Similarity learning

metric embedding of complex data

Siniša Šegvić
UniZg-FER

Plan

- motivation for similarity learing
- siamese models
- triplet loss
- implementation details and evaluation
- applications: stereoscopic vision, self-supervision

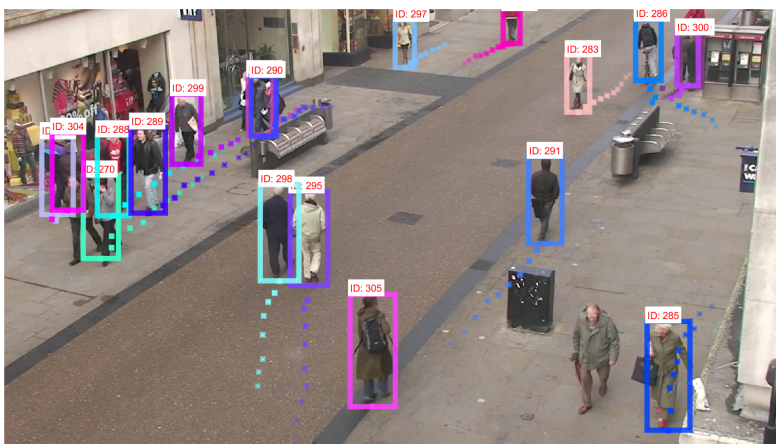
Introduction: limits of standard classification

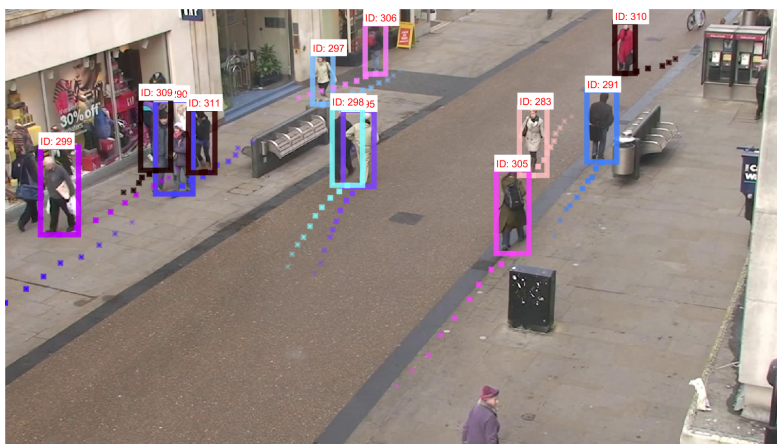
Classification models may become impractical when:

- there are no defined classes, or the classes are unknown
- the number of classes is too large

Application examples:

- stereoscopic correspondence
- self-supervised learing
- tracking, associative search biometric verification





[bicanic19fusion]

Introduction: limits of standard classification

The presented applications can be conveniently addressed by leveraging similarity between examples

Left pair: different. Right pair: similar

[chopra05cvpr]

The idea: embed the data into the appropriate vector space

- standard metrics (L2, ...) model the similarity between the data
- short name: metric embeddings

Introduction : metric embeddings

Before going forward, we must ask ourselves:

- why wouldn't we just measure the the similarity in the input space?

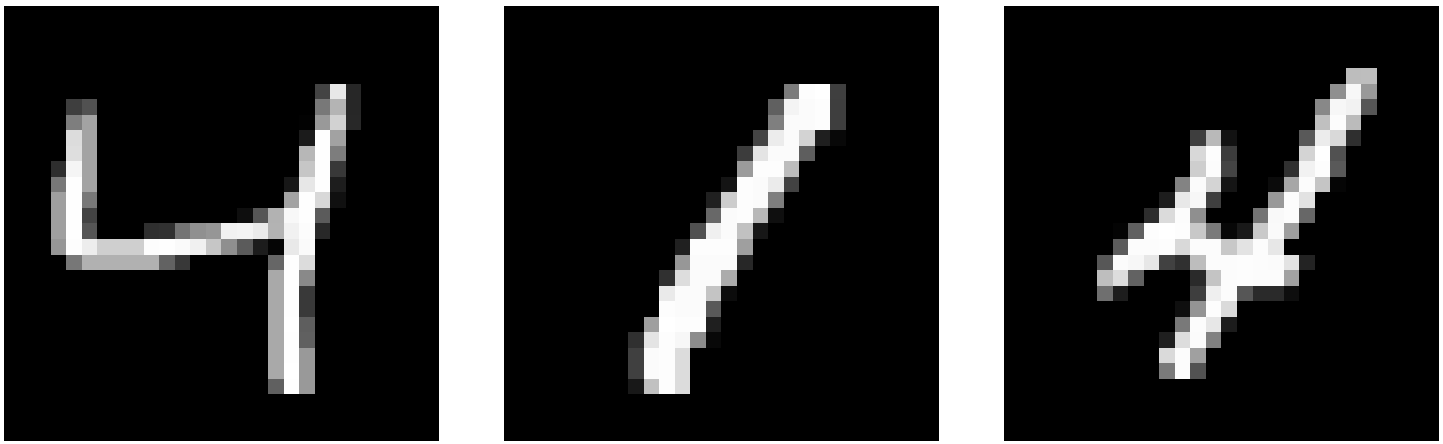
Answers:

- because distances in high dimensional vector spaces do not make sense (the curse of dimensionality)
- because vector representations of complex data is usually not appropriate for comparing different samples

Introduction: metric embeddings - an example

We look at distances between digits in the MNIST dataset

- L2 distance between the first and second digit: 121.4
- L2 distance between the first and third digit: 133.2
- L2 distance between the second and third digit: 114.9



[lecun98pieee]

Conclusion: there is no strong connection between the distance in the original space and data similarity

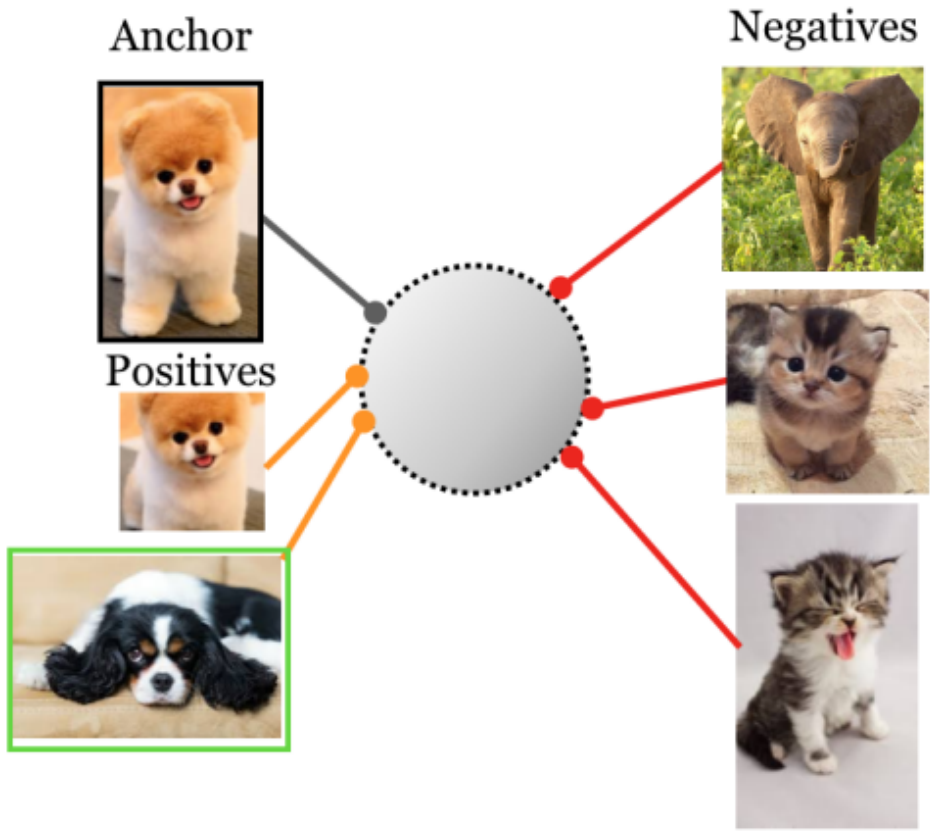
Introduction: metric embeddings - an example

```
import torch
import torchvision
mnist = torchvision.datasets.MNIST('data', download=True)
print(mnist.targets[:10])
# tensor([5, 0, 4, 1, 9, 2, 1, 3, 1, 4])
print(torch.sqrt(torch.sum((mnist.data[2]-mnist.data[3])**2,
      dtype=torch.float)))
# tensor(121.4)
print(torch.sqrt(torch.sum((mnist.data[2]-mnist.data[9])**2,
      dtype=torch.float)))
# tensor(133.2)
print(torch.sqrt(torch.sum((mnist.data[3]-mnist.data[9])**2,
      dtype=torch.float)))
# tensor(114.9)
import cv2
cv2.imwrite('m2.png', mnist.data[2].numpy())
cv2.imwrite('m3.png', mnist.data[3].numpy())
cv2.imwrite('m9.png', mnist.data[9].numpy())
```

Introduction: metric embeddings - the goal

Embed the data into a (relatively) low-dimensional space where standard metrics may model the similarity between the data

If we normalize the representations (e.g. by placing them inside the hypersphere), we may use scalar product to measure the similarity



[khosla20neurips]

Introduction : metric embeddings - advantages

We may associate data even when some of the classes are unknown during training

It is more difficult for the model to overfit:

- classification: O(N) learning samples
- similarity: $O(N^2)$, $O(N^3)$ or $O(N^B)$ learning samples.

Useful representations may also be trained in the self-supervised context where we expect the sample to be similar to its own perturbation, and different from other examples; e.g. SimCLR [chen20icml].

Metric embeddings may sometimes help with standard supervised tasks [khosla20neurips].

METRICS : TERMS

We consider the set X and the mapping $d : X \times X \to R$.

We refer to d as a metric, while (X, d) represent the metric space iff :

1. $d(a, b) \geq 0 \quad \forall a, b \in X$ (positivity),
2. $d(a, b) = 0 \iff a = b \quad \forall a, b \in X$ (zero-rule),
3. $d(a, b) = d(b, a) \quad \forall a, b \in X$ (symmetry),
4. $d(a, b) \leq d(a, c) + d(c, b) \quad \forall a, b, c \in X$ (triangle inequality).

This definition fits well with the concept of data similarity.

Some of these axioms are redundant, e.g. positivity and symmetry are a direct results of triangle inequality and the zero-rule.

In practice we usually learn a pseudo-metric that relaxes the zero-rule

- $d(a, b) \neq 0$ is hard to ensure $\forall a \neq b$, so we only require $d(a, a) = 0$

METRICS : STANDARD CHOICES

Euclidian metrics:

$$d_E(a,b) = \sqrt{(a-b)^\top (a-b)} \sim (a-b)^\top (a-b)$$

If the data is normalized, the scalar product is equal to the cosine similarity and results in the same ranking as a Euclidian metric:

$$\begin{aligned} d_E(a,b) &\sim (a-b)^\top (a-b) \\ &\sim a^\top a - 2 \cdot a^\top b + b^\top b = 2 - 2 \cdot a^\top b \\ &\sim -a^\top b \end{aligned}$$

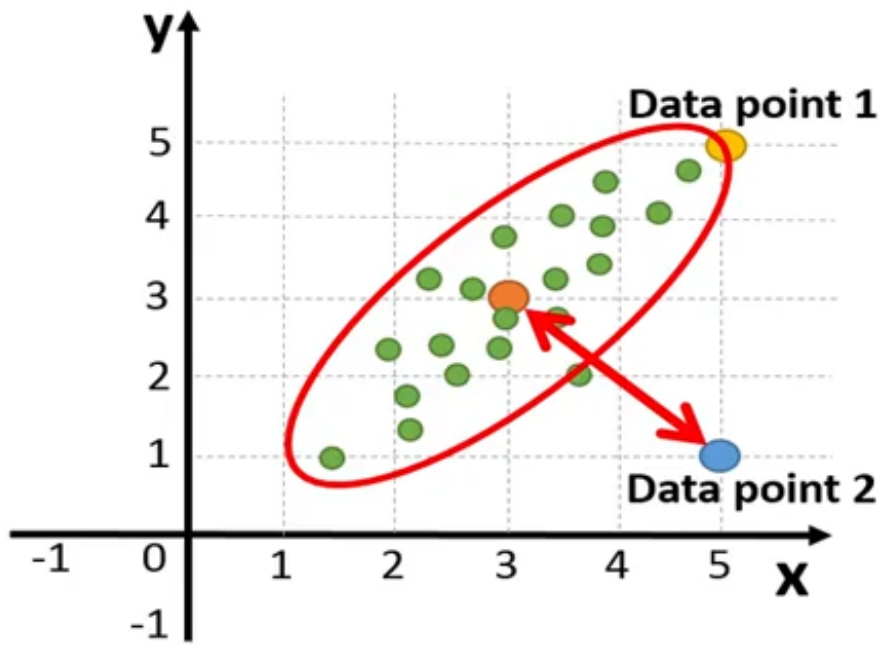
Mahalanobis metric (M is equal to the inverse of data covariance):

$$d_M(a,b) \sim (a-b)^\top \cdot M \cdot (a-b)$$

metrics : Mahalanobis

The Mahalanobis isohypse is marked in red:

- according to Mahalanobis, the yellow sample is much closer to the orange sample than the blue sample
- we can hypothesize that the blue sample does not belong to the green class
- the Euclidian metric does not support such reasoning.



metrics: Mahalanobis

Matrix M is real and symmetric $\Rightarrow$ it may be diagonalized: $M = W^\top W$

Mahalanobis metric may be interpreted as a shallow embedding into a Euclidian metric space W:

$$\begin{aligned} d_M(a,b) &\sim (a-b)^\top \cdot M \cdot (a-b) \\ &\sim (W \cdot (a-b))^\top \cdot (W \cdot (a-b)) \\ &\sim (W \cdot a - W \cdot b)^\top \cdot (W \cdot a - W \cdot b)) \\ &\sim d_E(Wa, Wb) \end{aligned}$$

There are analogous shallow approaches that consider association of the data with symbolic classes (Fisher LDA).

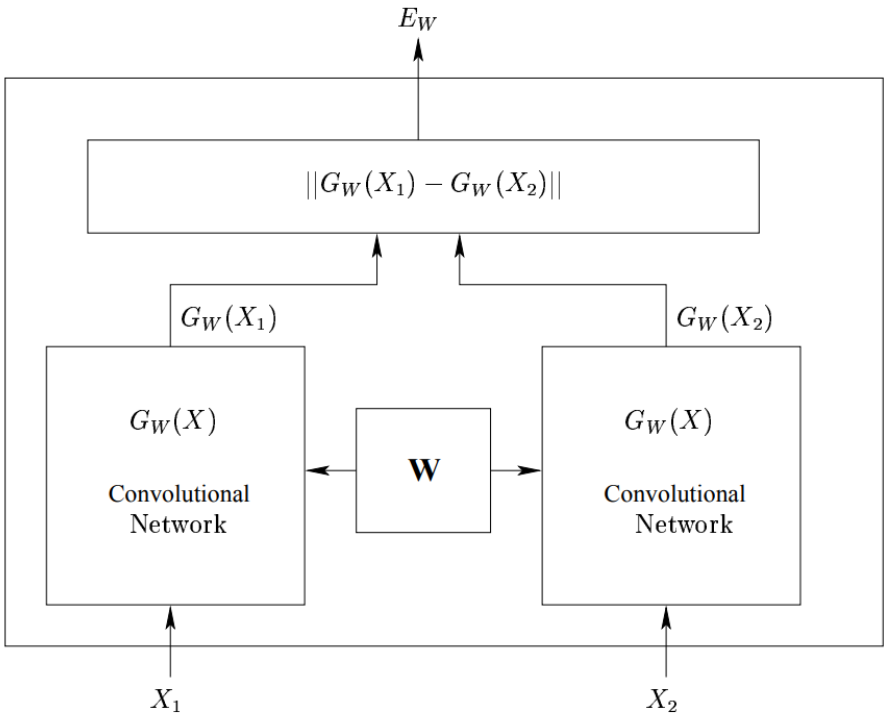
The next logical step: replace W with a **deep model** f_θ:

- what makes sense today, did not make sense in 2005...

Siamese training : the idea

Train a model G_W that embeds data X into a space where the Euclidian metric E_W reflects the (dis)similarity between samples

Siamese learning involves two instances of a parametric module G.



[chopra05cvpr]

The two instances share parameters W, while their gradients are accumulated through the two instances.

Siamese training : loss

Siamese learning involves some variant of the contrastive loss.

Contrastive loss depends on whether the inputs belong to the same "class":

$$L(\theta) = \sum_{y_q = y_p} L_{\text{pos}}(\theta | x_q, x_p) + \sum_{y_q \neq y_n} L_{\text{neg}}(\theta | x_q, x_n)$$

The loss L_{pos} forces the examples from the same class to come closer:

$$L_{\text{pos}}(\theta | x_q, x_p) = \| f_\theta(x_q) - f_\theta(x_p) \|^2$$

The loss L_{neg} forces foreign examples to move from each other:

$$L_{\text{neg}}(\theta | x_q, x_n) = [\max(0, m - \| f_\theta(x_q) - f_\theta(x_n) \|)]^2$$

Siamese training: gradients

Consider the gradients of losses L_{pos} and L_{neg} with respect to metric embeddings $z_q = f_\theta(x_q)$:

$$\frac{\partial L_{\text{pos}}}{\partial z_p} = 2 \cdot (z_p - z_q)$$

$$\frac{\partial L_{\text{neg}}}{\partial z_n} = -2 \cdot \max(0, m - \|z_q - z_n\|) \cdot \frac{z_n - z_q}{\|z_n - z_q\|}$$

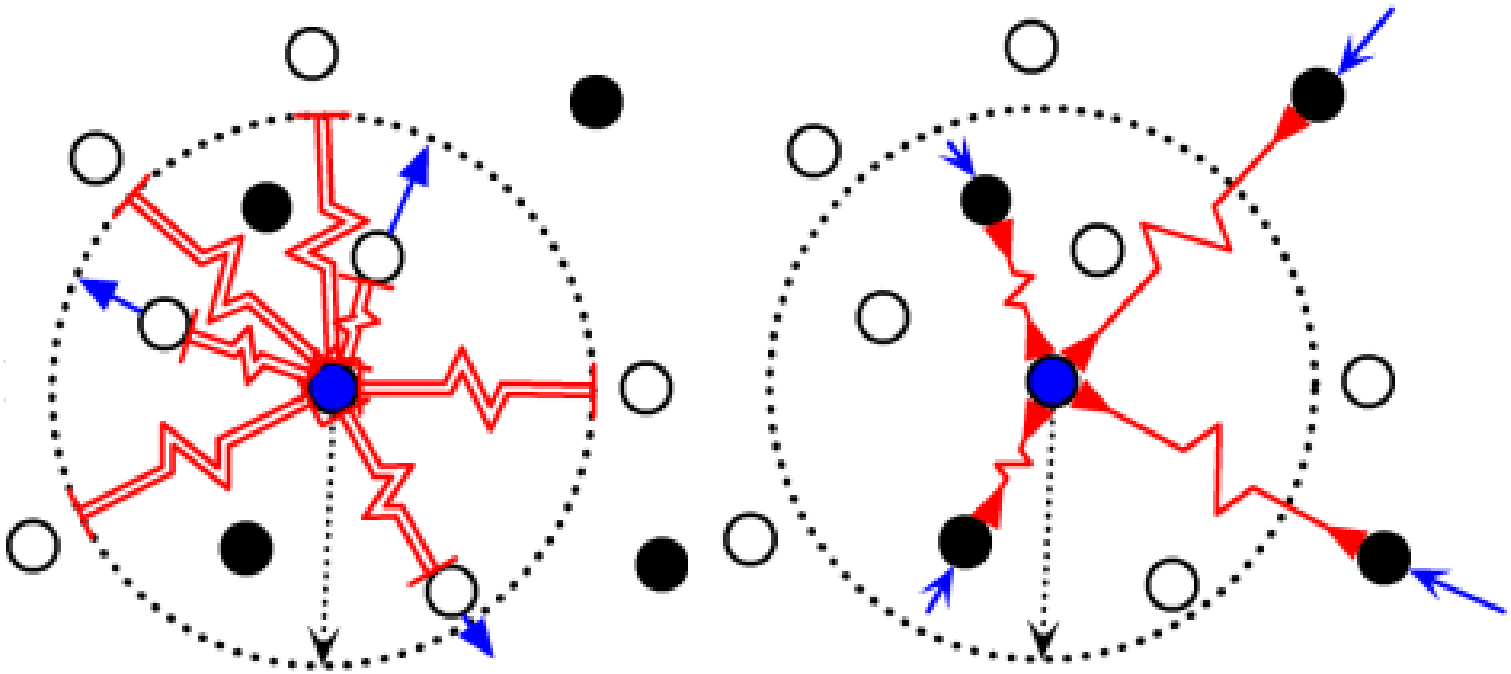
These gradients encourage the positive pairs to move closer, and the negative pairs to move away from each other.

The repellence stops when the distance becomes greater than the margin m.

Siamese training: gradients

We may illustrate the decribed dynamics using a system of mechanical springs (where the force of the spring is proportional to the distance)

- black and white circles represent positive and negative examples with respect to the blue sample
- the negatives outside of radius m do not feel the repellent push from the blue sample



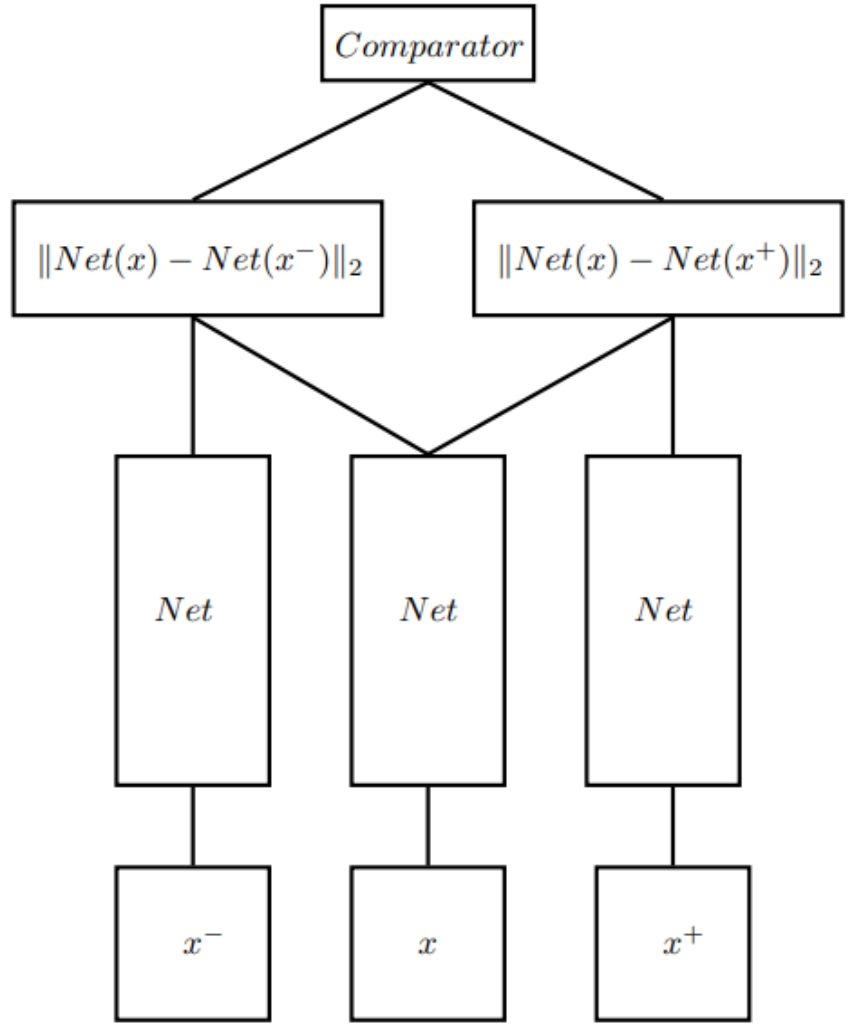
Triplet learning: the idea

In Siamese training, we compare the same sample x with both negative and positive examples.

Embeddings for such samples need to be calculated twice.

The problem is addressed with triplet learning:

- the anchor is compared to the positive example x^+ and the negative example x^-.



[hoffer15iclrw]

The three model instances share parameters and contribute to loss gradients in equal measure.

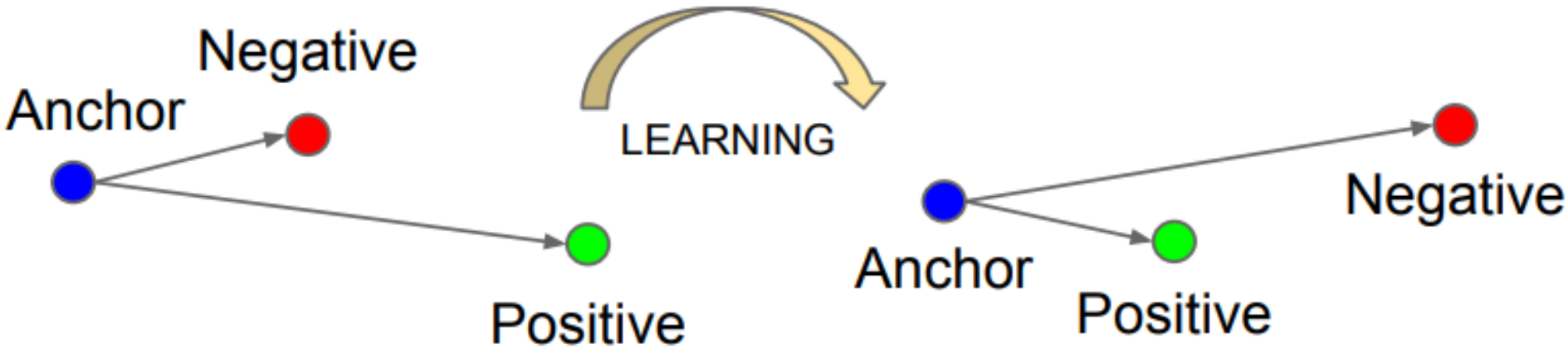
Triplet learning : loss

Triplet learning typically uses the triplet loss.

The triplet loss merges the two components of contrastive loss into a single expression:

$$L(\theta) = \sum_i \max(0, \|f_\theta(x_{ia}) - f_\theta(x_{ip})\| - \|f_\theta(x_{ia}) - f_\theta(x_{in})\| + \alpha)$$

The triplet loss brings the anchor and the positive example closer while increasing the distance between the anchor and the negative:



Triplet learning: gradients

Consider the gradients of the triplet loss with respect to embeddings $f_a = f_\theta(x_{ia})$, $f_p = f_\theta(x_{ip})$ and $f_n = f_\theta(x_{in})$.

These gradients encode the triplet condition that tests whether f_p is closer to f_a than $\|f_n - f_a\| - \alpha$.

If the above condition is not met, then the gradients encourage f_p to come closer to f_a, and f_n to move away from f_a:

$$\frac{\partial L}{\partial f_p} = [\![\|f_a - f_p\| + \alpha - \|f_a - f_n\|]\!] \cdot \frac{f_p - f_a}{\|f_p - f_a\|}$$
$$\frac{\partial L}{\partial f_n} = [\![\|f_a - f_p\| + \alpha - \|f_a - f_n\|]\!] \cdot \frac{f_a - f_n}{\|f_a - f_n\|}$$

Details: soft ReLU

Classic triplet loss ingnores triplets where the **triplet condition** is met.

The advantage of this approach is that it prevents overfitting: when the data is sufficiently well positioned --- the training stops

On the other hand, such approach may be too cautious and hurt generalization

It is sometimes possible to avoid this pitfall by replacing the hard hinge loss ReLU(x) = [x]$_+$ with its relaxed version:

$$\text{softplus}(x) = \ln(1 + e^x)$$

Details: scalar product version

As before, we denote embeddings with respect to x_p, x_p, x_n as:

$$f_a = f_\theta(x_{ia})$$
$$f_p = f_\theta(x_{ip})$$
$$f_n = f_\theta(x_{in})$$

If we normalize the latent representations $\|f_a\| = \|f_p\| = \|f_n\| = 1$, we may express the loss with cosine similarity:

$$L(\theta) = \sum_i \max(0, f_\theta(x_{ia})^\top f_\theta(x_{in}) - f_\theta(x_{ia})^\top f_\theta(x_{ip}) + \alpha)$$

Details: triplet formation

A simplified expression for the basic triplet loss [hermans17arxiv]:

$$L_3(\theta) = \sum_{y_a=y_p\neq y_n} [\alpha + D_{ap} - D_{an}]_+$$

Training triplet formation is an important implementation detail

- the main problem is training efficiency
- the number of triplets increases with $O(N \cdot N_p \cdot N_n)$

It may be worthwile to try the hard triplet loss (eng. batch-hard [hermans17arxiv]) that finds the most difficult negative and positive for a given anchor:

$$L_{BH}(\theta) = \sum_a [\alpha + \max_{y_a=y_p} D_{ap} - \min_{y_a\neq y_p} D_{an}]_+$$

Details: n pairs

Assume that we have [sohn16neurips]:

- n pairs that share an anchor ($\mathbf{x}_a$)
- only one positive ($\mathbf{x}_p$) and n-1 negatives ($\mathbf{x}_{ni}$)

We define the loss so that it increases when the anchor is similar to negatives and decreases when it is similar to positives:

$$\mathcal{L}_{\text{N-pairs}}(\mathbf{x}_a, \mathbf{x}_p, \{\mathbf{x}_{ni}\}) = \log\left(1 + \sum_{i=1}^{N-1} e^{f_\theta(\mathbf{x}_a)^\top f_\theta(\mathbf{x}_{ni}) - f_\theta(\mathbf{x}_a)^\top f_\theta(\mathbf{x}_p)}\right)$$

If n=2, $\mathcal{L}_{\text{N-pairs}}$ is very similar to the triplet loss with a scalar product.

Details: n pairs

After a few simple steps $\mathcal{L}_{\text{N-pairs}}$ may be reduced to the following form:

$$\begin{aligned}
&\mathcal{L}_{\text{N-pairs}}(\mathbf{x}_a, \mathbf{x}_p, \{\mathbf{x}_{ni}\}) = \\
&\quad = -\log \frac{\exp(f_\theta(\mathbf{x}_a)^\top f_\theta(\mathbf{x}_p))}{\exp(f_\theta(\mathbf{x}_a)^\top f_\theta(\mathbf{x}_p)) + \sum_{i=1}^{N-1} \exp(f_\theta(\mathbf{x}_a)^\top f_\theta(\mathbf{x}_{ni}))} \\
&\quad = -\log \frac{\exp(f_\theta(\mathbf{x}_a)^\top f_\theta(\mathbf{x}_p))}{\sum_{i=1}^{N} \exp(f_\theta(\mathbf{x}_a)^\top f_\theta(\mathbf{x}_i))}
\end{aligned}$$

We can see that $\mathcal{L}_{\text{N-pairs}}$ is equivalent to standard cross entropy with a softmax over a vector of similarities between data pairs.

Details: n pairs

The n pairs loss may be generalized to a case where we have several positive and negative examples in a group $\{\mathbf{x}_i, y_i\}_{i=1}^B$
(eng. soft nearest neighbours) [frosst19icml]:

$$\mathcal{L}_{\text{snn}} = -\frac{1}{B}\sum_{i=1}^{B} \log \frac{1}{|\{y_j = y_i, j \neq i\}|} \frac{\sum_{j\neq i, y_j=y_i} e^{f_\theta(\mathbf{x}_i)^\top f_\theta(\mathbf{x}_j)/\tau}}{\sum_{y_i \neq y_k} e^{f_\theta(\mathbf{x}_i)^\top f_\theta(\mathbf{x}_k)/\tau}}$$

- the hyper-parameter τ (temperature) modulates output entropy.

A stricter variant of this loss requires that **every** positive within the group is more similar than the negatives [khosla20neurips]:

$$\mathcal{L}_{\text{sum-out}} = -\frac{1}{B}\sum_{i=1}^{B} \frac{1}{|\{y_j = y_i, j \neq i\}|} \sum_{j\neq i, y_j=y_i} \log \frac{e^{f_\theta(\mathbf{x}_i)^\top f_\theta(\mathbf{x}_j)/\tau}}{\sum_{y_i \neq y_k} e^{f_\theta(\mathbf{x}_i)^\top f_\theta(\mathbf{x}_k)/\tau}}$$

- this generalizes better even though $\mathcal{L}_{\text{snn}} > \mathcal{L}_{\text{sum-out}}$ (Jensen)

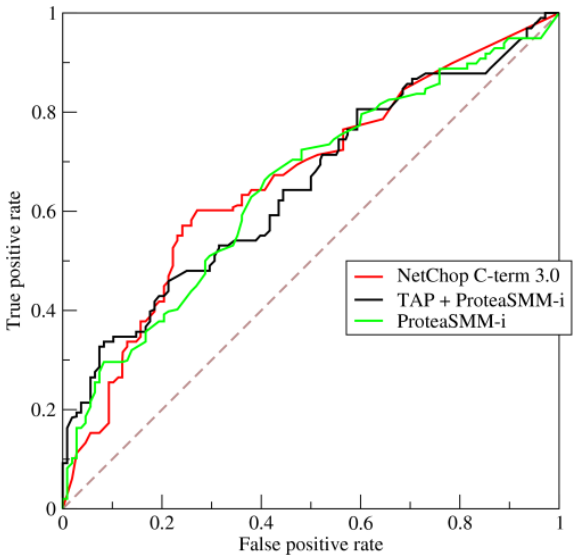
Details : evaluation

The trained similarity measures induce the ranking:

- we fix the sample x_a, sort all other examples according to decreasing similarity: $s_{ai} = -d(f_\theta(x_a), f_\theta(x_i))$
- if the ranking generalizes perfectly (this is usually not the case) then all the positives are ranked before negatives

Rank quality is measured by the area under the PR and ROC curves

- the larger the area $\Rightarrow$ the better the model



[Wikipedia]

Details: evaluation (2)

Here is how we calculate per-threshold metrics:

- we set the threshold at each index i and thus obtain P_i positive ($s_{ax} <= s_{ai}$) and N_i negative ($s_{ax} > s_{ai}$) predictions
- we use the labels to get the counts of:
  - true positives - TP_i
  - false positives - FP_i
  - false negatives - FN_i
  - and true negatives - TN_i
- now we determine the relevant metrics wrt the threshold i:
  - recall $R_i = TPR_i = TP_i/(TP_i+FN_i)$
  - precision $P_i = TP_i/(TP_i+FP_i)$
  - false positive rate = $FPR_i = FP_i/(TN_i+FP_i)$

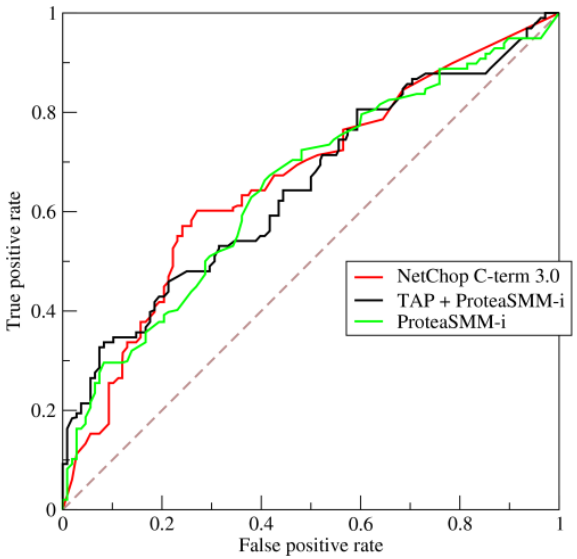
Details: evaluation (3)

We form the curves as follows:

- the precision-recall curve (PR) is made up of points (R_i, P_i)
- receiver operating characteristic curve (ROC) is made up of points (FPR_i, TPR_i)

Rank quality is measured by the area under the PR and ROC curves

- the larger the area $\Rightarrow$ the better the model
- this is a threshold-independent metric of binary classification



[Wikipedia]

Details : exercise

We are given samples x_1 until x_5.

Their labels are Y=[1, 0, 0, 1, 1]

Distances from the sample x_1 to the other samples are:

d(x_1, X) = [0.0, 5.0, 2.0, 3.0, 1.0]

Determine the area under the precision-recall curve (AUPR for short) for the predictions in the sample x_1

Note: AUPR is often referred to as average precision (AP)

Details: exercise - solution

We rank the data as: $[x_5^{(1)}, x_3^{(0)}, x_4^{(1)}, x_2^{(0)}]$

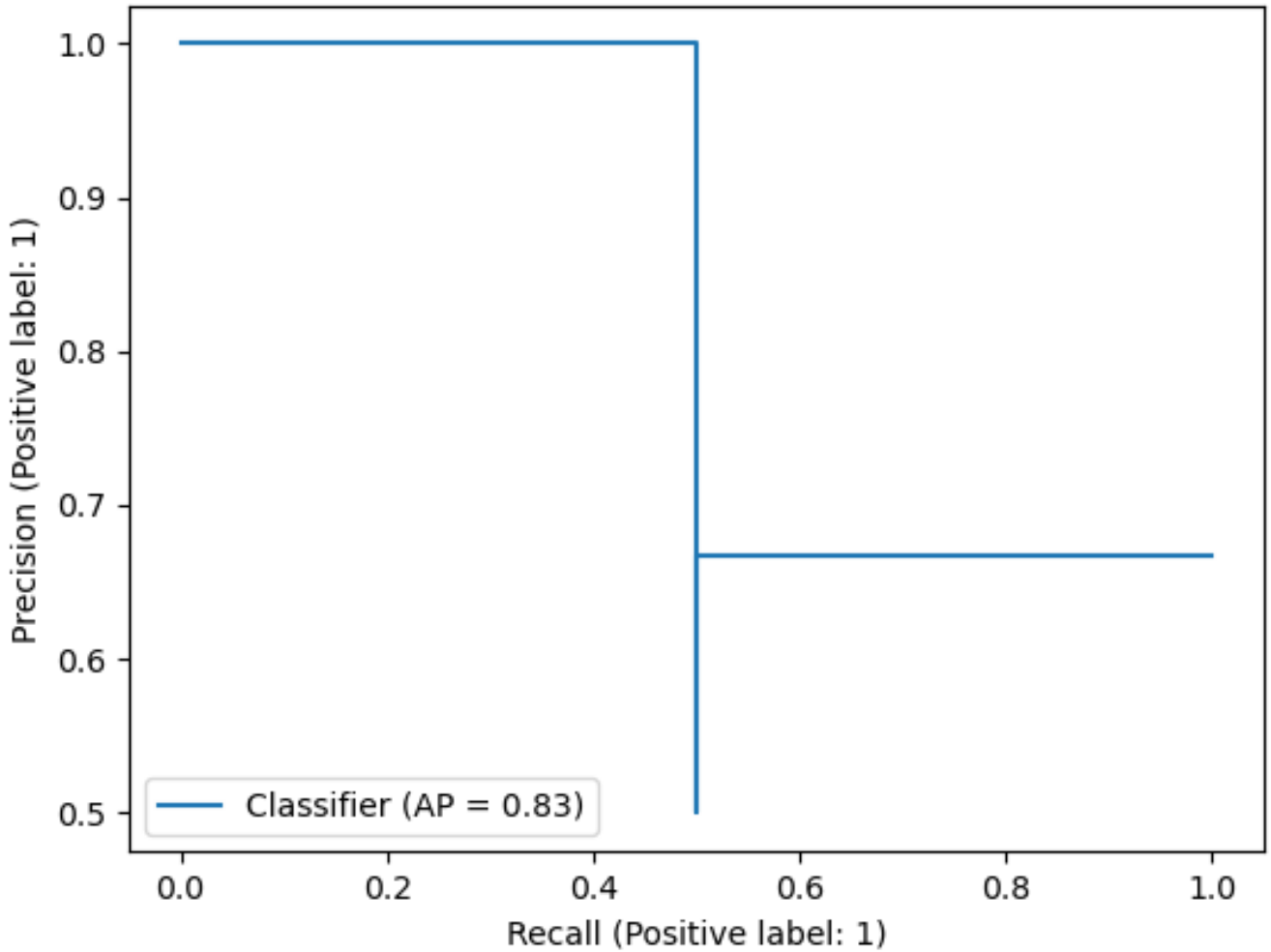
We set the threshold on each datapoint and measure precision P=TP/(TP+FP) and recall R=TP/(TP+FN):

positive predictions	TP	FP	FN	P	R
x_5, x_3, x_4, x_2	2	2	0	0.5	1.0
x_5, x_3, x_4	2	1	0	**0.7**	**1.0**
x_5, x_3	1	1	1	0.5	0.5
x_5	1	0	1	**1.0**	**0.5**

No threshold gives us R=0. We therefore add a point (R=0, P=precision for the lowest R).

If there are more P-s for the same R — we choose the best one.

Details : exercise - graph



Solution: AUPR = (0.5-0)$\times$1 + (1-0.5)$\times$0.66 = 0.83

Details : exercise - code

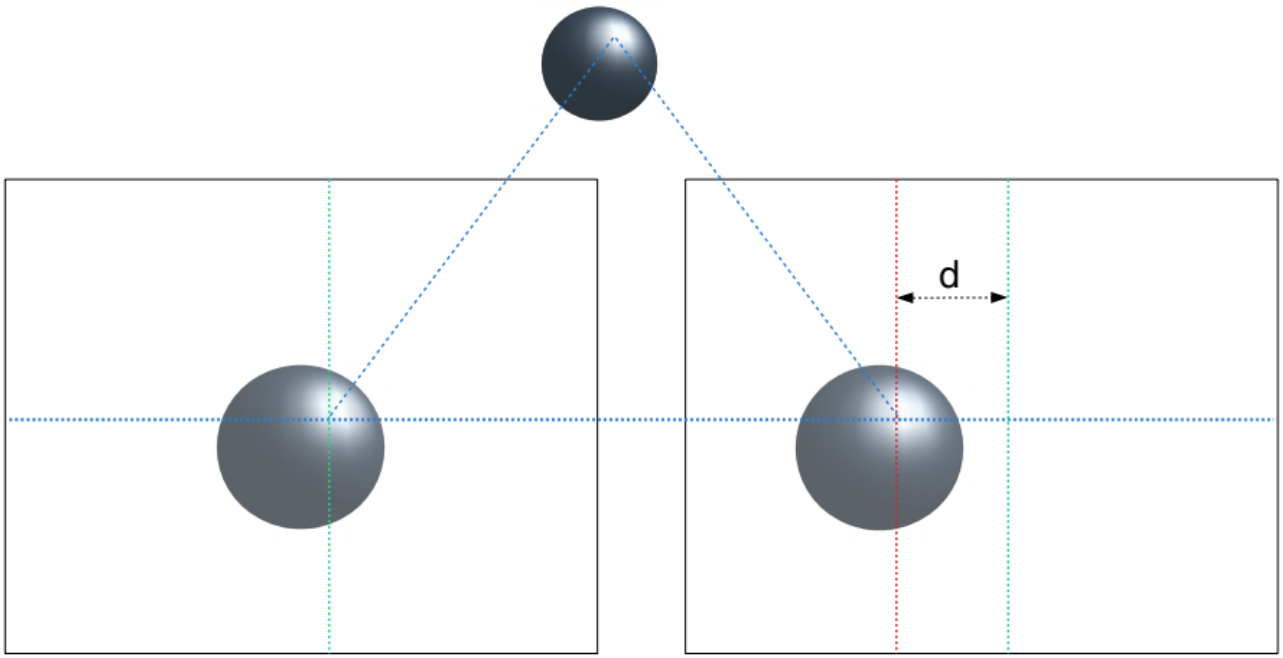
```
import numpy as np
from sklearn.metrics import average_precision_score
from sklearn.metrics import PrecisionRecallDisplay
import matplotlib.pyplot as plt

y_true = np.array([0, 0, 1, 1])
y_scores = np.array([-5, -2, -3, -1])
print(average_precision_score(y_true, y_scores))

PrecisionRecallDisplay.from_predictions(y_true, y_scores)
plt.show()
```

STEREO: EXERCISE

For each pixel in I_{left} we search for a correspondence in I_{right}:



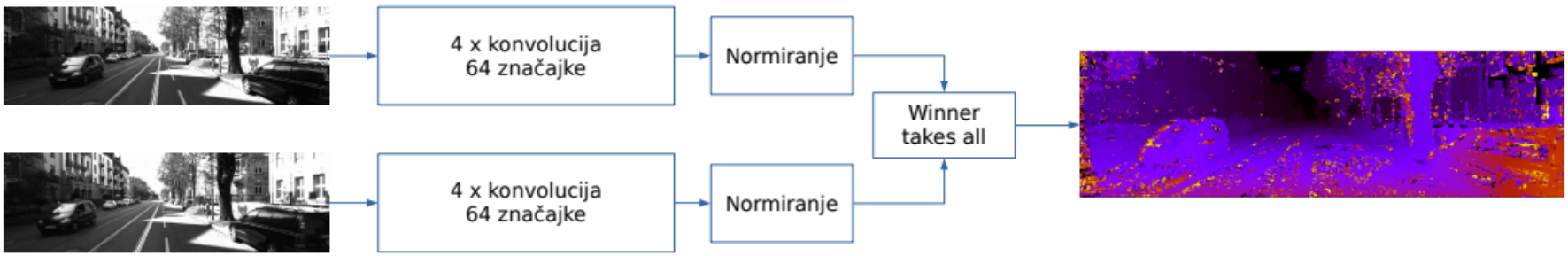
[orsic17ms]

Assume the calibrated case: correspondences in the same row

- we estimate a dense field of horizontal offsets ("disparities")
- if we know the disparity, the field of view, and the baseline distance, then we can reconstruct the scene depth in meters

STEREO: IDEA

Embed pixels from both images into the metric space with a convolutional model $f_\theta : \mathbb{R}^{3\times H\times W} \to \mathbb{R}^{F\times H\times W}$, F=64 [zbontar15cvpr].



[orsic17ms]

Form a dense cost volume of shape D×H×W:

- $V_{ijd} = \mathrm{cost}(f_\theta(I_L)_{i,j}, f_\theta(I_R)_{i,j+d})$

Determine the best disparity for each pixel (winner takes all):

- $D_{ij} = \arg\min_d V_{ijd}$

STEREO : DETAILS

The triplet loss is expressed as a scalar product over normalized metric embedidngs of crops of 9×9:

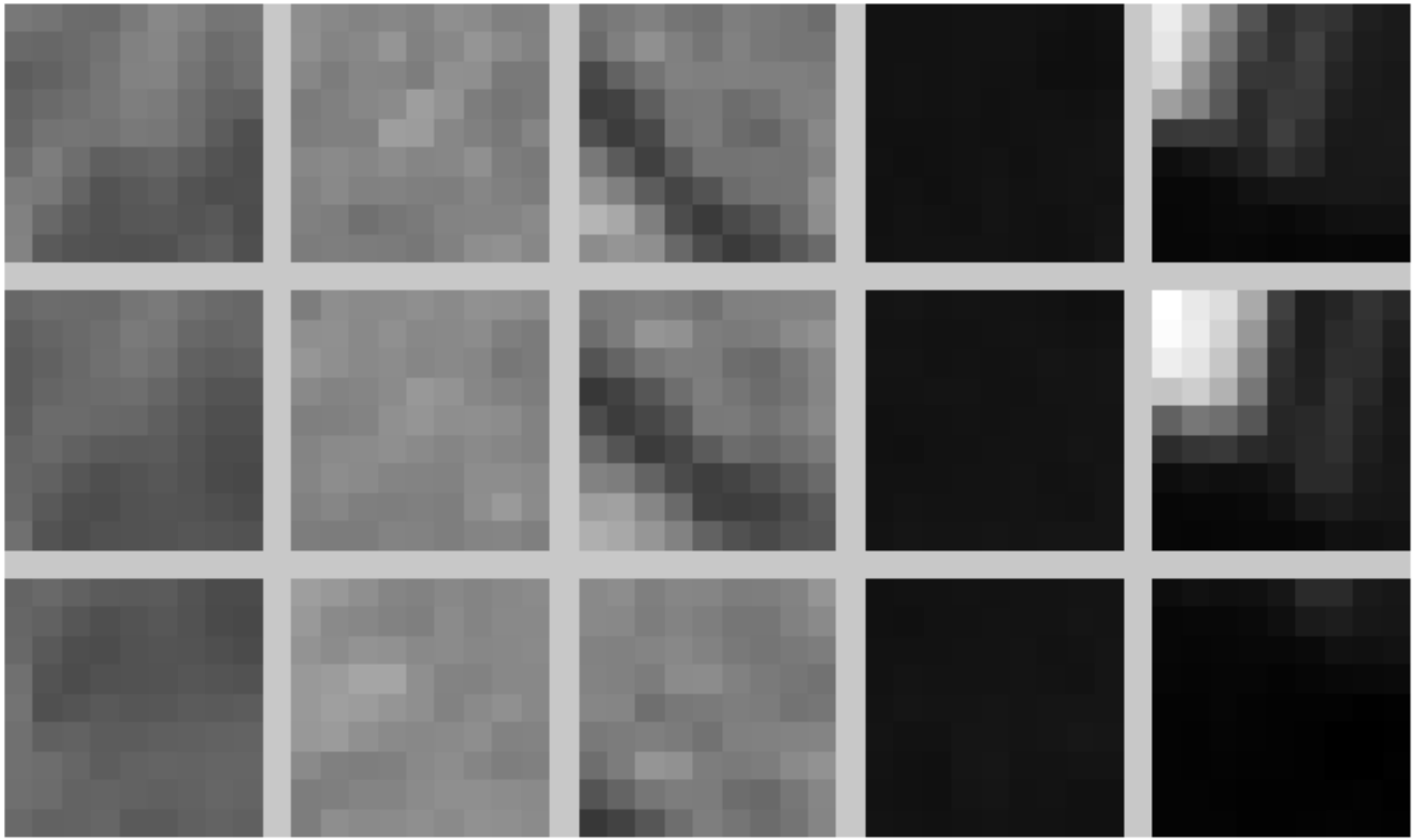
- three instances of model f_θ share parameters
- each instance holds activations and calculates the gradients
- we get a total gradient for each parameter by aggregating the contributions of all model instances

Model f_θ je equivariant: 4×conv3x3 bez sažimanja:

- input training samples have dimensions 128×3×9×9
- We do not use padding during training (we use it during inference)
- embeddings have a dimensionaliti of 64, $f_\theta : \mathbb{R}^{3\times9\times9} \rightarrow \mathbb{R}^{64}$
- image pixels are normalized to $N(0, \mathbf{I})$ during training and inference
- inference is done on whole images 2×3×H×W

STEREO : TRIPLETS

Columns show training triplets:

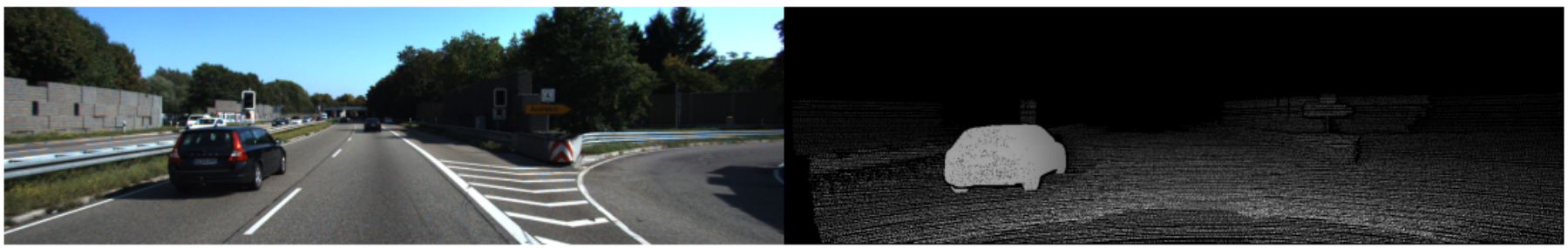


[orsic17ms]

STEREO: EXPERIMENTS

KITTI Datasets [geiger13ijrr]:

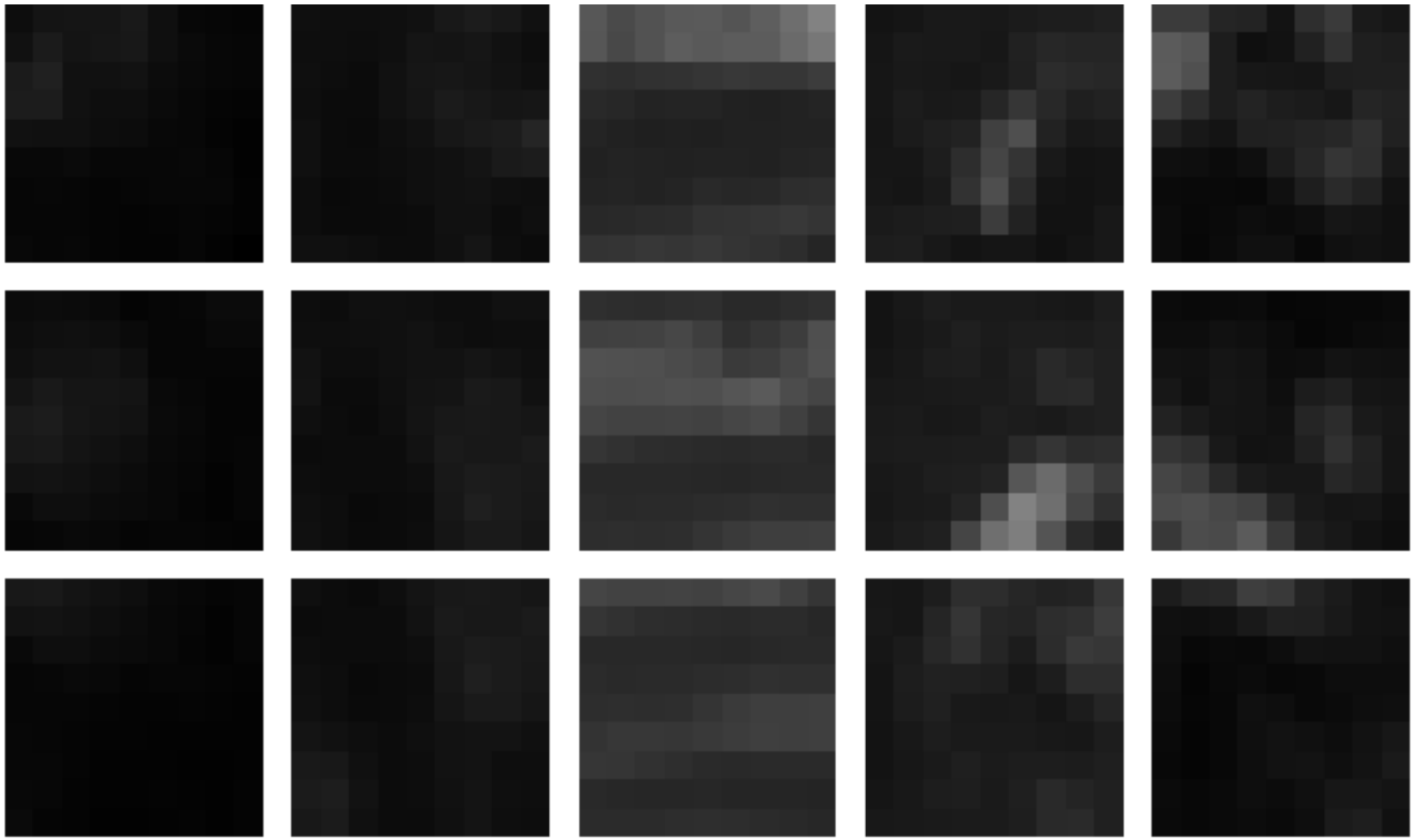
- 200 rectified images 1382 x 512
- training dataset: 80% slika (ostatak - skup za validaciju)
- true disparities are measured with LIDAR in 30% of pixels
- dense disparities on cars were acquired by fitting CAD models



[orsic17ms]

STEREO: ERRORS

Many pixels visible in only one image due to "stereoscopic shadow":



[orsic17ms]

The rows show i) query, ii) correspondence, iii) most similar negative.

STEREO : ACCURACY

Stereoscopic methods on KITTI are evaluated according to percentage of true disparities with tolerance of $\pm$1-3 pixels.

Experimental accuray is solid, though it is worse than SOTA:

Model	Točnost - treniranje	Točnost - testiranje
Sive ulazne slike	84.99%	82.32%
Sive ulazne slike - BN	74.51%	71.61%
Ulazne slike u boji	**85.50%**	**82.84%**
Ulazne slike u boji - BN	74.40%	71.33%

[orsic17ms]

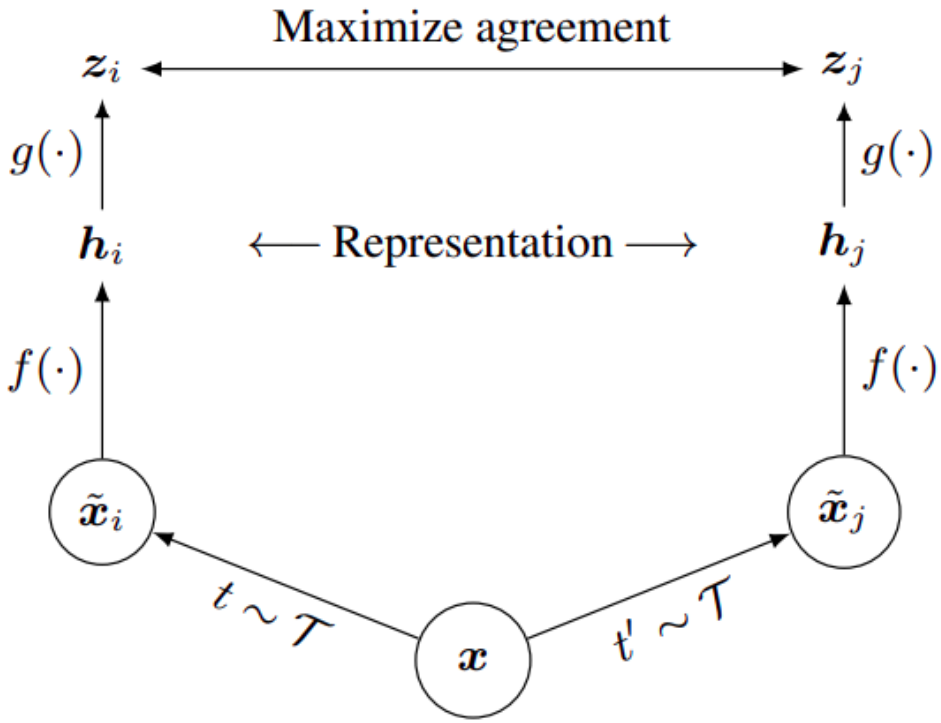
Advantage of metric approaches: resilience to overfitting.

Recent work uses other strategies to solve remaining challenges:

- training on unlabeled video [liu20cvpr]
- filling the areas without correspondences by analyzing the context

SELF-SUPERVISION : TASK

We wish to train useful representation without semantic labels.



[simclr20icml]

- f - convolutional backbone
  - e.g. ResNet-50
- g - projection module
  - e.g. 2× FC(128)
- $t, t' \in T$ - random perturbations
  - e.g. cropping, color augmentations, smoothing

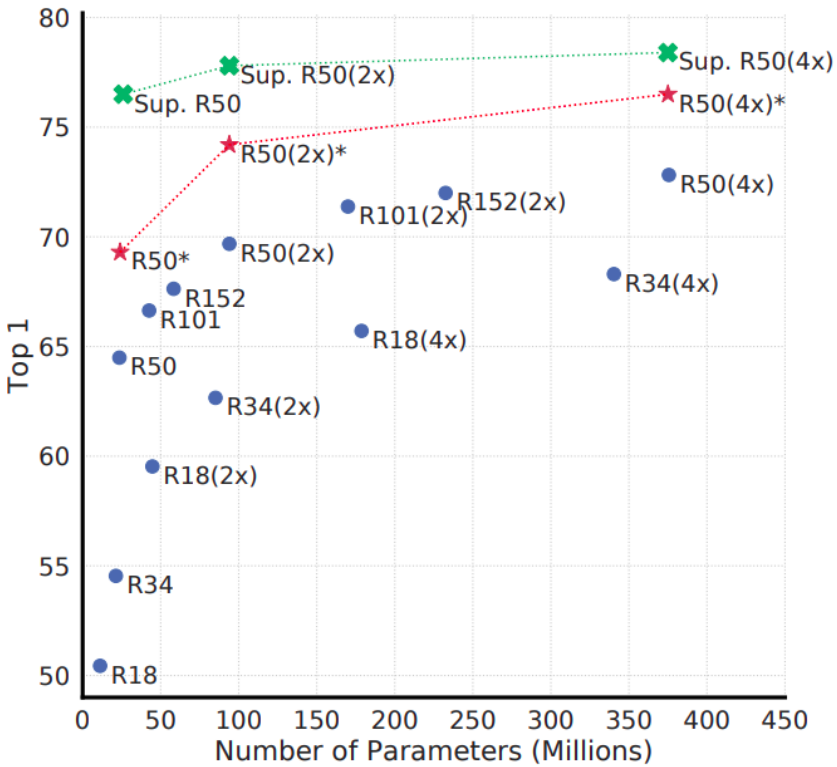
Advantages: we may learn without labels, better knowledge transfer!

Drawbacks: long training, big batches, trivial solutions?

SELF-SUPERVISION : SIMCLR

SimCLR: Simple Contrastive Learning of visual Representations

- n pairs loss: for each positive pair we have 2(B-1) negatives
- inference: discard g, use f for knowledge transfer
- linear probe: fine-tune f for multi-class logistic regression
- applications: pre-training + fine-tuning, semi-supervised learning



- larger models learn better
- SimCLRv2 + linear almost as good as supervised learning
- To achive the best results wee need more training epochs (10×) and large batches (4096)

SELF-SUPERVISION : SIMCLR (2)

Algorithm 1 SimCLR's main learning algorithm.

input: batch size N, constant τ, structure of $f, g, \mathcal{T}$.
for sampled minibatch $\{\boldsymbol{x}_k\}_{k=1}^{N}$ **do**
  for all $k \in \{1, \ldots, N\}$ **do**
    draw two augmentation functions $t \sim \mathcal{T}, t' \sim \mathcal{T}$
    # the first augmentation
    $\tilde{\boldsymbol{x}}_{2k-1} = t(\boldsymbol{x}_k)$
    $\boldsymbol{h}_{2k-1} = f(\tilde{\boldsymbol{x}}_{2k-1})$ # representation
    $\boldsymbol{z}_{2k-1} = g(\boldsymbol{h}_{2k-1})$ # projection
    # the second augmentation
    $\tilde{\boldsymbol{x}}_{2k} = t'(\boldsymbol{x}_k)$
    $\boldsymbol{h}_{2k} = f(\tilde{\boldsymbol{x}}_{2k})$ # representation
    $\boldsymbol{z}_{2k} = g(\boldsymbol{h}_{2k})$ # projection
  end for
  for all $i \in \{1, \ldots, 2N\}$ and $j \in \{1, \ldots, 2N\}$ **do**
    $s_{i,j} = \boldsymbol{z}_i^\top \boldsymbol{z}_j / (\|\boldsymbol{z}_i\| \|\boldsymbol{z}_j\|)$ # pairwise similarity
  end for
  define $\ell(i, j)$ **as** $\ell(i, j) = -\log \frac{\exp(s_{i,j}/\tau)}{\sum_{k=1}^{2N} \mathbb{1}_{[k \neq i]} \exp(s_{i,k}/\tau)}$
  $\mathcal{L} = \frac{1}{2N} \sum_{k=1}^{N} [\ell(2k-1, 2k) + \ell(2k, 2k-1)]$
  update networks f and g to minimize $\mathcal{L}$
end for
return encoder network $f(\cdot)$, and throw away $g(\cdot)$

[simclr20icml]

The loss has 2N terms:

- N: mini-batch size
- each term is $\ell_{NP}(i, j)$
  - $i = 2r - 1$: anchor
  - $j = 2r$: positive
  - $k \neq i, j$: negatives
- $\ell_{NP}(i, j)$: n-pairs loss
  - expressed in terms of similarities s_{ij}, s_{ik}
- the negative data $\{k\}$ correspond to the remaining samples in the mini-group

Conclusion

Quantifying similarity is one of the basic tasks of machine learning.

We use metric embeddings when it is not practical to use classification:

- the number of classes is too large or unknown in advance
- there is only a limited quantity of labeled data

The simplest metric setup involves training a deep model with a Siamese loss.

Applications:

- person tracking, face verification, stereo
- training with a small number of labeled examples (few-shot)
- self-supervised learning
- out-of-distibution detection.