Cryo-EM notebook

November 21, 2022

1 CT reconstruction via spectral decomposition of the atomic structure

First we load the atomic structure and the MD trajectory in order to generate the synthetic dataset. For the atomic model in each frame we select the C-α positions.

[1]:
```
import torch
import numpy as np
import mdtraj as md

## We load an MD trajectory.
## The molecular structure must be in a pdb or psf file.
## The trajectory is stored in a dcd file.

struct = 2 # 1 for the small protein (214 C-alpha atoms) and 2 for the big one␣
 ↪(590 C-alpha atoms).

if struct == 1:
    MD_traj = md.load('data/struct1/dims0001_fit-core.dcd',top = 'data/struct1/
 ↪adk4ake.psf')
    view_angle_plot = torch.tensor([[0, 0, 0]])
if struct == 2:
    MD_traj1 = md.load('data/struct2/MDtraj_sarscov_1.dcd',
                       top = 'data/struct2/
 ↪DESRES-Trajectory_sarscov2-12212688-5-2-no-water.pdb')
    MD_traj2 = md.load('data/struct2/MDtraj_sarscov_2.dcd',
                       top = 'data/struct2/
 ↪DESRES-Trajectory_sarscov2-12212688-5-2-no-water.pdb')
    MD_traj = MD_traj1.join(MD_traj2)
    del MD_traj1, MD_traj2
    view_angle_plot = torch.tensor([[0, 0, 0.5]])


## Here we select the C-alpha positions
indices = []
for m in MD_traj.topology.atoms_by_name('CA'):
    indices.append(m.index)
```

```
# We have to multily the trajectory by 10 so that the interatomic distance is␣
↪in Angstroms
CA_pos = torch.tensor(MD_traj.xyz[:,indices,:])*10
print('Average interatomic distance:',round(float((CA_pos[:,1:]-CA_pos[:,:-1]).
↪norm(dim=-1).mean()),2), 'Angstroms')
```

```
Average interatomic distance: 3.86 Angstroms
```

1.1 Parameters of the MD trajectory

Now we compute the parameters of each atomic model in the MD trajectory (rotation angles, orientation and location). This will serve as ground truth.

[2]:
```
## We set the reference atom. In this case, we select the atom which is most
# centred in the structure.

D = torch.zeros(CA_pos.shape[1])

for i in range(CA_pos.shape[1]):
    D[i] = (CA_pos[0] - CA_pos[0,i]).sum(dim=0).norm()

ref_idx = torch.argmin(D)

## We now compute the parameters for each of the chains in the MD trajectory
## These parameters represent the groud truth.

from tools.DFF import compute_parameters

Psi, Theta, x0s, Orientations, Dists = compute_parameters(CA_pos, ref_idx)

from data_generator.structure_batch import chain_structure

# We create a chain structure for the MD trajectory. This one will be used to␣
↪generate the synthetic dataset.
trajectory_ground_truth = chain_structure(Psi, Theta, x0s, Orientations, Dists,␣
↪ref_idx)

## We select the indexes that are to be estimated (those which may vary␣
↪depending on the conformation).
## Since we do not have prior information about secondary structures, we select␣
↪all the angles.
idx_model = list(range(Psi.shape[1]))
```

1.1.1 Synthetic cryo-EM data

We generate the synthetic cryo-EM dataset. We need to specify the features of the dataset such as the CTF parameters and noise levels.

For each cryo-EM image, the defocus of the CTF is chosen at random among the values in the vector 'Df'.

$$CTF(k,\lambda,\Delta f,C_s) = \left(\sqrt{1-\alpha^2}\cos(\theta(k)) - \alpha\sin(\theta(k))\right)\exp\left(-\frac{B|k|}{4}\right)$$

with

$$\theta(k) := 2\pi\left(-\frac{\Delta f\lambda|k|^2}{2} + \frac{C_s\lambda^3|k|^4}{4}\right) + \theta_0 \quad \text{for } k \in \left\{\left(\frac{i}{2L},\frac{j}{2L}\right), \quad (i,j) \in [-n_{px},\dots,n_{px}-1]^2\right\}$$

where Δf is the defocus in Angstroms, e.g. $2.5\cdot 10^4$, L is the length of the image side in Angstroms and n_{px} is the number of pixels of the image side.

The wavelength λ in terms of the voltage V in KeV:

$$\lambda = \frac{h}{\sqrt{2m_0E\left(1+\frac{E}{2m_0c^2}\right)}} = \frac{12.2643247}{\sqrt{V\cdot 10^3 + 0.978466V^2}}.$$

[3]:
```
### Here we generate the dataset

from data_generator.dataset import dataset

CTF_data_feat = {
        'kV' : 200, # voltage in KV
        'Df' : torch.tensor([1.5, 1.75, 2., 2.25, 2.5]), # List of Defocus␣
 ↪values in microns
        'Cs' : 2., # Spherical aberration in milimeters
        'alpha' : .1, # amplitude constrast
        'B_factor' : 0., # Decay factor in Angstroms
        'phase_shift': 0. # Degrees
        }

dataset_features = {
        'struct' : trajectory_ground_truth,
        'n_imgs' : 400,
        'orient_variability' : torch.Tensor([[0,2*np.pi],
                                             [0,2*np.pi],
                                             [0,2*np.pi]]),
        'n_px_3d' : 16,
        'sigma_gaussian_3d' : 3.,
        'mask_size_3d' : 5, # voxels
        'noise_3d' : .5,

        'n_px_2d' : 64,
        'noise_2d' : 3.,
        'density_type': 'Gaussian',
        'sigma_density' : 3.,
```

```
        'CTF_data_feat' : CTF_data_feat
        }


(vols_data, structure_data, orientation_diffs_data,
 CT_images_noise, CT_images_clean, clean_imgs_no_CTF,
 img_lims, Df) = dataset(dataset_features)

# Let's display the CTF filters
from tools.CTF import CTF_filters

img_side = img_lims[0]-img_lims[1]

print('Pixel size:', round(float(img_side/dataset_features['n_px_2d']), 2))

filters = CTF_filters(dataset_features['n_px_2d'], img_side,␣
 ↪CTF_data_feat['kV'], CTF_data_feat['Df'], CTF_data_feat['Cs'],
                       CTF_data_feat['alpha'], CTF_data_feat['B_factor'],␣
 ↪CTF_data_feat['phase_shift'])

from tools.plots import disp_image
disp_image(filters, multi_img = True)
```

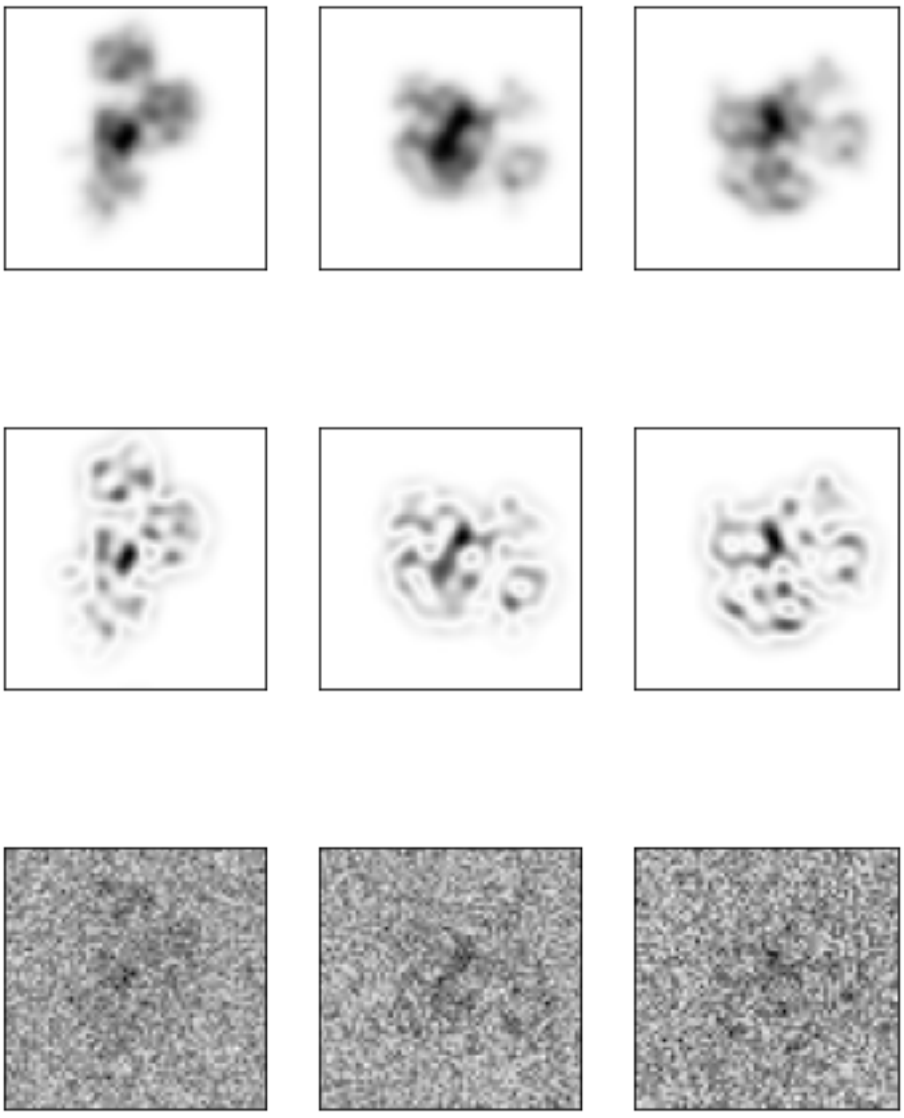
```
/Users/carlosesteveyague/anaconda3/lib/python3.8/site-
packages/torch/functional.py:504: UserWarning: torch.meshgrid: in an upcoming
release, it will be required to pass the indexing argument. (Triggered
internally at /Users/runner/work/_temp/anaconda/conda-
bld/pytorch_1666647174771/work/aten/src/ATen/native/TensorShape.cpp:3191.)
  return _VF.meshgrid(tensors, **kwargs)  # type: ignore[attr-defined]
100%|      | 400/400 [00:04<00:00, 92.02it/s]
```

```
Pixel size: 1.76
```

1.1.2 Examples of synthetic cryo-EM images

First row after applying the X-ray transform to the particle. Second row after convolving with the CTF Third row after adding gaussian noise

[4]:
```
# Let's display some images
imgs = clean_imgs_no_CTF[:3]
disp_image(imgs, multi_img = True)

imgs = CT_images_clean[:3]
disp_image(imgs, multi_img = True)

imgs = CT_images_noise[:3]
disp_image(imgs, multi_img = True)


## Signal to Noise Ratio (SNR)
print('Signal to noise ratio:', round(float(CT_images_clean.var()/
 ↪dataset_features['noise_2d']**2), 4))
```



```
Signal to noise ratio: 0.0823
```

1.2 Construct the model

Here we construct the model to make the predictions. We need to select the given conformation, which will be used as initialization, the number of eigenvectors in the spectral decomposition. We also need to specify the features of the images such as dimensions and CTF parameters.

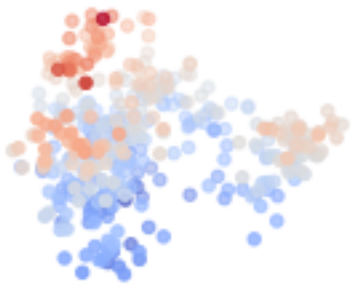
[5]:
```
# As given conformation we take the first structure in the MD trajectory
n_init = 0
Psi_init = trajectory_ground_truth.Psi[n_init]
Theta_init = trajectory_ground_truth.Theta[n_init]
ref_idx = trajectory_ground_truth.ref_idx

# The interatomic distance is set to be constant
# and equal to the average interatomic distance
# in the given conformation.
dists = trajectory_ground_truth.dists[n_init]
dists = dists*0 + dists.mean()

# We specify the number of eigenvectors for the spectral decomposition.
n_eigenval = 10

model_features = {
        'n_eigenval' : n_eigenval,

        # Given atomic model
        'Psi_init' : Psi_init,
        'Theta_init' : Theta_init,
        'dists' : dists,
        'ref_idx': ref_idx,

        # Features of the images
        'img_lims': img_lims,
        'n_px': CT_images_noise.shape[-1],
        'sigma_density' : dataset_features['sigma_density'], # sigma of the␣
 ↪Gaussian density for each atom
        'density_type' : dataset_features['density_type'],
        'CTF_feat' : CTF_data_feat,

        # List of indexes to be estimated
        'param_idxs' : idx_model

        }
```

```
from model.model_chain import model_chain

model = model_chain(model_features)
```

1.3 Graph Laplacian

[6]:
```
from time import time
from tools.graph import graph_gaussian_kernel, laplacian, eigen_graph

sigma_gauss_kernel = 5*dataset_features['n_px_3d']*dataset_features['noise_3d']

start = time()
graph = graph_gaussian_kernel(vols_data, sigma_gauss_kernel).detach()
print('Time to compute the graph Laplacian:', time() - start)

start = time()
Lap = laplacian(graph, True)
[eigen_val, eigen_vec] = eigen_graph(Lap, n_eigenval)
print('Time to compute the eigenvectors:', time() - start)

from tools.plots import disp_point_cloud

disp_point_cloud(eigen_vec[:, [1,2,3]])
disp_point_cloud(eigen_vec[:, [1,2,4]])
disp_point_cloud(eigen_vec[:, [1,3,4]])
```

```
Time to compute the graph Laplacian: 0.1920309066772461
Time to compute the eigenvectors: 0.053266286849975586
```



1.4 Training and test data

Here we split the cryo-EM dataset into training and test data,

[7]:
```
from tools.training import Dataset, train_model

# The orientation of the particles in the images are the sum of the
# orientation of the particle in the MD trajectory plus the rotation
# applied when generating the cryo-EM dataset.
# This information is assumed to be known.

orientations = structure_data.orientation + orientation_diffs_data

# The inputs for the model are the following, which we concatenate in a single␣
 ↪tensor:
# -the eigenvector coordinates of each particle
# -the orientation of each particle
# -the location of each particle
# -the defocus of each image
inputs = torch.cat([eigen_vec, orientations, structure_data.x0, Df.
 ↪unsqueeze(-1)], dim = -1)


train_pctg = .9
n_total = inputs.shape[0]
n_train = int(train_pctg*n_total)
n_test = n_total - n_train
train_idx, test_idx = torch.utils.data.random_split(range(n_total), [n_train,␣
 ↪n_test])

# Here we select the training data, i.e. the cryo-EM images with the␣
 ↪corresponding inputs
train_inputs = inputs[train_idx]
training_data = Dataset(train_inputs, CT_images_noise[train_idx])


# Here we select the test data, i.e. the cryo-EM images with the corresponding␣
 ↪inputs
# We also compute the atomic model for each particle, that are used to evaluate␣
 ↪the accuracy of the predictions.
struct_ground_truth_test = structure_data.
 ↪discrete_curves(orientation_diffs_data)[test_idx]
test_inputs = inputs[test_idx]

test_data_pointcloud = Dataset(test_inputs, struct_ground_truth_test)
test_data = Dataset(test_inputs, CT_images_clean[test_idx])


# Here we compute the CT images (without CTF) and the atomic models before␣
 ↪training the model
```

```
# These will be used at the end to compare with the predictions after training.
preds_init_imgs = model.pred_images_without_CTF(test_inputs[:20, :-1]).detach()

# we shall plot all the structures from the same viewing angle to better compare
test_inputs_plot = test_inputs.clone()
test_inputs_plot[:, -7:-4] = view_angle_plot.repeat(test_inputs.shape[0], 1)
struct_init = model.forward_disc_curve(test_inputs_plot).detach()
```

1.4.1 Apply SGD to estimate the coefficients

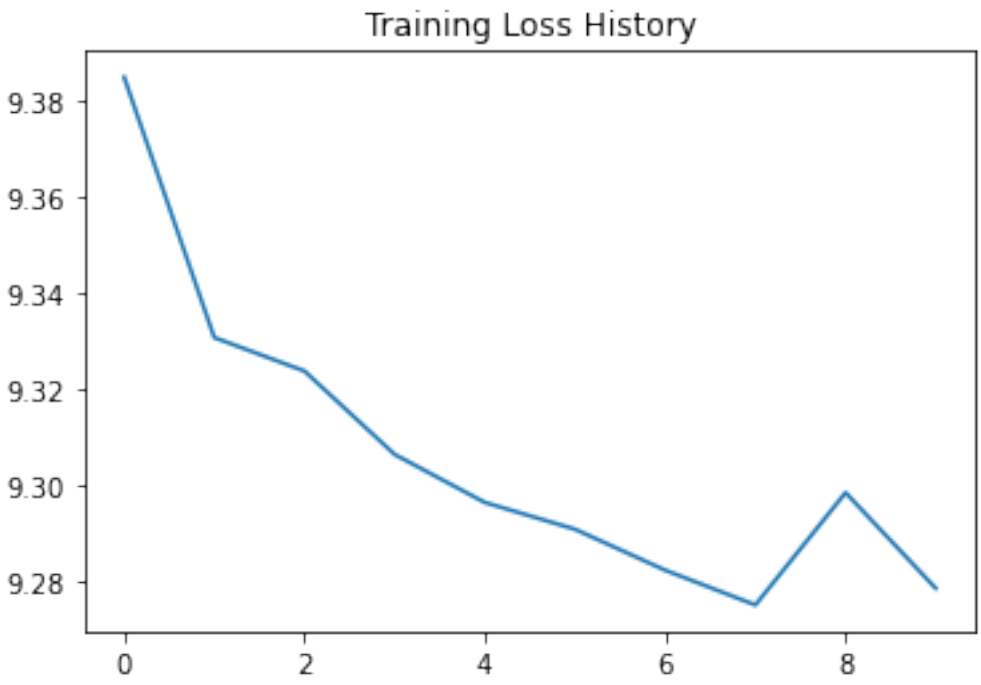
This make take several minutes.

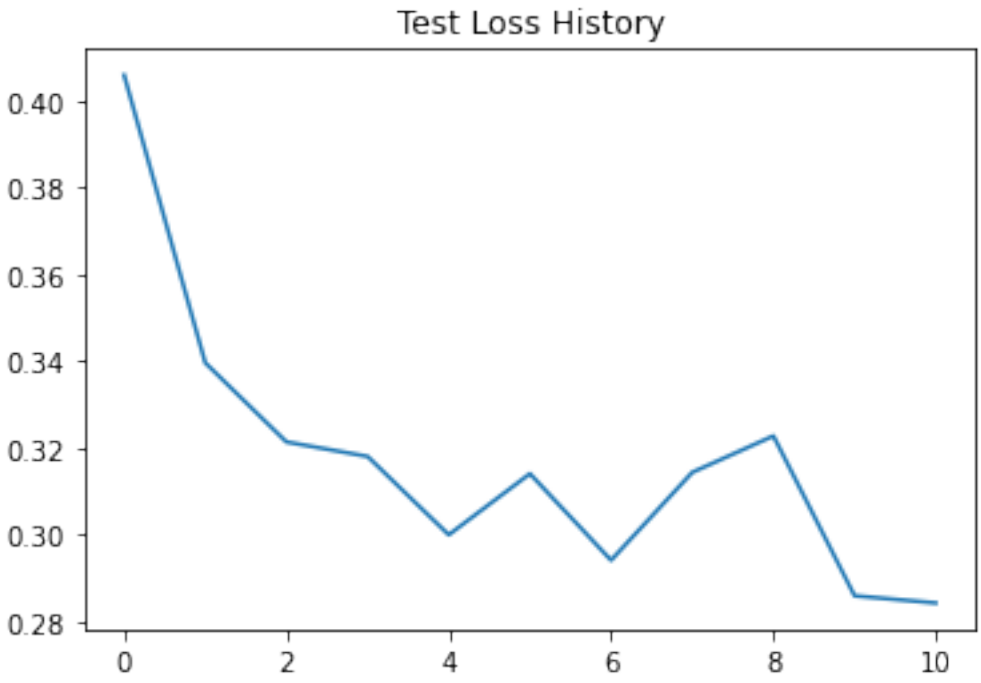
[8]:
```
### Training

# Generators
training_params = {'batch_size': 100,
          'shuffle': True,
          'max_epochs' : 10,
          'learning_rate' : .2,
          'momentum': .9}

train_model(model, training_data, test_data, test_data_pointcloud,␣
 ↪training_params)
```

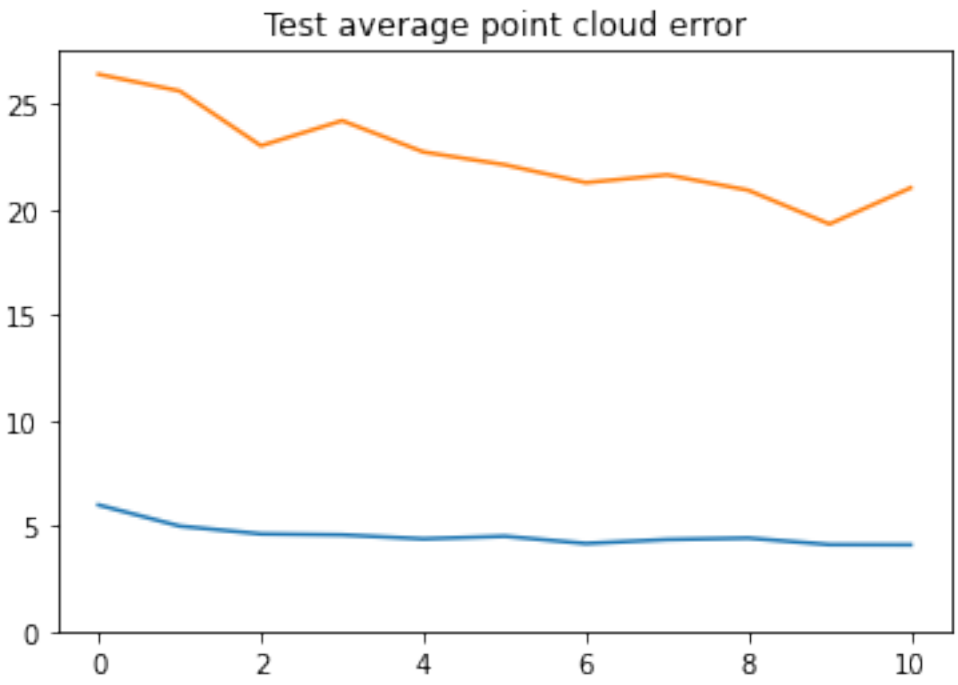
```
  0%|          | 0/4 [00:00<?, ?it/s]

Initial test loss: 0.40595731538260826
Initial test average point cloud error: 5.9886155128479
Initial test average max point cloud error: 26.395198822021484
Initial test max point cloud error: 48.33027267456055

100%|     | 4/4 [00:14<00:00,  3.65s/it]

Epoch 1 completed.
Training loss: 9.384734927865235

  0%|          | 0/4 [00:00<?, ?it/s]

Test loss: 0.33967084694612637
Test average point cloud error: 4.987405300140381
Test average max point cloud error: 25.602230072021484
Test  max point cloud error: 48.20282745361328

100%|     | 4/4 [00:14<00:00,  3.59s/it]

Epoch 2 completed.
Training loss: 9.330673519056287

  0%|          | 0/4 [00:00<?, ?it/s]

Test loss: 0.3214385493143105
Test average point cloud error: 4.617284774780273
```

```
Test average max point cloud error: 23.00677490234375
Test  max point cloud error: 47.991912841796875

100%|      | 4/4 [00:13<00:00,  3.46s/it]

Epoch 3 completed.
Training loss: 9.323723868678046

  0%|          | 0/4 [00:00<?, ?it/s]

Test loss: 0.3181026518566483
Test average point cloud error: 4.571664333343506
Test average max point cloud error: 24.196331024169922
Test  max point cloud error: 49.82202911376953

100%|      | 4/4 [00:15<00:00,  3.92s/it]

Epoch 4 completed.
Training loss: 9.306418654809073

  0%|          | 0/4 [00:00<?, ?it/s]

Test loss: 0.3000456686500973
Test average point cloud error: 4.382312774658203
Test average max point cloud error: 22.712495803833008
Test  max point cloud error: 50.44796371459961

100%|      | 4/4 [00:13<00:00,  3.49s/it]

Epoch 5 completed.
Training loss: 9.296448096446678

  0%|          | 0/4 [00:00<?, ?it/s]

Test loss: 0.3141201173718019
Test average point cloud error: 4.507026672363281
Test average max point cloud error: 22.108585357666016
Test  max point cloud error: 45.501434326171875

100%|      | 4/4 [00:13<00:00,  3.39s/it]

Epoch 6 completed.
Training loss: 9.290885242268898

  0%|          | 0/4 [00:00<?, ?it/s]

Test loss: 0.29419139089223506
Test average point cloud error: 4.162221431732178
Test average max point cloud error: 21.25967788696289
Test  max point cloud error: 49.3173828125

100%|      | 4/4 [00:13<00:00,  3.39s/it]

Epoch 7 completed.
Training loss: 9.282390738899515

  0%|          | 0/4 [00:00<?, ?it/s]
```

```
Test loss: 0.3144298505433679
Test average point cloud error: 4.35059118270874
Test average max point cloud error: 21.629077911376953
Test  max point cloud error: 49.89313888549805

100%|     | 4/4 [00:14<00:00,  3.52s/it]

Epoch 8 completed.
Training loss: 9.275169943367002

  0%|          | 0/4 [00:00<?, ?it/s]

Test loss: 0.3228298813270764
Test average point cloud error: 4.418216228485107
Test average max point cloud error: 20.899715423583984
Test  max point cloud error: 46.288394927978516

100%|     | 4/4 [00:13<00:00,  3.50s/it]

Epoch 9 completed.
Training loss: 9.298488753673364

  0%|          | 0/4 [00:00<?, ?it/s]

Test loss: 0.2859948012588259
Test average point cloud error: 4.111403942108154
Test average max point cloud error: 19.29727554321289
Test  max point cloud error: 42.71347427368164

100%|     | 4/4 [00:14<00:00,  3.63s/it]

Epoch 10 completed.
Training loss: 9.278629896580872
Test loss: 0.28434082072683353
Test average point cloud error: 4.100875377655029
Test average max point cloud error: 21.01426124572754
Test  max point cloud error: 45.888816833496094
Training finished. Elapsed time: 148.35826516151428
```


Training Loss History
9.38
9.36
9.34
9.32
9.30
9.28
0
2
4
6
8


Test Loss History
0.40
0.38
0.36
0.34
0.32
0.30
0.28
0
2
4
6
8
10



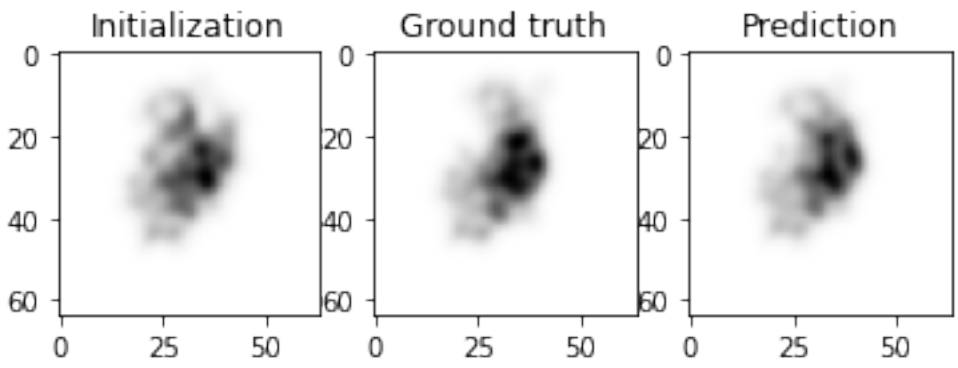

```
Training finished. Elapsed time: 148.67671394348145
Training loss: 9.278629896580872
Test loss: 0.28434082072683353
Test average point cloud error: 4.100875377655029
Test average max point cloud error: 21.01426124572754
Test  max point cloud error: 45.888816833496094
```

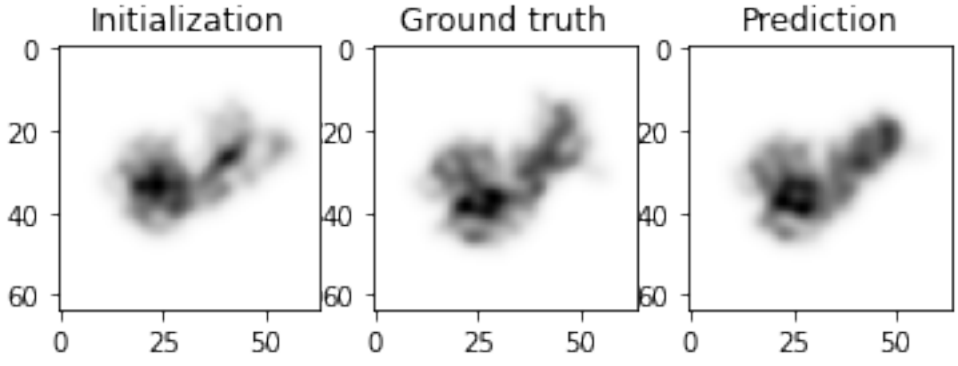
1.5 Display some predictions

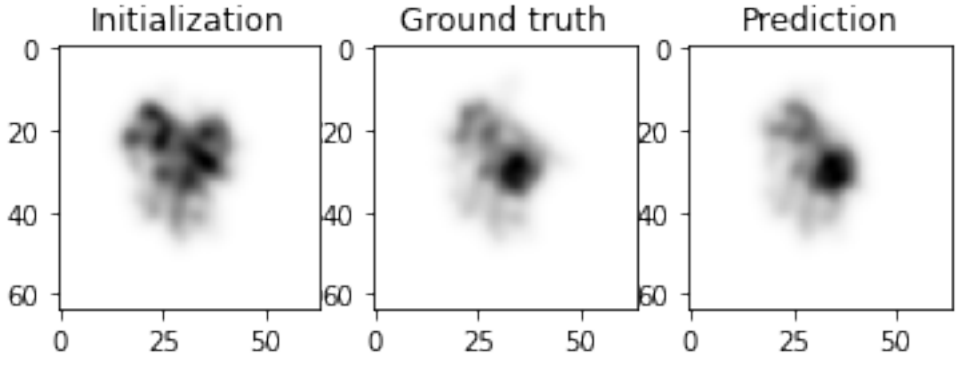
We compare the initial guess for the particle with the ground truth and the prediction after the optimization.

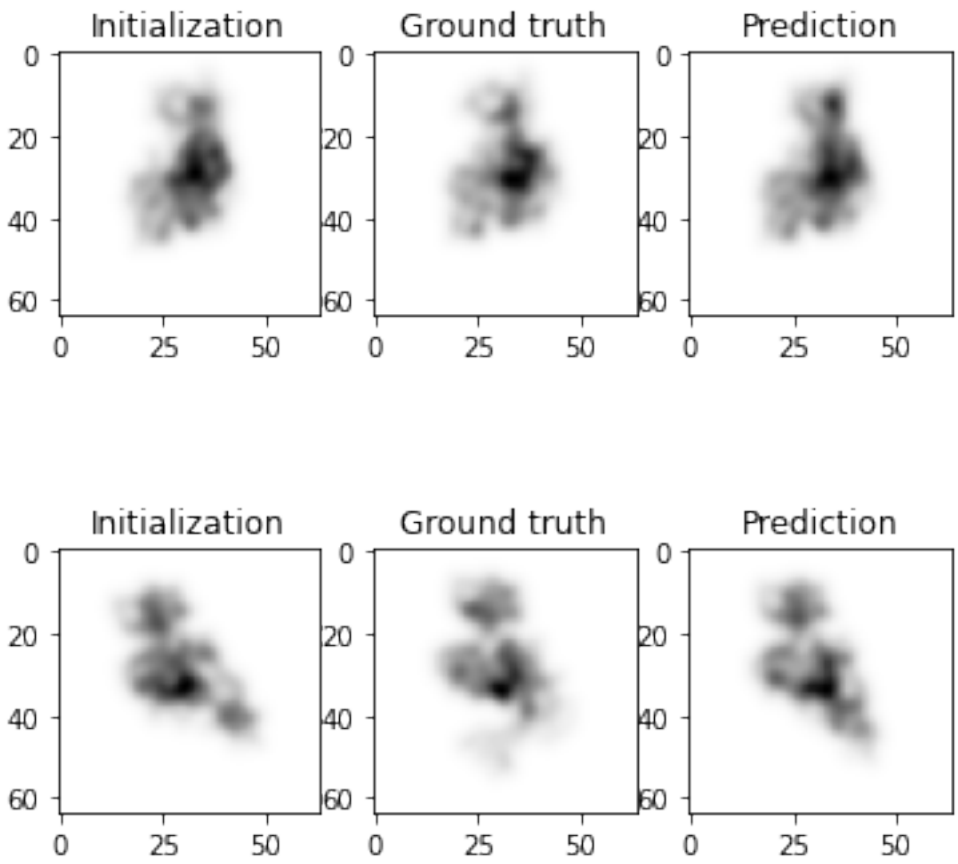
Here we show the tomographic projection of the particles (without the CTF).

[9]:
```
# Let's compare the tomographic projections of the predicted structures
# with the ground truth and the initialization.
from tools.plots import disp_img_validation

preds_imgs = model.pred_images_without_CTF(test_inputs[:5, :-1]).detach()

original_imgs = clean_imgs_no_CTF[test_idx]

disp_img_validation(preds_init_imgs, original_imgs, preds_imgs)
```


Initialization
Ground truth
Prediction
0
20
40
60
0
25
50


Initialization
Ground truth
Prediction
0
20
40
60
0
25
50

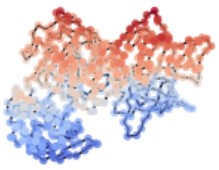

Initialization
Ground truth
Prediction
0
20
40
60
0
25
50



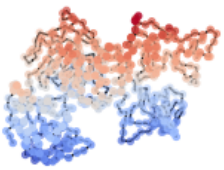
Predictions of the atomic structures

[10]:
```
# Let's compare the predicted structures
# with the ground truth and the initialization.

from tools.plots import disp_struct_validation

struct_preds = model.forward_disc_curve(test_inputs_plot).detach()

orientation_diffs_data_plot = view_angle_plot - structure_data.orientation
struct_ground_truth = structure_data.
 ↪discrete_curves(orientation_diffs_data_plot).detach()[test_idx]

disp_struct_validation(struct_init, struct_ground_truth, struct_preds[:10])
```

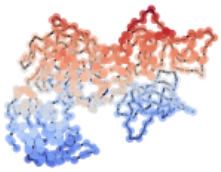
Initialization



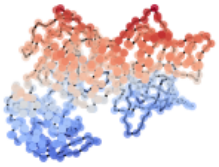
Original



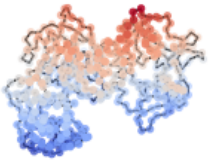
Prediction



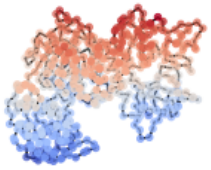
Initialization



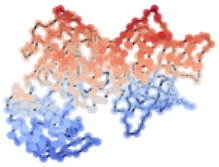
Original



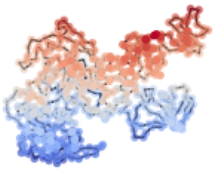
Prediction



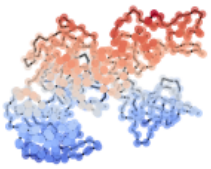
Initialization

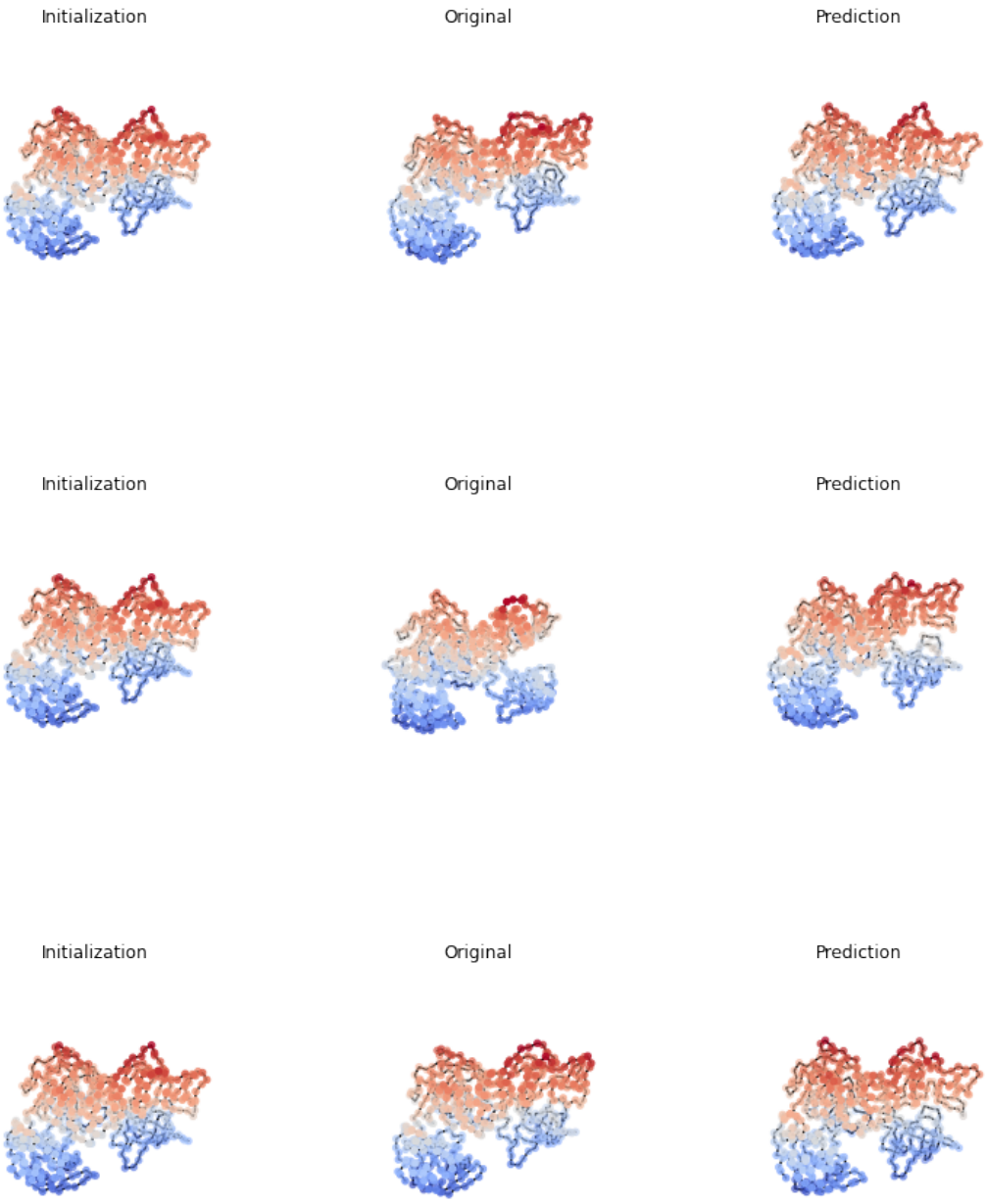


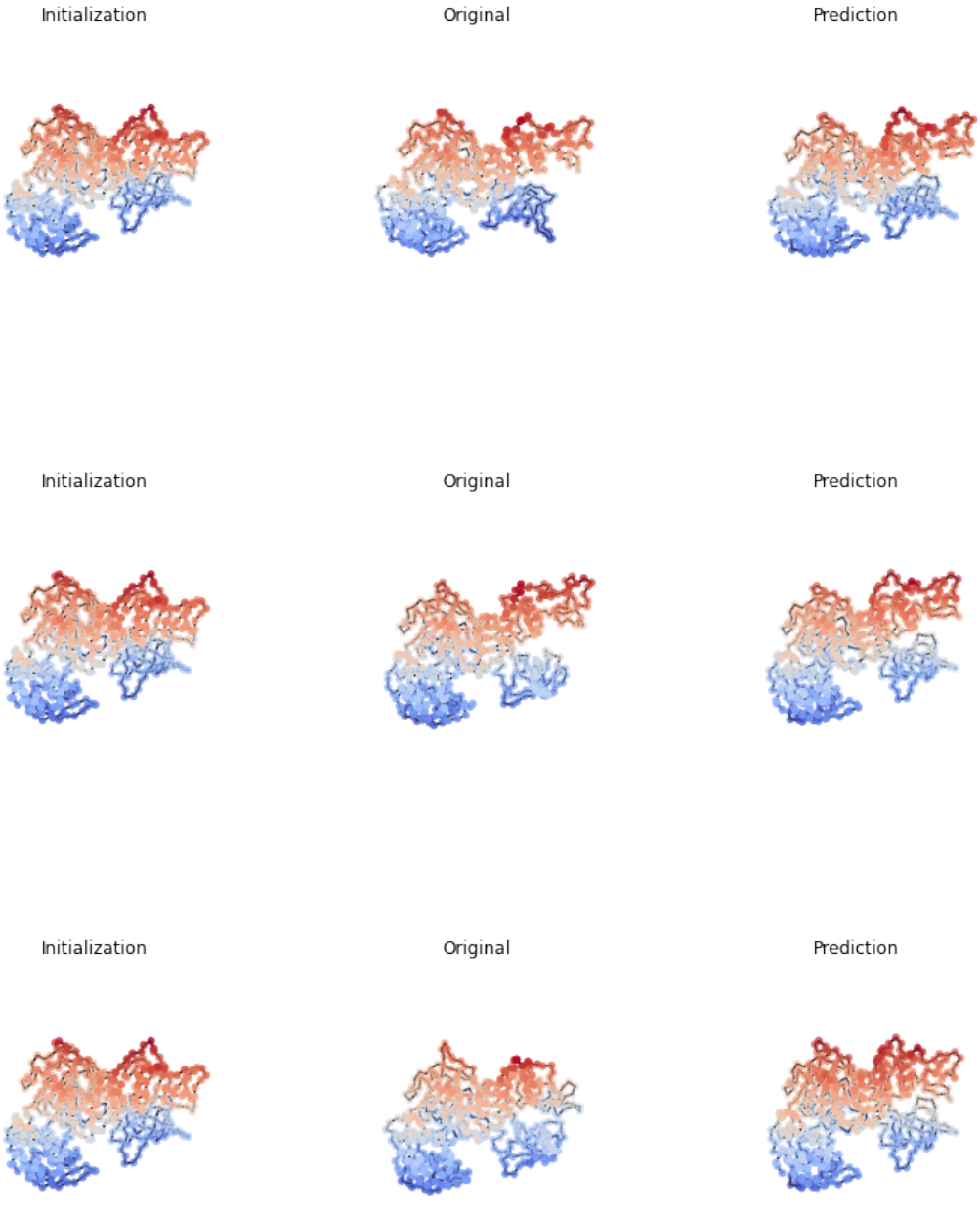
Original

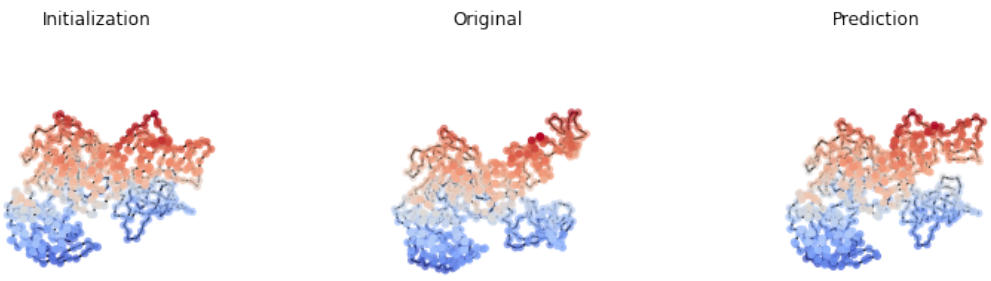


Prediction




Initialization
Original
Prediction
Initialization
Original
Prediction
Initialization
Original
Prediction


Initialization
Original
Prediction
Initialization
Original
Prediction
Initialization
Original
Prediction


Initialization
Original
Prediction