Exercise 8 - Cats, Cats, and EigenCats

Are you sad that you have watched all cat movies and seen all cat photos on the internet? Then be sad no more - in this exercise we will make a *Cat Synthesizer* where you can create all the cat photos you will ever need!

Secondly, a very unfortunate event happened so you are now in a situation, where you need to find a *perfect new twin cat*.

To be able to do these wonderful things we will harness the power of image based *principal component analysis*. The methods we will use, can be called *classical machine learning*.

Missing Cat

Learning Objectives

After completing this exercise, the student should be able to do the following:

1. Preprocess a batch of images so they can be used to machine learning. Preprocessing can include geometric transformations, intensity transformations and image cropping.
2. Use the python function `glob` to find all files with a given pattern in a folder.
3. Create an empty data matrix that can hold a given set of images and a given number of measurements per image.
4. Compute the number of features per image using the height, width and the number of channels in the image.
5. Use the function `flatten` to convert an image into a 1-D vector.
6. Create an image from a 1-D vector by using the `reshape` function.
7. Create an unsigned byte image from a float image using pixel value scaling and pixel type conversion.
8. Read a set of images and put their pixel values into a data matrix.
9. Compute an average image using the data matrix.
10. Visualize an average image
11. Preprocess one image so it can be used in machine learning.
12. Use sum-of-squared pixel differences (SSD) to compare one image with all images in a training set.
13. Identify and visualize the images in the training set with the smallest and largest SSD compared to a given image.
14. Do a principal component analysis (PCA) of the data matrix using the sci-kit learn PCA
15. Select the number of components that should be computed in the PCA.
16. Extract and visualize the amount of the total variation that each principal component explains.
17. Project the data matrix into PCA space.
18. Plot the first two dimensions of the PCA space. The PCA space is the positions of each sample when projected onto the principal components.

19. Identify and visualize the images that have extreme positions in PCA. For example the images that have the largest and smallest coordinates in PCA space.
20. Compute and visualize a synthetic image by adding linear combinations of principal components to the average image.
21. Select suitable weights based on the PCA space for synthesizing images.
22. Compute and visualize the major modes of variation in the training set, by sampling along the principal components.
23. Generate random synthetic images that lies within the PCA space of the training set.
24. Project a given image into PCA space
25. Generate a synthetich version of an image by using the image position in PCA space.
26. Compute the Euclidean distance in PCA space between a given image and all other images.
27. Identify and visualize the images in the training set with the smallest and largest Euclidean distance in PCA space to a given image.
28. Use `argpartition` to find the N closest images in PCA space to a given image.

Importing required Python packages

We will use the virtual environment from the previous exercise (`course02502`).

Let us start with some imports:

```
from skimage import io
from skimage.util import img_as_ubyte
import matplotlib.pyplot as plt
import numpy as np
import glob
from sklearn.decomposition import PCA
from skimage.transform import SimilarityTransform
from skimage.transform import warp
import os
import pathlib
```

Exercise data and material

The data and material needed for this exercise can be found here: exercise data and material

The main part of the data is a large database of photos of cats, where there are also a set of landmarks per photo. You should download the data here.

IMPORTANT: You can start by working with the smaller data set with 100 cats found here: smaller photo database of 100 cats. Later you can use the full data set and see if your computer has enough RAM to handle it.

Start by unpacking all the training photos in folder you choose. For example `training_data`.

Preprocessing data for machine learning

The photos contains cats in many situations and backgrounds. To make it easier to do machine learning, we will *preprocess* the data, so the photos only contains the face of the cat. Preprocessing is and important step in most machine learning approaches.

The preprocessing steps are:

- Define a model cat (`ModelCat.jpg`) with associated landmarks (`ModelCat.jpg.cat`)
- For each cat in the training data:
  - Use landmark based registration with a *similarity transform* to register the photo to the model cat
  - Crop the registered photo
  - Save the result in a fold called **preprocessed**

Exercise 1: *Preprocess all image in the training set. To do the preprocessing, you can use the code snippets supplied* here. There is also a **Model Cat** supplied.

The result of the preprocessing is a directory containing smaller photos containing cat faces. All the preprocessed photos also have the same size.

Gathering data into a data matrix

To start, we want to collect all the image data into a data matrix. The matrix should have dimensions `[n_samples, n_features]` where **n_samples** is the number of photos in our training set and **n_features** is the number of values per image. Since we are working with RGB images, the number of features are given by `n_features = height * width * channels`, where `channels = 3`.

The data matrix can be constructed by:

- Find the number of image files in the **preprocessed** folder using `glob`. Look at the `preprocess_all_cats` function to get an idea of how to use `glob`.
- Read the first photo and use that to find the height and width of the photos
- Set **n_samples** and **n_features**
- Make an empty matrix `data_matrix = np.zeros((n_samples, n_features))`
- Read the image files one by one and use `flatten()` to make each image into a 1-D vector (flat_img).
- Put the image vector (flat_img) into the data matrix by for example `data_matrix[idx, :] = flat_img` , where idx is the index of the current image.

Exercise 2: *Compute the data matrix.*

Compute and visualize a mean cat

In the data matrix, one row is one cat. You can therefore compute an average cat, **The Mean Cat** by computing one row, where each element is the average of the column.

Exercise 3: *Compute the average cat.*

You can use the supplied function `create_u_byte_image_from_vector` to create an image from a 1-D image vector.

Exercise 4: *Visualize the Mean Cat*

Find a missing cat or a cat that looks like it (using image comparison)

You have promised to take care of your neighbours cat while they are on vacation. But…Oh! no! You were in such a hurry to get to DTU that you forgot to close a window. Now the cat is gone!!! What to do?

Exercise 5: *Decide that you quickly buy a new cat that looks very much like the missing cat - so nobody notices*

Luckily, the training set is actually photos of cats that are in a *get a new cat cheap* nearby store.

To find a cat that looks like the missing cat, you start by comparing the missing cat pixels to the pixels of the cats in the training set. The comparison between the missing cat data and the training data can be done using the sum-of-squared differences (SSD).

Exercise 6: *Use the `preprocess_one_cat` function to preprocess the photo of the poor missing cat*

Exercise 7: *Flatten the pixel values of the missing cat so it becomes a vector of values.*

Exercise 8: *Subtract you missing cat data from all the rows in the data_matrix and for each row compute the sum of squared differences. This can for example be done by:*

```
sub_data = data_matrix - im_miss_flat
sub_distances = np.linalg.norm(sub_data, axis=1)
```

Exercise 9: *Find the cat that looks most like your missing cat by finding the cat, where the SSD is smallest. You can for example use `np.argmin`.*

Exercise 10: *Extract the found cat from the data_matrix and use `create_u_byte_image_from_vector` to create an image that can be visu-*

alized. Did you find a good replacement cat? Do you think your neighbour will notice? Even with their glasses on?

Exercise 11: *You can use* `np.argmax` *to find the cat that looks the least like the missing cat.*

You can also use your own photo of a cat (perhaps even your own cat). To do that you should:

- Place a jpg version of your cat photo in the folder where you had your missing cat photo. Call it for example **mycat.jpg**
- Create a landmark file called something like **mycat.jpg.cat**. It is a text file.
- In the landmark file you should create three landmarks: `3 189 98 235 101 213 142` . Here the first `3` just say there are three landmarks. The following 6 numbers are the (x, y) positions of the right eye, the left eye and the nose. You should manually update these numbers.
- Use the `preprocess_one_cat` function to preprocess the photo
- Now you can do the above routine to match your own cat.

Optional Exercise: *Use a photo of your own cat to find its twins*

Principal component analysis on the cats

We now move to more classical machine learning on cats. Namely Principal component analysis (PCA) analysis of the cats image.

To compute the PCA, we use the sci-kit learn PCA. Note that this version of PCA automatically *centers* data. It means that it will subtract the average cat from all cats for you.

Exercise 12: *Start by computing the first 50 principal components:*

```
print("Computing PCA")
cats_pca = PCA(n_components=50)
cats_pca.fit(data_matrix)
```

This might take some time. If your compute can not handle so many images, you can manually move or delete som photos out of the **preprocessed** folder before computing the data matrix.

The amount of the total variation that each component explains can be found in `cats_pca.explained_variance_ratio_`.

Exercise 13: *Plot the amount of the total variation explained by each component as function of the component number.*

Exercise 14: *How much of the total variation is explained by the first component?*

We can now project all out cat images into the PCA space (that is 50 dimensional):

Exercise 15: *Project the cat images into PCA space*:

```
components = cats_pca.transform(data_matrix)
```

Now each cat has a position in PCA space. For each cat this position is 50-dimensional vector. Each value in this vector describes *how much of that component* is present in that cat photo.

We can plot the first two dimensions of the PCA space, to see where the cats are placed. The first PCA coordinate for all the cats can be found using `pc_1 = components[:, 0]` .

Exercise 16: *Plot the PCA space by plotting all the cats first and second PCA coordinates in a (x, y) plot*

Cats in space

We would like to explore what the PCA learnt about our cats in the data set.

Extreme cats

We start by finding out which cats that have the most *extreme coordinates* in PCA space.

Exercise 17: *Use `np.argmin` and `np.argmax` to find the ids of the cats that have extreme values in the first and second PCA coordinates. Extract the cats data from the data matrix and use `create_u_byte_image_from_vector` to visualize these cats. Also plot the PCA space where you plot the extreme cats with another marker and color.*

Exercise 18: *How do these extreme cat photo look like? Are some actually of such bad quality that they should be removed from the training set? If you remove images from the training set, then you should run the PCA again. Do this until you are satisfied with the quality of the training data.*

The first synthesized cat

We can use the PCA to make a so-called **generative model** that can create synthetic samples from the learnt data. It is done by adding a linear combination of principal components to the average cat image:

$$I_{\text{synth}} = I_{\text{average}} + w_1 * P_1 + w_2 * P_2 + ... + w_k * P_k \ ,$$

where we P_1 is the first principal component, P_2 the second and so on. Here we use k principal components.

The principal components are stored in `cats_pca.components_`. So the first principal component is `cats_pca.components_[0, :]` .

Exercise 19: *Create your first fake cat using the average image and the first principal component. You should choose experiment with different weight values (w)* :

```
synth_cat = average_cat + w * cats_pca.components_[0, :]
```

Exercise 20: *Use* `create_u_byte_image_from_vector` *visualize your fake cat.*

You can use the PCA plot we did before to select some suitable values for w.

Exercise 21: *Synthesize some cats, where you use both the first and second principal components and select their individual weights based on the PCA plot.*

The major cat variation in the data set

A very useful method to get an overview of the **major modes of variation** in a dataset is to synthesize the samples that are lying on the outer edges of the PCA space.

If we for example move a distance out of the first principal axis we can synthesize the cat image there. In this case we will try to move to $\pm\sqrt{(}$explained variance), where *explained variance* is the variance explained by that principal component. In code, this will look like:

```
synth_cat_plus = average_cat + 3 * np.sqrt(cats_pca.explained_variance_[m]) * cats_pca.compo
synth_cat_minus = average_cat - 3 * np.sqrt(cats_pca.explained_variance_[m]) * cats_pca.comp
```

here **m** is the principal component that we are investigating.

Exercise 22: *Synthesize and visualize cats that demonstrate the first three major modes of variation. Try show the average cat in the middle of a plot, with the negative sample to the left and the positive to the right. Can you recognise some visual patterns in these modes of variation?*

The Cat Synthesizer (EigenCats)

We are now ready to make true cat synthesizer, where cat images are synthesized based on random locations in PCA space. You can start by setting your `synth_cat = average_cat`. Then you can add all the components you want by for example (this number should be less or equal to the number of components we asked the PCA to compute):

```
n_components_to_use = 10
synth_cat = average_cat
for idx in range(n_components_to_use):
    w = random.uniform(-1, 1) * 3 * np.sqrt(cats_pca.explained_variance_[idx])
    synth_cat = synth_cat + w * cats_pca.components_[idx, :]
```

Exercise 23: *Generate as many cat photos as your heart desires..*

Cat identification in PCA space

Now back to your missing cat. We could find similar cats by computing the difference between the missing cat and all the photos in the databased. Imagine that you only needed to store the 50 weights per cats in your database to do the same type of identification?

Exercise 24: *Start by finding the PCA space coordinates of your missing cat:*

```
im_miss = io.imread("data/cats/MissingCatProcessed.jpg")
im_miss_flat = im_miss.flatten()
im_miss_flat = im_miss_flat.reshape(1, -1)
pca_coords = cats_pca.transform(im_miss_flat)
pca_coords = pca_coords.flatten()
```

The `flatten` calls are needed to bring the arrays into the right shapes.

Exercise 25: *Plot all the cats in PCA space using the first two dimensions. Plot your missing cat in the same plot, with another color and marker. Is it placed somewhere sensible and does it have close neighbours?*

We can generate the synthetic cat that is the closest to your missing cat, by using the missing cats position in PCA space:

```
n_components_to_use = ?
synth_cat = average_cat
for idx in range(n_components_to_use):
    synth_cat = synth_cat + pca_coords[idx] * cats_pca.components_[idx, :]
```

Exercise 26: *Generate synthetic versions of your cat, where you change the n_components_to_use from 1 to for example 50.*

We can compute Euclidean distances in PCA space between your cat and all the other cats by:

```
comp_sub = components - pca_coords
pca_distances = np.linalg.norm(comp_sub, axis=1)
```

Exercise 27: *Find the id of the cat that has the smallest and largest distance in PCA space to your missing cat. Visualize these cats. Are they as you expected? Do you think your neighours will notice a difference?*

You can also find the n closest cats by using the `np.argpartition` function.

Exercise 28: *Find the ids of and visualize the 5 closest cats in PCA space. Do they look like your cat?*

What we have been doing here is what has been used for face identification and face recognition (*Matthew Turk and Alex Pentland: Eigenfaces for Recognition, 1991*)

In summary, we can now synthesize all the cat photos you will only need and we can help people that are loooking for cats that looks like another cat. On

top of that, we can now use methods that are considered state-of-the-art before the step into deep learning.

References

- Cat data set
- sci-kit learn PCA