Detektor ključnih točk

Poročilo

In [1]:
```
%matplotlib inline
import matplotlib.pyplot as plt
import numpy as np
import scipy.ndimage as ndimage
import torch
```

Primeri slik

Generator naključno vrača sitnetične vzorce in njihov pričakovano predkcijo.

Vsak vzorec ima kot ozadje nizko filtriran Gaussovo šum.

Sintetični vzorci vsebujejo:

- poligone z 3, 4, 5 in 6 točkami
- zvezde
- šahovnice

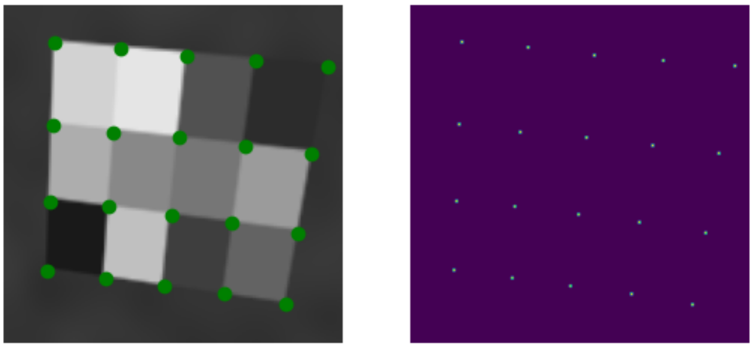
Vsak vzorec je modificiran z naključno homografijo. Vsi vzorci so črno-beli.

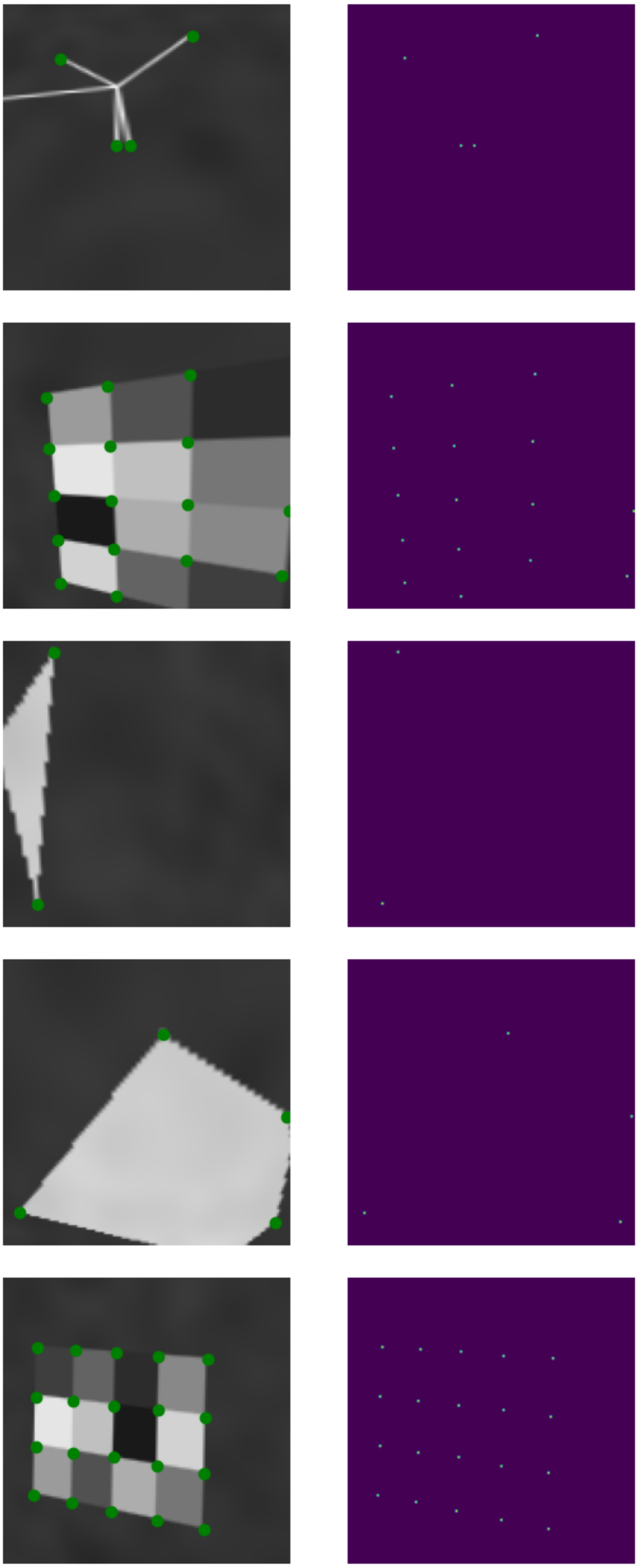
V tem delu naloge manjka:

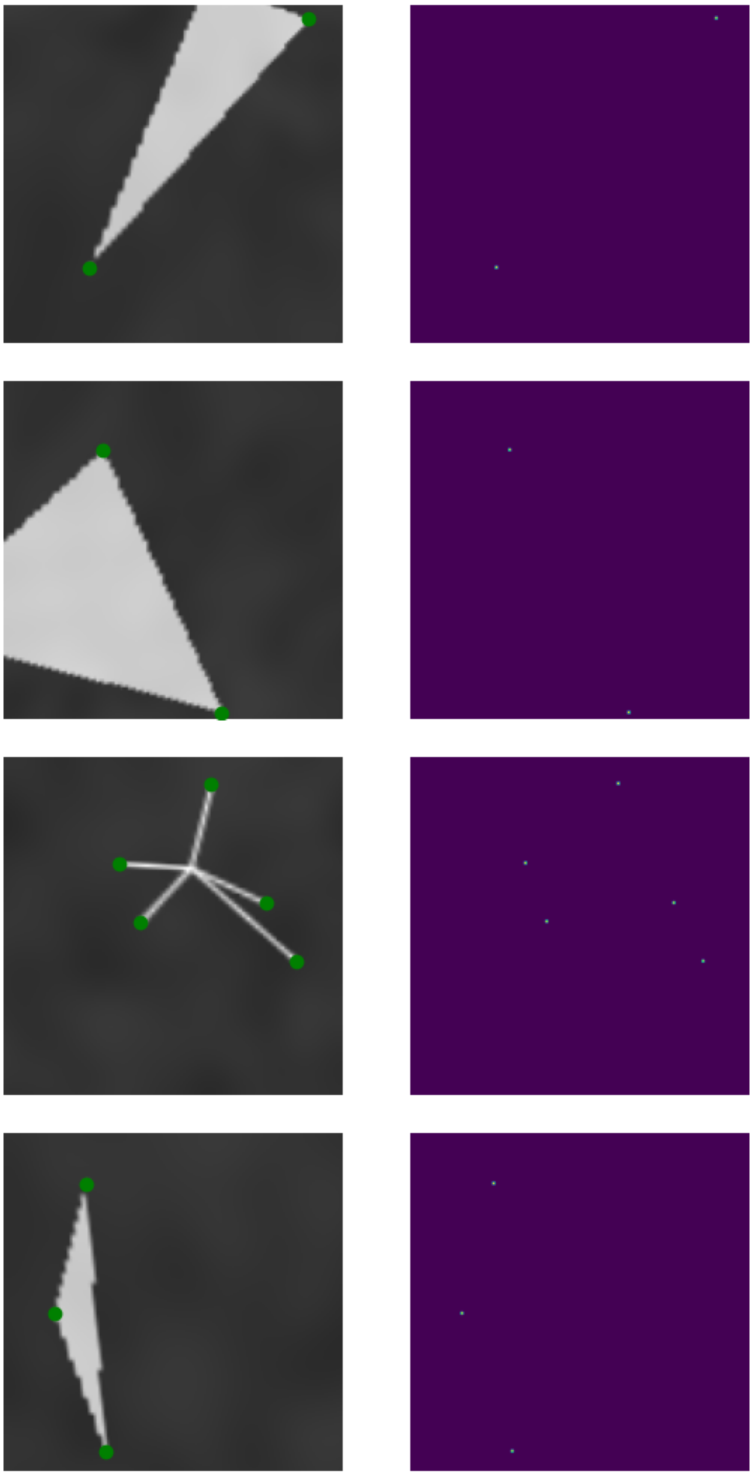
- **generiranje 3D kocke**
- **generiranje večih likov v eni sliki**
- **brez barv**

In [2]:
```
from generator_podatkov import data_generator_function

data_gen = data_generator_function(1, (128, 128), color=True)
```

In [3]:
```
for i in range(10):
    img, target = next(data_gen)
    _, _, H_t, W_t = target.shape
    img=img[0].transpose(1, 2, 0)
    target = target[0, :-1, :, :].reshape(8, 8, H_t, W_t)\
            .transpose(2, 0, 3, 1)\
            .reshape(8*H_t, 8*W_t)

    py, px= np.where(target>0)

    fig, ax = plt.subplots(1, 2, sharex=True, sharey=True)
    ax[0].imshow(img, cmap='gray')
    ax[0].plot(px, py, 'og')
    ax[0].set_axis_off()
    ax[1].imshow(target)
    ax[1].set_axis_off()
```







Izpis nevronske mreže

In [4]:
```
device = torch.device('cpu')
from mreza import *
kp_model = torch.load('keypoint_detector_4.pt', map_location=device)
```

In [5]:
```
import torchinfo
torchinfo.summary(kp_model, input_size=(1,1,256,256))
```

Out[5]:
```
==========================================================================================
Layer (type:depth-idx)                   Output Shape              Param #
==========================================================================================
KeypointDetectionNet                     [1, 65, 32, 32]           --
├─FeatureMap: 1-1                        [1, 128, 32, 32]          --
│    └─ResNetBlock: 2-1                  [1, 64, 256, 256]         --
│    │    └─Conv2d: 3-1                  [1, 64, 256, 256]         640
│    │    └─BatchNorm2d: 3-2             [1, 64, 256, 256]         128
│    │    └─ReLU: 3-3                    [1, 64, 256, 256]         --
│    │    └─Conv2d: 3-4                  [1, 64, 256, 256]         36,928
│    │    └─BatchNorm2d: 3-5             [1, 64, 256, 256]         128
│    │    └─Conv2d: 3-6                  [1, 64, 256, 256]         128
│    │    └─ReLU: 3-7                    [1, 64, 256, 256]         --
│    └─ResNetBlock: 2-2                  [1, 64, 256, 256]         --
│    │    └─Conv2d: 3-8                  [1, 64, 256, 256]         36,928
│    │    └─BatchNorm2d: 3-9             [1, 64, 256, 256]         128
│    │    └─ReLU: 3-10                   [1, 64, 256, 256]         --
│    │    └─Conv2d: 3-11                 [1, 64, 256, 256]         36,928
│    │    └─BatchNorm2d: 3-12            [1, 64, 256, 256]         128
│    │    └─ReLU: 3-13                   [1, 64, 256, 256]         --
│    └─MaxPool2d: 2-3                    [1, 64, 128, 128]         --
│    └─ResNetBlock: 2-4                  [1, 64, 128, 128]         --
│    │    └─Conv2d: 3-14                 [1, 64, 128, 128]         36,928
│    │    └─BatchNorm2d: 3-15            [1, 64, 128, 128]         128
│    │    └─ReLU: 3-16                   [1, 64, 128, 128]         --
│    │    └─Conv2d: 3-17                 [1, 64, 128, 128]         36,928
│    │    └─BatchNorm2d: 3-18            [1, 64, 128, 128]         128
│    │    └─ReLU: 3-19                   [1, 64, 128, 128]         --
│    └─ResNetBlock: 2-5                  [1, 64, 128, 128]         --
│    │    └─Conv2d: 3-20                 [1, 64, 128, 128]         36,928
│    │    └─BatchNorm2d: 3-21            [1, 64, 128, 128]         128
│    │    └─ReLU: 3-22                   [1, 64, 128, 128]         --
│    │    └─Conv2d: 3-23                 [1, 64, 128, 128]         36,928
│    │    └─BatchNorm2d: 3-24            [1, 64, 128, 128]         128
│    │    └─ReLU: 3-25                   [1, 64, 128, 128]         --
│    └─MaxPool2d: 2-6                    [1, 64, 64, 64]           --
│    └─ResNetBlock: 2-7                  [1, 128, 64, 64]          --
│    │    └─Conv2d: 3-26                 [1, 128, 64, 64]          73,856
│    │    └─BatchNorm2d: 3-27            [1, 128, 64, 64]          256
│    │    └─ReLU: 3-28                   [1, 128, 64, 64]          --
│    │    └─Conv2d: 3-29                 [1, 128, 64, 64]          147,584
│    │    └─BatchNorm2d: 3-30            [1, 128, 64, 64]          256
│    │    └─Conv2d: 3-31                 [1, 128, 64, 64]          8,320
│    │    └─ReLU: 3-32                   [1, 128, 64, 64]          --
│    └─ResNetBlock: 2-8                  [1, 128, 64, 64]          --
│    │    └─Conv2d: 3-33                 [1, 128, 64, 64]          147,584
│    │    └─BatchNorm2d: 3-34            [1, 128, 64, 64]          256
│    │    └─ReLU: 3-35                   [1, 128, 64, 64]          --
│    │    └─Conv2d: 3-36                 [1, 128, 64, 64]          147,584
│    │    └─BatchNorm2d: 3-37            [1, 128, 64, 64]          256
│    │    └─ReLU: 3-38                   [1, 128, 64, 64]          --
│    └─MaxPool2d: 2-9                    [1, 128, 32, 32]          --
│    └─ResNetBlock: 2-10                 [1, 128, 32, 32]          --
│    │    └─Conv2d: 3-39                 [1, 128, 32, 32]          147,584
│    │    └─BatchNorm2d: 3-40            [1, 128, 32, 32]          256
│    │    └─ReLU: 3-41                   [1, 128, 32, 32]          --
│    │    └─Conv2d: 3-42                 [1, 128, 32, 32]          147,584
│    │    └─BatchNorm2d: 3-43            [1, 128, 32, 32]          256
│    │    └─ReLU: 3-44                   [1, 128, 32, 32]          --
│    └─ResNetBlock: 2-11                 [1, 128, 32, 32]          --
│    │    └─Conv2d: 3-45                 [1, 128, 32, 32]          147,584
│    │    └─BatchNorm2d: 3-46            [1, 128, 32, 32]          256
│    │    └─ReLU: 3-47                   [1, 128, 32, 32]          --
│    │    └─Conv2d: 3-48                 [1, 128, 32, 32]          147,584
│    │    └─BatchNorm2d: 3-49            [1, 128, 32, 32]          256
│    │    └─ReLU: 3-50                   [1, 128, 32, 32]          --
├─Conv2d: 1-2                            [1, 256, 32, 32]          295,168
├─ReLU: 1-3                              [1, 256, 32, 32]          --
├─Conv2d: 1-4                            [1, 65, 32, 32]           16,705
==========================================================================================
Total params: 1,689,473
Trainable params: 1,689,473
Non-trainable params: 0
Total mult-adds (G): 12.80
==========================================================================================
Input size (MB): 0.26
Forward/backward pass size (MB): 417.87
Params size (MB): 6.76
Estimated Total Size (MB): 424.89
==========================================================================================
```

Rezultati

Sitnetične slike

V teh slikah so detekcije večinoma dobre.

Iz generatorja bomo vzeli 10 vzorcev in za njih opravili predikcijo.

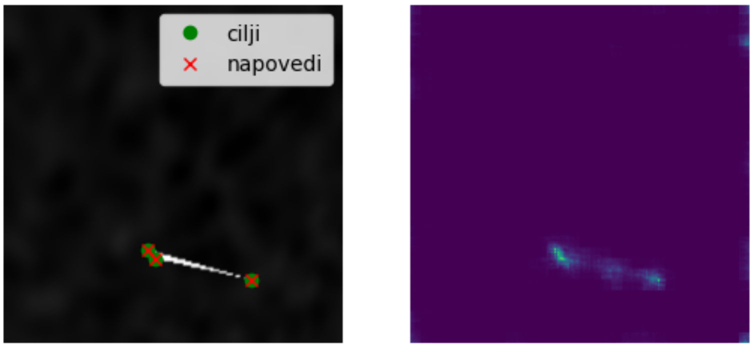
Predikciji odrežemo zadnji kanal (ki pomeni, da ni detekcije) in jo preoblikujemo v sliko enakih dimenzij kot vhodni vzorec.

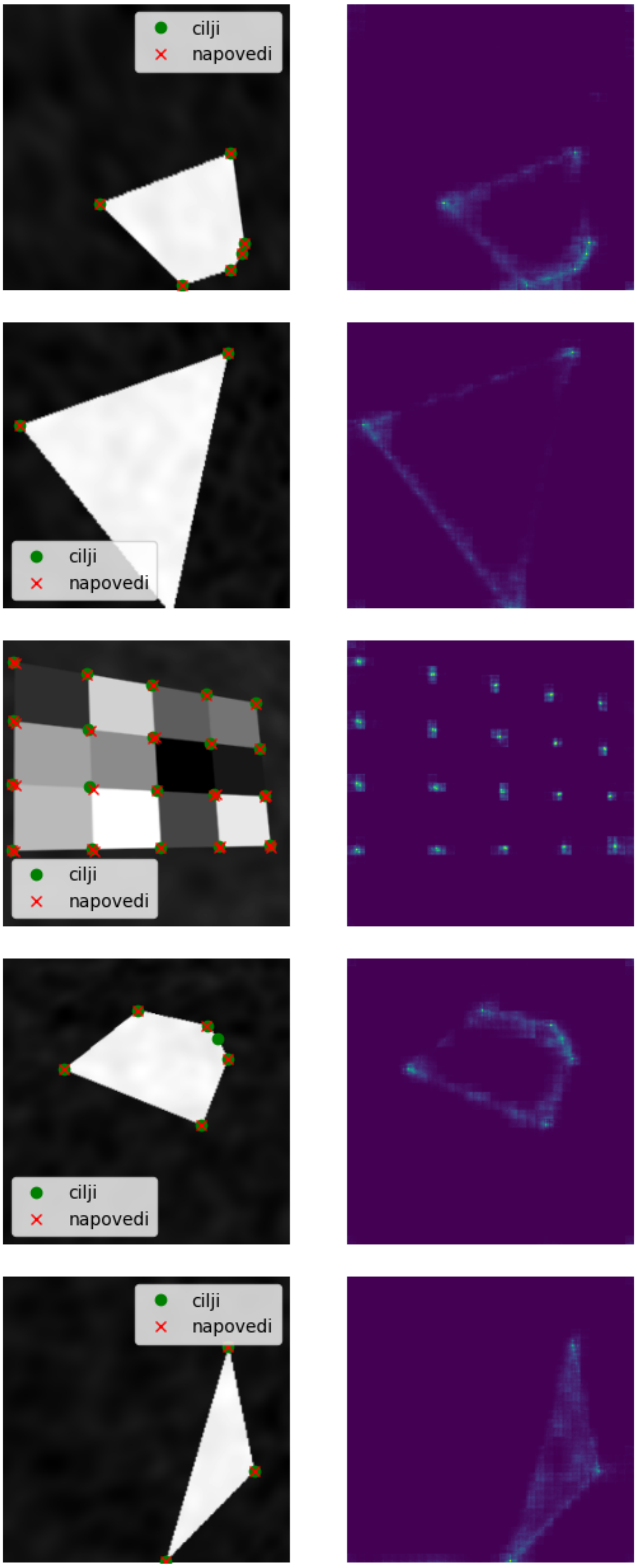
V sliki predikcij moramo poiskati lokalne maksimume, ki so večji od nekega pragu (pravzaprav bi morali preveriti, ali je lokalni maksimum višji od vrednosti zadnjega kanala v ustrezni celici). To naredimo s pomočjo sivinske morfološke operacije širjenja.

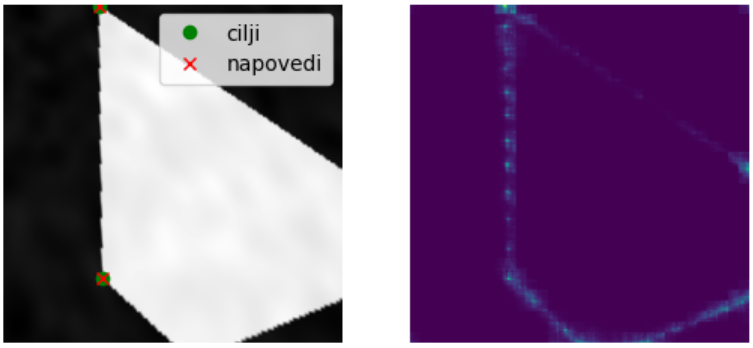
Detektirane točke nato narišemo preko vzorca v levem stolpcu slik. V desnem stolpcu pa narišemo predikcijo mreže, kot sliko. Nizke vrednosti predikcije ojačamo z potenciranjem na 0.1.

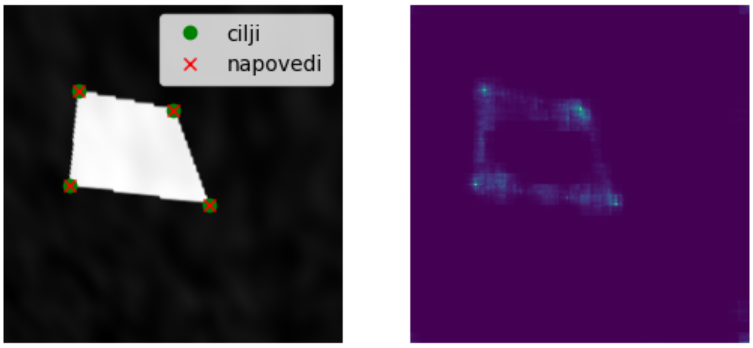
In [6]:
```
data_gen = data_generator_function(1, (256, 256), color=False)
```

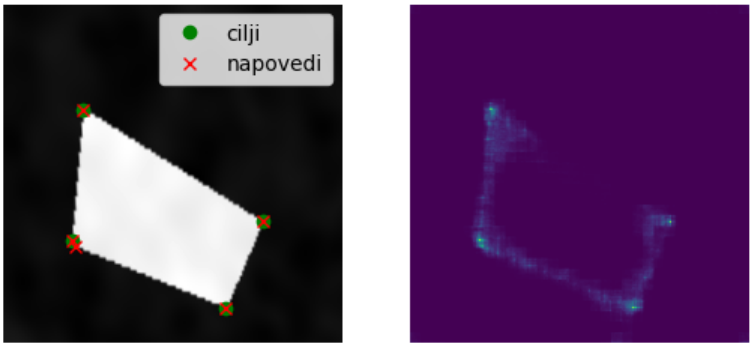
In [7]:
```
for i in range(10):
    img, target = next(data_gen)
    pred = kp_model(torch.from_numpy(img))
    pred = torch.nn.functional.softmax(pred, dim=1)
    pred = pred.detach().numpy()

    # pretvori cilj
    _, _, H_t, W_t = target.shape
    target = target[0, :-1, :, :].reshape(8, 8, H_t, W_t)\
            .transpose(2, 0, 3, 1)\
            .reshape(8*H_t, 8*W_t)

    py_tar, px_tar = np.where(target>0)

    # pretvori napoved
    _, _, H_t, W_t = pred.shape
    img=img[0].transpose(1, 2, 0)
    pred = pred[0, :-1, :, :].reshape(8, 8, H_t, W_t)\
            .transpose(2, 0, 3, 1)\
            .reshape(8*H_t, 8*W_t)

    # najdi lokalne maksimume predikcije, vecje od minimalne vrednosti
    local_max = (pred==ndimage.grey_dilation(pred, size=3))*pred>0.2
    py_est, px_est = np.where(local_max)


    fig, ax = plt.subplots(1, 2)
    ax[0].imshow(img, cmap='gray')
    ax[0].plot(px_tar, py_tar, 'go', label='cilji')
    ax[0].plot(px_est, py_est, 'rx', label='napovedi')
    ax[0].set_axis_off()
    ax[0].legend()
    ax[1].imshow(pred**0.1)
    ax[1].set_axis_off()
```

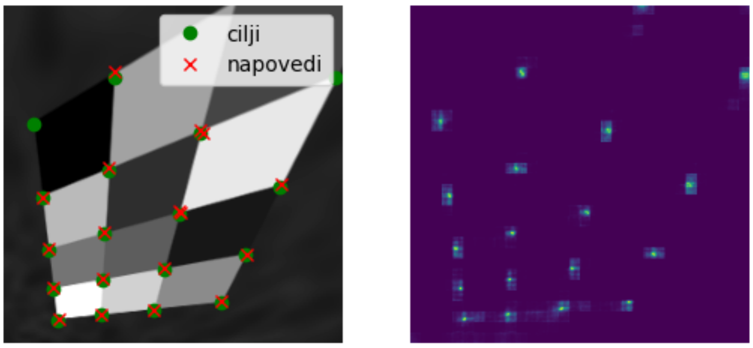



cilji
napovedi
cilji
napovedi
cilji
napovedi
cilji
napovedi
cilji
napovedi









Slike zbirke BSD (Berkeley Segmentation Dataset)

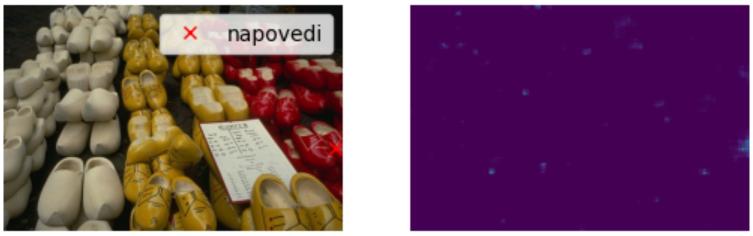
Za primerjavo naključno izberemo 10 slik iz BSD zbirke in prikažemo rezultate.

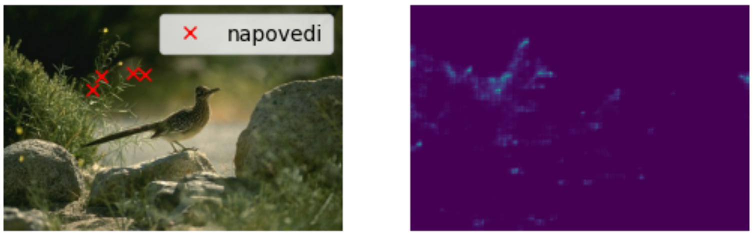
Predikcije tukaj so precej šibkejše. Da najdemo nekaj lokalnih maksimumov, sliko predikcije najprej normaliziramo.

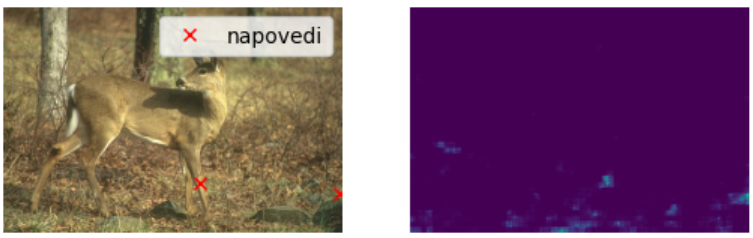
In [8]:
```
import pathlib
```

In [9]:
```
bsd_path = pathlib.Path('../../N1_ucenje_homografije/BSR/BSDS500/data/images/train/')
img_list = [*bsd_path.glob('*.jpg')]
np.random.shuffle(img_list)
```

In [10]:
```python
for n in range(10):
    img = plt.imread(img_list[n])
    if img.ndim==3:
        img_gray = img.mean(2)
    else:
        img_gray = img
    img_torch = torch.from_numpy(np.float32(img_gray/255.)[np.newaxis, np.newaxis, ...])
    pred = kp_model(img_torch)
    pred = torch.nn.functional.softmax(pred, dim=1)
    pred = pred.detach().numpy()


    # pretvori napoved
    _, _, H_t, W_t = pred.shape
    pred = pred[0, :-1, :, :].reshape(8, 8, H_t, W_t)\
            .transpose(2, 0, 3, 1)\
            .reshape(8*H_t, 8*W_t)


    pred /= pred.max()

    # najdi lokalne maksimume predikcije, vecje od minimalne vrednosti
    local_max = (pred==ndimage.grey_dilation(pred, size=3))*pred>0.1
    py_est, px_est = np.where(local_max)

    fig, ax = plt.subplots(1, 2)
    ax[0].imshow(img, cmap='gray')
    ax[0].plot(px_est, py_est, 'rx', label='napovedi')
    ax[0].set_axis_off()
    ax[0].legend()
    ax[1].imshow(pred**0.1)
    ax[1].set_axis_off()
```








napovedi
napovedi
napovedi
napovedi
napovedi

Rezultati homografske adaptacije

Za vsako sliko v BSD zbirki ponovimo detekcijo 100x.

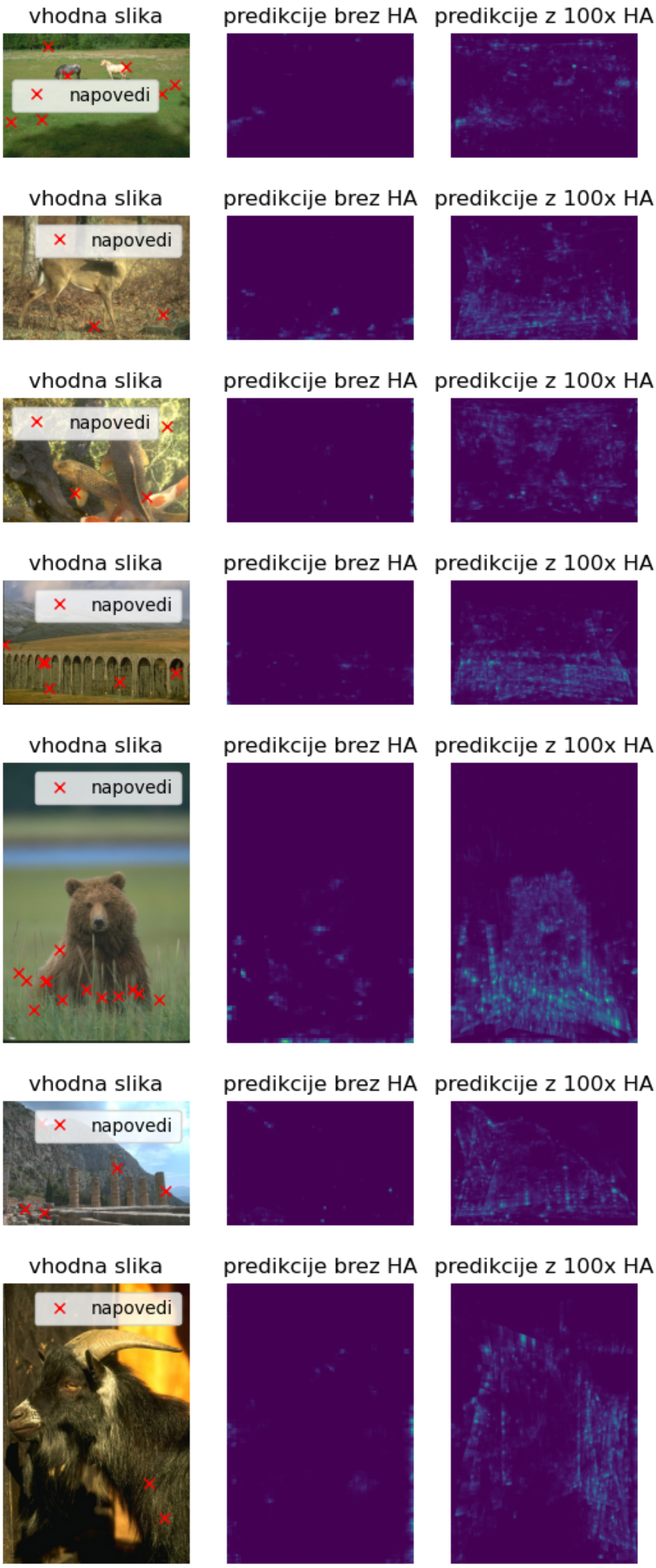
Za vsako ponovitev sliko transformiramo z naključno homografijo. Predikcijo mreže nato poravnamo nazaj z originalno sliko in seštejemo.

Predikcijo ponovno normaliziramo in poiščemo lokalne maksimume.

V desnem stolpcu so prikazane akumulirane predikcije mreže z ojačanimi temnimi vrednostmi. Če te primerjamo s predikcijami brez homografske adaptacije na sredini (rezultat prve detekcije) vidimo, da so točke ponekod jasneje detektirane, pojavijo se tudi detekcije, ki jih brez adaptacije ne dobimo. Vendar pa so same vrednosti predikcije še vedno zelo nizke, morda celo nižje z homografsko adaptacijo.

In [11]:
```
from generator_podatkov import homography_adapt, homography_adapt_inv
```

In [12]:
```python
for n in range(10):
    img = plt.imread(img_list[n])
    if img.ndim==3:
        img_gray = img.mean(2)
    else:
        img_gray = img

    # prva detekcija, brez homografske adaptacije
    img_torch = torch.from_numpy(np.float32(img_gray/255.)[np.newaxis, np.newaxis, ...])
    pred = kp_model(img_torch)
    pred = torch.nn.functional.softmax(pred, dim=1)
    pred = pred.detach().numpy()
    _, _, H_t, W_t = pred.shape
    pred_cum = pred[0, :-1, :, :].reshape(8, 8, H_t, W_t)\
            .transpose(2, 0, 3, 1)\
            .reshape(8*H_t, 8*W_t).copy()

    pred_no_ha = pred_cum.copy()

    for _ in range(99):
        # ostale detekcije, vsaka z homografsko adaptacijo - naključno homografsko transformacijo

        # transformacija slike, pridobitev transformacijske matrike
        img_ha, _, H = homography_adapt(img_gray, np.zeros((4, 2)))

        img_torch = torch.from_numpy(np.float32(img_ha/255.)[np.newaxis, np.newaxis, ...])
        pred = kp_model(img_torch)
        # pretvorba predikcije v sliko
        pred = torch.nn.functional.softmax(pred, dim=1)
        pred = pred.detach().numpy()
        _, _, H_t, W_t = pred.shape
        pred = pred[0, :-1, :, :].reshape(8, 8, H_t, W_t)\
                .transpose(2, 0, 3, 1)\
                .reshape(8*H_t, 8*W_t)

        # poravnava predikcije z originalno sliko
        pred_inv = homography_adapt_inv(pred, H)

        # akumulacija predikcij
        pred_cum += pred_inv

    # odstranimo morebitne nan vrednosti, normaliziramo
    pred = np.nan_to_num(pred_cum)
    pred /= pred.max()

    # najdi lokalne maksimume predikcije, vecje od rocno izbranega minimalnega pragu
    local_max = (pred==ndimage.grey_dilation(pred, size=3))*pred>0.1
    py_est, px_est = np.where(local_max)

    fig, ax = plt.subplots(1, 3)
    ax[0].imshow(img, cmap='gray')
    ax[0].plot(px_est, py_est, 'rx', label='napovedi')
    ax[0].set_axis_off()
    ax[0].set_title('vhodna slika')
    ax[0].legend()
    ax[1].imshow(pred_no_ha**0.1)
    ax[1].set_title('predikcije brez HA')
    ax[1].set_axis_off()
    ax[2].imshow(pred**0.1)
    ax[2].set_title('predikcije z 100x HA')
    ax[2].set_axis_off()
```


vhodna slika
predikcije brez HA
predikcije z 100x HA
napovedi
vhodna slika
predikcije brez HA
predikcije z 100x HA
napovedi
vhodna slika
predikcije brez HA
predikcije z 100x HA
napovedi
vhodna slika
predikcije brez HA
predikcije z 100x HA
napovedi
vhodna slika
predikcije brez HA
predikcije z 100x HA
napovedi
vhodna slika
predikcije brez HA
predikcije z 100x HA
napovedi
vhodna slika
predikcije brez HA
predikcije z 100x HA
napovedi


vhodna slika
napovedi
predikcije brez HA
predikcije z 100x HA

In [ ]: