SuperPoint: A Case Study in Self-Supervised Networks for Keypoint Detection

ECE661: Computer Vision

Avi Kak

Purdue University

Thursday 19th September, 2024 11:14

Preamble

Neural networks for computer vision applications are traditionally trained using human annotated images, with the annotations denoting what it is that a neural network is supposed to see in an input image.

What is described above is used for creating neural-network based image classifiers, object detectors in images, object trackers in videos, etc.

Unfortunately, such supervised training approaches cannot be used for, say, a neural network that extracts keypoints in images.

Keypoints in images are supposed to be locally high-variance points that remain substantially unchanged despite in-plane rotations of the image, significant changes in illumination and in the viewpoint angle. Associated with each keypoint is a descriptor vector. This descriptor vector must also remain substantially invariant to in-plane rotations and to changes in scene illumination and the viewpoint angle for the image.

Preamble (contd.)

In general, it is very difficult for a human to decide whether or not a locally "sharp" looking feature in an image would qualify as an keypoint. Probably the most important reason for that is that the keypoints are also supposed to be *scale invariant*.

To appreciate scale invariance, consider a photo of a building facade that has a red door. Assume that, except for for its doors and windows, the building is generally painted in a shade of gray. When you apply a keypoint detector to the photo, you would get keypoints at each of the four corners of the door, as you would expect. However, since keypoint detectors carry out a multi-scale analysis of an image, you are also likely to get an keypoint somewhere in the middle of the door itself.

Figuratively speaking, multiscale amounts to looking at an object from more and more distant viewpoints and, at each distance, extracting the keypoints from the photo recorded. For computer analysis, this process can be simulated (albeit only approximately) by analyzing the image in a Gaussian pyramid and extracting the high-curvature points as keypoints at different levels of smoothing applied to the image.

Preamble (contd.)

For extended objects in an image, it would be difficult for the human eye to foretell the precise location of the keypoints that would be produced by a multi-scale keypoint operator. And, therefore, it would be impossible for a human to provide the annotations you would need for training a neural network that extracts keypoints.

Therefore, if a neural network is to serve as an keypoint detector, its learning must be **self-supervised**. That is, it should NOT need human-generated annotations.

The goal of this tutorial is to present the SuperPoint neural network as an attempt at building a self-supervised network for the purpose of extracting keypoints.

The training of SuperPoint starts by using synthetic images in which the locations of the keypoints are known *a priori*. For the synthetic image dataset, you construct images with different shapes — as in the PurdueShapes5 and PurdueDrEvalMultiDataset datasets used in our Deep Learning class — with the additional feature that, for each images, the dataset also includes the coordinates of the shape corners.

Preamble (contd.)

As is to be expected, a network trained as described on the previous slide will work reasonably well but only on simple images like those used in the training dataset.

You can increase the power of the network through a self-supervised process called *Homographic Adaptation*.

In Homographic Adaptation, you subject the images in the dataset described above to a wide range of randomly chosen 3×3 homographies. Since you know exactly how each homography will alter the coordinates of the keypoints, you can generate the ground-truth coordinates for the keypoints in the output images. The data generated in this manner can be used to fine-tune the learnable weights produced in the previous step.

The goal of this tutorial is to explain these ideas as implemented in the SuperPoint network.

You can use this link to download the SuperPoint paper by Daniel DeTone, Tomasz Malisiewicz, and Andrew Rabinovic, all from Magic Leap:

`https://arxiv.org/pdf/1712.07629.pdf`

Outline

Outline

The SuperPoint Framework

- SuperPoint is one of the first self-supervised networks that outputs the keypoints and descriptor vectors for images.
- The network is trained initially on synthetic images in which you know *a priori* the locations of the keypoints.
- Subsequently, the framework uses a concept called Homographic Adaptation for generalizing the learning acquired initially from the synthetic images.

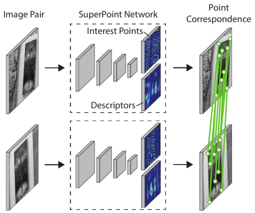


Figure: The basic idea of SuperPoint is to feed into the network an image pair (I, I') that are related by a known random homography $\mathcal{H}$ and then to put to use the fact that for each input image, we know the actual locations of the keypoints at the output.

The SuperPoint Framework (contd.)

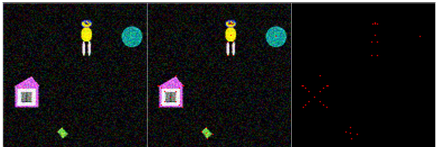
- There are only two ways to generate the sort of synthetic image dataset needed by SuperPoint like network: You write a Python script to create images of simple polygonal shapes (triangles, rectangles, checkerboard patterns, etc.) And, subsequently, you apply a corner detector to the shapes to calculate the locations of the keypoints.
- Alternatively, you record the coordinates of the corner points in the data generation script itself and, for each image, make a list of all such point coordinates as the annotation for that image.
- To the best of what I can tell, the SuperPoint folks used the former approach. [Our own self-supervised deep-learning expert Rahul Deshmukh tells me that the authors of SuperPoint have not made their dataset public.]
- Since such a dataset is easy to generate, I have modified one of the scripts I wrote originally for our Deep Learning class for generating our own synthetic images. [I'll be making my dataset public when I also release my implementation of a SuperPoint like network for training a neural-network based keypoint detector.]

Outline

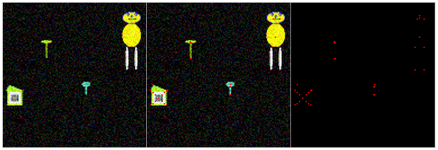
Purdue Keypoint Detector Dataset

- The goal of this and the next few slides is to convey a sense of what sort of synthetic images we are talking about.
- Each slide shows two images from the dataset. The left half of each slide shows an original image as it exists in the dataset, along with the same image with the keypoints superimposed, and, finally, an image of just the keypoints.
- The right half of each slide does the same for a different dataset image.
- Shown below the two image entries in each slide are the numerical values for the keypoint coordinates.
- All images in the dataset are of size 128×128.

The Dataset (contd.)



(a) An image, the image with keypoints superimposed, and just the keypoints



(b) An image, the image with keypoints superimposed, and just the keypoints

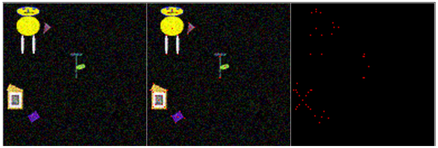
Keypoint coordinates:

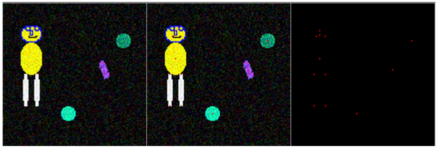
```
[(9.0, 73.0), (32.0, 73.0), (25.0, 64.0), (10.0, 92.0), (31.0, 73.0), (10.0,
73.0), (31.0, 92.0), (12.0, 90.0), (28.0, 75.0), (12.0, 75.0), (28.0, 90.0),
(15.0, 87.0), (25.0, 78.0), (15.0, 78.0), (25.0, 87.0), (20.0, 83.0), (52,
115), (48, 114), (58, 116), (52, 110), (53, 120), (74.0, 28.0), (76, 18),
(72, 18), (74, 17), (74, 18), (71, 34), (71, 46), (76, 34), (76, 46), (114,
29)]
```

Keypoint coordinates:

```
[(114.0, 30.0), (118, 14), (112, 14), (114, 11), (114, 13), (110, 39), (110,
59), (118, 39), (118, 59), (73.0, 81.0), (74.0, 74.0), (73.0, 74.0), (74.0,
81.0), (74.0, 72.0), (38.0, 49.0), (39.0, 35.0), (38.0, 35.0), (39.0, 49.0),
(38.0, 34.0), (3.0, 77.0), (18.0, 77.0), (5.0, 72.0), (3.0, 90.0), (17.0, 77.0),
(3.0, 77.0), (17.0, 90.0), (5.0, 89.0), (15.0, 79.0), (5.0, 79.0), (15.0, 89.0),
(7.0, 87.0), (13.0, 81.0), (7.0, 81.0), (13.0, 87.0), (10.0, 84.0)]
```

The Dataset (contd.)


(a) An image, the image with keypoints superimposed, and just the keypoints


(b) An image, the image with keypoints superimposed, and just the keypoints

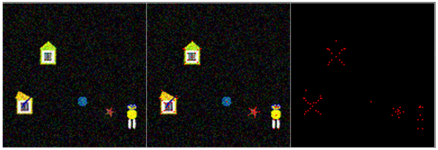
Keypoint coordinates:

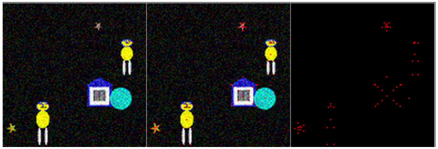
[(2.0, 77.0), (18.0, 77.0), (5.0, 71.0), (4.0, 94.0), (17.0, 77.0), (4.0, 77.0), (17.0, 94.0), (5.0, 92.0), (15.0, 79.0), (5.0, 79.0), (15.0, 92.0), (7.0, 90.0), (13.0, 82.0), (7.0, 82.0), (13.0, 90.0), (10.0, 86.0), (27, 101), (21, 100), (33, 102), (29, 96), (25, 106), (22.0, 22.0), (26, 8), (18, 8), (22, 5), (22, 7), (17, 28), (17, 45), (27, 28), (27, 45), (42, 21), (37, 17), (36, 27), (38, 21), (64.0, 66.0), (65.0, 47.0), (64.0, 47.0), (65.0, 66.0), (65.0, 45.0), (69, 56)]

Keypoint coordinates:

[(25.5, 49.5), (30, 29), (22, 29), (25, 24), (25, 28), (20, 63), (20, 91), (30, 63), (30, 91), (107, 33), (58, 98), (90, 59)]

The Dataset (contd.)


(a) An image, the image with keypoints superimposed, and just the keypoints


(b) An image, the image with keypoints superimposed, and just the keypoints

Keypoint coordinates:

```
[(11.0, 84.0), (27.0, 84.0), (13.0, 77.0), (12.0, 98.0), (26.0, 84.0), (12.0,
84.0), (26.0, 98.0), (13.0, 96.0), (24.0, 86.0), (13.0, 86.0), (24.0, 96.0),
(15.0, 95.0), (22.0, 88.0), (15.0, 88.0), (22.0, 95.0), (19.0, 91.0), (17,
93), (17, 89), (26, 82), (20, 88), (32.0, 40.0), (48.0, 40.0), (40.0, 33.0),
(33.0, 54.0), (47.0, 40.0), (33.0, 40.0), (47.0, 54.0), (34.0, 52.0), (45.0,
41.0), (34.0, 41.0), (45.0, 52.0), (36.0, 50.0), (43.0, 44.0), (36.0, 44.0),
(43.0, 50.0), (40.0, 47.0), (71, 87), (115.0, 99.0), (117, 92), (114, 92),
(115, 91), (115, 92), (113, 103), (113, 111), (117, 103), (117, 111), (91,
93), (93, 96), (90, 99), (95, 98), (96, 101), (97, 97), (100, 96), (96, 95),
(97, 92), (95, 94), (95, 96)]
```

Keypoint coordinates:

```
[(35.0, 103.0), (38, 92), (33, 92), (35, 89), (35, 91), (32, 110), (32, 125),
(38, 110), (38, 125), (6, 106), (6, 110), (3, 111), (6, 112), (6, 115), (8,
112), (11, 113), (9, 110), (11, 108), (8, 109), (7, 110), (110.0, 45.0),
(113, 35), (109, 35), (110, 34), (110, 35), (108, 51), (108, 63), (113,
51), (113, 63), (83, 17), (84, 19), (81, 21), (84, 21), (85, 24), (86, 21),
(88, 21), (86, 20), (87, 17), (85, 18), (85, 20), (74.0, 72.0), (98.0, 72.0),
(85.0, 65.0), (75.0, 91.0), (97.0, 72.0), (75.0, 72.0), (97.0, 91.0), (77.0,
89.0), (94.0, 74.0), (77.0, 74.0), (94.0, 89.0), (81.0, 86.0), (91.0, 77.0),
(81.0, 77.0), (91.0, 86.0), (85.0, 82.0), (105, 84)]
```

Outline

SuperPoint – The Learning Framework

- The dataset of the sort previously described is fed into a basic detector network shown in the left third of the figure below. This is a binary detector that gives a 1/0 class label to each pixel in the input image based on whether that pixel is a keypoint or not.
- The network uses the Cross Entropy Loss for measuring the errors in the labels assigned to the pixels. [As to why it is a Cross Entropy Loss and not a Binary Cross Entropy Loss is an interesting story unto itself. More on that later.]

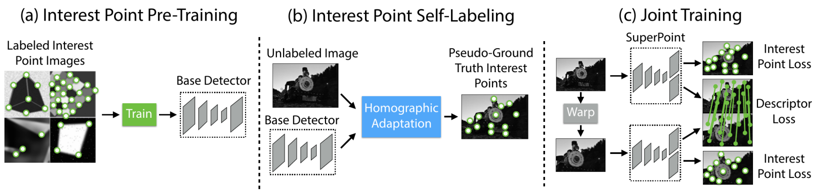


Figure: The SuperPoint learning framework

Homographic Adaption for Self-Supervision

- The network is trained further – under self-supervision — with the help of Homographic Adaptation.

- Homographic Adaptation consist of further training the network with pairs of images, (I, I') that are related by $\mathbf{x}' = \mathcal{H}\mathbf{x}$, $\mathbf{x} \in I$ and $\mathbf{x}' \in I'$, for *known* but randomly chosen 3×3 homographies $\mathcal{H}$. For a given pair (I, I'), assume that I belongs to the training dataset. During training, each dataset image I may be subject to multiple randomly chosen homographies $\mathcal{H}$.

- When I is at the input to the network, we obviously know from its associated annotation where its keypoints are located. In this case, the cross-entropy loss at the output of the network is based on a comparison of the output with the information derived from the annotation for the localization of the keypoints.

- But what about the descriptor vectors? See the next slide.

Homographic Adaption for Self-Supervision (contd.)

- For calculating the loss related to the descriptor vectors, we must next place the second image in the pair, I', at the input. As far as the keypoint localization is concerned, we can again compare the output of the network with the new locations for the keypoints as given by the transformations $\mathbf{x}' = \mathcal{H} \cdot \mathbf{x}$.
- In addition, we can now also estimate a loss related to the quality of predictions for the descriptor vectors as explained in what follows.
- If $\mathbf{d}$ is the descriptor vector for a given keypoint in the image I and $\mathbf{d}'$ the descriptor vector for the *corresponding keypoint* in I', we require that the two descriptor vectors be mutually consistent by insisting that the dot-product $\mathbf{d}^T\mathbf{d}$ be close to 1. It is this logic that is used to construct a loss function for estimating the descriptor vectors.

Homographic Adaption for Self-Supervision (contd.)

- In summary, further training of the network consists of: (1) Having the network in its current state identify the keypoints **in real-world images**; (2) Subject the images to several randomly chosen homographies that significantly increase the "complexity" of the images. And, (3) while training the network with pairs of images (I, I'), require that the predicted descriptor vectors for the corresponding pairs of keypoints in the two images be mutually consistent.

Why SIFT, SURF, etc., Cannot be Used for Training a Neural Keypoint Detector

- The most straightforward way to train a neural network is to present at its output the groundtruth for what it is you are looking for in the input image. But that approach would be impossible if you want your neural network to not only predict the pixel locations of the keypoints **but also their associated descriptor vectors.**

- If you had to use a traditional approach, you could try to generate your training data by applying, say, the SIFT operator to a large number of images and using the images along with the SIFT-discovered keypoints and descriptor vectors as the training data. Unfortunately, this idea suffers from a basic problem mentioned below.

- The descriptor vectors calculated by keypoint operators like SIFT, SURF, etc., are essentially normalized histograms for the pixels in the vicinity of a keypoints. Since a histogram is not differentiable directly, it CANNOT be used for neural learning.

Outline

The SuperPoint Architecture

- As is common for the neural networks designed for image related tasks (classification, object detection, segmentation, etc.), the first part of the network, labeled as **Encoder** in the figure on the next slide, makes the network aware of the relationships between the neighboring pixels.
- This is accomplished by progressively reducing the size of the images but without loss of information.
- As the image size is reduced, more and more information is projected into the channels associated with the image. While an input image shrinks in its (x,y)-size from layer to layer, whatever information may appear to be lost spatially is encoded in the channel dimension that is orthogonal to the (x,y)-plane.
- In our case, in the Encoder part of the network, the information that is projected into the channel dimension at a pixel is related to the pixel's properties as a keypoint.

The SuperPoint Architecture (contd.)

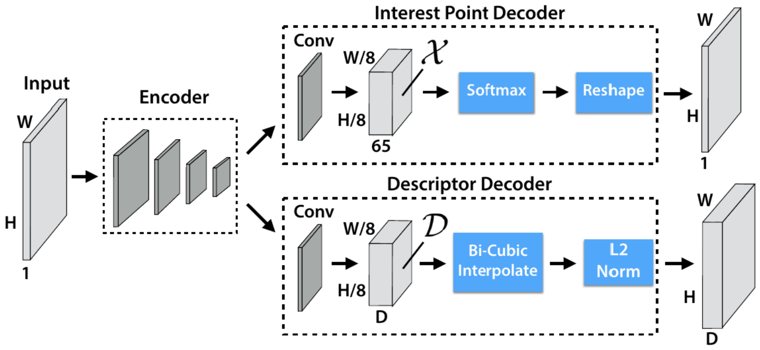


Figure: The SuperPoint Network

The SuperPoint Architecture (contd.)

- Since a keypoint at a pixel at **x** is defined by the relationships between the pixels in a blob around **x**, it would be reasonable to assume that, with proper training, the vector of the values in all the channels at **x** would tell us whether or not that point is a keypoint at **x**.
- SuperPoint uses 64 channels for the final layer of the Encoder.
- Given the memory capacity of 64 real numbers along the channel dimension, it would also be reasonable to assume that the same 64 numbers would also contain information related to the Descriptor Vector at the pixel — in the event that it is a keypoint.
- In general, in the design of neural networks, the two assumptions made above (in blue) would lead to the following sort of questions: (1) What should be the progression of image-size reduction and channel expansion as we go from layer to layer in the Encoder; (2) What should be the size of the image and how many layers to use in the final layer of the Encoder; etc.

The SuperPoint Architecture (contd.)

- In deep learning, in general, answers to the sort of question listed in the last bullet on the previous slide are found by experimentation. That is, there does not exist a theory that would guide you with regard to finding answers to those questions.

Outline

The SuperPoint Encoder

- In the Encoder part of the SuperPoint network, an image of size $H \times W$ is reduced to the size $\frac{H}{8} \times \frac{W}{8}$ as the number of channels increases from 3 to 65. As you would expect, the input images have 3 channels for the RGB color planes.
- As a result of the reduction in size by a factor of 8, each 8×8 patch of pixels in the input image is represented by one pixel in the final layer of the Encoder.
- The numerical values in the channels at the output of the Encoder may be *directly* thought of as the probability that the corresponding 8×8 patch in the input image contains a keypoint. We will refer to the pixels in the final layer of the Encoder as cells, with each cell representing an 8×8 patch of the input image.
- While admitting of a probability interpretation, one can also hope that the 65 numerical values along the channel dimension for a cell would lend themselves to the construction of a descriptor vector with further training.

The SuperPoint Encoder (contd.)

- There is a very important reason for why the SuperPoint encoder uses 65 channels at the output of Encoder: When a pixel is accepted as a keypoint, 64 of those 65 values are meant to be used to eventually lead to: (a) the exact pixel in the 8×8 patch that is a keypoint in the patch; and (b) a descriptor vector for the keypoint. One of the channels, referred to as the *dustbin*, is used as an indicator for whether or not the cell supports a keypoint.

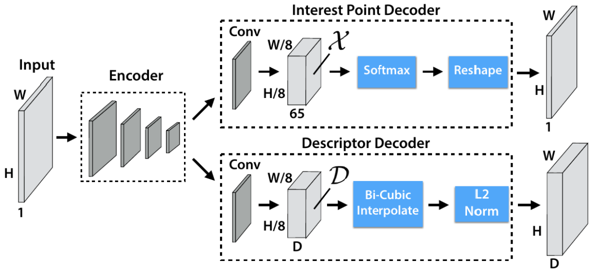


Figure: The SuperPoint Network (again)

The SuperPoint Encoder (contd.)

- Presented below is a more precise interpretation of the 65 channel values at the output of the Encoder. This interpretation is particularly relevant to the upper arm of the Decoder shown in the figure on the previous slide.
- Of the 65 channels for a cell, we use 64 channels to encode the exact location of a keypoint in the 8×8 patch of the input. Obviously, an 8×8 patch has 64 possible locations for a keypoint. As long as we use the same mapping rule for all the images in the training dataset, we can uniquely map each pixel location in an 8×8 patch to one of the 64 channels in the output of the encoder.
- What gives credibility to the above interpretation of the channels is how we specify the loss function for keypoint localization in the upper arm of the Decoder.

The SuperPoint Encoder (contd.)

- To further elaborate the role of the *dustbin* channel, when there is actually a keypoint in the corresponding 8×8 patch of the input, we want the probability mass in the channels values to focus primarily on the channel that represents the exactly location of the keypoint.

- On the other hand, when there does NOT exist a keypoint in the 8×8 patch, we want the probability mass to shift to the dustbin channel. Therefore, a high value in the dustbin channel means no keypoint in the corresponding 8×8 patch.

- This interpretation of the high and low values in the dustbin channel is necessitated by the constraints imposed by how the loss is calculated for the errors in keypoint localization. As you will see in the next section, that loss uses the Cross-Entropy criterion that requires a probabilistic interpretation of the channel values through what's known as the *softmax* operation. What that means is that the total probability mass in the 65 channels values must add up to 1 for all cases, when a keypoint is present and when it absent.

Outline

The Decoder

- As shown in network figure on Slide 28, the SuperPoint Decoder is double-headed: the head in the upper arm of the Decoder is devoted to predicting whether a pixel is a keypoint, whereas the head in the lower arm is for estimating the Descriptor Vector.
- As you know already from the Encoder discussion, the purpose of the upper head is to tell which exact pixel in the 8×8 patch corresponding to a cell is the keypoint — assuming one is there. If none of the pixels is a keypoint, the probability mass among the 65 channel values must shift to the dustbin channel as explained earlier.
- In this section, I'll first add to what you already know (from the previous section) about the localization of a keypoint in the upper arm of the Decoder. Subsequently, I will address how the descriptor vectors are estimated by SuperPoint in the lower arm of the Decoder. While the SuperPoint network overall is highly innovative, how the Descriptor Vectors are calculated takes the cake (in the good sense of the phrase).

Decoder Logic for Keypoint Localization

- If you have done deep-learning programming, you will recognize the $\frac{H}{8} \times \frac{W}{8} \times 65$ data object shown as $\mathcal{X}$ in the upper arm of the Decoder as a tensor that represents the output of the Encoder. The 65 values are the channel values at each of the cells in the $\frac{H}{8} \times \frac{W}{8}$ array that is input into the upper arm of the Decoder.

- As mentioned earlier, one of the 65 values stands for the *dustbin* and the other 64 to the channel-wise encoding of the pixel positions in the 8×8 patch of the input image.

- The question we must address now is how to evaluate the the values in the 65 channels in the upper arm of the Decoder?

- The developers of the SuperPoint network have used the Cross-Entropy loss to evaluate all 65 channels of the tensor $\mathcal{X}$ shown in the figure.

Decoder Logic for Keypoint Localization (contd.)

- To continue with the thought in the last bullet on the previous slide, an alternative would be to use the Binary Cross Entropy Loss (BCELoss) for the dustbin channel — since its job is to serve as an indicator for the presence or the absence of a keypoint in the 8×8 patch. However, as a part of the Cross-Entropy loss applied to all 65 channels, the end result would be the same.

- The 65 values are subject to the following Softmax activation that turns them into a valid probability distribution:

$$Softmax(x_i) \;=\; \frac{e^{x_i}}{\sum_{j=1}^{64} e^{x_j}} \qquad i = 1, 2, \ldots, 65$$

where each x_i is a value in one the 65 channels. [See the discussion on Cross Entropy in the Slides for my Week 7 lecture for Purdue's Deep Learning class (ECE 60146 and BME 646) that will be taught again in Spring 2025. You will see there a detailed review of Cross-Entropy Loss and Softmax activation.]

Decoder Logic for Keypoint Localization (contd.)

- These 65 values are then subject to Cross Entropy Loss calculation. What that amounts to is that if k-th channel value corresponds to where the keypoint is in the input image, we set the output of the loss function to

$$-\log_2\left(\frac{e^{x_k}}{\sum_{j=1}^{65} e^{x_j}}\right)$$

- The final step in the upper arm of the Decoder is mapping the 64 channels that encode the exact location of a keypoint in the 8×8 patch back to the exact pixel coordinates in the original $H \times W$ image by using what's known as pixel shuffling.

- As the PyTorch documentation page says, pixel shuffling entails mapping from a tensor of shape $(B, C \times r^2, h, w)$ into a tensor of shape $(B, C, h \times r, w \times r)$. In this notation, B stands for the batch size, C for the number of channels, and h and w for the 'height' and the 'width' of the tensor.

Outline

Decoder Head for Descriptor Vector Prediction

- The lower arm of the Decoder is for estimating 256-element Descriptor Vectors (DV) for the keypoints.

- The DVs are estimated semi-densely, meaning one DV per cell in the output of the Encoder.

- The alternative would be to estimate them densely, that is, at every pixel. But that would be computationally too expensive and not needed anyway. [**Remember, the classical keypoints (SIFT, SURF, etc.) are situated at local extrema. What that implies is that each local neighborhood will support at most a single keypoint. For obvious reasons, the size of this local neighborhood is specific to each different keypoint detector.**]

- Learning of the DVs starts out by assuming that they are random vectors in each cell of the $\frac{H}{8} \times \frac{W}{8}$ array at the output of the Encoder. The tensor $\frac{H}{8} \times \frac{W}{8} \times \mathcal{D}$ in the lower arm of the Decoder stores these random vector. The value of $\mathcal{D}$ is 256 in SuperPoint.

Descriptor Vector Prediction (contd.)

- To understand how the DV-related learning takes place, please review the material presented on Slides 17 and 18 of this slide deck.
- As stated in the earlier material, let I and I' be the image pair related by a randomly chosen homography $\mathcal{H}$. We again assume that I is an image from the training dataset and that the image I' is obtained by applying $\mathcal{H}$ to I.
- As mentioned earlier on Slides 17 and 18, in order to train the network for DV prediction, the training is based on feeding the (I, I') image pairs into the network, one pair at a time, but feeding in the two images sequentially as explained in what follows.
- That is, you first feed I into the network and estimate the keypoint localization errors by calculating the cross-entropy loss as explained earlier. It is easy to estimate this loss since you are comparing the output with known keypoint coordinates in the annotation associated with the image.

Descriptor Vector Prediction (contd.)

- To continue the last bullet on the previous slide, for estimating the loss related to the prediction of the DVs, you hold off until you have also presented the image I' to the network as its next input.
- When you feed the image I' into the network, what happens in the upper arm of the decoder is no different from what was done for the I input — except for the fact that the true keypoint locations in I' are obtained on the fly by applying $\mathcal{H}$ to the keypoint coordinates in I.
- With the information you have for the lower arm of the Decoder for both I and I', you are now ready to measure the loss that is relevant to DV learning. This loss is based on the mutual consistency of the corresponding DVs in the $\frac{H}{8} \times \frac{W}{8} \times \mathcal{D}$ tensor in the lower arm of the Decoder.
- To elaborate, the homography $\mathcal{H}$ will move the keypoint coordinates $\mathbf{p} = (x, y)$ in image I to a new location $\mathbf{p}' = (x', y')$ in the image I'.

Descriptor Vector Prediction (contd.)

- To continue the last bullet on the previous slide, I'll use the notation $cell_p$ and $cell_{p'}$ to refer to the cells in the two outputs of the Encoder that correspond to the pixel coordinates $\mathbf{p}$ and $\mathbf{p}'$.
- When we pass the first image, I, through the network, let the DV stored for the $cell_p$ in the $\frac{H}{8} \times \frac{W}{8} \times \mathcal{D}$ tensor be $\mathbf{d}$. And when we pass I' through the network, let the DV stored in the same tensor at the $cell_{p'}$ be $\mathbf{d}'$.
- You would want the DVs $\mathbf{d}$ and $\mathbf{d}'$ to be mutually consistent since, after all, they correspond to the same keypoint in the original image.
- If the neural network is already well trained and working perfectly, the DV vectors $\mathbf{d}$ and $\mathbf{d}'$ will both be high and mutually consistent, meaning that their dot-product will also be large. To the extent that is not the case, we have DV loss.

Descriptor Vector Prediction (contd.)

- For DV learning, our goal then becomes one of maximization of $\mathbf{d}^T\mathbf{d}'$ products for $(\mathbf{d}, \mathbf{d}')$ pairs that belong to the corresponding cells, the correspondences being induces by the known homography $\mathcal{H}$ for an image pair (I, I') and the minimization of the $\mathbf{d}^T\mathbf{d}'$ products for the case when the such cell-to-cell correspondences do not apply.

- By maximization, I mean that the products $\mathbf{d}^T\mathbf{d}'$ should be as close to $+1$ as possible. And, by minimization, the same products should be as close to -1 as possible.