This document accompanies the submission of the paper "Intrinsic dimension of data representations in deep neural networks".

Main requirements

The code can be run on a python environment. The *main* requirements are (we specify also the exact versions in which we tested it)

- python 3.6.7
- PyTorch 1.0 (torch 1.0.1.post2, torchvision 0.2.2)
- numpy (1.16.2)
- scikit.learn (0.20.3)
- tqdm (4.31.1)
- torchsummary

Code

The code is part of these supplementary materials [1]. Here and in what follow we will refer with root as the root directory of the code

`IntrinsicDimDeep`

Data

The data required to run all the experiments is made available through the the following link.

Download and unzip it in the root directory of the code. The ImageNet data samples are samples from the training set used in ILSVRC2014 (DET dataset).

Reproducing our results

Hereafter we give instructions as to reproduce the main results and figures.

We follow the order of appearance in the main text.

All the scripts are contained in

`scripts/***`

where *** will be specified in each case.

[1] We will create a public repository in case of acceptance.

Fig 2: (VGG-16-R on custom dataset)

You can plot the results of a run just by opening the jupyter notebook

```
scripts/custom/plot.ipynb
```

Otherwise you can do the training and analyze your results in the following way.

- Before launching the training change dir in data/custom and generate the train/test split

  ```
  $ python train_test_split.py
  ```

- Then go back to scripts/custom and launch finetuning

  ```
  python finetune.py
  ```

  this will finetune a VGG-16 pre-trained on ImageNet (it will also extract representations and save them in data/custom/results by default).

- Then run the bash script

  ```
  $ ./run.sh
  ```

  This will analyze the extracted representations and generate the data (saving it by default) required for Figure 2 (there can be small fluctuations due to the number of epochs of training that you use and/or the random splitting between train and test).

  You can visualize your results opening the jupyter notebook plot.ipynb, decommenting the line that specifies to use your results.

  This pattern of usage will be maintained for all the experiments below.

Fig 3 (generality of the hunchback shape)

In scripts/pretrained open the notebook hunchback.ipynb.

If you want to collect new results:

```
$ ./hunchback.sh
```

Fig 4

In scripts/pretrained open the notebook lasthidden.ipynb.

If you want to collect new results:

```
$ ./last_hidden.sh
```

Fig 5

In scripts/pretrained open the notebook lasthiddenpca.ipynb. This will reproduce panels A,B of Fig. 5.

If you want to collect new results:

```
$ ./last_hidden_pca.sh
```

The notebook hunchback trained vs. untrained will reproduce panel C of the same figure.

Fig 6

In order to recreate this figure just open the notebook plots.ipynb in scripts/mnist.

To recreate the results from scratch the following steps are required:

- train the small convnet on all the datasets: MNIST, MNIST* and MNIST†

  ```
  $ ./train_all.sh
  ```

  or, if you prefer, separately

  ```
  $ ./mnist.sh
  ```

  ```
  $ ./mnist_grad.sh
  ```

  ```
  $ ./mnist_shuffled.sh
  ```

- analyze results launching

  ```
  $ ./analyze_all.sh
  ```

  this will analyze the data for all the three cases MNIST, MNIST* and MNIST†.

  The parameter epochs should be the same used to train the different models. Anyway, for exploration purposes you can choose at which epoch to extract the data.

  ```
  $ python analyze.py --dataset yourdataset --epoch yourepoch
  ```

- (OPTIONAL) The shuffled dataset (MNIST†) and the one perturbed with the gradient of luminance (MNIST*) are provided. Anyway, you can create a new shuffled dataset

  ```
  $ python create_shuffled_mnist.py --save 1
  ```

  and/or new MNIST perturbed with the gradient of luminance. You can optionally specify the strength of the perturbation λ.

  ```
  $ python create_mnist_with_gradient.py --save 1 --lambdavar yourvalue
  ```