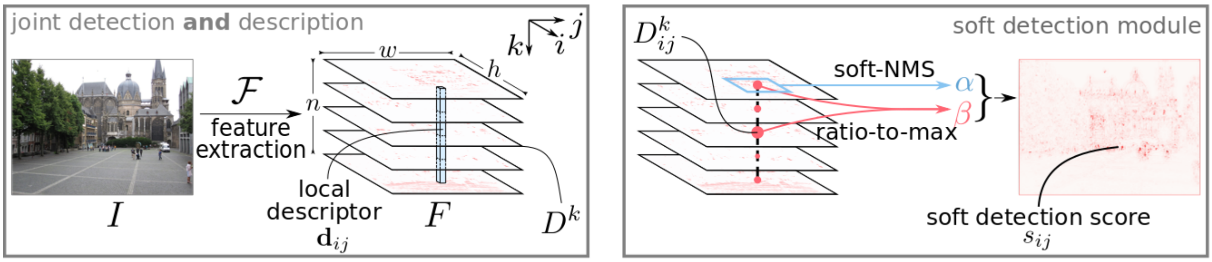
Paper Review: D2-Net

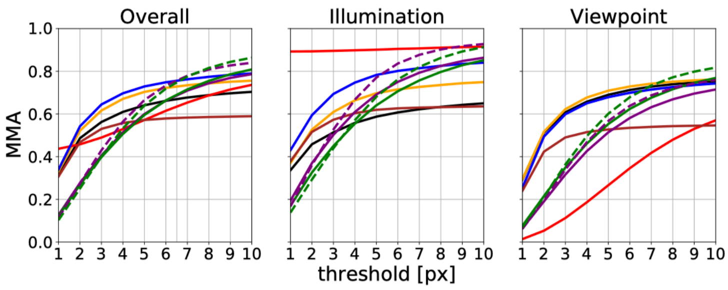
- **Title**: D2-Net: A Trainable CNN for Joint Description and Detection of Local Features
- **Authors**: Mihai Dusmanu, Ignacio Rocco, Tomas Pajdla, Marc Pollefeys, Josef Sivic, Akihiko Torii, Torsten Sattler
- **Link**: http://openaccess.thecvf.com/content_CVPR_2019/papers/Dusmanu_D2-Net_A_Trainable_CNN_for_Joint_Description_and_Detection_of_CVPR_2019_paper.pdf
- **Tags**: Joint Feature Description and Detection, Correspondence, Convolutional Neural Network
- **Year**: 2019

Summary

- **What:**
  - The authors proposed a CNN architecture for simultaneous dense feature description and detection in order to find reliable pixel-level correspondences under difficult imaging conditions.
  - D2-Net obtains state-of-the-art performance on **Aachen Day-Night** (outdoor) and **InLoc** (indoor) localization datasets.
  - The method can be integrated into image matching and 3D reconstruction pipelines.
- **How:**
  - It's a "single-shot" detect-and-describe (D2) approach. A VGG-16 (up to the `conv4_3` layer) backbone is fine-tuned for extracting feature maps:

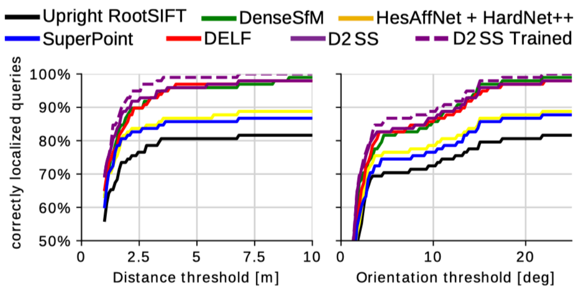


  - Local descriptors (d_ij) are obtained by traversing n feature maps (l2-normalized across channels) at a spatial position (i,j)
  - Detections (scores -- s_ij) are obtained by performing a soft versions of non-local-maximum suppression on a feature map (soft local-maximum score α) + non-maximum suppression across each descriptor (ratio-to-maximum score per descriptor β).
  - Also, during the inference authors propose to create image pyramids for 3 scales: 0.5, 1, 2; then pass through the network and sum the feature maps (using bilinear interpolation for larger iamges and masking already detected regions to prevent re-detection)
  - The objective corresponds to the repeatability of the detector and the distinctiveness of the descriptor. It is an extended triplet margin ranking loss.
- **Results:**
  - **HPatches**



Method	# Features	# Matches
Hes. det. + RootSIFT	6.7K	2.8K
HAN + HN++ [35,36]	3.9K	2.0K
LF-Net [39]	0.5K	0.2K
SuperPoint [13]	1.7K	0.9K
DELF [38]	4.6K	1.9K
D2 SS (ours)	3.0K	1.2K
D2 MS (ours)	4.9K	1.7K
D2 SS Trained (ours)	6.0K	2.5K
D2 MS Trained (ours)	8.3K	2.8K

- **Aachen Day-Night localization dataset**



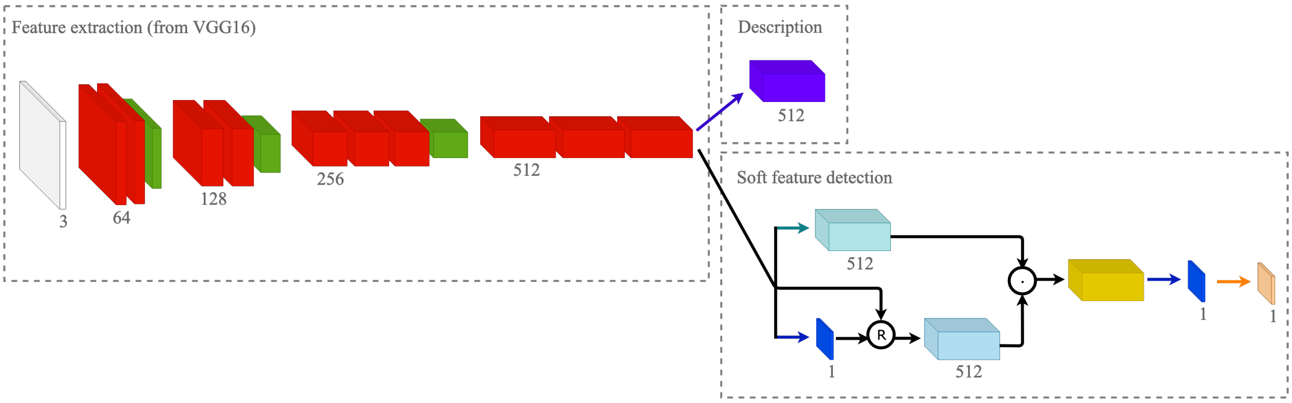
Method	# Features	Correctly localized queries (%) 0.5m, 2°	1.0m, 5°	5.0m, 10°	10m, 25°
Upright RootSIFT [30]	11.3K	36.7	54.1	72.5	81.6
DenseSfM [46]	7.5K / 30K	39.8	60.2	84.7	99.0
HAN + HN++ [35,36]	11.5K	39.8	61.2	77.6	88.8
SuperPoint [13]	6.6K	42.8	57.1	75.5	86.7
DELF [38]	11K	38.8	62.2	85.7	98.0
D2 SS (ours)	7K	41.8	66.3	85.7	98.0
D2 MS (ours)	11.4K	43.9	**67.3**	87.8	99.0
D2 SS Trained (ours)	14.5K	**44.9**	66.3	**88.8**	**100**
D2 MS Trained (ours)	19.3K	**44.9**	64.3	**88.8**	**100**

- **InLoc indoor localization dataset**

Method	Localized queries (%) 0.25m	0.5m	1.0m
Direct PE - Aff. RootSIFT [4,30,32]	18.5	26.4	30.4
Direct PE - D2 MS (ours)	**27.7**	**40.4**	**48.6**
Sparse PE - Aff. RootSIFT – 5MB	21.3	32.2	44.1
Sparse PE - D2 MS (ours) – 15MB	**35.0**	**48.6**	**62.6**
Dense PE [59] – 44MB	**35.0**	46.2	58.1
Sparse PE - Aff. RootSIFT + **Dense PV**	29.5	42.6	54.5
Sparse PE - D2 MS + **Dense PV** (ours)	38.0	54.1	65.4
Dense PE + **Dense PV** (= **InLoc**) [59]	**41.0**	**56.5**	**69.9**
InLoc + D2 MS (ours)	**43.2**	**61.1**	**74.2**

CNN Visualization: D2-Net

Architecture



Blocks description

Output tensors

- Convolution
- Max Pooling
- L2-Normalization (channel-wise)
- Soft Local-Maximum
- Maximum (channel-wise) 1
- Soft Channel Selection
- Multiplication (element-wise)
- Normalization (image-level) 1

Operations

- Data Flow
- L2-Normalization (channel-wise) $y_{ij}^k = \dfrac{x_{ij}^k}{\|x_{ij}\|_2}$
- Soft Local-Maximum $y_{ij}^k = \dfrac{\exp\left(x_{ij}^k\right)}{\sum_{(p,l)\in\mathcal{N}(i,j)} \exp\left(x_{pl}^k\right)}$
- Maximum (channel-wise) $y_{ij} = \max_t \left\{x_{ij}^t\right\}$
- x, $\hat{x}$, R: Ratio-to-max $y_{ij}^k = \dfrac{x_{ij}^k}{\hat{x}_{ij}}$ (here $\hat{x}_{ij} = \max_t \{x_{ij}^t\}$)
- $\cdot$: Multiplication (element-wise) $y = x_1 \cdot x_2$
- Normalization (image-level) $y_{ij} = \dfrac{x_{ij}}{\sum_{(p,l)} x_{pl}}$

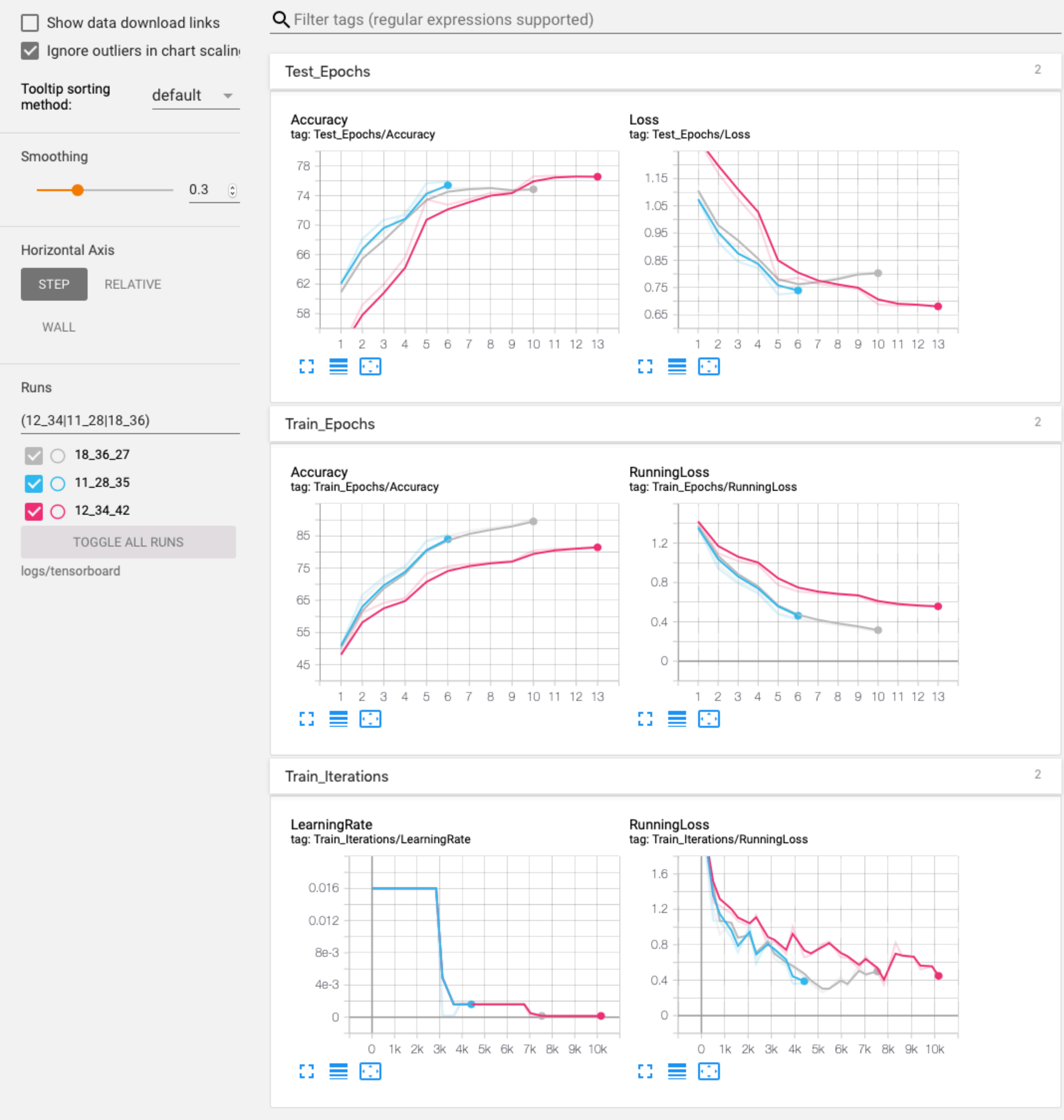
Inputs / Outputs

- Input image
- Soft Detection Scores
- Local Descriptors

Experiments with CIFAR-10

- For CIFAR-10 classification problem I trained a CNN with GeForce GTX 1080.
- I slightly modified `cifar10.ipynb` to get insights.
- I put all essentials in `net.py`, `utils.py`, `train.py`.
- **The experimental pipeline is in** Experiments.ipynb
- The top-3-performance models are following:

timestamp	model	criterion	batch_size	optimizer	scheduler	n_epochs	device	test_accuracy
12_34_42	NetCustom((pool): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False) (conv1): Conv2d(3, 16, kernel_size=(3, 3), stride=(1, 1)) (bn1): BatchNorm2d(16, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True) (conv2): Conv2d(16, 32, kernel_size=(3, 3), stride=(1, 1)) (bn2): BatchNorm2d(32, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True) (conv3): Conv2d(32, 32, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1)) (bn3): BatchNorm2d(32, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True) (fc1): Linear(in_features=1152, out_features=128, bias=True) (fc2): Linear(in_features=128, out_features=64, bias=True) (fc3): Linear(in_features=64, out_features=10, bias=True))	CrossEntropyLoss()	64	SGD(lr: 0.016, momentum: 0.9, dampening: 0, weight_decay: 0.01, nesterov: True)	StepLR(gamma: 0.1, step_size: 5)	13	cuda	76.88%
11_28_35	NetCustom((pool): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False) (conv1): Conv2d(3, 16, kernel_size=(3, 3), stride=(1, 1)) (bn1): BatchNorm2d(16, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True) (conv2): Conv2d(16, 32, kernel_size=(3, 3), stride=(1, 1)) (bn2): BatchNorm2d(32, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True) (conv3): Conv2d(32, 32, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1)) (bn3): BatchNorm2d(32, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True) (fc1): Linear(in_features=1152, out_features=128, bias=True) (fc2): Linear(in_features=128, out_features=64, bias=True) (fc3): Linear(in_features=64, out_features=10, bias=True))	CrossEntropyLoss()	64	SGD(lr: 0.016, momentum: 0.9, dampening: 0, weight_decay: 0, nesterov: True)	StepLR(gamma: 0.1, step_size: 5)	6	cuda	75.89%
18_36_27	NetCustom((pool): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False) (conv1): Conv2d(3, 16, kernel_size=(3, 3), stride=(1, 1)) (bn1): BatchNorm2d(16, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True) (conv2): Conv2d(16, 32, kernel_size=(3, 3), stride=(1, 1)) (bn2): BatchNorm2d(32, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True) (conv3): Conv2d(32, 32, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1)) (bn3): BatchNorm2d(32, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True) (fc1): Linear(in_features=1152, out_features=128, bias=True) (fc2): Linear(in_features=128, out_features=64, bias=True) (fc3): Linear(in_features=64, out_features=10, bias=True))	CrossEntropyLoss()	64	SGD(lr: 0.016, momentum: 0.9, dampening: 0, weight_decay: 0, nesterov: False)	StepLR(gamma: 0.1, step_size: 5)	10	cuda	74.90%



- The best model gained 76.88% test accuracy. It was trained with Stochastic Gradient Descent (batch size 64, learning rate 0.016, with Nesterov momentum; 5-step learning rate decay by 0.1; cross-entropy objective and regularization on weights with 0.01 multiplier) for 10 epochs, then continued up to 15 epochs and stopped on 13th epoch due to Plateau. The architecture is:
  - 16 conv3x3 - relu - bn - maxpool2x2 ->
  - 32 conv3x3 - relu - bn - maxpool2x2 ->
  - 32 conv3x3 - relu - bn ->
  - 128 fc - relu ->
  - 64 fc - relu ->
  - 10 fc
- For the details see Experiments.ipynb
- I also compared the training time with CPU / GPU. See below the results:

1. **GPU (2 min)**

time python train.py

real 0m46.655s
user 1m51.989s
sys 0m9.353s

2. **CPU (22 min)**

time python train.py --no-cuda

real 3m12.462s
user 22m2.023s
sys 0m14.506s

The configurations fot this experiment were:

model	criterion	batch_size	optimizer	scheduler	n_epochs
NetCustom((pool): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False) (conv1): Conv2d(3, 16, kernel_size=(3, 3), stride=(1, 1)) (bn1): BatchNorm2d(16, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True) (conv2): Conv2d(16, 32, kernel_size=(3, 3), stride=(1, 1)) (bn2): BatchNorm2d(32, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True) (conv3): Conv2d(32, 32, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1)) (bn3): BatchNorm2d(32, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True) (fc1): Linear(in_features=1152, out_features=128, bias=True) (fc2): Linear(in_features=128, out_features=64, bias=True) (fc3): Linear(in_features=64, out_features=10, bias=True))	CrossEntropyLoss()	64	SGD(lr: 0.016, momentum: 0.9, dampening: 0, weight_decay: 0, nesterov: False)	StepLR(gamma: 0.1, step_size: 5)	10