Finalized_working_code_ImageNet_VGG16_Visualizing_Embeddings_TCA

April 4, 2022

1 Building Neuron Tensors

Resources that have been useful: * [key resource] Stanford tutorial: https://cs231n.github.io/convolutional-networks/ * [key resource] Stackexchange that makes all this clear with a simple example: https://stackoverflow.com/questions/52272592/how-many-neurons-does-the-cnn-input-layer-have —- * https://adeshpande3.github.io/A-Beginner%27s-Guide-To-Understanding-Convolutional-Neural-Networks/ * https://adeshpande3.github.io/A-Beginner%27s-Guide-To-Understanding-Convolutional-Neural-Networks-Part-2/ * https://stanford.edu/~shervine/teaching/cs-230/cheatsheet-convolutional-neural-networks * https://hackernoon.com/learning-keras-by-implementing-vgg16-from-scratch-d036733f2d5

[ ]:
```
import tensorflow as tf
import numpy as np
import matplotlib.pyplot as plt
from tensorflow.python.keras.layers import InputLayer, Input
from keras.models import Model
from keras.layers import Dense
from keras.layers import Flatten
from tensorflow.python.keras import backend as K
import math
import pathlib
import datetime
from scipy.io import savemat
from scipy.io import loadmat
from sklearn.cluster import KMeans
from sklearn.decomposition import PCA
import time
```

[ ]:
```
!pip install git+https://github.com/ahwillia/tensortools
import tensortools as tt
from scipy.ndimage import gaussian_filter
```

[ ]:
```
import numpy as np
import torch
```

```python
import random
import os

default_seed = 4142
def seed_everything(seed = 1234):
  random.seed(seed)
  tseed = random.randint(1,1E6)
  tcseed = random.randint(1,1E6)
  npseed = random.randint(1,1E6)
  ospyseed = random.randint(1,1E6)
  torch.manual_seed(tseed)
  torch.cuda.manual_seed_all(tcseed)
  np.random.seed(npseed)
  os.environ['PYTHONHASHSEED'] = str(ospyseed)

seed_everything(default_seed)
```

- load a pre-trained network (VGG16)
- input images from CIFAR-10 to VGG16
- choose some random unit in one of the layers and plot its activity for different images

1.1 Load VGG16 model

[ ]:
```python
from keras.applications.vgg16 import VGG16
MAX_SIZE = 64
VGG16_Model = VGG16(include_top=False,␣
 ↪weights='imagenet',input_shape=(MAX_SIZE, MAX_SIZE, 3))
VGG16_Model.summary()
```

[ ]:
```python
# add new classifier layers
flat1 = Flatten()(VGG16_Model.layers[-1].output)
class1 = Dense(1024, activation='relu')(flat1)
output = Dense(10, activation='softmax')(class1)
VGG16_Model = Model(inputs=VGG16_Model.inputs, outputs=output)
```

[ ]:
```python
VGG16_Model.summary()
```

[ ]:
```python
from google.colab import drive
drive.mount('/content/drive/')
%cd '/content/drive/My Drive/Embeddings/code'
```

2 Get images:

[ ]:
```
!python3 -m pip install --upgrade pip
!python3 -m pip install --upgrade Pillow
from PIL import Image, ImageTk
img_array_starfish = np.load('imgnet_starfish.npy')
img_array_strawberry = np.load('imgnet_strawberry.npy')
img_array_husky = np.load('imgnet_husky.npy')
img_array_guitar = np.load('imgnet_guitar.npy')

def get_images_selected_classes(num_images, n_classes):
  images_selected_classes = []
  MAX_SIZE = 64
  for i in range(int(num_images/n_classes)):
    im = Image.fromarray(img_array_starfish[i,:,:,:].astype(np.uint8))
    im.thumbnail((MAX_SIZE, MAX_SIZE), Image.ANTIALIAS)
    images_selected_classes.append(np.array(im))
  for i in range(int(num_images/n_classes)):
    im = Image.fromarray(img_array_strawberry[i,:,:,:].astype(np.uint8))
    im.thumbnail((MAX_SIZE, MAX_SIZE), Image.ANTIALIAS)
    images_selected_classes.append(np.array(im))
  for i in range(int(num_images/n_classes)):
    im = Image.fromarray(img_array_husky[i,:,:,:].astype(np.uint8))
    im.thumbnail((MAX_SIZE, MAX_SIZE), Image.ANTIALIAS)
    images_selected_classes.append(np.array(im))
  for i in range(int(num_images/n_classes)):
    im = Image.fromarray(img_array_guitar[i,:,:,:].astype(np.uint8))
    im.thumbnail((MAX_SIZE, MAX_SIZE), Image.ANTIALIAS)
    images_selected_classes.append(np.array(im))
  ##  (#images, #nrow, #ncol, #channels)
  images_selected_classes = np.array(images_selected_classes)

  return images_selected_classes

num_images = 20
n_classes = 4
images_selected_classes = get_images_selected_classes(num_images, n_classes)
print(images_selected_classes.shape)
for i in range(20):
  plt.imshow(images_selected_classes[i])
  plt.show()
```

[ ]:
```
images_selected_classes.shape
```

3 Create 3D neuron tensor:

[ ]:
```python
def apply_all_shifts(im, shift_step):
    '''
    arg(s):
        DEBUG NOTE: PYTORCH IS (#channels, rows, cols)!!
        im, an image of shape (3, im_size, im_size)
    return:
        im_all_shifts, a list of all shifted images from the input image
        n_shifts, number of shifted images
    '''

    ## vertical size might not be the same as the horizontal, note that the␣
 ↪channel for tf and pytorch are in different dimension
    im_size_vertical = im.shape[0]
    im_size_horizontal = im.shape[1]

    n_shifts_vertical = int(math.ceil(im_size_vertical/ shift_step))
    n_shifts_horizontal = int(math.ceil(im_size_horizontal/ shift_step))
    n_shifts = n_shifts_vertical * n_shifts_horizontal + 1

    im_all_shifts = []
    # start with the unshifted im
    im_shift = im
    im_all_shifts.append(im)
    for i in range(n_shifts_vertical):
        ## for pytorch axis = 1, for tensorflow axis = 0
        im_shift = np.roll(im_shift, shift = shift_step, axis=0)
        im_all_shifts.append(im_shift)
        for j in range(n_shifts_horizontal):
            ## for pytorch axis = 2, for tensorflow axis = 1
            im_shift = np.roll(im_shift, shift = shift_step, axis=1)
            im_all_shifts.append(im_shift)

    im_all_shifts =  np.array(im_all_shifts)
    return im_all_shifts, n_shifts
```

Note: output size = [(W−K+2P)/S]+1 = (128-3)/3+1 = ~43

e.g. conv_layer W is the input volume - 128 K is the Kernel size - 3 P is the padding - 0 S is the stride - 3

edge_neuron = floor(filter_size /stride) = floor(3/3) = 1

[ ]:
```python
def compute_neuron_output(model, layer_indices, im_all_shifts,  max_indices =␣
 ↪None, n_max_feature_maps = 5, plot_activity = False):

    '''
```

```
    arg(s):
        layer_names: list of strings indicating the names of the layers we want␣
↪to take neuron outputs from
        im_all_shifts: all shifts of one particular image in the for loop
    return:

    '''
    print(im_all_shifts.shape)
    n_shifts = im_all_shifts.shape[0]
    n_layers = len(layer_indices)
    neuron_output_highest = []
    all_fm_avg = []
    im_all_shifts = np.array(im_all_shifts)

    is_first_layer = True
    for layer_index in layer_indices:
        layer = model.layers[layer_index]

        # check for convolutional layer
        if 'conv' not in layer.name:
            continue

        ## note: always take the images as inputs
        output_layer = K.function(inputs=[model.layers[1].input],
                                  outputs=[layer.output])

        ## output is n feature maps
        neuron_output = output_layer(im_all_shifts)[0]

        ## number of neurons is nrow * ncol * n_feature_maps
        n_shifts, n_row, n_col, n_feature_maps = neuron_output.shape[:]

        ## remove the neurons at the edges

        # filter_size = 3
        # shift_step = 3
        # edge_neuron = math.floor(filter_size / shift_step)
        edge_neuron = int(0.1 * n_row)
        neuron_output = neuron_output[:, edge_neuron:(n_row - edge_neuron),␣
↪edge_neuron:(n_col - edge_neuron),:]
        n_shifts, n_row, n_col, n_feature_maps = neuron_output.shape[:]
        neuron_output = neuron_output.reshape((n_shifts, n_row * n_col,␣
↪n_feature_maps))

        ## number of neurons for each feature map is nrow * ncol
        n_neurons = n_row * n_col
```

```
        # neuron_index = np.empty((n_row,n_col),dtype=int)
        # index = 0
        # for i in range(n_row):
        #     for j in range(n_col):
        #         neuron_index[i, j] = index
        #         index += 1

        # ## obtain the index of the neurons at the edges
        # neuron_edge_index = np.hstack((neuron_index[[0,1,n_row-2,n_row-1],:].
↪reshape((4*32,1)), neuron_index[:,[0,1,n_col-2,n_col-1]].reshape((4*32,1))))
        # neuron_edge_index = neuron_edge_index.reshape((256,1))


        # ## re-label the neurons at the edge with a different color
        # neuron_labels = []
        # for i in range(10):
        #     neuron_labels = np.hstack((neuron_labels, [i] * n_neurons))
        #     neuron_labels = np.array(neuron_labels)
        #     neuron_labels[n_neurons * i + neuron_edge_index-1] = 15

        # neuron_output = neuron_output.reshape((n_shifts, n_neurons,␣
↪n_feature_maps))

        ## transpose to organize by feature maps
        ## the shape of neuron_output_by_fm is (n_shifts, n_feature_maps,␣
↪#neurons)
        neuron_output_by_fm = neuron_output.transpose(0, 2, 1)

        ## compute avg neuron firing rate in each feature map
        ## fm_avg is of shape (n_shifts, n_feature_maps)
        fm_avg = neuron_output_by_fm.sum(axis=2) / neuron_output_by_fm.shape[2]

        if is_first_layer is True:
            fm_avg_all_layers = fm_avg
            neuron_output_by_fm_all_layers = neuron_output_by_fm
        else:
            fm_avg_all_layers = np.hstack((fm_avg_all_layers, fm_avg))
            neuron_output_by_fm_all_layers = np.
↪concatenate((neuron_output_by_fm_all_layers,neuron_output_by_fm), axis = 1)
            is_first_layer = False

    if plot_activity == True:
        print("Indices of FM with highest average firing rate in response to␣
↪each image: " )
        max_fm_ind = np.argmax(neuron_output_by_fm_all_layers,axis = 1)
        print(max_fm_ind)
```

```
        print("#neurons in the FM with highest average firing rate: " +␣
↪str(n_neurons))
        for f_i in range(n_feature_maps):
          plt.plot(fm_avg_all_layers[:,f_i]/max(fm_avg_all_layers[:,f_i]))
        plt.show()
        print('----------------------------------------------------\n')

    neuron_output_highest = np.empty((n_shifts, n_max_feature_maps * n_neurons))
    feature_map_side = int(np.sqrt(n_neurons))
    for i in range(n_shifts):
        if max_indices is None:
            max_indices = np.argsort(-1*fm_avg_all_layers[i], axis = 0)[:
↪n_max_feature_maps]
        temp = neuron_output_by_fm_all_layers[i, max_indices, :]
        neuron_output_highest[i] = neuron_output_by_fm_all_layers[i,␣
↪max_indices, :].reshape((1, n_max_feature_maps * n_neurons))

        for f_i in range(n_max_feature_maps):
          normalizing_constant = temp[f_i, :].max()
          if normalizing_constant == 0:
            neuron_output_highest[i, (f_i*feature_map_side**2):
↪((f_i+1)*feature_map_side**2)] = 0
          else:
            neuron_output_highest[i, (f_i*feature_map_side**2):
↪((f_i+1)*feature_map_side**2)] /= normalizing_constant

    neuron_labels = []
    for i in range(n_max_feature_maps):
      neuron_labels += [i] * n_neurons
    neuron_labels = np.array(neuron_labels)

    return neuron_output_highest, fm_avg_all_layers, neuron_labels,␣
↪max_indices, n_neurons
```

[ ]:
```
def show_stimuli_3D(model, layer_indices, images_selected_classes, shifts,␣
↪max_indices, n_images_selected_classes, shift_step = 3, n_max_feature_maps =␣
↪5, plot_activity = False):
    '''
    arg(s):
        layer_indices, interested layers
        images_selected_classes, all the selected images
        shifts = True/False
    return:
        neuron_output_shifts_avg, (n_images, n_neurons)
    '''
```

```
    if shifts is False:
        neuron_output_highest_final, fm_avg_all_layers, neuron_labels,␣
↪max_indices, n_neurons = compute_neuron_output(
            model, layer_indices, images_selected_classes, max_indices,␣
↪n_max_feature_maps, plot_activity)

    else:
        # generate shifts for each selected image and then stack:
        neuron_output_highest_with_shifts = []
        for i in range(n_images_selected_classes):
            im = images_selected_classes[i]
            im_all_shifts, n_shifts =  apply_all_shifts(im, shift_step)
            im_all_shifts = torch.Tensor(im_all_shifts)

            ## neuron_output_highest is of shape (n_shifts, n_neurons *␣
↪n_max_feature_maps)
            neuron_output_highest, fm_avg_all_layers, neuron_labels,␣
↪max_indices, n_neurons = compute_neuron_output(
                model, layer_indices, im_all_shifts, max_indices,␣
↪n_max_feature_maps, plot_activity)

            ## instead of taking average, we create a dimension for all shifts␣
↪(analogous to the time dimension)

            neuron_output_highest_with_shifts.append(neuron_output_highest)

            fm_shifts_avg = fm_avg_all_layers.sum(axis=0) / fm_avg_all_layers.
↪shape[0]
            fm_shifts_avg = fm_shifts_avg.reshape((1,fm_shifts_avg.shape[0]))

            if i == 0:
                fm_avg_all_layers_with_shifts = fm_shifts_avg
            else:
                fm_avg_all_layers_with_shifts = np.
↪vstack((fm_avg_all_layers_with_shifts, fm_shifts_avg))

        ## out of for loop!
        ## neuron_output_highest_with_shifts is of shape␣
↪(n_images_selected_classes, n_shifts, n_neurons * n_max_feature_maps)
        neuron_output_highest_with_shifts = np.
↪array(neuron_output_highest_with_shifts)
        neuron_output_highest_final = neuron_output_highest_with_shifts
        fm_avg_all_layers = fm_avg_all_layers_with_shifts

    return neuron_output_highest_final, fm_avg_all_layers, neuron_labels,␣
↪max_indices
```

```
[ ]: neuron_output_highest_with_shifts, fm_avg_all_layers_with_shifts,␣
     ↪neuron_labels, max_indices = show_stimuli_3D(
        VGG16_Model, range(2), images_selected_classes, shifts = True,  max_indices␣
     ↪= None, n_images_selected_classes = num_images, n_max_feature_maps=10,␣
     ↪plot_activity = True)
```

```
[ ]: neuron_output_highest_with_shifts.shape
```

```
[ ]: data = np.transpose(neuron_output_highest_with_shifts, (2, 0, 1))
     from scipy.io import savemat
     mdic = {"neuron_output_3D": data}
     mdic
     savemat("neuron_output_3D_vgg_block1_10fm.mat", mdic)
```

```
[ ]: data.shape
```

```
[ ]: neuron_output_highest_with_shifts, fm_avg_all_layers_with_shifts,␣
     ↪neuron_labels, max_indices = show_stimuli_3D(
        VGG16_Model, range(7,11), images_selected_classes, shifts = True, ␣
     ↪max_indices = None, n_images_selected_classes = num_images,␣
     ↪n_max_feature_maps=5, plot_activity = False)
```

```
[ ]: data = np.transpose(neuron_output_highest_with_shifts, (2, 0, 1))
     from scipy.io import savemat
     mdic = {"neuron_output_3D": data}
     mdic
     savemat("neuron_output_3D_vgg_block3_20fm.mat", mdic)
```

```
[ ]: data.shape
```

484 is the number of shifts = 22 * 22 5 is the number of selected images 19220 = 5 * 62 * 62 , where 5 is the number of feature maps and 62 is the number of neurons

4 Create 2D neuron tensor:

```
[ ]: def show_stimuli_2D(model, layer_indices, images_selected_classes, shifts,␣
     ↪max_indices, n_images_selected_classes, shift_step = 3, n_max_feature_maps =␣
     ↪5, plot_activity = False):
         '''
         arg(s):
             layer_indices, interested layers
             images_selected_classes, all the selected images
             shifts = True/False
         return:
             neuron_output_shifts_avg, (n_images, n_neurons)
         '''
```

```
    if shifts is False:
        neuron_output_highest_final, fm_avg_all_layers, neuron_labels,␣
↪max_indices, n_neurons = compute_neuron_output(
            model, layer_indices, images_selected_classes, max_indices,␣
↪n_max_feature_maps, plot_activity)


    else:
        # generate shifts for each selected image and then stack:
        neuron_output_highest_with_shifts = []
        for i in range(n_images_selected_classes):
            im = images_selected_classes[i]
            im_all_shifts, n_shifts =  apply_all_shifts(im, shift_step)
            im_all_shifts = torch.Tensor(im_all_shifts)

            ## neuron_output_highest is of shape (n_shifts, n_neurons *␣
↪n_max_feature_maps)
            neuron_output_highest, fm_avg_all_layers, neuron_labels,␣
↪max_indices, n_neurons = compute_neuron_output(
                model, layer_indices, im_all_shifts, max_indices,␣
↪n_max_feature_maps, plot_activity)


            ## take the average over all shifts of im
            neuron_output_highest_shifts_avg = neuron_output_highest.
↪sum(axis=0) / neuron_output_highest.shape[0]
            neuron_output_highest_shifts_avg = neuron_output_highest_shifts_avg.
↪reshape((1, neuron_output_highest_shifts_avg.shape[0]))


            fm_shifts_avg = fm_avg_all_layers.sum(axis=0) / fm_avg_all_layers.
↪shape[0]
            fm_shifts_avg = fm_shifts_avg.reshape((1,fm_shifts_avg.shape[0]))

            if i == 0:
              neuron_output_highest_with_shifts =␣
↪neuron_output_highest_shifts_avg
              fm_avg_all_layers_with_shifts = fm_shifts_avg
            else:
              neuron_output_highest_with_shifts = np.
↪vstack((neuron_output_highest_with_shifts, neuron_output_highest_shifts_avg))
              fm_avg_all_layers_with_shifts = np.
↪vstack((fm_avg_all_layers_with_shifts, fm_shifts_avg))

        ## out of for loop!
```

```
        ## neuron_output_highest_with_shifts is of shape␣
↪(n_images_selected_classes, n_neurons * n_max_feature_maps)
        neuron_output_highest_with_shifts = np.
↪array(neuron_output_highest_with_shifts)
        neuron_output_highest_final = neuron_output_highest_with_shifts
        fm_avg_all_layers = fm_avg_all_layers_with_shifts

    return neuron_output_highest_final, fm_avg_all_layers, neuron_labels,␣
↪max_indices
```

[ ]:
```
neuron_output_highest_with_shifts, fm_avg_all_layers_with_shifts,␣
↪neuron_labels, max_indices = show_stimuli_2D(
    VGG16_Model, range(2), images_selected_classes, shifts = True,  max_indices␣
↪= None, n_images_selected_classes = num_images, n_max_feature_maps=5,␣
↪plot_activity = True)
```

[ ]:
```
neuron_output_highest_with_shifts.shape
```

[ ]:
```
data = np.transpose(neuron_output_highest_with_shifts, (1,0))
from scipy.io import savemat
mdic = {"neuron_output_2D": data}
mdic
savemat("neuron_output_2D_vgg_block1.mat", mdic)
```

[ ]:
```
neuron_output_highest_with_shifts, fm_avg_all_layers_with_shifts,␣
↪neuron_labels, max_indices = show_stimuli_2D(
    VGG16_Model, range(7,11), images_selected_classes, shifts = True, ␣
↪max_indices = None, n_images_selected_classes = num_images,␣
↪n_max_feature_maps=5, plot_activity = True)
```

[ ]:
```
data = np.transpose(neuron_output_highest_with_shifts, (1,0))
from scipy.io import savemat
mdic = {"neuron_output_2D": data}
mdic
savemat("neuron_output_2D_vgg_block3.mat", mdic)
```

5 Streamline tensor factorization:

[ ]:
```
from scipy.io import loadmat
N_vgg_3D_block1 = loadmat("neuron_output_3D_vgg_block1.mat")['neuron_output_3D']
N_vgg_3D_block3 = loadmat("neuron_output_3D_vgg_block3.mat")['neuron_output_3D']
N_vgg_2D_block1 = loadmat("neuron_output_2D_vgg_block1.mat")['neuron_output_2D']
N_vgg_2D_block3 = loadmat("neuron_output_2D_vgg_block3.mat")['neuron_output_2D']
```

```
[ ]: from scipy.io import loadmat
     N_vgg_3D_block1 = loadmat("neuron_output_3D_vgg_block1_10fm.
      ↪mat")['neuron_output_3D']
```

```
[ ]: def get_tensor_factors(N, dim = 3, ranks = [10, 20, 30], reps = 1):
     ## note that for 2D tensor (ie a matrix), rank + nullity = num_columns => rank␣
      ↪<= num_columns
       if dim == 3:
         N_filtered = N
         # N_filtered = np.empty(N.shape)
         # for i in range(N.shape[0]):
         #   for j in range(N.shape[1]):
         #     filtered = gaussian_filter(N[i,j,:].reshape((n_vertical_shifts,␣
      ↪n_vertical_shifts)), sigma=1).reshape((n_vertical_shifts *␣
      ↪n_vertical_shifts,))
         #     N_filtered[i,j,:] = filtered[:]
       else:
         N_filtered = N.reshape((N.shape[0], N.shape[1],1))

       # Fit ensembles of tensor decompositions:
       methods = (
         'ncp_hals',  # fits nonnegative tensor decomposition.
       )

       ensembles = {}
       for m in methods:
           ensembles[m] = tt.Ensemble(fit_method=m,  fit_options=dict(tol=1e-5))
           ensembles[m].fit(N_filtered, ranks=ranks, replicates=reps)
           ## replicates: int, number of models to fit at each rank

       ## plot objective, similarity, factors:
       """
       Customized plotting routines for CP decompositions
       """

       # Plotting options for the unconstrained and nonnegative models.
       plot_options = {
         'ncp_hals': {
           'line_kw': {
             'color': 'blue',
             'alpha': 1,
             'label': 'ncp_hals',
           },
           'scatter_kw': {
             'color': 'blue',
             'alpha': 1,
             's': 1,
```

```
        },
    },
}

def plot_objective(ensemble, partition='train', ax=None, jitter=0.1,
                   scatter_kw=dict(), line_kw=dict()):
    """Plots objective function as a function of model rank.
    Parameters
    ----------
    ensemble : Ensemble object
        holds optimization results across a range of model ranks
    partition : string, one of: {'train', 'test'}
        specifies whether to plot the objective function on the training
        data or the held-out test set.
    ax : matplotlib axis (optional)
        axis to plot on (defaults to current axis object)
    jitter : float (optional)
        amount of horizontal jitter added to scatterpoints (default=0.1)
    scatter_kw : dict (optional)
        keyword arguments for styling the scatterpoints
    line_kw : dict (optional)
        keyword arguments for styling the line
    """

    if ax is None:
        ax = plt.gca()

    if partition == 'train':
        pass
    elif partition == 'test':
        raise NotImplementedError('Cross-validation is on the TODO list.')
    else:
        raise ValueError("partition must be 'train' or 'test'.")

    # compile statistics for plotting
    x, obj, min_obj = [], [], []
    for rank in sorted(ensemble.results):
        # reconstruction errors for rank-r models
        o = ensemble.objectives(rank)
        obj.extend(o)
        x.extend(np.full(len(o), rank))
        min_obj.append(min(o))

    print(o)
    print(obj)
    print(x)
    # add horizontal jitter
```

```
    ux = np.unique(x)
    x = np.array(x) + (np.random.rand(len(x))-0.5)*jitter

    # make plot
    # customized: plot objectives for all iterations
    ax.scatter(x, obj, **scatter_kw)
    ax.plot(ux, min_obj, **line_kw)
    ax.set_xlabel('model rank')
    ax.set_ylabel('objective')

    return ax


def plot_similarity(ensemble, ax=None, jitter=0.1,
                    scatter_kw=dict(), line_kw=dict()):
    """Plots similarity across optimization runs as a function of model rank.
    Parameters
    ----------
    ensemble : Ensemble object
        holds optimization results across a range of model ranks
    ax : matplotlib axis (optional)
        axis to plot on (defaults to current axis object)
    jitter : float (optional)
        amount of horizontal jitter added to scatterpoints (default=0.1)
    scatter_kw : dict (optional)
        keyword arguments for styling the scatterpoints
    line_kw : dict (optional)
        keyword arguments for styling the line
    References
    ----------
    Ulrike von Luxburg (2010). Clustering Stability: An Overview.
    Foundations and Trends in Machine Learning.
    https://arxiv.org/abs/1007.1075
    """

    if ax is None:
        ax = plt.gca()

    # compile statistics for plotting
    x, sim, mean_sim = [], [], []
    for rank in sorted(ensemble.results):
        # reconstruction errors for rank-r models
        s = ensemble.similarities(rank)[1:]
        sim.extend(s)
        x.extend(np.full(len(s), rank))
        mean_sim.append(np.mean(s))
```

```
        # add horizontal jitter
        ux = np.unique(x)
        x = np.array(x) + (np.random.rand(len(x))-0.5)*jitter

        # make plot
        # customized: plot similarities for all iterations
        ax.scatter(x, sim, **scatter_kw)
        ax.plot(ux, mean_sim, **line_kw)

        ax.set_xlabel('model rank')
        ax.set_ylabel('model similarity')
        ax.set_ylim([0, 1.1])

        return ax

    # Plot similarity and error plots.
    plt.figure()
    for m in methods:
        plot_objective(ensembles[m], **plot_options[m])
    plt.legend()

    # plt.figure()
    # for m in methods:
    #     plot_similarity(ensembles[m], **plot_options[m])
    # plt.legend()

    plt.show()

    return ensembles ## A LIST!
```

[ ]:
```
def get_embeddings(tensor_factors_best_rank,neuron_labels,n_dim_PCA = 10):
    neuron_factor = tensor_factors_best_rank[0]
    # PCA on tensor factors
    pca = PCA(n_dim_PCA)
    neuron_output_highest_with_shifts_PCA = pca.fit_transform(neuron_factor)
    neuron_output_highest_with_shifts_PCA.shape
    vectors, lambdas = pca.components_, pca.explained_variance_
    plt.plot(pca.explained_variance_ratio_)
    plt.show()

    # plot embeddings:

    import plotly.graph_objects as go
    import plotly.express as px

    fig = go.Figure()
    traces = []
```

```
    colors_palette = px.colors.qualitative.Dark24
    data = neuron_output_highest_with_shifts_PCA
    print(data.shape)
    for i, label in enumerate(set(neuron_labels)):
        mask = (neuron_labels == label)
        print(mask.shape)
        print(label, sum(mask))
        traces.append(go.Scatter3d(
            x=data[mask,0],
            y=data[mask,1],
            z=data[mask,2],
            mode='markers',
            marker=dict(
                size=4,
                color=colors_palette[int(label)],
                opacity=1,
                #showscale= True,
            )))

    for trace in traces:
        fig.add_trace(trace)
    fig.update_layout(

      width=700,
      margin=dict(r=20, l=10, b=10, t=10))

    fig.update_layout(margin=dict(l=0, r=0, b=0, t=0),showlegend=True,
                      # scene = dict(xaxis = dict(range=[-1.5,2.5],),
                      #              yaxis = dict(range=[-1.5,1.5],),
                      #              zaxis = dict(range=[-1.5,1.5],),),
                      )
    fig.show()
    return neuron_output_highest_with_shifts_PCA
```

[ ]:
```
def colorFromUnivariateData(Z1, cmap1 = plt.cm.Reds):
    # Rescale values to fit into colormap range (0->255)
    Z1_plot = np.array(255*(Z1-Z1.min())/(Z1.max()-Z1.min()), dtype=np.int)

    Z1_color = cmap1(Z1_plot)

    # Color for each point
    Z_color = np.array(Z1_color[:,0:3])
    return Z_color

## ## https://stackoverflow.com/questions/49871436/
 ↪scatterplot-with-continuous-bivariate-color-palette-in-python
```

```
def colorFromBivariateData(Z1,Z2,cmap1 = plt.cm.Blues, cmap2 = plt.cm.Reds):
    # Rescale values to fit into colormap range (0->255)
    Z1_plot = np.array(255*(Z1-Z1.min())/(Z1.max()-Z1.min()), dtype=np.int)
    Z2_plot = np.array(255*(Z2-Z2.min())/(Z2.max()-Z2.min()), dtype=np.int)

    Z1_color = cmap1(Z1_plot)
    Z2_color = cmap2(Z2_plot)

    # Color for each point
    Z_color = np.sum([Z1_color , Z2_color ], axis=0)/2.0
    Z_color = np.array(Z_color[:,0:3])
    return Z_color

def get_spatial_order_plot(cluster_index,␣
↪neuron_output_highest_with_shifts_PCA, neuron_labels, n_max_feature_maps =␣
↪5):
  n_neurons = int(neuron_labels.shape[0]/n_max_feature_maps)
  feature_map_side = int(np.sqrt(n_neurons))
  neuron_cluster = neuron_output_highest_with_shifts_PCA[cluster_index *␣
↪n_neurons : (cluster_index + 1) * n_neurons]
  neuron_labels_cluster = neuron_labels[cluster_index * n_neurons :␣
↪(cluster_index + 1) * n_neurons]
  xs, ys = np.mgrid[0:feature_map_side,0:feature_map_side]
  xs = xs.reshape((n_neurons,))
  ys = ys.reshape((n_neurons,))

  import plotly.graph_objects as go
  import plotly.express as px

  fig = go.Figure()
  traces = []
  colors_palette = px.colors.qualitative.Dark24
  data = neuron_cluster

  for i, label in enumerate(set(neuron_labels)):
      mask = (neuron_labels_cluster == label)
      print(label, sum(mask))
      traces.append(go.Scatter3d(
          x=data[mask,0],
          y=data[mask,1],
          z=data[mask,2],
          mode='markers',
          marker=dict(
              size=4,
              color=colors_palette[int(label)],
              opacity=1,
              #showscale= True,
```

```
        )))
for trace in traces:
    fig.add_trace(trace)
fig.update_layout(margin=dict(l=0, r=0, b=0, t=0),showlegend=True,)
fig.show()

fig = plt.figure(figsize=(12, 12))
ax = fig.add_subplot(projection='3d')
data = neuron_cluster
ax.scatter(
        data[:,0],
        data[:,1],
        data[:,2],
        c = colorFromUnivariateData(xs, cmap1 = plt.cm.Blues),
        s = 100,
        alpha= 1
        )
plt.show()

fig = plt.figure(figsize=(12, 12))
ax = fig.add_subplot(projection='3d')
data = neuron_cluster
ax.scatter(
        data[:,0],
        data[:,1],
        data[:,2],
        c = colorFromUnivariateData(ys, cmap1 = plt.cm.Reds),
        s = 100,
        alpha= 1
        )
plt.show()
```

[ ]:
```
def get_tensor_factors_plot(tensor_factors_best_rank, best_rank,␣
↪n_max_feature_maps):
  ## neuron_factor shape: (#neuron, #factors=best_rank)
  neuron_factor = tensor_factors_best_rank[0]
  ## neuron_factor shape: (#shifts, #factors=best_rank)
  time_factor = tensor_factors_best_rank[2]

  # neuron_factor_first = neuron_factor[:,0]
  plt.plot(neuron_factor)
  plt.show()

  # time_factor_first = time_factor[:,0]
  plt.plot(time_factor)
  plt.show()
```

```
  n_neurons = neuron_factor.shape[0]
  feature_map_side = int(np.sqrt(n_neurons/n_max_feature_maps))
  for i in range(best_rank):
    neuron_factor_i_th = neuron_factor[:,i]
    vmin = neuron_factor_i_th.min()
    vmax = neuron_factor_i_th.max()
    print(neuron_factor.shape)
    f_i, axes = plt.subplots(1,n_max_feature_maps,figsize=(10,1))
    for f_i, ax in enumerate(axes):
      feature_map_matrix = neuron_factor_i_th[(f_i * feature_map_side ** 2):
 ↪((f_i+1) * feature_map_side ** 2)].reshape((feature_map_side,
 ↪feature_map_side))
      ax.imshow(feature_map_matrix, vmin = vmin, vmax = vmax)
      ax.set(xticks = [], yticks = [])
    plt.show()
```

##3D:

[ ]:
```
N_vgg_3D_block1.shape
```

Check shifts:

[ ]:
```
im_ind = 0
n_max_feature_maps = 5
feature_map_side = 62
# f_i, axes = plt.subplots(1,n_max_feature_maps,figsize=(10,1))
neuron_matrix = N_vgg_3D_block1[:,im_ind,:]
for f_i in range(n_max_feature_maps):
  neuron_matrix_fi =  neuron_matrix[(f_i * feature_map_side ** 2): ((f_i+1) *
 ↪feature_map_side ** 2)]
  N_frames = 484
  feature_map_side = 62
  for t in range(N_frames):
    neuron_matrix_fi_t = neuron_matrix_fi[:, t].reshape((feature_map_side,
 ↪feature_map_side))
    print(neuron_matrix_fi_t)
    vmin = neuron_matrix_fi_t.min()
    vmax = neuron_matrix_fi_t.max()
    plt.imshow(neuron_matrix_fi_t, vmin = vmin, vmax = vmax)
    # plt.set(xticks = [], yticks = [])
    plt.show()
```

[ ]:
```
## pick the central neurons (15,15)
```

[ ]:
```
ensembles = get_tensor_factors(N_vgg_3D_block1, ranks = [30])
```

```
[ ]: rep = 0
     ranks = [30]
     tensor_factors = []
     for rank in ranks:
       tensor_factors.append(ensembles['ncp_hals'].results[rank][rep].factors)
     tensor_factors_best_rank = tensor_factors[0]
```

```
[ ]: tensor_factors_best_rank[0].shape
```

```
[ ]: tensor_factors_best_rank[2].shape
```

```
[ ]: get_tensor_factors_plot(tensor_factors_best_rank, best_rank=30,␣
      ↪n_max_feature_maps=5)
     neuron_output_highest_with_shifts_PCA =␣
      ↪get_embeddings(tensor_factors_best_rank, neuron_labels, n_dim_PCA = 10)
     get_spatial_order_plot(1, neuron_output_highest_with_shifts_PCA, neuron_labels)
```

```
[ ]: ensembles = get_tensor_factors(N_vgg_3D_block3, ranks = [30])
     rep = 0
     ranks = [30]
     tensor_factors = []
     for rank in ranks:
       tensor_factors.append(ensembles['ncp_hals'].results[rank][rep].factors)
     tensor_factors_best_rank = tensor_factors[0]
     get_tensor_factors_plot(tensor_factors_best_rank, best_rank=30,␣
      ↪n_max_feature_maps=5)
```

```
[ ]: neuron_labels = []
     n_max_feature_maps = 5
     for i in range(n_max_feature_maps):
       neuron_labels += [i] * 196
     neuron_labels = np.array(neuron_labels)

     neuron_output_highest_with_shifts_PCA =␣
      ↪get_embeddings(tensor_factors_best_rank, neuron_labels, n_dim_PCA = 10)
     get_spatial_order_plot(1, neuron_output_highest_with_shifts_PCA, neuron_labels)
```

```
[ ]: neuron_output_highest_with_shifts.shape
```

```
[ ]: fm_avg_all_layers_with_shifts.shape
```

5.1 2D:

```
[ ]: N_vgg_2D_block1.shape
```

```
[ ]: ensembles_2D = get_tensor_factors(N_vgg_2D_block1, dim =2, ranks=[1,2,3,4,5])
```

[ ]:
```
rep = 0
ranks=[1,2,3,4,5]
tensor_factors = []
for rank in ranks:
  tensor_factors.append(ensembles_2D['ncp_hals'].results[rank][rep].factors)
best_rank = 3
best_rank_ind = 2
tensor_factors_best_rank = tensor_factors[best_rank_ind]

neuron_labels = []
n_max_feature_maps = 5
for i in range(n_max_feature_maps):
  neuron_labels += [i] * 2704
neuron_labels = np.array(neuron_labels)
```

[ ]:
```
get_tensor_factors_plot(tensor_factors_best_rank, best_rank=2,␣
 ↪n_max_feature_maps=5)
neuron_output_highest_with_shifts_PCA =␣
 ↪get_embeddings(tensor_factors_best_rank, neuron_labels, n_dim_PCA = 3)
get_spatial_order_plot(1, neuron_output_highest_with_shifts_PCA, neuron_labels)
```

[ ]:
```
ensembles_2D = get_tensor_factors(N_vgg_2D_block3, dim =2, ranks=[1,2,3,4,5])
```

[ ]:
```
rep = 0
ranks=[1,2,3,4,5]
tensor_factors = []
for rank in ranks:
  tensor_factors.append(ensembles_2D['ncp_hals'].results[rank][rep].factors)
best_rank = 3
best_rank_ind = 2
tensor_factors_best_rank = tensor_factors[best_rank_ind]
get_tensor_factors_plot(tensor_factors_best_rank, best_rank=2,␣
 ↪n_max_feature_maps=5)
neuron_output_highest_with_shifts_PCA =␣
 ↪get_embeddings(tensor_factors_best_rank, neuron_labels, n_dim_PCA = 3)
get_spatial_order_plot(1, neuron_output_highest_with_shifts_PCA, neuron_labels)
```

6 Retina:

[ ]:
```
N_retina1 = loadmat("retina1.mat")['X']
N_retina1.shape
# (n_neurons, stimuli, time)
# (n_neurons, n_images, n_shifts)
```

[ ]:
```
N_retina2 = loadmat("retina2.mat")['X']
N_retina2.shape
```

[ ]:
```
33*8
```

[ ]:
```
ensembles = get_tensor_factors(N_retina1, ranks = [300,400])
```

[ ]:
```
def get_tensor_factors_plot_retina(tensor_factors_best_rank, best_rank):
  ## neuron_factor shape: (#neuron, #factors=best_rank)
  neuron_factor = tensor_factors_best_rank[0]
  ## neuron_factor shape: (#shifts, #factors=best_rank)
  time_factor = tensor_factors_best_rank[2]

  # neuron_factor_first = neuron_factor[:,0]
  plt.plot(neuron_factor)
  plt.show()

  # time_factor_first = time_factor[:,0]

  for i in range(50):
    plt.imshow(time_factor[:,i].reshape(8,33))
    plt.show()

  stimuli_factor = tensor_factors_best_rank[1]
  plt.plot(stimuli_factor)
  plt.show()


  # n_neurons = neuron_factor.shape[0]
  # feature_map_side_row = 8
  # feature_map_side_col = 33
  # for i in range(best_rank):
  #   neuron_factor_i_th = neuron_factor[:,i]
  #   vmin = neuron_factor_i_th.min()
  #   vmax = neuron_factor_i_th.max()
  #   print(neuron_factor.shape)
  #   f_i, axes = plt.subplots(1,n_max_feature_maps,figsize=(10,1))
  #   for f_i, ax in enumerate(axes):
  #     feature_map_matrix = neuron_factor_i_th[(f_i * feature_map_side ** 2):
 ↪((f_i+1) * feature_map_side ** 2)].reshape((feature_map_side,
 ↪feature_map_side))
  #     ax.imshow(feature_map_matrix, vmin = vmin, vmax = vmax)
  #     ax.set(xticks = [], yticks = [])
  #   plt.show()
```

[ ]:
```
tensor_factors_best_rank[2].shape
```

[ ]:
```
rep = 0
ranks = [400]
tensor_factors = []
```

```
for rank in ranks:
  tensor_factors.append(ensembles['ncp_hals'].results[rank][rep].factors)
tensor_factors_best_rank = tensor_factors[0]
get_tensor_factors_plot_retina(tensor_factors_best_rank, best_rank=400)
neuron_output_highest_with_shifts_PCA =␣
↪get_embeddings(tensor_factors_best_rank, neuron_labels, n_dim_PCA = 15)
get_spatial_order_plot(1, neuron_output_highest_with_shifts_PCA, neuron_labels)
```

[ ]:

```

```