Classification of identity documents using a deep convolutional neural network

Pere Vilàs[1]
January 20, 2018

[1]Universitat Oberta de Catalunya (UOC)

pvilas@uoc.edu

Abstract. *In this work, we review the main subjects that we have to serve to build an image classifier, then we design and implement a system to classify identity documents. Especially relevant is the description of the workflow that we have come up with after several ways of approaching the problem and which we hope can serve other machine learning practitioners. We evaluate some feature extractor algorithms to find the most suitable for identity documents classification purposes. Using virtual machines on the cloud, we run feature extractors in parallel to label at a speed of 16 images/s. Then, we select a neural network architecture and hyperparameters to train a convolutional neural network. Our results give an accuracy of 98%. We detail the responses of the convolutional filters over the images. Results and source code in the appendix.*

Keywords: *machine-learning, computer-vision, image-classification, identity-document, convolutional-network.*

Contents

List of Figures

List of Algorithms

List of Listings

1. Introduction

1.1. Work form

Títol del treball	Classification of identity documents using a deep convolutional neural network
Nom de l'autor	Pere Vilás Marí
Nom del consultor	Lourdes Meler Corretjé
Nom del PRA	David García Solórzano, Jose Antonio Morán Moreno
Data de lliurament	01/2018
Titulació programa	Grau de Tecnologíes de la Telecomunicació
Àrea del treball final	11.602 TFG- Aplicacions multim. basades en processament del senyal
Idioma del treball	anglès
Paraules clau	machine-learning, computer-vision, image-classification

Resum del treball En aquest treball, repassam els coneixements que necessitam per construir un classificador d'imatges. Després dissenyam i implementam un sistema per classificar documents d'identitat. És especialment rellevant la descripció del flux de treball al qual hem arribat després d'aproximar-mos al problema des de diferents vessants. Esperam que l'experiència pugui servir a altres practicants de machine learning. Avaluem alguns algorismes d'extracció de característiques per identificar el més adequat per classificar documents d'identitat. Usant màquines virtuals al núvol, hem executat extractors de característiques en paral·lel a una velocitat de 16 imatges per segon. Després, hem seleccionat una arquitectura i uns hiperparàmetres per entrenar una xarxa neuronal convolucional. Els nostres resultats ens donen una precisió del 98%. Detallam les respostes dels filtres de convolució sobre les imatges. Els resultats i el codi font estan disponibles als apèndix.

Abstract In this work, we review the main subjects that we have to serve to build an image classifier, then we design and implement a system to classify identity documents. Especially relevant is the description of the workflow that we have come up with after several ways of approaching the problem and which we hope can serve other machine learning practitioners. We evaluate some feature extractor algorithms to find the most suitable for identity documents classification purposes. Using virtual machines on the cloud, we run feature extractors in parallel to label at a speed of 16 images/s. Then, we select a neural network architecture and hyperparameters to train a convolutional neural network. Our results give an accuracy of 98%. We detail the responses of the convolutional filters over the images. Results and source code in the appendix.

1.2. Introduction

As an example of image classification, we could help to classify the pictures of a group of zoologists working into the African savannah. We could have a database with one image and a category of each of the different wild animals:

Figure 1. Animal database: each animal has an image and a category

We would create a system for the zoologists that, given an image of an animal, will classify it into the right category and give its probability.

In our case, (fig. 2) we have taken an image of a gazelle, and the system gives us the right category of the animal. Each input image would be sorted by only one category of a set. So it is a **problem of multi-class, single-label classification**.

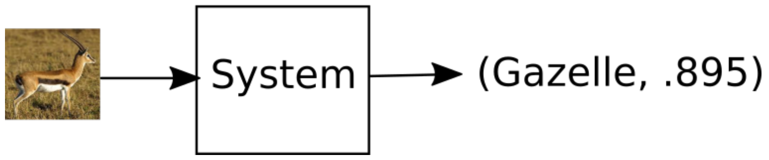


Figure 2. Given an image, get the category

To endow this work with some industrial interest, we have decided to design a classificator of identity documents although we hope our techniques should serve other people to perform different classification jobs.

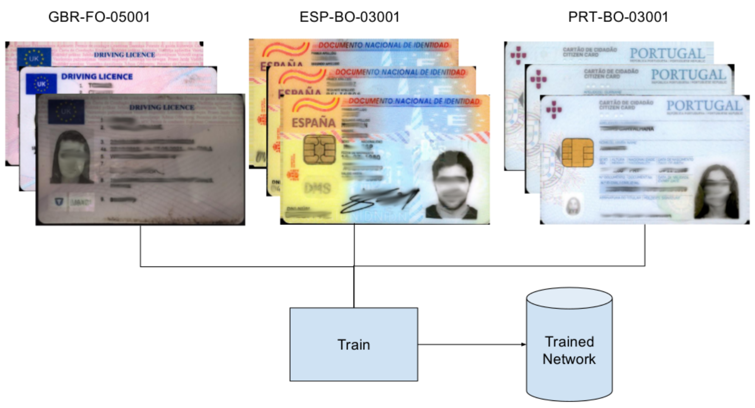


Figure 3. Training the network

It works in the following way: first we take many samples of the same document and assign them a label; ESP-BO-03001 for example. With these samples we train a structure called convolutional neuronal network and keep it. Subsequently, through a process called inference, we use that network to predict the category of a document never seen by the system.

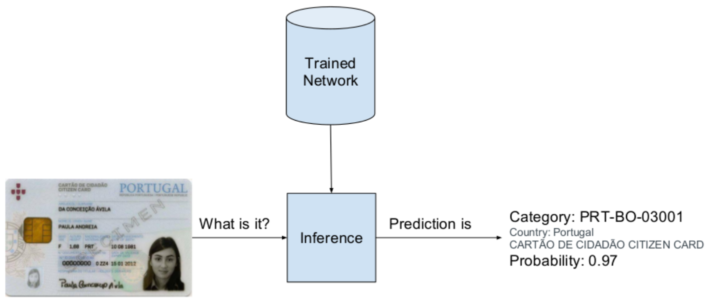


Figure 4. Making inference with a new document

We have divided this work into four parts: first, we will describe the areas of knowledge that we must know minimally to create an image classifier. Then we will explain a possible design and its implementation. Finally, with the acquired experience, we will give some ideas of how to extend the classifier.

1.3. Objectives

Our objective is to create an image classification system using a deep convolutional network.

This system will be specialized on identification documents like id cards, passports, driving licenses, etc.

We aim to an accuracy equal or greater than 95%.

2. What we need for image classification?

2.1. Previous knowledge

We need some understanding of several areas as mathematics, optics, electronics and programming.

Regarding mathematical knowledge, to achieve enough comprehension about the field it is only necessary a reasonable level of calculus, algebra, and statistics.

Some idea about how an image is captured an digitized can help. Some concepts about lenses, focal length, aperture, optical aberrations and scene illumination would be useful. In electronics, it would be using the idea of CCD, CMOS capture device, quantum efficiency or spectral response (QE).

Obviously, we would need to program a computer to do the classification task. The standard languages on machine learning (ML) are C and Python.

On the field of machine learning, we would need the concepts of supervised and unsupervised learning and the characteristics of each of the algorithms.

A reasonable level of calculus is necessary to understand the error minimization techniques as *gradient descent*. We need the concept of partial derivatives as we will often work with multivariate functions.

Regarding algebra, we will use the vectorized form of most equations because the representation is more compact and easy to understand. The matrix representation of the features of the study subjects is natural to us. Moreover, as we do need lots of data to feed our models, the matrix form becomes the best option to handle it.

Years of investigation in the field have produced very highly optimized algebra software libraries that perform the calculations easy and even run the tricky algorithms in a predictable and numerically stable way.

An essential concept taken from signal processing, called signal **convolution**, is used in machine learning although in a slightly different way. As an image convey 2D information, we can apply 2D matrix-based filters to increase or attenuate the image signal on the features we are interested in.

Machine learning is heavily based on mathematical statistics. We use concepts of this subject as the probability distribution, regression, maximum likelihood, and entropy.

We would need the concepts of image distortion, histogram, equalized histogram, color spaces, contrast, filters, deskew and image file format.

The use of GPU,s to do these calculations is one of the keys to the rise of the machine learning (ML). In ML, the GPUs are not much more used than matrix-multiplicators-and-adders. The high memory bandwidth permits to load and calculate amounts of data, but the principal difference with traditional CPUs is that GPUs can perform these calculations in very a parallelized way.

The brand NVIDIA is the the-facto standard in ML due to the availability of the CUDA programming library that makes *relatively* easy to perform these calculations on it.

Concerning cost, in these moments (Novembre 2017), one high-end computer graphics card to play games (and more serious work!) like the NVIDIA GeForce GTX2800Ti has the following specifications:

- Architecture: Pascal
- Memory: 12GB GDDR5X, Interface 384 bits, clock 10GHz, **Bandwith 480GB/s**!
- Cuda cores: **3384**

And the cost is less than **800$** [40].

But the new and powerful **Titan V**, with Volta architecture, can peak to 100 TeraFLOPS and outperforms the last generation on ML operations in a factor over 10! In Europe it's **3.100€**.[25]

Of course, not all algorithms can be vectorized and optimized for GPUs. In these cases, we can use multi-core CPUs or other forms of parallel computation.

2.2. Machine Learning

We are profiling the workflow of ML categorization projects. First, we need the categories where the input images will correspond. Then, we need a significant number of examples, categorize them and use them to **train** our **estimator** model or hypothesis. Finally, we can use the trained model to **predict** the category of new inputs never seen by the model.

Once we have the data labeled (categorized), we take one part to train the model and other to test the trained model. The data we use to teach the model is called *training dataset*. Also, the information we use to check the model is called *test dataset*.

2.2.1. ML primer

We are going to introduce machine learning from the perspective of their application to solving our classification problem.

Given the data labeled, we then make a hypothesis of what features of the input data prompt the outcome (in our case, the category). This assumption h can be a linear, polynomical, exponential or any other type or function of the input features.

$$h : X \to Y$$

Were h is the model, $X = (x_1, x_2, ..., x_n)$ are the input feature (the input image or some extracted features) and Y is the outcome of the model (the category in our case). We denote $x^{(i)}$ the $i-th$ input (image) and $y^{(i)}$ the output of the model when the input is $x^{(i)}$. $x^{(i)}$ can be a vector (or matrix) of features. Both X and Y are also called *tensors*.

When the target variable y to predict with the hypothesis $h(x)$ is continuous we call the problem a regression problem, when the outcome is a discrete variable we call it a classification problem.

If we assign an ordinal position to each of the possible outcomes (categories) of the model, then $y^{(i)}$ can be an array that is non-zero in the position of the class (this is named a one-hot vector representation).

The input dataset has a size of m inputs of $(x^{(i)}, y^{(i)})$ tuples. In our case, the $x^{(i)}$ are the images (or a set of features of the images) and $y^{(i)}$ is the category of the image i encoded as one-hot matrix.

2.2.2. Regression

To simplify, let's suppose we hypothesize that the output is a **linear function** of the input as in

$$h_\theta(x^{(i)}) = \theta_0 + \theta_1\, x^{(i)}$$

Let's define the *cost function* $J(\theta_0, \theta_1)$ as the **average quadratic difference** between our hypotesis and the real outcome,

$$J(\theta_0, \theta_1) = \frac{1}{2m}\sum_{i=1}^{m}(h_\theta(x^{(i)}) - y^{(i)})^2$$

Sometimes the results of this function are named the loss.

2.2.3. What Machine Learning really is

Much of *Machine Learning* sutff is related on how to find the parameters $(\theta_0, \theta_1, ..., \theta_n)$ that **minimize** the cost function J of the hypotesis.

2.2.4. Minimize the cost function

The most used method to find these parameters is to repeatedly take **small steps** α of each parameter in the **opposite direction given by the partial derivative** of the hypothesis function for that parameter. Eventually, we will reach a (global or local) minimum when the parameters converge. The algorithm can be expressed as:

repeat until convergence:

$$\theta_j := \theta_j - \alpha \frac{\partial}{\partial \theta_j} J(\theta_0, \theta_1)$$

where $j \in \{0, 1\}$ is the feature index number and α is the learning rate or also called the *step*.

Note that in our linear example θ_0 is a constant, so its partial derivative is zero and for parameter θ_1 the $\frac{\partial}{\partial \theta_1} J(\theta_1)$ part will tend to zero as we will approximate to the bottom of the cost function thus the algorithm converge because we get:

$$\theta_1 := \theta_1 - \alpha * 0$$

This method is called gradient descent.

2.2.5. Regression for classification

For classification tasks, we do not use a linear regression model but another called **logistic regression model**. It is a **binary model** that only gives us the probability that an input belongs to an output domain.

Our hypotesis will be now $h_\theta(x) = \sigma(\theta^T x)$ being θ^T the transposed parameter matrix of the hypotesis and σ the logistic function defined as:

$$\sigma(z) = \frac{1}{1 + e^{-z}}$$

The cost function for the logistic regression is similar to the regression one.

When we have several output categories (*multi-class classification*), we run the logistic regression on each class versus the rest. We call these probabilities *logits* and we will use the *maximum entropy function* to create a one-hot array that will contain a “1” on the most likely category and zeros in the others.

2.2.6. Large number of features and non-linear hypotesis

When we work with a small number of input features *may be* we can find a hypothesis (parametrized linear or polynomic combination of the input features) that fit with the output, but when this number is large it would be impractical or impossible to find any model.

2.2.7. Neural networks

An alternative method for this case are the neural networks (NN). The name comes from the similitudes with that we know of the brain operation.

A neuron receives weighted inputs w_i (the parameters called θ in the literature about regression) from the input x_i or other neurons and activates his output based on some "activation" function, e.g., surpass some threshold (step function). That is, the output of the neuron will be 1 if $\sum_j w_i x_i > threshold$ and 0 otherwise. In matrix notation we can simplify $\sum_j w_i x_i$ to $WX + B > 0$ with B is the threshold or bias. When the *bias* is high, it is easy for the neuron to fire a 1 and if it is very negative, it is difficult.

It is stated that an NN can model any non-linear function[24].

When the activation function is a sigmoid $\sigma(x)$ the output the neuron will be:

$$\frac{1}{1 + e^{-\sum_j wjx_j - b}} \in R$$

Moreover, the transition will be more smoothly than the with the step function. In this form, the neuron returns a real number and has the property that small changes in weights Δw_i or bias Δb excite a little change in the output.

Another widely used activation function is the Rectified Linear Unit(ReLU) that returns 0 for negative inputs and the same value for positives, that is it $max(0, x)$ or

$$f(x) = \begin{cases} 0 & \text{for } x < 0 \\ x & \text{for } x \geq 0 \end{cases}$$

The cost function of an NN is similar to the logistic but taking all the "connections" into account[23].

To minimize the cost function, we will use a quick version of the gradient descent called *stochastic gradient descent*. We use only some randomly chosen samples and average them to calculate the gradient.

An NN can have many layers of many neurons. These are named *deep neural networks* (DNN). An algorithm named *backpropagation* is used to train them.

On the output layer of the NN we have real numbers but as we are making a classificator it is better to transform these numbers to probabilities. That is, to represent a categorical probability distribution over k different outcomes. We can use the *softmax* function defined as

$$\sigma(z)_j = \frac{e^{z_j}}{\sum_k e^{z_k}}$$

2.2.8. Overfitting and regularization

When we are finding a hypothesis that explains the distribution of the data it could appear the phenomena of overfitting.

On the figure 5, we observe that the green line gives us the best accuracy on this exact set, but it is the black one that generalizes better for new input data. One solution is to smooth the green curve by regularization.

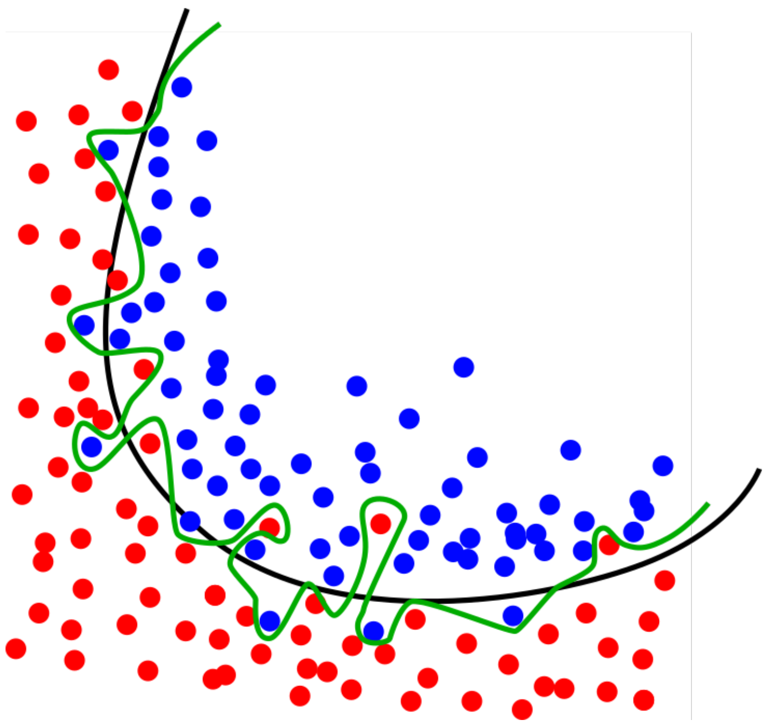


Figure 5. Overfitting. Credit: wikimedia/Chabacano

2.2.9. Convolution

When we are talking about the *input feature array* $x^{(i)}$ we speak about a one-dimensional matrix. We feed the estimator with this vector and use the outcome to train it. However, what about 2D information like images?. In this case, we can flatten the 2D image array into a larger 1D by concatenating the rows. Sure this will work but, unfortunately, we will **lose the spatial relations** that are intrinsic to the images[7].

A typical operation taken from maths and discrete signal processing although it *is not precisely defined in the same way* is the **convolution**. We slide a signal, kernel, filter or feature map h over an input signal (an image x), multiply element by element and then add them up.

More graphically, if we have an input image I and a kernel K, the convolution of the two, $I * K$, will be like in fig. 6.

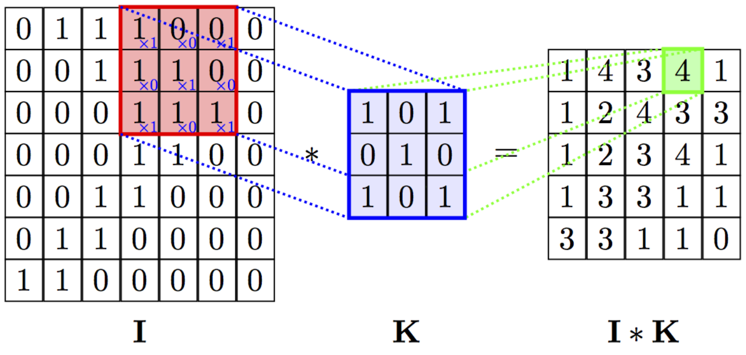


Figure 6. Convolution I*K

Note that, in this case, we have deleted the borders of the image that do not fit into the dimensions (3×3) of the filter. This way to apply the convolution is named "valid" convolution.

As we do not flip the kernel before slice it (like in signal processing) the operation is the same that a **cross-correlation** and can be written as:

$$S(i,j) = (I * K)(i,j) = \sum_m \sum_n I(i+m, j+n)K(m,n)$$

By designing these filters, we can *amplify* (increasing or attenuating) some features contained in the 2D information of the image. That is it,

> the convolution operation works increasing the signal of one feature while attenuating the rest

An essential property of the convolution is the equivariance to translation[4]: if we shift the input image, its correlation will be proportionality moved.

Typical examples are the gradient-based edge filters, gaussian blur, color filters and much more. In any way, applying filters to an image gives us a *higher level of abstraction* as we extract features in a way similar that our brain does.

A convolutional neural network (CNN) is an NN that uses the convolution operation in one or more layers. For a comprehensive review of their use in image classification see [Rawat and Wang] [5].

2.2.10. Pooling layers

A convolution layer is typically followed by a *Pool layer* that summarizes some rectangular area e.g. max pooling returns the max value of a neighborhood.

Unlike convolution (cross-correlation) pooling has some invariance to small translations because it takes a statistic of a particular area.

Pooling layers capture essential information, and it reduces the size of the next network layer. We will need pooling on the last layer of a categorization network as it will have a fixed number of neurons.

2.2.11. Other ML tools

We can include some other tools in our arsenal to make the machines *learn*.

KMeans

When we have much *unlabeled data*, and we want to cluster it automatically, we need some *unsupervised learning* to do the task.

Kmeans accepts an initial number of cluster centroids[30] and for each data sample, recalculates the position of each centroid minimizing the inertia (the quadratic sum of cluster's distances). The final result will be that the initial set is classified in groups of equal variance.

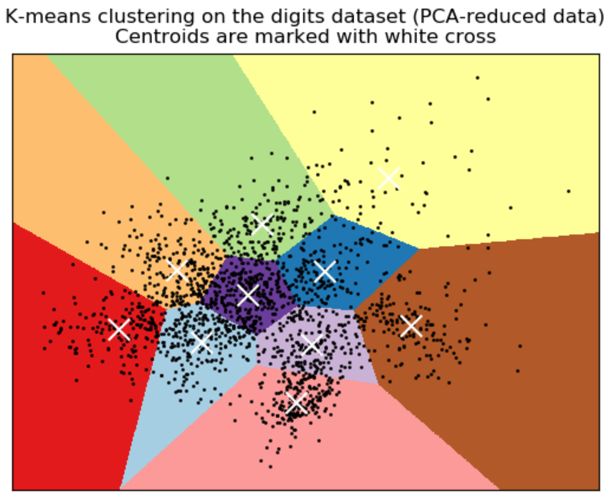


Figure 7. Clustering digits dataset. Credit: scikit-learn

Regarding our problem of how to label a bunck of images, perhaps we can use kmeans to have an initial classification.

SVM

Support Vector Machines is classification algorithm[29]. It is said it belongs to the group of *supervised* learning methods because we can use it when we already have the data labeled and want to classify new one. SVM tries to distribute the data taking the maximum (hyperplane) distance between the points of another class.

We can use different *kernels* that would best fit according to our target features: linear, polynomial, and RBF or even create our kernel.

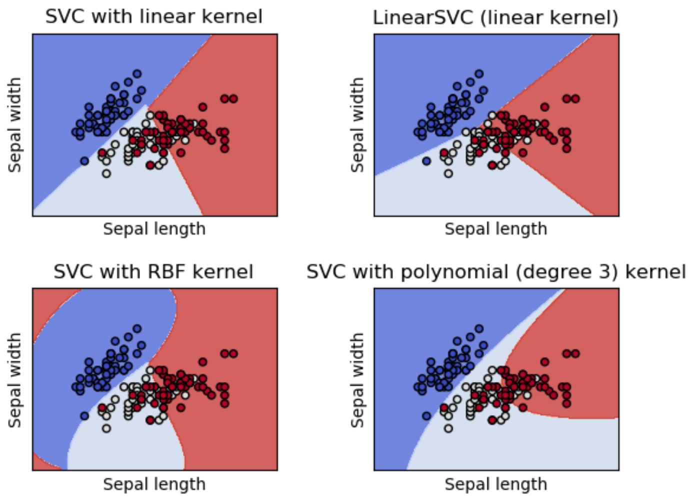


Figure 8. Apply SVM with different kernels to the same dataset. Credit: scikit-learn

2.2.12. Principal component analysis (PCA)

With PCA we can reduce the dimensionality of a tensor. The new set is an orthogonal transformation of the original, and the first component retains the maximum possible variance, the second component the second maximum and so on.

The method was invented by the statistician Karl Pearson in 1901[38]. It is also called Karhunen-Loève transform (KLT) in signal processing or Singular Value Decomposition (SVD) in mechanical engineering.

On the figure, we show the projection of the two principal components of the four that have the IRIS[31] (a type of flower) dataset. Note that in this situation we can easily apply an SVG to classify new flowers. We have simplified the dataset while retaining most of the variance.

2.2.13. The importance of good datasets in image ML

As we can see, *gradient descent* only works if we have a large dataset to train the network. The *Learning* part is done by automatically fine-adjusting the parameters w_i, but as the steps are so small we need a large number of examples to capture the subtleties of data like images. Shall we finish this chapter with this idea in mind:

The predictor will be better trained as more well-labeled data is provided.

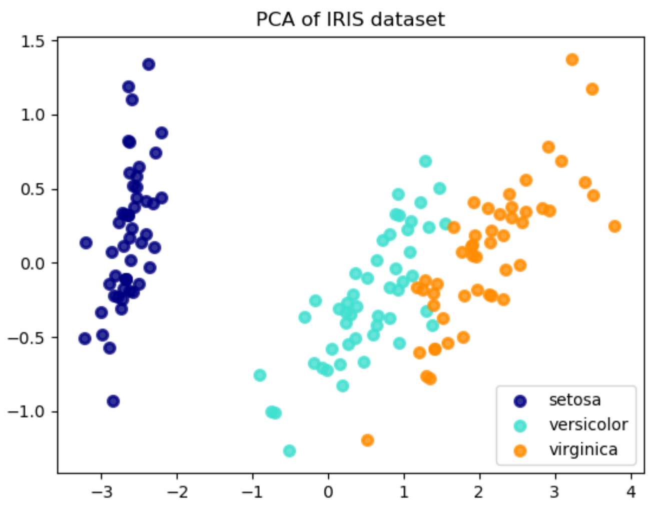


Figure 9. Example of PCA. Credit: scikit-learn

2.3. Computer vision

According to the wikipedia[36], computer vision (CV) is an interdisciplinary field that deals with how computers can acquire, process and analyze images to produce numerical or symbolical information.

As vision is the primary sense, humans use to get information about the world. Many studies are dedicated to how we can reproduce this behavior on a machine.

The broadest used library in CV is OpenCV. Initially developed by Intel, now is in public domain. It is written in C but has links in python. The python links are easy to use than C, but not all modules have the bindings. In these moments, there are not python binds to the CUDA modules (to use the GPU).

2.3.1. Feature extractors

To characterize an image, we try to extract some features that uniquely describe the model and helps us to both differentiate from others or search other similar photos. These unique features can be corners or abrupt changes of color.

When we find these unique parts of the image, it is said to perform an image **feature detection**. To get the ability to find the same features on other images, we need the characteristics of the region around the key point. This is called **feature description**.

It is important to note that all the feature extraction methods that we are going to describe have some degree of invariance to scale and rotations.

Corners are useful descriptors, and there are the Harris and Shi-Tomasi algorithms to locate them although there are not invariant but these methods also suffer in changes of illumination.

In 2004, David G. Lowe[1] wrote a paper describing a method to extract distinctive key points from images instead of having changes in orientation, scale, illumination or noise. A kind of mini-revolution became in this field because this method[32] allowed us to find images in image sets or other pictures by matching the key points.

Unfortunately, the algorithm, like others as SURF was patented in the USA, but it was the beginning of a whole new area of investigation.

The OpenCV[27] library comes with a free feature extractor named ORB (Oriented FAST and rotated BRIEF) and other designated BRISK. It was delivered by Ethan Rublee[6] and others in their paper *ORB: an efficient alternative to SIFT or SURF* in 2011. It is faster than SIFT and takes ideas from other extractors like FAST and BRIEF.

The latest algorithm included in the OpenCV library is **KAZE**[2] and its fast version **AKAZE**[3], created by Pablo F. Alcantarilla of the University of Alcalá and others.

Let's see an example of both extractors, AKAZE and ORB:

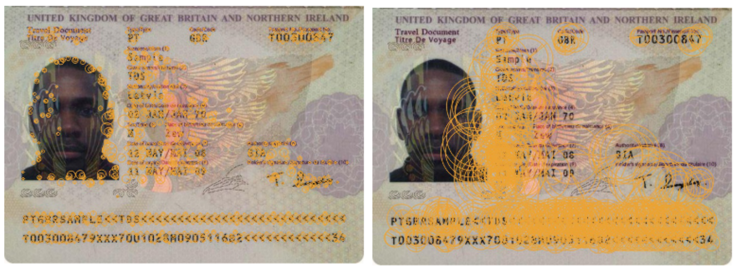


Figure 10. Example of AKAZE on the left and ORB on the right.

Image capture hardware

Some capture hardware that can be:

- a table scanner
- a traditional photographic camera (handheld or with support)
- a mobile phone
- a video camera

Each system has with its particular combination of lenses, sensor, movement compensation and image processor.

It is also crucial the illumination of the scene that ideally would be uniform and with and spectral composition that match with the sensor used. Regarding this, some id documents are sensitive to the ultraviolet light.

A high-quality table scanner is the best scan option. Some documents like the passports are likely to be bend when scanning it because of the cover.

Blur

The photography terms *unfocused* and *blurred* are easy to interchange because **their effects on the image are very similar**, but the phenomena is caused by different things:

- *Unfocus* is due to the lack of coincidence of the focal plane and the sensor plane.
- *Blur* is caused by the movement of the sensor while capturing the image mostly because of the shutter movement.

In an out of focus image (or image area) the energy of a light ray is spread across a non-punctual area (a circle of confusion). To blur a picture, we convolve it with a kernel that follows a Gaussian distribution. The effect is attenuate high frequencies (low-filter), and that resembles an out-of-focus image.

Motion

When we extract the document from the scanner before the scan is terminated, it can appear a distortion named motion (linear).

Low contrast

Low contrast is one of the more significant problems when obtaining images in a non-controlled environment as it depends on the scanner quality, the illumination of the scene and the physical state of the document.

The contrast of an image is the difference between the maximum and the minimum luminance. It is also essential *the distribution* across the available range (the more variance, the better). When the range is small or bad distributed, there isn't enough information to distinguish one object from another.

Salt-and-peeper noise

This type of noise can come from a low illuminated scene. The number of photons counted by each photocell is proportional to:

- the illumination of the scene
- the area of each photocell
- the sensor spectral response
- the exposure time (more time, more photons can be counted)

In low-quality scanners or mobile phones noise can appear mostly because of the *thermal noise* of the sensor counting photons that are not there. Moreover, in low light conditions, the SNR will be low and the noise will be more relevant in the final image. It is said that median filters can remove some of this type of noise but at the cost of change image properties that help extractors.

This type of noise is not usual on high-quality table scanners or high-end cameras.

Partial occlusion

This case can be tricky depending on the area occluded (see fig. 11).

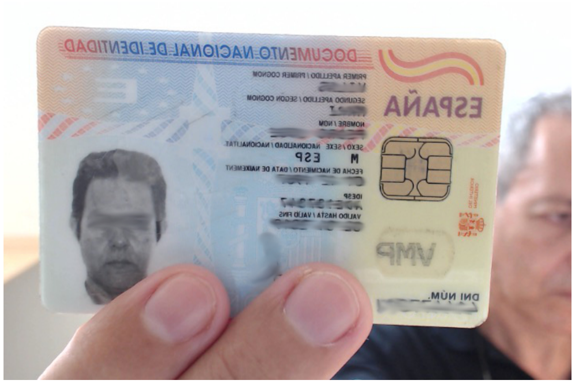


Figure 11. Occlusion

2.3.2. Other feature descriptors - HOG

There are other feature descriptors available. The histogram of oriented gradients[37] (HOG) is intended to recognize objects (like pedestrians) on a scene and it is scale invariant.

HOG counts occurrences of gradient orientations in the image that will represent shapes. Because it only takes the gradient and not the absolute value, it is resistant to changes in illumination. (See fig. 12)

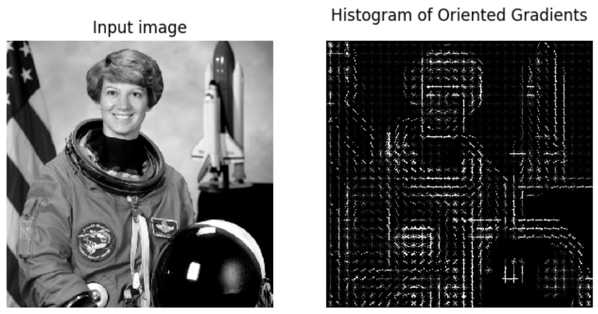


Figure 12. HOG example. Credit: scikit-image.

2.3.3. Feature matchers

When the features were extracted, we would found which picture on the target data set is more similar.

The feature extractors return tuples of (key point, point-descriptor). The key point contains the coordinates of the point, and their descriptor provides information about the point[28].

On the ML field, we call

- **query image** to the input image that we want to match
- **train image** (or trained image) to each of the images we have labeled.

We usually create a matcher object and then use it over several query and train images. Doing this will save us some initialization time but has some issues in multiprocessing as we will see in short.

We establish a threshold to the correspondence between descriptors (Dr. Lowe paper proposed 0.7) to take it for good. Besides, we give a minimum number of descriptor matches as a low number could be a spurious correspondence.

If we set these parameters high, we could not find a document match, but we will avoid false match (wrong labeled documents). If we set the parameters low, we will have more document matches, but they can be wrong.

To calculate the distances between points, we will use the $L2-norm$ (Euclidean)[34] for methods that returns vectors of real-valued distances and the Hamming distance[33] for methods that return a binary vector (like ORB).

The matcher return a set dmatch objects with the following attributes:

- *distance* between descriptors (lower is better)
- *trainidx* is the index of the descriptor in train descriptors
- *queryidx* is the index of the descriptor in query descriptors
- *imgidx* index of the train image

We have two types of matching algorithms: brute force and FLANN.

Brute force matcher

Brute force matchers will examine each of the points on the query image and try to match it with each of the train images; only the closest points will be returned.

knn Matcher

FLANN stands for Fast Library for Approximate Nearest Neighbors. It uses a series of optimisations (trees) to search neighbors points and performs better than brute force on large datasets.

2.3.4. Homography

To find an object image into another image we use homography. The query image can have the document rotated, translated or in other plane orientation. The homography operation returns the **affine transformation** we need to transform the query image plane into the train image plane.

Given this affine transformation, we can project the query image into the target image plane to rectify it. In OpenCV we can use `cv2.findHomography` and `cv2.prespectiveTransform` functions to do it.

2.4. NN frameworks

To later train the NN we can use one of these two libraries.

2.4.1. TensorFlow+keras

TensorFlow[18] from Google is widely used, and there are much code examples and documentation. Keras[11] facilitates the implementation of the model graphs and the training.

2.4.2. Mxnet+gluon

Mxnet[17] is the open source ML library from Apache Software Foundation. There is only one place to get information, but it is very well documented with lots of jupyter notebooks. Gluon[16] is the keras equivalent for mxnet.

2.5. Preprocess the images

Before post an image to the feature extractor and matcher it would be convenient to rectify it. It takes less time to calculate the homography and we are surer that we are using consistent data. If the scan comes from a table scanner, it is common that it is rotated.

3. Design

As it has been said, we will use a convolutional neural network to classify the documents. Our first task is going to be to create the image database to train this network.

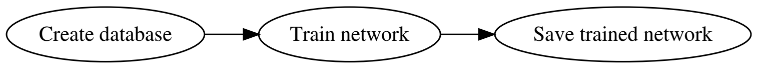


Figure 13. Build the network

The training itself consists of model, train, validate and test our classifier with the database.

Once the network is tuned, in a second phase, we can load it and use it to make inferences.

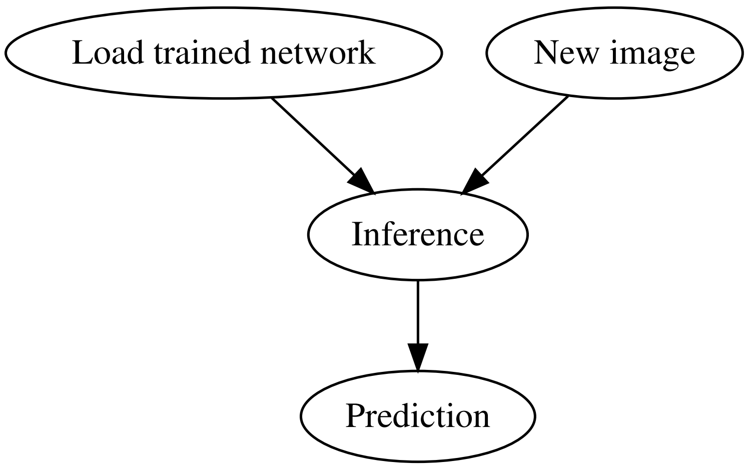


Figure 14. Use the network inference

3.1. Database creation: getting and labeling the data

With no doubt, if we are creating a classifier from scratch, **the primary job is to collect, organize, categorize and label the sample images**. It is worth to spend enough time in design the strategy to make the sample set.

The accuracy of the neural network classifier will increase with the number of samples and their variance[8].

In general, we start finding the images for any method like:

- image databases
- Internet
- taking the images for ourselves
- scanning paper-based docs
- etc.

From the first bunch of images, the first step will be to infer the categories in which the images will fall.

One initial condition of the problem would be the availability of a category index of images, that is it, a set of already labeled images (one image is enough). An example will be a marked image of each terrestrial mammal of the savannah if we are classifying pictures of these animals. In this case, we can get the category by comparing a given image with all the index using some comparison method.

There are some known methods to compare two images, but the most used are the comparison of histograms and the feature extractors.

Given the characteristics of the expected input images (id-documents took in unknown conditions of light, resolution, exposure) the histogram method can be discarded.

Regarding the feature extractors, they work relatively well, but they take **enormous amounts of computing time**. Moreover, this kind of algorithms are not easily *vectorizable*, and thus they are not appropriate to implement them on a GPU although they exist in the same opencv library (there is not a cuda-AKAZE). Fortunately, we can extract image features in parallel using several regular CPUs.

The features of two images can be compared (they are invariants to the size of the images) and thus calculate the percent of feature matches as an estimation of the likelihood that the two images belong to the same category.

One time we had an image index (see 3.1.6), our next step will be to pre-calculate the features of the index images to speed-up the matching process.

3.1.1. Tagging the images

We will explore three techniques to tag the input images:

HOG+Kmeans

Basically, we calculate the oriented histogram of the images and cluster them with kmeans in order to be able to visually identify the category of the images next to each centroid.

Algorithm 1 Tagging with HOG+Kmeans

```
for all images do
    Calculate HOG of image
end for
Cluster HOGs with kmeans
Visually identify the document model of the cluster and assign it to members next of
the centroid
Check or discard images not belonging to any category
```

Image feature extraction+Kmeans

Could be better than HOG as the precision increases but impractical with a large number of documents because of the kmeans initialization. This method can be adviseable when we know the number of categories before the clustering.

Image feature extraction+kNN Matching

Can be a good option but at the cost of calculate the features of all images (set and index). The k-nearest neighbor should be a good matcher option for a large number of files. The number of matching points over the total can be used as a probabilty.

Algorithm 2 Tagging with AKAZE+kNN

```
for all index images do
    Extract AKAZE features
    Save index features as a separate file
end for
for all images do
    Extract AKAZE features
    Save features as a separate file
end for
for all feature files do
    for all index files do
        matches:=kNN(feature_file, index_file)
        prob:=len(matches)/total_keypoints
        if prob > THRESHOLD then
            assign index category to image
        end if
    end for
end for
```

Whatever the method we will take, we will need to supervise the results manually. We find the maximum quality of our training dataset as it is critical to get good results later.

3.1.2. Creating the category index

If we do not have an index set, we can first try to **cluster** the images with some clustering algorithm like kmeans[26] and then manually infer the labels. Unfortunately, kmeans does not work well if we do not have an initial estimation of the number of categories or this number is higher and the images are similar between classes.

Algorithm 3 Extracting categories from a bunch of images

```
guess an initial number of different documents
cluster images with kmeans
visually identify the clustered documents
label all the documents of the same cluster with the same label
visually inspect all documents to check if the label is correct for all documents
```

In our experiments, we had a kind of reasonable results by first extracting the features of all images and then try to cluster them. Then, we manually checked the clusters identifying the images nearest to the centers and created the categories.

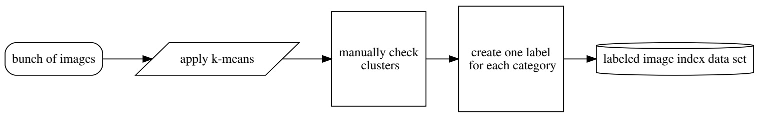

bunch of images
apply k-means
manually check clusters
create one label for each category
labeled image index data set

Figure 15. Build the category index

3.1.3. Categorizing the images

Once the index built, we would apply the feature matching methods.

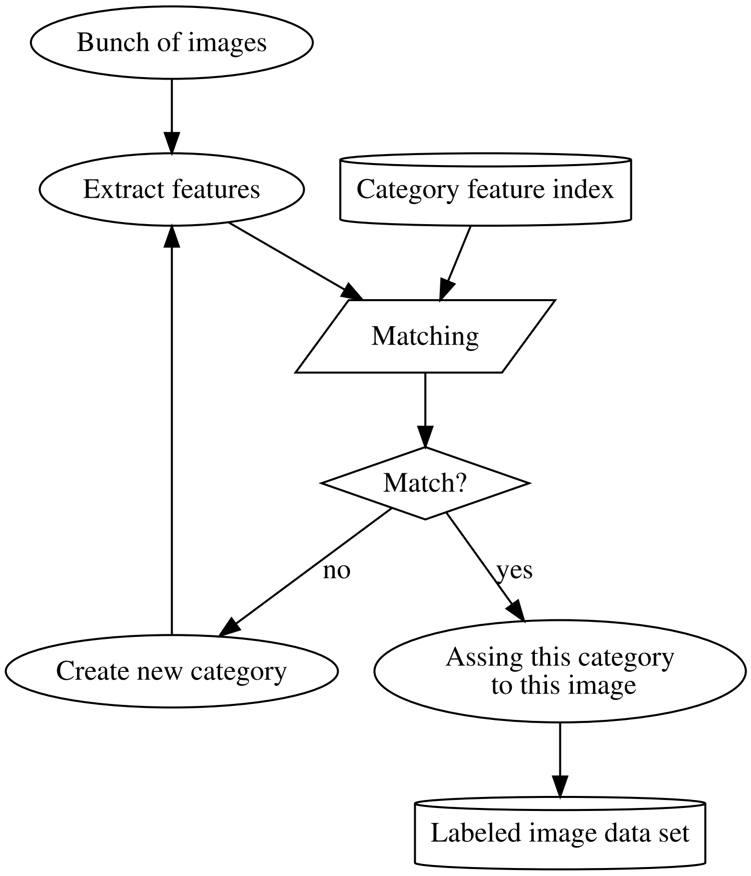


Figure 16. Categorizing process

As we cannot be sure that with the last step we have created all the categories, we can tune our category index and at the same time label all the images of the dataset with the following process:

Algorithm 4 Categorizing the images

```
 1: img:=1
 2: while img $\leq$ numImages do
 3:    get image number $img$
 4:    if find a match with the index then
 5:       assign the category label to the image
 6:    else
 7:       create the category
 8:       append category to the index
 9:       img:=0 {restart loop}
10:    end if
11:    img:=img+1
12: end while
```

Each time that we create a new category, we must re-check all the dataset (see alg. 4 lin. 9) because an image that belonged to a class can finally belong to a newly created one. See diagram on fig. 16.

3.1.4. Mixed strategy

If we have a huge dataset, **the feature matching process can be impracticable in time**. That contrasts with the blazing fast inference speed of a neural network classifier. We can get an alternative approach if we first label a random sample of the dataset with feature matching methods, build a network and then use it to classify the rest of the data.

Algorithm 5 Mixed strategy

```
new_cat:=true
while new_cat is true do
   new_cat:=false{assume there wont be more categories}
   random_set:=random(images)
   rindex:=classify_with_feature_matching(random_set)
   network:=create_cnn(rindex) {Create network}
   for all images not in random set do
      prob_category:=network.predict(image)
      if prob_category > THRESHOLD then
         assign category to image
      else
         insert new category into the index
         new_cat:=true
         break
      end if
   end for
end while
```

Once the subset is exhausted, we can build a new network with all the dataset. The key here is to use the probability as an estimator of new categories so that we will need a *softmax activation function* on the last layer of the network. The size of the random set will determine the number of times the whole process run as the value of THRESHOLD the quality of the results. See fig. 17 and alg. 6.

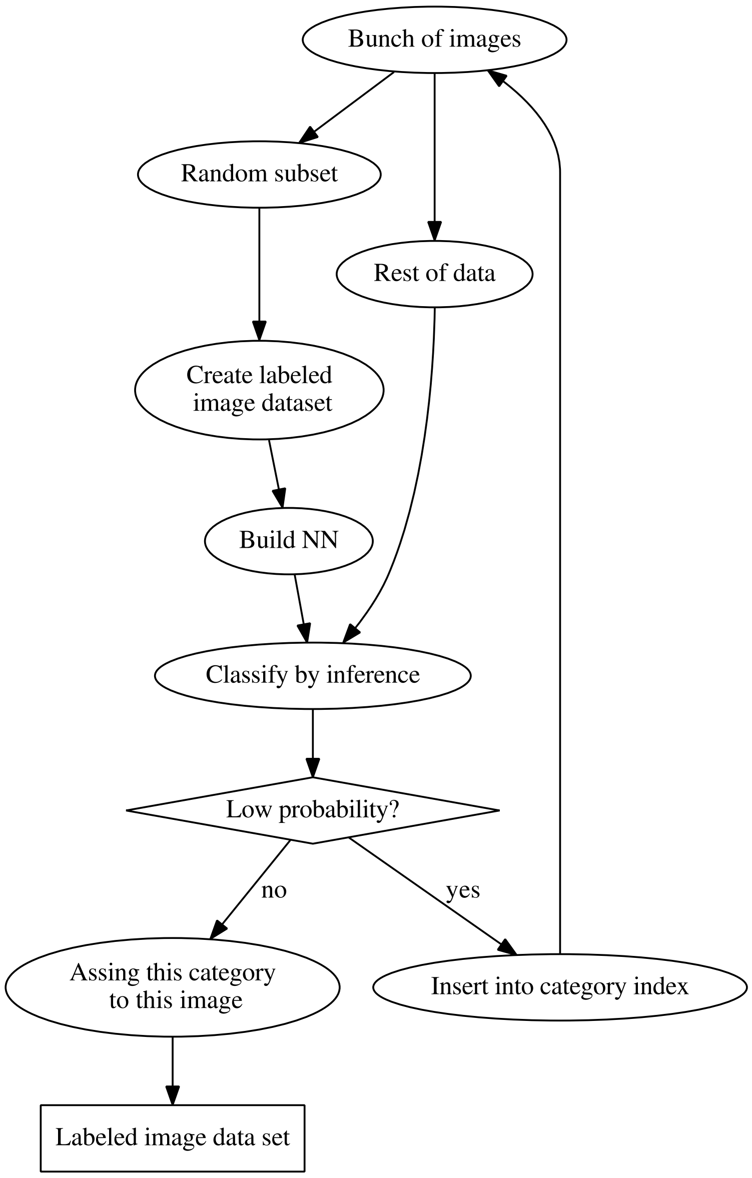


Figure 17. Labeling with a mixed strategy

Quality of the dataset

It is **mandatory** to manually inspect all the dataset before the training phase. We will need some tool to check and eventually update or delete images of the set.

Most of the time spent on the project can be doing checking tasks.

3.1.5. Learning about

When starting an ML project, it is useful to teach oneself about the subject. Reading books, websites, forums, watching videos, etc... oneself can become to get an idea about the relevant features of the issue. That is named feature engineering[41].

Quotting Jason Brownlee[9],

> Feature engineering is the process of transforming raw data into features that better represent the underlying problem to the predictive models, resulting in improved model accuracy on unseen data.

In our case, as an example, we could try to use the calculated keypoints of the images as the input data instead of the images itself, but we our requirements need very high precision in the prediction.

Deciding the categories

The study of the subject will give us the labels we will need to work with the dataset.

It is usually better not to reinvent the wheel and go for academic, industrial or other types of standard. Most of the time, the set of categories will come from the project. Some examples can be: types of cells in microscopy images, types of stars, authors in sound registers, recognition of the speaker's language, ...

As explained before, the design of the system must be flexible enough to accept new categories at any time of the development since it is a situation in which we can often find ourselves in projects in which we start from scattered data.

3.1.6. Our index: PRADO

In our particular domain, we have selected the PRADO project as our target.

PRADO[14] stands for Public Register of Authentic travel and identity Documents Online. It is a document identification database maintained by the European Council, Directorate General Justice and Home Affairs. It is used by many administrations like Interpol or Frontex. The PRADO glossary[13] can be a good starting point to learn about id documents.

The documents are organized by country, category, type, document-number, and version.

For example the document code **FRA-AO-01001** consists of:

- FRA for the France country
- A for pAssport
- O for Ordinary
- 01 for the document number (two digits)

- 001 the version of the document number (three digits)

Each document has almost an image in visible light on the recto side.

Each folio of the document has the recto and verso[39] side referring to the "front" and "back" sides. The terms come from Latin *rectō foliō* and *versō foliō*, translating to "on the upright side of the page" and "on the turned side of the page", respectively.

It should be noted that the recto of a **passport document (A)** is the cover of the passport and the verso is where the information is written.

The complete tables of categories and types are

Document categories according PRADO

- A - Passport
- B - Identity card
- J - Travel document issued to non-nationals
- C - Visa
- D - Stamp
- E - Entry paper
- H - Residence related document
- F - Driving license
- G - Vehicle license/log book
- I - Seafarers identity document
- P - Civil status/other official document
- S - Special authorization card
- W - Work permit
- X - Other document

Document types according PRADO

- **O** Ordinary document
- **D** Diplomatic
- **S** Service/official
- **P** temporary/provisional/emergency
- **Y** Related/associated document

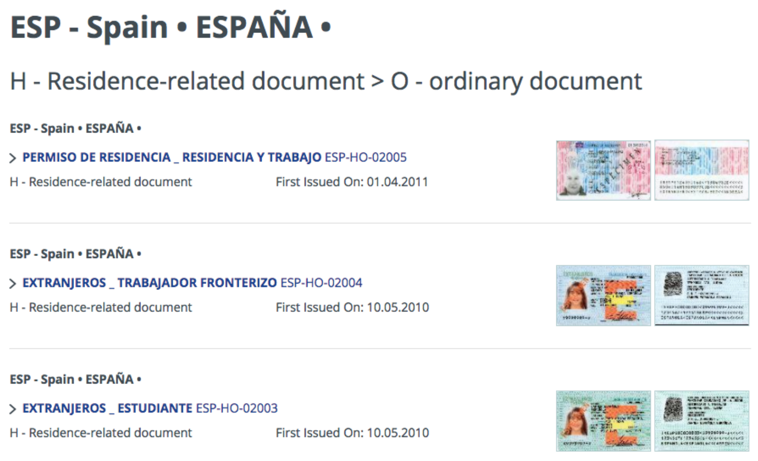
The civil status (**P**) category has the following types:

- **B** Birth
- **M** Mariage
- **T** Death certificate
- **N** Nationality/citizenship
- **R** Divorce

3.1.7. Downloading resources from the web

It is great to have a database available online but how can we download and format it? It can be easy or difficult depending on how the website is structured.

In our case, PRADO, we denoted that although there is not a general document index, we have one by country[12].



ESP - Spain • ESPAÑA •

H - Residence-related document > O - ordinary document

ESP - Spain • ESPAÑA •
> PERMISO DE RESIDENCIA _ RESIDENCIA Y TRABAJO ESP-HO-02005
H - Residence-related document First Issued On: 01.04.2011

ESP - Spain • ESPAÑA •
> EXTRANJEROS _ TRABAJADOR FRONTERIZO ESP-HO-02004
H - Residence-related document First Issued On: 10.05.2010

ESP - Spain • ESPAÑA •
> EXTRANJEROS _ ESTUDIANTE ESP-HO-02003
H - Residence-related document First Issued On: 10.05.2010

Figure 18. PRADO documents by country. Credit: EU Council.

3.1.8. Pandas or sql?

To handle the documents we can use both pandas (a kind of standard in python) or any SQL database server.

We prefer SQL only because we are personally more used to it. Nevertheless, it is worth to make a note of the fact that when you are processing much (possibly unstructured) data, an error can quickly happen. If you use pandas, you may not save the results until the process is terminated overall data and a program error could lose the calculations performed until that time. If you use SQL, you may commit after each count and no loss of data will happen in case of a failure.

3.1.9. Handle image files

We have found a combination that works surprisedly well moving large amounts of data (as images).

If we work with zipped or tar files of many gigabytes of images, it is difficult with traditional ssh-copy (SCP) to move them from the workstation to the calculation servers on Amazon Web Services (AWS).

With Dropbox or AmazonS3 you can first copy the files to this service and then create a shared link. Then, form AWS you can download the file with the link with *wget* at a speed of many Gb/m! If you work with S3 you can use the excellent boto3[21] library.

3.1.10. Tasks parallelization

When calculating the image features, we will need all the available computer capacity that we can achieve or otherwise we will spend days or weeks until they terminate.

To speed up the job we will want to use all the capacity of our CPU but this is not as easy it seems.

For Python, the best option is to use the multiprocessing package. It relies on the os to perform the tasks and saves the problems with the Python Global Interpreter Lock and the traditional spawn of threads. Nevertheless, multiprocessing programming is a bit tricky, and it is advisable to read the documentation carefully. Isolate the main module. If we share one or more variables with other processes take care with the type or the implicit locking mechanism will block multiprocessing.

3.1.11. AWS

To extract image features, we will need many cores to parallelize the task and to train a CNN we will need a powerful GPU. However, these machines are expensive.

The best economical option is to *rent* virtual machines from a provider like Amazon.

Amazon has even SOs already tailored for machine learning, with all the libraries (including CUDA) included or compiled.

3.1.12. Initial features

When downloading the images, we should make sure we catch all the associated features we can.

The reason is straightforward: imagine the target has a size of 2400 images (approx. PRADO on Nov 2017). Then, for each query image, we will need 2400 matching operations what it will consume much computation resources and time because the cost is linear with the number of images $\mathcal{O}(n)$.

If we would have any input **feature** that match with the target (in our case the country or the document category) we can massively reduce the number of matching operations because only this nation, class (or both) target images were necessary to match.

3.1.13. Statistics

We have designed a document table with the following structure:

Listing 1 Document table

```
CREATE TABLE DOCUMENTS (
    ID integer NOT NULL PRIMARY KEY,
    COUNTRY TEXT,
    YEAR TEXT,
    recto text, /* visually checked model */
    verso text,
    recto_akaze text, /* discovered model with akaze method */
    recto_brisk text,
    recto_orb text,
    recto_svm text, /* predicted by svm */
    recto_nn text, /* predicted by nn */
    recto_cnn text, /* predicted by convolutional nn */
    unframed integer default 0, /* the image is  unframed */
    noprado integer default 0, /* image does not match prado db */
    test integer default 0 /* use this image as a test image */
);
```

The `recto` and `verso` fields are the visualy inspected. `akaze`, `brisk` and `orb` are the models matched by their respective methods. Then, to calculate the number of hits of each method e.g. akaze we can do

```
select count(*) from documents where recto=recto_akaze;
```

3.2. CNN Modeling

As we are dealing with neural networks, our first step will be to imagine a network architecture that will model the behavior of the classifier. There are no rules about the number of layers, its connections, its types or its activation functions. We can only go to the literature to find the model with best performance on a problem given. We could need dozens or hundreds of iterations until we see the design and the hyper-parameters that provide the best results, that is, it generalizes better and makes predictions in a more precise way.

In every moment, we will be aware of not to overfitting the network by observing the loss and accuracy of the model.

3.2.1. Model

In our case of an image classifier, we will use a deep convolutional network architecture.

We use the term deep in the sense that the network will have more than one hidden layer. Some of these layers will be pooling layers that will summarize the results of the upper layers and others will be dropping layers that will help to mitigate the phenomenon of overfitting. The convolution layers will make the operation of the convolution of the upper layer with a filter bank. The signal level that the neurons of a layer are transporting to the next is given by the activation function of the layer.

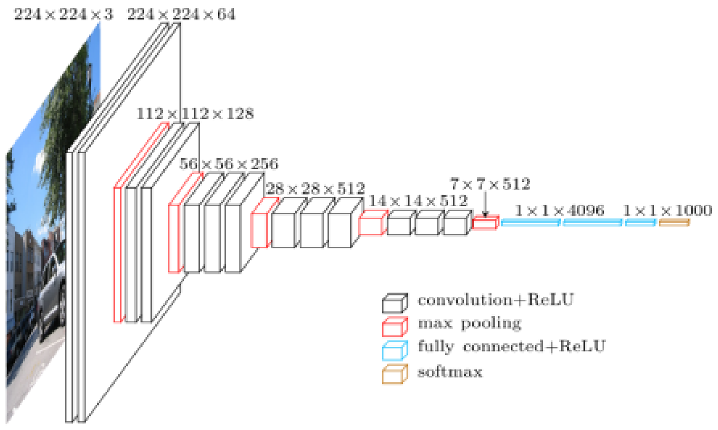


Figure 19. VGG16 architecture. Credit: cs.toronto.edu

We have chosen a VGG16-like[22], a model developed in 2014 by the renowned Visual Geometry Group of the Department of Engineering Science, University of Oxford.

It is a bunch of convolution layers, each followed by a max pooling. The activation function is always a RELU. Finally, a dense (fully connected) layer connects to a final softmax with a size of the number of categories of the model.

Our exact architecture is:

Listing 2 Model summary

```
Model summary
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d_1 (Conv2D)            (None, 218, 152, 64)      1792
_________________________________________________________________
conv2d_2 (Conv2D)            (None, 216, 150, 64)      36928
_________________________________________________________________
max_pooling2d_1 (MaxPooling2 (None, 108, 75, 64)       0
_________________________________________________________________
conv2d_3 (Conv2D)            (None, 106, 73, 64)       36928
_________________________________________________________________
max_pooling2d_2 (MaxPooling2 (None, 53, 36, 64)        0
_________________________________________________________________
conv2d_4 (Conv2D)            (None, 51, 34, 64)        36928
_________________________________________________________________
max_pooling2d_3 (MaxPooling2 (None, 25, 17, 64)        0
_________________________________________________________________
flatten_1 (Flatten)          (None, 27200)             0
_________________________________________________________________
dense_1 (Dense)              (None, 512)               13926912
_________________________________________________________________
dropout_1 (Dropout)          (None, 512)               0
_________________________________________________________________
dense_2 (Dense)              (None, 144)               73872
=================================================================
Total params: 14,113,360
Trainable params: 14,113,360
Non-trainable params: 0
```

Where we can see a first convolution layer that accepts images resized to a tensor of (218, 152) and a bank of 64 filters. The second layer has 64 filters itself, and then we summarize by applying max-pooling which gives us tensors of (108, 75). From here we use sequentially two layers of conv+maxpooling always with 64 filters.

The eighth layer will flatten the results in a tensor of size (27200) that is (25x17x64). Finally, we link dense1 and dense2 with a dropout(0.5) to avoid overfitting the model[19].

The last layer is a tensor (144) that reflects the **number of categories** of identity documents. It is activated by a softmax function, so each dimension is the probability that the input image belongs to the class located in this position. A max operator over this tensor we will get the most likely category of the input document.

3.3. Dataset partition

Once we reach a quality sample labeled database, we must divide it into three parts[35]:

1. Training set (60% of samples)
2. Validation set (20% of samples)
3. Test set (20% of samples)

We will use the training set to efficiently train the network, namely to adjust the parameters (weights and bias) of the net in a way that output the same label when the input is an image of some category.

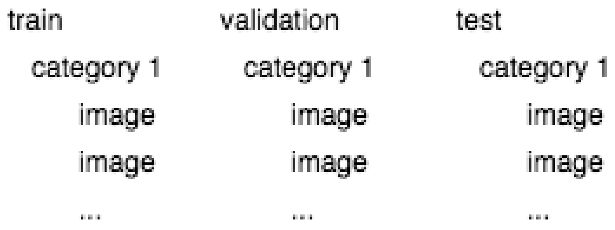
The validation set is used to tune some hyper-parameters of the network like the learning rate, the initialization strategy or even the number of hidden layers. We can detect problems like overfitting with this set.

Finally, we use the testing set to feed the network with never seen images and make a statistic of the accuracy of the classification.

Probably, we will need a large number of iterations over model-train-validate-testing until getting the desired accuracy. In this phase, it is worth to use a high-performance GPU (or TPU) in our facilities or external virtual machines like Amazon or Google.

It is imperative to shuffle our dataset before the partition to avoid biasing the network. If for any cause the system were influenced by the data partition, it will not generalize well.

Once the data is shuffled, we will organize it in category directories so our final structure will be like in the figure 20



```
train             validation        test
  category 1        category 1        category 1
      image             image             image
      image             image             image
      ...               ...               ...
```

Figure 20. Organization of the dataset before training

One important detail is to have the same categories in each partition. We must check it before the training.

4. Implementation and results

In this section, we are going to describe the implementation of the ideas discussed before. First, we study the tolerance to image aberrations of the feature extractors, some useful web utilities and a short application of the rectification image process. Then, we will download and label the documents from the PRADO website. These will be our categories. Next, we will extract the features of these images to build the feature index. We will match the feature index with all the pictures of our dataset to categorize them. Immediately, we train, validate and test the network.

Finally, we will explain what the system is doing and how can classify the images.

4.1. Feature extraction

4.1.1. Tolerance to aberrations of the feature extractors

Observing how people acquire (scan) documents, the physical state of the documents and the scanners themselves we have identified some input image issues that can compromise the feature extraction:

- Low image contrast
- bended document
- lousy condition document (scratches and blemishes)
- non-uniform illumination
- light flares (internal lens reflections)
- gaussian noise (blurred document, not in focus)
- laplacian noise
- poisson noise
- salt-and-pepper noise
- linear motion (extract document from scanner before the scan is completed)
- pixelation
- lens refraction (wide angle lens)
- rotation (the document is not straight on the scanner)
- partially occluded document

Listing 3 Aberrate an image

```
convert $1 -blur 2 gaussian2x2.jpg
convert $1 -blur 5 gaussian5x5.jpg
convert $1 +contrast +contrast +contrast +contrast +contrast contrast.jpg
convert $1 +noise laplacian +noise laplacian laplacian.jpg
convert $1 +noise multiplicative +noise multiplicative multiplicative.jpg
convert $1 +noise poisson +noise poisson  poisson.jpg
convert $1 -motion-blur 5x5+145 motion-linear-5x145.jpg
```

It is possible to artificially create and automate some of these aberrations with libraries like imagemagik[20] and observe how to affect to the extractors. On listing 3 we have an example.

Each of the extractors has strengths and weaknesses regarding image aberrations. We have prepared set of tests for ORB, BRISK, and AKAZE for the **specific domain of document identification**. We present here *some* of our results.

4.1.2. Test image

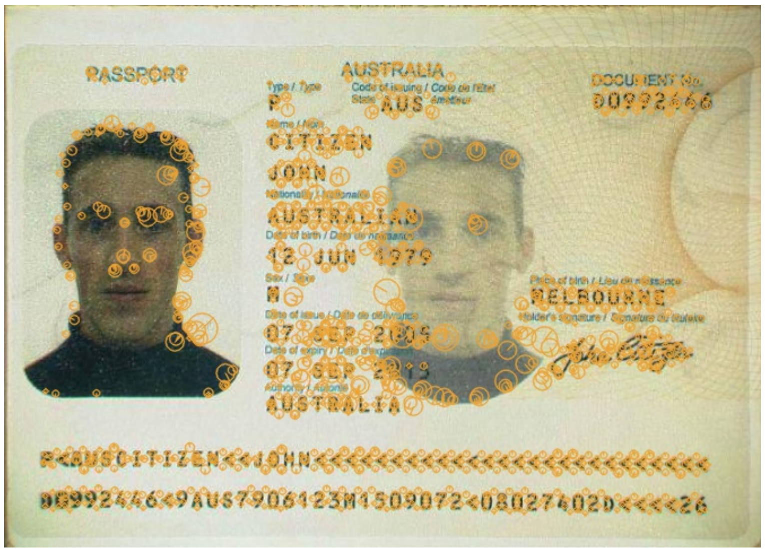


Figure 21. Test image

Fig. 21 is our test image with akaze key points, we can see that all relevant features are detected.

4.1.3. Bended document

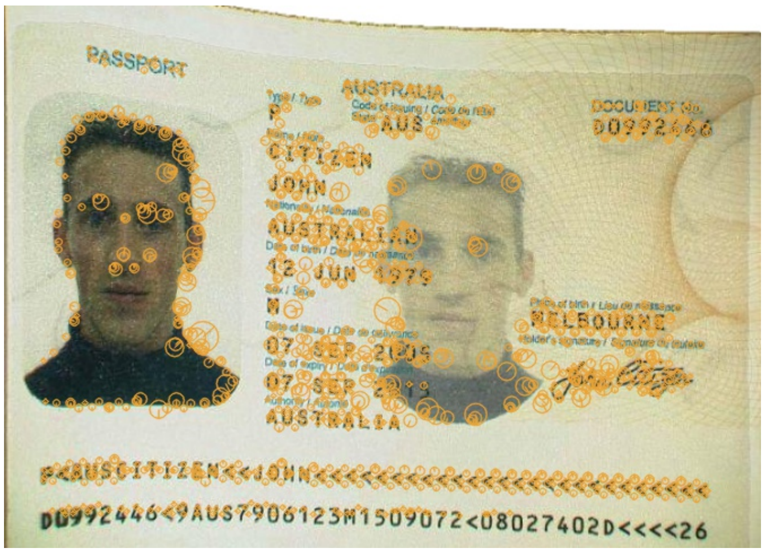


Figure 22. Bend

Bend does not have a substantial effect because the method has some invariance to rotation, but we lose points in the bottom right area.

4.1.4. Flares and gaussian blur

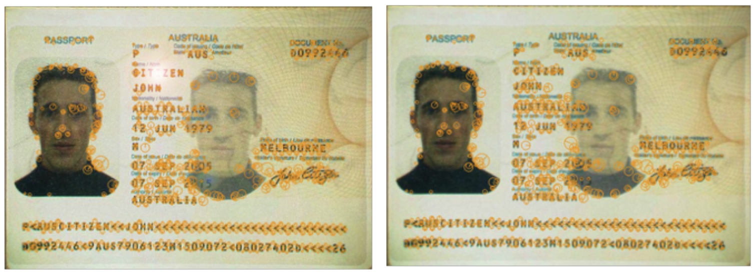


Figure 23. AKAZE. Flares and gaussian2x2

The flares and gaussian blur2x2 doesn't affect the extraction but in the title area for AKAZE.

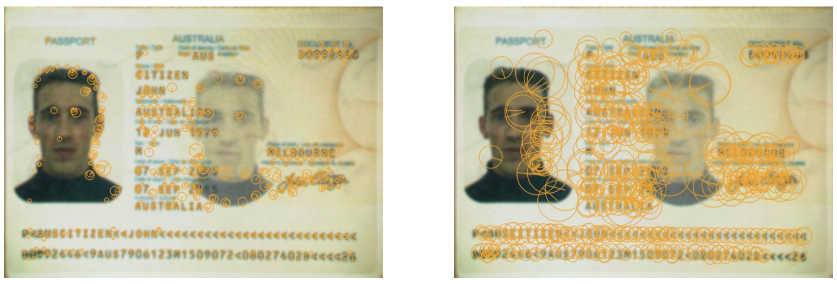
We incresase the gaussian kernel to 5x5 and compare akaze with brisk on fig.24 we see that the blurred image retain some good points yet on both, but AKAZE performs slightly better.

This type of aberration can come from hand held cameras like webcams or smartphones but is rare in table scanners.

4.1.5. Motion

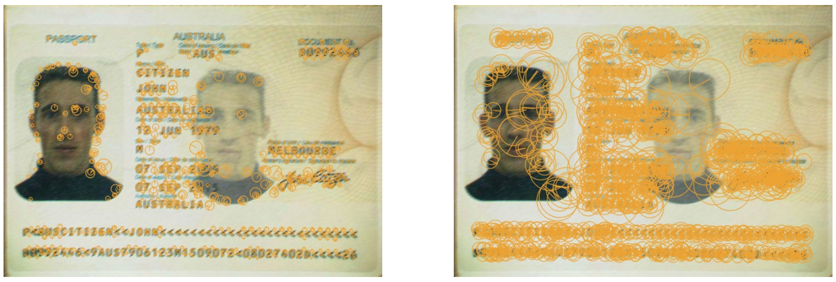
When we extract the document from the scanner before the scan is terminated, it can appear a distortion named motion (linear).

Comparing akaze and brisk (fig. 25), we can see that the second can be more resistant to motion.



(a) akaze **(b)** brisk

Figure 24. AKAZE and BRISK. Blured versions with gaussian kernel 5x5



(a) akaze **(b)** brisk

Figure 25. Tolerance to motion

4.1.6. Low contrast

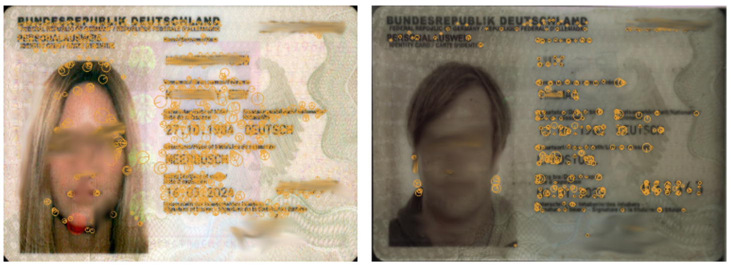
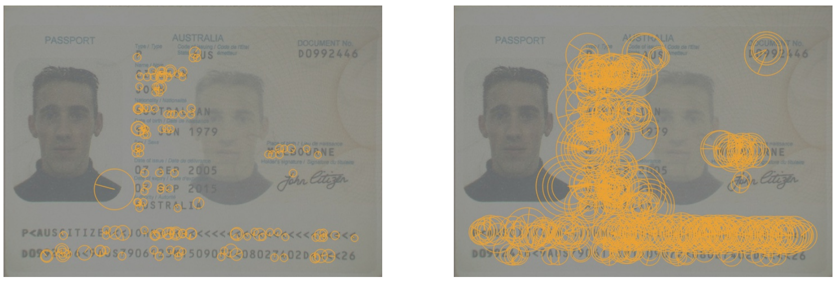


Figure 26. Effects of low contrast in feature extractors: we lost several key points.

On fig. 26, notice that in the regular focused left image we have a high number of points spread over all the document. On the low contrasted picture of the right, there are fewer points and quite anyone on the title of the material.Expresely comparing BRISK and ORB on the same low-contrast image like fig.27 we found that orb performs better but the result is barely usable to perform matching.

Going to an extreme, in the example given on fig. 28, we have reduced the contrast 90 times. Only 3 points are recognized. Although it is possible to enhance the histogram of a low contrasted image with an operation named *equalisation* thre results are not much better due to the low dynamic range of the image.



(a) brisk **(b)** orb

Figure 27. Effects of low contrast on BRISK and ORB

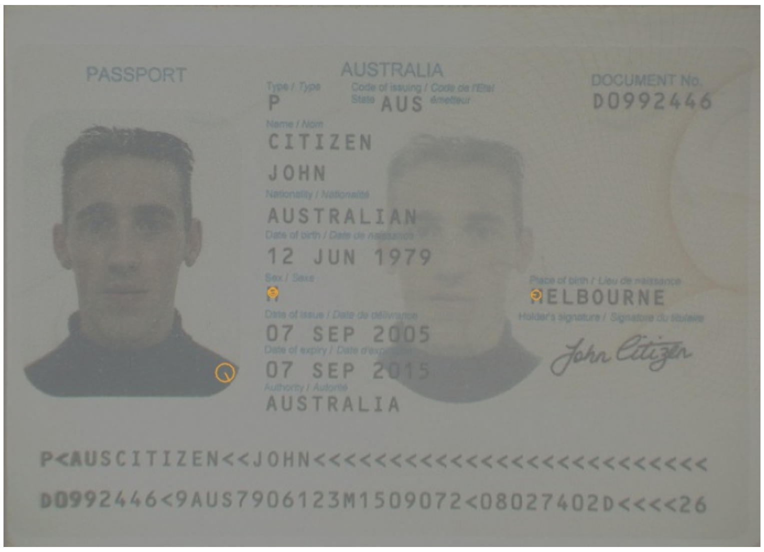

PASSPORT
AUSTRALIA
Type / Type
P
Code of issuing / Code de l'Etat
State émetteur
AUS
DOCUMENT No.
D0992446
Name / Nom
CITIZEN
JOHN
Nationality / Nationalité
AUSTRALIAN
Date of birth / Date de naissance
12 JUN 1979
Sex / Sexe
Place of birth / Lieu de naissance
MELBOURNE
Date of issue / Date de délivrance
07 SEP 2005
Holder's signature / Signature du titulaire
Date of expiry / Date d'expiration
07 SEP 2015
Authority / Autorité
AUSTRALIA
P<AUSCITIZEN<<JOHN<<<<<<<<<<<<<<<<<<<<<<<<<<
D0992446<9AUS7906123M1509072<08027402D<<<<26

Figure 28. AKAZE with contrast reduced 90 times.

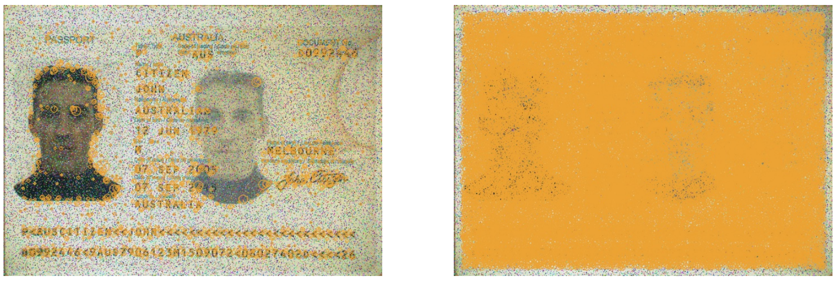

(a) akaze **(b)** brisk

Figure 29. Effects of noise

4.1.7. Salt-and-peeper noise

Regarding this type of noise, AKAZE is quite tolerant, but brisk becomes **unusable** when it is present because the noise points are recognized as key points. See fig. 29.

4.1.8. Relevancy

Perhaps the most important characteristic. On fig. 30, we compare AKAZE and ORB to visually decide which will give us better results when matching documents with an index. In other words, which of them will be able to discern between two models of similar documents. Note that AKAZE detects points spread over all the document and so it is more suitable to compare the document with an index.

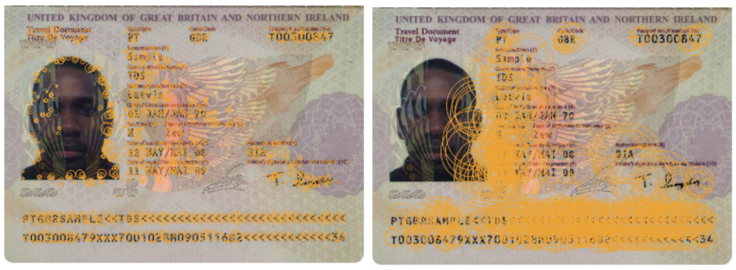


Figure 30. Example of feature extraction: AKAZE on the left and ORB on the right.

4.1.9. Speed

He have timed and averaged the extraction of 4000 document images. The original data can be found on appendix A.

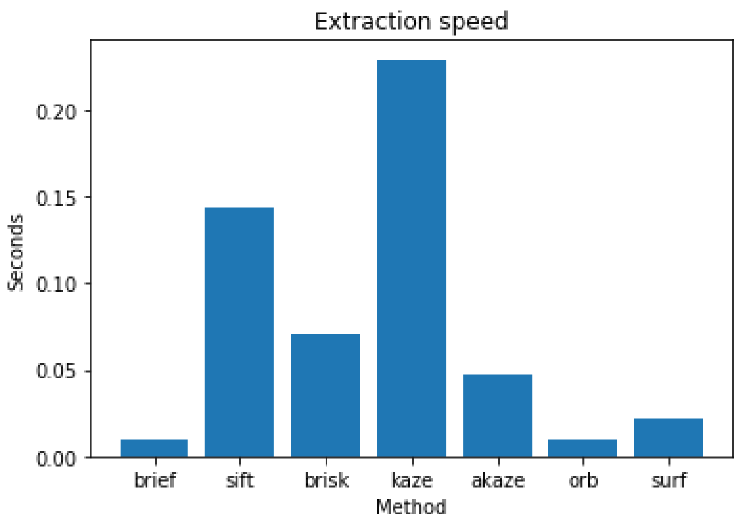


Figure 31. Extraction speed. Average of 4000 images.

BRIEF and ORB have the best results while BRISK and AKAZE have the worst. We must take in account that BRISK and AKAZE extract a total of 4836 and 1317 points what contrasts with 462 and 500 of the two first.

4.1.10. Conclusion about feature extractors

Our very early tests showed that HOG+Kmeans is not very precise (around 60%) because HOG has difficulties with very similar images like distinct versions of the same documents. The need to guess the final number of clusters for kmeans is a headache as well. For that reason we started our investagations about feature extractors and kNN.

We have tested SIFT, SURF, ORB, BRIEF, BRISK and AKAZE feature extractors under different conditions. We have considered the absolute number of key points detected, their variance and their relevance in order to recognise a document. The test were made over the documents AUS-AD-02001, BGR-JO-09001, FIN-AO-06001 and FRA-CO-02001. On the **appendix A** there are the results of AUS-AD-02001, for each method column we have the three first detected models ordered by its probabilities. The probabilty is calculated as the number of matching key points of the two images over the total key points of the index image.

On the table 1 we have summarized some of the results. Note that, in addition to the absolute numerical information we have taken into account the **quality** of the key points distribution over the image features that can best separate two similar document models.

	AKAZE	BRISK	ORB
Bend	Good	Good	Good
Flares	Good	Good	Good
Gaussian 2x2	Good	Good	Bad
Gaussian 5x5	Good	Good	Bad
Motion	Bad	Good	Good
Low contrast	Bad	Good	Bad
Noise	Good	Bad	Good
Speed	Bad	Bad	Good
Relevancy	Very good	Good	Bad

Table 1. Comparison of feature extractors for identity documents

We say an extractor is better than other concerning matching reliability with our category database.

AKAZE gives richer key points on the faces, id numbers and texts while ORB seems to focus on the center area of the document.

Facing low contrast, we can see that BRISK performs slightly better than AKAZE and ORB is definitively more resistant. Unfortunately, we extract features to categorize the document, and this ORB-point distribution is familiar to many models of passports and this fact makes it useless for our needs. This explains why the methods that extract an absolute number of points bigger than others are not necessarily better.

AKAZE performs good on all tests but fails on low-contrast images. BRISK performs better on low-contrast but **poorly** on noisy images.

In our tests, we have found that **AKAZE** surpasses other extractors in the task to obtain features from id documents.

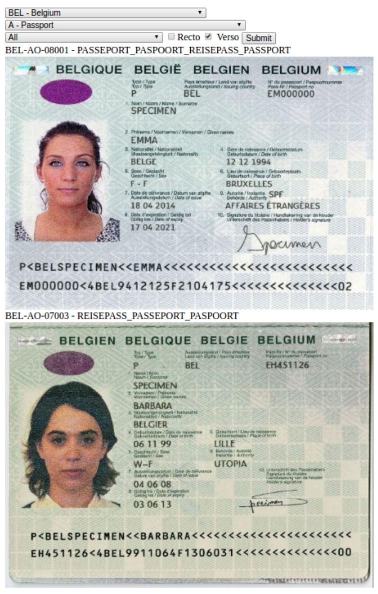
As **the quality** of the identification is our main issue we conclude that

> AKAZE seems to be the best general extractor method to work with id documents.

4.2. Utility web applications

We need some help to handle the target better than the raw files and a database with tags. We can write a small python script that generates dynamic web pages.

To navigate the PRADO database, we wrote one, and we called it **prado-explorer**. See fig. 32.



BEL - Belgium
A - Passport
All
Recto ✓ Verso Submit
BEL-AO-08001 - PASSEPORT_PASPOORT_REISEPASS_PASSPORT

P<BELSPECIMEN<<EMMA<<<<<<<<<<<<<<<<<<<<<<<<<
EM000000<4BEL9412125F2104175<<<<<<<<<<<<<<02

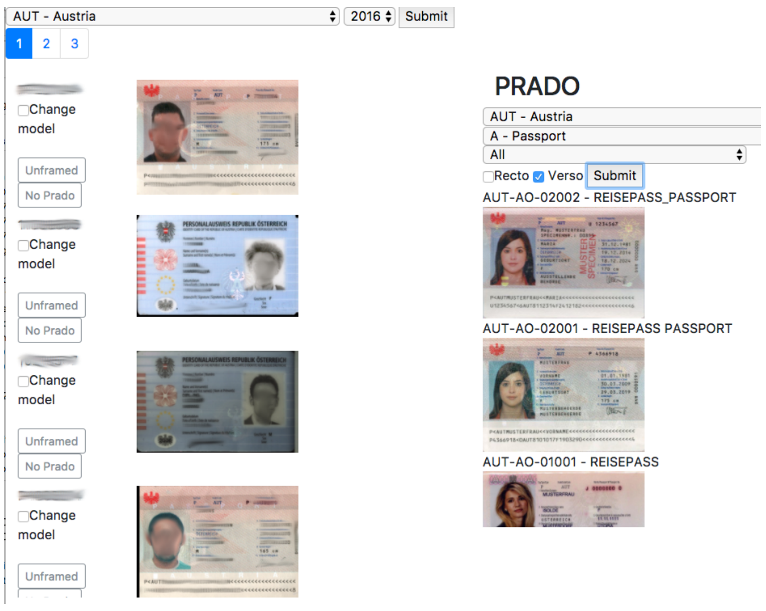
BEL-AO-07003 - REISEPASS_PASSEPORT_PASPOORT

P<BELSPECIMEN<<BARBARA<<<<<<<<<<<<<<<<<<<<<<
EH451126<4BEL9911064F1306031<<<<<<<<<<<<<<00

BELGIQUE BELGIË BELGIEN BELGIUM
SPECIMEN
EMMA
BELGIEN BELGIQUE BELGIE BELGIUM
SPECIMEN
BARBARA

Figure 32. PRADO explorer web server

The task of visually inspecting and correcting the training dataset can be tedious but it is not difficult to program an small web server that links to the database and allows us to visualize and correct the training set.



AUT - Austria 2016 Submit

PRADO
AUT - Austria
A - Passport
All
Recto ✓ Verso Submit
AUT-AO-02002 - REISEPASS_PASSPORT
AUT-AO-02001 - REISEPASS PASSPORT
AUT-AO-01001 - REISEPASS

Figure 33. The labeler webapp

The **labeler** (see fig. 33) web app connects the SQL document database and the images. On the right, we can see the PRADO explorer. If we select the ”Change model” checkbox and press the right Prado image, the model is saved for the picture. All are made with ajax calls to avoid rebuilding the whole page on each document change.

4.3. Preprocessing the images

Each image that enters the system must be preprocessed. Moreover, before make inference of new images they must be preprocessed in the same way that the training images in order to obtain consistent results.

Before process any image we will be sure that:

- it has the same resolution and color channels
- it has the same size
- it is straight

We cannot be sure that the input image is straight. Quite the contrary, it can be rotated and shifted. Also, note the use of the OpenCV library.

4.3.1. The rectification pipeline

Basically, once detected the contour of the document, we calculate the slope θ of the border and counter rotate the whole document $atan(\theta)$. We also need to add a margin before the rotation because not to overlap the rotated document.

Algorithm 6 List of sequential operations to rectify an incoming image

```
Resize
Add a border to avoid cut an area if the document is somewhat rotated
Binarize
Determine the contour (area of maximum luminance change)
Deskew
Rotate
Translate
Crop
```

The author entirely creates the function `rectify` that accepts an image and returns an straight and standard resized image. It can be found on the **appendix B**.

We believe that, although the function is a bit large, it is not difficult to follow. For the interested readers, it is worth to have the OpenCV reference at hand and follow the steps sequentially.

4.4. Downlading and formatting PRADO

As PRADO uses ISO-3 codes[15] on the URL, it would suffice to get the pages for all countries usign this key. A few lines of javascript or python can do the job as in **appendix C**.

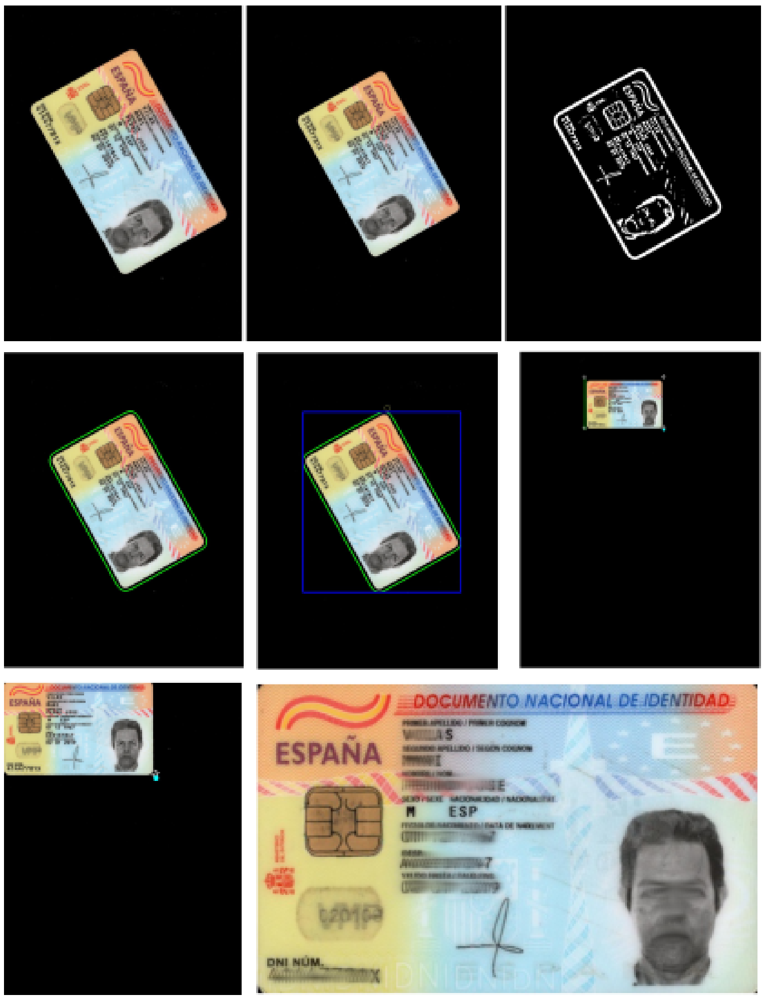


Figure 34. Pipeline to rectify an image

With the HTML code downloaded, we may find the URL of the image of interest `div.doc-thumbnails a img` and the document model. The function `format_doc` on lin. 42 creates the SQL code to insert the document into the database.

4.5. Extracting and matching features of the images

One relevant characteristic of our extractor implementation is that it is optimized to work in a parallel way. We use the multiprocessing python module because, in our tests, the overall performance is better than using one process with multithreaded jobs. The main loop looks like in listing 4.

Listing 4 Feature extractor main loop

```
1  jobs = []
2  j_count = 0
3  for image_fn in files:
4      p = mp.Process(target=spawn,
5          args=( ms, image_fn,
6                 args.debug, args.dump))
7      jobs.append(p)
8      p.start()
9      j_count += 1
10     if j_count \% MAX_THREADS == 0:
11         for j in jobs:
12             j.join()
13
14 print('Terminating computations')
15 for j in jobs:
16     j.join()
```

We can see the jobs list, in line 4 we create the processes and append them to the jobs, finally we join them in line 10 to wait until the last execution.

The spanw itself spawns processors of one or several of this types of feature extractors. Note that the OpenCV library has moved the copyrighted processors to `cv2.xfeatures2d` while the free remains on `cv2`.

We have extracted the key points of the PRADO index images and grouped them in a single pickle file. It is faster than load each image keypoint file because it is loaded only one time on memory. Otherwise we would need $num\_query\_images \times index\_size$ file loadings.

Regarding the feature matching process, we have used a Pool to handle the jobs. On listing 5 line 6 `pool.map` maps the jobs to the function `load_and_match` that creates the matchers for the wanted methods and calls to match. Note that we cannot reuse the matcher because they store internal state variables that would be overridden by other processes.

4.5.1. Matching with the index

The function match it is a bit long, but we know it is important enough in the project to completely describe it.

Listing 5 Pool of matchers

```
pool = mp.Pool(NUM_CORES)
print('Final number of image matches\
       to calculate: {}'.format(len(jobs)))
print('Identification process started. Please wait...')
ts = get_time()
pool.map(load_and_match, jobs)
pool.close()
pool.join()
conn.close()
print(print_time('Identification process ended', ts))
```

Remember that the trained image is the comparison image (a picture of the category index) while the query image is the image that we are evaluating.

The complete python implementation of the funtcion `match` is found on appendix D. All of the concepts have been explained before. On listing 20, line 26 we define the ratio distance between two points to be considered neighbors and the minimum number of matches to consider that the two images can be of same document. In line 41 we perform the matching using the k-nearest neighbor's algorithm. In line 65 we try to find the homography between the index and the input (this allows to match a rotated and shifted image). Finally, in line 68 we take account of the matching points over the total points. That gives us the probability that the two images belong to the same document.

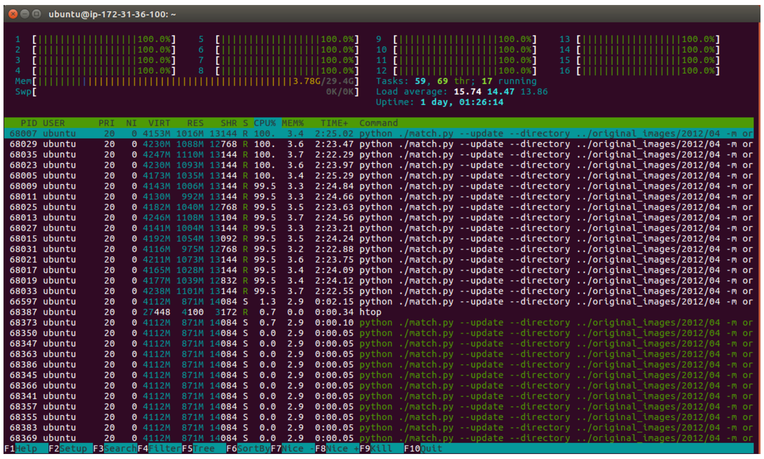
Listing 6 A matching run

```
(venv)myMac:scan-id-documents pvilas$ ./match.py
-m akaze
--directory  /Users/pvilas/Downloads/images/03
--database docs/docs2.db
--update
Loading features
Points of 2317 trained documents loaded in 1.946526102s
System with 4 cores
Feature processor/s: ['akaze']
Number of query images: 671
Number of featured trained images: 2317
Max number of matches to calculate (without country filter): 1554707
Preparing task list and processing. Please wait...
Final number of image matches to calculate: 32822
Identification process started. Please wait...
Identification process ended in 138.527946s
Preparing summary. Please wait...
Summary: 671 files, 671 results
Updating database with 671 results
Update terminated
Process terminated   in 184.059734s
```

On listing 6 we give a run on a Mac of 2014 (i5, 4 cores, 8GB), AKAZE only. The PRADO index is 2.317 in size and takes almost $2s$ to load. Note that in this example we have 671 query images, that is a total of $671 \times 2.317 = 1.554.707$ image comparaisons!. Fortunately, we can apply a country filter (see section 3. Design) and the image comparaisons down to 32.822. As a regular AKAZE description of a document has 1.188 points, the total points compared by the function in this run is $32.822 \times 1.188 = 38.992.536$. With four cores we have a matching speed of 3.6 images/sec.

We have rented a 16-core (c4.4xlarge) Intel Xeon machine with Ubuntu on Amazon AWS (see fig. 35). After many experimentations on the workstation, we finally optimized a process to parallelize the extraction. The cost of this machine in these moments (Nov-

2017) with 200Gb of SSH disk is 0.9$/hour but we reached 16.2 images/sec.



```
ubuntu@ip-172-31-36-100: ~

  1  [||||||||||||||||||100.0%]    5  [||||||||||||||||||100.0%]    9  [||||||||||||||||||100.0%]    13 [||||||||||||||||||100.0%]
  2  [||||||||||||||||||100.0%]    6  [||||||||||||||||||100.0%]    10 [||||||||||||||||||100.0%]    14 [||||||||||||||||||100.0%]
  3  [||||||||||||||||||100.0%]    7  [||||||||||||||||||100.0%]    11 [||||||||||||||||||100.0%]    15 [||||||||||||||||||100.0%]
  4  [||||||||||||||||||100.0%]    8  [||||||||||||||||||100.0%]    12 [||||||||||||||||||100.0%]    16 [||||||||||||||||||100.0%]
  Mem[||||||||||||||||||||||||||||||||||||||||||||3.78G/29.4G]    Tasks: 59, 69 thr; 17 running
  Swp[                                               0K/0K]    Load average: 15.74 14.47 13.86
                                                               Uptime: 1 day, 01:26:14

  PID USER      PRI  NI  VIRT   RES   SHR S CPU% MEM%   TIME+  Command
68007 ubuntu     20   0 4153M 1016M 13144 R 100.  3.4  2:25.02 python ./match.py --update --directory ../original_images/2012/04 -m or
68029 ubuntu     20   0 4230M 1088M 12768 R 100.  3.6  2:23.47 python ./match.py --update --directory ../original_images/2012/04 -m or
68035 ubuntu     20   0 4247M 1110M 13144 R 100.  3.7  2:22.29 python ./match.py --update --directory ../original_images/2012/04 -m or
68023 ubuntu     20   0 4230M 1093M 13144 R 100.  3.6  2:23.97 python ./match.py --update --directory ../original_images/2012/04 -m or
68005 ubuntu     20   0 4173M 1035M 13144 R 100.  3.4  2:25.29 python ./match.py --update --directory ../original_images/2012/04 -m or
68009 ubuntu     20   0 4143M 1006M 13144 R 99.5  3.3  2:24.84 python ./match.py --update --directory ../original_images/2012/04 -m or
68011 ubuntu     20   0 4130M  992M 13144 R 99.5  3.3  2:24.66 python ./match.py --update --directory ../original_images/2012/04 -m or
68025 ubuntu     20   0 4182M 1040M 12768 R 99.5  3.5  2:23.63 python ./match.py --update --directory ../original_images/2012/04 -m or
68013 ubuntu     20   0 4246M 1108M 13104 R 99.5  3.7  2:24.56 python ./match.py --update --directory ../original_images/2012/04 -m or
68027 ubuntu     20   0 4141M 1004M 13144 R 99.5  3.3  2:23.21 python ./match.py --update --directory ../original_images/2012/04 -m or
68015 ubuntu     20   0 4192M 1054M 13092 R 99.5  3.5  2:24.24 python ./match.py --update --directory ../original_images/2012/04 -m or
68031 ubuntu     20   0 4116M  975M 12768 R 99.5  3.2  2:22.88 python ./match.py --update --directory ../original_images/2012/04 -m or
68021 ubuntu     20   0 4211M 1073M 13144 R 99.5  3.6  2:23.75 python ./match.py --update --directory ../original_images/2012/04 -m or
68017 ubuntu     20   0 4165M 1028M 13144 R 99.5  3.4  2:24.09 python ./match.py --update --directory ../original_images/2012/04 -m or
68019 ubuntu     20   0 4177M 1039M 12832 R 99.5  3.4  2:24.12 python ./match.py --update --directory ../original_images/2012/04 -m or
68033 ubuntu     20   0 4238M 1101M 13144 R 99.5  3.7  2:22.55 python ./match.py --update --directory ../original_images/2012/04 -m or
66597 ubuntu     20   0 4112M  871M 14084 S  1.3  2.9  0:02.15 python ./match.py --update --directory ../original_images/2012/04 -m or
68387 ubuntu     20   0 27448  4100  3172 R  0.7  0.0  0:00.34 htop
68373 ubuntu     20   0 4112M  871M 14084 S  0.7  2.9  0:00.10 python ./match.py --update --directory ../original_images/2012/04 -m or
68350 ubuntu     20   0 4112M  871M 14084 S  0.0  2.9  0:00.05 python ./match.py --update --directory ../original_images/2012/04 -m or
68347 ubuntu     20   0 4112M  871M 14084 S  0.0  2.9  0:00.05 python ./match.py --update --directory ../original_images/2012/04 -m or
68363 ubuntu     20   0 4112M  871M 14084 S  0.0  2.9  0:00.05 python ./match.py --update --directory ../original_images/2012/04 -m or
68386 ubuntu     20   0 4112M  871M 14084 S  0.0  2.9  0:00.05 python ./match.py --update --directory ../original_images/2012/04 -m or
68345 ubuntu     20   0 4112M  871M 14084 S  0.0  2.9  0:00.05 python ./match.py --update --directory ../original_images/2012/04 -m or
68366 ubuntu     20   0 4112M  871M 14084 S  0.0  2.9  0:00.05 python ./match.py --update --directory ../original_images/2012/04 -m or
68341 ubuntu     20   0 4112M  871M 14084 S  0.0  2.9  0:00.05 python ./match.py --update --directory ../original_images/2012/04 -m or
68357 ubuntu     20   0 4112M  871M 14084 S  0.0  2.9  0:00.05 python ./match.py --update --directory ../original_images/2012/04 -m or
68355 ubuntu     20   0 4112M  871M 14084 S  0.0  2.9  0:00.05 python ./match.py --update --directory ../original_images/2012/04 -m or
68383 ubuntu     20   0 4112M  871M 14084 S  0.0  2.9  0:00.05 python ./match.py --update --directory ../original_images/2012/04 -m or
68369 ubuntu     20   0 4112M  871M 14084 S  0.0  2.9  0:00.05 python ./match.py --update --directory ../original_images/2012/04 -m or
F1Help  F2Setup F3SearchF4FilterF5Tree  F6SortByF7Nice -F8Nice +F9Kill  F10Quit
```

Figure 35. A beautiful image: 16 cores extracting features in parallel.

Now we have our dataset categorized. We will organize it slightly different for the next step.

4.6. Preparing the data for the network

Handling high volume of images is not easy, but most AI libraries have utilities to process the data in batches and feed the network. In the case of `keras` it implements a python generator interface that returns bunches from directories. Each directory represents a category, and all the images in this directory are supposed to belong to this category.

As we have explained later, we must divide the dataset into three parts. We need to convert a dataset organized by `(image, category)` to another `(partition, category, image)`. See listing 7.

Listing 7 Preparing the dataset

```python
# list_doc has the list of tuples (image, category)
# labels has the list of labels

num_labels = len(labels)
num_train_images = round(numdocs * PART_TRAIN_IMAGES) # 80\%
num_val_images = round(numdocs * PART_VAL_IMAGES) # 20\%
num_test_images = round(numdocs * PART_TEST_IMAGES) # 20\%

# shuffle documents
random.shuffle(list_doc)

# make partition
pre_train_images = list_doc[0:num_train_images]
pre_val_images = list_doc[num_train_images:num_train_images + num_val_images]
pre_test_images = list_doc[num_train_images + num_val_images:]

# copy images into its directories train, validation, test...
```

We must make sure that each directory (train, validation, test) have the same categories. We have created some helper functions like in listing 8. It is easy in python to calculate the intersection of two sets using list comprenhensions like in line 3. Being `cat_train, cat_val, cat_test` the categories of each partition, the function categories_are_not_disjoint must be true for each tuple `(cat_train, cat_val)`, `(cat_train, cat_test)`, `(cat_val, cat_test)`.

Listing 8 Checking categories are disjoint

```
def returnNotMatches(a, b):
    """ return categories of two sets that do not intersect """
    return [[x for x in a if x not in b], [x for x in b if x not in a]] \label{intersection}

def categories_are_not_disjoint(a, b):
    """ return True if any element of a is not in b
        or any element of b are not in a
    """
    dis = returnNotMatches(a, b)
    for d in dis:
        if len(d) > 0:
            print('{}'.format(dis))
            return True
    return False
```

The network training works best if we first normalize the data like in listing 9. In the case of the color images, we only rescale the tensor and convert it to float32 by dividing each element by 255. It is not necessary to shift the data to mean zero. The preprocessing.image.ImageDataGenerator can do the escalation of each image before passing it to the trainer. Note that we define the batch size on line 10. This is the number of images that will go in one batch and it will depend of each system but should not exceed the amount of the GPU memory.

Listing 9 Normalizing images and creating generators

```
1   from keras.preprocessing.image import ImageDataGenerator
2
3   # All images will be rescaled by 1./255
4   train_datagen = ImageDataGenerator(rescale=1. / 255)
5   test_datagen = ImageDataGenerator(rescale=1. / 255)
6
7   train_generator = train_datagen.flow_from_directory(
8       train_dir,
9       target_size=RESIZE_TO,
10      batch_size=BATCH_SIZE,
11      class_mode='categorical')
12  validation_generator = ...
```

4.7. Convolutional Neural Network

The implementation of our model is straightforward with keras. On lst.10 we create an instance of models.Sequential and then we add the layers discussed on the design section, model summary (see listing 2).

Listing 10 CNN model to classify identity documents

```python
# create a convolution layer
def conv(num_filters):
    return layers.Conv2D(num_filters, KERNEL_SIZE, activation='relu')

# craete a maxpooling layer
def maxp():
    return layers.MaxPooling2D((2, 2))

# create model
model = models.Sequential()
model.add(layers.Conv2D(64, (3, 3),
                        activation='relu',
                        input_shape=(IMAGE_WIDTH,
                                     IMAGE_HEIGHT,
                                     3
                                     )))
model.add(conv(64))
model.add(maxp())

model.add(conv(64))
model.add(maxp())

model.add(conv(64))
model.add(maxp())

model.add(layers.Flatten())
model.add(layers.Dense(512, activation='relu'))
model.add(layers.Dropout(0.5))
model.add(layers.Dense(train_generator.num_classes,
                       activation='softmax'))
```

Once the model is defined and compiled, we start the network training like in listing 11. If we are running on AWS, we must be aware of not to waste time because it is almost 1 euro/hour!.

We may start with a low number of epochs and if the network performs well we can increase it while does not overfitting. In our case, it will converge from the very first epochs.

Listing 11 Training the CNN

```python
1  history = model.fit_generator(
2      train_generator,
3      steps_per_epoch=train_generator.samples//BATCH_SIZE,
4      epochs=EPOCHS,
5      validation_data=validation_generator,
6      validation_steps=validation_generator.samples//BATCH_SIZE)
7
8  model.save('40epochs.h5')
```

Observe the object `history` in line 1. After the training, we can use it to plot the loss and the accuracy over the epochs like in fig. 36 where we can observe that the model slightly overfit from the twelfth epoch. It is not worth to train the network more than this.

Finally, we test the accuracy of our model with images that have never seen by the network in the train nor the validation steps.

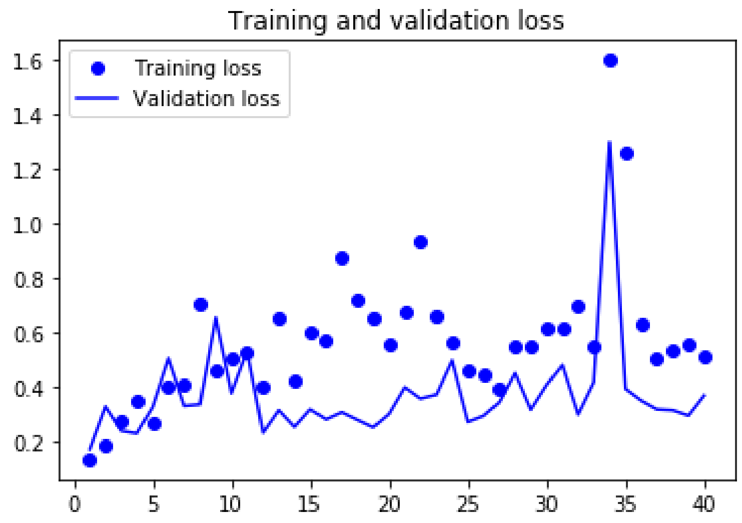


Figure 36. Train and validation loss

Listing 12 Testing the CNN

```
test_loss, test_acc = model.evaluate_generator(test_generator)
print("Model accuracy: {}".format(test_acc))
```

```
Model accuracy: 98%
```

On listing 12, the function evaluate_generator takes the test partition and averages the accuracy of the predictions. Our model accuracy after 12 epochs is 98%.

4.8. Making predictions

Making inference is very straightforward. We load the trained model and use the predict_classes and the predict_proba to get the prediction and its probability. Note that we preprocess the image and normalize the tensor image before making the prediction.

Listing 13 Making predictions

```
prediction = model.predict_classes(img_tensor, batch_size=1)[0]
proba=np.max(model.predict_proba(img_tensor)[0])

# get a list of category names
category_names=list(train_generator.class_indices.keys())

print("The document seems to be a {} with a probability of {}".format(
    category_names[prediction], proba))
```

```
The document seems to be a ESP-BO-03001
with a probability of 0.974
```

It is worth to comment that the inferences can be made with very little computing power like any smartphone what profoundly contrasts with the training phase.

4.9. What is the network seeing?

We can visualize the intermediate convolution outputs to understand how the network is generalizing an image until finally classify it.

To do this, we input an arbitrary image, and we plot the output of the successive layers. These feature maps have three dimensions because each color channel has its map.

It is important to note that the following ideas are an adaptation from F. Chollet[10].

Let's load and display an image on fig. 37.

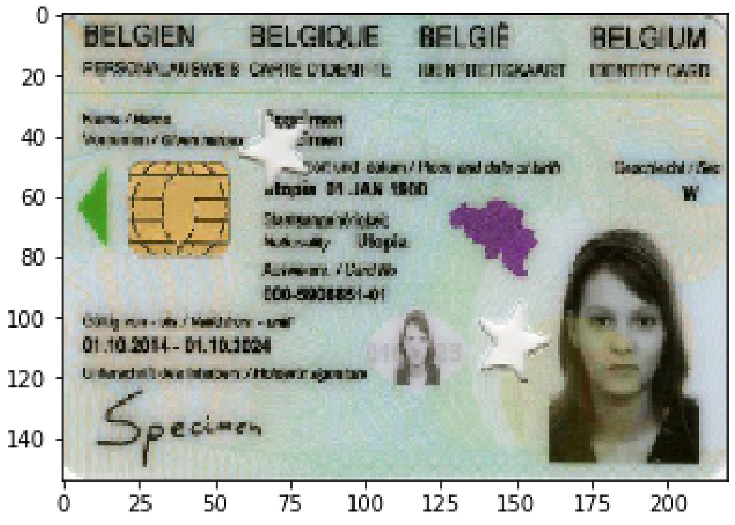


Figure 37. A test image from PRADO

Now, on lst. 14 we check the prediction

Listing 14 Check the prediction

```
# category index with max probability
prediction = model.predict_classes(img_tensor, batch_size=1)[0]
proba=np.max(model.predict_proba(img_tensor))
```

```
Document category is BEL-BO-07001 with a probability of 0.99
BEL-BO-07001: IDENTITEITSKAART_CARTE DIDENTITE
PERSONALAUSWEIS_IDENTITY CARD
Belgium Identity Card ordinary document
```

Note the pixelation due to downsample the image, typical on image classifiers as high-resolution images tend to confuse the network.

We now take the layers as outputs and create a model that returns these outputs to a given input (see lst:15). In line 2 we create a list for the layers and in line 4 we assign the output of the model to these layers. Next, we feed the model with the image. It will return one

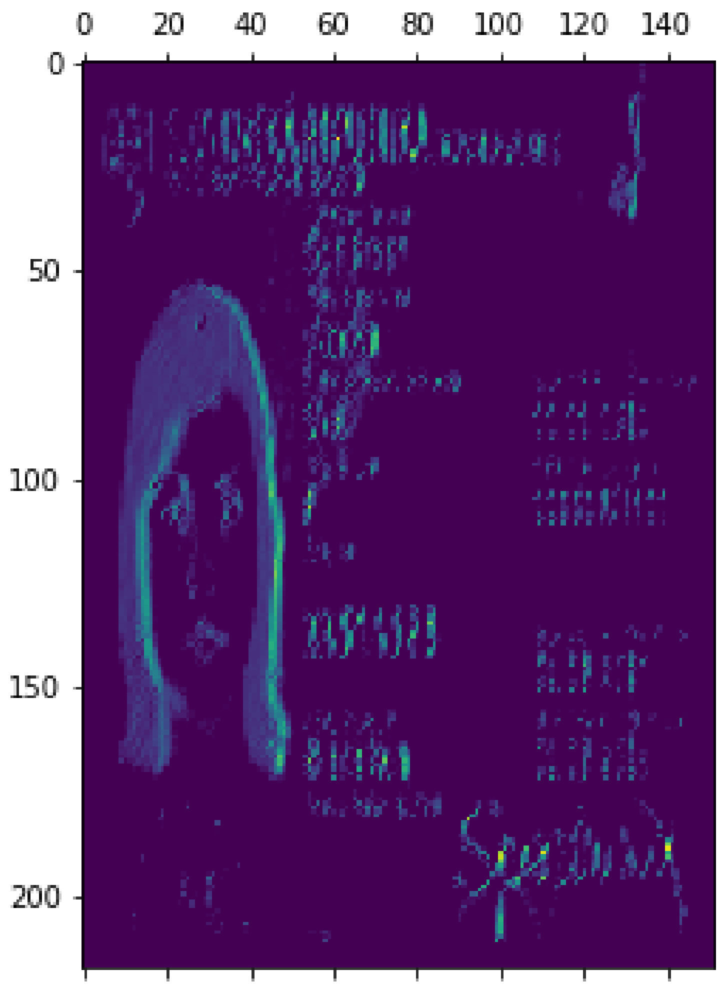


Figure 38. Response of the 17th filter of the first layer

Listing 15 Creating an activation model

```
1  # extract the outputs of the first layers
2  layer_outputs = [layer.output for layer in model.layers[:8]]
3  # creates a model that return these outputs given the model input
4  activation_model = models.Model(inputs=model.input, outputs=layer_outputs)
5  # feed model with an image
6  activations = activation_model.predict(img_tensor)
7  # take 17th filter of the first layer
8  fl=activations[0]
9  plt.matshow(fl[0, :, :, 16], cmap='viridis')
```

array per layer activation, i.e. the same shapes than the model.summary. Each of these layers has 64 filters, in line 9 we examine the 17th of the first layer.

The plot of the activated filter is on fig. 38. We can appreciate signal on the borders of the face and some prominent characters. This filter responds to some rounded-border, diagonal-like feature on the image. Each filter of the bank will respond to specific features on the image.

We can group the responses of the layers and plot them in one single map, see appendix E, listing 21 and the figure 39 where the conv1, conv2, and maxpool1 output layers are plotted. We can observe how the filters extract several features of the input image: some respond to high frequencies while others react to rounded diagonal features. Those that respond to the background of the document are especially noteworthy as it is not readily disguisable in the original image.

The black responses simply have not output signal for its activation pattern, i.e., the pattern is not found on the input image.

Note how the max_pooling ”summarizes” the output while reducing the ”complexity” grouping the most destacable features of each filter.

The response to last three layers is on fig. 40. Note how the response is more and more generalized until the final pool layer. We are making more **abstract** representations as **we deep** into layers.

On the training phase, the network will adjust their weights to recognize these activation maps as a category BEL-BO-07001 Belgium Identity Card.

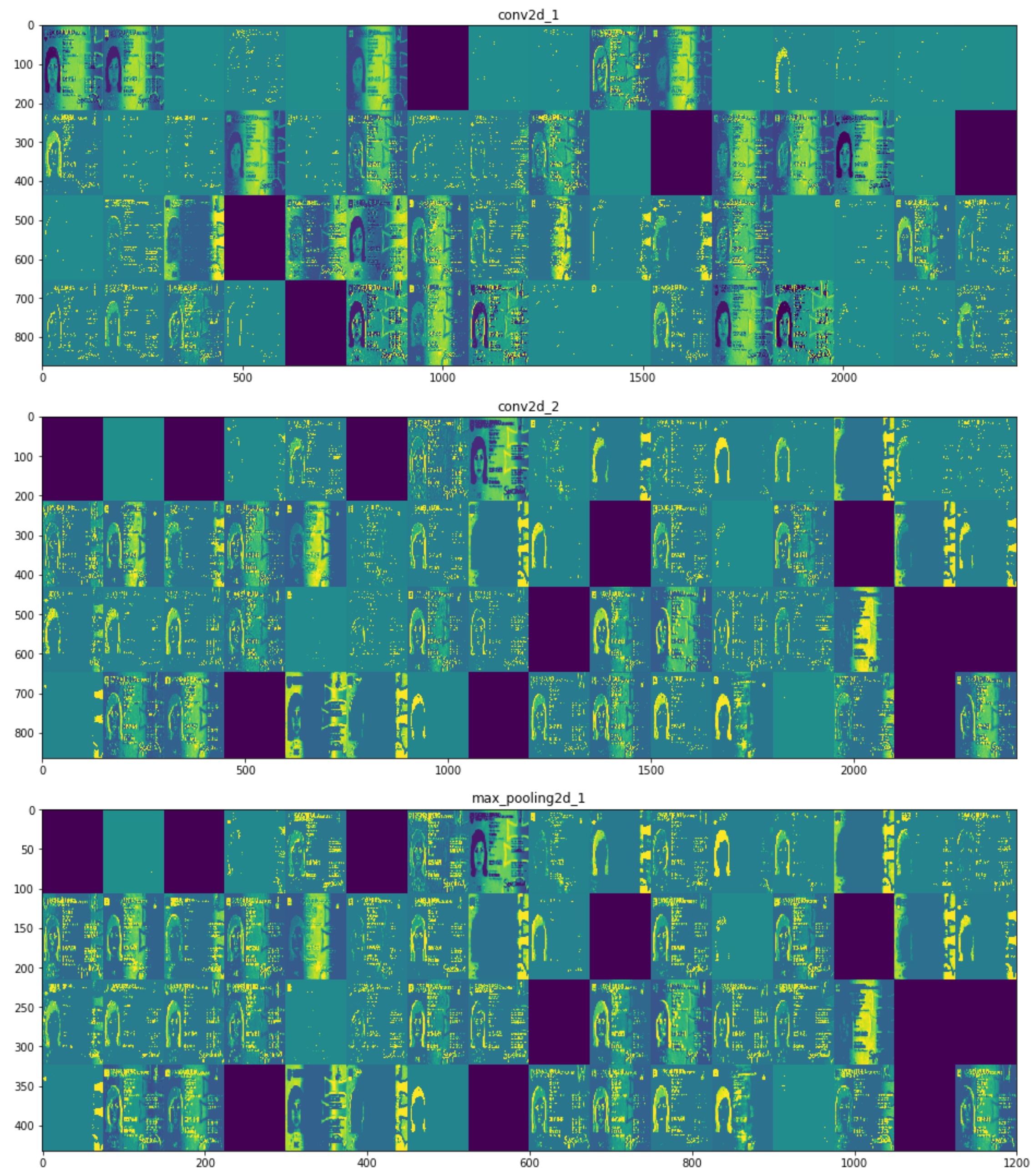


Figure 39. Responses of conv1, conv2 and maxpool1 output layers

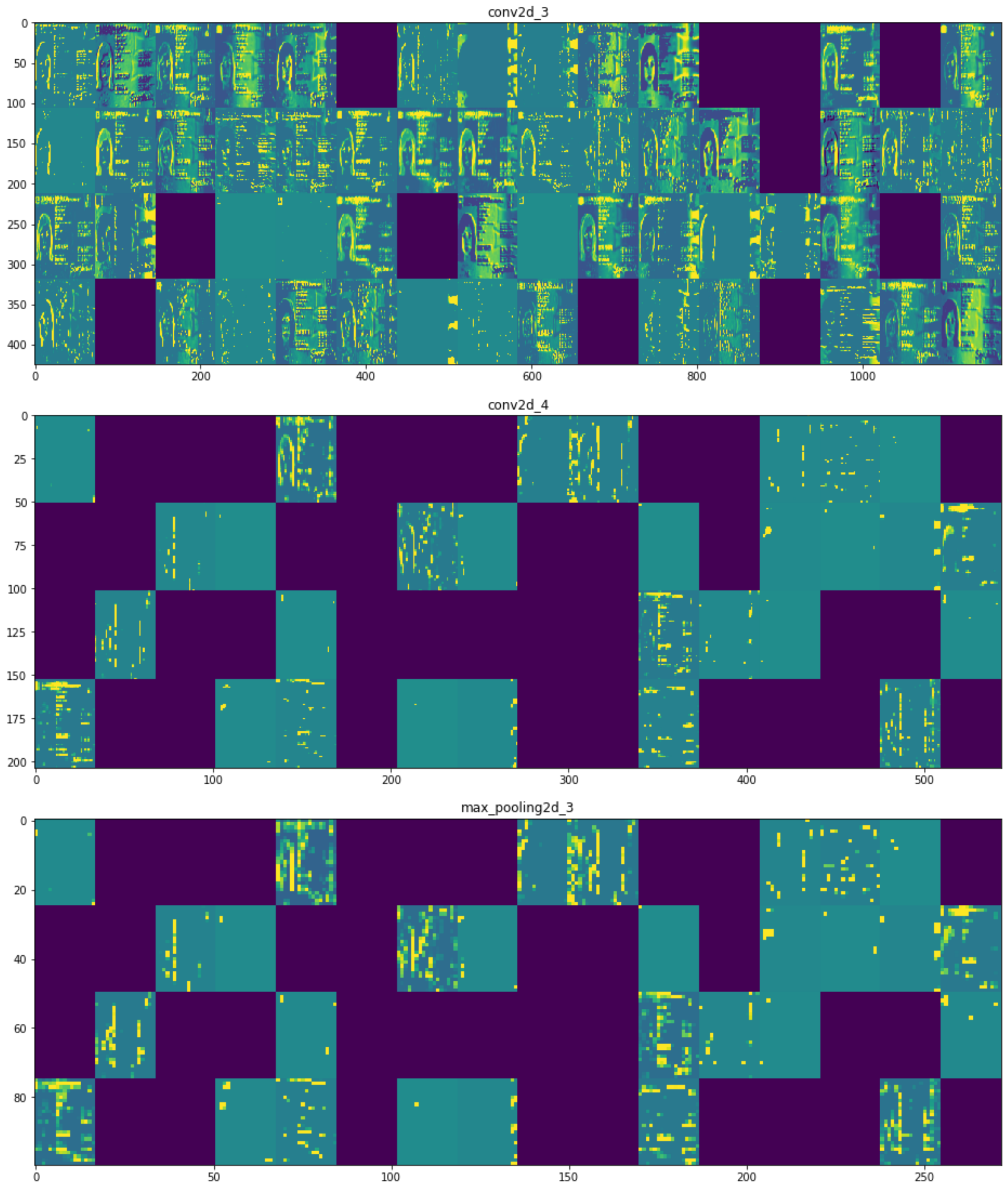


Figure 40. Last three layers activation maps

5. Conclusions and future work

5.1. Conclusions

We have reached a 98% of accuracy with our model but we believe we could have achieved better results if we would have had more time to try with different hyperparameters as there is not a way to find the best architecture other than trial and error.

We would need also more computational resources in the form of virtual machines or on-site GPUs.

And of course, more test data would be useful to be sure our network generalizes well with data captured on a diversity of situations.

As we do state, most of the hard work of building such type of systems is to get a suitable training dataset. Acquiring, handling and managing these big sets of data can be a specialitation by itself.

By way of conclusion, it is remarkable that a one-person team have created, in two months, a system that classifies 144 different identity documents, even in the bad state, and performs almost like a human being (two errors every hundred documents).

The final though of this work is about how surprisingly could be the future as AI technology became more and more available to the nonspecialist people.

5.2. Future work

The logical next step in this work seems to be to do the OCR of the documents like the professional software does.

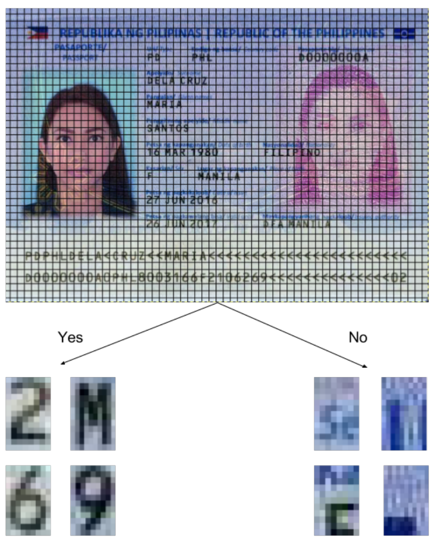


Figure 41. Binary classifier over a sliding window

We have done some promising experiments on enhancing the contrast of the text on the document over the background. At this moment we can extract the text part of the record based on the font size. Other geometric extracting methods like MSER are promising too but it would be worth to try with the approach enunciated by the professor Ng on their Coursera course of Machine Learning[23] based on a logistic binary NN.

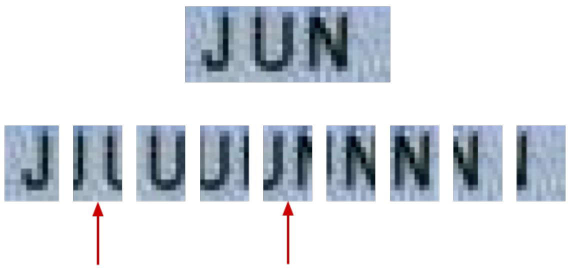


Figure 42. Segmenting the word JUN. The red arrows indicate the segments that a binary classifier recognizes as a space between characters

We prepare images with characters and images that don't contain a character, or it includes only a part (see fig. 41). Train a logistic binary classifier with these images. Then, we can slide a window over the document image and use this network to check if there is a character in the window. In the affirmative case, another network like the presented in this work will classify it on the alphabet.

Regarding the word segmentation, we can similarly address the problem (see fig. 42): a window slides from left to right and for each step tries to classify if it is in space or not with another binary classifier.

5.3. Acknowledgment

The author wishes to thank Professor Lourdes Meler Corretjé the support and the indications during the realization of this work.

Pere Vilàs Marí
Can Tià de Can Hereu
Sant Mateu d'Aubarca
Eivissa

Glossary

affine transformation In geometry, an affine transformation is a function between affine spaces which preserves points, straight lines and planes. Also, sets of parallel lines remain parallel after an affine transformation. An affine transformation does not necessarily preserve angles between lines or distances between points, though it does preserve ratios of distances between points lying on a straight line. 23, 24

BRIEF Binary Robust Independent Elementary Features. 43

BRISK Binary Robust Invariant Scalable Keypoints. 43

CNN Convolutional neural network. 6, 16, 34, 53

feature In computer vision and image processing, a feature is a piece of information which is relevant for solving the computational task related to a certain application. 10–13, 15, 16, 19, 20, 22, 24

homography a type of transformation that maps straight lines onto another plane. 23, 24, 49

inference Statistical inference is the process of deducing properties of an underlying probability distribution by analysis of data. 25, 29, 46, 53

MSER Maximally stable extremal region extractor. 60

ORB Oriented FAST and Rotated BRIEF. 43

PRADO Public Register of Authentic travel and identity Documents Online. 31

rectify Correct or adjust the measurements and orientation of an image. 24, 46

SIFT Scale-invariant feature transform. 43

SURF Speeded Up Robust Features. 43

Bibliography

[1] David G. Lowe. “Distinctive image features from scale-invariant keypoints”. In: *International Journal of Computer Vision, 60, 2*. Bristol, UK, 2004. URL: `https://link.springer.com/content/pdf/10.1023%5C%2FB%5C%3AVISI.0000029664.99615.94.pdf`.

[2] P. F. Alcantarilla, A. Bartoli, and A. J. Davison. “KAZE Features”. In: *Eur. Conf. on Computer Vision (ECCV)*. Fiorenze, Italy, 2012. URL: `https://www.researchgate.net/publication/236985005%5C_KAZE%5C_Features`.

[3] P. F. Alcantarilla, J. Nuevo, and A. Bartoli. “Fast Explicit Diffusion for Accelerated Features in Nonlinear Scale Spaces”. In: *British Machine Vision Conf. (BMVC)*. Bristol, UK, 2013.

[4] Aaron Courville Ian Goodfellow Yoshua Bengio. “Deep Learning”. In: *The Internet Protocol Journal*. Vol. 1. MIT Press, 2016. Chap. Chapter 9 - Convolutional Networks. URL: `http://www.deeplearningbook.org/contents/convnets.html`.

[5] Zenghui Wang Waseem Rawat. In: *Deep Convolutional Neural Networks for Image Classification: A Comprehensive Review*. Department of Electrical and Mining Engineering, University of South Africa: MIT Press Journals, 2017. URL: `https://www.mitpressjournals.org/doi/pdf/10.1162/neco_a_00990`.

[6] Ethan Rublee et al. *ORB: an efficient alternative to SIFT or SURF*. URL: `http://www.willowgarage.com/sites/default/files/orb%5C_final.pdf`.

[7] Yann Lecun et al. *Gradient-Based learning applied to Document Recognition*. URL: `http://yann.lecun.com/exdb/publis/pdf/lecun-98.pdf`.

[8] Peter Norvig Alon Halevy and Fernando Pereira. *The Unreasonable Effectiveness of Data*. URL: `https://static.googleusercontent.com/media/research.google.com/en//pubs/archive/35179.pdf`.

[9] Jason Brownlee. *Discover Feature Engineering, How to Engineer Features and How to Get Good at It*. URL: `https://machinelearningmastery.com/discover-feature-engineering-how-to-engineer-features-and-how-to-get-good-at-it/`.

[10] François Chollet. *Deep learning with python. Chapter 5. Deep learning for computer vision*. URL: `https://livebook.manning.com/#!/book/deep-learning-with-python/chapter-5/`.

[11] François Chollet. *Keras: The Python Deep Learning library*. URL: `https://keras.io/`.

[12] EU Council. *By Country. Public Register of Authentic travel and identity Documents Online*. URL: `http://www.consilium.europa.eu/prado/en/prado-documents/esp/h/o/index-type-per-cat.html`.

[13] EU Council. *PRADO glossary*. URL: `http://www.consilium.europa.eu/prado/en/prado-glossary/prado-glossary.pdf`.

[14] EU Council. *Public Register of Authentic travel and identity Documents Online*. URL: `http://www.consilium.europa.eu/prado/en/prado-start-page.html`.

[15] United Nations Statistics Division. *ISO 3166-1 alpha-3*. URL: https://unstats.un.org/unsd/tradekb/Knowledgebase/Country-Code.

[16] Apache Software Foundation. *Gluon. Deep Learning - The Straight Dope*. URL: http://gluon.mxnet.io/.

[17] Apache Software Foundation. *Mxnet. A Flexible and Efficient Library for Deep Learning*. URL: http://mxnet.apache.org/.

[18] Inc. Google. *TensorFlow. An open-source software library for Machine Intelligence*. URL: https://www.tensorflow.org/.

[19] Geoffrey E. et al. (2012) Hinton. *Improving neural networks by preventing co-adaptation of feature detectors*. URL: https://arxiv.org/abs/1207.0580.

[20] ImageMagick. *ImageMagick*. URL: https://www.imagemagick.org/script/index.php.

[21] Amazon Inc. *Amazon Web Services (AWS) SDK for Python*. URL: https://boto3.readthedocs.io/en/latest/.

[22] A. Zisserman K. Simonyan. *Very Deep Convolutional Networks for Large-Scale Image Recognition*. URL: https://arxiv.org/abs/1409.1556.

[23] Andrew Ng. *NN Model representation*. URL: https://www.coursera.org/learn/machine-learning/supplement/cRa2m/model-representation.

[24] Michael Nielsen. *A visual proof that neural nets can compute any function*. URL: http://neuralnetworksanddeeplearning.com/chap4.html.

[25] NVIDIA. *NVIDIA Titan V*. URL: https://www.nvidia.es/titan/titan-v/.

[26] OpenCV. *Clustering*. URL: http://scikit-learn.org/stable/modules/clustering.html.

[27] OpenCV. *Feature detection and description*. URL: https://docs.opencv.org/master/db/d27/tutorial%5C_py%5C_table%5C_of%5C_contents%5C_feature2d.html.

[28] OpenCV. *Feature Matching*. URL: https://docs.opencv.org/3.0-beta/doc/py%5C_tutorials/py%5C_feature2d/py%5C_matcher/py%5C_matcher.html.

[29] scikit-learn. *Cap 1.4 Classification*. URL: http://scikit-learn.org/stable/modules/svm.html%5C#svm.

[30] scikit-learn. *Cap 2.3 Clustering*. URL: http://scikit-learn.org/stable/modules/clustering.html%5C#clustering.

[31] scikit-learn. *Chapter 2.5. Decomposing signals in components (matrix factorization problems)*. URL: http://scikit-learn.org/stable/modules/decomposition.html.

[32] Utkarsh Sinha. *SIFT: Theory and Practice*. URL: http://www.aishack.in/tutorials/sift-scale-invariant-feature-transform-introduction/.

[33] Eric W. Weisstein. *Hamming Distance*. Ed. by Wolfram. URL: http://mathworld.wolfram.com/HammingDistance.html.

[34] Eric W. Weisstein. *L2-Norm*. Ed. by Wolfram. URL: http://mathworld.wolfram.com/L2-Norm.html.

[35] Wikipedia. *Training, test, and validation sets*. URL: https://en.wikipedia.org/wiki/Training,_test,_and_validation_sets#Validation_set.

[36] wikipedia. *Computer vision*. URL: https://en.wikipedia.org/wiki/Computer%5C_vision.

[37] wikipedia. *Histrogram of Oriented Gradients*. URL: https://en.wikipedia.org/wiki/Histogram_of_oriented_gradients.

[38] wikipedia. *Principal Component Analysis*. URL: https://en.wikipedia.org/wiki/Principal%5C_component%5C_analysis.

[39] wikipedia. *Recto and verso*. URL: https://en.wikipedia.org/wiki/Recto%5C_and%5C_verso.

[40] Xataka. *Nvidia Volta*. URL: https://www.xataka.com/componentes/nvidia-volta-y-sus-12-nm-finfet-nos-preparan-para-una-nueva-revolucion-grafica-en-2017-y-2018.

[41] Zdenˇek Zabokrtsk´y. *Feature Engineering in Machine Learning. Institute of Formal and Applied Linguistics, Charles University in Prague*. URL: https://ufal.mff.cuni.cz/~zabokrtsky/courses/npfl104/html/feature_engineering.pdf.

Appendix A Extraction speed

In this table, we compare the extraction speed of several methods averaged over 4000 documents.

iMac 2014

Method	Mean Time	Std. Dev.	Points/regions
brief	0.009680336	0.000465094	462.0
sift	0.14352594	0.018444292	3221.5
brisk	0.070546037	0.026112355	4836.0
kaze	0.228964102	0.014583509	1466.5
akaze	0.047307038	0.000482393	1317.5
mser	0.0366598845	0.0085691335	831.5
orb	0.009431679	0.001127997	500.0
surf	0.022489961	0.008004123	205.5

Table 2. Speed of extraction on iMac2014. Average of 4000 images

On a better machine the results are similar because the extractor uses only one core.

AWS c4.4xlarge amazon instance

Method	Mean Time	Std. Dev.	Points/regions
brief	0.00945078146612	0.00106868660076	492.2
sift	0.0617583851528	0.0071591356382	2700.5
brisk	0.0566153216137	0.0251370948462	3452.2
kaze	0.16076961502	0.0107957820419	1595.8
akaze	0.0400814163854	0.00119664243422	1187.9
mser	0.0530032530082	0.0199428392754	1197.5
orb	0.00895235451532	0.00145218528445	497.6
surf	0.013524252921	0.00380727612299	305.8

Table 3. Speed of extraction on AWS c4.4xlarge instance. Average of 4000 images

Appendix B Rectify an image

The function accepts an image and returns another straight and with the standard size.

Listing 16 Rectify an image

```python
def rectify(image):
    # 1. resize to an initial known size
    rsz_img = cv2.resize(image, dim, interpolation=inter)
    # 2. make a black border and translate to prevent corner clipping
    brd_img = cv2.copyMakeBorder(
        rsz_img,
        MARGIN_OFFSET, MARGIN_OFFSET, MARGIN_OFFSET, MARGIN_OFFSET,
        cv2.BORDER_CONSTANT, BACKGROUND_COLOR)
    # 3. convert the image to grayscale, blur it and pass Solvel filter
    gray_img = cv2.cvtColor(brd_img, cv2.COLOR_BGR2GRAY)
    gray_gauss = cv2.GaussianBlur(gray_img, (5, 5), 0)
    edged_img = cv2.adaptiveThreshold(gray_gauss, 255,
                                      cv2.ADAPTIVE_THRESH_GAUSSIAN_C,
                                      cv2.THRESH_BINARY_INV, 51, SOBEL_MARGIN)
    # 4. find the contours in the edged image, keeping only the
    # largest ones, and initialize the screen contour
    (_, cnts, _) = cv2.findContours(edged_img,
                                    cv2.RETR_EXTERNAL,
                                    cv2.CHAIN_APPROX_SIMPLE)
    # 5. Deskew the image
    # order by max area contour
    cnts = sorted(cnts, key=cv2.contourArea, reverse=True)
    # bounding rectangle
    x, y, w, h = cv2.boundingRect(cnts[0])
    # rotated rectangle containing document
    box = np.int0(cv2.boxPoints(cv2.minAreaRect(cnts[0])))
    # order points of de rectangle
    box=self.order_points(box)
    (tl, tr, br, bl) = box
    # find the gradient of the border top line
    if (tr[0]-tl[0]!=0):
        gradient=(tr[1]-tl[1])/(tr[0]-tl[0])
        # calculate the angle with x axis
        theta=np.arctan(gradient)
    else:
        theta=0
    theta_deg=180*theta/np.pi
    # calculate rotation center
    rot_x=tl[0]
    rot_y=tl[1]
    inf_corner=br # to crop the resulting image
    # test w>h
    delta_prop=0
    if FORCE_PROPORTION:
        lon_hline = np.abs(tr[0]-tl[0])
        lon_vline = np.abs(bl[1]-tl[1])
        if lon_vline>lon_hline:
            # if the document is so rotated, add up pi/2 rad to rot
            theta_deg+=90
            delta_prop=np.pi/2 # to rotate inferior corner
            rot_x=tr[0]
            rot_y=tr[1]
            inf_corner=bl
```

Listing 17 Rectify an image. Cont.

```
    # the document is rotated
    # we must straight it
    if theta!=0:
        # calculate the rotation matrix
        M=cv2.getRotationMatrix2D((rot_x,rot_y), theta_deg, 1.0)
        # rotate the resized image
        rot_img=self.rotate_img(brd_img, M)
        # calculate coords of the rotated inferior corner (resized image)
        t=theta+delta_prop
        Mr=np.array([[np.cos(t), -np.sin(t)],[np.sin(t),np.cos(t)]])
        m_inf=np.array([[inf_corner[0]-rot_x, inf_corner[1]-rot_y]])
        nr_inf=np.dot(m_inf, Mr) # rotate the inferior corner
        r_inf_x=int(nr_inf[0][0]+rot_x)
        r_inf_y=int(nr_inf[0][1]+rot_y)
        doc_w=np.abs(r_inf_x-rot_x) # rotated document width
        doc_h=np.abs(r_inf_y-rot_y) # rotated document height
    else:
        rot_img=brd_img
        doc_w=np.abs(rot_x-br[0]) # document width
        doc_h=np.abs(rot_y-br[1])
    # 6. Translate to the origin and crop, add 1/4 of margin
    Mt = np.float32([[1,0,-1*(rot_x+SOBEL_MARGIN)],[0,1,-1*(rot_y+SOBEL_MARGIN)]])
    (ih, iw) = brd_img.shape[:2]
    trs_img = cv2.warpAffine(rot_img, Mt, (iw, ih))
    # 7. Crop image
    crop_img=trs_img[0:int(doc_h-(2*SOBEL_MARGIN)), 0:int(doc_w-(2*SOBEL_MARGIN))]
    # return the cropped resized image
    return self.resize(crop_img, width=BASE_SIZE)
```

Appendix C Donwloading PRADO images

Listing 18 Getting the pages of each country

```
/* iso3 country code*/
var iso3=[ 'abw', 'afg', ..];

/* root url for all documents of de country */
var url_pre =  'http://www.consilium.europa.eu/prado/en'+
               '/prado-documents/'
var url_post = '/all/index.html'

for (var a=0; a<=iso3.length; a++) {
    extract_docs(url_pre+iso3[a]+url_post);
}
```

Listing 19 Download image documents of each country

```
function extract_docs(page_url)  {
    $.get( page_url, function( data ) {
        /* for each document */
        $(data).find('div.doc-info').each(function(index,element) {
            /* get description and doc code*/
            var doc_info =
                $(element).find('p.doc-info-other a').text();
            var toks=doc_info.split("  ")
            var doc_code = toks[toks.length-1]
            var doc_desc=toks[0]

            doc_desc=String(doc_desc).replace(/"/g,"")
            doc_desc=String(doc_desc).replace(/''/g,"")

            /* image number of recto and verso */
            var recto="", verso="";
            $(element).
              find('div.doc-thumbnails a img').
                each(function() {
                    var source = $(this).attr('src').split("_")
                    if (recto=="")
                        recto=source[1]; else verso=source[1];
            })

            /* obtain country, category, type and number */
            var cod_tok=doc_code.split('-')
            var country=cod_tok[0]
            var category=cod_tok[1][0]
            var type=cod_tok[1][1]
            var number=cod_tok[2].substring(0,2)
            var version=cod_tok[2].substring(2,6)

            /* obtain date first issued */
            var first_issued=""
            $(element).find('p.doc-info-other').each(function() {
                var text=$(this).text()
                if (text.includes('First Issued On:')) {
                    first_issued=$(this).text().split(':')[1]
                }
            })

            format_doc(doc_code, country, category, type,
            number, version, doc_desc,
                        recto, verso)
        })
    }).fail(function() {
        console.log('Failed to download ', page_url)
    })
}
```

Appendix D The match function

Listing 20 Matching function

```python
def match(name='DOC', method='', query=[], trained=[]):
    """ perform the match
        name - to identify the document on the jobs list
        method - desired matching method
        query - dict of method with tuples with
               (kps, des) of the query image
        trained - dict of methods with tuples with
               (kps, des) of the trained image

        return a dict with method/result
        The result is the \% of points that are found
        in the homography
    """
    # create and initialize the results dict
    # one key for each method
    results = {}
    for k in matchers.keys():
        results[k] = 0.0

    # iterate over methods
    for key_matcher in matchers.keys():
        matcher = matchers[key_matcher]

        ratio=0.9 if key_matcher=='orb' else 0.7
        minMatches=20 if key_matcher=='orb' else 40

        kpsA = np.array(query[key_matcher][0])
        feaA = np.array(query[key_matcher][1])
        kpsB = np.array(trained[key_matcher][0][0])
        feaB = np.array(trained[key_matcher][1])

        """
        Nearest neighbors are defined by the smallest Euclidean
        distance between feature vectors. The two feature vectors
        with the smallest Euclidean distance are considered to be
        neighbors
        """
        try:
            rawMatches = matcher.knnMatch(feaB, trainDescriptors=feaA, k=2)

            # take only matches that are on the ratio distance
            matches = []
            for m in rawMatches:
                if len(m) == 2 and
                   m[0].distance < m[1].distance * ratio:
                    matches.append((m[0].trainIdx, m[0].queryIdx))
            """
              if there is more matches than the
              thresold, calculate homography.
              the hessian matrix H will contain
              the parameters to perform the inverse
              affine transform
            """
            if len(matches) > minMatches:
                try:
                    # convert to fp32
                    ptsA = np.float32([kpsA[i] for (i, _) in matches])
                    ptsB = np.float32([kpsB[j] for (_, j) in matches])

                    # find homography
                    (H, status) =
                       cv2.findHomography(ptsA, ptsB, cv2.RANSAC, 4.0)
                    # return \% of matching keypoints
                    # store result in a dict with key matcher-name
                    results[key_matcher] =
                        float(status.sum()) / status.size
                except Exception as ex:
                    print('ERROR: {} file {} method {} - {}'.format(
                        funcname(), name, key_matcher, ex
                    ))
                    results[key_matcher] = 0.0
        except Exception as ex:
            print('ERROR knnMatcher: {} file {} method {} - {}'.format(
                funcname(), name, key_matcher, ex
            ))
            results[key_matcher] = 0.0

    # convert results to ordered list
    l = []
    for m in method:
        l.append(results[m])
    return (name, l)
```

Appendix E Plotting filter responses

Listing 21 Plotting the responses

```
layer_names = []
for layer in model.layers[:7]:
    layer_names.append(layer.name)


images_per_row = 16

# Now let's display our feature maps
for layer_name, layer_activation in zip(layer_names, activations):
    # This is the number of features in the feature map
    n_features = layer_activation.shape[-1]

    # The feature map has shape (1, size, size, n_features)

    size_h = layer_activation.shape[1]
    size_w = layer_activation.shape[2]

    # We will tile the activation channels in this matrix
    n_cols = n_features // images_per_row
    display_grid = np.zeros((size_h * n_cols, images_per_row * size_w))

    # We'll tile each filter into this big horizontal grid
    for col in range(n_cols):
        for row in range(images_per_row):
            channel_image = layer_activation[0,
                                             :, :,
                                             col * images_per_row + row]
            # Post-process the feature to make it visually palatable
            channel_image -= channel_image.mean()
            channel_image /= channel_image.std()
            channel_image *= 64
            channel_image += 128
            channel_image = np.clip(channel_image, 0, 255).astype('uint8')
            display_grid[col * size_h : (col + 1) * size_h,
                         row * size_w : (row + 1) * size_w] = channel_image

    # Display the grid
    scale = 1. / size_w
    plt.figure(figsize=(scale * display_grid.shape[1],
                        scale * display_grid.shape[0]))
    plt.title(layer_name)
    plt.grid(False)
    plt.imshow(display_grid, aspect='auto', cmap='viridis')
```

Appendix F Matching with aberrated images

This table shows the results to try to match the aberrated image with the PRADO index. For each method (AKAZE, BRIS, ORB) we see the three first recognized documents and their probability.

These tables can be used to find the optimal method for matching.

Listing 22 Results of matching with aberrated images. page 1/2

~~~
# Results of matching with aberrated images
## AUS-AD-02001
Command used
```shell
./match.py -m all --features helpers/prado/prado-features
-d docs/tolerance/AUS-AD -c AUS
~~~
Filename: docs/tolerance/AUS-AD/AUS-AD-02001_verso-original.jpg
----
akaze   brisk   orb
AUS-AD-02001 (100.000)   AUS-AD-02001 (100.000)   AUS-AD-02001 (100.000)
AUS-AO-02002 (86.688)   AUS-AO-02002 (90.878)   AUS-AO-02002 (49.446)
AUS-AS-02002 (86.688)   AUS-AS-02002 (90.878)   AUS-AS-02002 (49.446)
Filename: docs/tolerance/AUS-AD/bend.jpg
----
akaze   brisk   orb
AUS-AD-02001 (84.028)   AUS-AD-02001 (83.507)   AUS-AD-02001 (58.788)
AUS-AO-02002 (74.951)   AUS-AO-02002 (75.610)   AUS-AO-02002 (40.784)
AUS-AS-02002 (74.951)   AUS-AS-02002 (75.610)   AUS-AS-02002 (40.784)
Filename: docs/tolerance/AUS-AD/contrast-90.jpg
----
akaze   brisk   orb
AUS-AD-02001 (0.000)   AUS-AD-02001 (62.255)   AUS-AD-02001 (80.902)
AUS-AO-02002 (0.000)   AUS-AO-02002 (29.927)   AUS-AO-02002 (36.719)
AUS-AS-02002 (0.000)   AUS-AS-02002 (29.927)   AUS-AS-02002 (36.719)
Filename: docs/tolerance/AUS-AD/contrast.jpg
----
akaze   brisk   orb
AUS-AD-02001 (70.109)   AUS-AD-02001 (92.120)   AUS-AD-02001 (42.969)
AUS-AO-02002 (57.895)   AUS-AO-02002 (70.513)   AUS-AO-02002 (21.951)
AUS-AS-02002 (57.895)   AUS-AS-02002 (70.513)   AUS-AS-02002 (21.951)
Filename: docs/tolerance/AUS-AD/flare-on-name.jpg
----
akaze   brisk   orb
AUS-AD-02001 (99.044)   AUS-AD-02001 (99.066)   AUS-AD-02001 (81.058)
AUS-AO-02002 (91.699)   AUS-AO-02002 (90.650)   AUS-AO-02002 (37.719)
AUS-AS-02002 (91.699)   AUS-AS-02002 (90.650)   AUS-AS-02002 (37.719)
Filename: docs/tolerance/AUS-AD/gaussian2x2.jpg
----
akaze   brisk   orb
AUS-AD-02001 (96.010)   AUS-AD-02001 (88.686)   AUS-AD-02001 (80.282)
AUS-AO-02002 (93.978)   AUS-AO-02002 (77.673)   AUS-AO-02002 (43.657)
AUS-AS-02002 (93.978)   AUS-AS-02002 (77.673)   AUS-AS-02002 (43.657)
Filename: docs/tolerance/AUS-AD/gaussian5x5.jpg
----
akaze   brisk   orb
AUS-AD-02001 (94.388)   AUS-AD-02001 (96.547)   AUS-AD-02001 (77.562)
AUS-AO-02002 (91.435)   AUS-AO-02002 (84.127)   AUS-AO-02002 (43.846)
AUS-AS-02002 (91.435)   AUS-AS-02002 (84.127)   AUS-AS-02002 (43.846)
Filename: docs/tolerance/AUS-AD/laplacian.jpg
----
akaze   brisk   orb
AUS-AD-02001 (97.125)   AUS-AD-02001 (98.637)   AUS-AD-02001 (79.639)
AUS-AO-02002 (86.879)   AUS-AO-02002 (87.824)   AUS-AO-02002 (44.649)
AUS-AS-02002 (86.879)   AUS-AS-02002 (87.824)   AUS-AS-02002 (44.649)
Filename: docs/tolerance/AUS-AD/motion-linear-5x145.jpg
----
akaze   brisk   orb
AUS-AD-02001 (89.418)   AUS-AD-02001 (76.699)   AUS-AD-02001 (36.170)
AUS-AO-02002 (84.314)   AUS-AO-02002 (64.246)   AUS-AO-02002 (35.533)
AUS-AS-02002 (84.314)   AUS-AS-02002 (64.246)   AUS-AS-02002 (35.533)
```
```

Listing 23 Results of matching with aberrated images. page 2/2

```
Filename: docs/tolerance/AUS-AD/multiplicative.jpg
----------------------------------------------------------------
akaze                    brisk                    orb
AUS-AD-02001 (85.806)    AUS-AD-02001 (92.895)    AUS-AD-02001 (49.213)
AUS-AO-02002 (79.938)    AUS-AO-02002 (85.024)    AUS-AO-02002 (35.398)
AUS-AS-02002 (79.938)    AUS-AS-02002 (85.024)    AUS-AS-02002 (35.398)
Filename: docs/tolerance/AUS-AD/noise20.jpg
----------------------------------------------------------------
akaze                    brisk                    orb
AUS-AD-02001 (89.167)    AUS-AD-02001 (93.084)    AUS-AD-02001 (41.767)
AUS-AO-02002 (88.066)    AUS-AO-02002 (92.045)    AUS-AO-02002 (30.698)
AUS-AS-02002 (88.066)    AUS-AS-02002 (92.045)    AUS-AS-02002 (30.698)
Filename: docs/tolerance/AUS-AD/noise40.jpg
----------------------------------------------------------------
akaze                    brisk                    orb
AUS-AO-02002 (85.714)    AUS-AD-02001 (86.747)    AUS-AD-02001 (26.562)
AUS-AS-02002 (85.714)    AUS-AO-02002 (70.690)    AUS-AO-02002 (14.773)
AUS-AD-02001 (75.497)    AUS-AS-02002 (70.690)    AUS-AS-02002 (14.773)
Filename: docs/tolerance/AUS-AD/pixelize2x2.jpg
----------------------------------------------------------------
akaze                    brisk                    orb
AUS-AD-02001 (97.674)    AUS-AD-02001 (85.424)    AUS-AD-02001 (75.783)
AUS-AO-02002 (92.391)    AUS-AO-02002 (81.313)    AUS-AO-02002 (43.295)
AUS-AS-02002 (92.391)    AUS-AS-02002 (81.313)    AUS-AS-02002 (43.295)
Filename: docs/tolerance/AUS-AD/poisson.jpg
----------------------------------------------------------------
akaze                    brisk                    orb
AUS-AD-02001 (85.417)    AUS-AD-02001 (92.035)    AUS-AD-02001 (42.742)
AUS-AO-02002 (75.197)    AUS-AO-02002 (73.203)    AUS-AO-02002 (31.081)
AUS-AS-02002 (75.197)    AUS-AS-02002 (73.203)    AUS-AS-02002 (31.081)
Filename: docs/tolerance/AUS-AD/refraction105.jpg
----------------------------------------------------------------
akaze                    brisk                    orb
AUS-AO-02002 (90.818)    AUS-AO-02002 (87.423)    AUS-AD-02001 (43.986)
AUS-AS-02002 (90.818)    AUS-AS-02002 (87.423)    AUS-AO-02002 (40.304)
AUS-AD-02001 (79.799)    AUS-AD-02001 (81.369)    AUS-AS-02002 (40.304)
Filename: docs/tolerance/AUS-AD/rotate4offset5.jpg
----------------------------------------------------------------
akaze                    brisk                    orb
AUS-AD-02001 (97.029)    AUS-AD-02001 (96.825)    AUS-AD-02001 (65.749)
AUS-AO-02002 (95.681)    AUS-AO-02002 (87.543)    AUS-AO-02002 (45.522)
AUS-AS-02002 (95.681)    AUS-AS-02002 (87.543)    AUS-AS-02002 (45.522)
total execution time: 22.606020873s
```