Department of Engineering

Artificial Intelligence Systems

Final Report

Image Recapturing

Supervisors	Students
DE NATALE Francesco	GUGLIELMI Matteo
VERDE Sebastiano	MAZZARO Roberto
	RECH Antonella

Academic Year : 2021/2022

Contents

Abstract Nowadays, with the availability of sofisticated digital cameras, images can be acquire with high-quality also from LCD monitor screens quite easily. In light of this, an hypothetical attacker may want to reacpture a forged image in order to hide imperfections and to increase its authenticity.
The underlying idea of this work is to develop an algorithm, following a piece of the paper cited in the bibliography [2], which is able to recognize whether a specific image has been recaptured or not.
As quoted in the reference article, the authors address that '[...] aliasing and blurriness are the least scene dependent features'; since aliasing can be ereased by setting the parameters of the recapturing system to predetermined values, in this work we focus on the bluriness feature in order to perform the final recognition. More precisely, a set of learned edge bluriness are used to perform a classification of each input image.
Every time a scene is captured with a digital camera, a certain amount of blur is introduced into the picture by the image acquisition system. This blur can be characterized by the so called Point Spread Function of the capture device. Unfortunately, the PSF of a system is not easily achievable and the Line Spread Function (LSF) is used instead. 'By definition, a line spread function is a $1-$D function corresponding to the first derivative of the edge spread function (ESF)' [3]
Another important component are the two over-complete dictionaries built by means of the K-SVD algorithm, one for both recaptured and single captured images in the available dataset [1]. Each dictionary is trained to provide an optimal sparse representation of the Line Spread Profile function extracted from the example images.
The purpose of the two dictionaries is to describe how well wach dictionary fits the line spread profile matrix (Q_i) of a query image. In poor words, an approximation error is computed as a measure of likeliness between an unknown query image and D_{SC} and D_{RC}.
These dictionary, which will be referred to as D_{SC} and D_{RC} for the dictionary trained with single captured and recaptured images respectively, have a key role the approximation error which is one of the two features used.
The other feature used, λ_{avg}, refers to the average of Line Spread Profile widths computed as follows : the value of λ is defined as the minimum distance of the ESF such that at least 95% of the spectral energy of the spread function falls within it.
Eventually, after having computed λ_{avg} and E_d for every image of the training set, a support vector machine classifier (SVC) is trained.
Whenever an unknown query image has to be classified, the aforesaid features has to be computed and then feed the already built SVM with the respective values.

1 Implementation

The developped algorithm is written in Python coding language.
It was deemed appropriate to subdivide the whole script in five main modules in order to have a clean and modular package. This also allows for saving files with intermediate results to save computational power since some processes are very demanding.
Enumerating the modules :

- **dataset split** into training and test set;
- **calculus** of the Line Spread Profile matrix Q_i for every image;
- **building** of the over-complete **dictionaries** D_{SC} and D_{RC};
- SVM **training**
- address the performane of the **SVC testing** it on the test set.

How to use Sequence of action in order to make this code work:

1. run 'phase1_QiCalculusdataset.py' (no need to do that because they are already computed in the given complete_dataset folder) : compute all the matrices for each camera and type of image;
2. run 'dataset_acquisition_txt.py' : perform the training and test sets formation;
3. run 'make_dictionary_from_txt.py' or 'make_dictionary_from_txt_.py' : the only difference here is that in the first module the building process start with complitely empty dictionaries; instead in the second one, dictionaries are initialized;
4. run 'SVM_training.py' : compute and save the hyperplane equation which better separates the plane linearly;
5. run 'SVM_classifier.py' : saves significant statistics about achieved performances ;

1.1 Dataset formation

This module automatically splits the whole dataset into training and test set as explained in the *Section VI-A* of the reference paper [2].
As specified in the latter :

- for single captured images, 15 randomly chosen images are selected for training purposes;
- concerning recaptured images, for every pair of cameras involved, where the first one is the device used to capture the original image and the second one the system used to recapture it, we randomly select 3 pictures for training purposes.

The remaining ones, both for recaptured and single captured images, are kept for performance testing.
For each camera, both in single captured and recaptured images, the outputs are two .txt files containing the paths of the Line Spread Profile matrices for training and test images respectively.

1.2 Q_i Calculus

In this section we implement the method described in the paper [2] for the calculus of the matrix Q.
The proposed process determines, for any given single or recaptured image, a LSP matrix Q_i. Since this part is very computationally demanding, we stored the resulting matrices in the corresponding directory of each camera model to used them later on in the next steps.

The feature matrix extracted for each dataset's image contains the LSP of any columns presents in the selected blocks. We achieved these blocks by applying the following criterion, explained in details in the paper :

- firstly, the query image is converted to greyscale and all edges contained in the image are detected using a Canny Edge Detector (Canny Filter);
- the query image is, then, divided into non-overlapping blocks $B(m,n)$ of size $W \times W$ with $W = 16$ pixels;
- then, we check for horizontal, near horizontal, vertical and near vertical single edges and the blocks are chosen by counting the number of columns containing only one non-zero value. A block is considered only when the condition $\eta \geq \beta W$ is satisfied, where $\beta = 0.6$;
- the LS function q_i is then calculated by normalizing the gradient of each columns given the previously obtained blocks. Subsequently, all the q_is are interpolated by $4\times$ to increase the number of data points to 64;
- to determine the λ_{avg}, we compute the distance that embeds the 90% of the spectral energy of each interpolated column of the Q_i matrix. Consequently, we compute the mean of those distances;
- at this level, only the greylevel blocks, which meet some contraints, are selected and the corresponding interpolated q_i inserted in the Q_i matrix of the image. These contraints are :
  - the block-based variance $\sigma_{m,n}$ has to fall within the largest 20% of all the computed values;
  - the average lambda's value $\overline{\lambda}_{m,n}$ has to fall between the smallest 10% of all the computed values.

From now on, the Line Spread matrices for recaptured and single captured images are stored and they wll be used for training purposes.

1.3 Dictionary building

This script perform dictionary learning [4]. Given the training feature matrix S, which is obtained by the horizontal composition of all the previously computed Q_i, the goal of this technique is to obtain the best sparse dictionary, D $\in \mathbb{R}^{W\times K}$, that provides an optimal sparse representation for all the LSP matrices in S.
Once the matrix S has been built, a very large matrix is obtained. In order to reduce its size, a sort of resizing operation is applied : the training feature matrix is obtained by keeping only one column out of four in the original matrix.
The result is used to fed the *K-SVD algorithm* whose main parameters are briefly discussed in the following sections (1.3.1)
Since the whole process requires a consistent amount of time, once both D_{SC} and D_{RC} have been built, they're saved in a .txt file. Doing so, they just have to be loaded in the program when they're needed later on.

1.3.1 K-SVD Parameters

In our implementation, we used the K-SVD function provided by the sklearn library even though we hard coded an alternative version which turned out to be a little less efficient.
This function takes as inputs some specific parameters which define the characteristics the output has to meet. Unfortunately, the **reference paper doesn't clearly specify the parameters**, same for the SVM implementation, so we had to make several trials in order to find a set of values which lead to a decent overall result.
More precisely, the parameters to be specified are :

- *Number of components* : number of elements the output dictionary contains. Those ones may be zero or non-zero valued elements depending on the grade of sparsity defined (see below). The exact number is not defined in the paper so we sought a value which may represent a good trade-off between quality and computational demand.
  In the final implementation we set this parameter to 50 but increasing the number of components may likely lead to better results;
- *Alpha* : consists in the sparsity controlling parameter. Reading the paper, at first it seems it has to be set to 0 but, with this particular value, the obtained results are very low in quality. Guessing an alpha parameter equal to 1 we observed a consistent increse in quality performance;
- *Maximum number of iterations* : integer number which indicates an upper bound on the number of iteration to perform. In the reference paper [2] it is clearly specified to use 160;
- *Tollerance* : tollerance for numerical error. As defualt is to 10^{-8}. In this case we used 10^{-6} as shown in a similar purpose algorithm developed in MathLab;

- *Transformation algorithm* : algorithm to process the data. We used the OMP algorithm, as specified in the reference paper, to estimate the sparse solution:
- *Number of non-zero coefficients in the transformation* : number of non-zero coefficients to the target for each column of the solution. This corresponds to the degree of sparsity of the output, in this specific case of the two dictionaries. This parameter in the reference paper is named as L an indicates the optimal number of atoms in the dictionaries. It is explained that this parameters is set to 3 because it coincide with the peak of the second derivative for our training sets.

1.3.2 Dictionary initialization

As explained in section $VI - A$ of the reference paper 'The initial set of atoms was constructed from the Line Spread functions of the nine single capture cameras and 63 different LS functions determined from randomly selected image recapture camera combinations'. We actually implement this part but we have not noticed any significant difference.

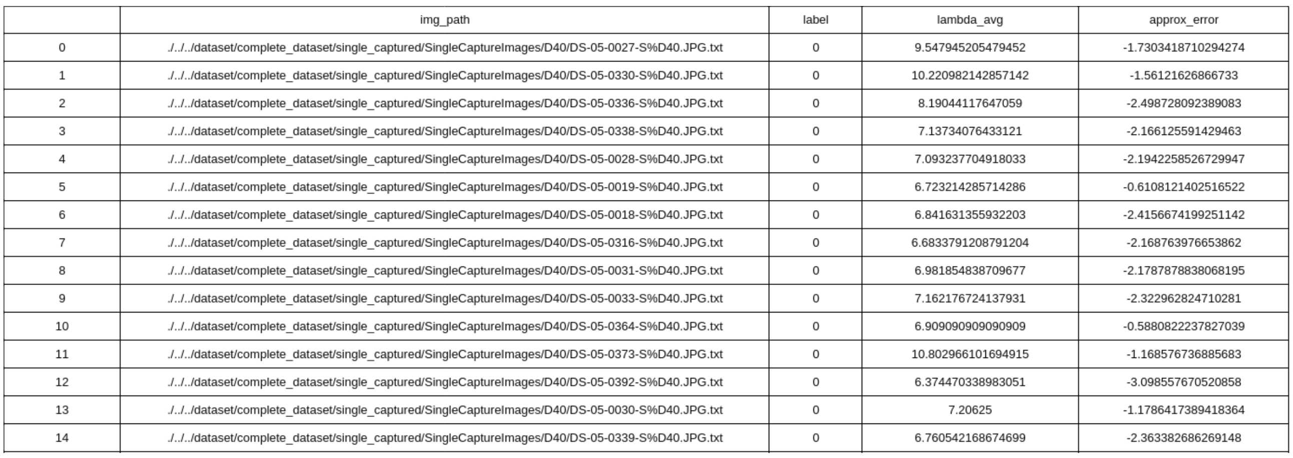
1.4 Support Vector Machine

1.4.1 Training phase

After having extracted the aforesaid features, as indicated in the followed paper [2], a support vector machine (SVM) has been used in order to classify unknown images.
The kernel used is of the linear kind and the training phase has been structured as follows :

- paths of images used for training purposes are stored in two separated lists, both for recaptured and single captured;
- pre-computed dictionaries are loaded;
- iteratively, for each element within the two lists :
  - the corresponding already calculated list of Q_i is loaded;
  - the features (approximation error E_d and average line spread function width λ_{avg}) are computed and saved within a common Pandas DataFrame [**Picture** 1.1] organized in columns as follows:
    - first column : it's composed by integer indexes;
    - second column : it's formed, cell by cell, by the path of a specific processed image;
    - third column : it contains the true label of a specific image;
    - fourth column : it displays the computed values of λ_{avg} for the corresponding image;
    - fifth and last column : element-wise, it showcases approximation errors.

  It's worth mentioning that the previously calculated Q_i and dictionaries are loaded in order to save some computational power;
- at the end, a unified csv table made of training samples including both recaptured and single captured values, is used to train the SVM. Once the fitting has finished, the resulting model is saved in preparation for future predicting operations.



	img_path	label	lambda_avg	approx_error
0	./../../dataset/complete_dataset/single_captured/SingleCaptureImages/D40/DS-05-0027-S%D40.JPG.txt	0	9.547945205479452	-1.7303418710294274
1	./../../dataset/complete_dataset/single_captured/SingleCaptureImages/D40/DS-05-0330-S%D40.JPG.txt	0	10.220982142857142	-1.56121626866733
2	./../../dataset/complete_dataset/single_captured/SingleCaptureImages/D40/DS-05-0336-S%D40.JPG.txt	0	8.19044117647059	-2.498728092389083
3	./../../dataset/complete_dataset/single_captured/SingleCaptureImages/D40/DS-05-0338-S%D40.JPG.txt	0	7.13734076433121	-2.166125591429463
4	./../../dataset/complete_dataset/single_captured/SingleCaptureImages/D40/DS-05-0028-S%D40.JPG.txt	0	7.093237704918033	-2.1942258526729947
5	./../../dataset/complete_dataset/single_captured/SingleCaptureImages/D40/DS-05-0019-S%D40.JPG.txt	0	6.723214285714286	-0.6108121402516522
6	./../../dataset/complete_dataset/single_captured/SingleCaptureImages/D40/DS-05-0018-S%D40.JPG.txt	0	6.841631355932203	-2.4156674199251142
7	./../../dataset/complete_dataset/single_captured/SingleCaptureImages/D40/DS-05-0316-S%D40.JPG.txt	0	6.6833791208791204	-2.168763976653862
8	./../../dataset/complete_dataset/single_captured/SingleCaptureImages/D40/DS-05-0031-S%D40.JPG.txt	0	6.981854838709677	-2.1787878838068195
9	./../../dataset/complete_dataset/single_captured/SingleCaptureImages/D40/DS-05-0033-S%D40.JPG.txt	0	7.162176724137931	-2.322962824710281
10	./../../dataset/complete_dataset/single_captured/SingleCaptureImages/D40/DS-05-0364-S%D40.JPG.txt	0	6.909090909090909	-0.5880822237827039
11	./../../dataset/complete_dataset/single_captured/SingleCaptureImages/D40/DS-05-0373-S%D40.JPG.txt	0	10.802966101694915	-1.168576736885683
12	./../../dataset/complete_dataset/single_captured/SingleCaptureImages/D40/DS-05-0392-S%D40.JPG.txt	0	6.374470338983051	-3.098557670520858
13	./../../dataset/complete_dataset/single_captured/SingleCaptureImages/D40/DS-05-0030-S%D40.JPG.txt	0	7.20625	-1.1786417389418364
14	./../../dataset/complete_dataset/single_captured/SingleCaptureImages/D40/DS-05-0339-S%D40.JPG.txt	0	6.760542168674699	-2.363382686269148

Figure 1.1: Pieces of Panda DataFrame table in .csv file format.

1.4.2 Classification

At this level of implementation, the SVM has already been trained and the model stored.
This part of implementation consists of testing the SVM on the test set in order to adress its performances.
By analogy with the training phase, this part's explanation is structured in points as well :

- the first step consists in extracting the path of all images used for test, both recaptured and single captured, shot by all the cameras;
- for each picture, the feature extraction process is performed and the ouput is saved within a .csv file equivalently to the training part;
- consequently, a predition is carried out on each image and the results are showed in a table which exhibits: the amount of correctly and erroneously classified rate and the accuracy achieved for each camera model. Furthermore, the confusion matrix for each camera is video-printed, where :
  - the first element represents the True Negatives (images correctly classified as Single Captured);
  - the first element represents the False Negatives (images wrongly classified as Single Captured);
  - the first element represents the True Positives (images correctly classified as Recaptured);
  - the first element represents the False Positives (images wrongly classified as Recaptured);

  It is noteworthy to adress that in order to obtain independent statistics, the classification process is performed separately on each camera model's test set.

2 Conclusions

General results overview The obtained results are not as good as the ones presented in the paper [5] but it was expected since usually the shown results are obtained with the omission of some particulars, sometimes important, e.g. missing parameters.
Concerning the recaptured images, the final results on the test set are worse than the expected values. However, the divergence between presented and achieved accuracies is not that large. Indeed, for certain camera models we achieve a classification accuracy of 95% and 97% in the best cases and of 80% and 78% in the worst cases. As average recognition accuracy, for the recaptured images we obtained an 88% of correct classification rate against an expected 99.03%.
Results of lower quality are obtained when trying to recognize single captured images; in this case the best accuracy achievable is around 80% with an average of 56% against the expected 94.89%.

Analysis on the camera models Along the paper has been explained that cameras provided with cheaper or older sensors perform poorely because their pictures are less sharp than most recent digital ones. In light of this, as expected, for low-cost cameras such as Kodak V550B, V550S and V610 we obtained lower accuracies, still considerably lower than the presented ones in the reference paper.
Eventually, we were surprised to see that TZ7 model performed badly in our algorithm while it's within the bests on according to the results shown in the paper.

Reasons for the differencies Some possible reasons for the observed differences in the final results, compared to the ones presented in the paper, may be the following ones :

- ambiguity of the paper :
  - sometimes, some steps are not clear enough in order to deeply understand the specific operation the authors make. In other words, they leave the door open to free interpratation of what they write leading to ambiguity and confusion when implementing a code;
  - the lack of parameters indication, i.e. for the dictionary learning algorithm most of the parameters are missing and we had to assume some random values. It would have been possible to execute a grid search but, since the whole program implies a large amount of time, this wasn't possible.
    Another unclear point is how they initialize the sparse dictionaries because, in our case, we did not observed significant improvements doing so.

Bibliography

[1] Recapture image database. http://www.commsp.ee.ic.ac.uk/ pld/research/Rewind/Recaptured/. Accessed Gen. 12, 2022.

[2] Thongkamwitoon, T., Member, S., IEEE, Muammar, H., Member, IEEE, , Member, P.-L. D. S., and IEEE. Compressed sensing. *IEEE TRANSACTIONS ON INFORMATION FORENSICS AND SECURITY 10*, 5 (MAY 2015), 1289–1306.

[3] Thongkamwitoon, T., Member, S., IEEE, Muammar, H., Member, IEEE, , Member, P.-L. D. S., and IEEE. *Compressed Sensing.* IEEE, 2015, ch. II.

[4] Thongkamwitoon, T., Member, S., IEEE, Muammar, H., Member, IEEE, , Member, P.-L. D. S., and IEEE. *Compressed Sensing.* IEEE, 2015, ch. IV-C.

[5] Thongkamwitoon, T., Member, S., IEEE, Muammar, H., Member, IEEE, , Member, P.-L. D. S., and IEEE. *Compressed Sensing.* IEEE, 2015, ch. VI-B.