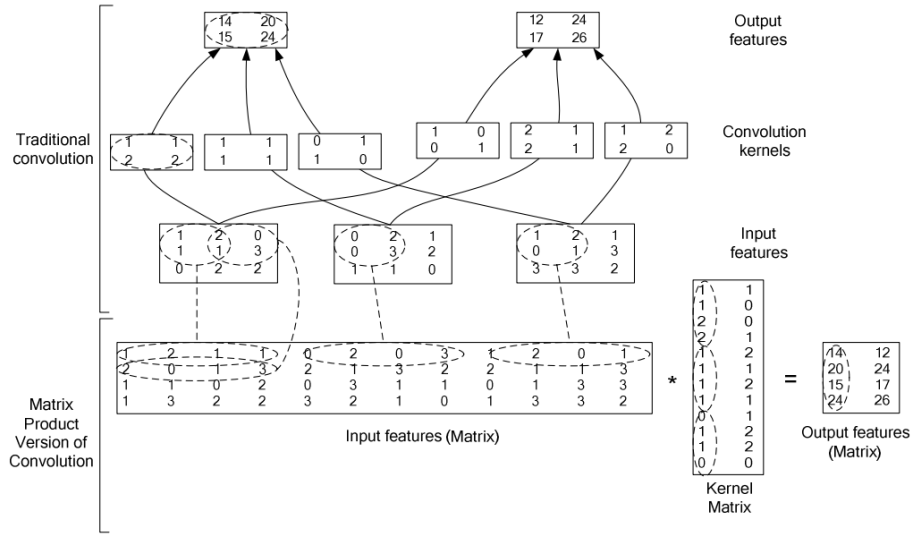


Figure 2. Example convolution operations in a convolutional layer (biases, sub-sampling, and non-linearity omitted). The top figure presents the traditional convolution operations, while the bottom figure presents the matrix version.

figure is from [3].
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/`

6.2 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.

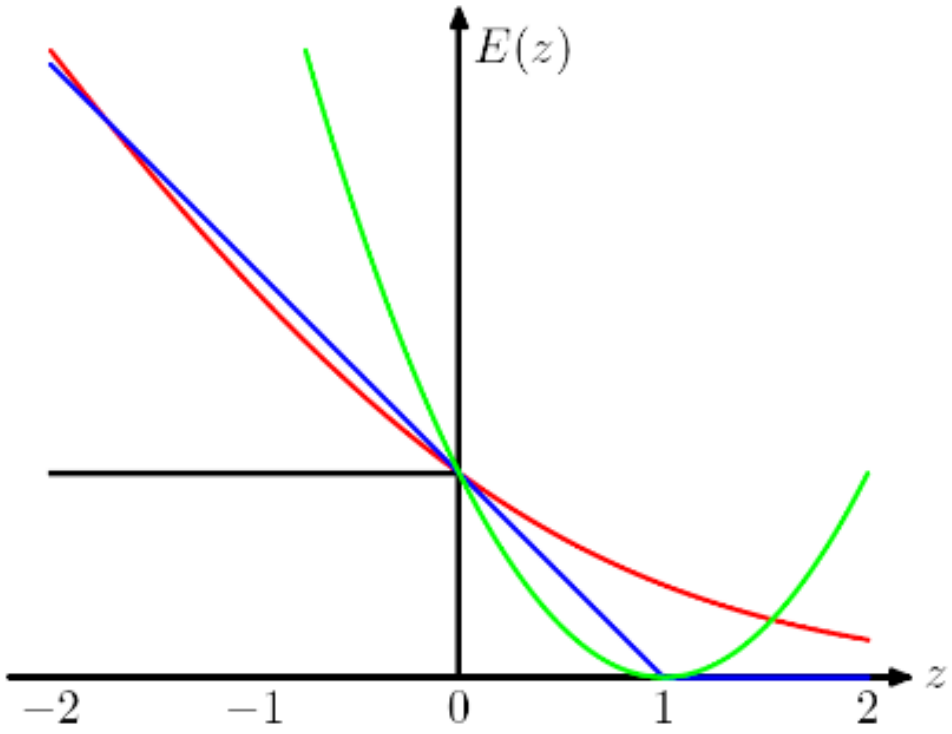
Following plot is coming from Chris Bishop's PRML book. The Hinge Loss is plotted in blue, the Log Loss in red, the Square Loss in green and the 0/1 error in black.

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!**



6.2.1 Loss function example

Center Loss[27]

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:

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

6.3 WitchCraft

6.3.1 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

6.3.2 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

6.3.3 ResNet Pre Activation[14]

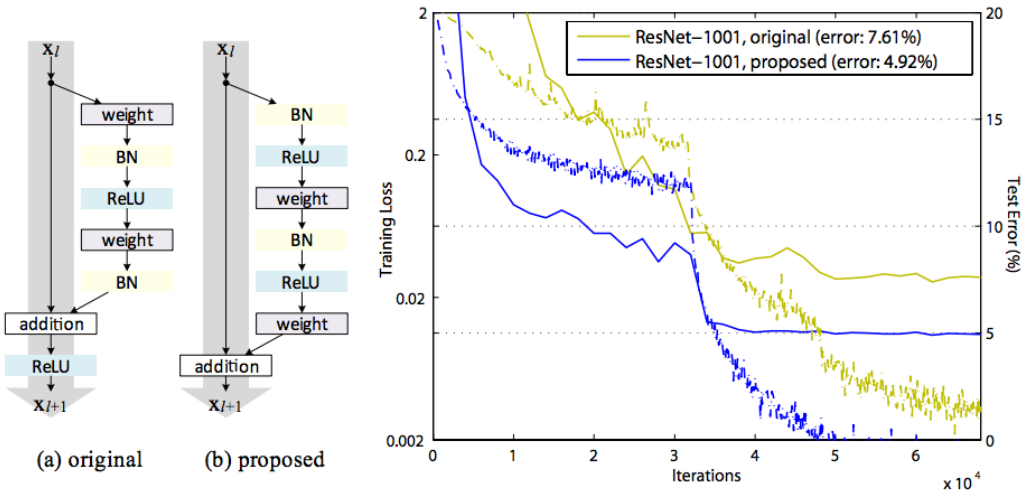


Figure 1. Left: (a) original Residual Unit in [1]; (b) proposed Residual Unit. The grey arrows indicate the easiest paths for the information to propagate, corresponding to the additive term "$\mathbf{x}_l$" in Eqn.(4) (forward propagation) and the additive term "1" in Eqn.(5) (backward propagation). **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!!

6.4 Batch Normalization[15]

$$\mu_{\mathcal{B}} \leftarrow \frac{1}{m}\sum_{i=1}^{m} x_i$$

$$\sigma_{\mathcal{B}} \leftarrow \sqrt{\frac{1}{m}\sum_{i=1}^{m}(x_i - \mu_{\mathcal{B}})^2 + \epsilon}$$

$$\widehat{x}_i \leftarrow \frac{x_i - \mu_{\mathcal{B}}}{\sigma_{\mathcal{B}}}$$

$$y_i \leftarrow \gamma\widehat{x}_i + \beta \equiv \mathrm{BN}(x_i)$$

6.5 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

7 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 :

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

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

7.1 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 $A = US_{\beta^{-1}}(\Sigma)V^T$.

7.2 Adaptive Penalty for ADMM

Lin et al.[23] 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.

8 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

9 Mainfold Learning

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

9.1 Laplace Matrix

9.2 Normalized Cut

9.3 Linear Dimension Reduction

PCA,CDMS,RP

9.3.1 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:

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

9.4 NonLinear Dimension Reduction

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

10 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 $\mathbf{U}_{k,l}$:

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

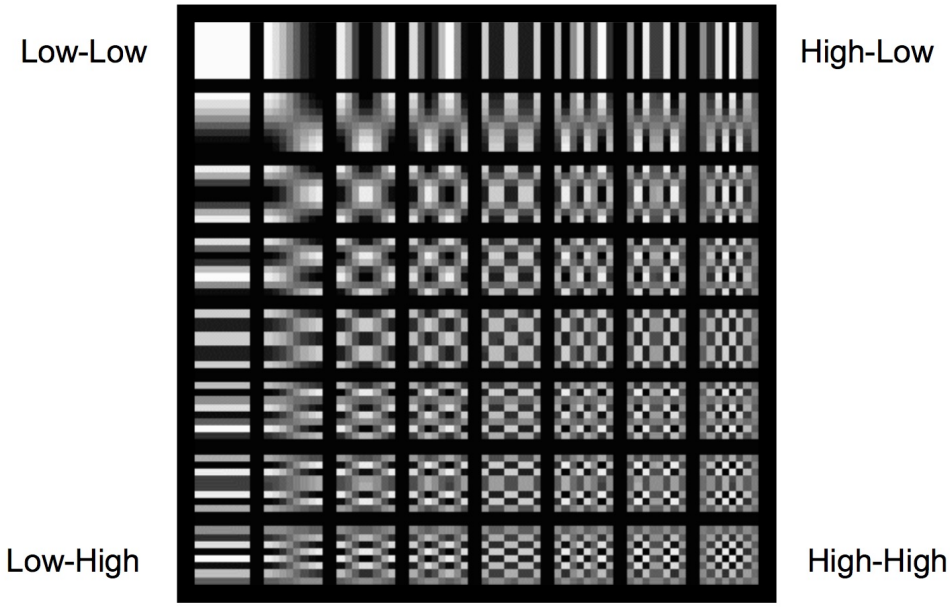
- Basis image = outer product of 1D DCT basis vector

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

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

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

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

11 KL-divergence Application

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

12 GAN

optimization problem:

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

notation:
p_g:generator's distribution
p_d: training data's distribution
$p_g(x)$: the probability that x comes from generator
$p_d(x)$: 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

extra reference: the summary of different similarity metrics between distribution. `https://www.zhihu.com/question/39872326`

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

the difference between forward KL and reverse KL: `https://wiseodd.github.io/techblog/2016/12/21/forward-reverse-kl/`

KL distance drawbacks:

- non-symmetric

- the range is infinite if KL(P||Q) > KL(R||Q), we cannot conclude that R is closer to Q than P.

practice: how to calculate 1d Gassian Distribution Distance

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(range from 0 to 1) and zero only when they are equal.

F-divergence:
$D_f(p||q) = \int_x q(x)f(\frac{p(x)}{q(x)})dx$

if we set f(t) = tlogt or f(t) = -logt, we can get KL-divergence.

The relationship between GAN and KL-divergence:
Firstly, let's imagine 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):
$V(G, D) = \int_x [p_d(x)logD(x) + p_g(x)log(1 - D(x))]dx$
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
$D^*_G(x) = \frac{p_d(x)}{p_d(x)+p_g(x)}$

when D(x) get optimal, we can convert C(G) as:
$C(G) = max_D V(G, D) = -log(4) + E_{x\sim P_d}[log\frac{p_d}{\frac{p_d+p_g}{2}}] + E_{x\sim P_g}[log\frac{p_g}{\frac{p_d+p_g}{2}}]$
$= -log(4) + KL(p_d||\frac{p_d+p_g}{2}) + KL(p_g||\frac{p_d+p_g}{2})$
$= -log(4) + JS(p_d||p_g)$

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 $p_g = p_d$, i.e., the generative model perfectly replicating the data generating process.

12.1 Why is maxD then minG

reference in Boyed's Book 5.4.3

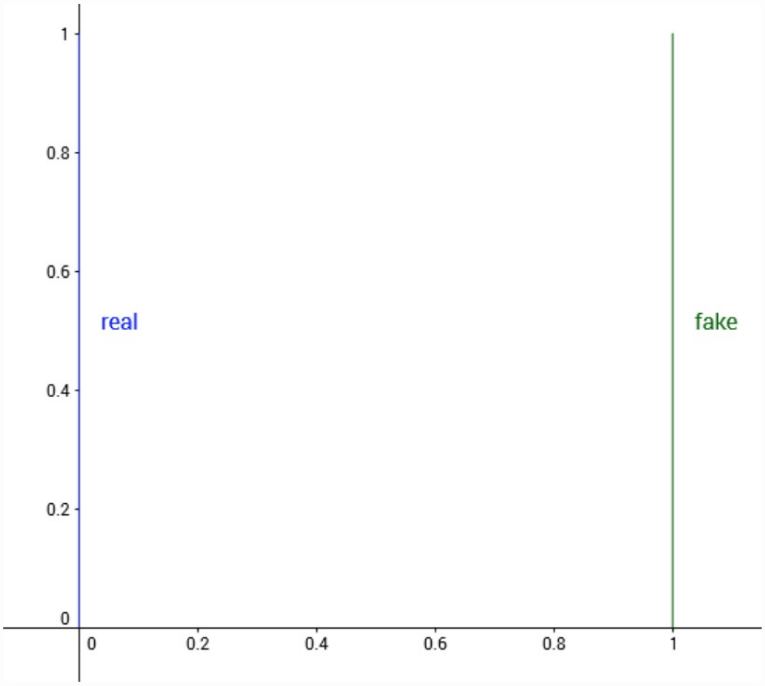
12.2 WGAN

`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.

12.3 Improved Training of Wasserstein GANs[12]

propose Wasserstein GAN (WGAN), which does not require a special network design like DCGAN. WGAN uses the Wasserstein distance to replace the original loss function in GAN and solves the unreliable gradient problem in the original GAN. Using Wasserstein distance also provides an estimate of the quality of the generated samples. However, since WGAN satisfies the K-Lipschitz constraint by weight clipping, it pushes weights towards two values (the extremes of the clipping range) and is hard to tune the clipping parameters. Gulrajani et al. [17] replace the weight clipping strategy by gradient penalty. Gradient penalty is an additional term in the loss function that directly enforces the discriminator's gradient norm around K. The result shows that the improved training strategy of [12] is much faster and more stable than WGAN.

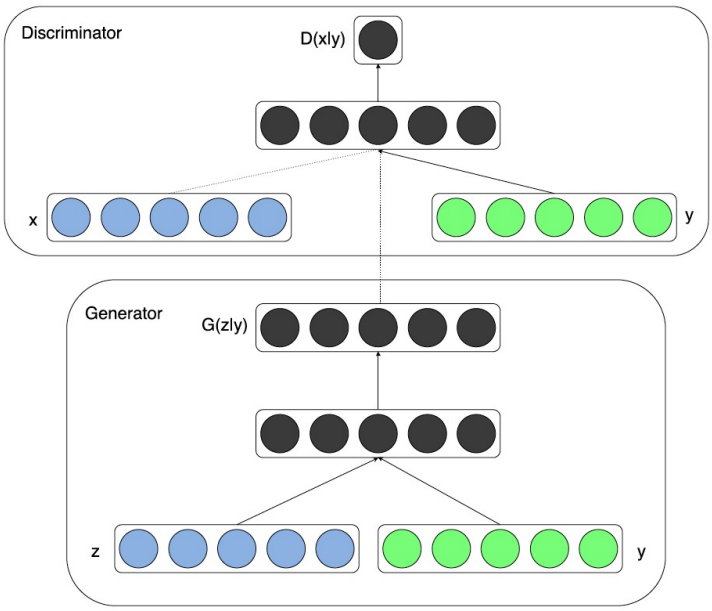
12.4 Conditional GAN[25]

only one thing: just use z,y as input

The objective function of a two-player minimax game would be as Eq 2

$$\min_G \max_D V(D,G) = \mathbb{E}_{\boldsymbol{x}\sim p_{\text{data}}(\boldsymbol{x})}[\log D(\boldsymbol{x}|\boldsymbol{y})] + \mathbb{E}_{\boldsymbol{z}\sim p_z(\boldsymbol{z})}[\log(1 - D(G(\boldsymbol{z}|\boldsymbol{y})))]. \quad (2)$$

Fig 1 illustrates the structure of a simple conditional adversarial net.



12.5 BEGAN: Boundary Equilibrium Generative Adversarial Networks[2]

The BEGAN objective is:

$$\begin{cases} \mathcal{L}_D = \mathcal{L}(x) - k_t.\mathcal{L}(G(z_D)) & \text{for } \theta_D \\ \mathcal{L}_G = \mathcal{L}(G(z_G)) & \text{for } \theta_G \\ k_{t+1} = k_t + \lambda_k(\gamma\mathcal{L}(x) - \mathcal{L}(G(z_G))) & \text{for each training step } t \end{cases}$$

13 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

13.1 Model-based Method

13.1.1 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.

13.1.2 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.

13.2 Model-Free Method

13.2.1 on-policy TD

TD0

TD(n)

TDλ

13.2.2 on-policy MCMC

13.2.3 off-policy method

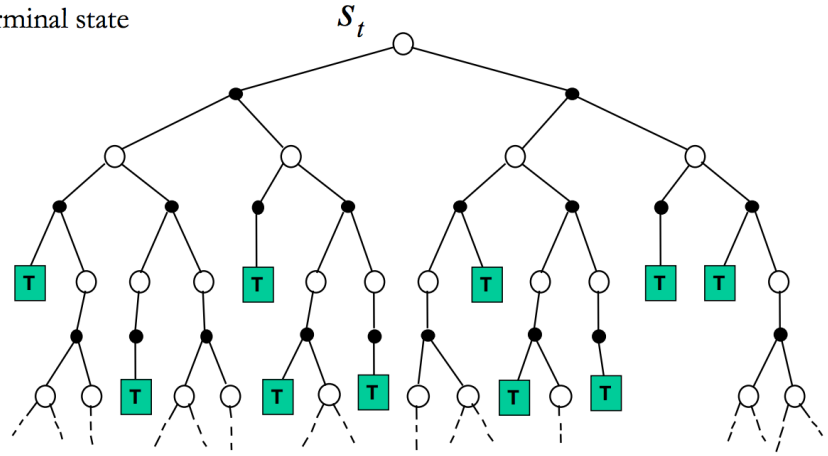
13.2.4 Comparisons: DP, MC, TD

Comparisons: DP, MC, TD

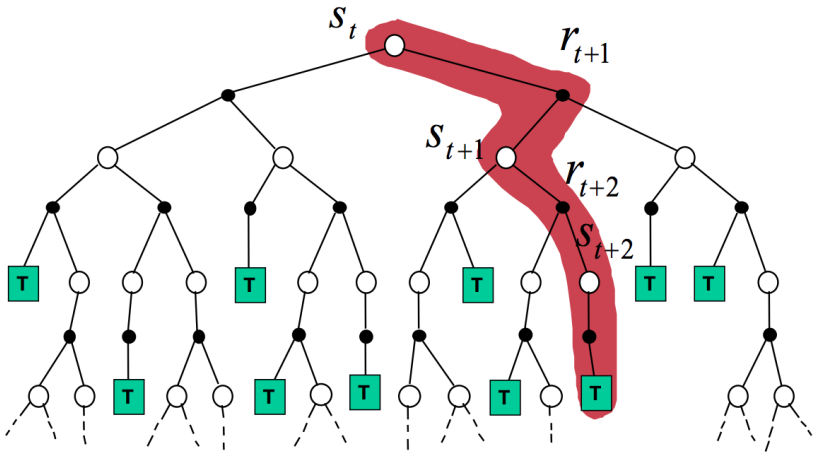
- They all estimate V^{π}
- DP: $V_k(s_t) \approx E_{\pi}\left(r_{t+1} + \gamma V_{k-1}(s_{t+1}) \mid s_t\right)$
  - Estimate comes from the Bellman equation
  - It needs to know the model
- TD: $V_k(s_t) \approx \left(r_{t+1} + \gamma V_{k-1}(s_{t+1})\right)$
  - Expectation is approximated with random 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

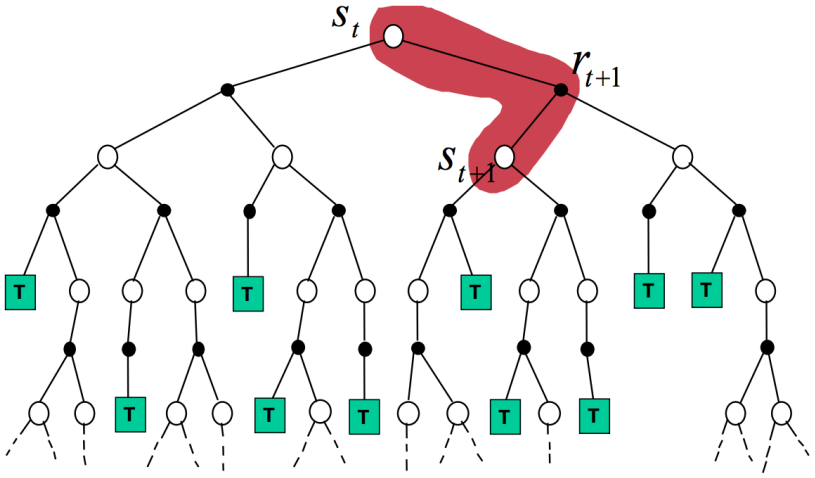


Monte Carlo Backup Diagram



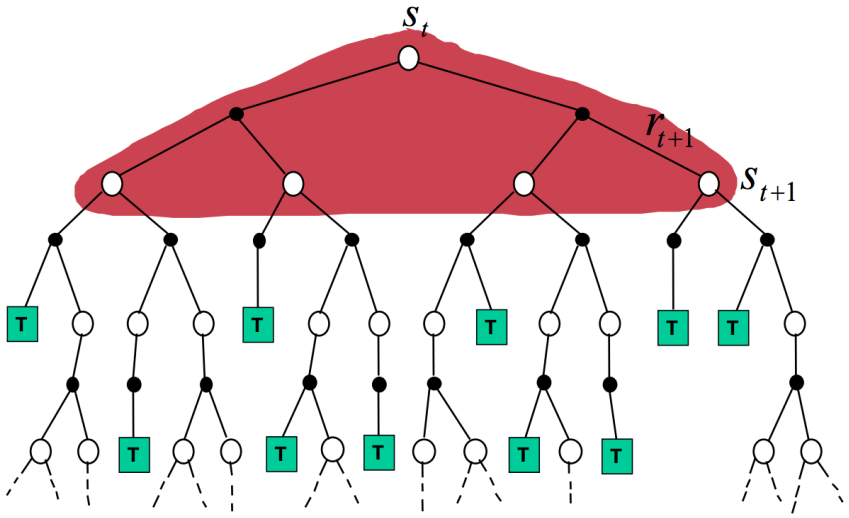
MC estimate: $V_k(s_t) := V_{k-1}(s_t) + \alpha_k \cdot (R_k(s_t) - V_{k-1}(s_t))$

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

14 RNN

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

14.1 BPTT

14.2 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:
$f_t = \sigma(W_f[c_{t-1}, h_{t-1}, x_t] + b_f)$
$i_t = \sigma(W_i[c_{t-1}, h_{t-1}, x_t] + b_i)$
$o_t = \sigma(W_o[c_t, h_{t-1}, x_t] + b_o)$

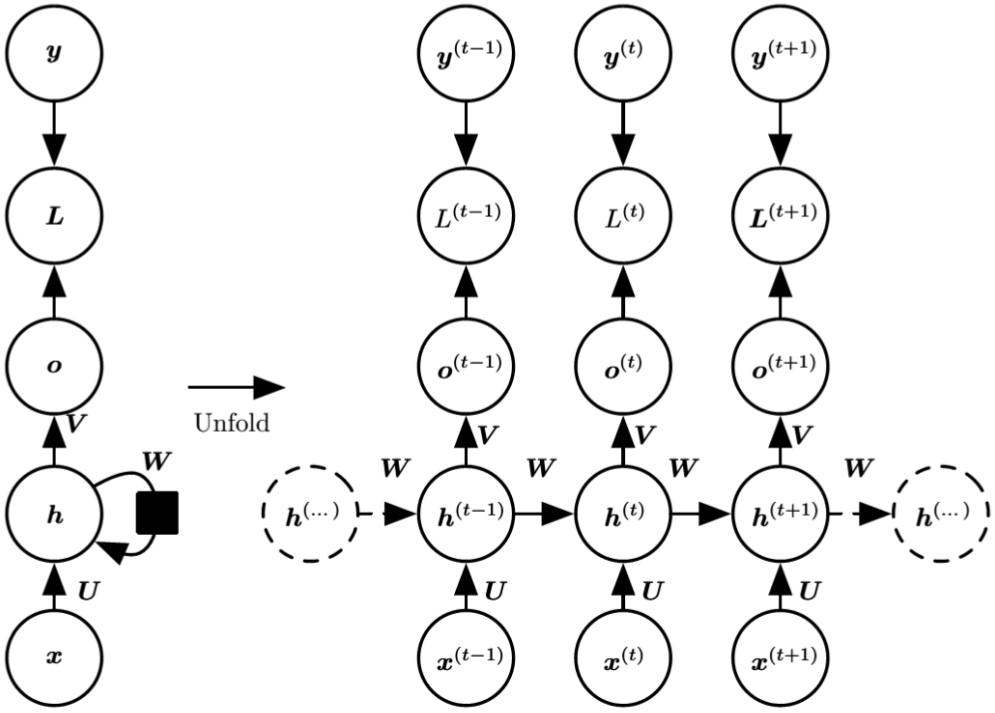
SRU:
$\tilde{x_t} = W x_t$
$f_t = \sigma(W_f x_t + b_f)$
$r_t = \sigma(W_r x_t + b_r)$
$c_t = f_t \circ c_{t-1} + (1 - f_t) \circ \tilde{x_t}$
$h_t = r_t \circ g(c_t) + (1 - r_t) \circ x_t$

the formula 1-3 is matrix multiplication, which is very time-consuming, however, the the formula 4-5 is Harmad Product,which is very computation cheap.

the only time dependence occurs in formula 4 between c_t and c_{t-1}. therefore, formula 1-3 can realize parallelization, which is originally very heavy(matrix multiplication) and formula 4,5 cannot realize parallelization,which is luckily very cheap in computation. In the whole, we can think that SRU is much faster than traditional LSTM.

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



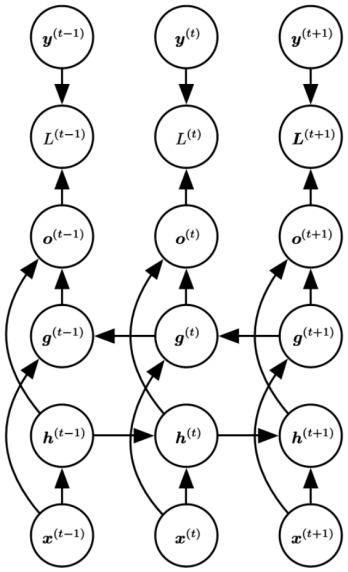
https://cs224d.stanford.edu/lecture_notes/LectureNotes4.pdf,the two RNNs can be imagined parallelized. the following is the structure of Bidirectional LSTM.

- 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)

14.3 Multidimension RNN

MDRNN[11] 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**
          $a \leftarrow a + \sum_j h_j^{(x_1,\ldots,x_i-1,\ldots,x_n)} w_{kj}$
      $h_k^{\mathbf{x}} \leftarrow tanh(a)$

Algorithm 1: MDRNN Forward Pass

15 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

16 Determinantal Point Processes

the following introduction is copied from [24].

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

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

a application of DPP is video summarization:[29].

17 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.

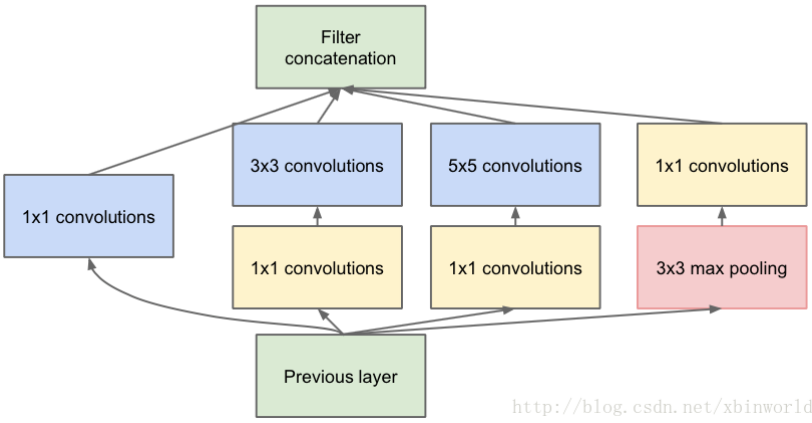
several mainstream:

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

17.1 VGGNet

17.2 Inception Series

17.2.1 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.

17.2.2 Inception v2,v3

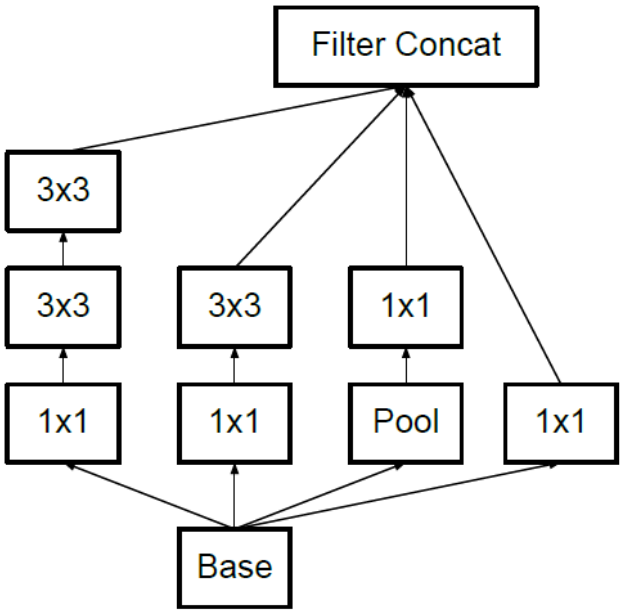


Figure 5. Inception modules where each 5×5 convolution is replaced by two 3×3 convolution, as suggested by principle 3 of Section 2.

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

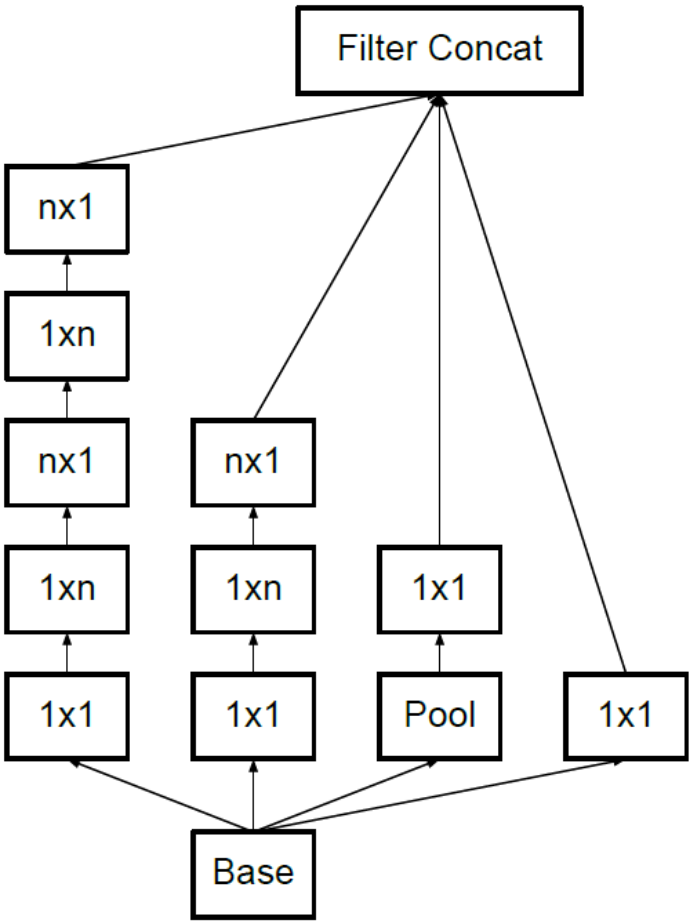


Figure 6. Inception modules after the factorization of the $n \times n$ convolutions. In our proposed architecture, we chose $n = 7$ for the 17×17 grid. (The filter sizes are picked using principle 3)

17.2.3 Inception v4

too complicated, ignore temporarily.

17.3 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.

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

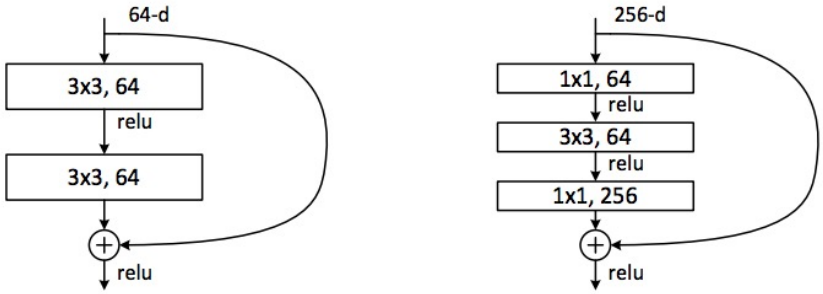


Figure 5. A deeper residual function $\mathcal{F}$ for ImageNet. Left: a building block (on 56×56 feature maps) as in Fig. 3 for ResNet-34. Right: a "bottleneck" building block for ResNet-50/101/152.

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				
		3×3 max pool, stride 2				
conv2_x	56×56	[3×3, 64; 3×3, 64] ×2	[3×3, 64; 3×3, 64] ×3	[1×1, 64; 3×3, 64; 1×1, 256] ×3	[1×1, 64; 3×3, 64; 1×1, 256] ×3	[1×1, 64; 3×3, 64; 1×1, 256] ×3
conv3_x	28×28	[3×3, 128; 3×3, 128] ×2	[3×3, 128; 3×3, 128] ×4	[1×1, 128; 3×3, 128; 1×1, 512] ×4	[1×1, 128; 3×3, 128; 1×1, 512] ×4	[1×1, 128; 3×3, 128; 1×1, 512] ×8
conv4_x	14×14	[3×3, 256; 3×3, 256] ×2	[3×3, 256; 3×3, 256] ×6	[1×1, 256; 3×3, 256; 1×1, 1024] ×6	[1×1, 256; 3×3, 256; 1×1, 1024] ×23	[1×1, 256; 3×3, 256; 1×1, 1024] ×36
conv5_x	7×7	[3×3, 512; 3×3, 512] ×2	[3×3, 512; 3×3, 512] ×3	[1×1, 512; 3×3, 512; 1×1, 2048] ×3	[1×1, 512; 3×3, 512; 1×1, 2048] ×3	[1×1, 512; 3×3, 512; 1×1, 2048] ×3
	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)

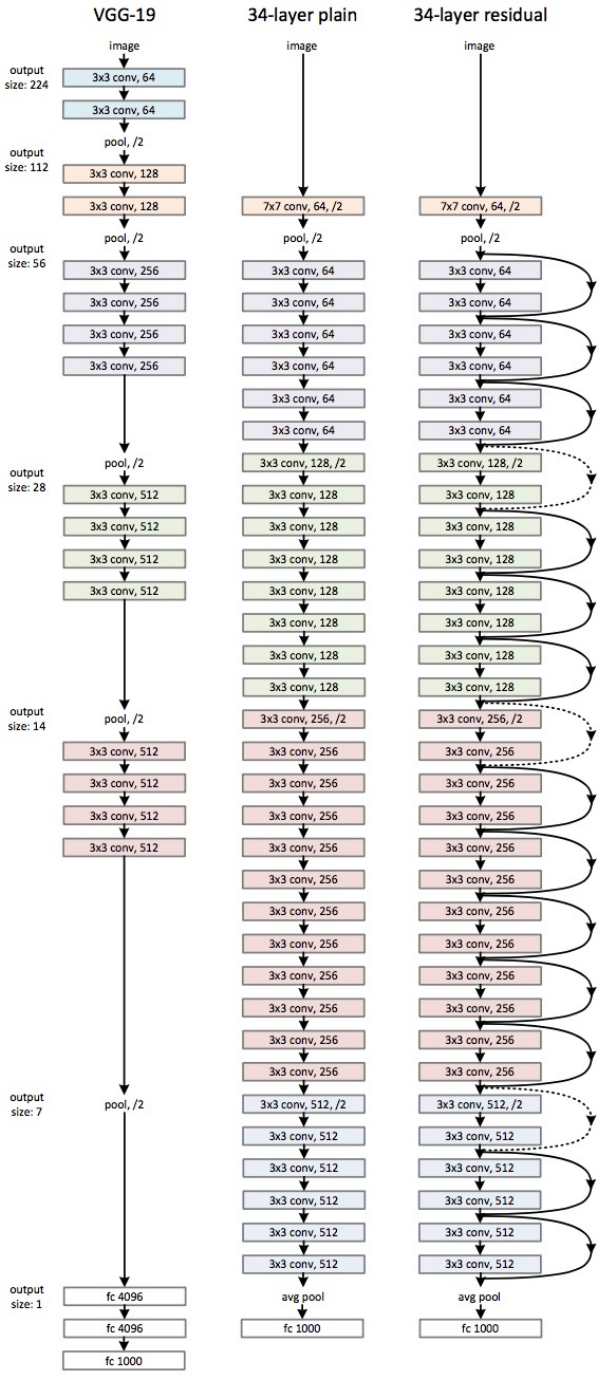


Figure 3. Example network architectures for ImageNet. **Left**: the VGG-19 model [40] (19.6 billion FLOPs) as a reference. **Middle**: a plain network with 34 parameter layers (3.6 billion FLOPs). **Right**: a residual network with 34 parameter layers (3.6 billion FLOPs). The dotted shortcuts increase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

17.4 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.

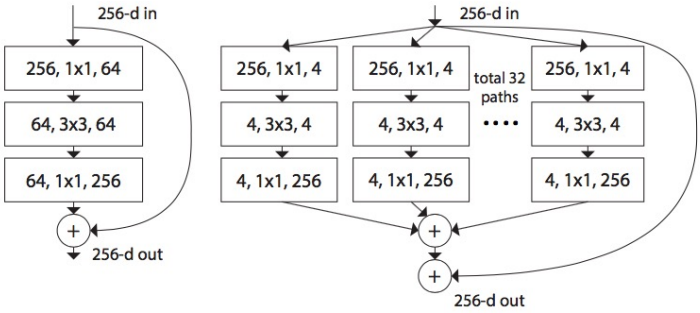


Figure 1. **Left**: A block of ResNet [13]. **Right**: A block of ResNeXt with cardinality = 32, with roughly the same complexity. A layer is shown as (# in channels, filter size, # out channels).

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
		[1×1, 64 3×3, 64 1×1, 256] ×3	[1×1, 128 3×3, 128, C=32 1×1, 256] ×3
conv3	28×28	[1×1, 128 3×3, 128 1×1, 512] ×4	[1×1, 256 3×3, 256, C=32 1×1, 512] ×4
conv4	14×14	[1×1, 256 3×3, 256 1×1, 1024] ×6	[1×1, 512 3×3, 512, C=32 1×1, 1024] ×6
conv5	7×7	[1×1, 512 3×3, 512 1×1, 2048] ×3	[1×1, 1024 3×3, 1024, C=32 1×1, 2048] ×3
	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

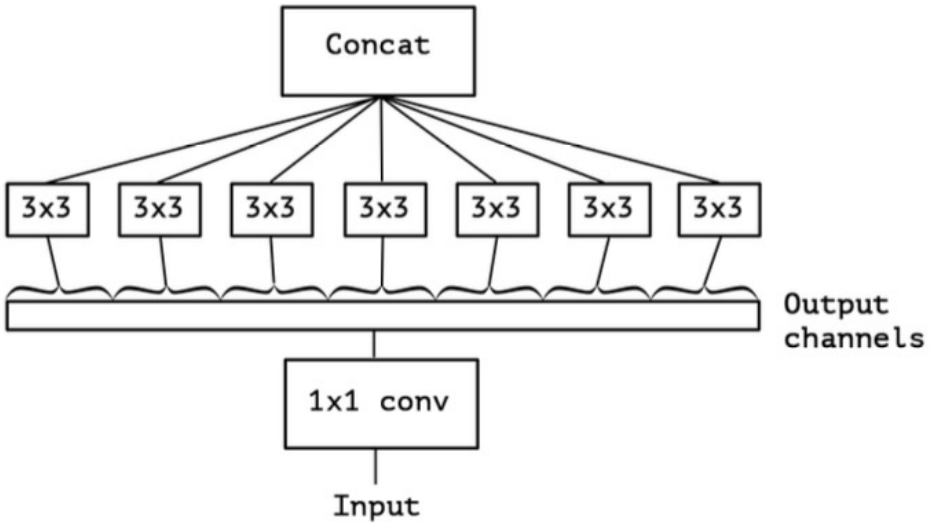
17.5 Xception

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`

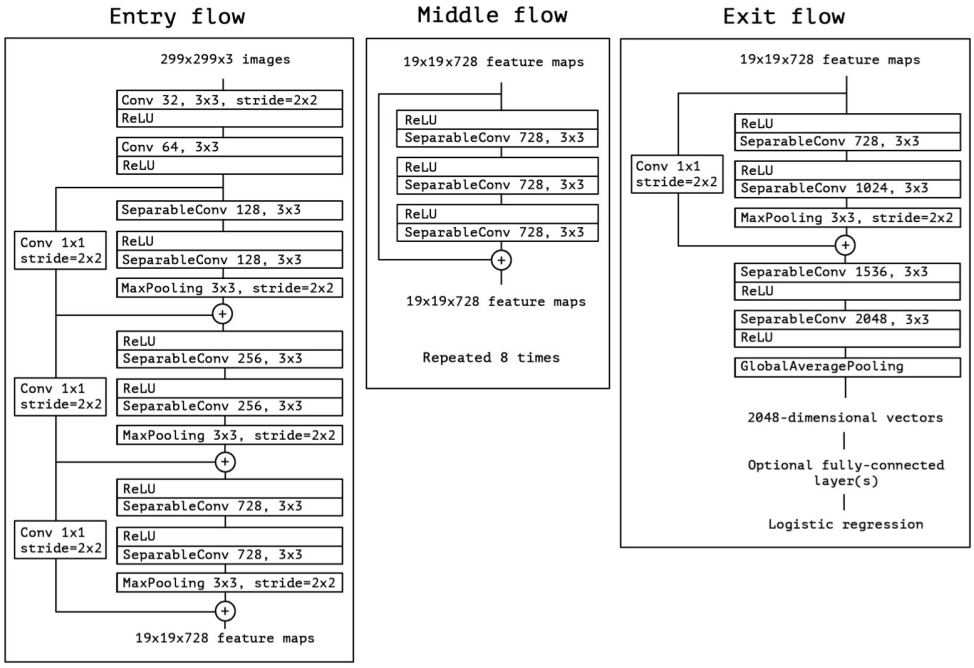
Figure 4. An "extreme" version of our Inception module, with one spatial convolution per output channel of the 1x1 convolution.



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.

17.6 DenseNet

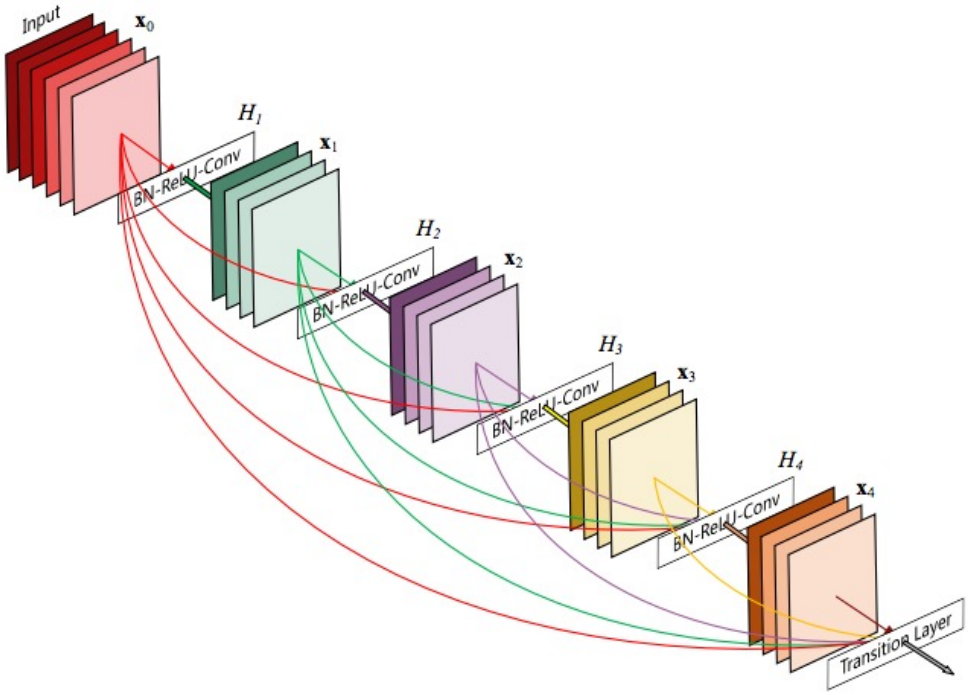


Figure 1. A 5-layer dense block with a growth rate of $k = 4$. Each layer takes all preceding feature-maps as input.

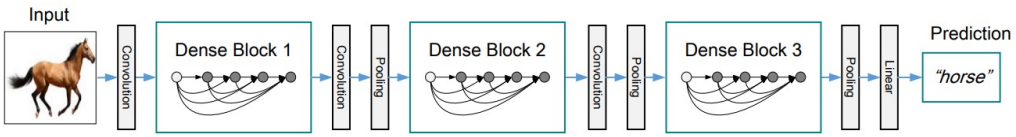


Figure 2. A deep DenseNet with three dense blocks. The layers between two adjacent blocks are referred to as transition layers and change feature map sizes via convolution and pooling.

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.

17.7 Interleaved Group Convolution

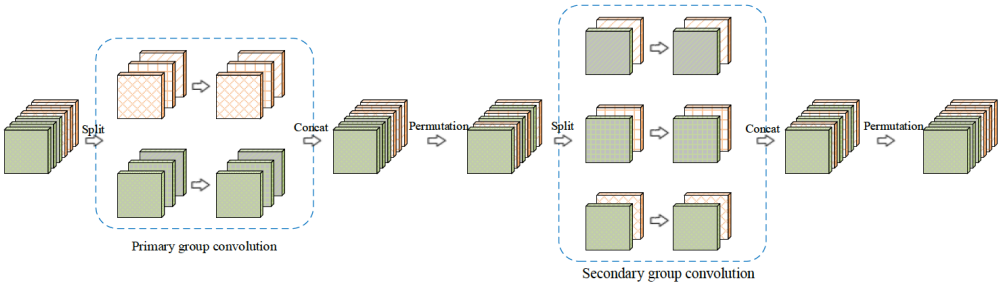


Figure 1. Illustrating the interleaved group convolution, with $L = 2$ primary partitions and $M = 3$ secondary partitions. The convolution for each primary partition in primary group convolution is spatial. The convolution for each secondary partition in secondary group convolution is point-wise (1×1). Details are given in Section 3.1.

- Xception Network is a specific instance of interleaved group convolutions when M=1

- Even you have known the idea, you could not write a paper as excellent as the author. the ability of formulation and summarization!

17.8 Accurate, Large Minibatch SGD: Training ImageNet in 1 Hour

- Linear Scaling Rule: When the minibatch size is multiplied by k, multiply the learning rate by k.

- Gradual warmup. We present an alternative warmup that gradually ramps up the learning rate from a small to a large value. This ramp avoids a sudden increase from a small learning rate to a large one, allowing healthy convergence at the start of training.

- We also note that the BN statistics should not be computed across all workers, not only for the sake of reducing communication, but also for maintaining the same underlying loss function being optimized.

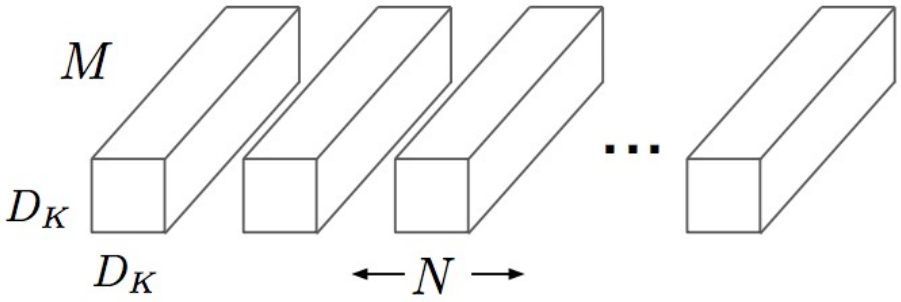
- Remarks:
  Remark 1: Scaling the cross-entropy loss is not equivalent to scaling the learning rate.
  Remark 2: Apply momentum correction after changing learning rate if using (10).
  Remark 3: Normalize the per-worker loss by total minibatch size kn, not per-worker size n.
  Remark 4: Use a single random shuffling of the training data (per epoch) that is divided amongst all k workers.

- we do not apply weight decay on the learnable BN coefficients β, γ.

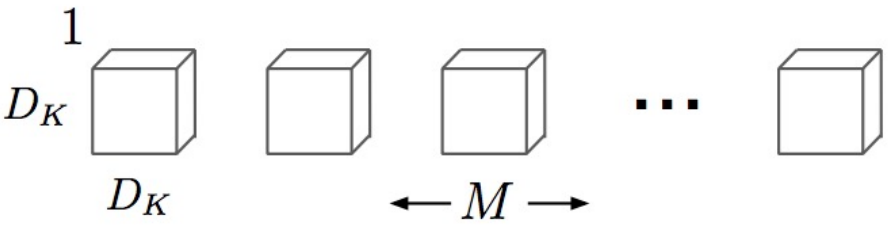
- For BN layers, the learnable scaling coefficient is initialized to be 1, except for each residual block's last BN where is initialized to be 0. Setting = 0 in the last BN of each residual block causes the forward/backward signal initially to propagate through the identity shortcut of ResNets, which we found to ease optimization at the start of training. This initialization improves all models but is particularly helpful for large minibatch training as we will show.

18 Network Compression

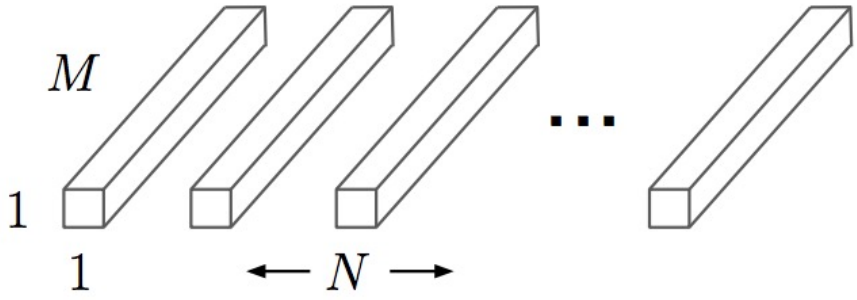
18.1 MobileNet



(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.

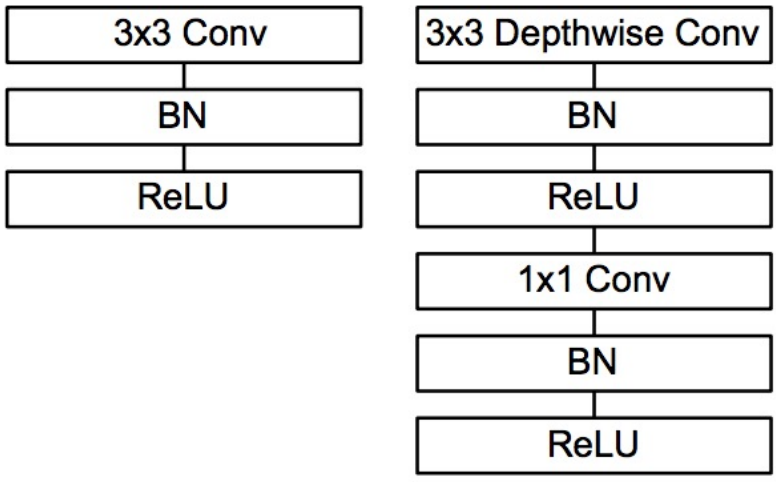


Figure 3. Left: Standard convolutional layer with batchnorm and ReLU. Right: Depthwise Separable convolutions with Depthwise and Pointwise layers followed by batchnorm and ReLU.

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.

18.2 ShuffleNet

http://ethereon.github.io/netscope/#/gist/952fa930c43576a94c805a045037b943

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.

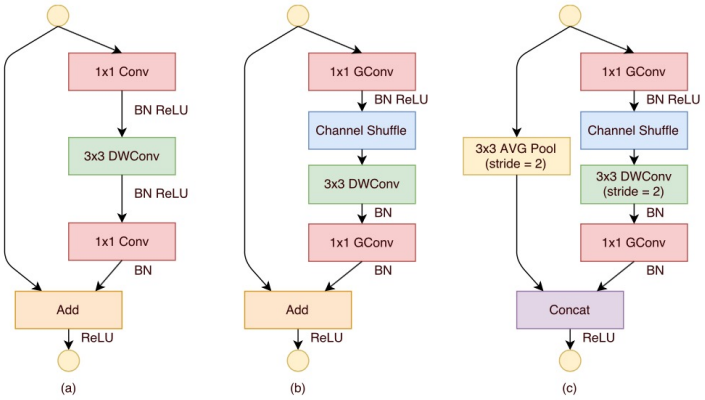
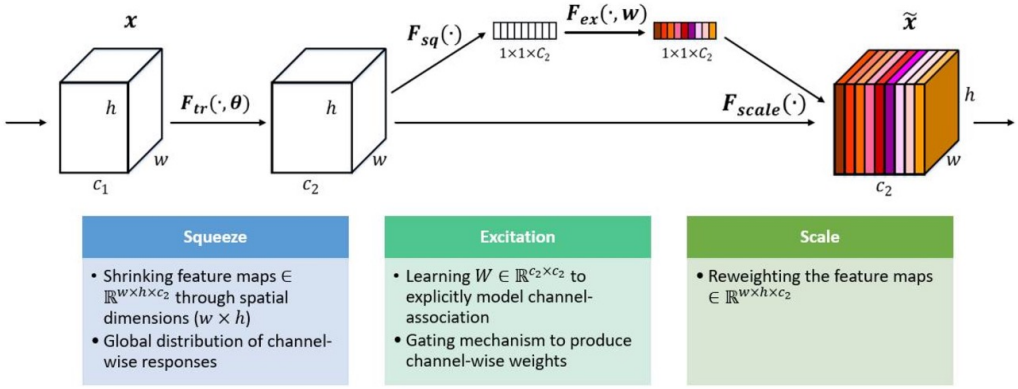
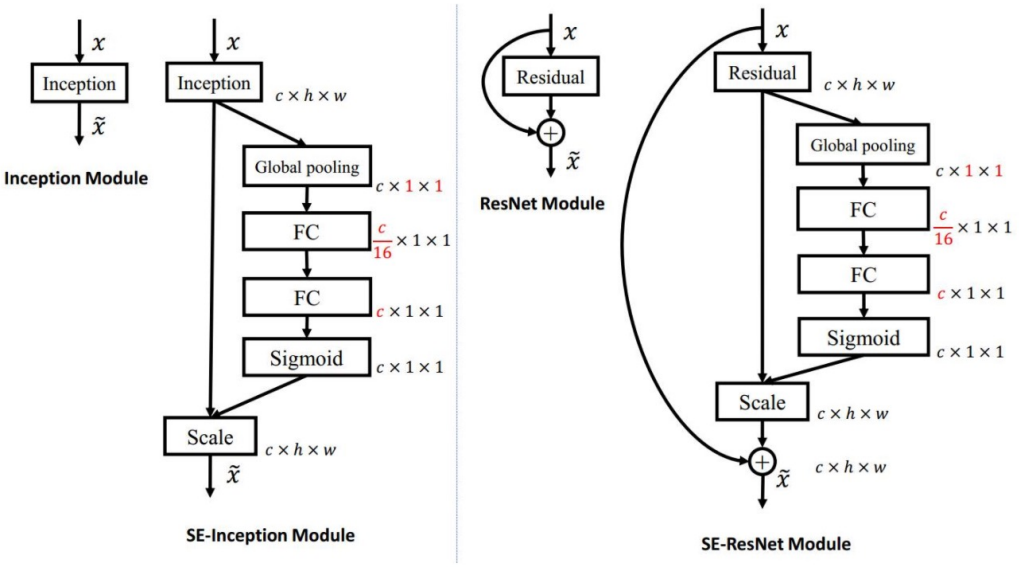


Figure 2: ShuffleNet Units. a) bottleneck unit [9] with depthwise convolution (DWConv) [3, 12]; b) ShuffleNet unit with pointwise group convolution (GConv) and channel shuffle; c) ShuffleNet unit with stride = 2.

18.3 SENet

Squeeze and Excitation Network(http://www.sohu.com/a/161633191_465975)





- label smooth
- The parameters (mean,variance) of all BN layers were frozen for the last few training epochs to ensure consistency between training and testing.
- first fc layer with ReLu, second fc layer with BN,ReLu

18.4 Dilated Residual Networks

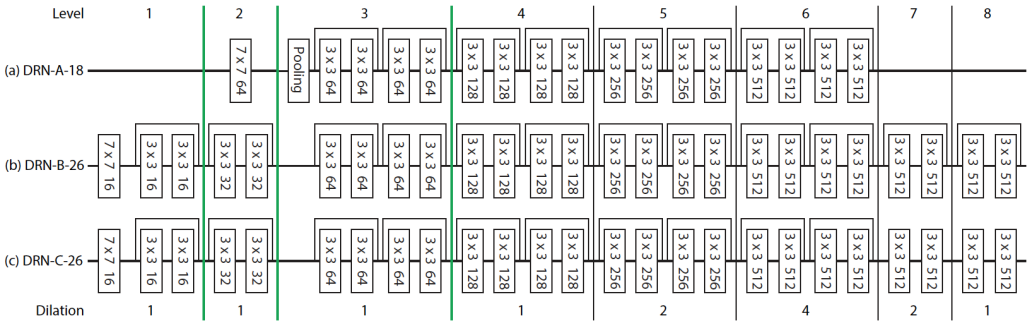


Figure 5: Changing the DRN architecture to remove gridding artifacts from the output activation maps. Each rectangle is a Conv-BN-ReLU group and the numbers specify the filter size and the number of channels in that layer. The bold green lines represent downsampling by stride 2. The networks are divided into levels, such that all layers within a given level have the same dilation and spatial resolution. (a) DRN-A dilates the ResNet model directly, as described in Section 2. (b) DRN-B replaces an early max pooling layer by residual blocks and adds residual blocks at the end of the network. (c) DRN-C removes residual connections from some of the added blocks. The rationale for each step is described in the text.

not only classification, but also in weak-supervised localization and segmentation

	Road	Sidewalk	Building	Wall	Fence	Pole	Light	Sign	Vegetation	Terrain	Sky	Person	Rider	Car	Truck	Bus	Train	Motorcycle	Bicycle	mean IoU
DRN-A-50	96.9	77.4	90.3	35.8	42.8	59.0	**66.8**	74.5	91.6	57.0	93.4	78.7	55.3	92.1	43.2	59.5	36.2	**52.0**	**75.2**	67.3
DRN-C-26	97.4	80.7	90.4	36.1	47.0	56.9	63.8	73.0	91.2	**57.9**	93.4	77.3	53.8	92.7	45.0	70.5	48.4	44.2	72.8	68.0
DRN-C-42	**97.7**	**82.2**	**91.2**	**40.5**	**52.6**	**59.2**	66.7	**74.6**	**91.7**	57.7	**94.1**	**79.1**	**56.0**	**93.6**	**56.0**	**74.3**	**54.7**	50.9	74.1	**70.9**

Table 3: Performance of dilated residual networks on the Cityscapes validation set. Higher is better. DRN-C-26 outperforms DRN-A-50, despite lower depth. DRN-C-42 achieves even higher accuracy. For reference, a comparable baseline setup of ResNet-101 was reported to achieve a mean IoU of 66.6.

19 Detection

19.1 SPPNet

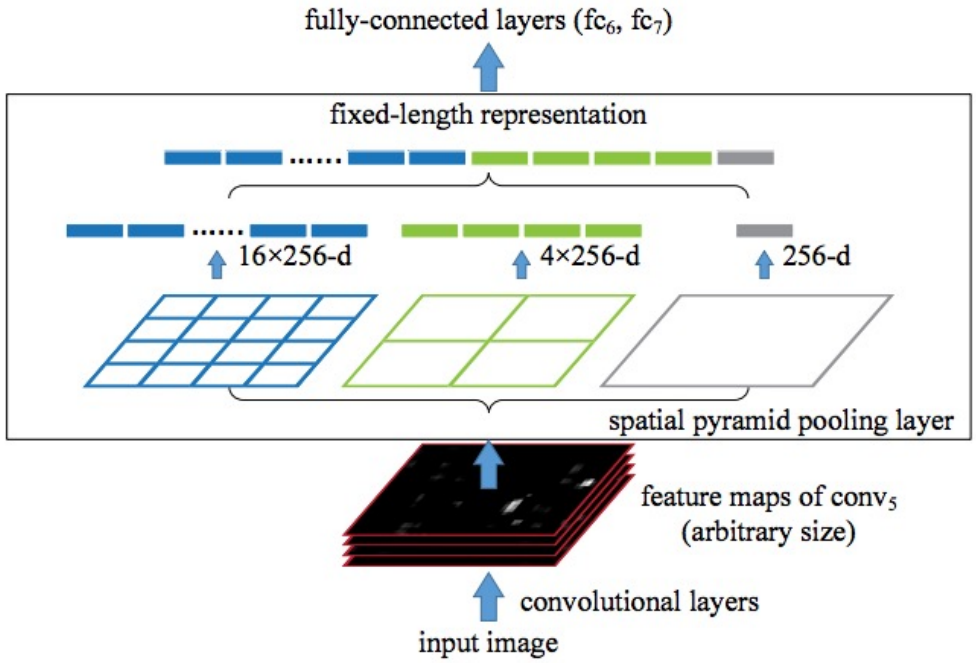


Figure 3: A network structure with a **spatial pyramid pooling layer**. Here 256 is the filter number of the $conv_5$ layer, and $conv_5$ is the last convolutional layer.

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.

19.2 Faster RCNN

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

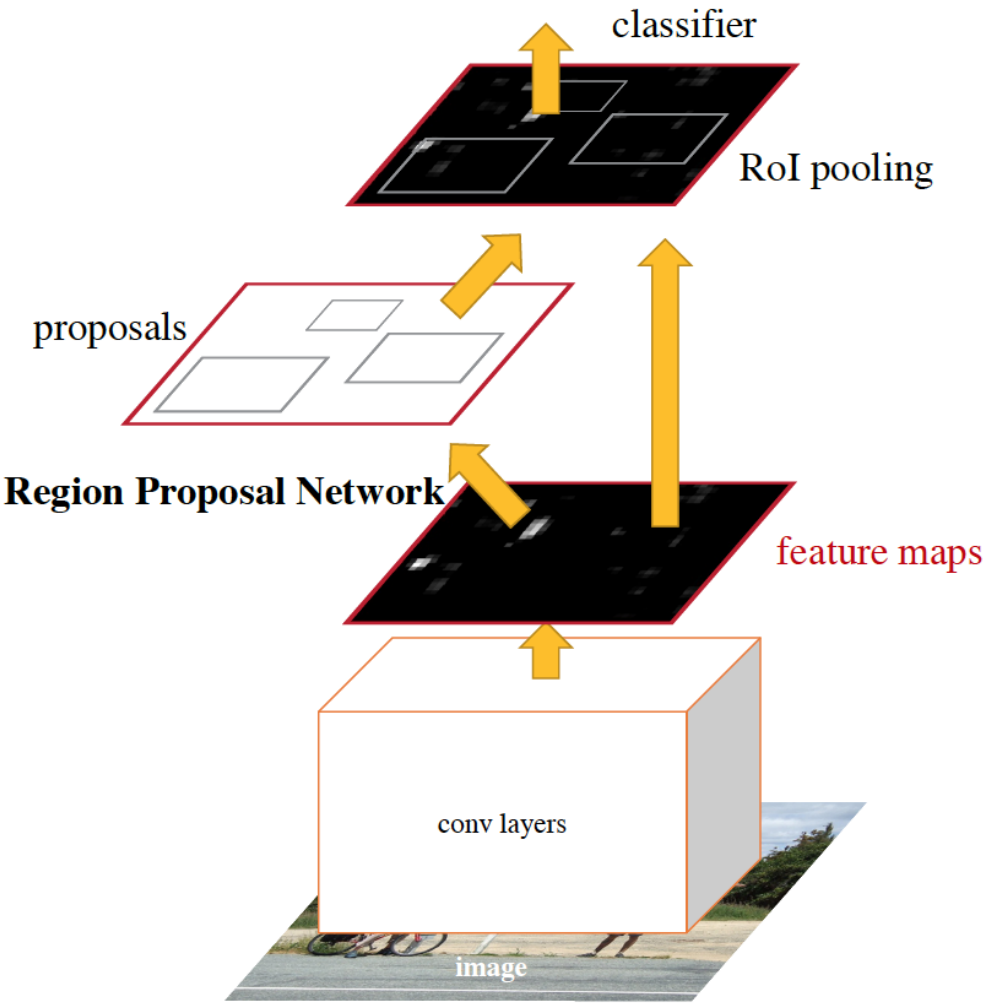


Figure 2: Faster R-CNN is a single, unified network for object detection. The RPN module serves as the 'attention' of this unified network.

19.3 Instance FCN[7]

task: instance proposal, can't segment instance!

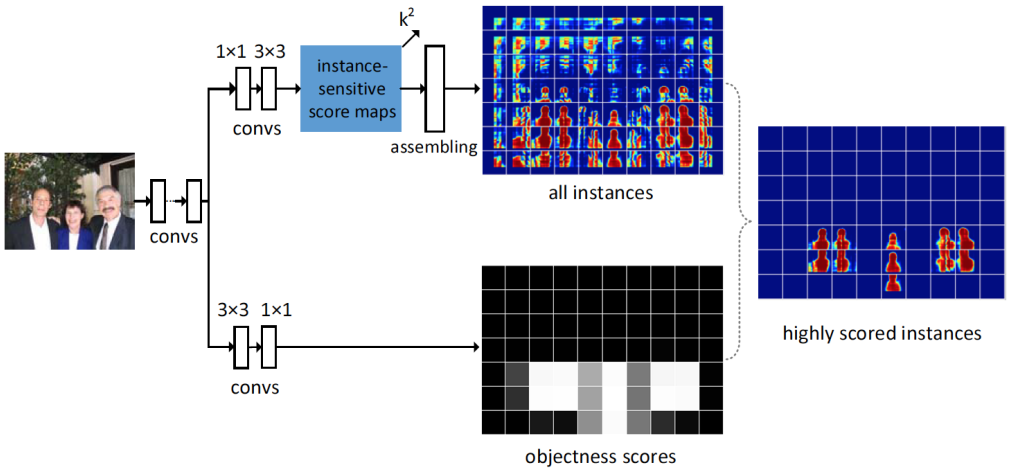


Figure 4. Details of the InstanceFCN architecture. On the top is a fully convolutional branch for generating k^2 instance-sensitive score maps, followed by the assembling module that outputs instances. On the bottom is a fully convolutional branch for predicting the objectness score of each window. The highly scored output instances are on the right. In this figure, the objectness map and the "all instances" map have been sub-sampled for the purpose of illustration.

In contrast to the original FCN where each output pixel is a classifier of an object category, we propose an FCN where each output pixel is a classifier of relative positions of instances.

How to Train:
Our network is trained end-to-end. We adopt the image-centric strategy in [21,19]. The forward pass computes the set of instance-sensitive score maps and the objectness score map. After that, a set of 256 sliding windows are randomly sampled [21,19], and the instances are only assembled from these 256 windows for computing the loss function. The loss function is defined as:

$$\sum_i (\mathcal{L}(p_i, p_i^*) + \sum_j \mathcal{L}(S_{ij}, S_{ij}^*))$$

Here i is the index of a sampled window, p_i is the predicted objectness score of the instance in this window, and p_i is 1 if this window is a positive sample and 0 if a negative sample. S_i is the assembled segment instance in this window, S_i is the ground truth segment instance, and j is the pixel index in the window. L is the logistic regression loss.

Notice:

- no RPN,only sliding windows(sparsely when training, densely when inference)
- Instance-sensitive score maps,Instance assembling module
- InstanceFCN mainly used to generate mask proposal, however, it cannot discriminate the semantic label. therefore need a post processing network to help discriminate the semantic label, which means the InstanceFCN is not an end-to-end structure.

19.4 R-FCN[20]

task: object detection!

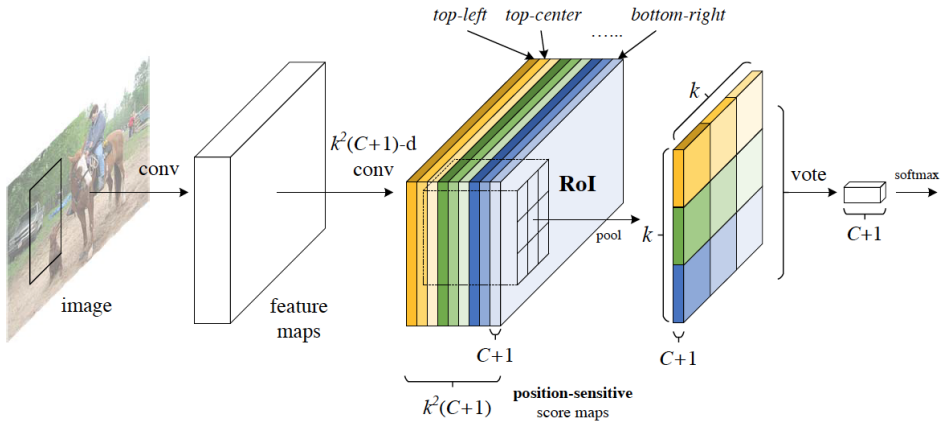


Figure 1: Key idea of **R-FCN** for object detection. In this illustration, there are $k \times k = 3 \times 3$ position-sensitive score maps generated by a fully convolutional network. For each of the $k \times k$ bins in an RoI, pooling is only performed on one of the k^2 maps (marked by different colors).

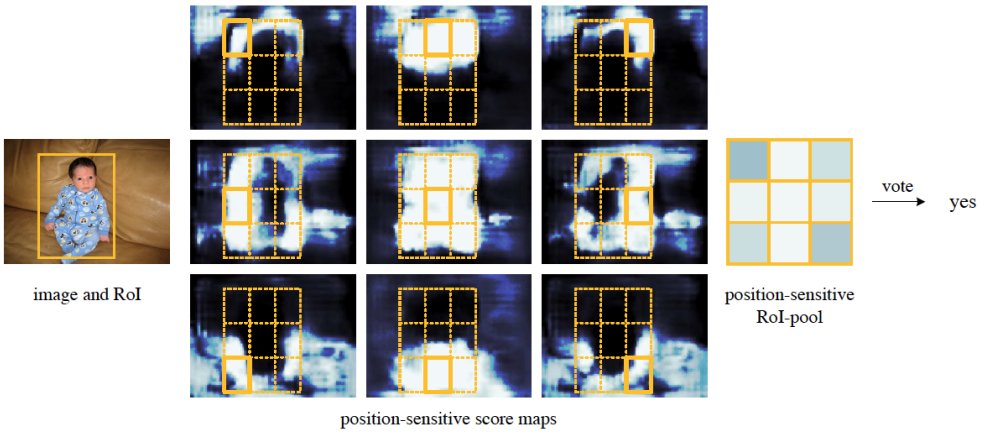


Figure 3: Visualization of R-FCN ($k \times k = 3 \times 3$) for the *person* category.

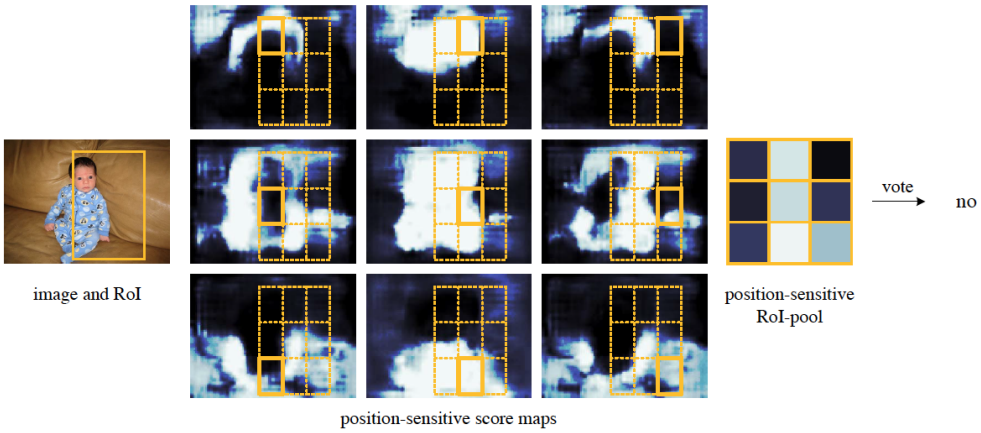


Figure 4: Visualization when an RoI does not correctly overlap the object.

How to train:
With pre-computed region proposals, it is easy to end-to-end train the R-FCN architecture. Following

Fast-RCNN[10], our loss function defined on each RoI is the summation of the cross-entropy loss and the box regression loss:

$$L(s, t_{x,y,w,h}) = L_{cls}(s_{c^*}) + \lambda[c^* > 0)]L_{reg}(t, t^*)$$

Here c is the RoI's ground-truth label (c=0 means background). is the cross-entropy loss for classification, L_{reg} is the bounding box regression loss as defined in fast rcnn[10], and t is ground truth box. [c>0] is an indicator which equals to 1 if the argument is true and 0 otherwise. we set the balance weight as 1 in fast-RCNN[10].

How to inference:
the feature maps shared between RPN and R-FCN are computed (on an image with a single scale of 600). Then the RPN part proposes RoIs, on which the R-FCN part evaluates category-wise scores and regresses bounding boxes. During inference we evaluate 300 RoIs as in [18] for fair comparisons. The results are post-processed by non-maximum suppression (NMS) using a threshold of 0.3 IoU, as standard practice.

19.5 FCIS[21]

task: instance segmentation!

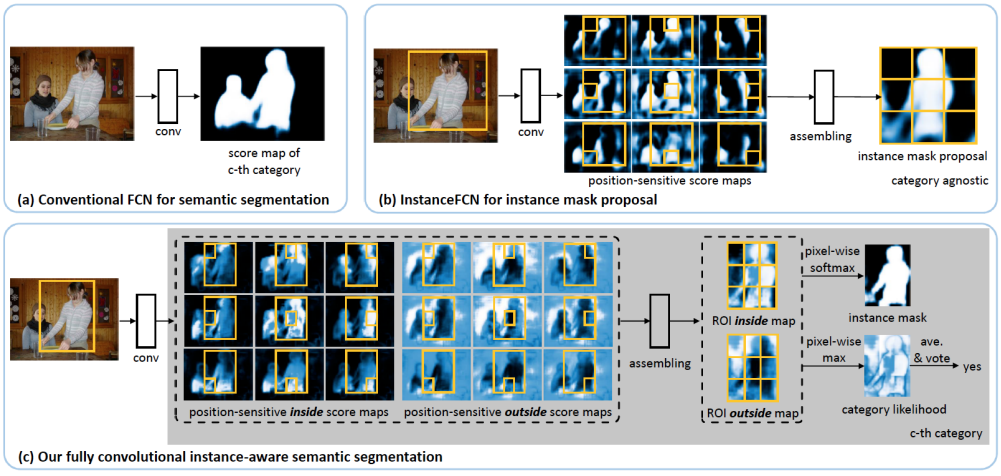


Figure 1. Illustration of our idea. (a) Conventional fully convolutional network (FCN) [29] for semantic segmentation. A single score map is used for each category, which is unaware of individual object instances. (b) InstanceFCN [5] for instance segment proposal, where 3×3 position-sensitive score maps are used to encode relative position information. A downstream network is used for segment proposal classification. (c) Our fully convolutional instance-aware semantic segmentation method (FCIS), where position-sensitive inside/outside score maps are used to perform object segmentation and detection jointly and simultanously.

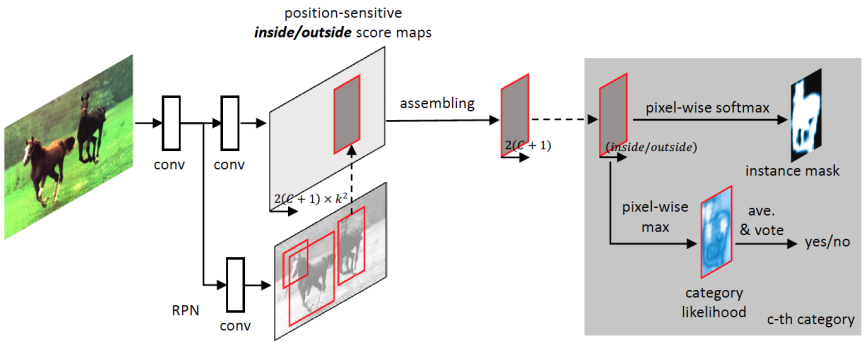


Figure 3. Overall architecture of FCIS. A region proposal network (RPN) [34] shares the convolutional feature maps with FCIS. The proposed region-of-interests (ROIs) are applied on the score maps for joint object segmentation and detection. The learnable weight layers are fully convolutional and computed on the whole image. The per-ROI computation cost is negligible.

How to train:
An ROI is positive if its box IoU with respect to the nearest ground truth object is larger than 0:5, otherwise it is negative. Each ROI has three loss terms in equal weights: a softmax detection loss over C + 1 categories, a softmax segmentation loss 1 over the foreground mask of the ground-truth category only, and a bbox regression loss as in Fast RCNN. The latter two loss terms are effective only on the positive ROIs.

Notice:

-

FCIS exhibits systematic errors on overlapping instances and creates spurious edges (Figure 5), showing that it is challenged by the fundamental difficulties of segmenting instances.

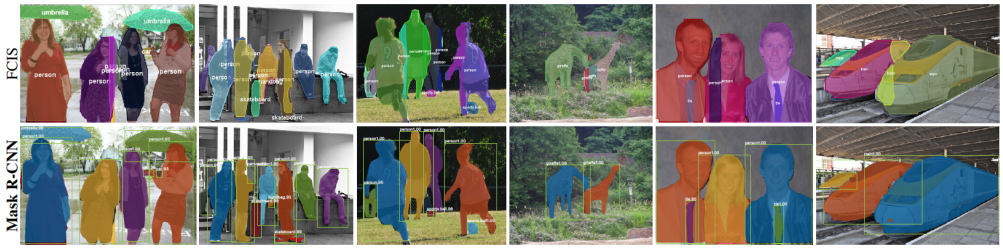


Figure 5. FCIS+++ [20] (top) *vs*. Mask R-CNN (bottom, ResNet-101-FPN). FCIS exhibits systematic artifacts on overlapping objects.

19.6 Mask-RCNN[13]

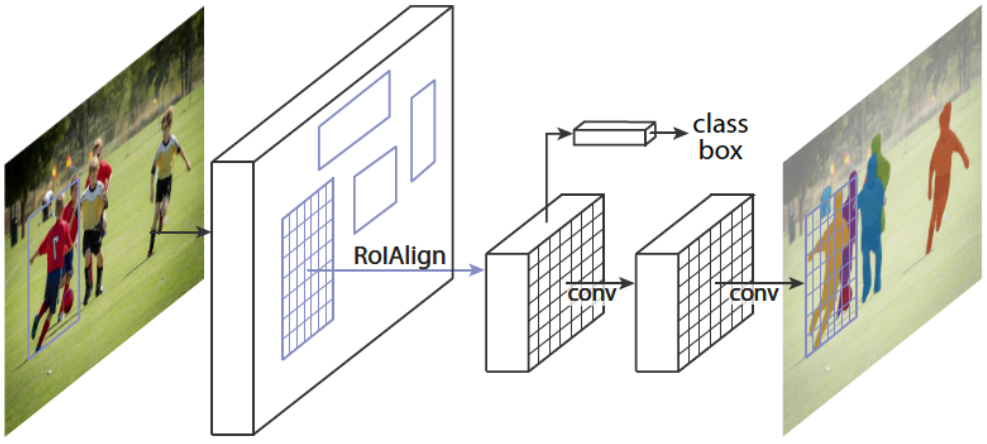


Figure 1. The **Mask R-CNN** framework for instance segmentation.

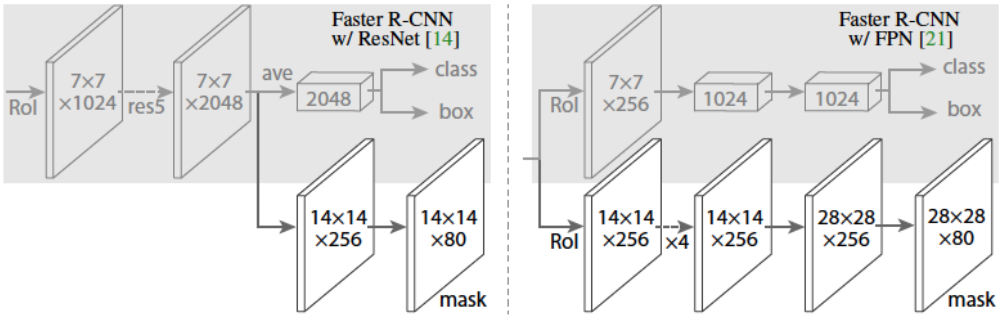


Figure 3. **Head Architecture**: We extend two existing Faster R-CNN heads [14, 21]. Left/Right panels show the heads for the ResNet C4 and FPN backbones, from [14] and [21], respectively, to which a mask branch is added. Numbers denote spatial resolution and channels. Arrows denote either conv, deconv, or *fc* layers as can be inferred from context (conv preserves spatial dimension while deconv increases it). All convs are 3×3, except the output conv which is 1×1, deconvs are 2×2 with stride 2, and we use ReLU [24] in hidden layers. *Left*: 'res5' denotes ResNet's fifth stage, which for simplicity we altered so that the first conv operates on a 7×7 RoI with stride 1 (instead of 14×14 / stride 2 as in [14]). *Right*: '×4' denotes a stack of four consecutive convs.

About Mask

Formally, during training, we define a multi-task loss on each sampled RoI as $L = L_{cls} + L_{box} + L_{mask}$. The classification loss L_{cls} and bounding-box loss L_{box} are identical as those defined in [9]. The mask branch has a Km^2-d dimensional output for each RoI, which encodes K binary masks of resolution $m * m$, one for each of the K classes. To this we apply a per-pixel sigmoid, and define L_{mask} as the average binary cross-entropy loss. For an RoI associated with ground-truth class k, L_{mask} is only defined on the k-th mask (other mask outputs do not contribute to the loss).

About ROIAlign

we propose an RoIAlign layer that removes the harsh quantization of RoIPool, properly aligning the extracted features with the input. Our proposed change is simple: we avoid any quantization of the RoI boundaries or bins (i.e., we use x/16 instead of [x/16]). We use bilinear interpolation [17] to compute the exact values of the input features at four regularly sampled locations in each RoI bin, and aggregate the result (using max or average).

Summary:

- ROIAlign is good
-

19.7 FPN[22]

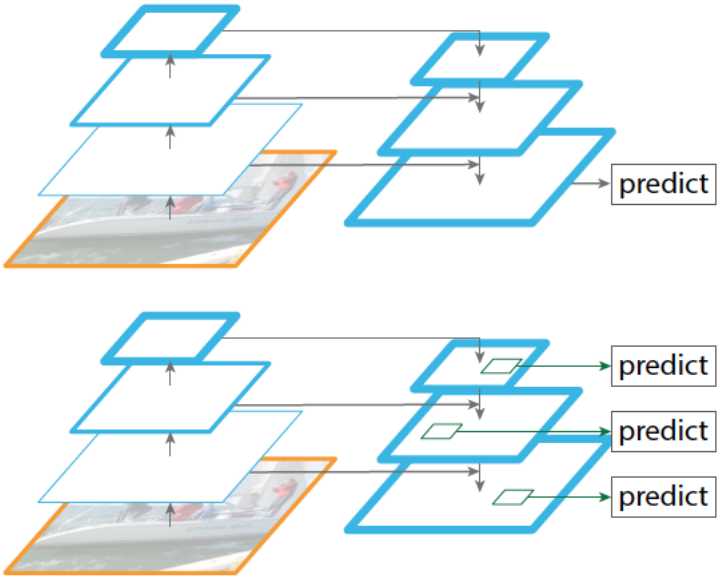


Figure 2. Top: a top-down architecture with skip connections, where predictions are made on the finest level (*e.g.*, [28]). Bottom: our model that has a similar structure but leverages it as a *feature pyramid*, with predictions made independently at all levels.

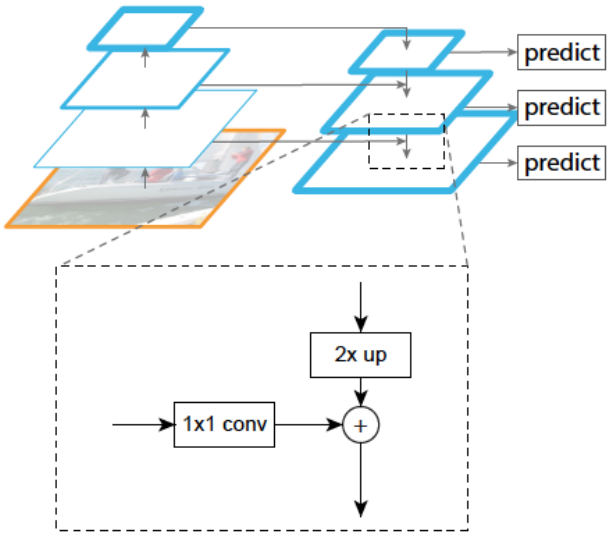


Figure 3. A building block illustrating the lateral connection and the top-down pathway, merged by addition.

How to apply into RPN
RPN [29] is a sliding-window class-agnostic object detector. In the original RPN design, a small

subnetwork is evaluated on dense 3*3 sliding windows, on top of a singlescale convolutional feature map, performing object/nonobject binary classification and bounding box regression. This is realized by a 3*3 convolutional layer followed by two sibling 1*1 convolutions for classification and regression, which we refer to as a network head. The object/nonobject criterion and bounding box regression target are defined with respect to a set of reference boxes called anchors [29]. The anchors are of multiple pre-defined scales and aspect ratios in order to cover objects of different shapes.

How to apply into object detection
Fast R-CNN [11] is a region-based object detector in which Region-of-Interest (RoI) pooling is used to extract features. Fast R-CNN is most commonly performed on a single-scale feature map. To use it with our FPN, we need to assign RoIs of different scales to the pyramid levels.

Summary

- trivial idea

19.8 MNC

19.9 SSD

19.10 YOLO

19.11 DSOD

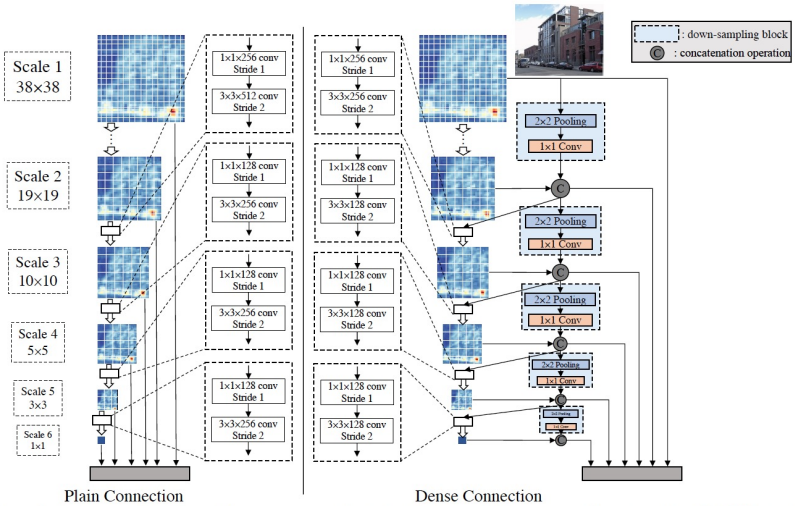


Figure 1: DSOD prediction layers with plain and dense structures (for 300×300 input). Plain structure is introduced by SSD [21] and dense structure is ours. See Section 3 for more details.

Layers		Output Size (Input 3×300 × 300)	DSOD
Stem	Convolution	64×150×150	3×3 conv, stride 2
	Convolution	64×150×150	3×3 conv, stride 1
	Convolution	128×150×150	3×3 conv, stride 1
	Pooling	128×75×75	2×2 max pool, stride 2
Dense Block (1)		416×75×75	$\begin{bmatrix} 1\times 1 \text{ conv} \\ 3\times 3 \text{ conv} \end{bmatrix} \times 6$
Transition Layer (1)		416×75×75	1×1 conv
		416×38×38	2×2 max pool, stride 2
Dense Block (2)		800×38×38	$\begin{bmatrix} 1\times 1 \text{ conv} \\ 3\times 3 \text{ conv} \end{bmatrix} \times 8$
Transition Layer (2)		800×38×38	1×1 conv
		800×19×19	2×2 max pool, stride 2
Dense Block (3)		1184×19×19	$\begin{bmatrix} 1\times 1 \text{ conv} \\ 3\times 3 \text{ conv} \end{bmatrix} \times 8$
Transition w/o Pooling Layer (1)		1120×19×19	1×1 conv
Dense Block (4)		1568×19×19	$\begin{bmatrix} 1\times 1 \text{ conv} \\ 3\times 3 \text{ conv} \end{bmatrix} \times 8$
Transition w/o Pooling Layer (2)		1568×19×19	1×1 conv
DSOD Prediction Layers		–	Plain/Dense

Table 1: DSOD architecture (growth rate $k = 48$ in each dense block).

- Stem block is important! Motivated by Inception-v3 [33] and v4, we define stem block as a stack of three 3*3 convolution layers followed by a 2*2 max pooling layer. The first conv-layer works with stride = 2 and the other two are with stride = 1. We find that adding this simple stem structure can evidently improve the detection performance in our experiments. We conjecture that, compared with the original design in DenseNet (7*7 conv layer, stride = 2 followed by a 3*3 max pooling, stride = 2), the stem block can reduce the information loss from raw input images.

19.12 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).

20 Segmentation

20.1 PSPNet

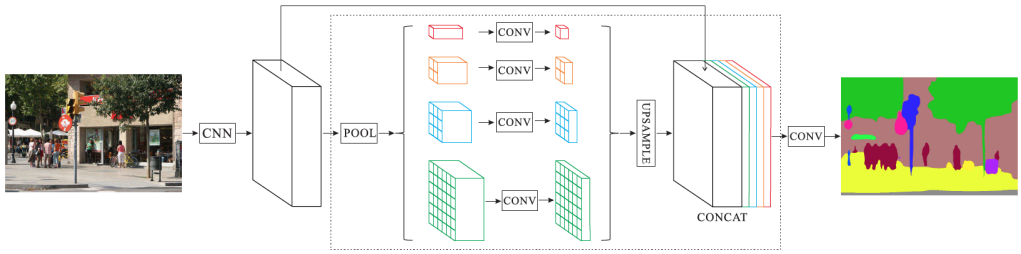


Figure 3. Overview of our proposed PSPNet. Given an input image (a), we first use CNN to get the feature map of the last convolutional layer (b), then a pyramid parsing module is applied to harvest different sub-region representations, followed by upsampling and concatenation layers to form the final feature representation, which carries both local and global context information in (c). Finally, the representation is fed into a convolution layer to get the final per-pixel prediction (d).

Method	Mean IoU(%)	Pixel Acc.(%)
PSPNet(50)	41.68	80.04
PSPNet(101)	41.96	80.64
PSPNet(152)	42.62	80.80
PSPNet(269)	**43.81**	**80.88**
PSPNet(50)+MS	42.78	80.76
PSPNet(101)+MS	43.29	81.39
PSPNet(152)+MS	43.51	81.38
PSPNet(269)+MS	**44.94**	**81.69**

Table 3. Deeper pre-trained model get higher performance. Number in the brackets refers to the depth of ResNet and 'MS' denotes multi-scale testing.

Method	Mean IoU(%)	Pixel Acc.(%)
FCN [26]	29.39	71.32
SegNet [2]	21.64	71.00
DilatedNet [40]	32.31	73.55
CascadeNet [43]	34.90	74.52
ResNet50-Baseline	34.28	76.35
ResNet50+DA	35.82	77.07
ResNet50+DA+AL	37.23	78.01
ResNet50+DA+AL+PSP	**41.68**	**80.04**
ResNet269+DA+AL+PSP	43.81	80.88
ResNet269+DA+AL+PSP+MS	**44.94**	**81.69**

Table 4. Detailed analysis of our proposed PSPNet with comparison with others. Our results are obtained on the validation set with the single-scale input except for the last row. Results of FCN, SegNet and DilatedNet are reported in [43]. 'DA' refers to data augmentation we performed, 'AL' denotes the auxiliary loss we added and 'PSP' represents the proposed PSPNet. 'MS' means that multi-scale testing is used.

Rank	Team Name	Final Score (%)
1	**Ours**	**57.21**
2	Adelaide	56.74
3	360+MCG-ICT-CAS_SP	55.56
-	(our single model)	(55.38)
4	SegModel	54.65
5	CASIA_IVA	54.33
-	DilatedNet [40]	45.67
-	FCN [26]	44.80
-	SegNet [2]	40.79

Table 5. Results of ImageNet scene parsing challenge 2016. The best entry of each team is listed. The final score is the mean of Mean IoU and Pixel Acc. Results are evaluated on the testing set.

20.2 Deformable Convolution Network

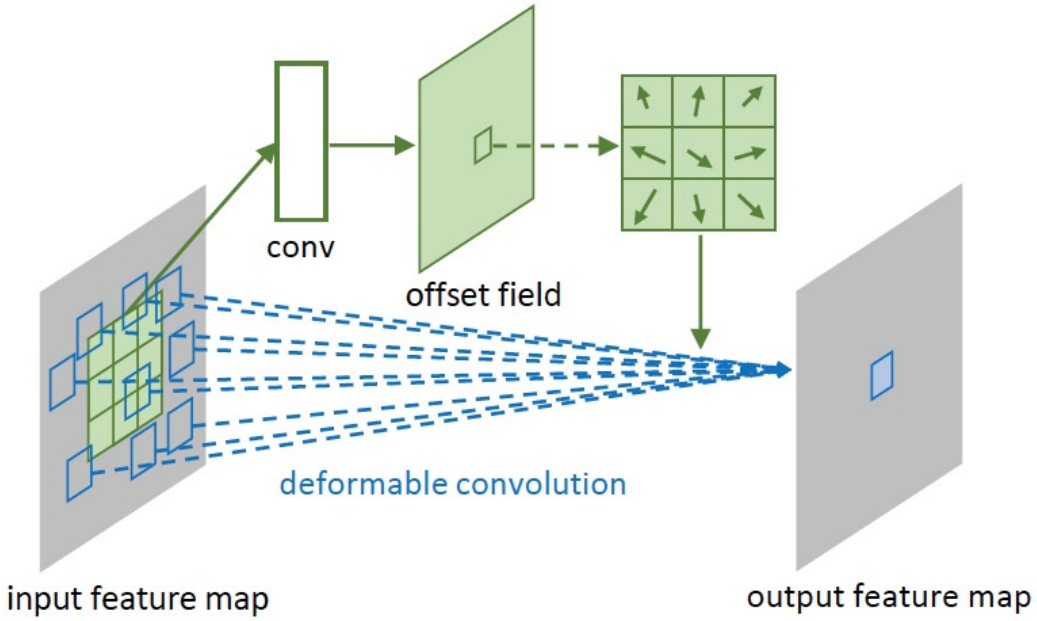


Figure 2: Illustration of 3×3 deformable convolution.

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 $G(q,p) = g(q_x, p_x) g(q_y, p_y)$
$g(a,b) = max(0, 1 - |a - b|)$

21 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!

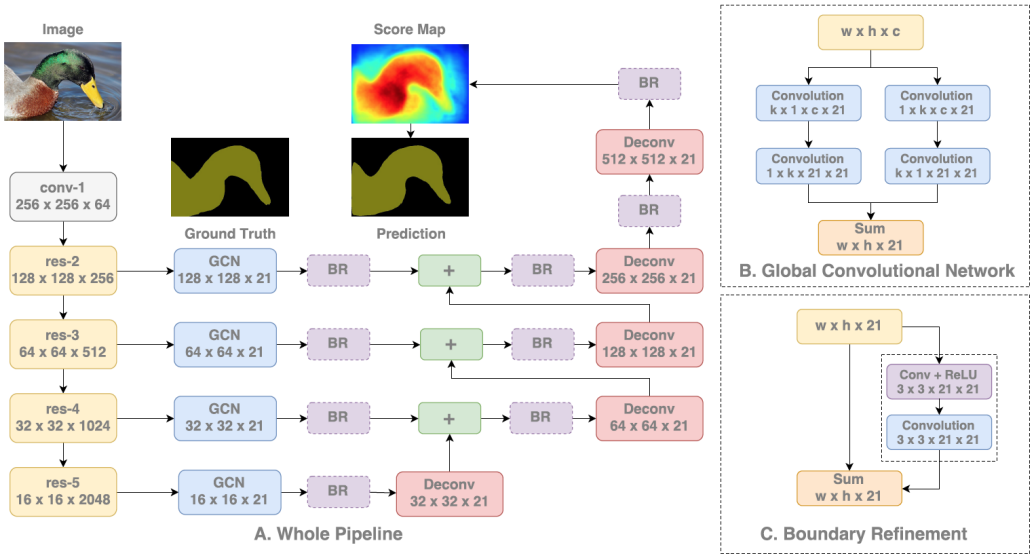
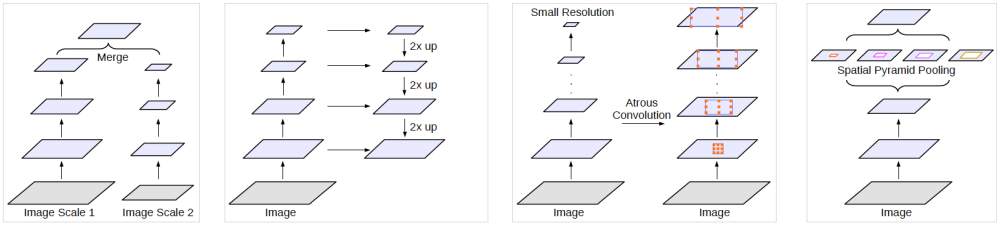


Figure 2. An overview of the whole pipeline in (A). The details of Global Convolutional Network (GCN) and Boundary Refinement (BR) block are illustrated in (B) and (C), respectively.

21.1 DeeplabV3[4]



(a) Image Pyramid (b) Encoder-Decoder (c) Deeper w. Atrous Convolution (d) Spatial Pyramid Pooling

Figure 2. Alternative architectures to capture multi-scale context.

Summary:

- design modules which employ atrous convolution in cascade or in parallel to capture multi-scale context by adopting multiple atrous rates.
- augment our previously proposed Atrous Spatial Pyramid Pooling module, which probes convolutional features at multiple scales, with image-level features encoding global context and further boost performance.
- also elaborate on implementation details and share our experience on training our system.(another contribution.....)
- better multi-grib setting, data augmantation, => 79.35
- pretraining to coco , => 82.7
- effective bootstrapping: In particular, we duplicate the images that contain hard classes (namely bicycle, chair, table, pottedplant, and sofa) in the training set. As shown in Fig. 6, the simple bootstrapping method is effective for segmenting the bicycle class. 82.7=>85.7

final mIoU in PascalVOC 85.7, 0.3 larger than PSPNet(85.4).

21.2 DUC[]

21.3 FRRN[]

21.4 SegFlow: Joint Learning for Video Object Segmentation and Optical Flow(ICCV2017)

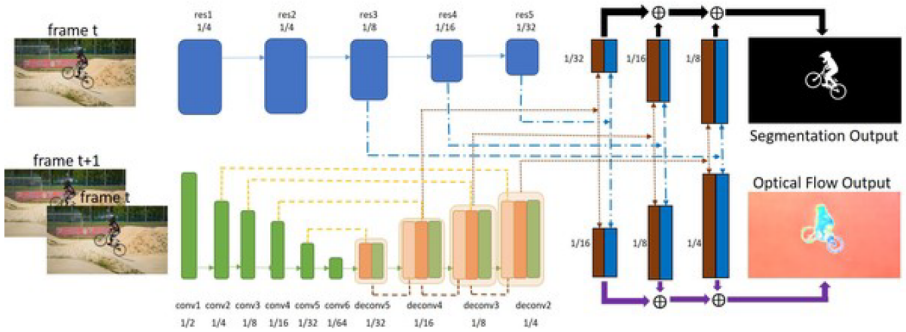


Figure 2. The proposed *SegFlow* architecture. Our model consists of two branches, the segmentation network based on a fully-convolutional ResNet-101 and the flow branch using the FlowNetS [12] structure. In order to construct communications between two branches, we design an architecture that bridges two networks during the up-sampling stage. Specifically, feature maps are propagated bi-directionally through concatenations at different scales with proper operations (i.e., up-sampling or down-sampling) to match the size of different features. Then an iterative training scheme is adopted to jointly optimize the loss functions for both segmentation and optical flow tasks.

- the fusion of segmentation and opticalflow is tricky
- Iterative Offline Training. To start training the joint model, we initialize two branches using the weights from ResNet-101 [18] and FlowNetS [12], respectively. When optimizing the segmentation branch, we freeze the weights of the optical flow branch, and train the network on the DAVIS training set. We use SGD optimizer with batch size 1 for training, starting from learning rate 1e-8 and decreasing it by half for every 10000 iterations. For training the optical flow branch, similarly we fix the segmentation branch and only update the weights in the flow network using the target optical flow dataset (described in Section 5.1). To balance

21.5 One-Shot Learning for Semantic Segmentation

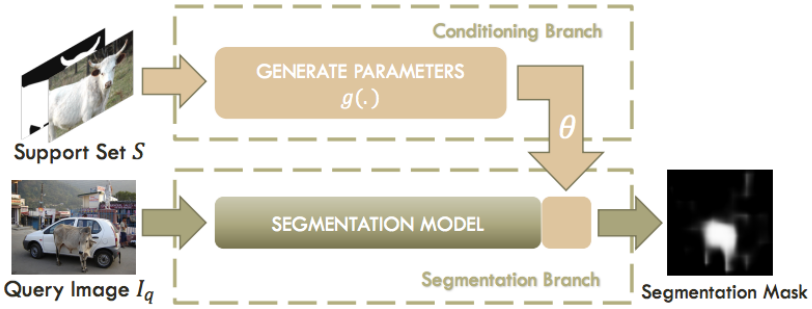


Figure 1: Overview. S is an annotated image from a new semantic class. In our approach, we input S to a function g that outputs a set of parameters θ. We use θ to parameterize part of a learned segmentation model which produces a segmentation mask given I_q.

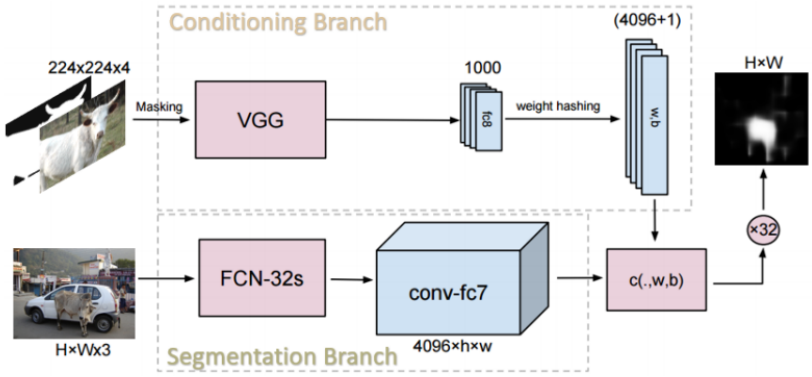


Figure 2: Model Architecture. The conditioning branch receives an image-label pair and produces a set of parameters $\{w, b\}$ for the logistic regression layer $c(\cdot, w, b)$. The segmentation branch is an FCN that receives a query image as input and outputs strided features of conv-fc7. The predicted mask is generated by classifying the pixel-level features through $c(\cdot, w, b)$, which is then upsampled to the original size.

22 Skeleton

22.1 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.

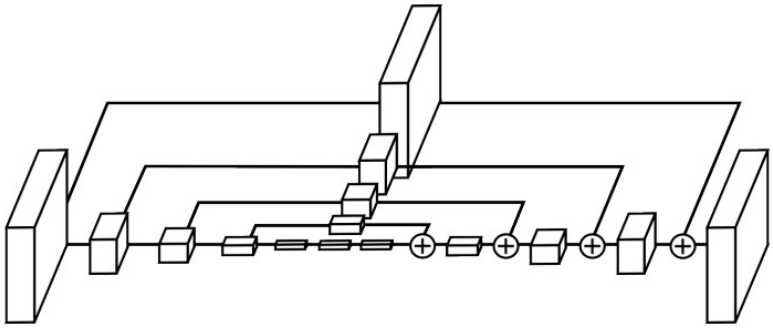


Fig. 3. An illustration of a single "hourglass" module. Each box in the figure corresponds to a residual module as seen in Figure 4. The number of features is consistent across the whole hourglass.

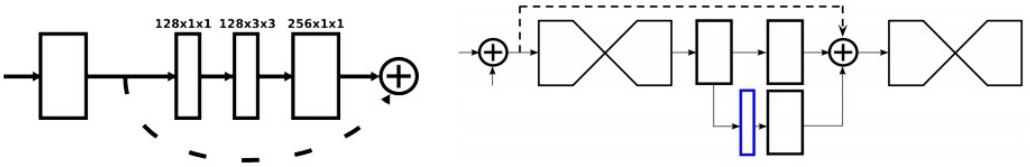


Fig. 4. Left: Residual Module [14] that we use throughout our network. **Right:** Illustration of the intermediate supervision process. The network splits and produces a set of heatmaps (outlined in blue) where a loss can be applied. A 1x1 convolution remaps the heatmaps to match the number of channels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

22.2 Convolutional Pose Machine[26]

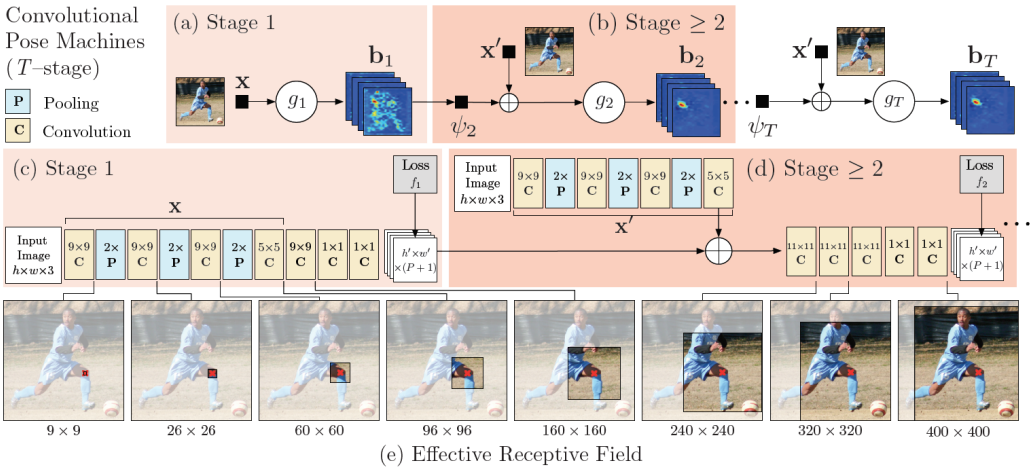


Figure 2: Architecture and receptive fields of CPMs. We show a convolutional architecture and receptive fields across layers for a CPM with any T stages. The pose machine [29] is shown in insets (a) and (b), and the corresponding convolutional networks are shown in insets (c) and (d). Insets (a) and (c) show the architecture that operates only on image evidence in the first stage. Insets (b) and (d) shows the architecture for subsequent stages, which operate both on image evidence as well as belief maps from preceding stages. The architectures in (b) and (d) are repeated for all subsequent stages (2 to T). The network is locally supervised after each stage using an intermediate loss layer that prevents vanishing gradients during training. Below in inset (e) we show the effective receptive field on an image (centered at left knee) of the architecture, where the large receptive field enables the model to capture long-range spatial dependencies such as those between head and knees. (Best viewed in color.)

22.3 Self Adversarial Training for Human Pose Estimation[5]

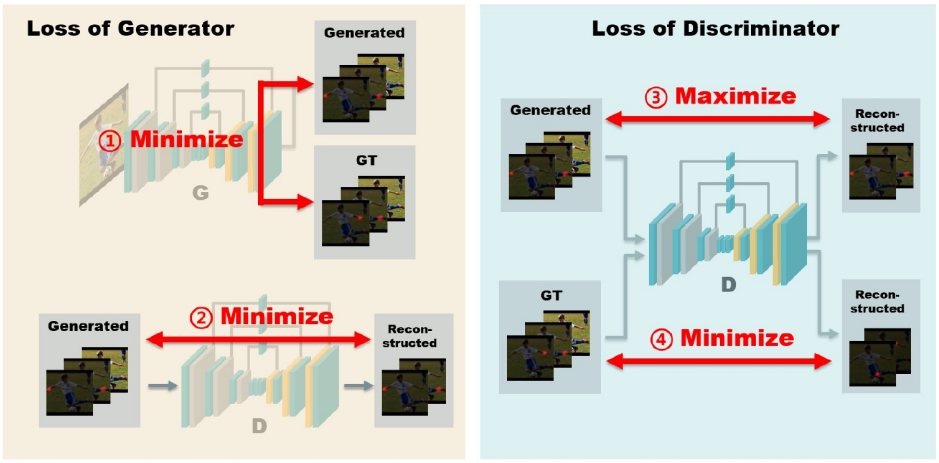


Figure 5. Summary of $\mathcal{L}_G$ and $\mathcal{L}_D$. Losses in the orange box are used to update the generator. Losses in the blue box are used to update the discriminator.

$$
\begin{aligned}
L_{MSE} &= \sum_{i=1}^{N}\sum_{j=1}^{M}(C_{ij} - \tilde{C_{ij}})^2 \\
L_{adv} &= \sum_{i=1}^{M}(\tilde{C_j} - D(\tilde{C_j}, X))^2 \\
L_G &= L_{MSE} + \lambda_G L_{adv} \\
L_{real} &= \sum_{i=1}^{M}(C_j - D(C_j, X))^2 \\
L_{fake} &= \sum_{i=1}^{M}(\tilde{C_j} - D(\tilde{C_j}, x))^2 \\
L_D &= L_{real} - k_t L_{fake} \\
k_{t+1} &= k_t + \lambda_k(rL_{real} - L_{fake})
\end{aligned}
$$

L_{adv} item don't exist in standard BEGAN[2]. but in this paper the author add it,detailed reason in [5] section 3.3

22.4 Recurrent Human Pose Estimation[1]

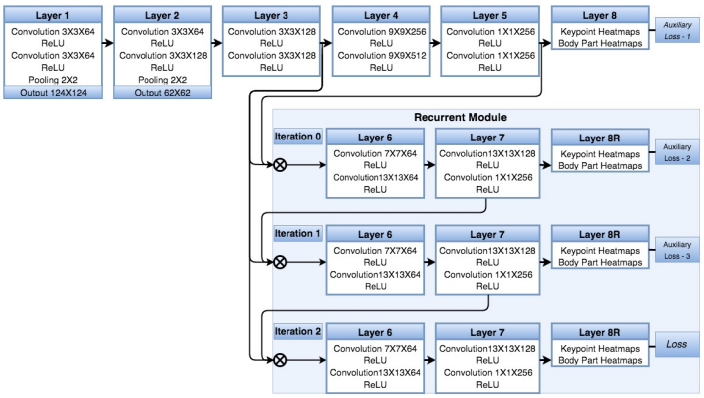


Fig. 3: **ConvNet with Recurrent Module**: Our network is composed of 7 layers. The recurrent model is introduced for Layer 6 and 7. In the current example, a network with 2 iterations is visualized. Note that all loss functions are auxiliary for facilitating the optimization and the final outcome comes from Layer 8D. Moreover, the body part heatmaps is an auxiliary task for additional data augmentation. In our graph, the symbol $\otimes$ corresponds to the concatenation operation.

- not only keypoint heatmap, but also body part heatmap(we define the body part heatmap by taking the midpoint between the two keypoint as the center of the Gaussian distribution and define the variance based on the Euclidean distance between the two keypoints.), both of them are auxiliary loss and final loss!
- We do not aim to predict body parts, but observe that this additional objective is helpful for capturing additional body constraints, propagate back more gradients and thus it brings a boost to the model's performance. **PAF in openpose make this character to a extreme pink.**
- small kernel at the beginning, large kernel in the recurrent module.
- even though ROC less than hourglass, the model size is much smaller than hourglass.

22.5 Multi-Context Attention for Human Pose Estimation[6]

22.6 Learning Feature Pyramids for Human Pose Estimation

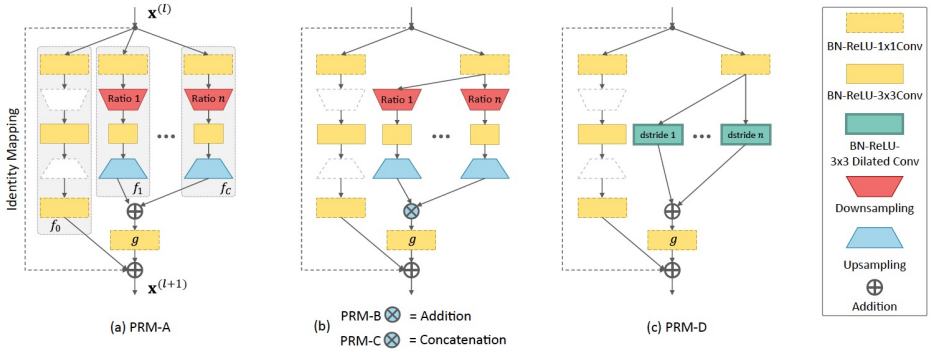


Figure 3. Structures of PRMs. Dashed links indicate identity mapping. (a) PRM-A produces separate input feature maps for different levels of pyramids, while (b) PRM-B uses shared input for all levels of pyramids. PRM-C use concatenation instead of addition to combine features generated from pyramids, which is similar to inception models. (c) PRM-D use dilated convolutions, which are also used in ASPP-net [9], instead of pooling to build the pyramid. The dashed trapezoids mean that the subsampling and upsampling are skipped.

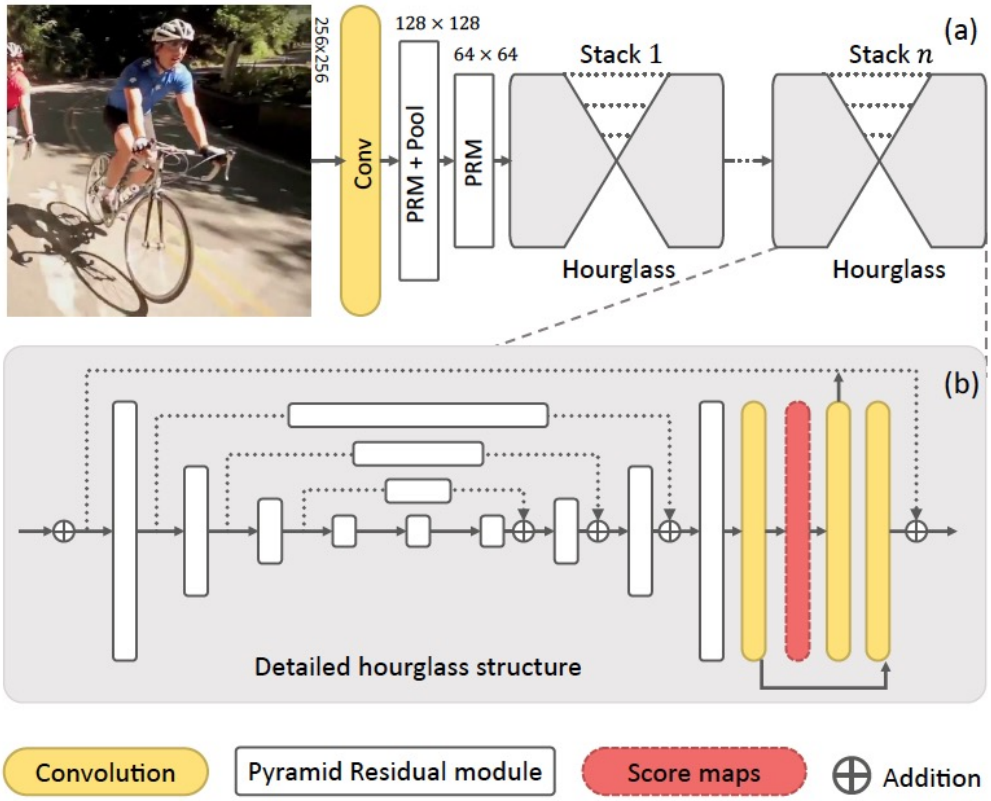


Figure 2. Overview of our framework. (a) demonstrates the network architecture, which has n stacks of hourglass network. Details of each stack of hourglass is illustrated in (b). Score maps of body joint locations are produced at the end of each hourglass, and a squared-error loss is also attached in each stack of hourglass.

Summary

- Pyramid Residual Modules (PRMs)
- A weight initialization scheme for multi-branch is provided
- network goes deeper, the variance will increase drastically. In this paper, we use a 1*1 convolution preceding with batch normalization and ReLU to replace the identity mapping when the output of two residual units are summed up, as illustrated in Figure. 6(b). This simple replacement stops the variance explosion, as demonstrated in Figure. 6(c). In experiments, we find that breaking the variance explosion also provide a better performance
- don't know how much gain the weight initialization and variation accumulation stuff can bring to.

22.7 Look Into Person

train joints and parsing, joint loss is L2 loss, parsing loss is cross-entropy loss, the final loss is the product of joint loss and parsing loss.

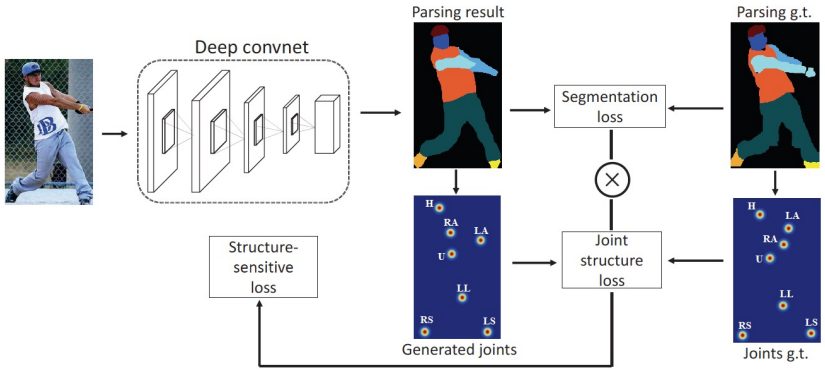


Figure 6: Illustration of our Self-supervised Structure-sensitive Learning for human parsing. An input image goes through parsing networks including several convolutional layers to generate the parsing results. The generated joints and joints ground truth represented as heatmaps are obtained by computing the center points of corresponding regions in parsing maps, including head (H), upper body (U), lower body (L), right arm (RA), left arm (LA), right leg (RL), left leg (LL), right shoe (RS), left shoe (LS). The structure-sensitive loss is generated by weighting segmentation loss with joint structure loss. For clear observation, here we combine nine heatmaps into one map.

22.8 Recurrent 3D Pose Sequence Machines

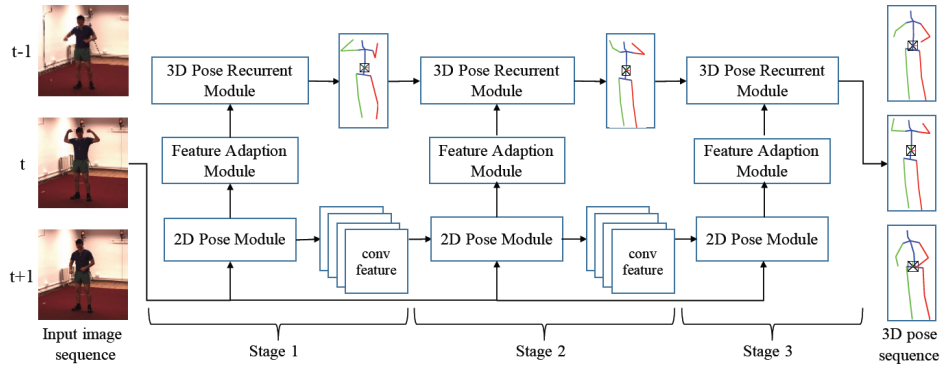


Figure 2: An overview of the proposed Recurrent 3D Pose Sequence Machine architecture. Our framework predicts the 3D human poses for all of the monocular image frames, and then sequentially refines them with multi-stage recurrent learning. At each stage, every frame of the input sequence is sequentially passed into three neural network modules: i) a 2D pose module extracting the image-dependent pose representations; 2) a feature adaption module for transforming the pose representations from 2D to 3D domain; 3) a 3D pose recurrent module predicting the human joints in 3D coordinates. Note that, the parameters of 3D pose recurrent module for all frames are shared to preserve the temporal motion coherence. Given the initial predicted 3D joints and 2D features from the first stage, we perform the multi-stage refinement to recurrently improve the pose accuracy. From the second stage, the previously predicted 17 joints (51 dimensions) and the 2D pose-aware features are also posed as the input of 2D pose module and 3D pose recurrent module, respectively. The final 3D pose sequence results are obtained after recurrently performing the multi-stage refinement.

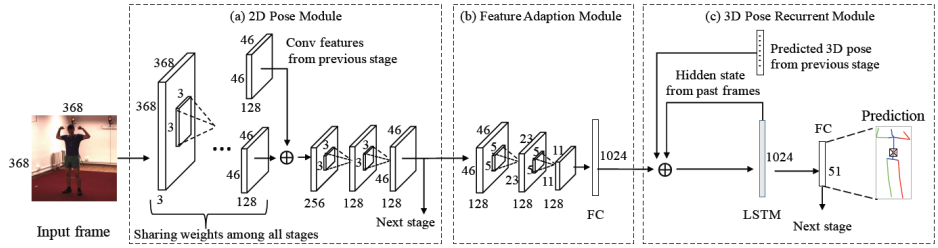


Figure 3: Detailed network architecture of our proposed RPSM at the k-th stage. An input frame with the 368×368 size is subsequently fed into 2D pose module, feature adaption module and 3D pose recurrent module to predict the locations of 17 joint points (51 dimensions output). The 2D pose module consists of 15 shared convolution layers across all stages and 2 specialized convolution layers for each stage. The specialized convolution layers take the shared features and the 2D pose-aware features at previous stage as the input, and output specialized features to the feature adaption module as well as the next stage. The feature adaption module consists of two convolution layers and one fully-connected layer with 1024 units. Finally, the adapted features, the hidden states of the LSTM layer and previously predicted 3D poses are concatenated together as the input of 3D pose recurrent module to produce the 3D pose of each frame. The symbol $\oplus$ means the concatenation operation.

22.9 Coarse-to-Fine Volumetric Prediction for Single-Image 3D Human Pose

- have implementation code
- hourglass-like structure
- To deal with this curse of dimensionality, we propose to use a coarse-to-fine prediction scheme. In particular, the first steps are supervised with lower resolution targets for the (most challenging and technically unobserved) z-dimension. Precisely, we use targets of size 64*64 d per joint, where d typically takes values from the set f1; 2; 4; 8; 16; 32; 64g. An illustration of this supervision approach is given in Figure 2.
- Volume Regression(fully convolution in the end) is better than Coordinate Regression(fully connected in the end).

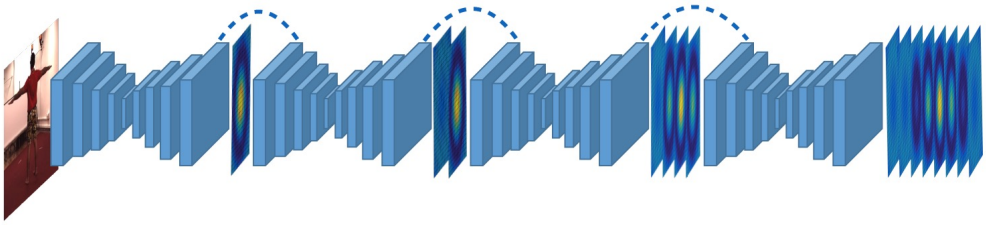


Figure 2: Illustration of our coarse-to-fine volumetric approach for 3D human pose estimation from a single image. The input is a single color image and the output is a dense 3D volume with separate per voxel likelihoods for each joint. The network consists of multiple fully convolutional components [21], which are supervised in a coarse-to-fine fashion, to deal with the large dimensionality of our representation. 3D heatmaps are synthesized for supervision by increasing the resolution for the most challenging z-dimension (depth) after each component. The dashed lines indicate that the intermediate heatmaps are fused with image features to produce the input for the next fully convolutional component. For presentation simplicity, the illustrated heatmaps correspond to the location of only one joint.

22.10 LCR-Net: Localization-Classification-Regression for Human Pose

too complicated

22.11 3D Human Pose Estimation from a Single Image via Distance Matrix Regression

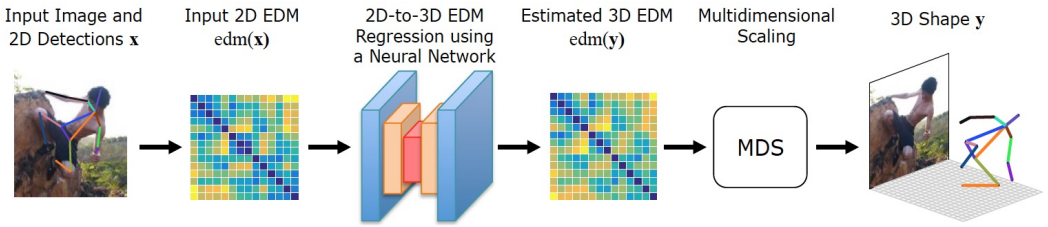


Figure 1. **Overview.** We formulate the 3D human pose estimation problem as a regression between two Euclidean Distance Matrices (EDMs), encoding pairwise distances of 2D and 3D body joints, respectively. The regression is carried out by a Neural Network, and the 3D joint estimates are obtained from the predicted 3D EDM via Multidimensional Scaling.

- matrix formulation
- EDM can be regressed by fully connected network or fully convolutional network(append a matrix symmetrization: $z = (z + z^T)/2$)
- multidimensional scaling is tricky.
- Human3.6 result is 73.98
- the train 2D coordinate is Ground Truth (GT), CPM or GT+CPM. the test 2D coordinate is always CPM.
- the "Figure 2. EDMs vs Cartesian representations." and "Robustness to Occlusions.", "Robustness to 2D Noise." are stories.

22.12 Lifting from the Deep: Convolutional 3D Pose Estimation from a Single Image

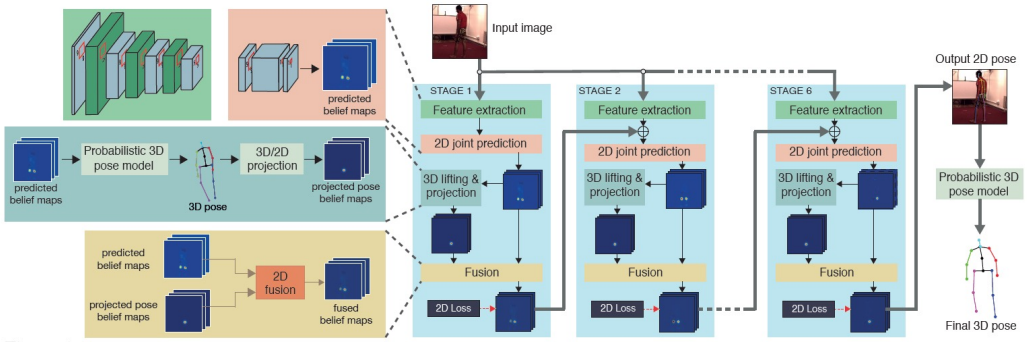


Figure 1: The multistage deep architecture for 2D/3D human pose estimation. Each stage produces as output a set of belief maps for the location of the 2D landmarks (one per landmark). The belief maps from each stage, as well as the image, are used as input to the next stage. Internally, each stage learns to combine: *(a)* belief maps provided by convolutional 2D joint predictors, with *(b)* projected pose belief maps, proposed by the probabilistic 3D pose model. The 3D pose layer is responsible for lifting 2D landmark coordinates into 3D and projecting them onto the space of valid 3D poses. These two belief maps are then fused into a single set of output proposals for the 2D landmark locations per stage. The accuracy of the 2D and 3D landmark locations increases progressively through the stages. The loss used at each stage requires only 2D pose annotations, not 3D. The overall architecture is fully differentiable – including the new projected-pose belief maps and 2D-fusion layers – and can be trained end-to-end using back-propagation. [Best viewed in color.]

- Human3.6M: 88.39
- 3D pose inference from a single RGB image makes use of a multistage deep convolutional architecture, trained end-to-end, that repeatedly fuses and refines 2D and 3D poses, and a second module which takes the final predicted 2D landmarks and lifts them one last time into 3D space for the final estimate

22.13 3D Human Pose Estimation = 2D Pose Estimation + Matching

confusing

22.14 A simple yet effective baseline for 3d human pose estimation(ICCV2017)

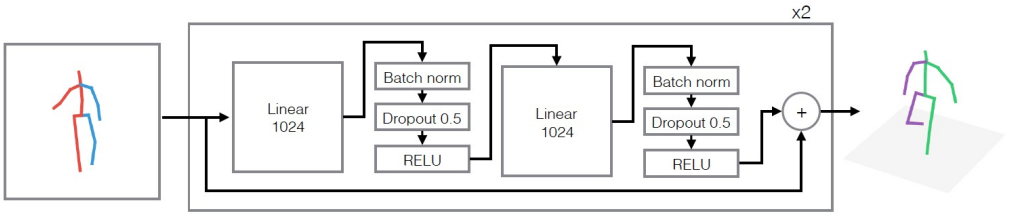


Figure 1. A diagram of our approach. The building block of our network is a linear layer, followed by batch normalization, dropout and a RELU activation. This is repeated twice, and the two blocks are wrapped in a residual connection. The outer block is repeated twice. The input to our system is an array of 2d joint positions, and the output is a series of joint positions in 3d.

- Human3.6M Ground Truth Accuracy: 45.5
- about Human 3.6M protocol: On Human3.6M we follow the standard protocol, using subjects 1, 5, 6, 7, and 8 for training, and subjects 9 and 11 for evaluation. We report the average error in millimetres between the ground truth and our prediction across all joints, after alignment of the root (central hip) joint. Typically, training and testing is carried out independently in each action. We refer to this as protocol #1. However, in some of our baselines, the prediction has been further aligned with the ground truth via a rigid transformation (e.g. [7,25]). We call this post-processing protocol #2. Similarly, some recent methods have trained one model for all the actions, as opposed to building action-specific models. We have found that this practice consistently improves results, so we report results for our method under these two variations.

23 Human Parsing

23.1 Human Parsing With Contextualized Convolutional Neural Network

ICCV 2015

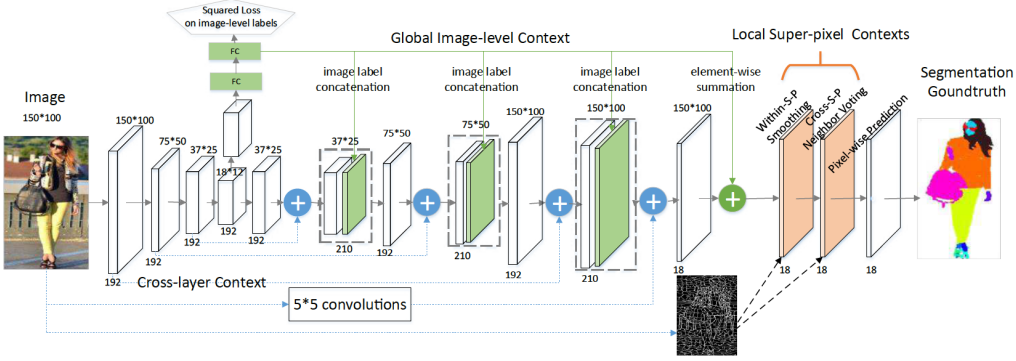


Figure 1. Our Co-CNN integrates the cross-layer context, global image-level context and local super-pixel contexts into a unified network. It consists of cross-layer combination, global image-level label prediction, within-super-pixel smoothing and cross-super-pixel neighborhood voting. First, given an input 150×100 image, we extract the feature maps for four resolutions (i.e., 150×100, 75×50, 37×25 and 18×12). Then we gradually up-sample the feature maps and combine the corresponding early, fine layers (blue dash line) and deep, coarse layers (blue circle with plus) under the same resolutions to capture the cross-layer context. Second, an auxiliary objective (shown as "Squared loss on image-level labels") is appended after the down-sampling stream to predict global image-level labels. These predicted probabilities are then aggregated into the subsequent layers after the up-sampling (green line) and used to re-weight pixel-wise prediction (green circle with plus). Finally, the within-super-pixel smoothing and cross-super-pixel neighborhood voting are performed based on the predicted confidence maps (orange planes) and the generated super-pixel over-segmentation map to produce the final parsing result. Only down-sampling, up-sampling, and prediction layers are shown; intermediate convolution layers are omitted. For better viewing of all figures in this paper, please see original zoomed-in color pdf file.

24 Graph Convolution Network

For these models, the goal is then to learn a function of signals/features on a graph G=(V,E) which takes as input:

- A feature description x_i for every node i; summarized in a N*D feature matrix X (N: number of nodes, D: number of input features)

- A representative description of the graph structure in matrix form; typically in the form of an adjacency matrix A (or some function thereof)

Every neural network layer can then be written as a non-linear function:

$$H^{(l+1)} = f(H^l, A)$$

generally,

$$f(H^l, A) = \sigma(AH^lW^l)$$

in ICLR 2017 paper [16] :

$$f(H^l, A) = \sigma(\tilde{D}^{-\frac{1}{2}}\tilde{A}\tilde{D}^{-\frac{1}{2}}H^lW^l)$$

with $\tilde{A} = A + I$,where I is the identity matrix and $\tilde{D}$ is the diagonal node degree matrix of $\tilde{A}$

24.1 Convolutional Neural Networks on Graphs with Fast Localized Spectral Filtering[8]

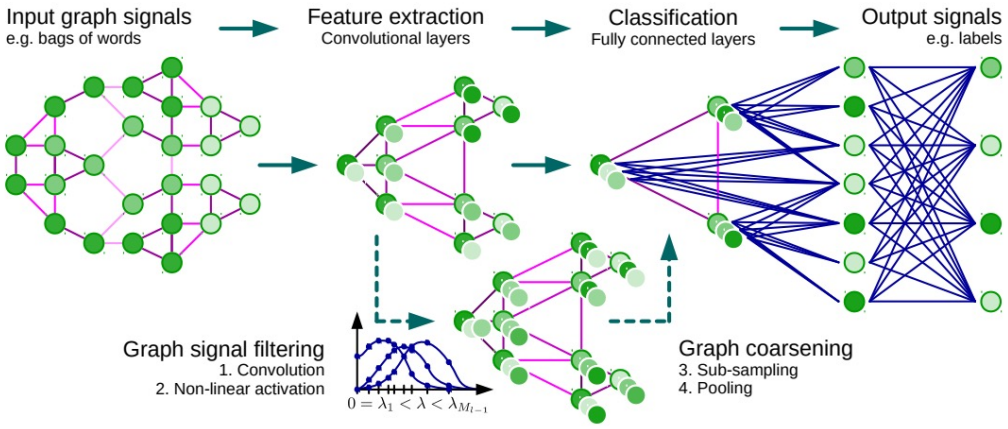


Figure 1: Architecture of a CNN on graphs and the four ingredients of a (graph) convolutional layer.

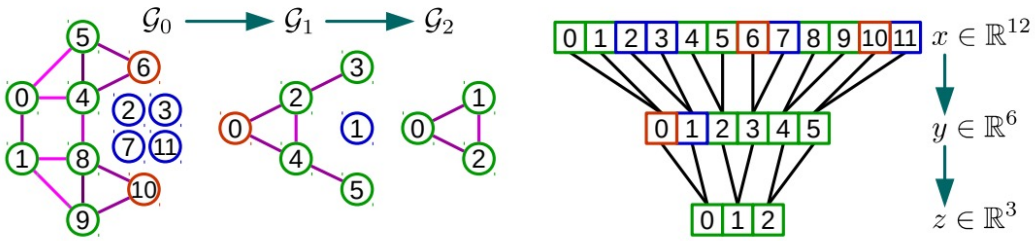


Figure 2: **Example of Graph Coarsening and Pooling.** Let us carry out a max pooling of size 4 (or two poolings of size 2) on a signal $x \in \mathbb{R}^8$ living on $\mathcal{G}_0$, the finest graph given as input. Note that it originally possesses $n_0 = |\mathcal{V}_0| = 8$ vertices, arbitrarily ordered. For a pooling of size 4, two coarsenings of size 2 are needed: let Graclus gives $\mathcal{G}_1$ of size $n_1 = |\mathcal{V}_1| = 5$, then $\mathcal{G}_2$ of size $n_2 = |\mathcal{V}_2| = 3$, the coarsest graph. Sizes are thus set to $n_2 = 3$, $n_1 = 6$, $n_0 = 12$ and fake nodes (in blue) are added to $\mathcal{V}_1$ (1 node) and $\mathcal{V}_0$ (4 nodes) to pair with the singeltons (in orange), such that each node has exactly two children. Nodes in $\mathcal{V}_2$ are then arbitrarily ordered and nodes in $\mathcal{V}_1$ and $\mathcal{V}_0$ are ordered consequently. At that point the arrangement of vertices in $\mathcal{V}_0$ permits a regular 1D pooling on $x \in \mathbb{R}^{12}$ such that $z = [\max(x_0, x_1), \max(x_4, x_5, x_6), \max(x_8, x_9, x_{10})] \in \mathbb{R}^3$, where the signal components x_2, x_3, x_7, x_{11} are set to a neutral value.

24.2 SEMI-SUPERVISED CLASSIFICATION WITH GRAPH CONVOLUTIONAL NETWORKS[16]

25 Attention Mechanism

26 Traditional CV method

26.1 HOG

26.2 SIFT

27 DPM

28 MISC

28.1 Spatial Transformer Networks

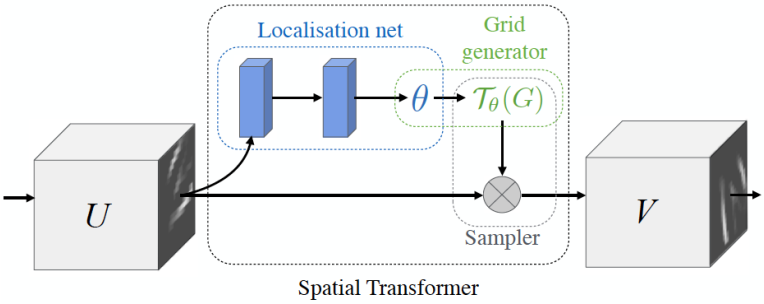


Figure 2: The architecture of a spatial transformer module. The input feature map U is passed to a localisation network which regresses the transformation parameters θ. The regular spatial grid G over V is transformed to the sampling grid $\mathcal{T}_\theta(G)$, which is applied to U as described in Sect. 3.3, producing the warped output feature map V. The combination of the localisation network and sampling mechanism defines a spatial transformer.

- localization, regular spatial grid,

- localization net must be large enough to capture the context info.
- the initialization of Spatial Transformer Layer is important, (at the begining, must satisfy that the input and output is the same)

29 Bayesian

H: Hypothesis D: Data

prior: P(H) likelihood: P(D|H) posterior: P(H|D)

$P(H|D) = \sum P(H)P(D|H)$

- frequentist(an optimization problem, maximum likelihood estimation)
- Bayesian
- they are equal,from the perspective of max posterior probability:

$$\begin{aligned} argmin_{\theta \in \mathcal{D}} \quad f(\theta) = \\ argmin_{\theta \in \mathcal{D}} \quad -\sum_{i=1}^{n} p(x_i|\theta) = \\ argmin_{\theta} \quad f(\theta) + \lambda Q(\theta) = \end{aligned} \tag{1}$$

add exponent function,we can get:

$$argmax_{\theta} \quad exp(-f(\theta))exp(-\lambda Q(\theta)) \tag{2}$$

$exp(-f(\theta))$ is the likelihood item, $exp(-\lambda Q(\theta))$ is the prior item.

29.1 Distribution Introduction

29.1.1 Gaussian Distribution

29.1.2 Laplace Distribution

a random variable has a Laplace(μ, b) distribution if its probability density function is

$$f(x|\mu, b) = \frac{1}{2b} exp(-\frac{|x-\mu|}{b}) \tag{3}$$

when $\mu = 0$, we call it Doubly exponential distribution. (more smooth)

29.2 Student's t-Distribution

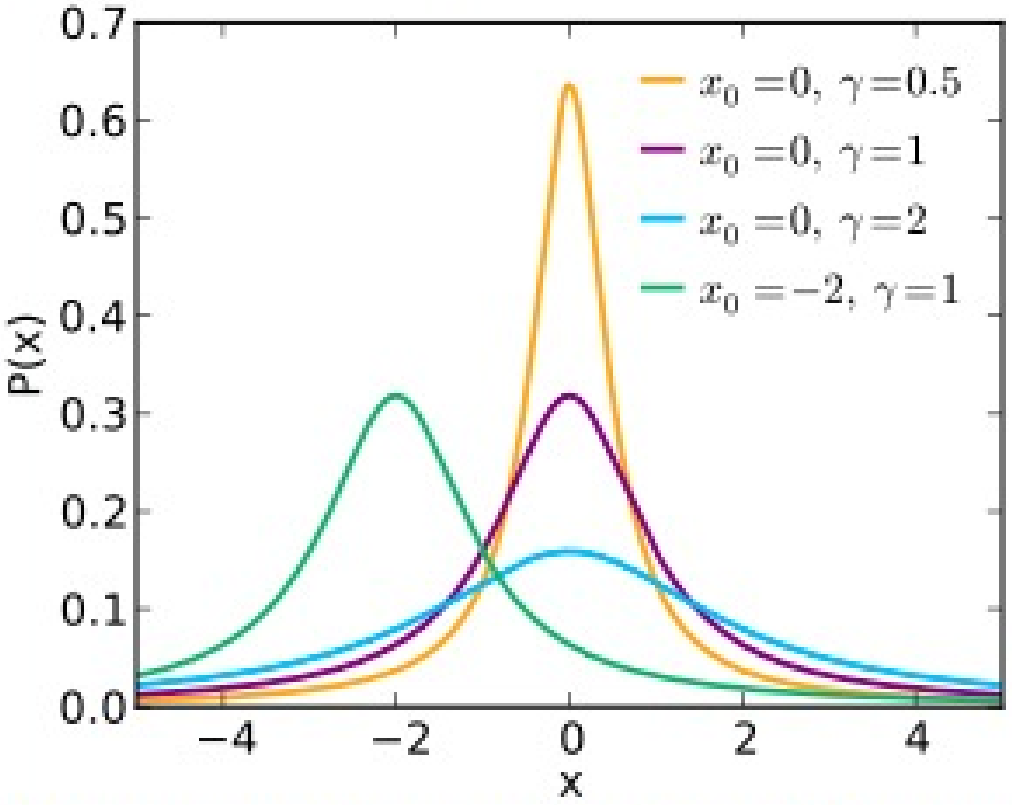
when mu-> +8, it's a Gaussian distribution.

29.3 Cauchy Distribution

$$f(x; x_0, \gamma) = \frac{1}{\pi\gamma[1+(\frac{x-x_0}{\gamma})^2]}$$

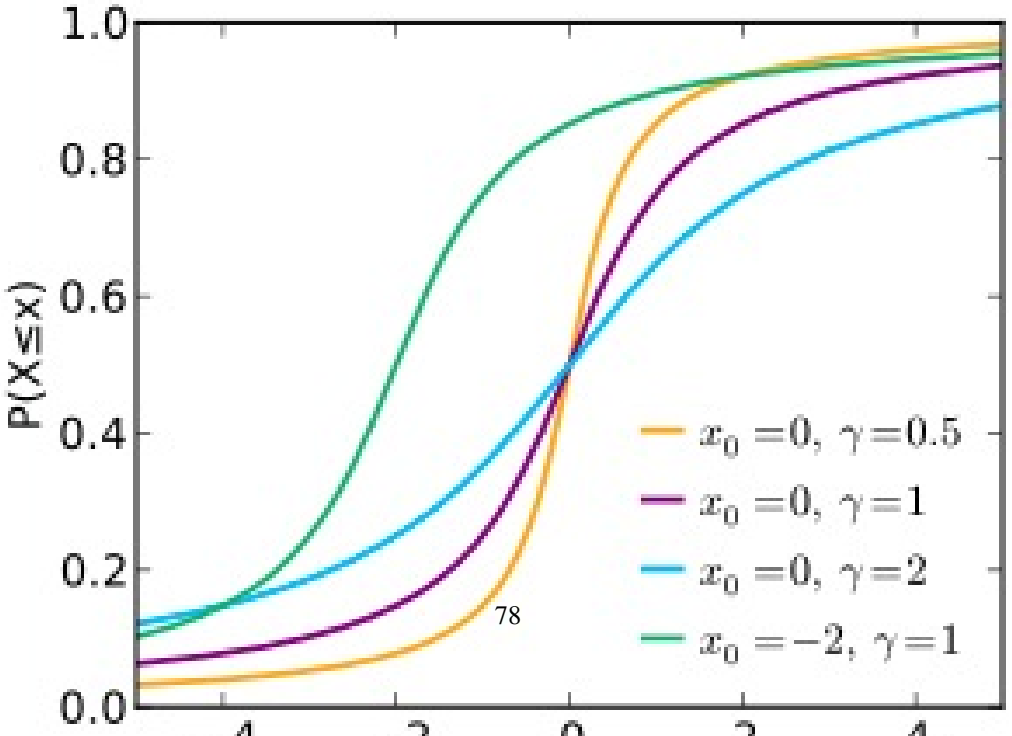
Cauchy

Probability density function



The purple curve is the standard Cauchy distribution

Cumulative distribution function



Stirling's approximation

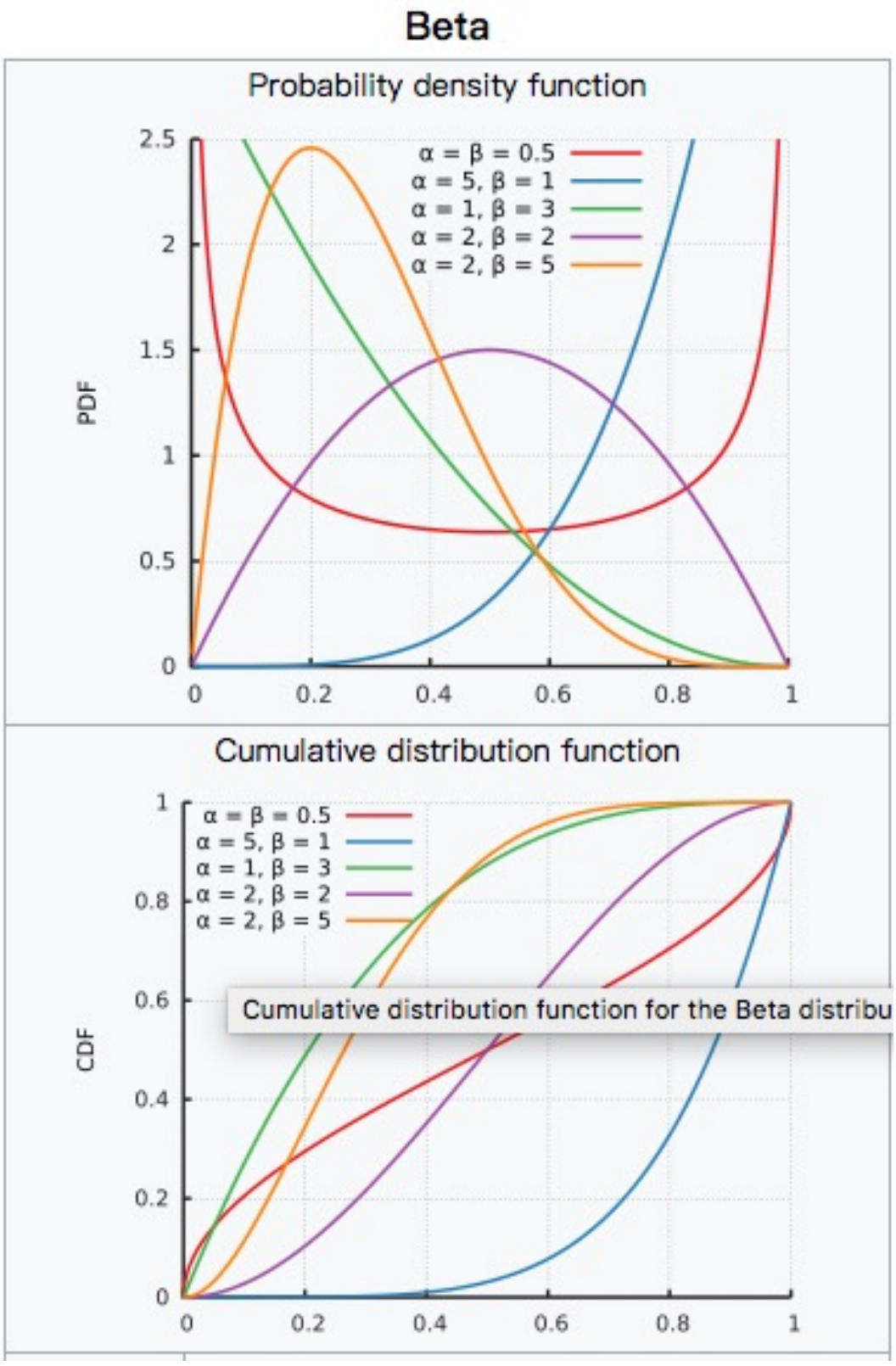
In mathematics, Stirling's approximation (or Stirling's formula) is an approximation for factorials. It is a good-quality approximation, leading to accurate results even for small values of n.

$$lnn! = nlnn - n + O(lnn)$$

29.4 Beta Distribution

$f(x;\alpha,\beta) = \frac{1}{B(\alpha,\beta)} x^{\alpha-1}(1-x)^{\beta-1}$

where B is a beta function,whose value can be checked in table.


Beta
Probability density function
PDF
α = β = 0.5
α = 5, β = 1
α = 1, β = 3
α = 2, β = 2
α = 2, β = 5
2.5
2
1.5
1
0.5
0
0
0.2
0.4
0.6
0.8
1
Cumulative distribution function
CDF
α = β = 0.5
α = 5, β = 1
α = 1, β = 3
α = 2, β = 2
α = 2, β = 5
Cumulative distribution function for the Beta distribu
1
0.8
0.6
0.4
0.2
0
0
0.2
0.4
0.6
0.8
1

29.5 Generalized Inverse Gaussian Distribution(GIG)

29.6 Bernoulli Distribution

Pr(X=1) = p; Pr(X=0) = 1-p;

$p(x|a) = p^x(1-p)^{1-x}$

29.7 Possion Distribution

$f(x) = e^{-\lambda}\frac{\lambda^x}{x!}, x = 0, 1, 2, ...$

if x1,...,xn $\sim Bernoulli(P)$,then $\sum_{i=1}^{n} \sim Possion(P)$

topic model!

29.8 The Negative Binomial Distribution

f(k;r,p)= when r=1, equals geometric distribution.

29.9 Gaussian Process and Inverse Gaussian Process

"equal in distribution" != "equal"

mode, median,CDF, PMF

30 Graphic Models

- end-to-end?
- what parameter to optimize?
- how to inference?
- how many V and E? is sparse or dense graph?
- the unary potential is always fomulated into probability ratio in the log space: $log(\frac{1-P}{P})$

30.0.1 CRFasRNN

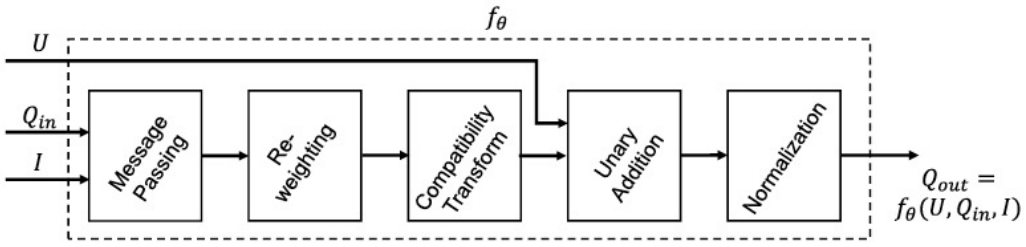


Figure 1. **A mean-field iteration as a CNN.** A single iteration of the mean-field algorithm can be modelled as a stack of common CNN layers.

Algorithm 1 Mean-field in dense CRFs [29], broken down to common CNN operations.

$Q_i(l) \leftarrow \frac{1}{Z_i} \exp\left(U_i(l)\right)$ for all i ▷ Initialization

while not converged **do**

$\tilde{Q}_i^{(m)}(l) \leftarrow \sum_{j \neq i} k^{(m)}(\mathbf{f}_i, \mathbf{f}_j) Q_j(l)$ for all m ▷ Message Passing

$\breve{Q}_i(l) \leftarrow \sum_m w^{(m)} \tilde{Q}_i^{(m)}(l)$ ▷ Weighting Filter Outputs

$\hat{Q}_i(l) \leftarrow \sum_{l' \in \mathcal{L}} \mu(l, l') \breve{Q}_i(l')$ ▷ Compatibility Transform

$\breve{Q}_i(l) \leftarrow U_i(l) - \hat{Q}_i(l)$ ▷ Adding Unary Potentials

$Q_i \leftarrow \frac{1}{Z_i} \exp\left(\breve{Q}_i(l)\right)$ ▷ Normalizing

end while

- Message Passing can be simplified by some trick.

- Message Passing break the Q into m part; Weighting Filter Outputs union the m parts together.

- Weighting Filter Outputs is 1*1*m*1 kernel; Compatibility Transform is 1*1*L*L kernel.

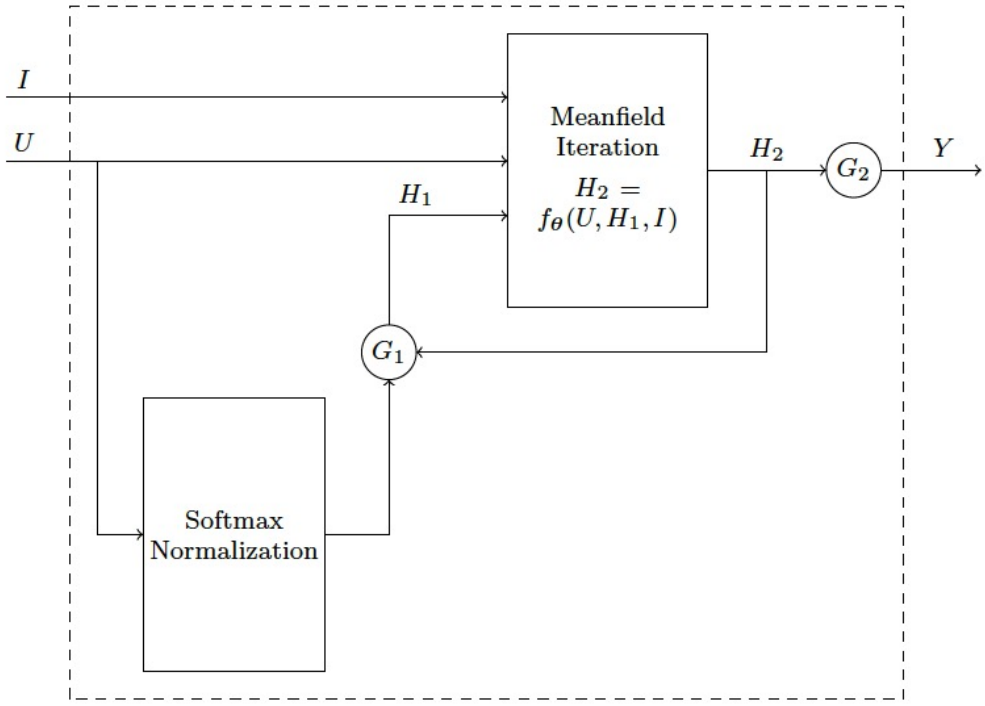


Figure 2. **The CRF-RNN Network.** We formulate the iterative mean-field algorithm as a Recurrent Neural Network (RNN). Gating functions G_1 and G_2 are fixed as described in the text.

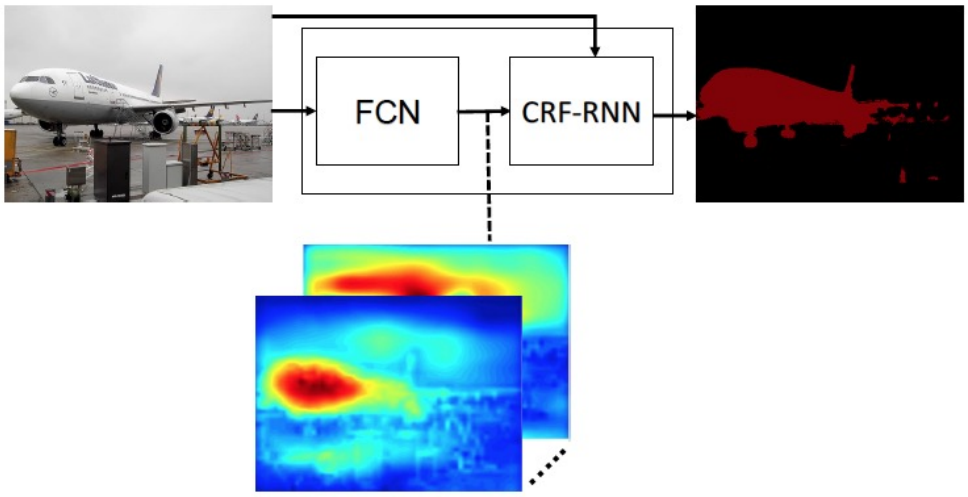


Figure 3. **The End-to-end Trainable Network.** Schematic visualization of our full network which consists of a CNN and the CNN-CRF network. Best viewed in colour.

30.1 Efficient Inference in Fully Connected CRFs with Gaussian Edge Potentials[17]

reference ppt:http://swoh.web.engr.illinois.edu/courses/IE598/handout/fall2016_slide15.pdf

after the CRF, the Q is finally determined, we can use it to do segmentation

Minimizing the KL-divergence, while constraining $Q(\mathbf{X})$ and $Q_i(X_i)$ to be valid distributions, yields the following iterative update equation:

$$Q_i(x_i = l) = \frac{1}{Z_i} \exp\left\{ -\psi_u(x_i) - \sum_{l' \in \mathcal{L}} \mu(l, l') \sum_{m=1}^{K} w^{(m)} \sum_{j \neq i} k^{(m)}(\mathbf{f}_i, \mathbf{f}_j) Q_j(l') \right\}. \quad (4)$$

A detailed derivation of Equation 4 is given in the supplementary material. This update equation leads to the following inference algorithm:

Algorithm 1 Mean field in fully connected CRFs

Initialize Q ▷ $Q_i(x_i) \leftarrow \frac{1}{Z_i} \exp\{-\phi_u(x_i)\}$
while not converged **do** ▷ See Section 6 for convergence analysis
  $\tilde{Q}_i^{(m)}(l) \leftarrow \sum_{j \neq i} k^{(m)}(\mathbf{f}_i, \mathbf{f}_j) Q_j(l)$ for all m ▷ **Message passing** from all X_j to all X_i
  $\hat{Q}_i(x_i) \leftarrow \sum_{l \in \mathcal{L}} \mu^{(m)}(x_i, l) \sum_m w^{(m)} \tilde{Q}_i^{(m)}(l)$ ▷ **Compatibility transform**
  $Q_i(x_i) \leftarrow \exp\{-\psi_u(x_i) - \hat{Q}_i(x_i)\}$ ▷ **Local update**
  normalize $Q_i(x_i)$
end while

for segmentation task, the pairwise potential is:

For multi-class image segmentation and labeling we use contrast-sensitive two-kernel potentials, defined in terms of the color vectors I_i and I_j and positions p_i and p_j:

$$k(\mathbf{f}_i, \mathbf{f}_j) = w^{(1)} \underbrace{\exp\left(-\frac{|p_i - p_j|^2}{2\theta_\alpha^2} - \frac{|I_i - I_j|^2}{2\theta_\beta^2} \right)}_{\text{appearance kernel}} + w^{(2)} \underbrace{\exp\left(-\frac{|p_i - p_j|^2}{2\theta_\gamma^2} \right)}_{\text{smoothness kernel}}. \quad (3)$$

The *appearance kernel* is inspired by the observation that nearby pixels with similar color are likely to be in the same class. The degrees of nearness and similarity are controlled by parameters θ_α and θ_β. The *smoothness kernel* removes small isolated regions [19]. The parameters are learned from data, as described in Section 4.

30.2 Holistic, Instance-level Human Parsing[19]

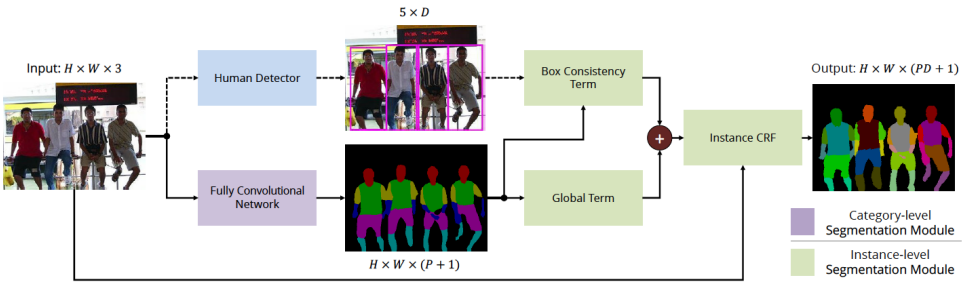


Figure 2: Our proposed approach. An $H \times W \times 3$ image is input to a human detection network and a body parts semantic segmentation network, producing D detections of human and an $H \times W \times (P+1)$ dimensional feature map respectively, where $(P+1)$ is the size of the semantic label space including a background class. These results are used to form the unary potentials of an Instance CRF which performs instance segmentation by associating labelled pixels with human detections. In the above diagram, dotted lines represent forward only paths, and solid lines show routes where both features and gradients flow. The green boxes form the instance-level segmentation module (Sec. 3.2). Best viewed in colour.

Insight:

- Box Consistency Term: make the pixel potential in detection boundary non-zero; outside of the boundary zero.

- Global Term: A possible shortcoming for the box consistency potential is that if some pixels belonging to a human instance fall outside the bounding box and are consequently assigned 0 for the box consistency term potential, they would be lost in the final instance segmentation prediction.

- Pairwise Term is the same as denseCRF[17], which takes the light, position into consideration.

- a tricky permutation-invariant loss function is designed, which is backpropagated through both our instance- and categorylevel segmentation networks. the segment match can be formed by the Hungarian algorithm as it can be formulated as a bipartite graph matching problem, then the cross-entropy loss can be applied into it.

- like CRFasRNN, the parameter of unary potential and pairwise potential(weighting coefficients) are learned via back-propagation.

We formulate a Conditional Random Field over these V variables, where the energy of the assignment $\mathbf{v}$ to all of the instance variables $\mathbf{V}$ consists of two unary terms, and one pairwise term (whose weighting co-efficients are all learned via backpropagation):

$$E(\mathbf{V}=\mathbf{v}) = -\sum_{i}^{N} \ln\left[w_1 \psi_{Box}(v_i) + w_2 \psi_{Global}(v_i) + \varepsilon\right] + \sum_{i<j}^{N} \psi_{Pairwise}(v_i, v_j). \quad (1)$$

The unary and pairwise potentials are computed within our neural network, differentiable with respect to their input and parameters, and described in Sec. 3.2.1 through 3.2.3. The Maximum-a-Posteriori (MAP) estimate of our CRF (since the energy in Eq. 1 characterises a Gibbs distribution) is computed as the final labelling produced by our network. We perform the iterative mean-field inference algorithm to approximately compute the MAP solution by minimising Eq. 1. As shown by Zheng *et al.* [51], this can be formulated as a Recurrent Neural Network (RNN), allowing it to be trained end-to-end as part of a larger network. However, as our network is input a variable number of detections per image, D, the label space of the CRF is dynamic. Therefore, unlike [51], the parameters of our CRF are not class-specific to allow for this variable number of "channels".

30.3 Joint Multi-Person Pose Estimation and Semantic Part mentation[28]

Insight:

- there exists three types of map: pixel-wise joint score map, pixel-wise joint neighbour score map, part score map.
- joint score map forms the unary potential; joint neighbour score map part score map together form the pairwise potential
- the pairwise potential formulation is very tricky.
  $\phi_{i,j} = l_{c_i,c_j} log \frac{1-P_{i,j}(l_{c_i},l_{c_j}|P_n,P_s)}{P_{i,j}(l_{c_i},l_{c_j}|P_n,P_s)}$
  where $P_{i,j}(l_{c_i}, l_{c_j}) = \frac{1}{exp(-wf(c_i,c_j,l_{c_i},l_{c_j}))}$ obtained from the logistic regression results w.r.t a combined feature vector f from f(Pn) and f(Ps). the formulation of f(Pn) and f(Ps) is tricky.
- The feature vector f (Pn) encodes information to help decide whether the two proposals belong to the same person. We borrow the idea proposed in deepercut; The feature vector f (Ps) considers the correlation between joints and segments. Intuitively, joints are the connection points of parts.
- parameters to optimize: pairwise potential logistic regression weight: w;label $\mathcal{L} = \{l_{c_i} | c_i \in \mathcal{V} \cup \{l_{c_i,c_j} | (c_i, c_j) \in \varepsilon\}\}$. the parameters are optimized by ILP.
- the whole procedure can be divided into fours steps: 1). Faster RNN detector to produce human detection boxes. 2). joint proposal. 3). FCRF ILP optimization. 4) combine the previous result to generate semantic part segmentation.

unary potential formulation:

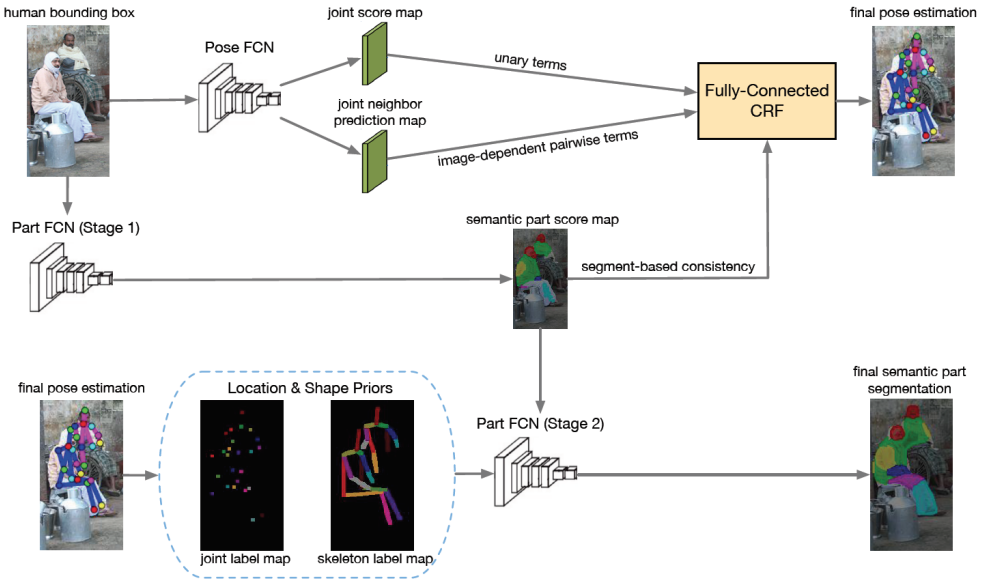


Figure 2: The framework of our approach for joint pose estimation and part segmentation. Initial joint scores and part segment scores are fused to yield better pose estimation results, and then the estimated poses are used to refine part segmentation.

30.4 PoseTrack

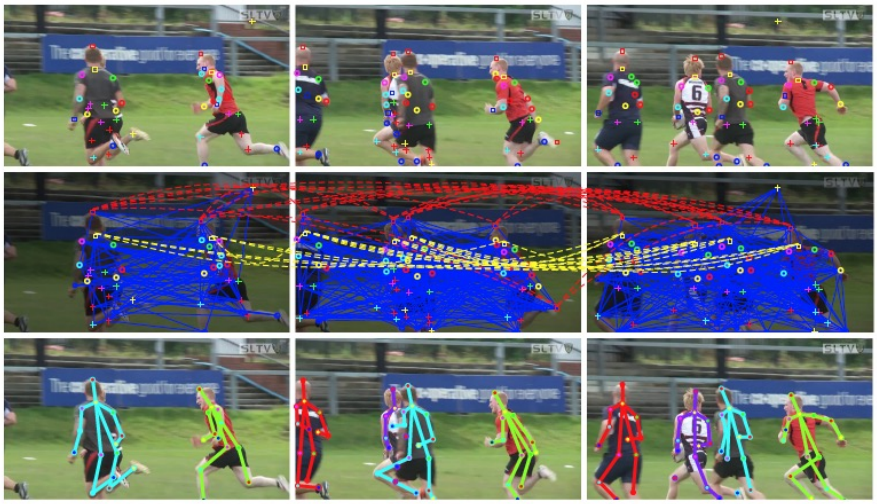


Figure 2: **Top:** Body joint detection hypotheses shown for three frames. **Middle:** Spatio-temporal graph with spatial edges (blue) and temporal edges for head (red) and neck (yellow). We only show a subset of the edges. **Bottom:** Estimated poses for all persons in the video. Each color corresponds to a unique person identity.

A partitioning is obtained by minimizing the cost function

$$\underset{v,s,t}{\operatorname{argmin}} \left(\langle v, \phi \rangle + \langle s, \psi_s \rangle + \langle t, \psi_t \rangle \right) \tag{3}$$

$$\langle v, \phi \rangle = \sum_{d \in D} v_d \phi(d) \tag{4}$$

$$\langle s, \psi_s \rangle = \sum_{(d_f, d'_f) \in E_s} s_{(d_f, d'_f)} \psi_s(d_f, d'_f) \tag{5}$$

$$\langle t, \psi_t \rangle = \sum_{(d_f, d'_{f'}) \in E_t} t_{(d_f, d'_{f'})} \psi_t(d_f, d'_{f'}). \tag{6}$$

This means that we search for a graph partitioning such that the cost of the remaining nodes and edges is minimal. The cost for a node d is defined by the unary term:

$$\phi(d) = \log \frac{1 - p_d}{p_d} \tag{7}$$

where $p_d \in (0, 1)$ corresponds to the probability of the joint hypothesis d. Note that $\phi(d)$ is negative when $p_d > 0.5$ and detections with a high confidence are preferred since they reduce the cost function (3). The cost for a spatial or tem-

poral edge is defined similarly by

$$\psi_s(d_f, d'_f) = \log \frac{1 - p^s_{(d_f, d'_f)}}{p^s_{(d_f, d'_f)}} \tag{8}$$

$$\psi_t(d_f, d'_{f'}) = \log \frac{1 - p^t_{(d_f, d'_{f'})}}{p^t_{(d_f, d'_{f'})}}. \tag{9}$$

While p^s denotes the probability that two joint detections d and d' in a frame f belong to the same person, p^t denotes the probability that two detections of a joint in frame f and f' are the same. In Sec. 3.4 we will discuss how the probabilities p_d, $p^s_{(d_f, d'_f)}$, and $p^t_{(d_f, d'_{f'})}$ are learned.

- parameter to optimize: v, s, t. all of which are 0-1 integer.

- the spatial pairwise potential is determined by IoU; the temporal pairwise potential is determined by DeepMatching correspondense and logistic regression on feature vector.

- the optimization is a ILP problem, the constraints are very tricky.

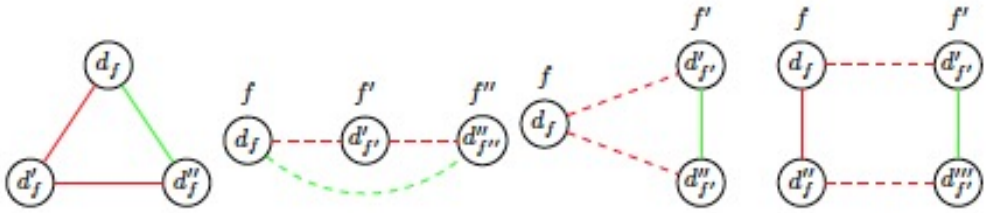


Figure 3: *(a)* The spatial transitivity constraints (12) ensure that if the two joint hypotheses d_f and d''_f are spatially connected to d'_f (red edges) then the cost of the spatial edge between d_f and d''_f (green edge) also has to be added. *(b)* The temporal transitivity constraints (13) ensure transitivity for temporal edges (dashed). *(c)* The spatio-temporal transitivity constraints (14) model transitivity for two temporal edges and one spatial edge. *(d)* The spatio-temporal consistency constraints (15) ensure that if two pairs of joint hypotheses $(d_f, d'_{f'})$ and $(d''_f, d'''_{f'})$ are temporally connected (dashed red edges) and d_f and d''_f are spatially connected (solid red edge) then the cost of the spatial edge between $d'_{f'}$ and $d'''_{f'}$ (solid green edge) also has to be added.

30.5 DeepCut

$y_{dd'}$ indicates that the body part candidates d and $d^{'}$ belong to the same person, and $z_{dd'cc'}$ are auxiliary variables to relate x and y through $z_{dd'cc'} = x_{dc}x_{d'c'}y_{dd'}$

c means class, d means body part candidate

The optimization problem that we call the *subset partition and labeling problem* is the ILP that minimizes over the set of feasible solutions X_{DC}:

$$\min_{(x,y,z)\in X_{DC}} \langle \alpha, x\rangle + \langle \beta, z\rangle, \tag{6}$$

where we used the short-hand notation

$$\alpha_{dc} := \log \frac{1-p_{dc}}{p_{dc}} \tag{7}$$

$$\beta_{dd'cc'} := \log \frac{1-p_{dd'cc'}}{p_{dd'cc'}} \tag{8}$$

$$\langle \alpha, x\rangle := \sum_{d\in D}\sum_{c\in C} \alpha_{dc}\, x_{dc} \tag{9}$$

$$\langle \beta, z\rangle := \sum_{dd'\in\binom{D}{2}} \sum_{c,c'\in C} \beta_{dd'cc'}\, z_{dd'cc'} \ . \tag{10}$$

The objective (6)–(10) is the MAP estimate of a probability measure of joint detections x and clusterings y, z of body parts, where prior probabilities p_{dc} and $p_{dd'cc'}$ are estimated *independently* from data, and the likelihood is a positive constant if (x, y, z) satisfies (1)–(4), and is 0, otherwise. The exact form (6)–(10) is obtained when minimizing the negative logarithm of this probability measure.

$$\forall d \in D \forall cc' \in \binom{C}{2}: \quad x_{dc} + x_{dc'} \leq 1 \tag{1}$$

$$\forall dd' \in \binom{D}{2}: \quad y_{dd'} \leq \sum_{c \in C} x_{dc}$$

$$y_{dd'} \leq \sum_{c \in C} x_{d'c} \tag{2}$$

$$\forall dd'd'' \in \binom{D}{3}: \quad y_{dd'} + y_{d'd''} - 1 \leq y_{dd''} \tag{3}$$

$$\forall dd' \in \binom{D}{2} \forall cc' \in C^2: \quad x_{dc} + x_{d'c'} + y_{dd'} - 2 \leq z_{dd'cc'}$$

$$z_{dd'cc'} \leq x_{dc}$$

$$z_{dd'cc'} \leq x_{d'c'}$$

$$z_{dd'cc'} \leq y_{dd'} \tag{4}$$

When at most one person is in an image, we further constrain the feasible solutions to a well-defined pose of a single person. This is achieved by an additional class of inequalities which guarantee, for any two distinct body parts that are not suppressed, that they must be clustered together:

$$\forall dd' \in \binom{D}{2} \forall cc' \in C^2: \quad x_{dc} + x_{d'c'} - 1 \leq y_{dd'} \tag{5}$$

- (1).guarantee that every body part is labeled with at most one body part class.

- (2). guarantee that distinct body parts d and $d^{'}$ belong to the same person only if neither d nor $d^{'}$ is suppressed.

- (3). for any three pairwise distinct body parts, d, $d^{'}$ and $d^{''}$, if d and $d^{'}$ are the same person and $d^{'}$ and $d^{''}$ are the same person , then also d and $d^{''}$ are the same person ($y^{dd^{''}}$ = 1), that is, transitivity

- the larger the probability is, the smaller the $\alpha_{dc}, \beta_{dd'cc'}$ will be. so the optimization will force the final result to choose the high probability joint proposal.

30.6 Thin-Slicing Network: A Deep Structured Model for Pose Estimation in Videos

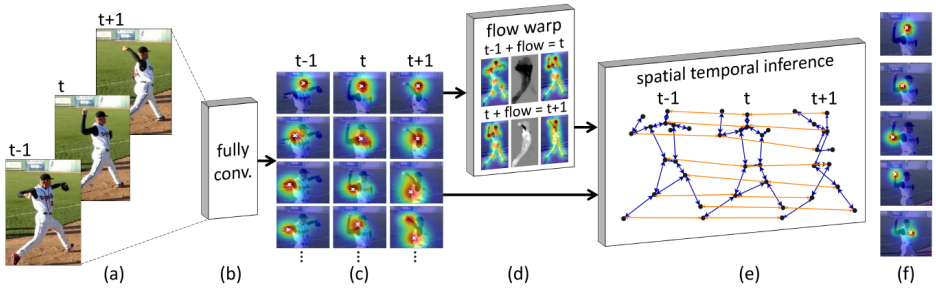


Figure 2. **Schematic overview of Thin-Slicing Network architecture**. Our model takes a small number of adjacent frames as input (a) and fully convolutional layers (b) regress initial body joint position estimates (c). We compute dense optical flow between neighboring frames to propagate joint position estimates through time. A flow based warping layer aligns joint heat-maps to the current frame (d). A spatio-temporal inference layer performs iterative message passing along both spatial and temporal edges of the loopy pose configuration graph (e) and computes final joint position estimates (f).

- serious drawbacks: only single person video is suitable!
- Flow warping layer: optical flow need to be evaluated per pixel.
- quadratic spring model is adopted in the spatial and temporal.
- if the skeleton points is 17, then the |V| = 17, |E| = 1716/2 + 17temporal_gap
  - $S(I,p) = \sum_{i \in V} \phi_i(p_i|I) + \sum_{(i,j) \in E_s} \phi_{i,j}(p_i, p_j)$

  where the $\phi_i(p_i|I)$ is the unary term, for pairwise term we use a spring energy model to measure the deformation cost, where $\phi_{i,j}(p_i, p_j)$ is defined as $w_{i,j}d(p_i - p_j)$. with standard quadratic deformation constraints $d(p_i - p_j) = [\Delta x, \Delta x^2, \Delta y, \Delta y^2]^T$, the $w_{i,j}$ can be learned by gradient descent.
  the objective score function of the entire slice with temporal constraints is then given by:
  $S(I,P)_{slice} = \sum_{t=1}^{T} S(I^t, p^t) + \sum_{(i,i^*) \in E_f} \phi_{i,i^*}(p_i, p_{i*}^{'})$
  specially, here $p_{i*}^{'} = p_{i*} + f_{i^*,i}(p_{i*})$ and $f_{i*,i}(p_{i*})$ is the optical flow evaluated at p_{i*}
  for temporal pairwise term, the same quadratic spring model to penalize the estimation drift between these neighbouring frames
- the Loopy Belief propagation algorithms such as the max-sum algorithm make approximate inference possible in intractable loopy models.

- the graph model optimization parameters: $w_{i,j}$,which includes spatial and temporal pairwise weight!
- The learning of Thin-Slicing Network is decomposed into two stages: (1) Training fully convolutional layers and (2) Joint training with flow warping and inference layers, by initializing the weights of the fully convolutional layers with the pre-trained paramters in stage(1). the Loss function of stage(1) is MSE, the loss function of stage(2) is Hinge Loss.
- the training of graph model is minimizing the hinge loss, the inference of the graph model is iteratively maximize the graph model until convergence.

Acknowledgments

References

[1] Vasileios Belagiannis and Andrew Zisserman. Recurrent human pose estimation. In *Automatic Face & Gesture Recognition (FG 2017), 2017 12th IEEE International Conference on*, pages 468–475. IEEE, 2017.

[2] David Berthelot, Tom Schumm, and Luke Metz. Began: Boundary equilibrium generative adversarial networks. *arXiv preprint arXiv:1703.10717*, 2017.

[3] 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.

[4] Liang-Chieh Chen, George Papandreou, Florian Schroff, and Hartwig Adam. Rethinking atrous convolution for semantic image segmentation. *arXiv preprint arXiv:1706.05587*, 2017.

[5] Chia-Jung Chou, Jui-Ting Chien, and Hwann-Tzong Chen. Self adversarial training for human pose estimation. *arXiv preprint arXiv:1707.02439*, 2017.

[6] Xiao Chu, Wei Yang, Wanli Ouyang, Cheng Ma, Alan L Yuille, and Xiaogang Wang. Multi-context attention for human pose estimation. *arXiv preprint arXiv:1702.07432*, 2017.

[7] Jifeng Dai, Kaiming He, Yi Li, Shaoqing Ren, and Jian Sun. Instance-sensitive fully convolutional networks. In *European Conference on Computer Vision*, pages 534–549. Springer, 2016.

[8] Michaël Defferrard, Xavier Bresson, and Pierre Vandergheynst. Convolutional neural networks on graphs with fast localized spectral filtering. In *Advances in Neural Information Processing Systems*, pages 3844–3852, 2016.

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

[10] Ross Girshick. Fast r-cnn. In *Proceedings of the IEEE international conference on computer vision*, pages 1440–1448, 2015.

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

[12] Ishaan Gulrajani, Faruk Ahmed, Martin Arjovsky, Vincent Dumoulin, and Aaron Courville. Improved training of wasserstein gans. *arXiv preprint arXiv:1704.00028*, 2017.

[13] Kaiming He, Georgia Gkioxari, Piotr Dollár, and Ross Girshick. Mask r-cnn. *arXiv preprint arXiv:1703.06870*, 2017.

[14] 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.

[15] Sergey Ioffe and Christian Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. In *International Conference on Machine Learning*, pages 448–456, 2015.

[16] Thomas N Kipf and Max Welling. Semi-supervised classification with graph convolutional networks. *arXiv preprint arXiv:1609.02907*, 2016.

[17] Philipp Krähenbühl and Vladlen Koltun. Efficient inference in fully connected crfs with gaussian edge potentials. In *Advances in neural information processing systems*, pages 109–117, 2011.

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

[19] Qizhu Li, Anurag Arnab, and Philip HS Torr. Holistic, instance-level human parsing.

[20] Yi Li, Kaiming He, Jian Sun, et al. R-fcn: Object detection via region-based fully convolutional networks. In *Advances in Neural Information Processing Systems*, pages 379–387, 2016.

[21] Yi Li, Haozhi Qi, Jifeng Dai, Xiangyang Ji, and Yichen Wei. Fully convolutional instance-aware semantic segmentation. *arXiv preprint arXiv:1611.07709*, 2016.

[22] Tsung-Yi Lin, Piotr Dollár, Ross Girshick, Kaiming He, Bharath Hariharan, and Serge Belongie. Feature pyramid networks for object detection. *arXiv preprint arXiv:1612.03144*, 2016.

[23] 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.

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

[25] Mehdi Mirza and Simon Osindero. Conditional generative adversarial nets. *arXiv preprint arXiv:1411.1784*, 2014.

[26] Shih-En Wei, Varun Ramakrishna, Takeo Kanade, and Yaser Sheikh. Convolutional pose machines. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 4724–4732, 2016.

[27] 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.

[28] Fangting Xia, Peng Wang, Xianjie Chen, and Alan Yuille. Joint multi-person pose estimation and semantic part segmentation. *arXiv preprint arXiv:1708.03383*, 2017.

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