HW2 – submission 24.12.20 23:55

Guide lines

1. Include all your personal details including name, id, and e-mail address.
2. You should submit all function and script files written in MATLAB or Python. Your code should be well documented and clear. The code should run from **any** computer and include all path definitions (You should take care of this in the code).
3. Please divide the code by questions.
4. Final report – should include explanations on the implementation and the execution, answers to the questions, results, conclusions and visual results. Do elaborate on all parts of the algorithms/solution. **Please submit a PDF file and not a DOC file.**
5. Please post question regarding this HW on the facebook group: https://www.facebook.com/groups/294298727746314/
6. The grades are highly depended upon the analysis depth of the report.
7. HW can be submitted in pairs.
8. Eventually submit one compressed file including the code + images PDF.

Good luck!

Question 1 – Eigen-Faces and PCA for face recognition (30 points):

In this exercise, you will implement the Eigen-faces algorithm for face recognition. Use the provided stub code (`eigenface.m`) and implement the algorithm according to following steps:

1. Prepare the training set:
   a. The training set is loaded using the function `readYaleFaces.m` (line 3 on the given code). Read the documentation for more details.
   b. Compute the mean face of the trainset and subtract it from all the training images.
   c. Show an image representing the mean face.

2. Compute the eigen-faces:
   a. Implement a function that finds the r largest eigen-vectors of the trainset covariance matrix:
   $$\Sigma = E\left[\left(X-\mu\right)\left(X-\mu\right)^T\right]$$
   To avoid memory issues, use SVD as describes in the tutorial, or read the relevant section in [1]. Implement your algorithm accordingly and explain in detail.
   b. Show the **first** five eigen-faces as images (corresponding to the **largest** eigenvalues).
3. Reconstruction:
   a. For each image in the train set $\{x_j\}$, subtract the average face (calculated in 1.b) and calculate its low dimension representation vector $y_j = W^T\left(x_j - \bar{x}\right)$ according to the $r = 25$ largest eigen-faces (W is a matrix build from the top r eigen-faces).
   b. Reconstruct each of the images in the train set $\{x_j\}$ according to its representation y_j. Compute the following representation error:
   RMSE error – The average error per pixel in absolute value and in percent (out of 256 gray levels). Please scale the reconstructed image to have values in the range of [0,255].
   c. Describe and explain your results. What is the average representation error? Could we have foreseen it (hint: eigenvalues)?
4. Recognition:
   a. For each image in the **Test set** $\{x_i\}$, find the representation vector y_i according to the $r = 25$ top eigen-faces (don't forget to subtract the train set average face). What is the mean representation error? Was there any change in comparison to the train set? Explain. Demonstrate a good reconstruction and a bad one.
   b. Classify each image from the test set (use only the images of people who appear also on the train set – use the $\mathrm{face\_id}$ parameter)**.** This can be done by using nearest neighbor classifier between each y_i to the train set represent vectors $\{y_j\}$. For nearest neighbor classification, use the matlab function `fitcknn.m`. What are the success ratios (for face_id)?
   c. Explain why each of the unsuccessful classifications failed
   d. Propose an algorithm that will improve the results obtained here. (tip: use the eigenfaces as an input to the new algorithm)

References:

[1] Turk, Matthew, and Alex Pentland. "Eigenfaces for recognition." *Journal of cognitive neuroscience* 3.1

[2] A Tutorial on Principal Component Analysis - Jonathon Shlens

Q2 - Introduction to CNN (70 points)

To implement this assignment you can use choose any deep learning framework of the following: MATLAB neural network toolbox/Pytorch/Tensorflow/Keras. Remember – you can simply use the google colab service which has everything already set up in case you choose to use python.

Instructions for google colab:

1. Connect to Google Colab: https://colab.research.google.com/
2. Choose "new python 3 notebook"
3. Edit -> Notebook setting -> Hardware accelerator -> GPU
4. Save
5. When finished save the file as a .ipynb: File->Download .ipynb and submit it with your report.

1. Download the CIFAR10 dataset. If you choose to use Keras, simply import the dataset:
   from keras.datasets import cifar10
   (X_train, y_train), (X_test, y_test) = cifar10.load_data()
   Otherwise, download it from here https://www.cs.toronto.edu/~kriz/cifar.html
2. Normalize the data such that it will reside in the range [-0.5,0.5]
3. Train a linear classifier using a softmax activation on the training set without any regularization.
   Plot the train and test accuracy vs. num of epochs. Make sure your network converged properly.
   Plot the rows of the linear classifier. Note that to plot each row you first need to normalize each row by subtracting its min and stretching its range to [0,1].
4. Train another linear classifier, this time use L2 regularization with a multiplier of 0.1. (https://keras.io/regularizers/)
   Plot the train and test accuracy vs. num of epochs. Make sure your network converged properly.
   Plot the rows of the linear classifier.
   Explain the differences in the results from this section and the previous one.
5. Train a convolutional neural network. Have at least 4 layers in it. You can choose the structure of the network. For example, you can use the LeNet-5 architecture:
   conv (6) – maxpool – conv (16) – maxpool – dense(120) – dense(84) – dense(10)
   In the report describe the architecture you chose to use (number of layers, number of filters, activation functions, regularization, etc.).
   Report the performance of your network on the train and test datasets.
   Plot the train and test accuracy vs. num of epochs. Make sure your network converged properly.
6. Add augmentations to your data, i.e. during training, apply the following procedures on the images (randomly): horizontal flip, random crop, shear, zoom, and others. You can definitely use library-ready implementations for these procedures. Feel free to experiment with the chosen augmentations and their parameters. Report which augmentations you chose and how it affected your results. Add a figure showing performance of your network on the train and test datasets.
7. Find 2 images on the web and use them as input to your trained model. Show the output of the network (the probabilities) for your input.

Remember: you will first need to downsample your image to 32x32. For this end, use a proper downsampling filter to avoid aliasing, or use a library function such as PIL:

```
from PIL import Image
image = Image.open('your path')
image.resize((32,32), resample= PIL.Image.LANCZOS)
```

Furthermore, to avoid distortion the image should be square (or crop it to be a square beforehand).