Last block:

Classify an image into one of two classes by

- Stack some pooling, FC layers on an encoder
- Adjust result into a vector
- Pass to linear classifier

Learn all parameters with SGD and various losses

- get training examples right

Idea:

- exploit this machinery to improve training
- exploit this machinery to segment image

Multi-class classification

Example:

Attach a label to each pixel in the image

Label is chosen from k alternatives

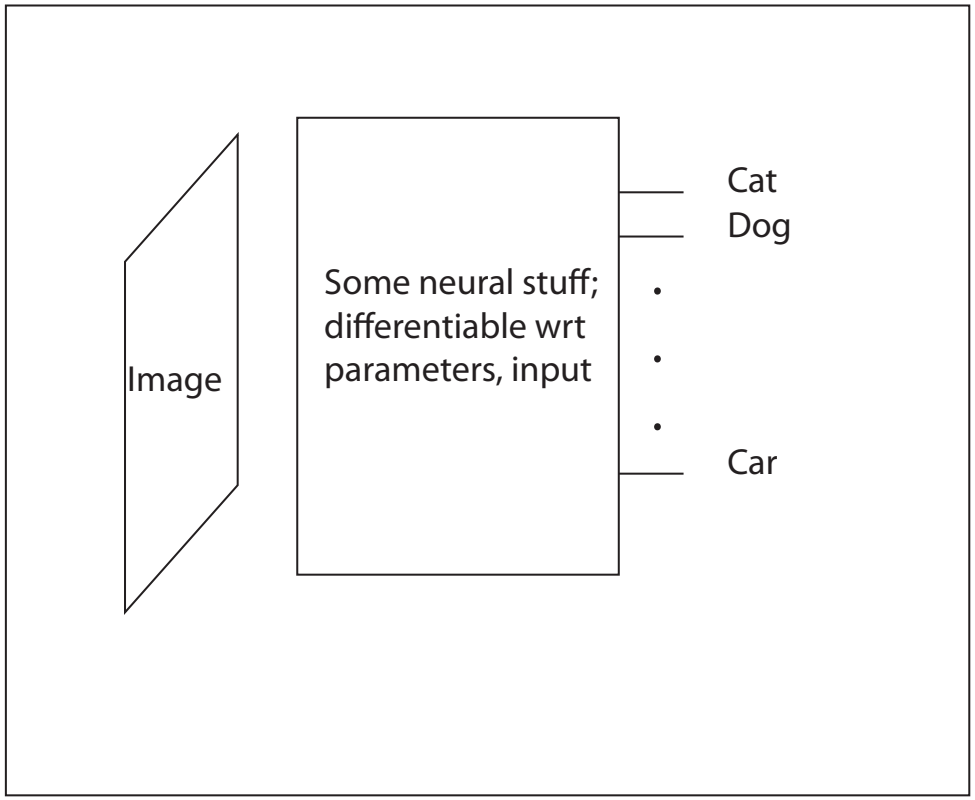
eg: road, car, sky, lamppost, tree, sidewalk, pedestrian, unknown

Multiclass classification

Attach a one-hot vector to each example

k dimensional, all entries zero except one which is one corresponding to true class

Produce one score per class per location



Multi-class classification

Imagine you wish to classify something into one of k classes. You have a set of examples where the classes are known for each example. You can encode this information by associating a *one hot vector* with each training example. This vector has k components, one per class. The component corresponding to the class is one, and all others are zero. Write $\mathbf{y}_i$ for that vector for the i'th training example.

Your classifier must now produce a k dimensional vector for an example. This is quite commonly called a *score* Write $\mathbf{x}$ for the example, $\mathbf{b}$ for the vector produced by the classifier, and θ for all the parameters of the classifier. Then interpret the score as a probability using

$$P(\text{item is class } u|\mathbf{x}, \theta) = \frac{\exp[b_u]}{\sum_w \exp[b_u]}.$$

Loss for multi-class classification

The loss could be the negative log-likelihood of the data $\mathbf{y}$ under the model. Write $\mathbf{p}$ for the vector whose u'th component is

$$\frac{\exp\left[b_u\right]}{\sum_w \exp\left[b_w\right]}.$$

Then the negative log-likelihood for the example is

$$C(\theta; \mathbf{x}, \mathbf{y}) = -\log\left[\mathbf{y}^T \mathbf{p}\right]$$

and for the whole dataset is

$$\sum_{i \in \text{examples}} C(\theta; \mathbf{x}_i, \mathbf{y}_i).$$

Evaluating this expression might strike you as a bit of a performance but notice there is only one non-zero element in $\mathbf{y}_i$, which makes things somewhat simpler. In practice, an API will do this for you very efficiently.

This is also a cross-entropy

Alternatively, the loss could be the cross-entropy of Section 20.2.3. Recall the cross-entropy between a discrete distribution p and another discrete distribution on the same space q is

$$H_x(p, q) = -\mathbb{E}[p]\left[\log q\right] = -\sum_u p_u \log q_u.$$

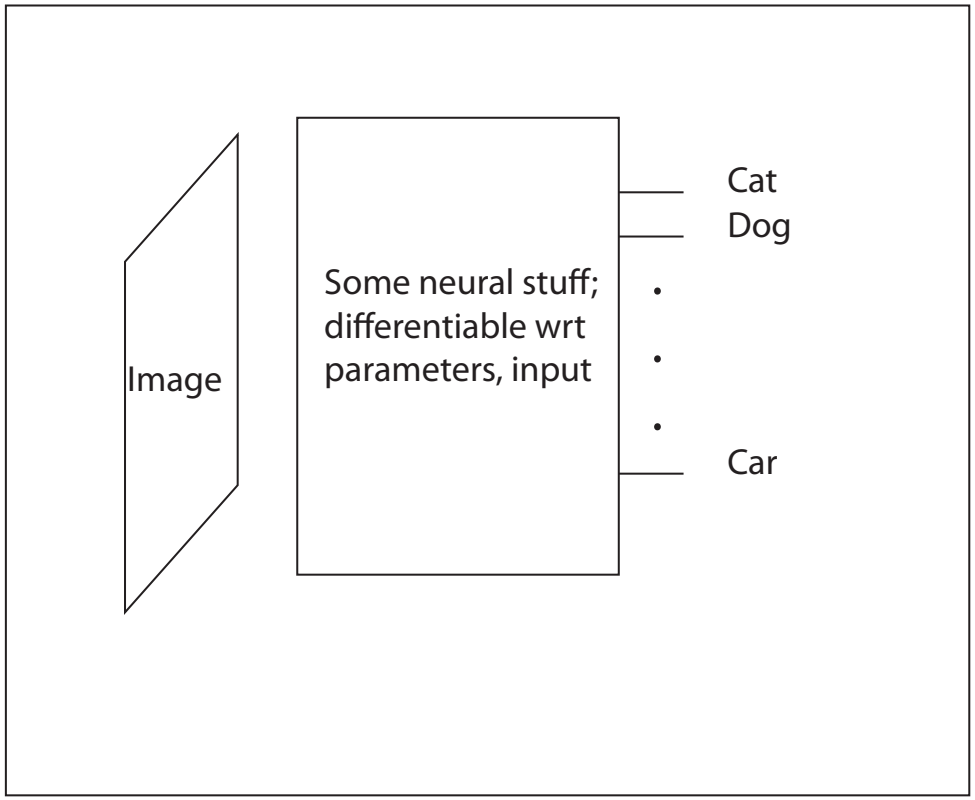
Regard $\mathbf{y}$ as a discrete distribution, and $\mathbf{p}$ as a discrete distribution on the same space. Check that the cross-entropy between these distributions

$$H_x(\mathbf{y}, \mathbf{p}) = -\log\left[\mathbf{y}^T\mathbf{p}\right] = C(\theta; \mathbf{x}, \mathbf{y})$$

(this is almost, but not quite, trivial - look at where the log appears). Most APIs call the loss I have derived in two ways the cross-entropy loss.

Some more detail...

Achieve this by constructing decoder to have k-dim vector at each location



Interest points: Finding interest points

Superpoint:

Break 8M X 8N image into 8x8 windows

Classify each window into 65 classes:

64 locations and one “no-interest point in this window”

so decoder produces 65 X M X N block

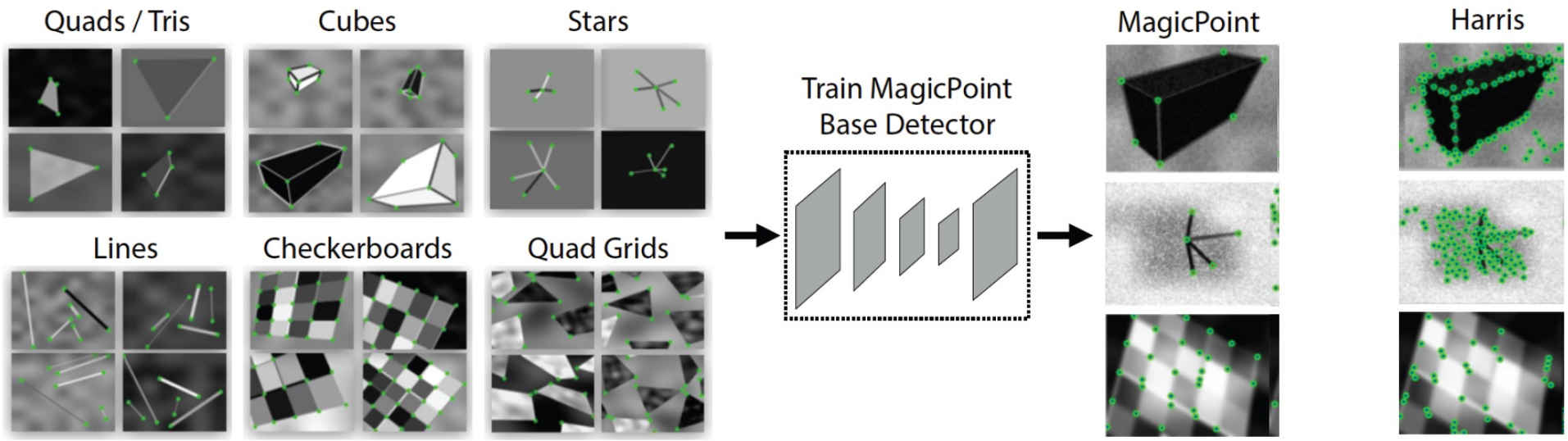
[This can be refined...]

Training:

First, make a lot of synthetic images with known corner locations

then train purely discriminatively (you know where the corners are!)

Yields MagicPoint



But you can do better...

Heatmaps

Take the 65 x M x N block, compute the probabilities,

drop the “no interest point” score to get a 64 x M x N block of scores

These don’t sum to one, cause one option is missing.

Turn this into a 1 x (8 M) x (8 N) “image” – each pixel has a value that indicates the ”probability” of an interest point at that location

Using MagicPoint heatmaps to train ....

Now take a natural image $\mathcal{I}$, form a heatmap $\mathcal{H}\left[\mathcal{I}\right]$

Apply an affine transform A, to get A(I) and H(A(I))

$\mathcal{A}^{-1}\left[\mathcal{H}\left[\mathcal{A}\left[\mathcal{I}\right]\right]\right]$ should be the same as $\mathcal{H}\left[\mathcal{I}\right]$ where they overlap!

Idea:

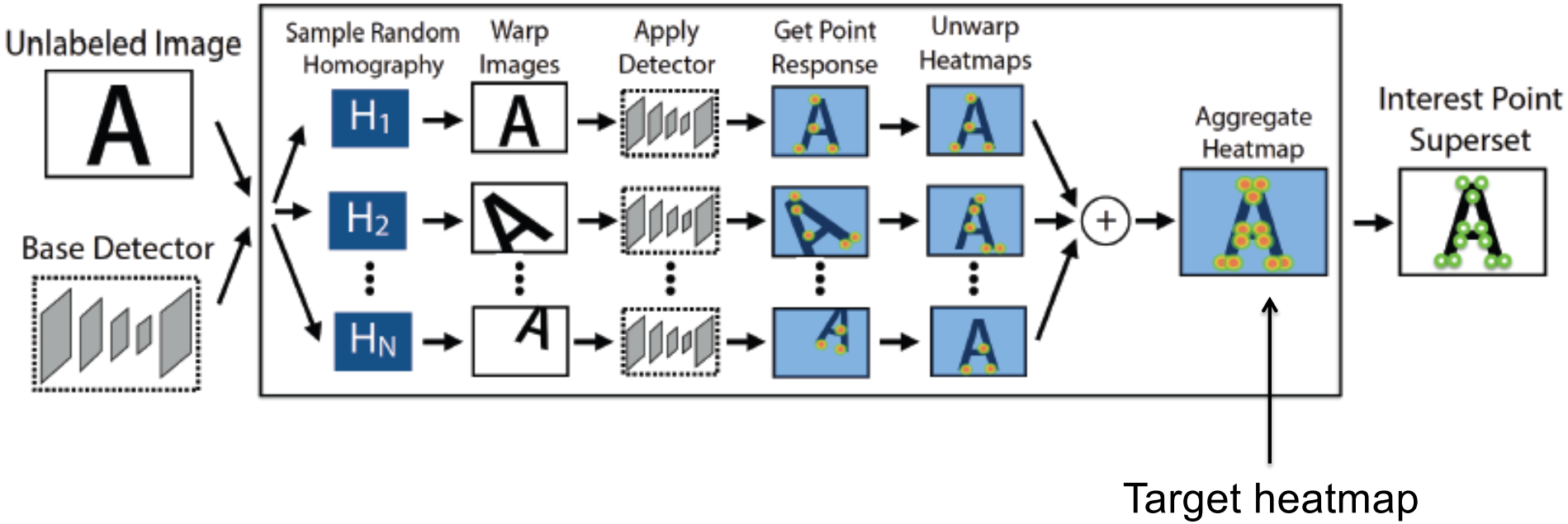
get a collection of random affine transformations A_n

Now compute an average of

$$\frac{1}{N}\sum_{i}\mathcal{A}^{-1}\left[\mathcal{H}\left[\mathcal{A}\left[\mathcal{I}\right]\right]\right]$$

Second, use this as a target distribution for training

SuperPoint


Unlabeled Image
Base Detector
Sample Random Homography
Warp Images
Apply Detector
Get Point Response
Unwarp Heatmaps
H1
H2
HN
Aggregate Heatmap
Interest Point Superset
Target heatmap

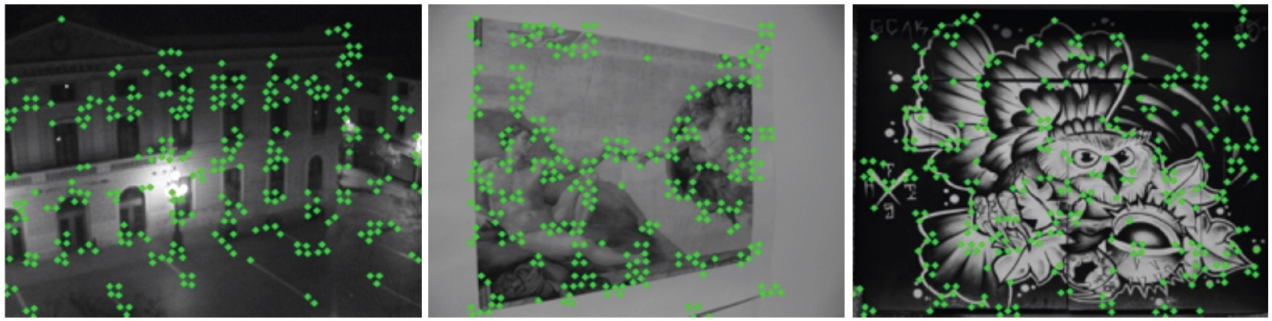
Locating the interest points

Take the scores from the trained decoder, threshold
interest points at locations where the score>threshold

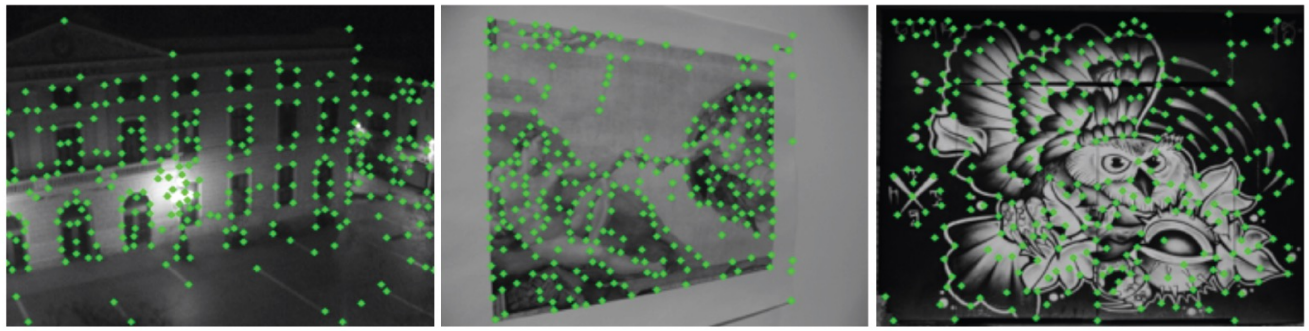
These are on 8M x 8N grid

Real improvements from heatmap

MagicPoint



SuperPoint



Learning to describe the interest point

Add a second decoder.

- This decodes to a d x M x N block of descriptors
  - (but the interest point locations are on (8M x8N) grid – upsample and interpolate)

Now use a version of the affine trick

- For image I and image A(I) you know which pairs of tiles correspond

Tiles

One decoder produces the interest points, as above, and the other produces their descriptions. Because there is only one interest point per tile, it is enough to have one description per tile and upsample the descriptions by interpolation. Now take an image $\mathcal{I}$ and apply a transform $\mathcal{T}$ to the image to obtain $\mathcal{T}(\mathcal{I})$. This transform induces a correspondence between tiles, but remember the tiles are relatively coarse. For a point located at $\mathbf{x} = (i, j)^T$ in the image, write $\mathbf{u} = (u, v)^T = \mathcal{T}(\mathbf{x})$ for the

Descriptors are unit vectors

Tile correspondences

transformed coordinates. Now write

$$s_{\mathbf{x},\mathbf{u}} = \begin{cases} 1 & \text{if } \|\mathbf{x}-\mathbf{u}\|_2 \leq 8 \\ 0 & \text{otherwise} \end{cases}$$

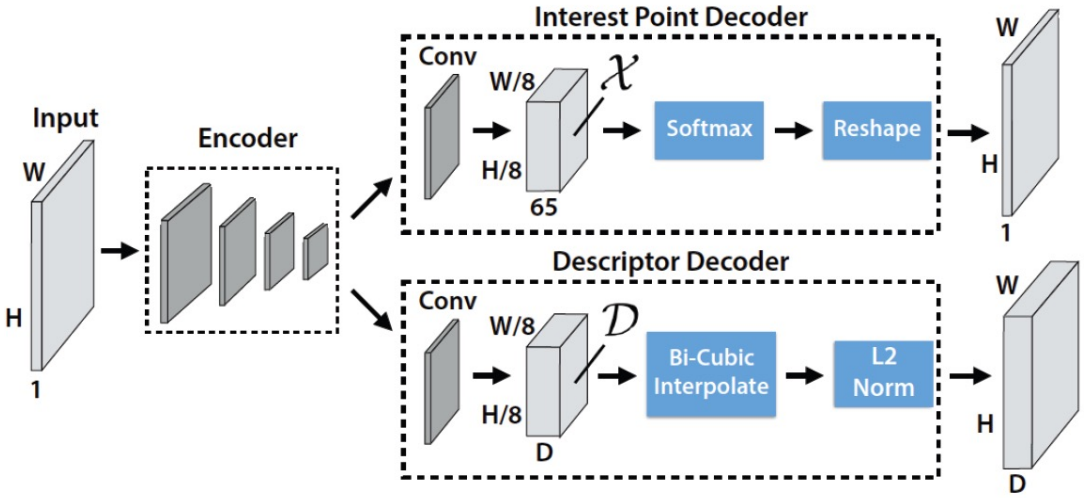
Here $s_{\mathbf{x},\mathbf{u}}$ is one if the tiles correspond (roughly - the tiles are on a grid) and zero otherwise. As in the case of edge detection, there are more pairs that do not correspond than there are pairs that correspond. Deal with this using a weight λ. Then for a pair $\mathbf{x} \in \mathcal{I}$tiles and $\mathbf{u} \in \mathcal{T}(\mathcal{I})$tiles, the cost

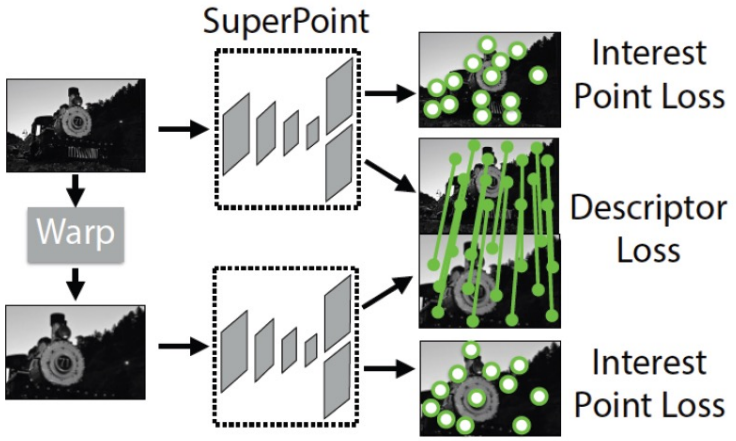
$$C(\mathbf{x},\mathbf{u}) = \lambda s_{\mathbf{x},\mathbf{u}} \max(0, c_1 - \mathbf{d}^T(\mathbf{x})\mathbf{d}(\mathbf{u})) + (1 - s_{\mathbf{x},\mathbf{u}})\max(0, \mathbf{d}^T(\mathbf{x})\mathbf{d}(\mathbf{u}) - c_2)$$

forces $\mathbf{d}(\mathbf{x})$ and $\mathbf{d}(\mathbf{u})$ to be similar when $\mathbf{x}$ and $\mathbf{u}$ correspond and different when they do not. This yields a loss

$$\sum_{\mathbf{x} \in \mathcal{I}\text{tiles}} \sum_{\mathbf{u} \in \mathcal{T}(\mathcal{I})\text{tiles}} C(\mathbf{x},\mathbf{u})$$

The resulting descriptors are comparable in accuracy with SIFT descriptors []


Input
W
H
1
Encoder
Interest Point Decoder
Conv
W/8
H/8
65
$\mathcal{X}$
Softmax
Reshape
W
H
1
Descriptor Decoder
Conv
W/8
H/8
D
$\mathcal{D}$
Bi-Cubic Interpolate
L2 Norm
W
H
D


SuperPoint
Warp
Interest Point Loss
Descriptor Loss
Interest Point Loss