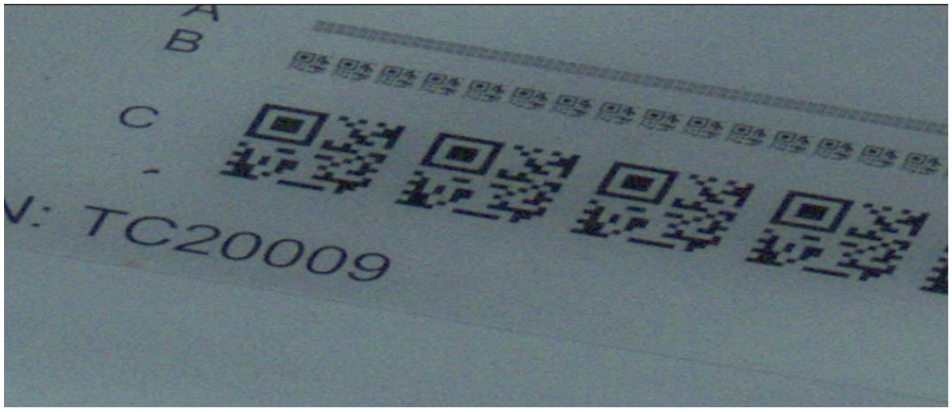
CLASSIC COMPUTER VISION

The image below is taken from a Microscope observing a special calibration specimen. The exercise consists in the identification of the given patterns (the big ones) Implement the detection of the big patterns and the pixels in Python or C++. Do not decode the content stored in these patterns.



QR finder detection

https://www.swisseduc.ch/informatik/theoretische_informatik/qr_codes/docs/qr_standard.pdf

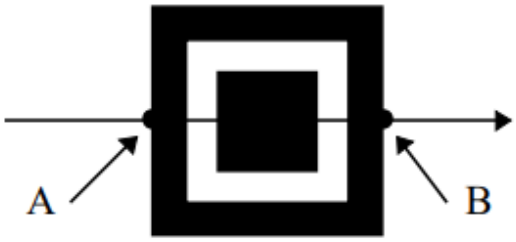
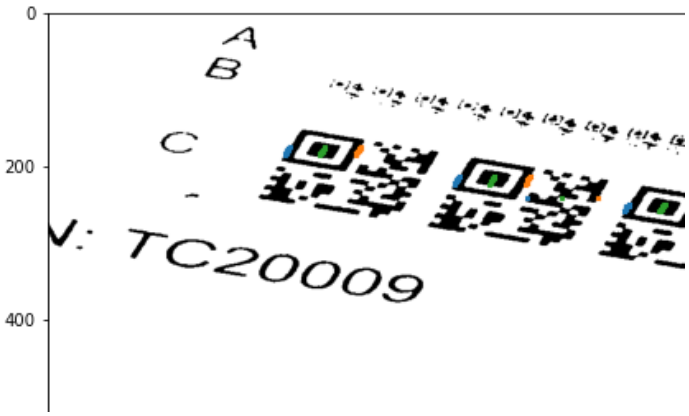


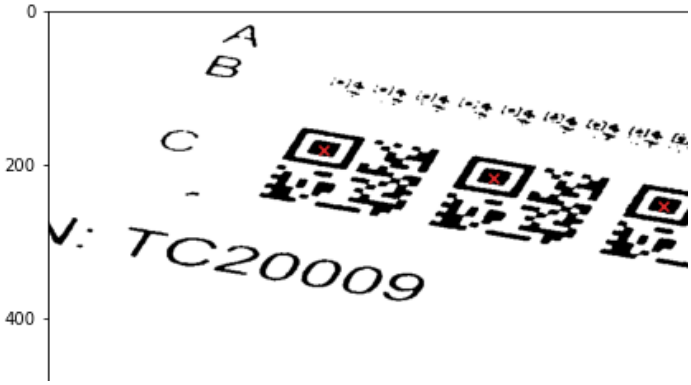
Figure 1 QR finder pattern and A, B points to detect

1. Determine a Global Threshold by taking a reflectance value midway between the maximum reflectance and minimum reflectance in the image. Convert the image to a set of dark and light pixels using the Global Threshold.
2. Locate the Finder Pattern. The finder pattern in QR Code consists of three identical Position Detection Patterns located at three of the four corners of the symbol. As described in 7.3.2, module width in each Position Detection Pattern is constructed of a dark-light-dark-light-dark sequence the relative widths of each element of which are in the ratios 1:1:3:1:1. For the purposes of this algorithm the tolerance for each of these widths is 0,5 (i.e. a range of 0,5 to 1,5 for the single module box and 2,5 to 3,5 for the three module square box).

We will determine the threshold by using the Otsu algorithm which aims to minimize intra-class intensity variance of pixel intensities.



Once binarized implement step 2 for the rows. Note that the 1:1:3:1:1 ratio holds under affine transformation but for perspective transformation it holds only up to a tolerance due to parallax (the further the closer).

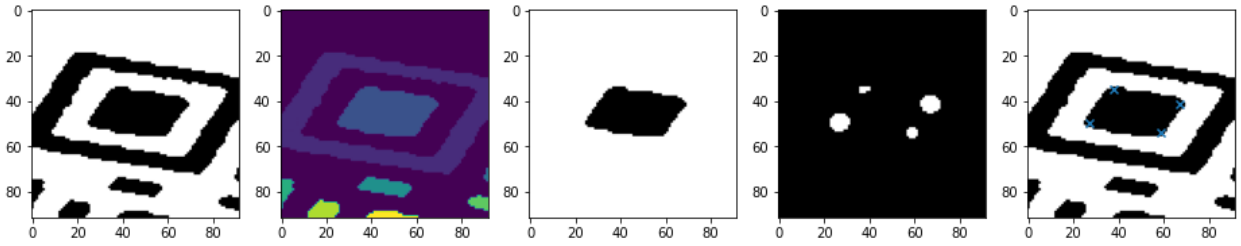


We will then proceed to cluster close detections to locate a single finder pattern using a density-based clustering algorithm (DBSCAN). The features used in this context are the center coordinates and the width. DBSCAN complexity is O(n^2) where n is the number of features points, these are aggregated based on a pairwise distance matrix. However, the number of candidates is around a 100 making it the best choice if we want most of all accurate results. Furthermore, we can get rid of noisy detections by setting a minimum number of samples per cluster.

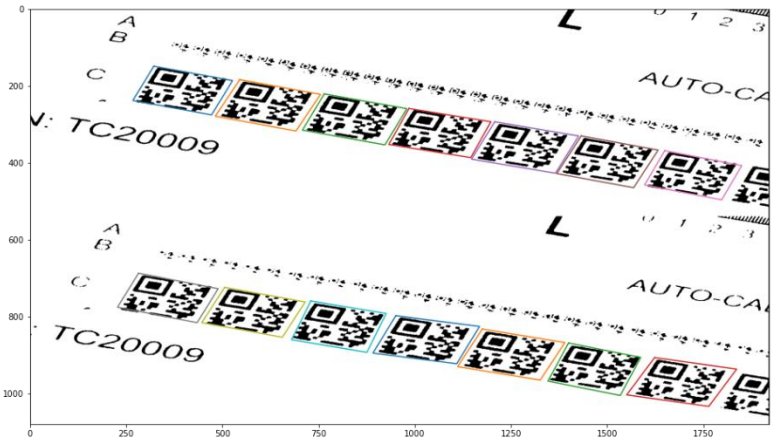
Extrapolation

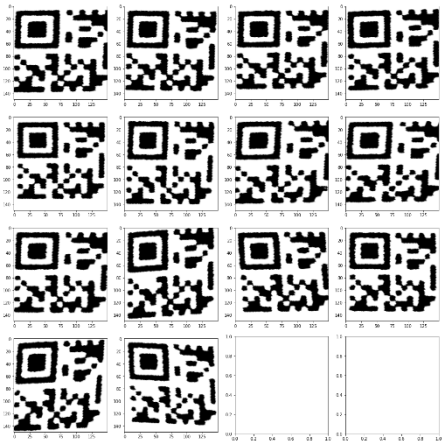
Now that we have the centers of the finder patterns, we will detect the square corners. For each finder centered in (estimated row, estimated column) we will process the sub image patch of size (estimated width x estimated width).

Each binary patch is processed by connected component labeling to isolate the central square and filtered to highlight corners with Harris features which are subsequently clustered for non-maximum suppression using DBSCAN (cheaper than a non-maxima suppression with sliding window).



Once the corners are detected one can compute the affine transform using 3 of the 4 points and extrapolate the limits of the pattern by knowing the original size (in this case we estimated it as we did not have a sample). We prefer to estimate the affine transformation as the perspective transformation has higher relative computational error (worst results).





STEP	ALGORITHM	COMPLEXITY
QR FINDERS	OTSU	$\Theta(\max(\sqrt{\#pixels}, \#histogrambis^2))$
	ROW RATIO	$\Theta(\#pixels)$
	DBSCAN	$\Theta(\#detected\ ratios^2)$
EXTRAPOLATION	LABELING	$\Theta(\max widht^2 \times \#finders)$
	HARRIS	$\Theta(\max widht^2 \times \#finders)$ *
	DBSCAN	$\Theta((\max(\#(Harris > thrs)\ **)^2) \times \#finders)$
	LinSolve	Θ (1)

*HARRIS corner detection has 2 Sobel filters with kernel size 3 and a gaussian smoothing of the sum of squared difference (SSD) gradient matrix, these factors are constant.

** Harris> thrs means number of locations with Harris values above threshold

Questions

1) How this can be used?

It can be used to share internet links in advertisement os as augmented reality (AR) tags show multimedia content on smarphones.

2) What type of code is this one?

A quick response (QR) code: it is an extension of the bar code as it can hold more information within a 2d matrix structure, infact a QR code may encode not only 8 bit alphanumeric characters but also kanji characters. It also includes a finder pattern at each corner so that cameras can be used as scanners.

3) How you would implement the recognition and decoding of this code

The QR code represent a bright and dark pattern wich can be interpreted as 1 and 0. Once detected these can be rectified to match a regular grid and subsequently read. Once the information has been read it can be decoded into characters.

Observations

1) These codes lie all on the same surface, with this knowledge one could implement a more precise detector as all patterns are distorted by a single plane transformation.

2) Normally QR codes have more finders' patterns to determine boundaries, this one has 4 dots at the top of the leftmost block and 4 dots to the left of the bottom block, maybe these could have been used to detect the boundaries more efficiently but idk if these are part of the message.

CODE: [CVAssignement/Ex1.ipynb at main · marioviti/CVAssignement (github.com)]

DEEP LEARNING EXERCISES

Given a CNN classifier we can consider the following building blocks, where we use H x W x C structure:

1) CONV2D-K-N is a convolutional of N filters with KxK, padding 0 and stride 1
2) POOL-K is a KxK pooling with stride K and padding 0
3) FC-N is a fully connected*

* FC does not specify the number of input and output features (the matrix height and width). Supposing that we flatten the feature map and that we only classify binary images the output and input channels would be 1 and 1024 respectively.

LAYER	ACTIVATION MAP	WEIGHT	BIASES
INPUT	64x64x3	0	0
CONV-9-32	56x56x32	3x32x9x9	0/32**
POOL-2	28x28x32	0	0
CONV-5-64	24x24x64	32x64x5x5	0/64**
POOL-2	12x12x64	0	0
CONV-5-64	8x8x64	32x64x5x5	0/64**
POOL-2	4x4x64	0	0
FC	1	1024x1	1***

** by default convolution does not require a bias but most popular frameworks implements CNN with a bias term as default.

*** The reverse is true for FCN layers, one may omit the bias (especially if batch norm follows) however the output bias is necessary to favor the most occurring class to be predicted more often (as this last term does not rely on the input at the time of inference it is "biased" towards the majority class).

UNET

Super Resolution

An image can be represented at a certain scale. The multiple representation of the image at different scales is called the scale space $I_\sigma = I * G_\sigma$by a set of gaussian kernels of varying support.

The idea of super resolution is to predict $I_{\sigma+1} = F(I_\sigma)$ therefore obtaining a recursive relationship, therefore having the possibility to have "infinite" resolution (at least in theory). But it is hard to generalize, to train one may needs lots of data and usually fits only one particular type of images.

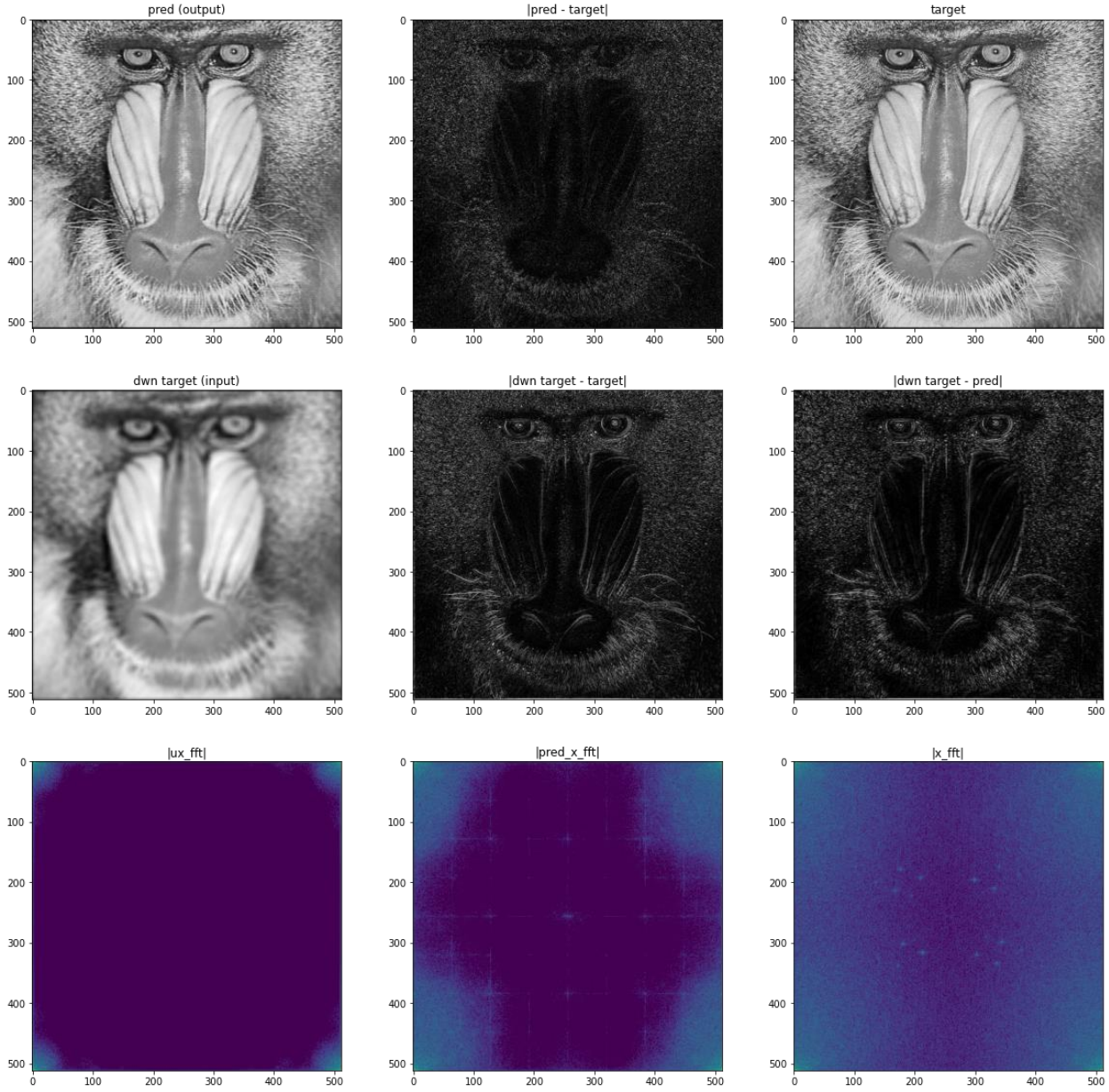


Figure 2: Here we overfit the baboon image which is downscaled of a factor 4 to reobtain an image quite similar to the original.

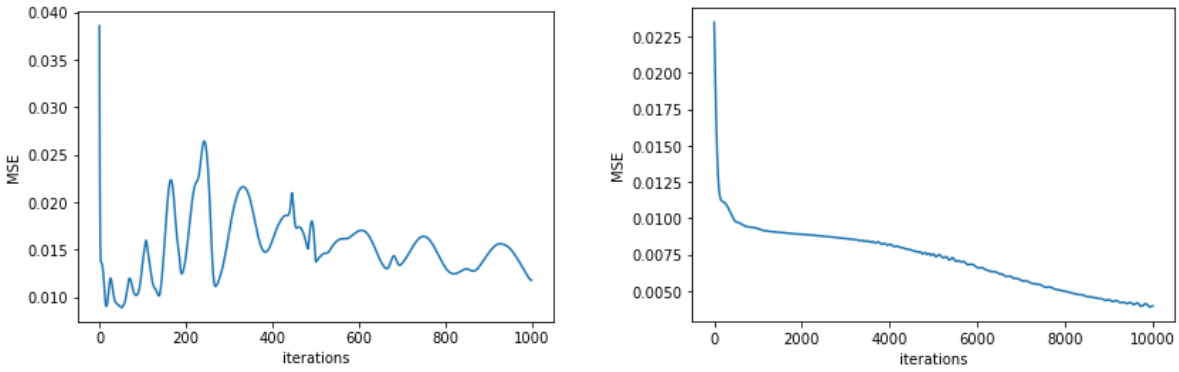
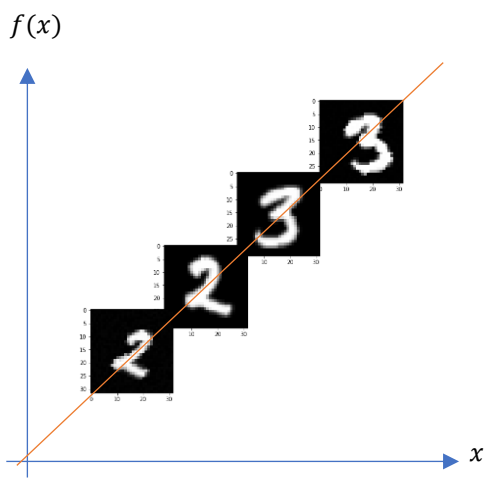


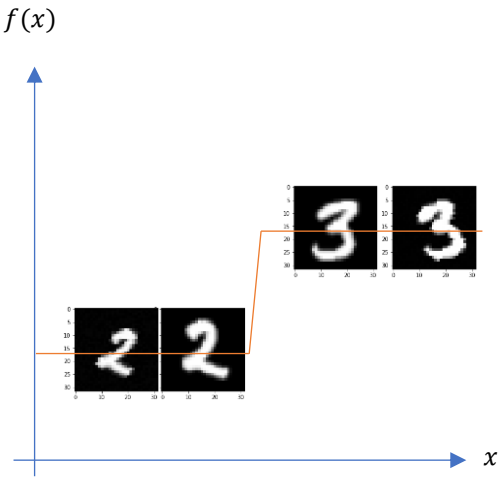
Figure 3 For this particular task the choice of the learning rate is the most important parameter. A high learning rate will eventually lead to overshooting. (left 1e-4 right, 1e-6 right)

Denoising Autoencoder

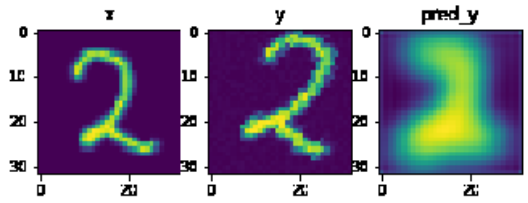
An auto encode implements the identity function: $f(x) = x$ with a constrain on $f = d \circ e, e(x) = x_e \in R^m, x \in R^n m \ll n$ to obtain a sparse representation of x. A denoising autoencoder however implements $f(x) = x_t, f\big(T(x)\big) = x_t$ where t subscript stands for template and T is an arbitrary transformation in the image domain (translation, rotation, sheer, noise etc etc). If we hypothesize that $e\big(T(x)\big) = x_e + eps$ an additional constraint is $e'(x) = 0$. f is a surjection so it can be used to learn a template representation of an image so that similar images are mapped to the same template.



Autoencoder



Denoising Autoencoder



If we do so we can learn structure from data in a unsupervised way and come up with clusters that correlates with classes. For example (left) the visual representation is not exact but it matches more the “ideal” template on the left than the central input image.

TrainPseudocode (f, train_dataset):

optimizer = Adam(f.parameters, lr=0.001) // choose Adam as default

For x in train_dataset:

x' = f(T(x)) // reconstruct original x from noisy T(x)

loss = Xent(x, x') // compute the loss as the binary crossentropy

loss.backprop() // backpropagate the loss to obtain the gradients (will be stored // int the attribute grad of f.parameters

Optimizer.step() // update f parameters according to the stored gradient and

// optimization policy

return f

TestPseudocode (f, test_dataset):

X_e = [x_e = e(x) for x in test_dataset } // get the encoding of the test image

Tsne_X = TSNE(components=2)(X_e)

The figure below show how the method learnt to aggregate together samples from the same class without any use of labels.

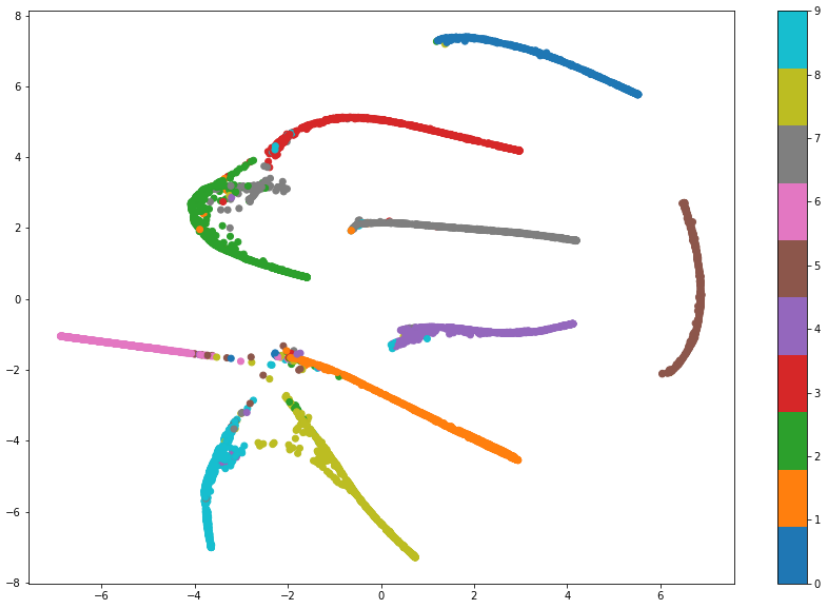


Figure 4 2D TSNE of the MNIST Test Set encodings (Tsne_X)

CODE: [CVAssignement/Ex2.ipynb at main · marioviti/CVAssignement (github.com)]