Neural Networks for Computer Vision

Part1. Using a pre-trained network

A. Classification [6 pts]

This part of the exercise examines how to load a pre-trained network and use it to classify new images.
part1a_pretrained_classification.ipynb: You should load the AlexNet network that is pre-trained on ImageNet. Then the notebook reads 14 images showing a panda and feeds them one by one as input to the network.

Q1 Load the AlexNet network from *torchvision* module [1 pts]
Hint: Do not forget to include the `pretrained` flag.
Q2 Transform input images with the transformations defined in the notebook and feed to the network one by one. [3 pts]
Hint: Do not forget to turn off the gradients to avoid unnecessary computations.
Q3 Print the predicted class and the corresponding probability for each image. [2 pts]
Hint: Use 'softmax' function to convert the output into a probability that sums up to 1.

B. Retrieval with Transfer Learning [9 pts]

This part of the exercise examines how to use a pre-trained network in order to extract descriptors for a different task (this process is commonly called transfer learning).

part1b_pretrained_descriptor.ipnyb: This notebook loads a subset of the CUB 200 2011 birds dataset for fine-grained recognition. It contains bird photos from 200 different species. 'EmbNet' takes as input a pre-trained network and creates a network that extracts a vector-descriptor per input image. Evaluation is performed for an image retrieval by measuring the percentage of queries that have at least one top-ranked positive image. [wiki]

Q1 Use different layers to extract descriptors and compare the performance. Report the results for each of the layers in the `classifier` module of AlexNet model. [4 pts]

Q2 Create class 'EmbNetGP' that uses only the convolutional layers and performs global pooling (max or average) on the tensor of activations of the last layer. What do you observe about its performance? Where is this attributed to? [5 pts]

Hint: Pytorch functional 'max_pool2d' or 'avg_pool2d'

Part2 – Training a network

A. Classification [10 pts]

This part of the exercise examines how to train a liteweight network for digit classification on the MNIST dataset.

part2a_train_classification.ipnyb: This notebook fully implements the model architecture and training of the network on the MNIST dataset. Your task is to implement the validation procedure. Then run the training and observe the training loss and validation accuracy.

Q1 Implement the network validation procedure in the `test` function. Report the accuracy on the validation step after each epoch. [5 pts]

Q2 Modify the network architecture to improve the performance. Experiment with *BatchNorm2d* and *Dropout* layers. [5 pts]

B. Retrieval with Metric Learning [15 pts]

This part of the exercise examines how to train a liteweight network in a metric learning fashion to extract a descriptor for digit retrieval/recognition.

part2b_train_descriptor.ipnyb: The notebook first defines a class to construct positive/negative pairs of MNIST images. A network that maps an image to a vector embedding is constructed by 'MnistNetEmb'. Function 'train2' performs the training in a siamese way with contrastive loss. Function 'test' evaluates on retrieval.

Contrastive loss for vectors x, y accompanied by pair label (1: positive, 0: negative):

$$0.5 * (label * d(x,y)^2 + (1 - label) * max\{margin - d(x,y)\,,\ 0\}^2)$$

d(x,y) is the Euclidean distance between vectors x and y

Triplet loss for vector a, p, n (anchor, positive, negative):

$$max\{d(a,\ p) - d(a,n) + margin,\ 0\}$$

More info about metric learning, contrastive and triplet losses can be found here.

Q1 The randomly initialized network is evaluated. The performance is much better than random retrieval. Why? [3 pts]

Q2 Implement training using triplet loss with *margin = 1.* [7 pts]

Hint: Modify 'MNISTpairs' to create 'MNISTtriplets' that constructs the required triplets. Modify 'train2' to create 'train3' to implement the training with triplet loss.

Q3 Use learning rate decay rate for training with triplet loss. [2 pts]

Hint: scheduler = lr_scheduler.StepLR(optimizer, 2, gamma=0.5)

Q4 Train a fully convolutional network with global pooling to construct image descriptors using triplet loss. Create descriptors of the same dimensionality as before for a direct comparison. [3 pts]