Deep Learning and Computer Vision

Vincent Lepetit

September 14, 2021

Today

“Deep” Embeddings:

- Embeddings and autoencoders;
- Image Embeddings;
- Embeddings and Siamese Networks;
- Variational Autoencoders;

Deep Learning applied to Computer Vision problems:

- Object Detection;
- Image Segmentation;

To Go Further

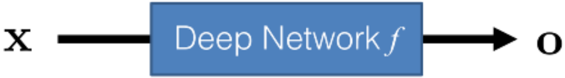
- Articulated Object Detection;
- Dealing with Unstructured Input;
- Spatial Transformers;
- Self-Learning;
- Differentiable Pipelines and End-to-End Learning.

“Deep” Embeddings

- Embeddings and autoencoders;
- Embeddings and features;
- Embeddings and Siamese Networks;
- Variational Autoencoders;

Beyond Supervised Learning

- We can use a Deep Network to approximate any continuous function;



$\mathbf{x}$

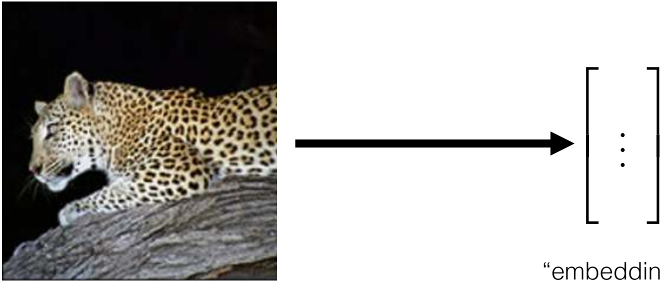
Deep Network f

$\mathbf{o}$

- We can use any loss function as long as it is differentiable;

→ very flexible!

“Embedding” Concept

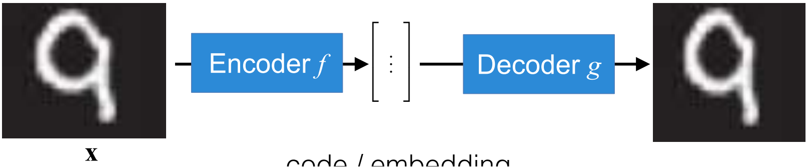


“embedding”

compact representation of the image, can be used to compare images, for example.

How can we compute a good embedding for a given image?

Autoencoders



$\mathbf{x}$

Encoder f

Decoder g

code / embedding

Loss function:

$$\mathcal{L}(\Theta_1, \Theta_2) = \sum_{\mathbf{x}} \|g(f(\mathbf{x}; \Theta_1); \Theta_2) - \mathbf{x}\|^2 \,. \tag{1}$$

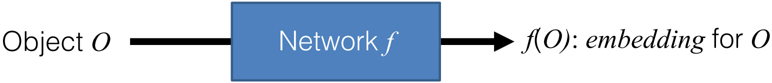
Dimensionality reduction and reconstruction:

$$\begin{aligned} \text{code} &\leftarrow f(\mathbf{x}; \Theta_1) \\ \mathbf{x}' &\leftarrow g(\text{code}; \Theta_2) \end{aligned} \tag{2}$$

Does not work well in practice without additional constraints.

Y. Bengio. “Learning Deep Architectures for AI”. In: *Foundations and Trends in Machine Learning* (2009).

Siamese Networks, Embeddings, and Metric Learning



How can we train f so that the distance $\|f(O_1;\Theta) - f(O_2;\Theta)\|$ is representative of the similarity between O_1 and O_2?

I. Bromley, I. Guyon, and Y. LeCun. “Signature Verification using a Siamese Time Delay Neural Network”. In: *Advances in Neural Information Processing Systems*. 1994.

Positive and Negative Pairs

$\mathbf{x}_1^p$

$\mathbf{x}_2^p$

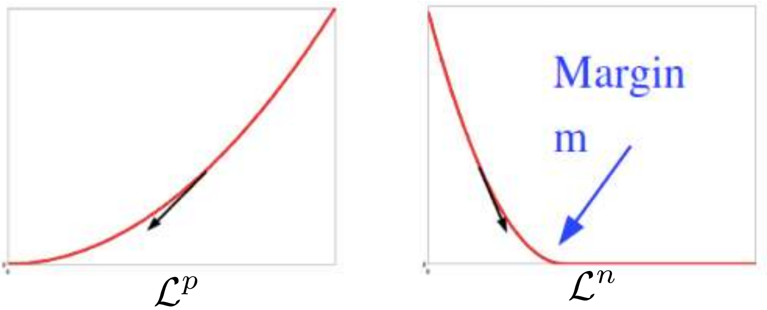
$\mathbf{x}_1^n$

$\mathbf{x}_1^n$

Make

- $\mathcal{L}^p(\mathbf{x}_1^p, \mathbf{x}_2^p; \Theta) = \| f(\mathbf{x}_1^p; \Theta) - f(\mathbf{x}_2^p; \Theta) \|$ small, and
- $\mathcal{L}^n(\mathbf{x}_1^n, \mathbf{x}_2^n; \Theta) = \| f(\mathbf{x}_1^n; \Theta) - f(\mathbf{x}_2^n; \Theta) \|$ large.

Contrastive Loss

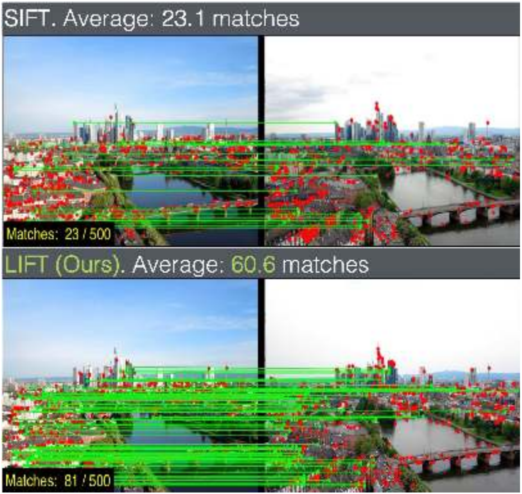


- $\mathcal{L}^p(\mathbf{x}_1^p, \mathbf{x}_2^p; \Theta) = \frac{1}{2}\|f(\mathbf{x}_1^p; \Theta) - f(\mathbf{x}_2^p; \Theta)\|^2$
- $\mathcal{L}^n(\mathbf{x}_1^n, \mathbf{x}_2^n; \Theta) = \frac{1}{2}[\max(0, m - \|f(\mathbf{x}_1^n; \Theta) - f(\mathbf{x}_2^n; \Theta)\|^2)]$

Need hard-example mining to avoid flat gradients.

See also triplet loss.

Application


SIFT. Average: 23.1 matches
Matches: 23 / 500
LIFT (Ours). Average: 60.6 matches
Matches: 81 / 500

K. M. Yi et al. “LIFT: Learned Invariant Feature Transform”. In: *European Conference on Computer Vision*. 2016.

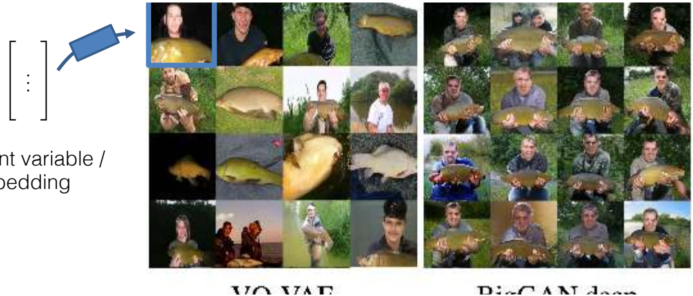
“Deep” Embeddings

- Embeddings and autoencoders;
- Embeddings and features;
- Embeddings and Siamese Networks;
- Variational Autoencoders.

Variational Autoencoders (VAE)

Very different from autoencoders! Only resemble them during training.

Given a training set X_i, we want to generate new samples X that follow the same distribution as the X_i's. (See also Generative Adversarial Networks)



(training set: fishing images)

References

Original paper:
Diederik P. Kingma and Max Welling. "Auto-encoding variational Bayes". In: *International Conference for Learning Representations*. 2014.

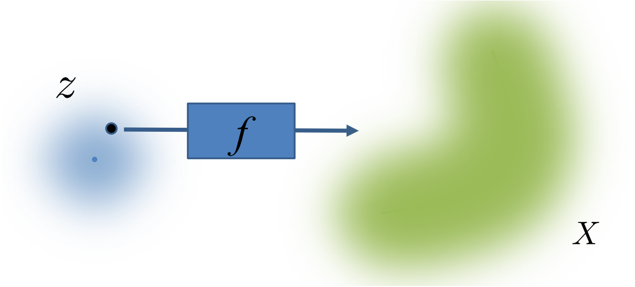
A good tutorial:
Carl Doersch. "Tutorial on Variational Autoencoders". In: *arXiv Preprint*. 2016.

The original paper is very general, we will focus here on some specific but common choices.

Variational Autoencoders (VAE)

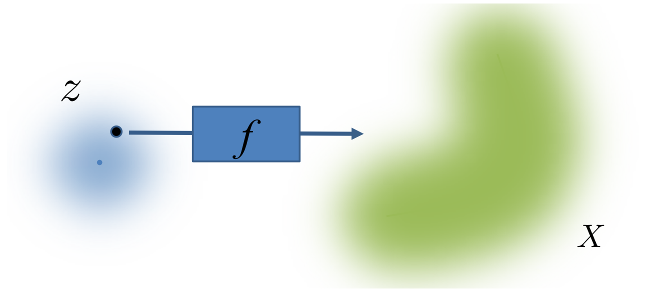
Given a set X_i, we want to generate new samples X that follow the same distribution as the X_i's.

We will do this by training a network $f(w;\Theta)$ such that, given a latent variable $z \sim P(z)$ (usually $z \sim \mathcal{N}(\mathbf{0},\mathbf{I})$), $f(z;\Theta) \sim P(X)$.



Diederik P. Kingma and Max Welling. "Auto-encoding variational Bayes". In: *International Conference for Learning Representations*. 2014.

Looking for Θ



We have:

$$P(X) = \int_z P(X \mid z; \Theta) P(z) dz \,. \tag{3}$$

We want to find the parameters Θ of network $f(z;\Theta)$.

This is done by optimizing:

$$\arg\max_{\Theta} \sum_i \log P(X_i) \,. \tag{4}$$

How can we compute $P(X_i)$?

Looking for Θ

We have:

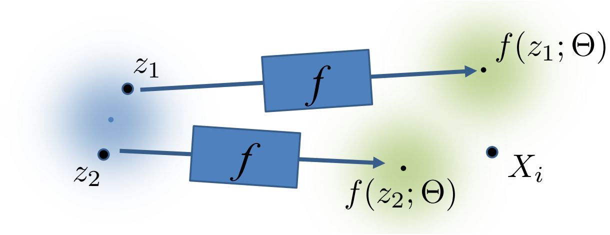
$$P(X) = \int_z P(X \mid z; \Theta) P(z) dz \,. \tag{5}$$

One way to estimate $P(X_i)$ is to sample many z_j, and take

$$P(X_i) \approx \frac{1}{N} \sum_j P(X \mid z_j; \Theta) \,. \tag{6}$$

Usually, $P(X \mid z; \Theta)$ is taken to follow a Gaussian distribution:

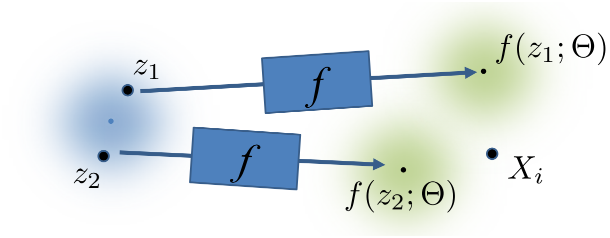
$$P(X \mid z; \Theta) = \mathcal{N}(X \mid f(z; \Theta), \sigma^2 \mathbf{I}) \,. \tag{7}$$



Looking for Θ

One way to estimate $P(X_i)$ is to sample many z_j, and take

$$P(X_i) \approx \frac{1}{N} \sum_j P(X \mid z_j; \Theta) \,. \tag{8}$$



Unfortunately, in large dimensions, for most z, $P(X \mid z; \Theta) \approx 0$, especially when Θ is initialized randomly.

Key idea in Variational Autoencoders: Sample values of z that are likely to have produced X.

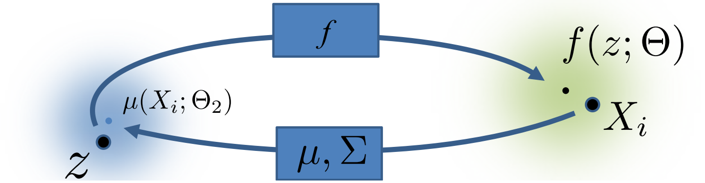
z given an X

Key idea in Variational Autoencoders: Sample values of z that are likely to have produced X.

We consider:

$$P(z \mid X; \Theta_2) = \mathcal{N}(z \mid \mu(X; \Theta_2), \Sigma(X; \Theta_2)), \tag{9}$$

where μ and Σ are implemented as neural networks. Σ predicts a diagonal matrix.



Deriving the Optimization Problem

After some derivations (see [Doersch16]):

$$\begin{aligned} &\log P(X) - \mathcal{D}[P(z|X;\Theta_2)\|P(z|X)] \\ =& E_{z\sim P(z\,|\,X;\Theta_2)}[\log P(X|z;\Theta)] - \mathcal{D}[P(z|X;\Theta_2)\|P(z)]\,, \end{aligned} \quad (10)$$

where $\mathcal{D}$ is the Kullback-Liebler divergence.

Remember our choices:

- $P(X \mid z;\Theta) = \mathcal{N}(X \mid f(z;\Theta), \sigma^2\mathbf{I})$
- $P(z \mid X;\Theta_2) = \mathcal{N}(z \mid \mu(X;\Theta_2), \Sigma(X;\Theta_2))$
- $P(z) = \mathcal{N}(\mathbf{0}, \mathbf{I})$

(a) About term $\mathcal{D}[P(z|X;\Theta_2)\|P(z|X)]$:

- $P(z|X)$ is the actual distribution of z given X. *We cannot compute it*;
- $P(z|X;\Theta_2)$ is the one we compute with networks μ and Σ.

These two distributions are the same only once the networks converged correctly, and then $\mathcal{D}[P(z|X;\Theta_2)\|P(z|X)] = 0$.

Deriving the Optimization Problem

We have

$$\begin{aligned} &\log P(X) - \mathcal{D}[P(z|X;\Theta_2)\|P(z|X)] \\ = &E_{z\sim P(z\,|\,X;\Theta_2)}[\log P(X|z;\Theta)] - \mathcal{D}[P(z|X;\Theta_2)\|P(z)], \end{aligned} \qquad (11)$$

with

- $P(X \mid z;\Theta) = \mathcal{N}(X \mid f(z;\Theta), \sigma^2\mathbf{I})$
- $P(z \mid X;\Theta_2) = \mathcal{N}(z \mid \mu(X;\Theta_2), \Sigma(X;\Theta_2))$
- $P(z) = \mathcal{N}(\mathbf{0}, \mathbf{I})$

(b) About term $E_{z\sim P(z\,|\,X;\Theta_2)}[\log P(X|z;\Theta)]$:

- $P(X|z;\Theta)$ can be computed easily given a z and an X;
- Note the expectation is over a distribution predicted by μ and Σ - this will have an influence on the final algorithm.

Deriving the Optimization Problem

We have

$$\begin{aligned} &\log P(X) - \mathcal{D}[P(z|X;\Theta_2) \| P(z|X)] \\ = &E_{z \sim P(z \mid X;\Theta_2)}[\log P(X|z;\Theta)] - \mathcal{D}[P(z|X;\Theta_2) \| P(z)]\,, \end{aligned} \tag{12}$$

with

- $P(X \mid z;\Theta) = \mathcal{N}(X \mid f(z;\Theta), \sigma^2 \mathbf{I})$
- $P(z \mid X;\Theta_2) = \mathcal{N}(z \mid \mu(X;\Theta_2), \Sigma(X;\Theta_2))$
- $P(z) = \mathcal{N}(\mathbf{0}, \mathbf{I})$

(c) Term $\mathcal{D}[P(z|X;\Theta_2) \| P(z)]$ can be computed in closed form (see [Doersch16] for exact formula).

Optimization Problem

We have:

$$\begin{aligned} &\log P(X) - \mathcal{D}[P(z|X;\Theta_2)\|P(z|X)] \\ = &E_{z\sim P(z\mid X;\Theta_2)}[\log P(X|z;\Theta)] - \mathcal{D}[P(z|X;\Theta_2)\|P(z)] \,. \end{aligned} \tag{13}$$

We would like to look for Θ and Θ_2 simultaneously by optimizing:

$$\max_{\Theta,\Theta_2} \sum_i \log P(X_i) \,, \tag{14}$$

but we do not know how to compute $\mathcal{D}[P(z|X;\Theta_2)\|P(z|X)]$!

However, $\mathcal{D}[P(z|X;\Theta_2)\|P(z|X)] = 0$ for correct values of Θ_2. Variational Autoencoders ignore this term and hope that maximizing the right hand side will make Θ_2 converge to good values, which will make this term converge to 0. The approximation made by ignoring this term will then become better and better.

Final Optimization Problem

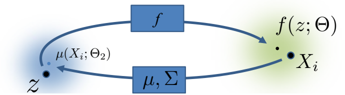
Variational Autoencoders look for Θ and Θ_2 simultaneously by optimizing:

$$\max_{\Theta,\Theta_2} \sum_i \mathcal{L}(X;\Theta,\Theta_2)\,,$$

with

$$\mathcal{L}(X;\Theta,\Theta_2) = E_{z\sim P(z|X;\Theta_2)}[\log P(X|z;\Theta)] - \mathcal{D}[P(z|X;\Theta_2)\|P(z)]\,.$$

Computing the Gradients



$$\mathcal{L}(X;\Theta,\Theta_2) = E_{z\sim P(z|X;\Theta_2)}[\log P(X|z;\Theta)] - \mathcal{D}[P(z|X;\Theta_2)\|P(z)]\,.$$

Possible algorithm:

- Draw 1 sample X from training data;
- Draw 1 z from $\mathcal{N}(\mu(X;\Theta_2),\Sigma(X;\Theta_2))$;
- $\frac{\partial\mathcal{L}}{\partial\Theta} \propto \|X - f(z;\Theta)\|^2$;

Problem: $E_{z\sim P(z\,|\,X;\Theta_2)}[\log P(X|z;\Theta)]$ depends on Θ_2 but we cannot compute its gradient wrt Θ_2! This is because the sampling operation for z is not differentiable.

Computing the Gradients: The Reparametrization Trick

$$\mathcal{L}(X;\Theta,\Theta_2) = E_{z\sim P(z|X;\Theta_2)}[\log P(X|z;\Theta)] - \mathcal{D}[P(z|X;\Theta_2)\|P(z)]\,.$$

New algorithm with the reparameterization trick:

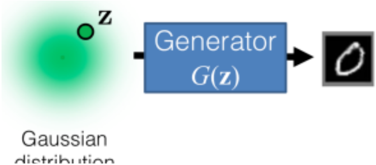
- Draw 1 sample X from training data;
- Reparameterisation trick: Draw 1 ϵ from constant distribution $\mathcal{N}(0,\mathbf{I})$;
- Take $z = \mu(X;\Theta_2) + \Sigma(X;\Theta_2)\epsilon$;

We can now compute the gradients of $\mathcal{L}$ wrt Θ and Θ_2 for samples X and z:

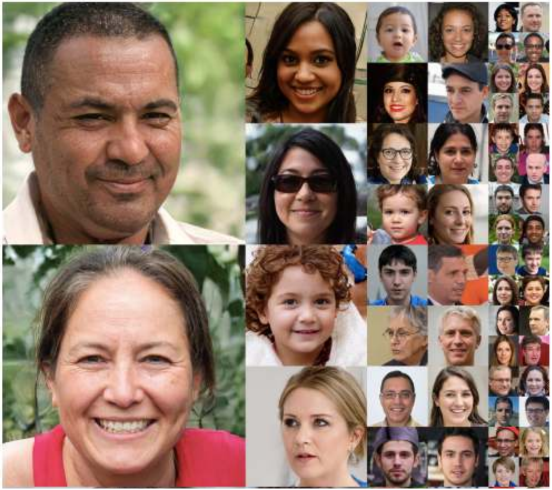
- $P(X|z;\Theta) = \mathcal{N}(X \mid f(z;\Theta), \sigma^2\mathbf{I})$ and its gradient can be computed in closed form (note z depends on Θ_2);
- $\mathcal{D}[P(z|X;\Theta_2)\|P(z)]$ can be computed in closed form so its gradient.

Generative Adversarial Networks

We would like to train a network G to generate images from some domain from random vectors $\mathbf{z}$:

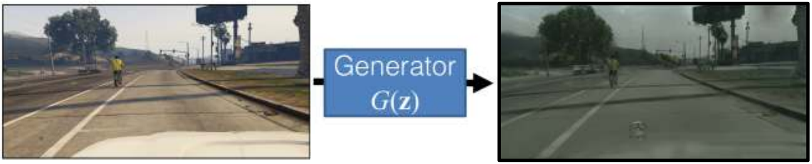


Applications: Realistic Rendering



Karras19.

Applications: Domain Adaptation



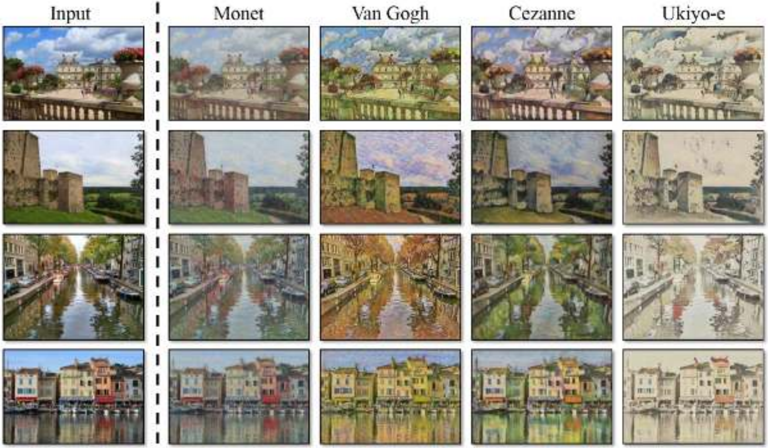
Input Synthetic Image z

Output image, made to look

Useful for training on synthetic images.

Zhu17.

Applications: Style Transfer



Zhu17.

Other Applications

- Efficient approximations of scientific models:

  Mustafa19.

  SHiP19.

- Generating new molecules:

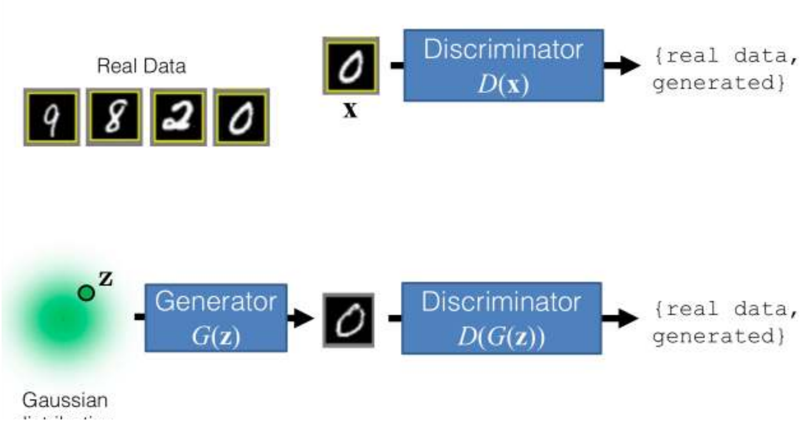
  Zhavoronkov19.

- Deep Fakes.

GANs: Original Formulation

Idea: Add a classification network (Discriminator D) jointly trained with the Generator G to recognize if an input is a real sample from the domain of interest or if it was created by the Generator.

When the Discriminator cannot distinguish the generated images from the real ones, the Generator generates realistic images.



GANs: Original Formulation

$$\min_G \max_D \mathbb{E}_{\mathbf{x}\sim p_{\mathsf{data}}(\mathbf{x})}[\log D(\mathbf{x})] + \mathbb{E}_{\mathbf{z}\sim p_{\mathbf{z}}(\mathbf{z})}[\log(1 - D(G(\mathbf{z})))]$$

or

$$\min_G \max_D \frac{1}{|\mathcal{T}|} \sum_{\mathbf{x}\in\mathcal{T}} \log D(\mathbf{x}) + \frac{1}{N_{\mathbf{z}}} \sum_{\mathbf{z}\sim\mathcal{N}(\mathbf{0};\mathbf{I})} \log(1 - D(G(\mathbf{z}))) \,,$$

with

- $\mathcal{T}$ is the set of real data;
- $D(\mathbf{x}) \in [0,1]$, $0 \leftrightarrow$ `synthetic`, $1 \leftrightarrow$ `real`.

Goodfellow14.

GANs: Original Formulation (2)

$$\min_G \max_D \frac{1}{|\mathcal{T}|} \sum_{\mathbf{x} \in \mathcal{T}} \log D(\mathbf{x}) + \frac{1}{N_{\mathbf{z}}} \sum_{\mathbf{z} \sim \mathcal{N}(\mathbf{0};\mathbf{I})} \log(1 - D(G(\mathbf{z}))) \,,$$

with $D(\mathbf{x}) \in [0,1]$, $0 \leftrightarrow$ `synthetic`, $1 \leftrightarrow$ `real`.

For a given Discriminator, we want:

$$\min_G \frac{1}{N_{\mathbf{z}}} \sum_{\mathbf{z} \sim \mathcal{N}(\mathbf{0};\mathbf{I})} \log(1 - D(G(\mathbf{z}))) \,,$$

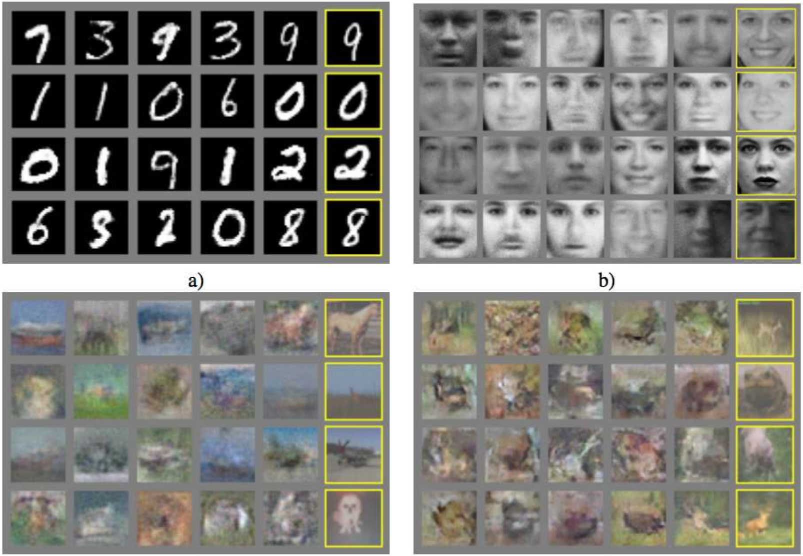
and for a given Generator, we want:

$$\max_D \frac{1}{|\mathcal{T}|} \sum_{\mathbf{x} \in \mathcal{T}} \log D(\mathbf{x}) + \frac{1}{N_{\mathbf{z}}} \sum_{\mathbf{z} \sim \mathcal{N}(\mathbf{0};\mathbf{I})} \log(1 - D(G(\mathbf{z}))) \,.$$

Original Algorithm

```
for number of training iterations do
    for $k$ steps do
        Sample minibatch of $m$ noise samples $\{\mathbf{z}^{(1)}, \ldots, \mathbf{z}^{(m)}\}$
        Sample minibatch of $m$ real samples $\{\mathbf{x}^{(1)}, \ldots, \mathbf{x}^{(m)}\}$
        Update the discriminator $D$ by stochastic gradient ascend.
    end for
    Sample minibatch of $m$ noise samples $\{\mathbf{z}^{(1)}, \ldots, \mathbf{z}^{(m)}\}$
    Update the generator $G$ by stochastic gradient descend.
end for
```

Results



a)

b)

c)

d)

Original Formulation: Limitation

Mode collapse: A Generator providing always the same (realistic) output whatever input $\mathbf{z}$ is an optimal (but not desirable) solution:

$$\min_{G} \max_{D} \frac{1}{|\mathcal{T}|} \sum_{\mathbf{x} \in \mathcal{T}} \log D(\mathbf{x}) + \frac{1}{N_{\mathbf{z}}} \sum_{\mathbf{z} \sim \mathcal{N}(\mathbf{0};\mathbf{I})} \log(1 - D(G(\mathbf{z}))) \,.$$

Solution: Wasserstein GANs

Reformulate the problem as:

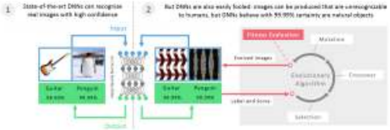
The distribution of the generated data should be as close as possible to the distribution of the real data.

Arjosky17.

Wasserstein GAN: Formalization

Denote:

- $\mathbb{P}_r$: the distribution of the real data;
- $\mathbb{P}_\theta$: the distribution of the generated data.


1
State-of-the-art DNNs can recognize real images with high confidence
2
But DNNs are also easily fooled: images can be produced that are unrecognizable to humans, but DNNs believe with 99.99% certainty are natural objects
Input
Guitar
Penguin
Output
Fitness Evaluation
Mutation
Evolved images
Evolutionary Algorithm
Crossover
Label and Score
Selection

Mostly because Deep Networks are trained in a discriminative way.

“Adversarial” examples can be used for retraining to improve robustness.

Optimization problem:

$$\min_\theta W(\mathbb{P}_r, \mathbb{P}_\theta),$$

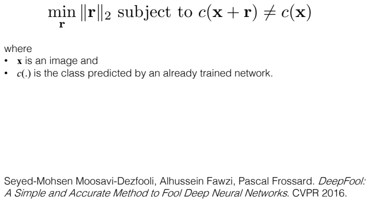
where $W(.,.)$ is a distance between two distributions.

θ will be the parameters of the Generator.

A Suitable Distance

Earth Mover distance between two distributions (also called Wasserstein metric).

The Wasserstein metric is the cost of optimal transport between the two distributions:



$$\min_{\mathbf{r}} \|\mathbf{r}\|_2 \text{ subject to } c(\mathbf{x}+\mathbf{r}) \neq c(\mathbf{x})$$

where

- $\mathbf{x}$ is an image and
- $c(.)$ is the class predicted by an already trained network.

Seyed-Mohsen Moosavi-Dezfooli, Alhussein Fawzi, Pascal Frossard. *DeepFool: A Simple and Accurate Method to Fool Deep Neural Networks*. CVPR 2016.

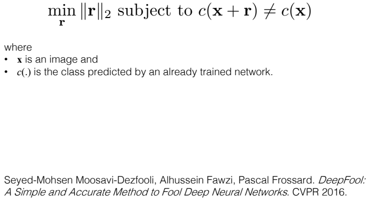
Formally:

$$W(\mathbb{P}_r, \mathbb{P}_\theta) = \inf_{\gamma \in \pi(\mathbb{P}_r, \mathbb{P}_\theta)} \mathbb{E}_{(x,y)\sim\gamma}\left[\|x-y\|\right],$$

where $\gamma(.,y) = \mathbb{P}_r$ and $\gamma(x,.) = \mathbb{P}_\theta$.

Wasserstein Metric: More Intuition

The Wasserstein metric is the cost of optimal transport between the two distributions:



$$\min_{\mathbf{r}} \|\mathbf{r}\|_2 \text{ subject to } c(\mathbf{x}+\mathbf{r}) \neq c(\mathbf{x})$$

where

- $\mathbf{x}$ is an image and
- $c(.)$ is the class predicted by an already trained network.

Seyed-Mohsen Moosavi-Dezfooli, Alhussein Fawzi, Pascal Frossard. *DeepFool: A Simple and Accurate Method to Fool Deep Neural Networks*. CVPR 2016.

Intuitive (but imperfect) view:

$$W(\mathbb{P}_r, \mathbb{P}_\theta) = \min_{\gamma \in \Gamma} \sum_{(x,y) \in \gamma} \left[\|x - y\| \right],$$

where Γ is the set of all possible sets of correspondences between x and y.

Important Property

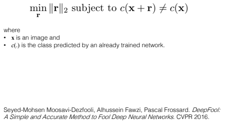
If $\mathbb{P}_\theta$ is the distribution of the output of a deep network g_θ, then the Wasserstein metric $W(\mathbb{P}_r, \mathbb{P}_\theta)$ is continuous in θ.

This is not the case for other distances and divergences between distributions.

Will allow us to train g_θ.

Proof in **Arjosky17**.

$\mathbb{P}_\theta$ is now the distribution of the network outputs $g_\theta(z)$ with $z \sim \mathcal{N}(\mathbf{0}; \mathbf{I})$.



$$\min_{\mathbf{r}} \|\mathbf{r}\|_2 \text{ subject to } c(\mathbf{x}+\mathbf{r}) \neq c(\mathbf{x})$$

where

- $\mathbf{x}$ is an image and
- $c(.)$ is the class predicted by an already trained network.

Seyed-Mohsen Moosavi-Dezfooli, Alhussein Fawzi, Pascal Frossard. *DeepFool: A Simple and Accurate Method to Fool Deep Neural Networks*. CVPR 2016.

How Can We Compute $W(\mathbb{P}_r, \mathbb{P}_\theta)$?

Wasserstein GANs rely on the following expression (Kantorovich-Rubinstein Duality):

$$W(\mathbb{P}_r, \mathbb{P}_\theta) = \sup_{f \text{s.t.} \|f\|_L < 1} \left(\mathbb{E}_{x \sim \mathbb{P}_r}[f(x)] - \mathbb{E}_{y \sim \mathbb{P}_\theta}[f(y)] \right),$$

where $\|f\|_L < 1 \Leftrightarrow \|f(x_1) - f(x_2)\| < \|x_1 - x_2\|$ i.e. f has to be 1-Lipschitz.

Kantorovich-Rubinstein Duality: Intuition (1)

When $W(\mathbb{P}_r, \mathbb{P}_\theta)$ is large, it is possible to find an f so that

$$\sup_{f \text{s.t.} \|f\|_L < 1} \left(\mathbb{E}_{x\sim\mathbb{P}_r}[f(x)] - \mathbb{E}_{y\sim\mathbb{P}_\theta}[f(y)]\right)$$

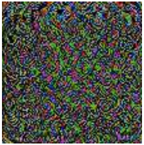
is large:

Other Adversarial Examples

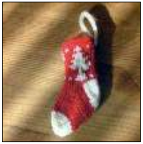
$$\min_{\mathbf{r}} \|\mathbf{r}\|_2 \text{ subject to } c(\mathbf{x}+\mathbf{r}) \neq c(\mathbf{x})$$

where

- $\mathbf{x}$ is an image and
- $c(.)$ is the class predicted by an already trained network.



$\mathbf{r}$



$\mathbf{x}+\mathbf{r}$

Predicted class: 'Indian elephant'

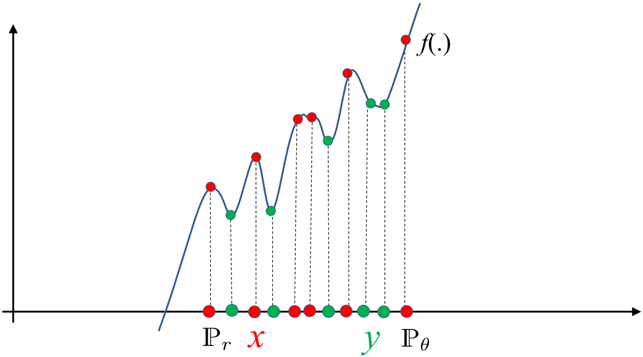
Seyed-Mohsen Moosavi-Dezfooli, Alhussein Fawzi, Pascal Frossard. *DeepFool: A Simple and Accurate Method to Fool Deep Neural Networks*. CVPR 2016.

Kantorovich-Rubinstein Duality: Intuition (2)

When $W(\mathbb{P}_r, \mathbb{P}_\theta)$ is small, it is not possible to find an f so that

$$\sup_{f \text{s.t.} \|f\|_L < 1} (\mathbb{E}_{x\sim\mathbb{P}_r}[f(x)] - \mathbb{E}_{y\sim\mathbb{P}_\theta}[f(y)])$$

is large under the constraint that f is Lipschitz:



So What?

$$W(\mathbb{P}_r, \mathbb{P}_\theta) = \sup_{f \text{s.t.} \|f\|_L < 1} \left(\mathbb{E}_{x \sim \mathbb{P}_r}[f(x)] - \mathbb{E}_{y \sim \mathbb{P}_\theta}[f(y)] \right),$$

where $\|f\|_L < 1 \Leftrightarrow \|f(x_1) - f(x_2)\| < \|x_1 - x_2\|$ i.e. f has to be 1-Lipschitz.

Why is this useful?

Using the Kantorovich-Rubinstein Duality to Compute $W(\mathbb{P}_r, \mathbb{P}_\theta)$

$$W(\mathbb{P}_r, \mathbb{P}_\theta) = \sup_{f \text{s.t.} \|f\|_L < 1} \left(\mathbb{E}_{x \sim \mathbb{P}_r}[f(x)] - \mathbb{E}_{y \sim \mathbb{P}_\theta}[f(y)] \right) ,$$

where $\|f\|_L < 1$.

If we have a parameterized family of functions f_w, where w are the parameters and $\|f_w\|_L < 1$ for any w, then we can estimate $W(\mathbb{P}_r, \mathbb{P}_\theta)$ using:

$$\begin{aligned} W(\mathbb{P}_r, \mathbb{P}_\theta) = & \max_w \left(\mathbb{E}_{x \sim \mathbb{P}_r}[f_w(x)] - \mathbb{E}_{y \sim \mathbb{P}_\theta}[f_w(y)] \right) \\ = & \max_w \left(\frac{1}{|\mathcal{T}|} \sum_{x \in \mathcal{T}} f_w(x) - \frac{1}{N_z} \sum_{z \sim \mathcal{N}(\mathbf{0};\mathbf{I})} f_w(g_\theta(z)) \right) \end{aligned}$$

Finally

Recall that we want:

$$\min_{\theta} W(\mathbb{P}_r, \mathbb{P}_\theta) \, .$$

We have:

$$W(\mathbb{P}_r, \mathbb{P}_\theta) = \max_{w} \left(\frac{1}{|\mathcal{T}|} \sum_{x \in \mathcal{T}} f_w(x) - \frac{1}{N_z} \sum_{z \sim \mathcal{N}(\mathbf{0};\mathbf{I})} f_w(g_\theta(z)) \right) \, .$$

Idea: Implement f_w with a deep network:

$$\min_{\theta} \max_{w} \left(\frac{1}{|\mathcal{T}|} \sum_{x \in \mathcal{T}} f_w(x) - \frac{1}{N_z} \sum_{z \sim \mathcal{N}(\mathbf{0};\mathbf{I})} f_w(g_\theta(z)) \right) \, .$$

(we will need to constrain f_w to be 1-Lipschitz)

Comparison with the Original Formulation

Original GANs:

$$\min_G \max_D \left(\frac{1}{|\mathcal{T}|} \sum_{\mathbf{x} \in \mathcal{T}} \log D(\mathbf{x}) + \frac{1}{N_{\mathbf{z}}} \sum_{\mathbf{z} \sim \mathcal{N}(\mathbf{0};\mathbf{I})} \log(1 - D(G(\mathbf{z}))) \right) ,$$

Wasserstein GANs:

$$\min_\theta \max_w \left(\frac{1}{|\mathcal{T}|} \sum_{x \in \mathcal{T}} f_w(x) - \frac{1}{N_z} \sum_{z \sim \mathcal{N}(\mathbf{0};\mathbf{I})} f_w(g_\theta(z)) \right) .$$

D is a classifier but f_w is not. It is usually called the critic.

Wassertein GAN Algorithm

```
for number of training iterations do
    for $k$ steps do
        Sample minibatch of $m$ samples $\{z^{(1)}, \ldots, z^{(m)}\}$
        Sample minibatch of $m$ samples $\{x^{(1)}, \ldots, x^{(m)}\}$
        Update parameters $w$ for the critic $f_w$ by stochastic gradient ascend.
        $w \leftarrow \mathrm{clip}(w, -c, +c)$ # $f_w$ should be Lipschitz
    end for
    Sample minibatch of $m$ samples $\{z^{(1)}, \ldots, z^{(m)}\}$
    Update parameters $\theta$ for the generator $g_\theta$ by stochastic gradient descend.
end for
```

Forcing f_w to be 1-Lipschitz

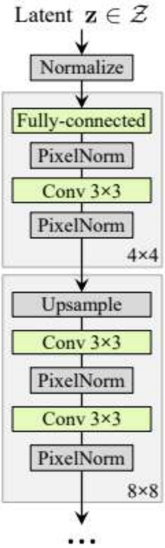
Original solution: $w \leftarrow \text{clip}(w, -c, +c)$

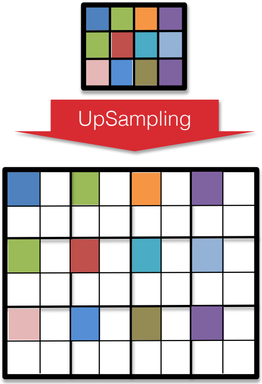
Better solution: Add a term that constrains the gradient of f_w wrt x.

$$\min_{\theta} \max_{w} \left(\frac{1}{|\mathcal{T}|} \sum_{x \in \mathcal{T}} f_w(x) \quad - \frac{1}{N_z} \sum_{z \sim \mathcal{N}(\mathbf{0};\mathbf{I})} f_w(g_\theta(z)) \right.$$
$$\left. - \lambda \frac{1}{|\mathcal{T}|} \sum_{x \in \mathcal{T}} (\|\nabla_x f_w(x)\|_2 - 1)^2 \right)$$

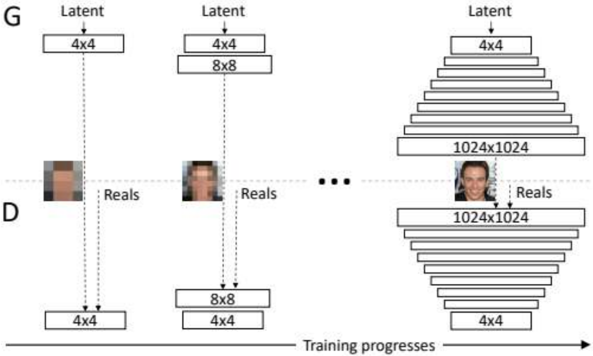
Gulrajani17.

Image Generators



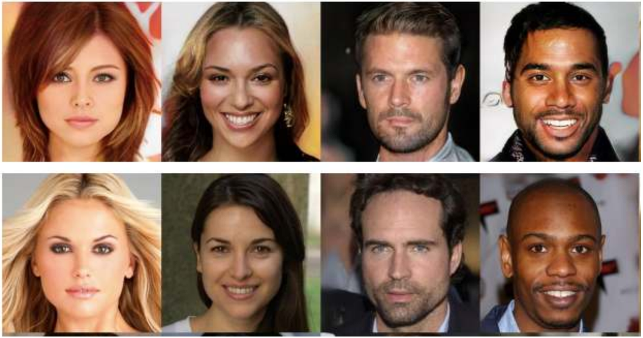


Progressive GANs



Karras18.

Progressive GANs: Results

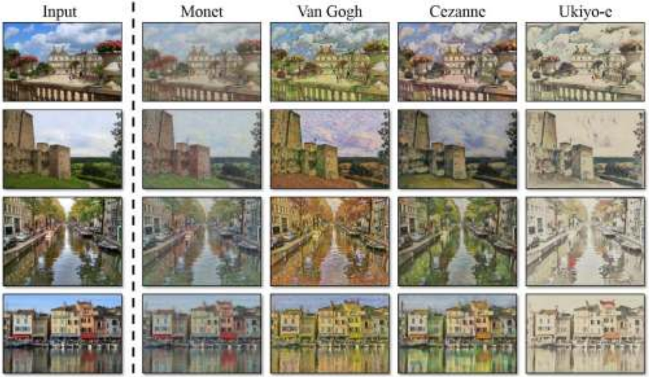


Generated Image

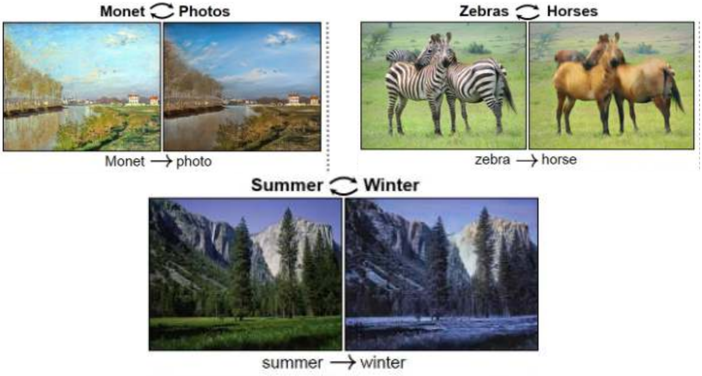
Nearest Neighbor in the training set

CycleGAN

How can we make sure we preserve the content of an input image?

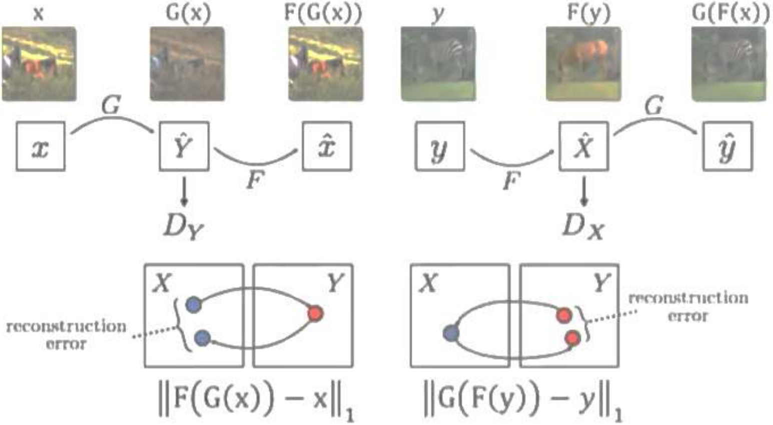


CycleGAN



Zhu17.

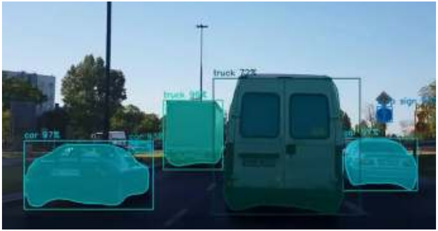
CycleGAN

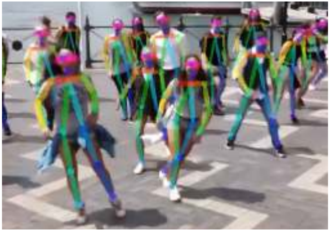


Zhu17.

Deep Learning applied to Computer Vision problems

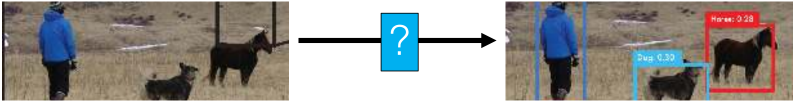
- Object Detection;
- Image Segmentation;
- Articulated Object Detection;
- etc.





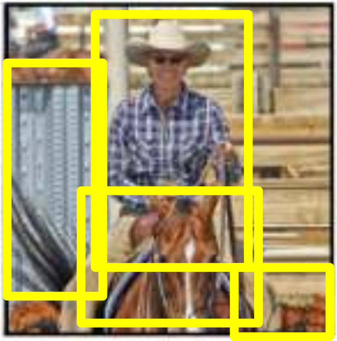
How can we formalize these problems into Deep Learning problems?

Object Detection and Localization: How?



R-CNN (1)

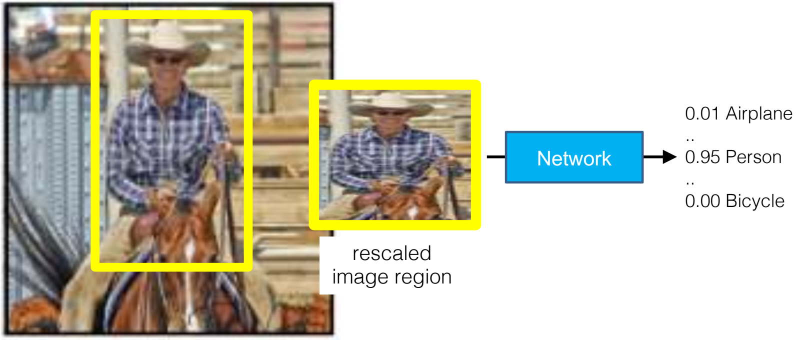
Input Image



Region Proposals
No learning (yet)

R.B. Girshick et al. “Rich Feature Hierarchies for Accurate Object Detection and Semantic Segmentation”. In: *Conference on Computer Vision and Pattern Recognition*. 2014, pp. 580–587.

R-CNN (2)



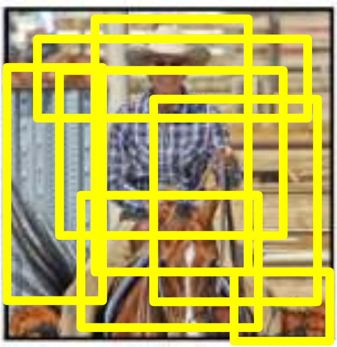
rescaled
image region

Network

0.01 Airplane
..
0.95 Person
..
0.00 Bicycle

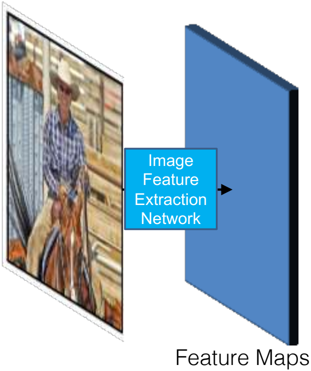
R-CNN (3)

Problem: In practice, many region proposals. R-CNN inefficient since image locations are processed many times.



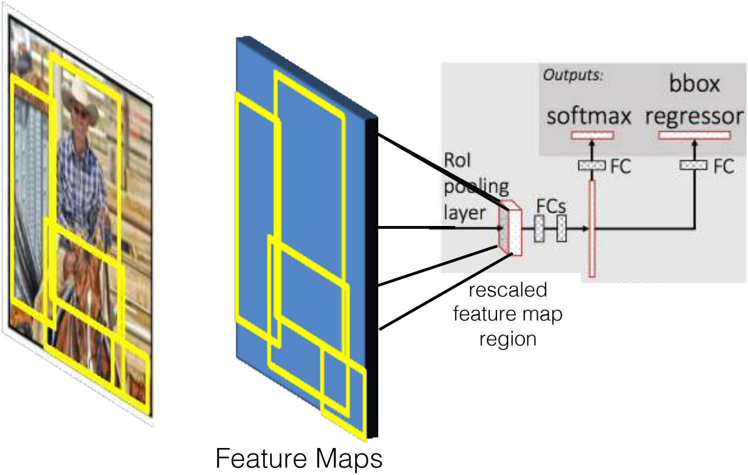
Fast R-CNN

Convolutions are applied only once to the image to extract image features: Much faster.

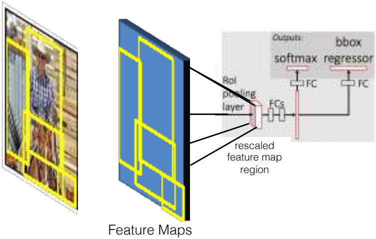


Ross B. Girshick. “Fast R-CNN”. In: *International Conference on Computer Vision*. 2015.

Fast R-CNN (2)


Outputs:
softmax
bbox
regressor
FC
FC
RoI
pooling
layer
FCs
rescaled
feature map
region
Feature Maps

Fast R-CNN: Loss Function (1)



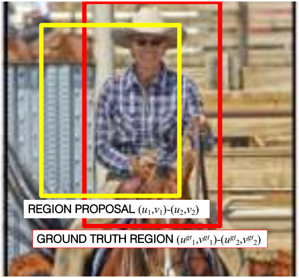
Loss function for 1 region:

$$\mathcal{L}(\Theta) = -\log p_c(f_{\mathsf{cl}}(\mathbf{x};\Theta)) + \lambda 1_{[c\geq 1]}\mathcal{L}_{\mathsf{bbox}}\,, \tag{15}$$

where:

- $\mathbf{x}$ is the rescaled region in the feature maps;
- c is the true class for $\mathbf{x}$. $c = 0$ corresponds to the background.
- $\mathcal{L}_{\mathsf{bbox}}$ is a loss term to refine the region bounding box (see next slide);
- λ is a weight.

Fast R-CNN: Loss Function (2)


REGION PROPOSAL (u_1,v_1)-(u_2,v_2)
GROUND TRUTH REGION $(u^{gt}{}_1,v^{gt}{}_1)$-$(u^{gt}{}_2,v^{gt}{}_2)$

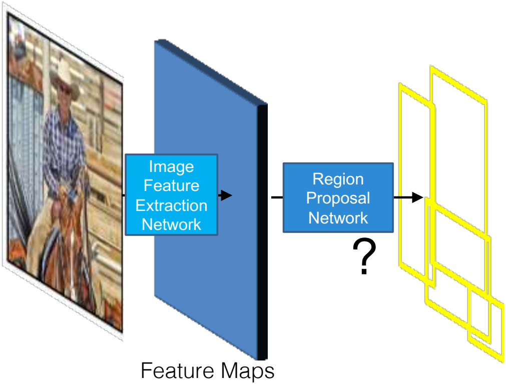
$\mathcal{L}_{\text{bbox}}$ is a loss term to refine the region bounding box:

$$\mathcal{L}_{\text{bbox}} = (u_1 + f_{\text{bbox}}(\mathbf{x};\Theta)[0] - u_1^{gt})^2 + (v_1 + f_{\text{bbox}}(\mathbf{x};\Theta)[1] - v_1^{gt})^2 + ..$$

where $(u_1, v_1) \times (u_2, v_2)$ are the coordinates of the region bounding box, and $(u_1^{gt}, v_1^{gt}) \times (u_2^{gt}, v_2^{gt})$ are the coordinates of the region bounding box.

Faster R-CNN

Learns to predict the region proposals:



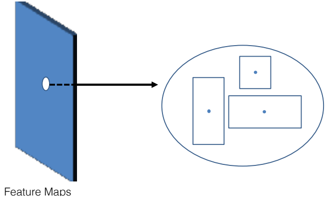
Challenges: the number of regions varies with the image, each region has a different size.

S. Ren et al. "Faster R-CNN: Towards Real-Time Object Detection with Region Proposal Networks". In: *Advances in Neural Information Processing Systems*. 2015.

Faster R-CNN: Region Proposal Network

For each 2D location in the feature maps: consider 3 “anchor boxes” and predict for each anchor box:

- If the anchor box overlaps with an object;
- An offset to adapt the anchor box.

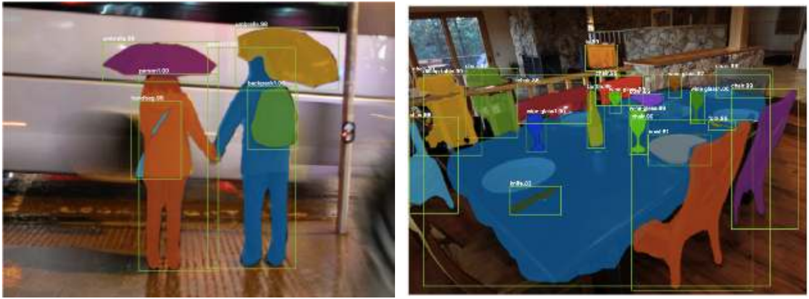


Loss function for 1 image $\mathbf{x}$ with $\mathcal{B} = \{B_i\}_i$ ground truth bounding boxes:

$$\mathcal{L}(\Theta_2) = -\sum_{A \in \mathcal{A}} \log p_{c(A,\mathcal{B})}(g(\mathbf{x};\Theta_2)[A]) + \lambda c(A,\mathcal{B})\mathcal{L}_{\mathsf{bbox}}\,, \qquad (16)$$

with $c(A,\mathcal{B}) = 1$ if Anchor box A overlaps with at least one bounding box B_i in $\mathcal{B}$, 0 otherwise.

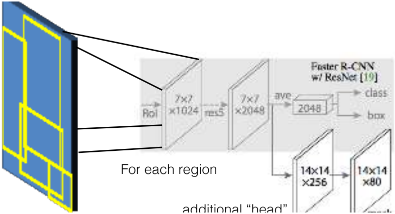
Mask-RCNN



Kaiming He et al. “Mask R-CNN”. In: *International Conference on Computer Vision*. 2017.

Mask-RCNN (2)

In addition to predicting the class and the “delta bounding box”, predict a binary mask for each possible class.



Loss function for one region:

$$\mathcal{L}(\Theta) = -\log p_c(f_{\text{cl}}(\mathbf{x};\Theta)) + \lambda 1_{[c\geq 1]}(\mathcal{L}_{\text{bbox}} + \lambda_2 \mathcal{L}_{\text{mask}}), \qquad (17)$$

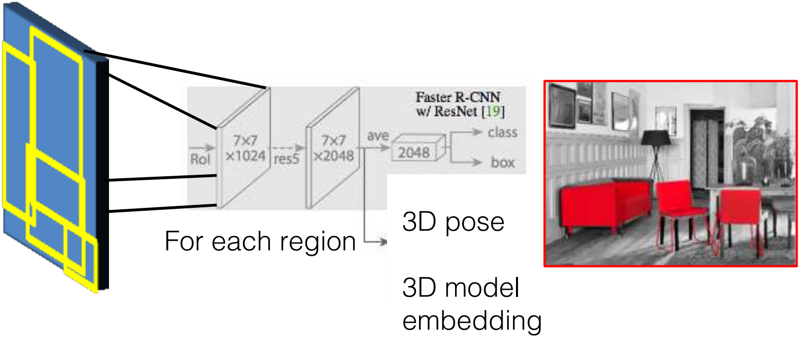
where:

- c is the true class for $\mathbf{x}$. $c = 0$ corresponds to the background.
- $\mathcal{L}_{\text{mask}}$ is a loss term to refine the region bounding box:

$$\mathcal{L}_{\text{mask}} = \|f_{\text{mask}}(\mathbf{x};\Theta)[c] - m\|^2$$

- m is the ground truth mask for the region.

Mask-RCNN: Extensions



For each region

3D pose

3D model embedding

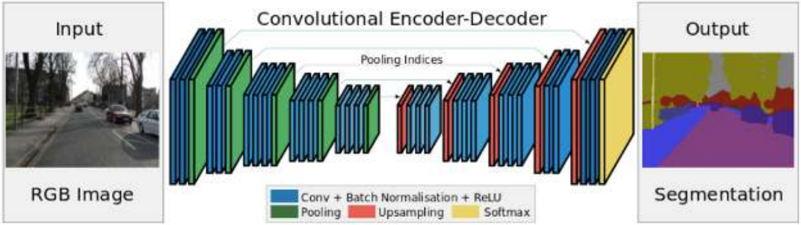
Alexander Grabner, Peter M. Roth, and Vincent Lepetit. "3D Pose Estimation and 3D Model Retrieval for Objects in the Wild". In: *CVPR*. 2018.

Deep Learning applied to Computer Vision problems

- Object Detection;
- Image Segmentation;
- Articulated Object Detection;
- etc.

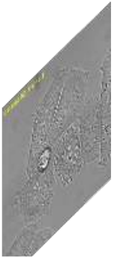
Image Segmentation

Typically relies on Encoder-Decoder architecture (also called Hour-Glass network):



V. Badrinarayanan, A. Kendall, and R. Cipolla. "Segnet: A Deep Convolutional Encoder-Decoder Architecture for Image Segmentation". In: *IEEE Transactions on Pattern Analysis and Machine Intelligence* (2015).

Image Segmentation with U-Net



Olaf Ronneberger, Philipp Fischer, and Thomas Brox. "U-Net: Convolutional Networks for Biomedical Image Segmentation". In: *MICCAI*. 2015.

Image Segmentation with U-Net

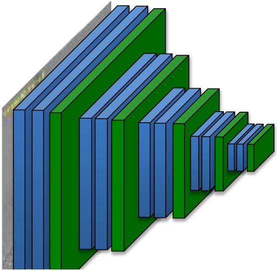
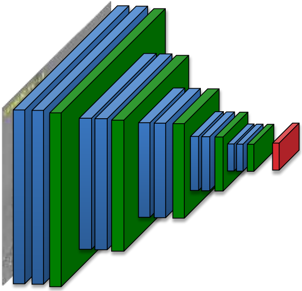


Image Segmentation with U-Net



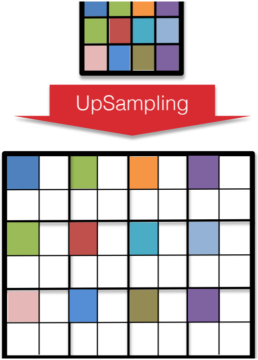


Image Segmentation with U-Net

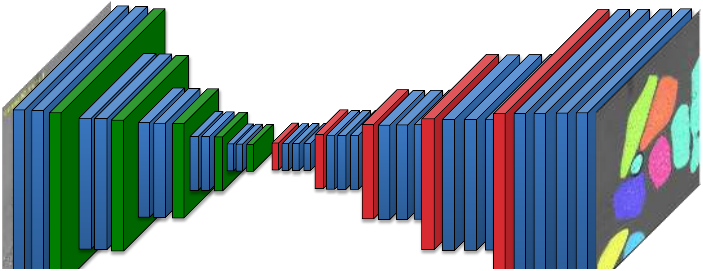
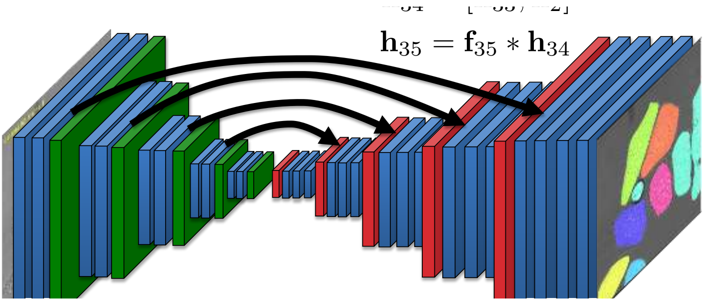


Image Segmentation with U-Net



$$\mathbf{h}_{35} = \mathbf{f}_{35} * \mathbf{h}_{34}$$

Loss function:

$$\mathcal{L}(\Theta) = \sum_{\mathbf{x}} \sum_{l} - \log p_{cl(\mathbf{x},l)}(f(\mathbf{x};\Theta)[l]) \tag{18}$$

Deep Learning applied to Computer Vision problems

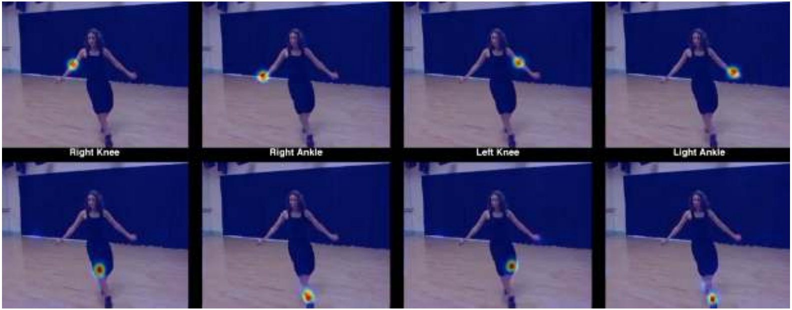
- Object Detection;
- Image Segmentation;
- Articulated Object Detection;
- etc.

Articulated Pose Prediction

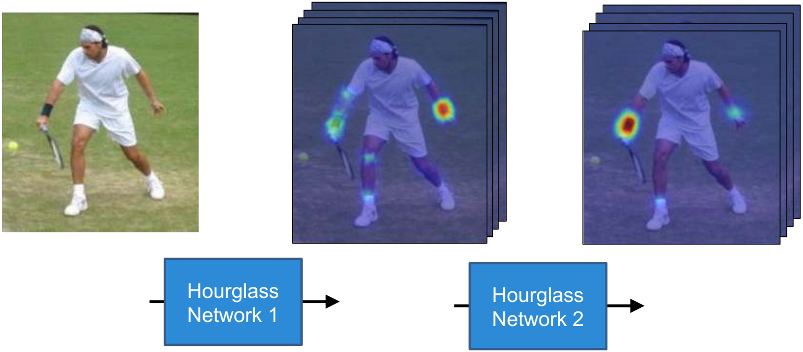
Z. Cao et al. “Realtime Multi-Person 2D Pose Estimation Using Part Affinity Fields”. In: *Conference on Computer Vision and Pattern Recognition*. 2017.

How can we formalize this problem?

Predicting Heat Maps

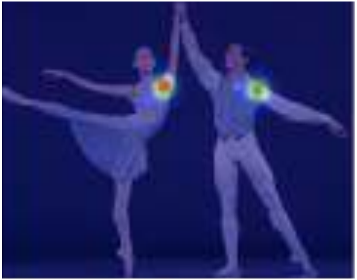


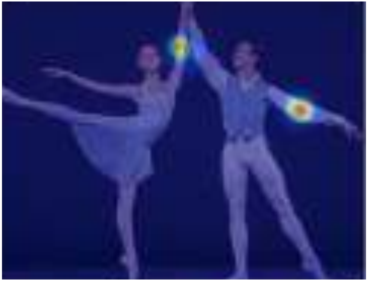
Convolutional Pose Machines

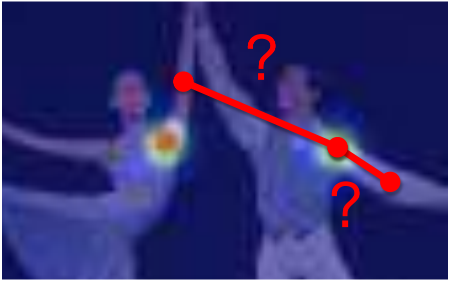


Shih-En Wei et al. "Convolutional Pose Machines". In: *Conference on Computer Vision and Pattern Recognition*. 2016.

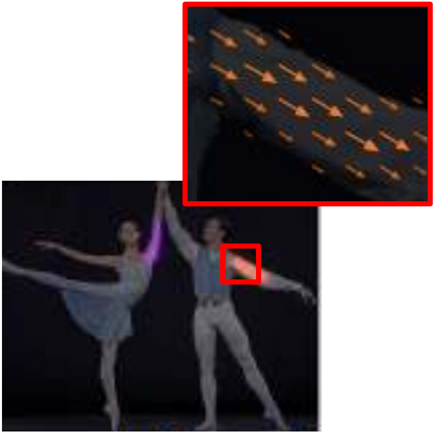
Multiple Articulated Objects

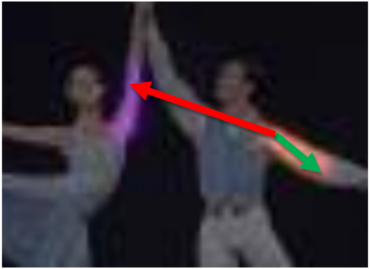






Multiple Articulated Objects





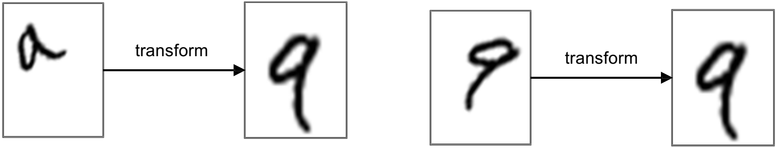
Z. Cao et al. “Realtime Multi-Person 2D Pose Estimation Using Part Affinity Fields”. In: *Conference on Computer Vision and Pattern Recognition*. 2017.

To Go Further

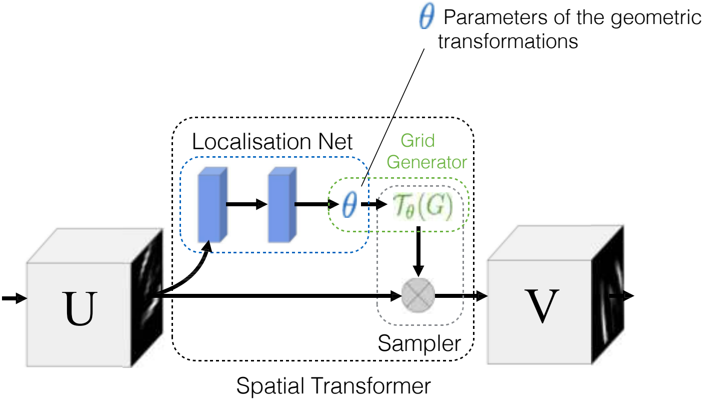
- Articulated Object Detection;
- Spatial Transformers;
- Dealing with Unstructured Input;
- Self-Learning;
- Differentiable Pipelines and End-to-End Learning.

Spatial Transformers

Computes and applies a spatial transformation on an image, from the image itself.

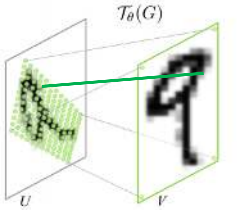


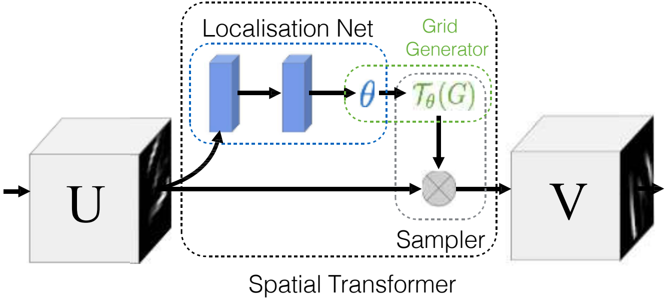
Differentiable Module



Example: Affine transformation

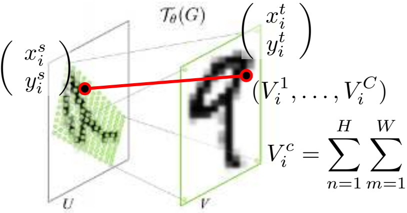
$$\begin{pmatrix} x_i^s \\ y_i^s \end{pmatrix} = \mathcal{T}_\theta(G_i) = \mathtt{A}_\theta \begin{pmatrix} x_i^t \\ y_i^t \\ 1 \end{pmatrix} = \begin{bmatrix} \theta_{11} & \theta_{12} & \theta_{13} \\ \theta_{21} & \theta_{22} & \theta_{23} \end{bmatrix} \begin{pmatrix} x_i^t \\ y_i^t \\ 1 \end{pmatrix}$$





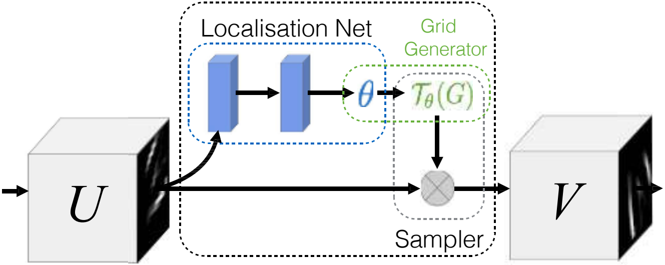
Example: Affine transformation

$$\begin{pmatrix} x_i^s \\ y_i^s \end{pmatrix} = \mathcal{T}_\theta(G_i) = \mathtt{A}_\theta \begin{pmatrix} x_i^t \\ y_i^t \\ 1 \end{pmatrix} = \begin{bmatrix} \theta_{11} & \theta_{12} & \theta_{13} \\ \theta_{21} & \theta_{22} & \theta_{23} \end{bmatrix} \begin{pmatrix} x_i^t \\ y_i^t \\ 1 \end{pmatrix}$$

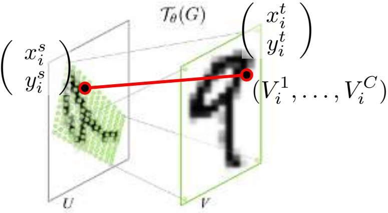

$\mathcal{T}_\theta(G)$
$\begin{pmatrix} x_i^t \\ y_i^t \end{pmatrix}$
$\begin{pmatrix} x_i^s \\ y_i^s \end{pmatrix}$
$(V_i^1, \ldots, V_i^C)$
U
V

$$V_i^c = \sum_{n=1}^{H} \sum_{m=1}^{W} U^c_{n \times m} k(x_i^s - m) k(y_i^s - n) \quad \forall i \in [1..HW] \; \forall c \in [1..C]$$

where *k* is a kernel.



Spatial Transformer


$\mathcal{T}_\theta(G)$
$\begin{pmatrix} x_i^t \\ y_i^t \end{pmatrix}$
$\begin{pmatrix} x_i^s \\ y_i^s \end{pmatrix}$
$(V_i^1, \ldots, V_i^C)$
U
V

$$V_i^c = \sum_{n=1}^{H} \sum_{m=1}^{W} U_{n \times m}^c k(x_i^s - m) k(y_i^s - n) \quad \forall i \in [1..HW] \, \forall c \in [1..C]$$

Bilinear sampling kernel:

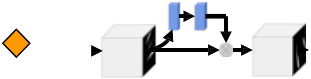
$$V_i^c = \sum_{n=1}^{H} \sum_{m=1}^{W} U_{n \times m}^c \max(0, 1 - |x_i^s - m|) \max(0, 1 - |y_i^s - n|)$$

$$\frac{\partial V_i^c}{\partial U_{n \times m}^c} = \sum_{n=1}^{H} \sum_{m=1}^{W} \max(0, 1 - |x_i^s - m|) \max(0, 1 - |y_i^s - n|)$$

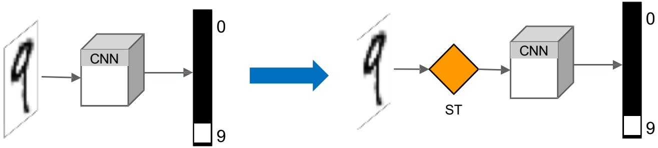
$$\frac{\partial V_i^c}{\partial x_i^s} = \sum_{n=1}^{H} \sum_{m=1}^{W} U_{n \times m}^c \max(0, 1 - |y_i^s - n|) \times \begin{cases} 0 & \text{if } |m - x_i^s| \geq 1 \\ 1 & \text{if } m \geq x_i^s \\ -1 & \text{if } m \leq x_i^s \end{cases}$$

Spatial Transformer Network

Spatial Transformers are differentiable, and so can be inserted at any point in a feed forward network and trained by back propogration

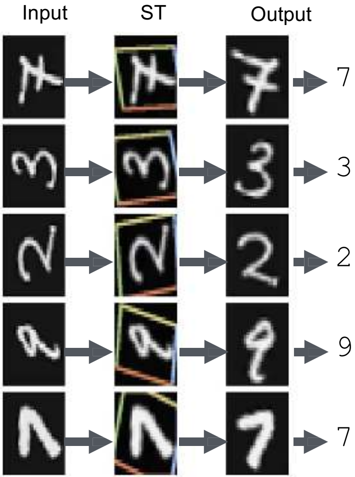


Example: digit classification, loss: cross-entropy for 10 way classification

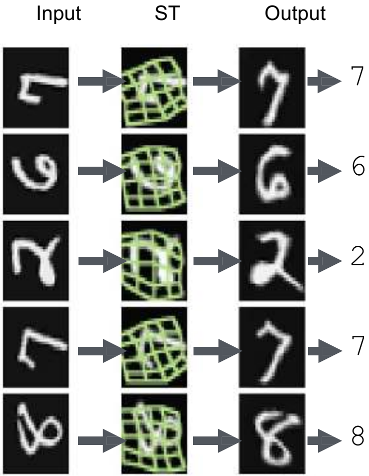


Rotated MNIST

ST-FCN Affine



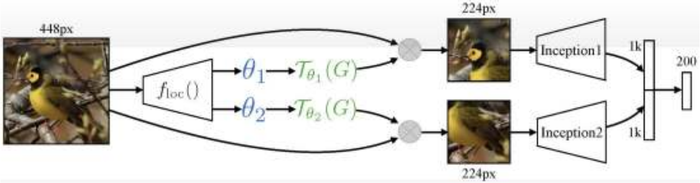
ST-FCN Thin Plate Spline



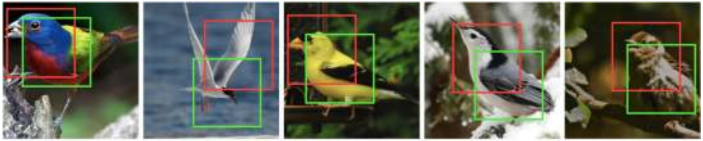
Fine Grained Visual Categorization

CUB-200-2011 birds dataset: 200 species of birds

Fine Grained Visual Categorization (2)



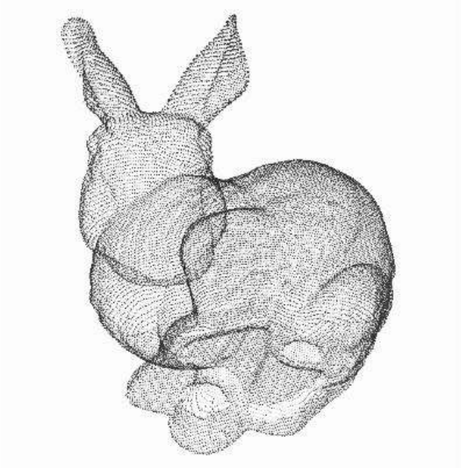
End-to-end learning: all the modules (f_{loc}, Inception classifiers are trained together)



To Go Further

- Articulated Object Detection;
- Spatial Transformers;
- Dealing with Unstructured Input;
- Self-Learning;
- Differentiable Pipelines and End-to-End Learning.

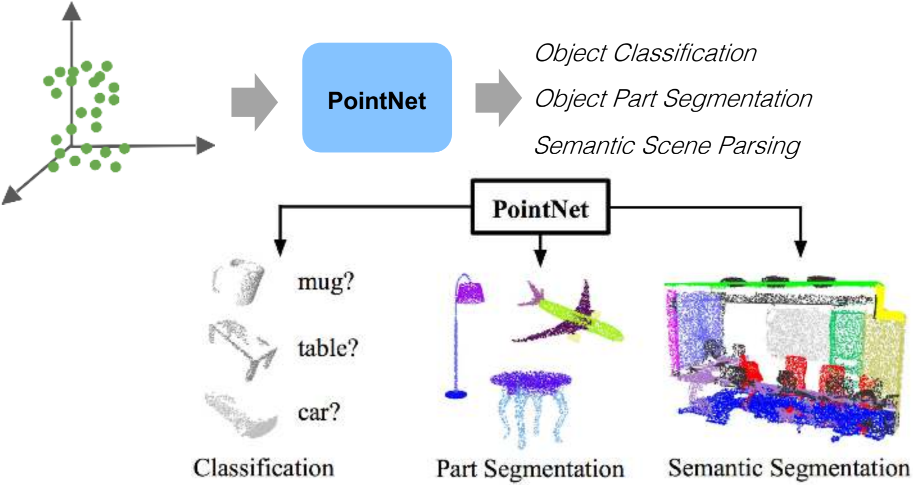
Dealing with Orderless Input



C.R. Qi et al. “PointNet: Deep Learning on Point Sets for 3D Classification and Segmentation”. In: *Conference on Computer Vision and Pattern Recognition*. 2017.

End-to-end learning for **scattered, unordered** point data

Unified framework for various tasks



PointNet

Object Classification

Object Part Segmentation

Semantic Scene Parsing

Challenges

- Unordered point set as input: Model needs to be invariant to $N!$ permutations;
- Also, model needs to be invariant under geometric transformations.

Permutation invariance:

$$f(x_1, x_2, \ldots, x_n) = f(x_{\sigma(1)}, x_{\sigma(2)}, \ldots, x_{\sigma(n)})\,. \tag{19}$$

Examples:

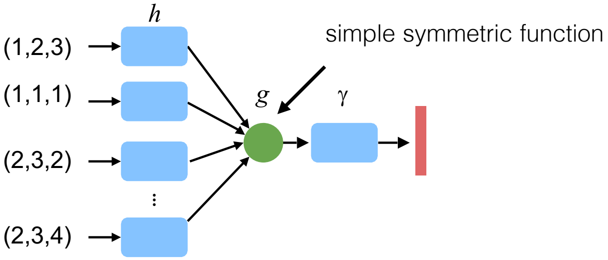
$$f(x_1, x_2, \ldots, x_n) = \max\{x_1, x_2, \ldots, x_n\} \tag{20}$$

$$f(x_1, x_2, \ldots, x_n) = x_1 + x_2 + \ldots + x_n \tag{21}$$

How can we construct a family of symmetric functions with neural networks?

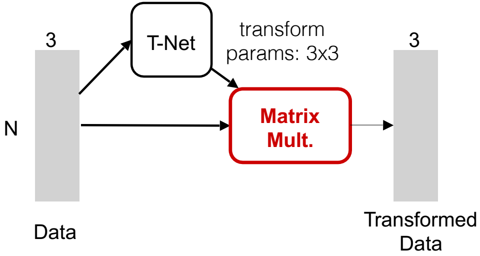
$$f(x_1, x_2, \ldots, x_n) = \gamma(g(h(x_1), h(x_2), \ldots, h(x_n))) \qquad (22)$$

is symmetric if g is symmetric.



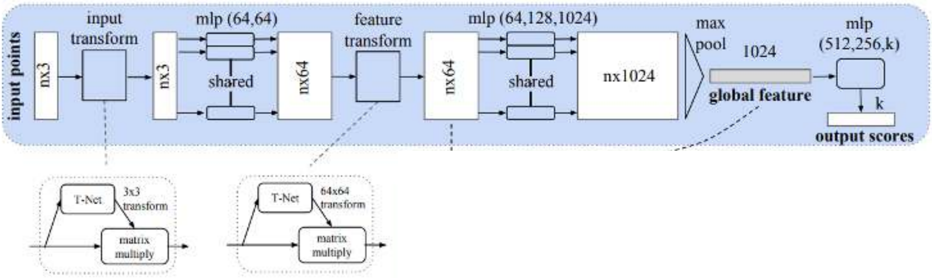
Use neural networks for h, γ, and max-pooling for g.

Invariance to Geometric Transformation



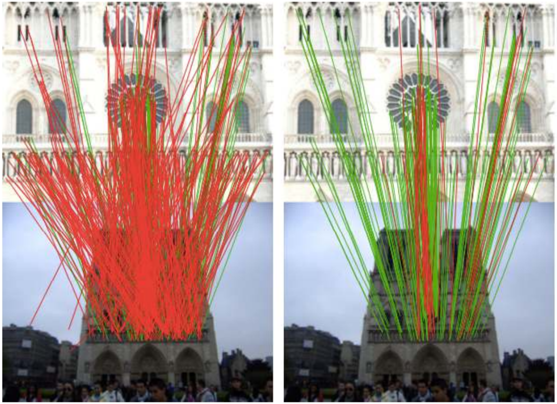
Full Network

Classification Network



n: number of points;
k: number of possible objects;
m: number of possible segments.

Application to Outlier Detection



(a) RANSAC (b) Our approach

K. M. Yi et al. "Learning to Find Good Correspondences". In: *Conference on Computer Vision and Pattern Recognition*. 2018.

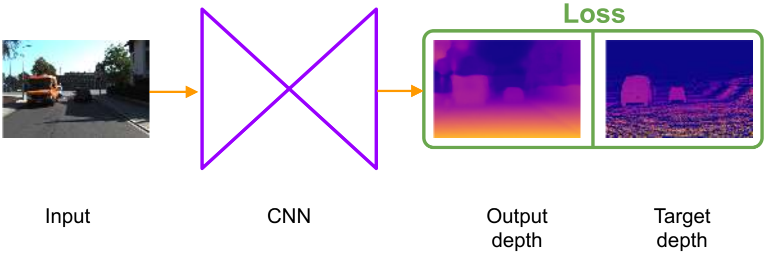
To Go Further

- Articulated Object Detection;
- Spatial Transformers;
- Dealing with Unstructured Input;
- Self-Learning;
- Differentiable Pipelines and End-to-End Learning.

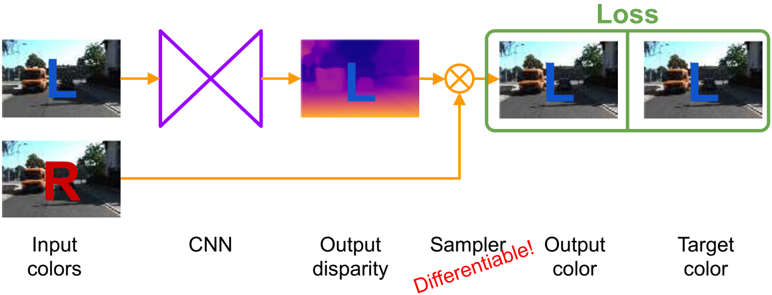
Self-Learning for Deep Monocular Depth Estimation

C. Godard, O. Mac Aodha, and G.J. Brostow. "Unsupervised Monocular Depth Estimation with Left-Right Consistency". In: *Conference on Computer Vision and Pattern Recognition*. 2017.

Supervised Depth Prediction



Unsupervised Depth Prediction



To Go Further

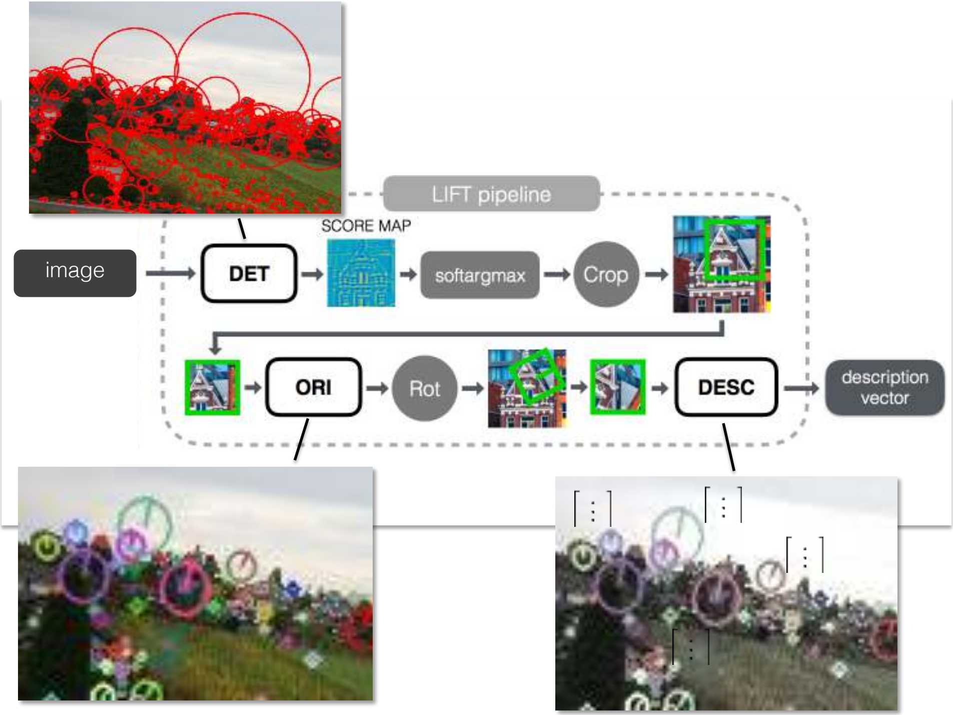
- Articulated Object Detection;
- Spatial Transformers;
- Dealing with Unstructured Input;
- Self-Learning;
- Differentiable Pipelines and End-to-End Learning.

Differentiable, End-to-End Computer Vision Pipelines

We can even train complex pipelines made of multiple networks so that the networks work well together.

For that, we need to formalize the problem with a single loss function that is differentiable. The networks are then trained jointly, "end-to-end" from the input to the final output.

Example:
K. M. Yi et al. "LIFT: Learned Invariant Feature Transform". In: *European Conference on Computer Vision*. 2016.



LIFT pipeline

image

DET

SCORE MAP

softargmax

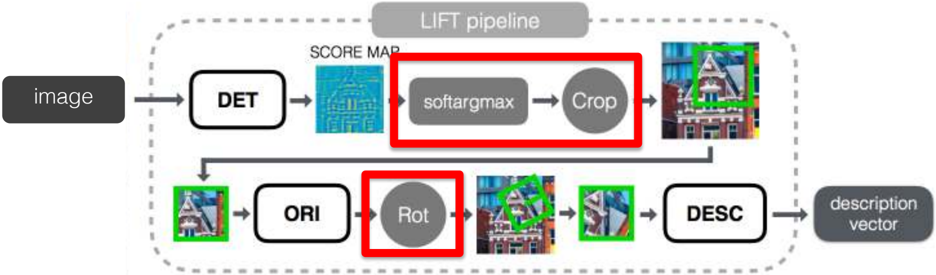
Crop

ORI

Rot

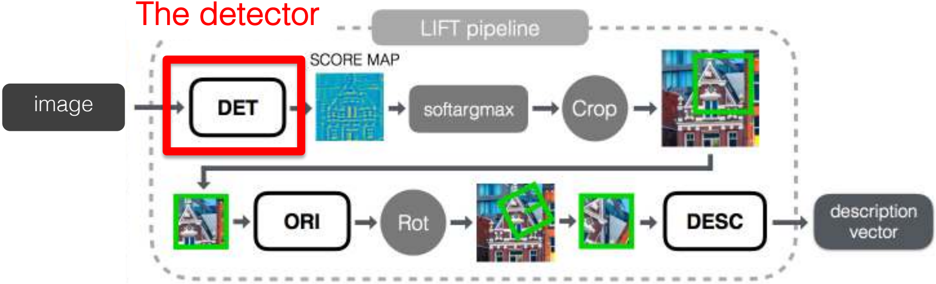
DESC

description vector

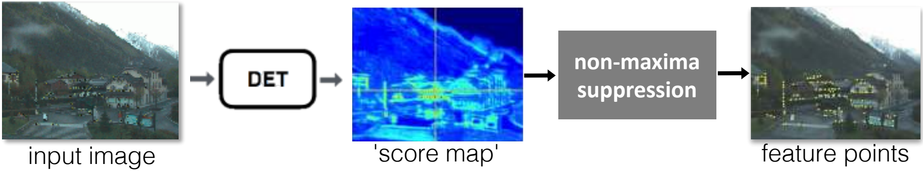


“Differentiable glue”

The detector



How the Detector is Used at Run-Time



input image

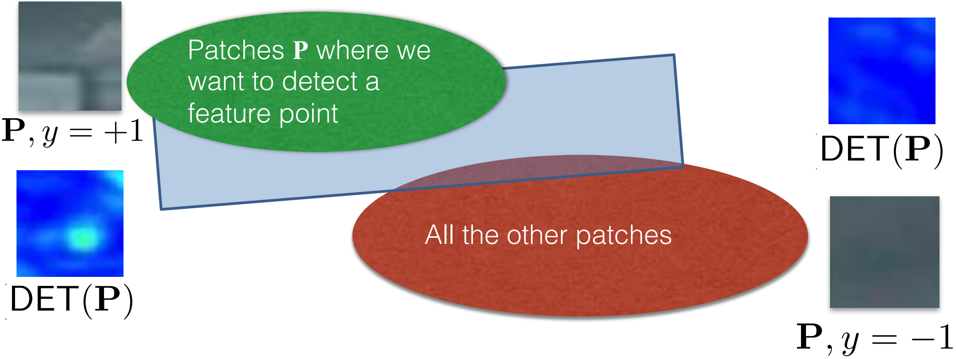
DET

'score map'

non-maxima suppression

feature points

Cost Function (1)



Patches $\mathbf{P}$ where we want to detect a feature point

All the other patches

$\mathbf{P}, y = +1$

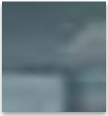
$\text{DET}(\mathbf{P})$

$\text{DET}(\mathbf{P})$

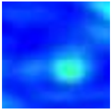
$\mathbf{P}, y = -1$

$$\mathcal{L}_{\text{class}}(\mathbf{P}) = \max(0, 1 - y \max(\text{DET}(\mathbf{P})))^2, y \in \{-1, +1\}$$

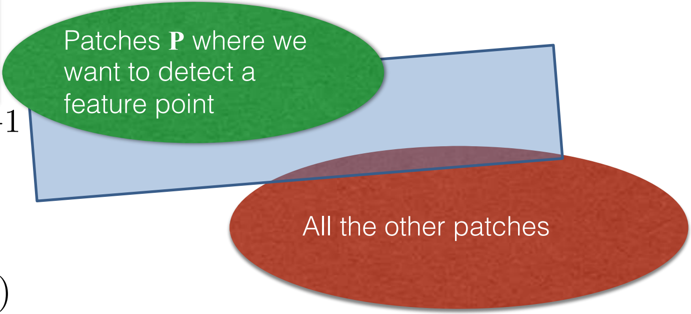
Cost Function (1)



$\mathbf{P}, y = +1$

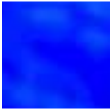


$\text{DET}(\mathbf{P})$

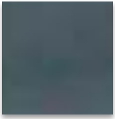


Patches $\mathbf{P}$ where we want to detect a feature point

All the other patches



$\text{DET}(\mathbf{P})$

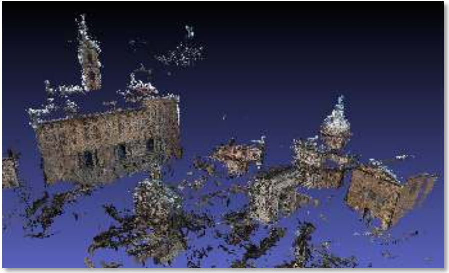


$\mathbf{P}, y = -1$

$$\mathcal{L}_{\text{class}}(\mathbf{P}) = \max(0, 1 - y\ \underline{\text{softmax}}(\text{DET}(\mathbf{P})))^2, y \in \{-1, +1\}$$

Training with SfM Keypoints

Piccadilly (pic)

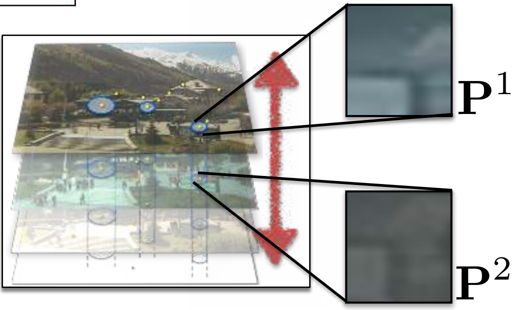


Roman Forum (rf)

- We need variability (illumination, perspective, etc). We build SfM reconstructions from **photo-tourism sets.**

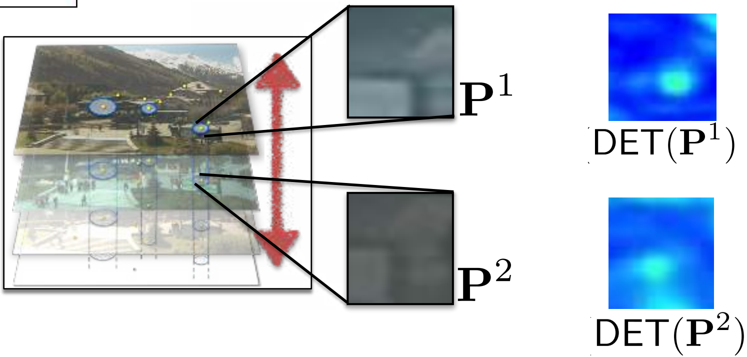
- We keep only **points with SfM correspondences** as positive examples, that is, we **learn to find repeatable points.**

Cost Function (2)

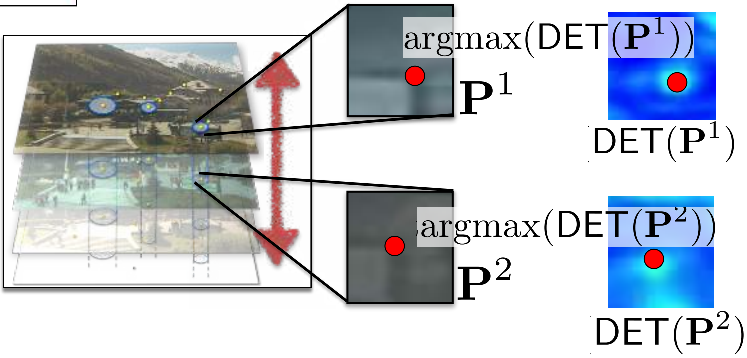


DET

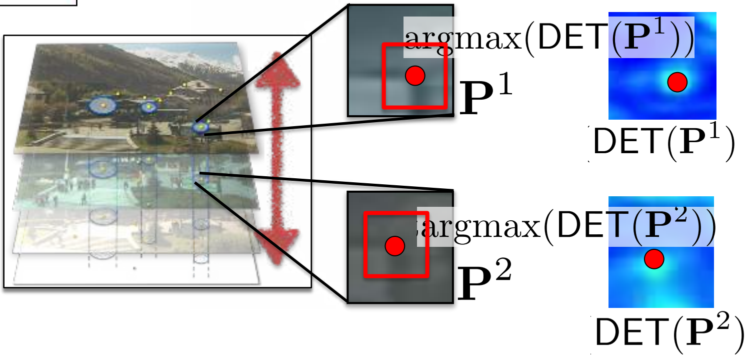
Cost Function (2)



Cost Function (2)


$\mathrm{argmax}(\mathsf{DET}(\mathbf{P}^1))$
$\mathbf{P}^1$
$\mathsf{DET}(\mathbf{P}^1)$
$\mathrm{argmax}(\mathsf{DET}(\mathbf{P}^2))$
$\mathbf{P}^2$
$\mathsf{DET}(\mathbf{P}^2)$

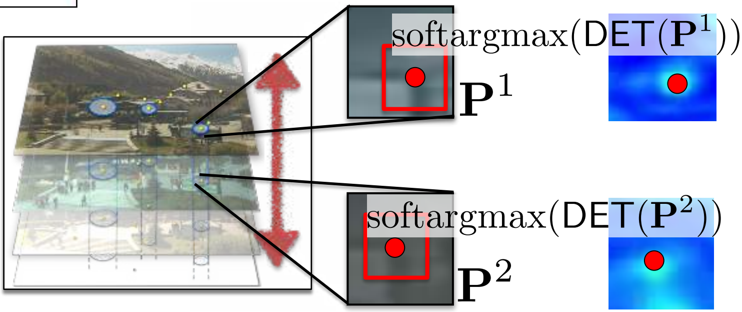
Cost Function (2)


$\text{argmax}(\text{DET}(\mathbf{P}^1))$
$\mathbf{P}^1$
$\text{DET}(\mathbf{P}^1)$
$\text{argmax}(\text{DET}(\mathbf{P}^2))$
$\mathbf{P}^2$
$\text{DET}(\mathbf{P}^2)$

$$\mathcal{L}_{\text{pair}}(\mathbf{P}^1, \mathbf{P}^2) = \| \quad \text{DESC}(\text{Crop}(\mathbf{P}^1, \text{argmax}(\text{DET}(\mathbf{P}^1)))) - \text{DESC}(\text{Crop}(\mathbf{P}^2, \text{argmax}(\text{DET}(\mathbf{P}^2)))) \quad \|^2$$

DET

Cost Function (2)



$\mathrm{softargmax}(\mathsf{DET}(\mathbf{P}^1))$
$\mathbf{P}^1$
$\mathrm{softargmax}(\mathsf{DET}(\mathbf{P}^2))$
$\mathbf{P}^2$

$$\mathcal{L}_{\mathrm{pair}}(\mathbf{P}^1, \mathbf{P}^2) = \| \quad \mathsf{DESC}(\mathrm{Crop}(\mathbf{P}^1, \underline{\mathrm{softargmax}}(\mathsf{DET}(\mathbf{P}^1)))) - \mathsf{DESC}(\mathrm{Crop}(\mathbf{P}^2, \underline{\mathrm{softargmax}}(\mathsf{DET}(\mathbf{P}^2)))) \quad \|^2$$

$$\mathrm{softargmax}(\mathbf{S}) = \frac{\sum_{\mathbf{x}} \exp(\beta \mathbf{S}(\mathbf{x}))\mathbf{x}}{\sum_{\mathbf{x}} \exp(\beta \mathbf{S}(\mathbf{x}))}$$

so far

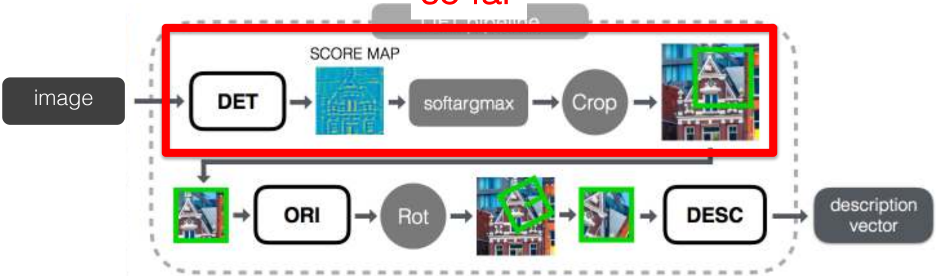
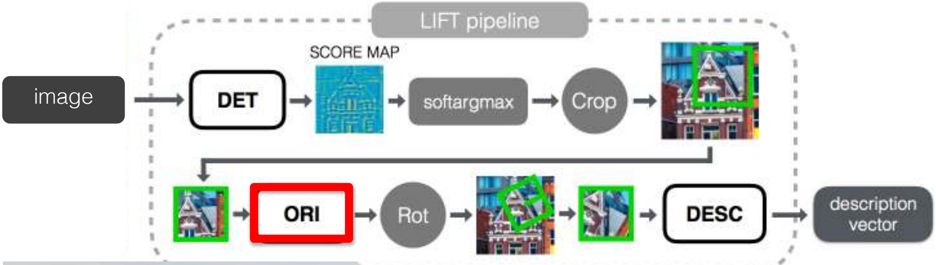


image
DET
SCORE MAP
softargmax
Crop
ORI
Rot
DESC
description vector



LIFT pipeline

image

DET

SCORE MAP

softargmax

Crop

ORI

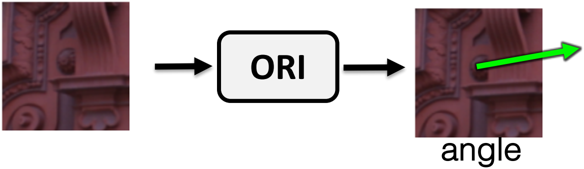
Rot

DESC

description vector

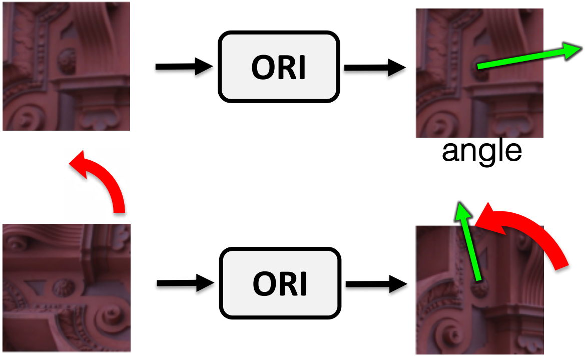
Learning Orientations Implicitly

We want the orientation estimator to provide **consistent** results, regardless of imaging changes

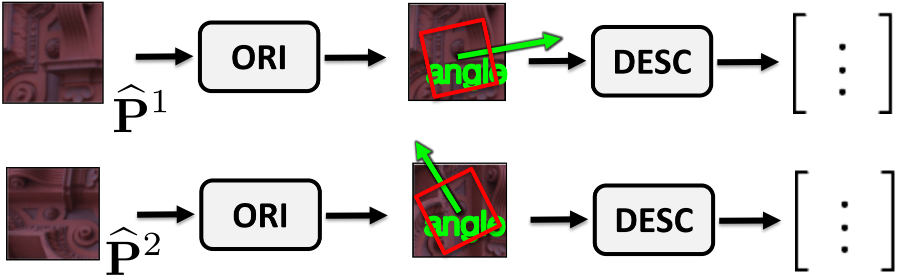


Learning Orientations Implicitly

We want the orientation estimator to provide **consistent** results, regardless of imaging changes

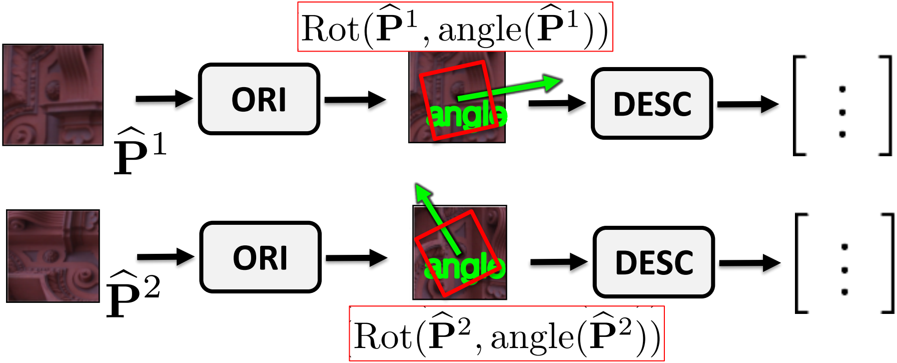


Learning Orientations Implicitly: Siamese Network



Learning Orientations Implicitly: Siamese Network

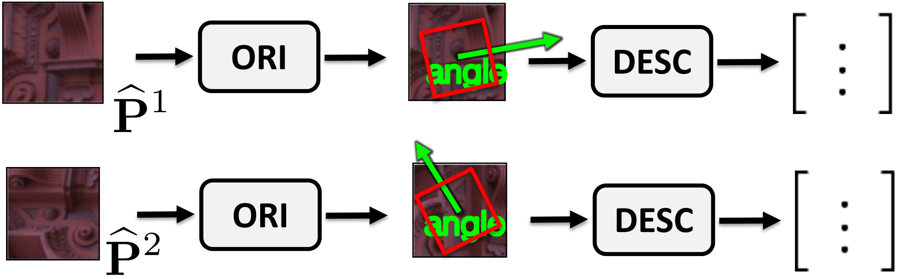
$$\mathcal{L}_{\text{pair}}(\widehat{\mathbf{P}}^1, \widehat{\mathbf{P}}^2) = \| \quad \text{DESC}(\text{Rot}(\widehat{\mathbf{P}}^1, \text{angle}(\widehat{\mathbf{P}}^1))) - \text{DESC}(\text{Rot}(\widehat{\mathbf{P}}^2, \text{angle}(\widehat{\mathbf{P}}^2))) \quad \|^2$$


$\text{Rot}(\widehat{\mathbf{P}}^1, \text{angle}(\widehat{\mathbf{P}}^1))$
$\widehat{\mathbf{P}}^1$
ORI
angle
DESC
$\widehat{\mathbf{P}}^2$
ORI
angle
DESC
$\text{Rot}(\widehat{\mathbf{P}}^2, \text{angle}(\widehat{\mathbf{P}}^2))$

Learning Orientations Implicitly: Siamese Network

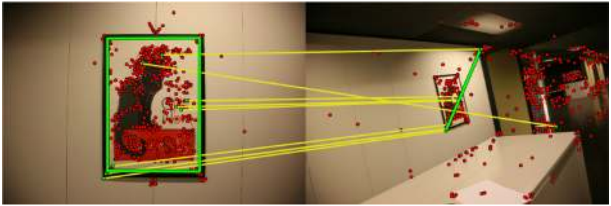
$$\mathcal{L}_{\text{pair}}(\widehat{\mathbf{P}}^1, \widehat{\mathbf{P}}^2) = \| \quad \mathsf{DESC}(\mathrm{Rot}(\widehat{\mathbf{P}}^1, \mathrm{angle}(\widehat{\mathbf{P}}^1))) - \mathsf{DESC}(\mathrm{Rot}(\widehat{\mathbf{P}}^2, \mathrm{angle}(\widehat{\mathbf{P}}^2))) \quad \|^2$$

$$\text{with } \mathrm{angle}(\widehat{\mathbf{P}}) = \mathrm{arctan2}(\mathsf{ORI}(\widehat{\mathbf{P}})^{[1]}, \mathsf{ORI}(\widehat{\mathbf{P}})^{[2]})$$

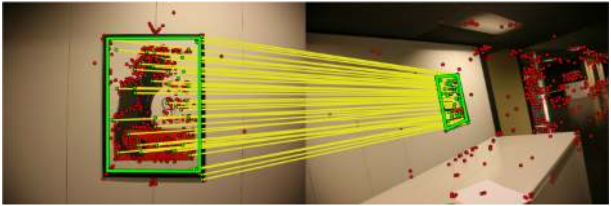


ORI

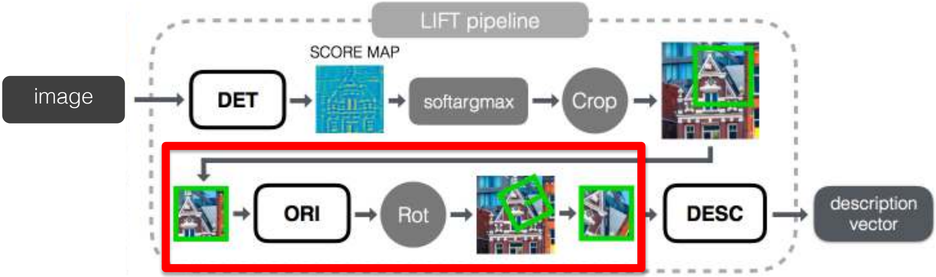
Learned Orientations

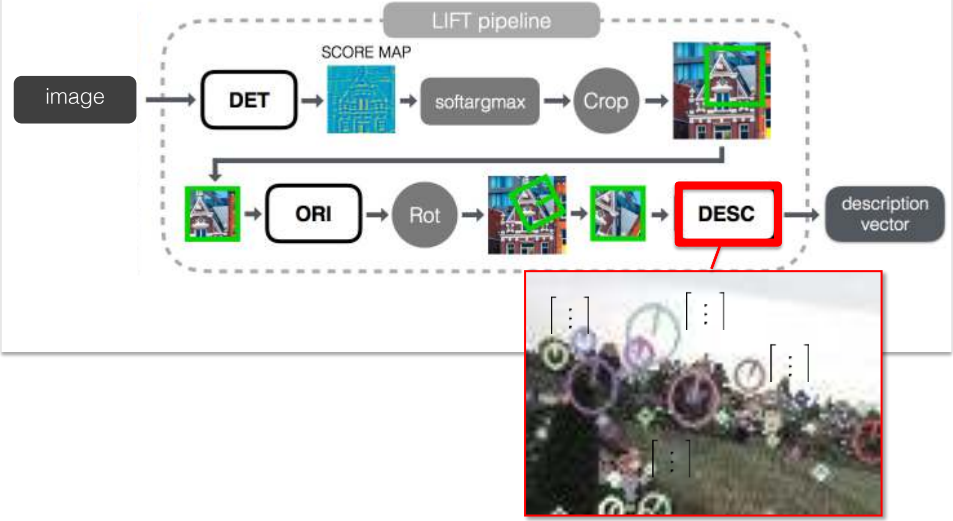


Dominant Gradient Orientations



Our Learned Orientations


LIFT pipeline
image
DET
SCORE MAP
softargmax
Crop
ORI
Rot
DESC
description vector



image

LIFT pipeline

DET

SCORE MAP

softargmax

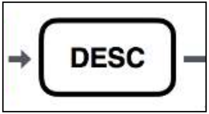
Crop

ORI

Rot

DESC

description vector



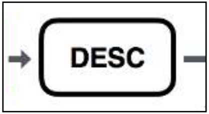
Learning the Descriptor

- Positive pairs:

$\widehat{\mathbf{P}^1}$

$\widehat{\mathbf{P}^2}$

$$\mathcal{L}_{\text{pos}}(\widehat{\mathbf{P}^1}, \widehat{\mathbf{P}^2}) = \|\text{DESC}(\widehat{\mathbf{P}^1}) - \text{DESC}(\widehat{\mathbf{P}^2})\|^2$$



Learning the Descriptor

- Positive pairs:

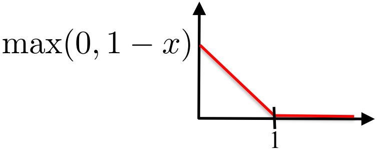
$\widehat{\mathbf{P}}^1$

$\widehat{\mathbf{P}}^2$

$$\mathcal{L}_{\text{pos}}(\widehat{\mathbf{P}}^1, \widehat{\mathbf{P}}^2) = \|\text{DESC}(\widehat{\mathbf{P}}^1) - \text{DESC}(\widehat{\mathbf{P}}^2)\|^2$$

- Negative pairs:

$$\mathcal{L}_{\text{neg}}(\widehat{\mathbf{P}}^1, \widehat{\mathbf{P}}^3) = \max(0, 1 - \|\text{DESC}(\widehat{\mathbf{P}}^1) - \text{DESC}(\widehat{\mathbf{P}}^3)\|^2)$$



$\widehat{\mathbf{P}}^1$

$\widehat{\mathbf{P}}^3$

Hard example mining is very important for training

A Single, Global Cost Function

$$\min_{\{\mathsf{DET},\mathsf{ORI},\mathsf{DESC}\}} \sum_{\{(\mathbf{P},y)\}} \max(0, 1 - y\ \mathrm{softmax}(\mathsf{DET}(\mathbf{P})))^2 +$$

$$\sum_{(\mathbf{P}_1,\mathbf{P}_2)} \|\mathsf{DESC}\left(G(\mathbf{P}^1, \mathrm{softargmax}(\mathsf{DET}(\mathbf{P}^1)))\right) - \mathsf{DESC}\left(G(\mathbf{P}^2, \mathrm{softargmax}(\mathsf{DET}(\mathbf{P}^2)))\right)\|^2 +$$

$$\sum_{(\mathbf{P}_1,\mathbf{P}_3)} \max(0, 1 - \|\mathsf{DESC}\left(G(\mathbf{P}^1, \mathrm{softargmax}(\mathsf{DET}(\mathbf{P}^1)))\right) - \mathsf{DESC}\left(G(\mathbf{P}^3, \mathrm{softargmax}(\mathsf{DET}(\mathbf{P}^3)))\right)\|^2)$$

$$G(\mathbf{P}, \mathbf{x}) = \mathrm{Rot}\left(\mathbf{P}, \mathbf{x}, \mathrm{angle}_{\mathsf{ORI}}(\mathrm{Crop}(\mathbf{P}, \mathbf{x}))\right)$$