E4-7.1_Init-NCA

January 14, 2026

1 imports

[1]:
```
#@title Imports and Notebook Utilities
import os
import io
import PIL.Image, PIL.ImageDraw
import base64
import zipfile
import json
import re
import requests
import numpy as np
import matplotlib.pylab as pl
import glob
import random
import h5py
import os
import piq
from IPython.display import Image, HTML, Markdown, clear_output
from tqdm import tqdm_notebook, tnrange

!nvidia-smi -L
```

```
GPU 0: NVIDIA GeForce RTX 3080 Laptop GPU (UUID:
GPU-6cf8054c-6c04-c256-2ebb-8137cfc2e798)
```

[2]:
```
import torch
import torchvision.models as models
from torch.utils.data import Dataset
import torch.nn.functional as F
from torch import nn
from IPython.display import clear_output, display, HTML
import h5py
from pathlib import Path


import sys
sys.path.append("C:/Users/GAI/Desktop/Scott/NCA_Research")
```

```python
from core_utils.plotting import *
from core_utils.utils_image import *
from core_utils.viz_train import *
from E4_PI_NCA.utils.helper import *
from E4_PI_NCA.utils.Dataset import *
from E4_PI_NCA.utils.PI_Metric import ScaledPhysicsMetrics

torch.set_default_device('cuda')
```

2 heliper

[3]:
```python
def normalize_to_01(x):
    x_min, x_max = x.amin(dim=(1,2,3), keepdim=True), x.amax(dim=(1,2,3),␣
 ↪keepdim=True)
    return (x - x_min) / (x_max - x_min + 1e-8)

def sperate_base_channels(x, base_channel_count=6):
    """      """
    base_channels = x[:, :base_channel_count, :, :]
    other_channels = x[:, base_channel_count:, :, :]
    return base_channels, other_channels
```

3 vgg loss

[4]:
```python
def pad_to_three_channels(img_slice):
    """
     1   2      /   3
      : (B, C, H, W)   C   1   2
      : (B, 3, H, W)
    """
    # B, C, H, W
    b, c, h, w = img_slice.shape

    if c == 3:
        #   3
        return img_slice
    elif c == 1:
        # 1   (C_k) ->   3   (C_k, C_k, C_k)
        return torch.cat([img_slice] * 3, dim=1)
    elif c == 2:
        # 2   (C_k, C_{k+1}) ->         (C_k, C_{k+1}, C_k)
        #
        return torch.cat([img_slice, img_slice[:, 0:1, :, :]], dim=1)
    else:
        #           3
        raise ValueError(f"Invalid number of channels for padding: {c}")

vgg16 = models.vgg16(weights='IMAGENET1K_V1').features

def calc_styles_vgg(imgs):
  style_layers = [1, 6, 11, 18, 25]
  mean = torch.tensor([0.485, 0.456, 0.406])[:,None,None]
  std = torch.tensor([0.229, 0.224, 0.225])[:,None,None]
  x = (imgs-mean) / std
  b, c, h, w = x.shape
  features = [x.reshape(b, c, h*w)]
  for i, layer in enumerate(vgg16[:max(style_layers)+1]):
    x = layer(x)
    if i in style_layers:
      b, c, h, w = x.shape
      features.append(x.reshape(b, c, h*w))
  return features

def project_sort(x, proj):
  return torch.einsum('bcn,cp->bpn', x, proj).sort()[0]

def ot_loss(source, target, proj_n=32):
  ch, n = source.shape[-2:]
  projs = F.normalize(torch.randn(ch, proj_n), dim=0)
  source_proj = project_sort(source, projs)
  target_proj = project_sort(target, projs)
  target_interp = F.interpolate(target_proj, n, mode='nearest')
  return (source_proj-target_interp).square().sum()

def loss_f(imgs, target_img):
    """
      N            3

       3              3
    """
    total_loss = 0.0
    N = imgs.shape[1]  #

    #    3      (          )
    weight_per_group = 1.0

    # --- 1.     3    ---
    i = 0
    while i + 3 <= N:
        #    i, i+1, i+2   (3   )
        imgs_slice = imgs[:, i:i+3, :, :]
        target_img_slice = target_img[:, i:i+3, :, :]

        #
        yy = calc_styles_vgg(target_img_slice)
        xx = calc_styles_vgg(imgs_slice)
        group_loss = sum(ot_loss(x, y) for x, y in zip(xx, yy))

        total_loss += weight_per_group * group_loss

        i += 3 #

    # --- 2.      3    (N mod 3 = 1   2) ---
    remaining_channels = N - i
    if remaining_channels > 0:
        #     1   2
        imgs_slice = imgs[:, i:N, :, :]
        target_img_slice = target_img[:, i:N, :, :]

        #    3    (     )
        imgs_padded = pad_to_three_channels(imgs_slice)
        target_img_padded = pad_to_three_channels(target_img_slice)

        #
        #    :     VGG      ImageNet
        #       1   2       ImageNet
        yy_padded = calc_styles_vgg(target_img_padded)
        xx_padded = calc_styles_vgg(imgs_padded)

        #
        final_group_weight = 0.5 #
        final_group_loss = sum(ot_loss(x, y) for x, y in zip(xx_padded,␣
↪yy_padded))

        total_loss += final_group_weight * final_group_loss

    return total_loss

def to_nchw(img):
  img = torch.as_tensor(img)
  if len(img.shape) == 3:
    img = img[None,...]
  return img.permute(0, 3, 1, 2)
```

4 NCA model

[5]:
```
#@title Minimalistic Neural CA
ident = torch.tensor([[0.0,0.0,0.0],[0.0,1.0,0.0],[0.0,0.0,0.0]])
sobel_x = torch.tensor([[-1.0,0.0,1.0],[-2.0,0.0,2.0],[-1.0,0.0,1.0]])
lap = torch.tensor([[1.0,2.0,1.0],[2.0,-12,2.0],[1.0,2.0,1.0]])
IDX_GEO_MASK = 2
def perchannel_conv(x, filters):
  '''filters: [filter_n, h, w]'''
  b, ch, h, w = x.shape
  y = x.reshape(b*ch, 1, h, w)
  y = F.pad(y, [1, 1, 1, 1], 'circular')
  y = F.conv2d(y, filters[:,None])
  return y.reshape(b, -1, h, w)

def perception(x):
  filters = torch.stack([ident, sobel_x, sobel_x.T, lap])
  return perchannel_conv(x, filters)

class CA(torch.nn.Module):
  def __init__(self, chn=12, hidden_n=96):
    super().__init__()
    self.chn = chn
    self.w1 = nn.Conv2d(chn*4, hidden_n, 1)
    self.w2 = nn.Conv2d(hidden_n, chn, 1, bias=False)
    self.w2.weight.data.zero_()

  def forward(self, x, update_rate=0.5):
    y = perception(x)
    y = self.w2(torch.relu(self.w1(y)))
    b, c, h, w = y.shape
    udpate_mask = x[:,IDX_GEO_MASK:IDX_GEO_MASK+1]
    no_change = x[:, :6, :, :]
    updated = x + y * x[:, 2:3, :, :]
    return torch.cat([no_change, updated[:, 6:, :, :]], dim=1)

  def seed(self, n, sz=128):
    return torch.zeros(n, self.chn, sz, sz)

def to_rgb(x):
  return x[...,:3,:,:]+0.5

param_n = sum(p.numel() for p in CA().parameters())
print('CA param count:', param_n)
```

```
CA param count: 5856
```

loss : 1.U ℓ Huber L1 noise ?Huber nosie

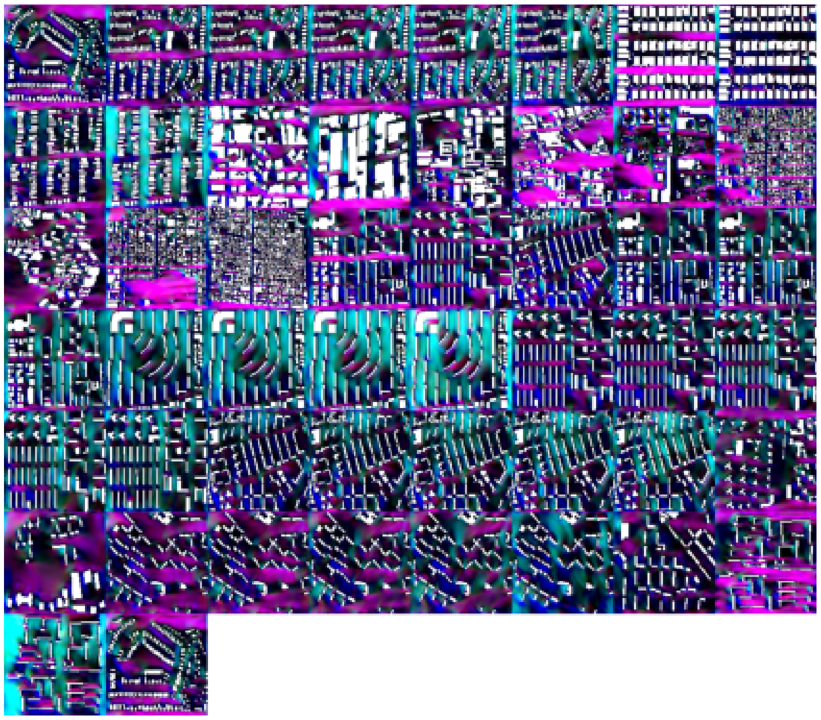
5 setup training

[6]:
```
device = torch.device("cuda" if torch.cuda.is_available() else "cpu")

DATASET_IMG_SIZE = (128,128)
dataset_dir = "../dataset_h5/train"
loaded_dataset,dataset_city_size = load_all_cases_to_BCHW(dataset_dir,␣
 ↪target_size=DATASET_IMG_SIZE)
print(loaded_dataset.shape)


CHANNEL_C = 24
LERNING_RATE = 1e-3
POOL_SIZE = 256
# setup training
ca = CA(chn=CHANNEL_C).to(device)
opt = torch.optim.Adam(ca.parameters(), LERNING_RATE, capturable=True)
lr_sched = torch.optim.lr_scheduler.MultiStepLR(opt, [1000, 2000], 0.3)
loss_log = []
metrics_log = []
# loaded_dataset = loaded_dataset[0:1]


gradient_checkpoints = False  # Set in case of OOM problems
```

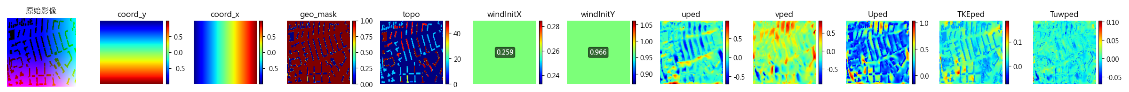
```
torch.Size([51, 11, 128, 128])
```

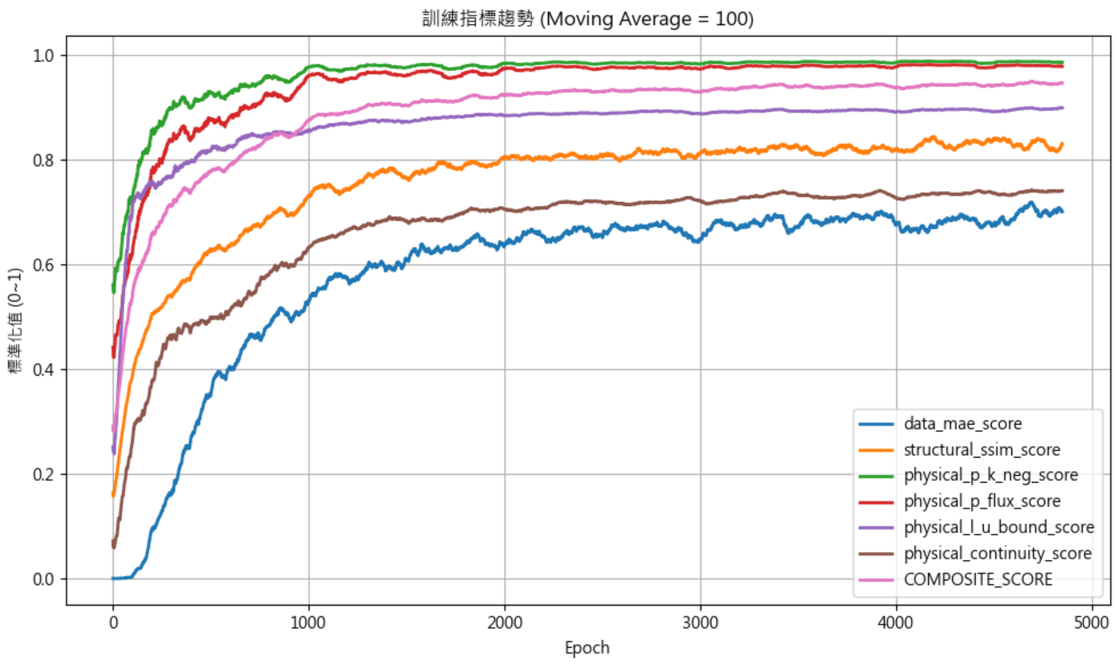
6 training loop

[7]:
```
# ---      (     ) ---
#       ca    opt
model_save_dir = get_output_path()   #
model_name_format = "ca_model_step_{}.pth" #           i

#   ca, opt, x_pool, y_pool, loss_f, lr_sched, metric_calculator,␣
 ↪gradient_checkpoints, loss_log, metrics_log

#
total_epochs =5000
for i in range(total_epochs):
    if i%500==0:
        x_pool, y_pool, pool_city_size = create_pool(loaded_dataset,␣
 ↪dataset_city_size, pool_size=POOL_SIZE, channel_c=CHANNEL_C)
        x_pool, y_pool = x_pool.to(device), y_pool.to(device)
        metric_calculator = ScaledPhysicsMetrics(y_pool,pool_city_size)
    # ---- 1.     (  GPU  ) ----
    with torch.no_grad():
        #    6
        batch_idx = np.random.choice(len(x_pool), 6, replace=False)
        batch_city_size = [pool_city_size[i] for i in batch_idx.tolist()]
        #       in-place
        x = x_pool[batch_idx].clone()
        y = y_pool[batch_idx]

        #   8          6
        if i % 8 == 0:
            x[:1, 6:, :, :] = 0

    #
    step_n = np.random.randint(32, 96)

    # ---- 2.      (Forward) ----
    if not gradient_checkpoints:
        #         step_n
        for k in range(step_n):
            x = ca(x)
    else:
        #
        x.requires_grad = True
        #   torch.utils.checkpoint
        x = torch.utils.checkpoint.checkpoint_sequential([ca]*step_n, 16, x)

    # ---- 3.     (Loss Calculation) ----
    #       x    [-1.0, 1.0]
    overflow_loss = (x - x.clamp(-1.0, 1.0)).abs().sum()
    #    =    (  6   10   ) +
    loss = loss_f(x[:,6:11], y[:,6:11]) + overflow_loss

    # ---- 4.       (Backward & Update) ----
    opt.zero_grad() #
    loss.backward() #

    with torch.no_grad():
        #     (Gradient Normalization)     L2
        for p in ca.parameters():
            if p.grad is not None:
                #
                p.grad /= (p.grad.norm() + 1e-8)

        opt.step() #
        lr_sched.step() #

        #          detach()
        x_pool[batch_idx] = x.detach()
    # ---- 5.    (Logging Loss) ----
    loss_log.append(loss.item())

    # ---- 6.    (Metric Calculation) ----
    with torch.no_grad():
        #
        metrics_result = metric_calculator(x, y,batch_city_size)
        #      CPU
        metrics_result = to_device(metrics_result, "cpu")
        metrics_log.append(metrics_result)

    # ---- 7.     ( 5 ) ----
    if i % 5 == 0:
        #   Markdown
        display(Markdown(f"""
        step_n: {len(loss_log)}
        loss: {loss.item()}
        lr: {lr_sched.get_last_lr()[0]}"""), display_id='stats')

    # ---- 8.      ( 50 ) ----
    if i % 50 == 0:
        clear_output(wait=True) #

        #
        viz_pool(x_pool, step_i=i)

        #
        plt_HWC_split_channels(to_HWC(x[0,:11]))

        #
        metrics_log = to_device(metrics_log, "cpu")
        plt_metrics_dict(metrics_log, window=100)

        #
        pl.plot(loss_log, '.', alpha=0.1)
        pl.yscale('log')
        pl.ylim(np.min(loss_log), loss_log[0])
        pl.tight_layout()
        imshow(grab_plot(), id='log') #

        #
        imgs = to_rgb(x[:,6:,:,:]).permute([0, 2, 3, 1]).detach().cpu().numpy()
        imshow(np.hstack(imgs), id='batch')
        target_imgs = to_rgb(y[:,6:,:,:]).permute([0, 2, 3, 1]).detach().cpu().
↪numpy()
        imshow(np.hstack(target_imgs), id='batch')
        # ---     (Save Model) ---
        #
        save_checkpoint(ca, opt, i, f"{model_save_dir}/ca_model_step_{i}.pth")


save_checkpoint(ca, opt, total_epochs, f"{model_save_dir}/
 ↪ca_model_step_{total_epochs}.pth")
```

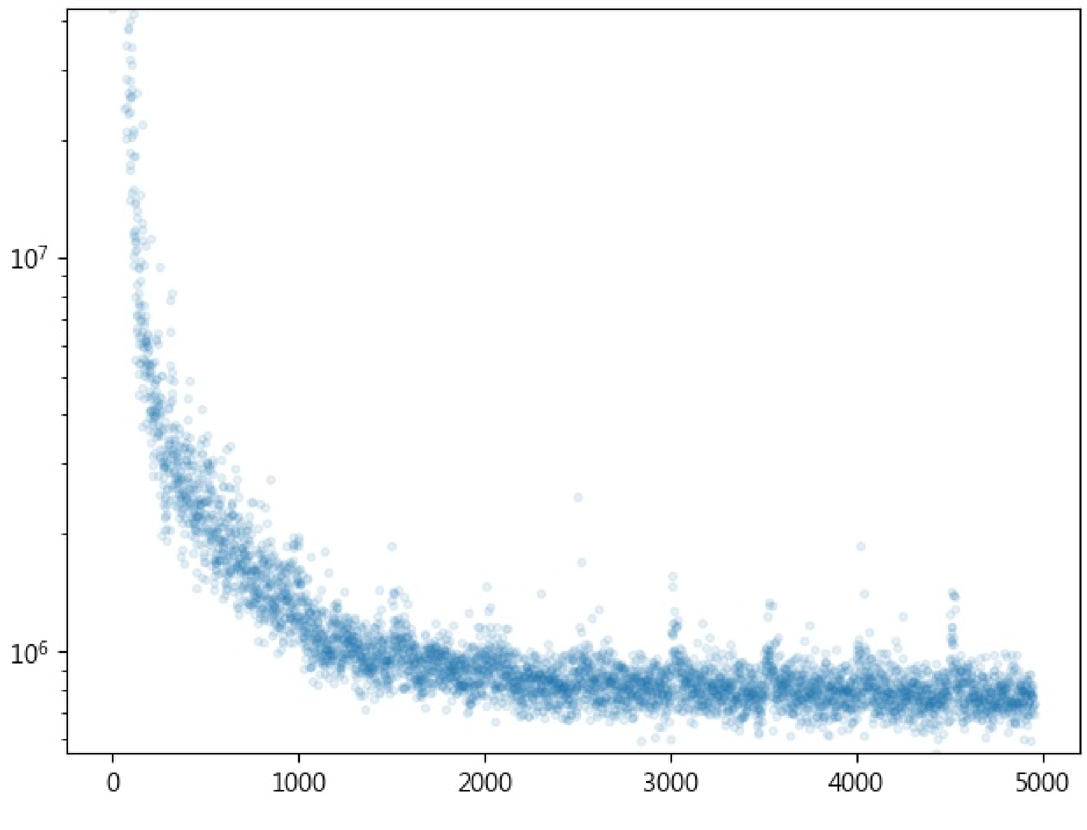


```
Clipping input data to the valid range for imshow with RGB data ([0..1] for
floats or [0..255] for integers). Got range [-0.9938965..1.0].
```

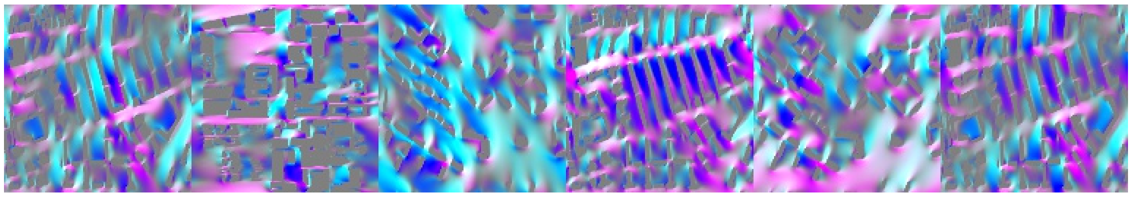




訓練指標趨勢 (Moving Average = 100)
標準化值 (0~1)
1.0
0.8
0.6
0.4
0.2
0.0
0
1000
2000
3000
4000
5000
Epoch
data_mae_score
structural_ssim_score
physical_p_k_neg_score
physical_p_flux_score
physical_l_u_bound_score
physical_continuity_score
COMPOSITE_SCORE



10^7
10^6
0
1000
2000
3000
4000
5000



```
  ..\outputs\E4-7.1_Init-NCA_20251102-153402/ca_model_step_4950.pth
  ..\outputs\E4-7.1_Init-NCA_20251102-153402\ca_model_step_4800.pth
  3
```

```
step_n: 4956
loss: 740612.75
lr: 8.999999999999999e-05

step_n: 4961
loss: 786226.875
lr: 8.999999999999999e-05

step_n: 4966
loss: 675797.0
lr: 8.999999999999999e-05

step_n: 4971
loss: 706239.625
lr: 8.999999999999999e-05

step_n: 4976
loss: 810384.5
lr: 8.999999999999999e-05

step_n: 4981
loss: 814943.0
lr: 8.999999999999999e-05

step_n: 4986
loss: 807717.75
lr: 8.999999999999999e-05

step_n: 4991
loss: 820642.25
lr: 8.999999999999999e-05
```

```
step_n: 4996
loss: 777261.375
lr: 8.999999999999999e-05
```

```
..\outputs\E4-7.1_Init-NCA_20251102-153402/ca_model_step_5000.pth
..\outputs\E4-7.1_Init-NCA_20251102-153402\ca_model_step_4850.pth
3
```

7 output video

[8]:
```
test_x_pool, test_y_pool = create_pool(loaded_dataset, pool_size=256,␣
 ↪channel_c=24)
test_x_pool, test_y_pool = test_x_pool.to(device), test_y_pool.to(device)
metric_calculator = ScaledPhysicsMetrics(test_y_pool)

with VideoWriter() as vid, torch.no_grad():
  batch_idx = np.random.choice(len(test_x_pool), 4, replace=False)
  test_x = test_x_pool[batch_idx].clone()  # clone  in-place
  test_y = test_y_pool[batch_idx]
  plt_HWC_split_channels(to_HWC(test_x[0,:11]))

  for k in tnrange(300, leave=False):
    step_n = min(2**(k//30), 8)
    for i in range(step_n):
      test_x[:] = ca(test_x)
    img = to_rgb(test_x[0,6:]).permute(1, 2, 0).cpu()
    vid.add(zoom(img, 2))
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
Cell In[8], line 1
----> 1 test_x_pool, test_y_pool =␣
  ↪create_pool(loaded_dataset, pool_size=256, channel_c=24)
      2 test_x_pool, test_y_pool = test_x_pool.to(device), test_y_pool.to(devic
      3 metric_calculator = ScaledPhysicsMetrics(test_y_pool)

TypeError: create_pool() missing 1 required positional argument: 'city_size'
```