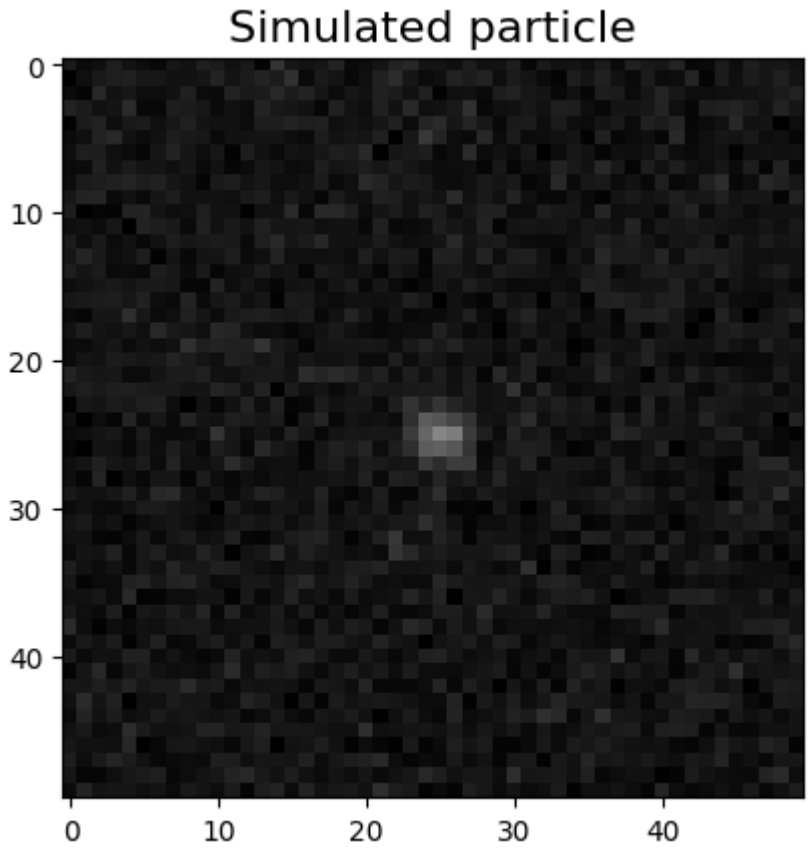
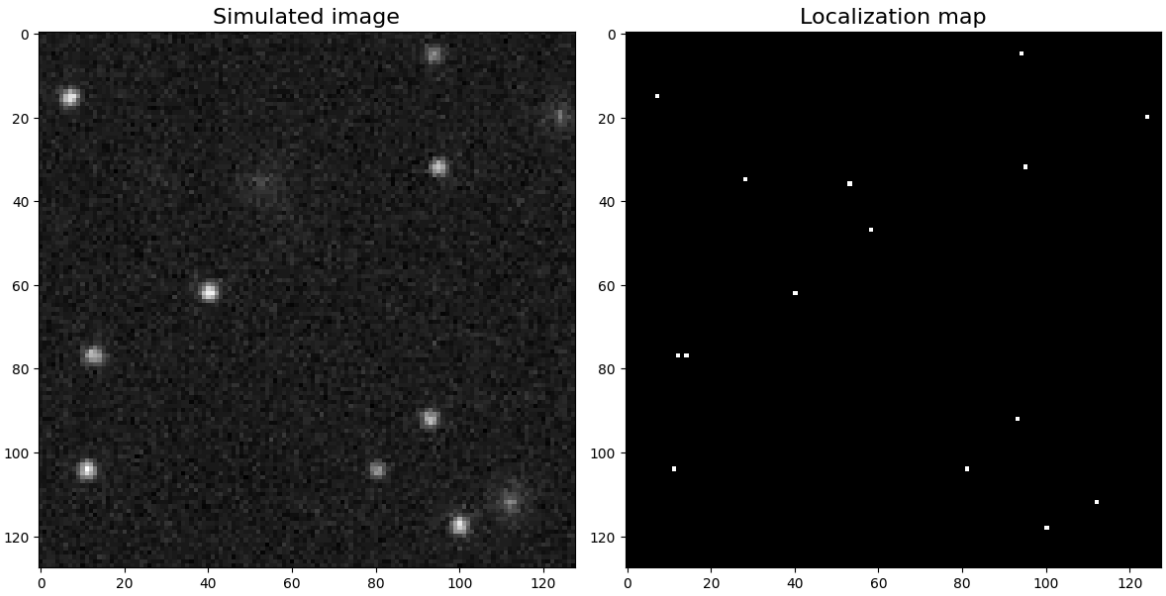
In [ ]:

```
#%% Import libraries and load data

from PIL import Image
import matplotlib.pyplot as plt
import deeptrack as dt
import numpy as np
import torch
from matplotlib.patches import Rectangle
import deeplay as dl
import scipy, tqdm
import scipy.spatial
import scipy.optimize

optics = dt.Fluorescence(
    wavelength=600 * dt.units.nm, NA=0.9, magnification=1,
    resolution=0.1 * dt.units.um, output_region=(0, 0, 50, 50),
)
particle = dt.PointParticle(position=(25, 25), intensity=1.2e4, z=0)
sim_im_pip = optics(particle) >> dt.Add(30) >> np.random.poisson >> dt.Add(82)

sim_im_pip.update()
sim_crop = sim_im_pip()

plt.plot()
plt.imshow(sim_crop, cmap="gray", vmin=100, vmax=250)
plt.title("Simulated particle", fontsize=16)
plt.show()

optics = dt.Fluorescence(
    wavelength=600 * dt.units.nm, NA=0.9, magnification=1,
    resolution=0.1 * dt.units.um, output_region=(0, 0, 128, 128),
)
particle = dt.PointParticle(
    position=lambda: np.random.uniform(0, 128, size=2),
    intensity=lambda: np.random.uniform(6e3, 3e4),
    z=lambda: np.random.uniform(-1.5, 1.5) * dt.units.um,
)
postprocess = (dt.Add(lambda: np.random.uniform(20, 40)) >> np.random.poisson
               >> dt.Add(lambda: np.random.uniform(70, 90)))
normalization = dt.AsType("float") >> dt.Subtract(110) >> dt.Divide(250)
particles = particle ^ (lambda: np.random.randint(10, 20))
sim_im_pip = (optics(particles)
              >> dt.Add(lambda: np.random.uniform(20, 40))
              >> np.random.poisson
              >> dt.Add(lambda: np.random.uniform(70, 90))
              >> dt.Clip(0, 255)
              >> dt.Divide(255) * 255)  # Ensure similar scale

sim_mask_pip = (particles
                >> dt.SampleToMasks(lambda: lambda particle: particle > 0,
                                    output_region=optics.output_region,
                                    merge_method="or")
                >> dt.AsType("int") >> dt.OneHot(num_classes=2))

sim_im_mask_pip = ((sim_im_pip & sim_mask_pip) >> dt.MoveAxis(2, 0)
                   >> dt.pytorch.ToTensor(dtype=torch.float))

sim_im, sim_mask = sim_im_mask_pip.update().resolve()
```

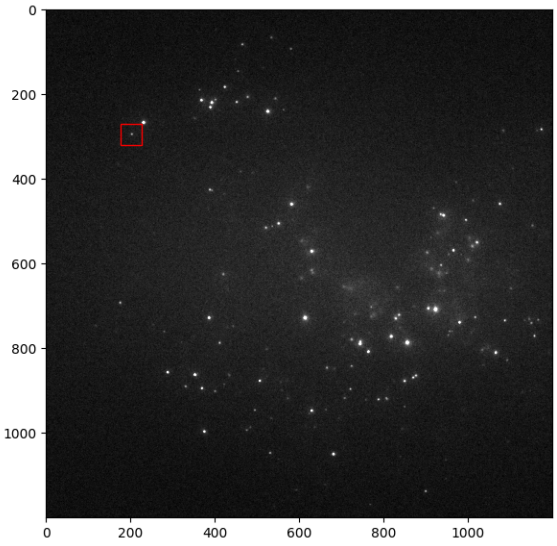
```
plt.figure(figsize=(12, 6))

plt.subplot(1, 2, 1)
plt.imshow(sim_im.squeeze(), cmap="gray")
plt.title("Simulated image", fontsize=16)

plt.subplot(1, 2, 2)
plt.imshow(sim_mask[1], cmap="gray")
plt.title("Localization map", fontsize=16)

plt.tight_layout()
plt.show()
```

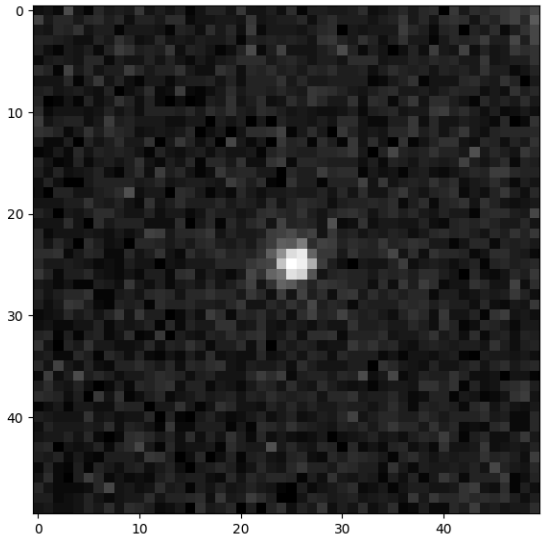
```
c:\Users\Green\AppData\Local\Programs\Python\Python311\Lib\site-packages\deeptrac
k\__init__.py:14: UserWarning: TensorFlow is detected in your environment. DeepTr
ack2 version 2.0++ no longer supports TensorFlow. If you need TensorFlow support,
please install the legacy version 1.7 of DeepTrack2:

    pip install deeptrack==1.7

For more details, refer to the DeepTrack documentation.
  warnings.warn(
WARNING:pint.util:Redefining '[magnetic_flux]' (<class 'pint.delegates.txt_defpar
ser.plain.DerivedDimensionDefinition'>)
```





In [ ]:
```
# %% Prepare training dot

image_of_particles = Image.open("frame_with_qdots.tif")

x0, y0, crop_size = 178, 271, 50
crop = image_of_particles.crop((x0, y0, x0 + crop_size, y0 + crop_size))
crop = np.array(crop)[..., np.newaxis]


plt.figure(figsize=(15, 10))

plt.subplot(1, 2, 1)
plt.imshow(image_of_particles, vmin=100, vmax=200, cmap="gray")
plt.gca().add_patch(Rectangle((x0, y0), crop_size, crop_size,
                              linewidth=1, edgecolor="r", facecolor="none"))

plt.subplot(1, 2, 2)
plt.imshow(crop, vmin=100, vmax=200, cmap="gray")

plt.show()
train_pip = (dt.Value(sim_crop) #Use sim_crop for simulated or crop for real
             >> dt.Multiply(lambda: np.random.uniform(0.9, 1.1))
             >> dt.Add(lambda: np.random.uniform(-0.1, 0.1))
             >> dt.MoveAxis(-1, 0) >> dt.pytorch.ToTensor(dtype=torch.float32))

train_dataset = dt.pytorch.Dataset(train_pip, length=400, replace=False)
dataloader = dl.DataLoader(train_dataset, batch_size=8, shuffle=True)
```





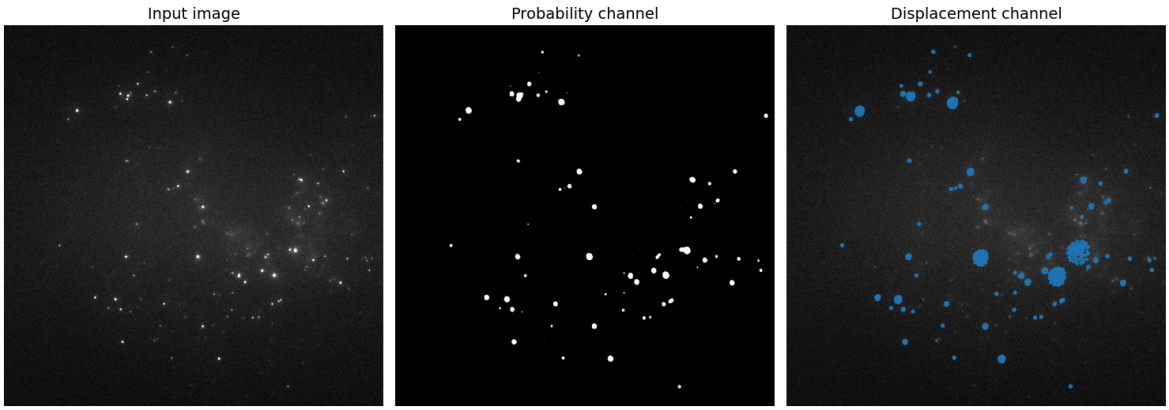
In [ ]:
```
#%% Training

lodestar = dl.LodeSTAR(n_transforms=4, optimizer=dl.Adam(lr=1e-4)).build()
trainer = dl.Trainer(max_epochs=200)
trainer.fit(lodestar, dataloader)
```

```
c:\Users\Green\AppData\Local\Programs\Python\Python311\Lib\site-packages\lightning\pytorch\trainer\configuration_validator.py:70: You defined a `validation_step` but have no `val_dataloader`. Skipping val loop.
c:\Users\Green\AppData\Local\Programs\Python\Python311\Lib\site-packages\lightning\pytorch\utilities\model_summary\model_summary.py:477: The total number of parameters detected may be inaccurate because the model contains an instance of `UninitializedParameter`. To get an accurate number, set `self.example_input_array` in your LightningModule.
INFO:
  | Name          | Type                        | Params | Mode
-------------------------------------------------------------------
0 | model         | ConvolutionalNeuralNetwork | 251 K  | train
1 | between_loss  | L1Loss                      | 0      | train
2 | within_loss   | L1Loss                      | 0      | train
3 | train_metrics | MetricCollection            | 0      | train
4 | val_metrics   | MetricCollection            | 0      | train
5 | test_metrics  | MetricCollection            | 0      | train
6 | optimizer     | Adam                        | 0      | train
-------------------------------------------------------------------
251 K     Trainable params
0         Non-trainable params
251 K     Total params
1.004     Total estimated model params size (MB)
39        Modules in train mode
0         Modules in eval mode
INFO:lightning.pytorch.callbacks.model_summary:
  | Name          | Type                        | Params | Mode
-------------------------------------------------------------------
0 | model         | ConvolutionalNeuralNetwork | 251 K  | train
1 | between_loss  | L1Loss                      | 0      | train
2 | within_loss   | L1Loss                      | 0      | train
3 | train_metrics | MetricCollection            | 0      | train
4 | val_metrics   | MetricCollection            | 0      | train
5 | test_metrics  | MetricCollection            | 0      | train
6 | optimizer     | Adam                        | 0      | train
-------------------------------------------------------------------
251 K     Trainable params
0         Non-trainable params
251 K     Total params
1.004     Total estimated model params size (MB)
39        Modules in train mode
0         Modules in eval mode
c:\Users\Green\AppData\Local\Programs\Python\Python311\Lib\site-packages\lightning\pytorch\trainer\connectors\data_connector.py:425: The 'train_dataloader' does not have many workers which may be a bottleneck. Consider increasing the value of the `num_workers` argument` to `num_workers=15` in the `DataLoader` to improve performance.
Training: |          | 0/? [00:00<?, ?it/s]
```

In [ ]:

```python
# %% Plot

image_np = np.array(image_of_particles)
image_np = image_np[..., np.newaxis]
torch_image = torch.from_numpy(image_np).permute(2, 0, 1).unsqueeze(0).float()
prediction = lodestar(torch_image)[0].detach().numpy()
x, y, rho = prediction[0], prediction[1], prediction[-1]

plt.figure(figsize=(15, 10))

plt.subplot(1, 3, 1)
```

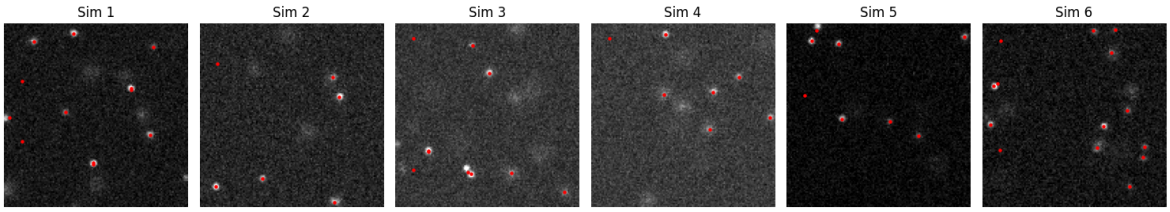
```
plt.imshow(image_np, vmin=100, vmax=200, cmap="gray")
plt.title("Input image", fontsize=14)
plt.axis("off")

plt.subplot(1, 3, 2)
plt.imshow(rho,cmap="gray")
plt.title("Probability channel", fontsize=14)
plt.axis("off")

plt.subplot(1, 3, 3)
plt.imshow(image_np, vmin=100, vmax=200,cmap="gray")
plt.scatter(y.flatten(), x.flatten(), alpha=rho.flatten() / rho.max(), s=5)
plt.title("Displacement channel", fontsize=14)
plt.axis("off")
plt.xlim(0, image_np.shape[1])
plt.ylim(image_np.shape[0], 0)

plt.tight_layout()
plt.show()
```



In [ ]:

```
# %% Evaluate performance

alpha = 0.05
beta = 1 - alpha
cutoff = 0.01
mode = "constant"

plt.figure(figsize=(15, 10))

for plot_idx in range(6):
    # Get a new synthetic image and mask
    sim_im, sim_mask = sim_im_mask_pip.update().resolve()

    torch_image = sim_im.unsqueeze(0).float()
    detections = lodestar.detect(torch_image, alpha=alpha, beta=beta,
                                 mode=mode, cutoff=cutoff)[0]

    image_np = torch_image.squeeze().numpy()
    mask_np = sim_mask[1].numpy()

    # Plot
    plt.subplot(1, 6, plot_idx + 1)
    plt.imshow(image_np, cmap="gray", vmin=100, vmax=200)
    plt.scatter(detections[:, 1], detections[:, 0], s=5, color="red")
    plt.title(f"Sim {plot_idx + 1}", fontsize=12)
    plt.axis("off")
```

```
plt.tight_layout()
plt.show()
```



In [ ]:

```
# %% F1

distance_th = 30
TP, FP, FN = 0, 0, 0

for _ in tqdm.tqdm(range(100)):
    sim_im, sim_mask = sim_im_mask_pip.update().resolve()

    torch_image = sim_im.unsqueeze(0).float()
    detections = lodestar.detect(torch_image, alpha=alpha, beta=beta,
                                 mode="constant", cutoff=cutoff)[0]

    # Get ground truth centroids from mask
    centroids = np.argwhere(sim_mask[1].numpy() > 0)[:, [1, 0]]  # (x, y)

    distance_matrix = scipy.spatial.distance_matrix(detections, centroids)
    row_idx, col_idx = scipy.optimize.linear_sum_assignment(distance_matrix)

    filtered_row_ind = row_idx[distance_matrix[row_idx, col_idx] < distance_th]
    filtered_col_ind = col_idx[distance_matrix[row_idx, col_idx] < distance_th]

    TP += len(filtered_row_ind)
    FP += len(detections) - len(filtered_row_ind)
    FN += len(centroids) - len(filtered_col_ind)

F1 = 2 * TP / (2 * TP + FP + FN + 1e-8)

print(f"TP: {TP}, FP: {FP}, FN: {FN}, F1: {F1:.4f}")
```

```
100%|██████████| 100/100 [00:10<00:00,  9.86it/s]
TP: 739, FP: 263, FN: 702, F1: 0.6050
```