CCP SIS _ 2023-10-24 _ Minkyeul Lee

Summary Report

Overview of what I learned

- Generate and save DLA data(p=0.01) for CNN
- PyTorch installation and tutorial
- Pretrained model: resnet18
- How to transform jpg image to vector
- Cosine similarity

Get DLA P(p=0.01) data

1) 목표: 확률 p에 따른 Stochastic DLA의 최종 모양을 예측하는 CNN 모델을 위한 모델 학습 데이터 500개 수집. (확률이 낮은 경우(p=0.01))
2) 0, 1로 이루어진 행렬의 형태로 txt 파일로 저장
3) 어려웠던 점

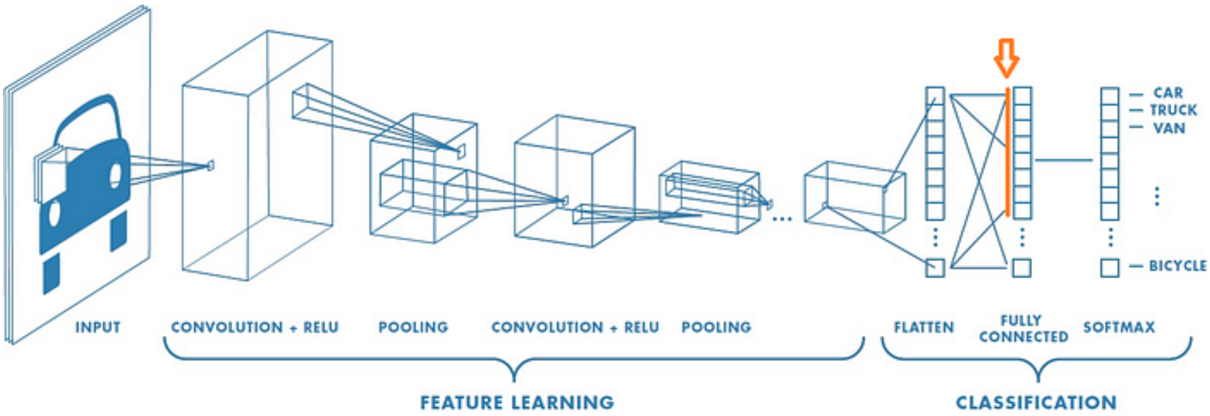
   - 확률 p값을 처음에 0.001로 했는데 데이터 1개당 약 3분, 500개의 데이터를 모으는 데 하루 이상의 시간이 걸리게 되므로, p값을 0.01로 수정하여 다시 데이터를 모았음

   - 추가로 데이터 수집 속도를 향상시키기 위해 로컬 컴퓨터에서 진행함

Example of CNN

1) PyTorch installation

   Python3.11 -m pip install img2vec_pytorch
2) Structure of CNN



   - avgpool layer
3) ResNet-18

   - expects images to be at least 224*224, as well as normalized with a specific mean and standard deviation

   - return vector length: 512
4) Transform jpg image to vector

   # 1. Load the image with Pillow library

   ```
   img = Image.open(image_name)
   ```

```
# 2. Create a PyTorch Variable with the transformed image
        t_img = Variable(normalize(to_tensor(scaler(img))).unsqueeze(0))
# 3. Create a vector of zeros that will hold our feature vector
#       The 'avgpool' layer has an output size of 512
        my_embedding = torch.zeros(512)
# 4. Define a function that will copy the output of a layer
        def copy_data(m, i, o):
                my_embedding.copy_(o.data)
# 5. Attach that function to our selected layer
        h = layer.register_forward_hook(copy_data)
# 6. Run the model on our transformed image
        model(t_img)
# 7. Detach our copy function from the layer
        h.remove()
# 8. Return the feature vector
        return my_embedding
```

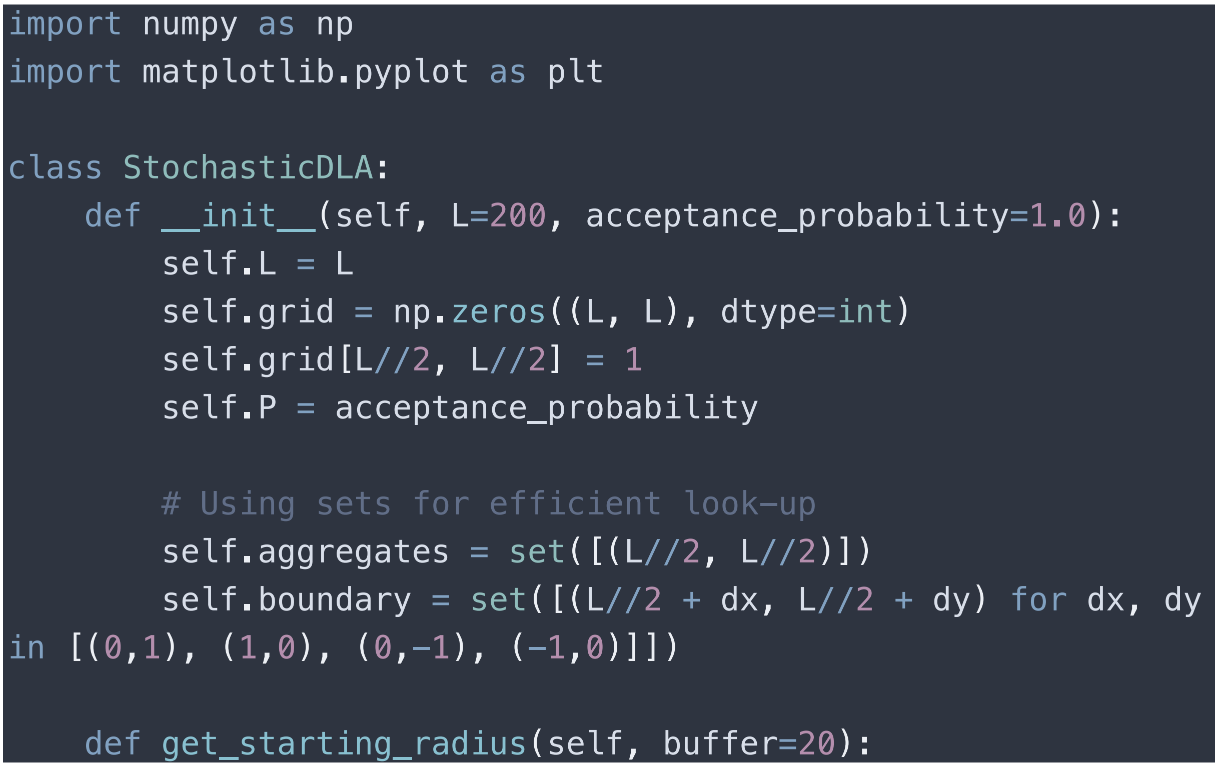
5) Cosine similarity

- torch.nn.CosineSimilarity(dim=1, eps=1e-08)

$$\text{similarity} = \frac{x_1 \cdot x_2}{\max(\|x_1\|_2 \cdot \|x_2\|_2, \epsilon)}.$$

- parameters
    - dim(int, optional)Dimension where cosine similarity is computed. Default 1
    - eps(float, optional)Small value to avoid division by zero. Default 1e-8

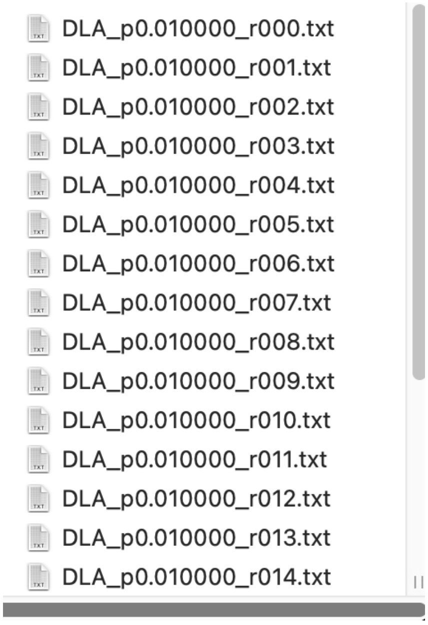
Code#1 Get DLA P(p=0.01) data



```
import numpy as np
import matplotlib.pyplot as plt

class StochasticDLA:
    def __init__(self, L=200, acceptance_probability=1.0):
        self.L = L
        self.grid = np.zeros((L, L), dtype=int)
        self.grid[L//2, L//2] = 1
        self.P = acceptance_probability

        # Using sets for efficient look-up
        self.aggregates = set([(L//2, L//2)])
        self.boundary = set([(L//2 + dx, L//2 + dy) for dx, dy
in [(0,1), (1,0), (0,-1), (-1,0)]])

    def get_starting_radius(self, buffer=20):
        max_dist = max(np.sqrt((x-self.L//2)**2 + (y-
self.L//2)**2) for x, y in self.boundary)
        return int(max_dist + buffer)

    def run(self, N, snapshot_intervals):
        directions = [(0,1), (1,0), (0,-1), (-1,0)]
        snapshots = []

        while len(self.aggregates) < N:
            r = self.get_starting_radius()
            theta = 2 * np.pi * np.random.random()
            x, y = int(self.L//2 + r * np.cos(theta)),
int(self.L//2 + r * np.sin(theta))
            walker_position = (x, y)

            while True:
                dx, dy = directions[np.random.randint(0, 4)]

                # Check boundaries
                if 0 < x + dx < self.L and 0 < y + dy <
self.L:
                    x += dx
                    y += dy
                    walker_position = (x, y)

                    # If walker is on boundary and it is an
empty site
                    if walker_position in self.boundary and
walker_position not in self.aggregates:
                        if np.random.random() < self.P:
                            # Update aggregates and boundary

self.aggregates.add(walker_position)
                            self.grid[x, y] = 1

                            # Remove current position from
boundary and add neighbors to boundary
```

```
self.boundary.remove(walker_position)
                            for dx, dy in directions:
                                neighbor = (x + dx, y + dy)
                                if 0 < neighbor[0] < self.L
and 0 < neighbor[1] < self.L and neighbor not in
self.aggregates:

self.boundary.add(neighbor)
                            break

            # Take snapshots at specified intervals
            if len(self.aggregates) in snapshot_intervals:
                snapshots.append(self.grid.copy())

        return snapshots

    def visualize(self, snapshots):
        fig, axes = plt.subplots(1, len(snapshots),
figsize=(15, 5))
        for ax, snapshot in zip(axes, snapshots):
            ax.imshow(snapshot, cmap='gray')
            ax.set_title(f"Aggregates: {np.sum(snapshot)}")
            ax.axis('off')
        plt.tight_layout()
        plt.show()

import os
os.chdir('CCPSIS_DLA_CNN_data_0.01/')

nrepeat = 500
for p in [0.01]:
  for r in range(nrepeat):
    dla = StochasticDLA(L=100, acceptance_probability=p)
    snapshots = dla.run(200, [200])
    #print (dla.grid)
    np.savetxt('DLA_p%f_r%03d.txt'%(p,r),dla.grid,fmt='%d')
    print(r)
```

result) p=0.01일 때의 DLA 데이터 500개 수집



DLA_p0.010000_r000.txt
DLA_p0.010000_r001.txt
DLA_p0.010000_r002.txt
DLA_p0.010000_r003.txt
DLA_p0.010000_r004.txt
DLA_p0.010000_r005.txt
DLA_p0.010000_r006.txt
DLA_p0.010000_r007.txt
DLA_p0.010000_r008.txt
DLA_p0.010000_r009.txt
DLA_p0.010000_r010.txt
DLA_p0.010000_r011.txt
DLA_p0.010000_r012.txt
DLA_p0.010000_r013.txt
DLA_p0.010000_r014.txt

Code#2 Example of CNN

```
import torch
import torch.nn as nn
import torchvision.models as models
import torchvision.transforms as transforms
from torch.autograd import Variable
from PIL import Image

for pic_two in ["cat2.jpg", "catdog.jpg", "face.jpg",
"face2.jpg"]:
    pic_one = "cat.jpg"

    # load the pretrained model
    model = models.resnet18(pretrained=True)

    # use the model object to select the desired layer
    layer = model._modules.get("avgpool")

    # set model to evaluation mode
    model.eval()

    scaler = transforms.Resize((224, 224))
    normalize = transforms.Normalize(
        mean=[0.485, 0.456, 0.406], std=[0.229, 0.224, 0.225]
    )
    to_tensor = transforms.ToTensor()

    def get_vector(image_name):
        # 1. Load the image with pillow library
        img = Image.open(image_name)

        # 2. Create a PyTorch Variable with the transformed
image
        t_img =
Variable(normalize(to_tensor(scaler(img))).unsqueeze(0))

        # 3. Create a vector of zeros that will hold our
feature vector
        # The 'avgpool' layer has an output size of 512
        my_embadding = torch.zeros(1, 512, 1, 512)

        # 4. Define a function that will copy the output of a
layer
        def copy_data(m, i, o):
            my_embadding.copy_(o.data)

        # 5. Attach that function to our selected layer
        h = layer.register_forward_hook(copy_data)

        # 6. Run the model on our transformed image
        model(t_img)

        # 7. Detach our copy function from the layer
        h.remove()

        # 8. Return the feature vector
        return my_embadding

    pic_one_vector = get_vector(pic_one)
    pic_two_vector = get_vector(pic_two)

    # Using PyTorch Cosine Similarity
    cos = nn.CosineSimilarity(dim=1, eps=1e-6)
    cos_sim = cos(pic_one_vector, pic_two_vector)
    print(f"\nCosine similarity with {pic_two}:
{cos_sim[0][0][0]}\n")
```

result)

```
Cosine  similarity  with  cat2.jpg:  0.7275727987289429
Cosine  similarity  with  catdog.jpg:  0.6408628225326538
Cosine  similarity  with  face.jpg:  0.5755692720413208
Cosine  similarity  with  face2.jpg:  0.5155152082443237
```