Assignment 1: Instance-level recognition

YACHAOUI Ayman
Object Recognition

Ecole Normale Supérieure de Cachan

I Sparse features for matching specific objects in images

I.A SIFT features detections

QIA.1: Why is it important to obtain similarity co-variant features for matching?
We want similarity features to be co-variant so that translation, rotation, scaling, etc, won't interfere with feature detection. Basically, two identical images taken from different camera angles and position will have a better chance of returning the same features.

QIA.2: Show the detected features in the image for three different values of the peakThreshold option.
Bellow are the features corresponding for three different settings of the peak threshold, respectively 0.001, 0.004 and 0.01. We can clearly observe that the higher the threshold is, the fewer features we are able to identify.

Figure 1: Features with peak-threshold = 0.001.

Figure 2: Features with peak-threshold = 0.004.

Figure 3: Features with peak-threshold = 0.01.

QIA.3: Note the change in density of detections across images. Why does it change? Which implications for image matching can this have? How can it be avoided?
The peak threshold controls the minimum of contrast needed to accept a feature in order to mitigate false-features detection because of noise. Thus by diminishing its value, more features are detected but on the other hand they might not be relevant to the object we're trying to identify. Practically this means that the same feature of the same object in two different images might require different threshold to be detected. To avoid this it might be interesting to pre-process the two images before doing the direct comparison, for instance by normalizing their histogram.

I.B SIFT features descriptors and matching between images

QIB.1: Note the descriptors are computed over a much larger region (shown in blue) than the detection (shown in green). Why should this be a good strategy?
Less information is needed to detect a feature than to describe it. In the case of image comparison, we need a lot of information on the feature, hence a bigger region to compute the descriptor, in order to be able to match different features with certainty and more precisely to discriminate between different features.

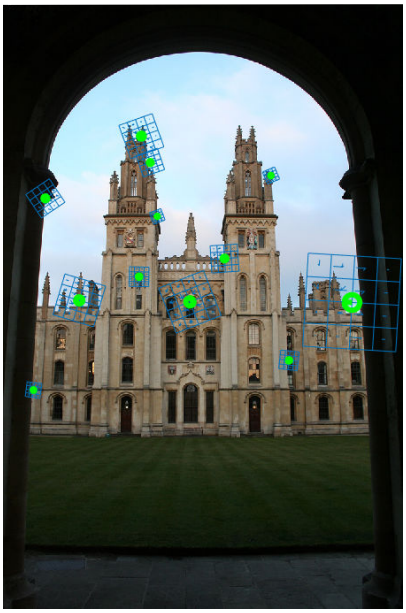

Figure 4: SIFT feature detection (green) and decriptor (blue).

QIB.2: Examine and visualize in your report some of the mismatches to understand why the mistakes are being made. For example, is the change in lighting a problem? What additional constraints can be applied to remove the mismatches?
There are multiple possible sources for mismatches, such as the presence of similar objects but at different positions, shadows or even noise. In order to limit the number of mismatches, we can first discriminate using the ratio of the first to second nearest neighbors -basically making sure that the neighborhood really means something- and then making sure that group of feature that are near each other in the first image all match with a group of feature that are also near each other on the second image.

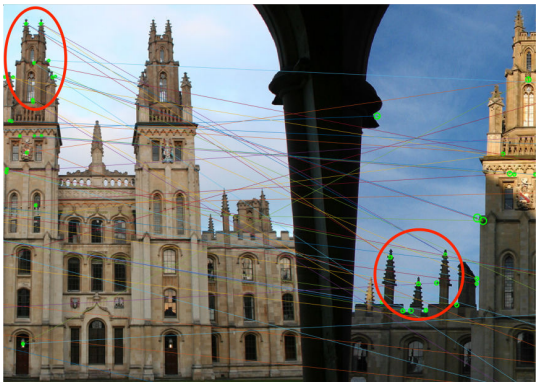

Figure 5: Mismatches due to resembling objects.

Figure 6: Mismatches due to shadow.

I.C Improving SIFT matching (i) using Lowe's second nearest neighbour test

QIC.1: Illustrate and comment on improvements that you can achieve with this step. Include in your report figures showing the varying number of tentative matches when you change the nnThreshold parameter.
Below are the figures corresponding to different nnThreshold values ranging from 0.1 to 0.9. Decreasing the threshold by 0.1 roughly corresponds to dividing the number of matches by 2. We can observe that for a ratio of 0.2, there are no more matches. If we look closely, the last mismatch disappears with a ratio of 0.4. This means that with this technique if we want absolutely no mismatch, then we severely decrease the overall number of features, thus it might be a better idea to combine this technique with others to achieve a lower number of mismatch while keeping enough features.

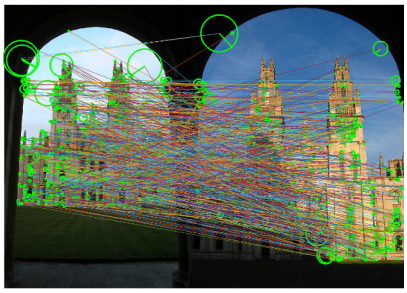

Figure 7: 1st-to-2nd nn equal to 0.9 (686 matches).

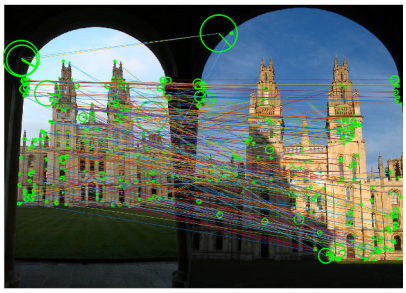

Figure 8: 1st-to-2nd nn equal to 0.8 (293 matches).

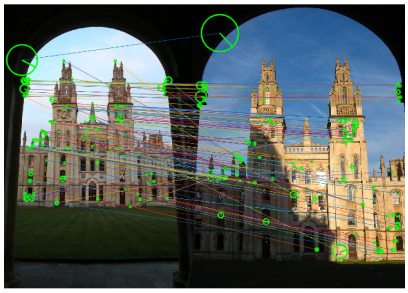

Figure 9: 1st-to-2nd nn equal to 0.7 (148 matches).

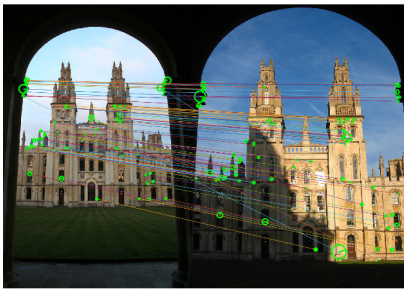

Figure 10: 1st-to-2nd nn equal to 0.6 (92 matches).

Figure 11: 1st-to-2nd nn equal to 0.5 (55 matches).

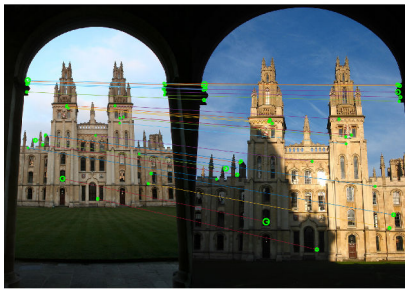

Figure 12: 1st-to-2nd nn equal to 0.4 (28 matches).

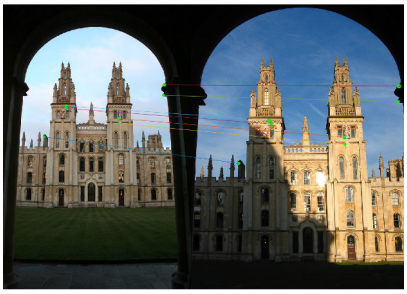

Figure 13: 1st-to-2nd nn equal to 0.3 (7 matches).

Figure 14: 1st-to-2nd nn equal to 0.2 (0 matches).

Figure 15: 1st-to-2nd nn equal to 0.1 (0 matches).

I.D Improving SIFT matching (ii) using a geometric transformation

QID.1: Work out how to compute this transformation from a single correspondence. Hint: recall from Stage I.A that a SIFT feature frame is an oriented circle and map one onto the other.
Let's consider the two matching pairs of points corresponding to the circle center and their oriented extremity. Then if we denote by $(r_1, \theta_1) \in \mathbb{R}_+ \times [0, 2\pi]$ and $(r_2, \theta_2) \in \mathbb{R}_+ \times [0, 2\pi]$ the radius and angle corresponding to the

circle in respectively the first and second image and by $(x_{c1}, y_{c1})^\intercal$ and $(x_{c2}, y_{c2})^\intercal$ the circle centers, we have:

$$\begin{pmatrix} x_{c2} \\ y_{c2} \end{pmatrix} = s \begin{pmatrix} \cos\theta & -\sin\theta \\ \sin\theta & \cos\theta \end{pmatrix} \begin{pmatrix} x_{c1} \\ y_{c1} \end{pmatrix} + \begin{pmatrix} t_x \\ t_y \end{pmatrix}$$

$$\begin{pmatrix} x_{c2} + r_2\cos\theta_2 \\ y_{c2} + r_2\sin\theta_2 \end{pmatrix} = s \begin{pmatrix} \cos\theta & -\sin\theta \\ \sin\theta & \cos\theta \end{pmatrix} \begin{pmatrix} x_{c1} + r_1\cos\theta_1 \\ y_{c1} + r_1\sin\theta_1 \end{pmatrix} + \begin{pmatrix} t_x \\ t_y \end{pmatrix}$$

Then we can rearrange the equations and:

$$\begin{pmatrix} r_2\cos\theta_2 \\ r_2\sin\theta_2 \end{pmatrix} = s \begin{pmatrix} \cos\theta & -\sin\theta \\ \sin\theta & \cos\theta \end{pmatrix} \begin{pmatrix} r_1\cos\theta_1 \\ r_1\sin\theta_1 \end{pmatrix} = \begin{pmatrix} sr_1\cos(\theta_1+\theta) \\ sr_1\sin(\theta_1+\theta) \end{pmatrix}$$

Thus we have:

$$\theta = \theta_2 - \theta_1$$

$$s = \frac{r_2}{r_1}$$

$$\begin{pmatrix} t_x \\ t_y \end{pmatrix} = \begin{pmatrix} x_{c2} - sx_{c1}\cos\theta + sy_{c1}\sin\theta \\ y_{c2} - sx_{c1}\sin\theta - sy_{c1}\cos\theta \end{pmatrix}$$

QID.2: Illustrate improvements that you can achieve with this step.
The difference between the second nearest neighbor technique and geometric verification is drastic, as is illustrated in the two figure bellow. The geometric verification seems to be way more accurate than the nearest neighbor, although it should be noted that in this pair of images the geometric transformation is quite mild.

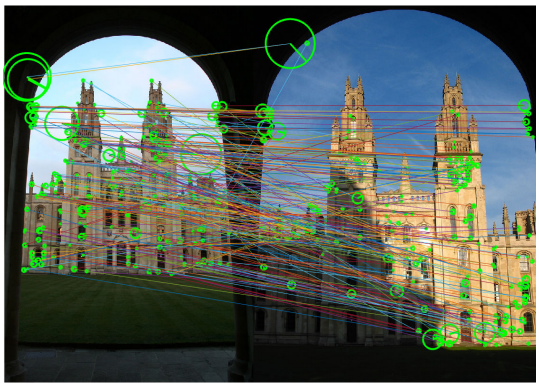

Figure 16: 2nd nearest neighbor matches.

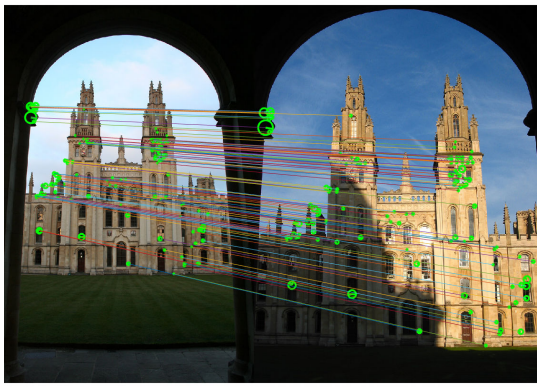

Figure 17: Geometric verification matches.

II Affine co-variant detectors

QII.1: Include in your report the plots showing the number of verified matches with changing viewpoint. At first there are more similarity detector matches than affine. Why?
Even though this is an affine transformation, at first it can be very well approximated by a similarity because the anisotropic scaling is very mild. As the distorsion worsen the similarity detector becomes less and less reliable. This becomes obvious when observing the different plots bellow:

Figure 18: Affinity from 1 to 2 (1579 matches).

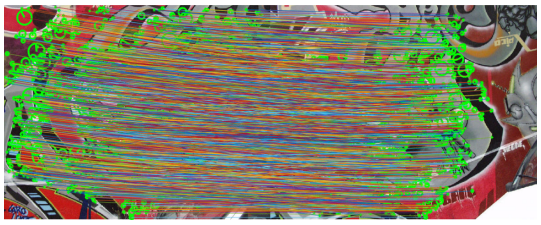

Figure 19: Similarity from 1 to 2 (1813 matches).

Figure 20: Affinity from 1 to 3 (790 matches).

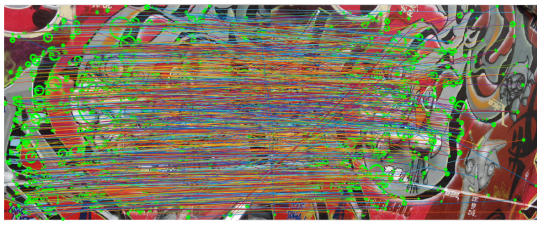

Figure 21: Similarity from 1 to 3 (905 matches).

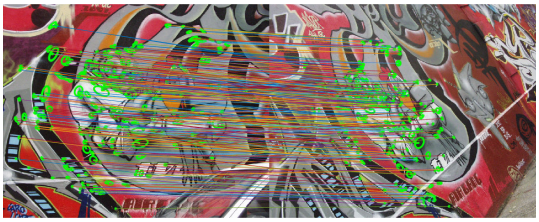

Figure 22: Affinity from 1 to 4 (383 matches).

Figure 23: Similarity from 1 to 4 (140 matches).

Figure 24: Affinity from 1 to 5 (114 matches).

Figure 25: Similarity from 1 to 5 (8 matches).

Figure 26: Affinity from 1 to 6 (43 matches).

Figure 27: Similarity from 1 to 6 (6 matches).

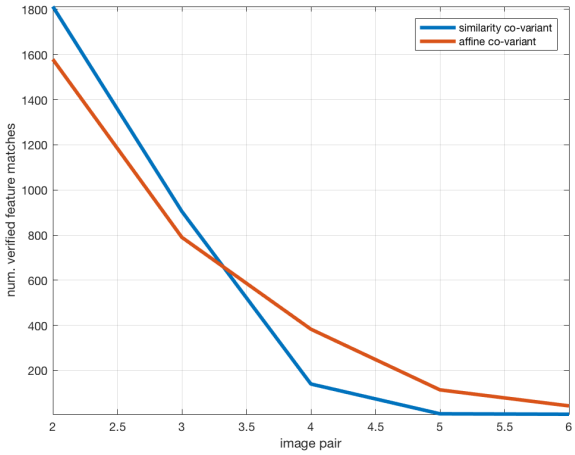

Figure 28: Evolution of verified features matches.

III Towards large scale retrieval

III.A Accelerating descriptor matching with visual words

QIIIA.1: In the above procedure the time required to convert the descriptors into visual words was not accounted for. Why the speed of this step is less important in practice?
In practice, if the dictionary is independent from the query image then all the visual words can be pre-computed for all the images in the database which implies that the descriptors have to be converted to visual words only for the query image.

QIIIA.2: What is the speedup in searching a large, fixed database of 10, 100, 1000 images?
Do evaluate the time speedup of the word vs raw descriptors techniques we used all the pictures in the database (actually we cycled through them and as there are less than 1000 images, some were used twice) and recorded the time it took for the two techniques. Then we simply computed the ratio. Everything is summarized in the table bellow:

iterations number	**raw time**	**word time**	**raw/word ratio**
10	240 ms	5 ms	46
100	2689 ms	46 ms	59
1000	25964 ms	453 ms	57

III.B Searching with an inverted index

QIIIB.1: Why does the top image have a score of 1?
The top image has a score of 1 due to the histogram normalization that was done as a refinement step.

QIIIB.2: How many erroneously matched images do you count in the top results?
Out of the top 25 results, 9 are correctly matched and 16 incorrectly, as can be seen in the figure bellow. There are 6 correct matches in the top 10.

Figure 29: Query image.

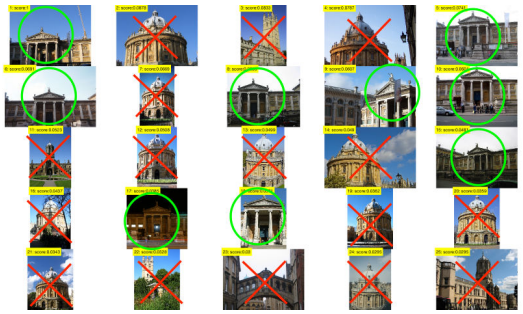

Figure 30: Top 25 match results with normalized histograms.

III.C Geometric rescoring

QIIIC.1: Why is the top score much larger than 1 now?
The top score is no more bounded to 1 because the normalized histogram just served as a first selection, the real score is then computed using the number of inliers which (hopefully) is larger than 1!

QIIIC.2: Illustrate improvements of retrieval results after geometric verification.
This added steps greatly improve the results. As can be seen in the figure bellow, 9 of the top 10 matches are correct matches and the incorrectly identified image is only the 10th. It is also to be noted that if we have only 9 images that correspond to the query in our database then we achieved the best possible results.

Figure 31: Top 25 match results with geometric verification.

III.D Full system

IV Large scale retrieval

QIV.1: How many features are there in the painting database?
There are 100000 features, meaning visual words in the database and each is described by a 128 dimension vector.

QIV.2: How much memory does the image database take?
The image database takes 350 MB of memory when loaded in Matlab, as retrieved using the *whos* function, and 192MB when saved on disk as a *.mat* file.

QIV.3: What are the stages of the search? And how long does each of the stages take for one of the query images?
The search follows 3 main steps that will be presented below:

(a) **Feature extraction.** The first step of the search is to extract the features of the query image, that is to compute the visual word histogram. The duration of this step may vary greatly depending on the query image. On the 3 query images available it took between 250ms and 2s.

(b) **Inverted index.** Then this histogram is used to compute the inverted index for all images. The result, when sorted, gives a first estimation of the most likely matches. As the database images index is precomputed, this step is quite fast and takes only 20-30ms.

(c) **Geometric verification.** Finally we take the top result estimation and do a full geometric verification on them to give them their proper ranking. After that the match with the highest score is the definitive best match for the query image. This last step duration depends a lot on the number of features found in the corresponding images. Thus in our case it took between 300ms and 2s depending on the query.

Assignment 2: Image classification

November 2, 2016

A. Data preparation and feature extraction

QA1: Why is the spatial tiling used in the histogram image representation?
The main advantage of spatial tiling is that by using a spatial grid to define correspondance we conserve a part of the spatial structure of the image even though we use non-localized vector representation afterwards.

B. Train a classifier for images containing aeroplanes

QB1: Show the ranked images in your report.
Below are the ranked training images for the airplane class.

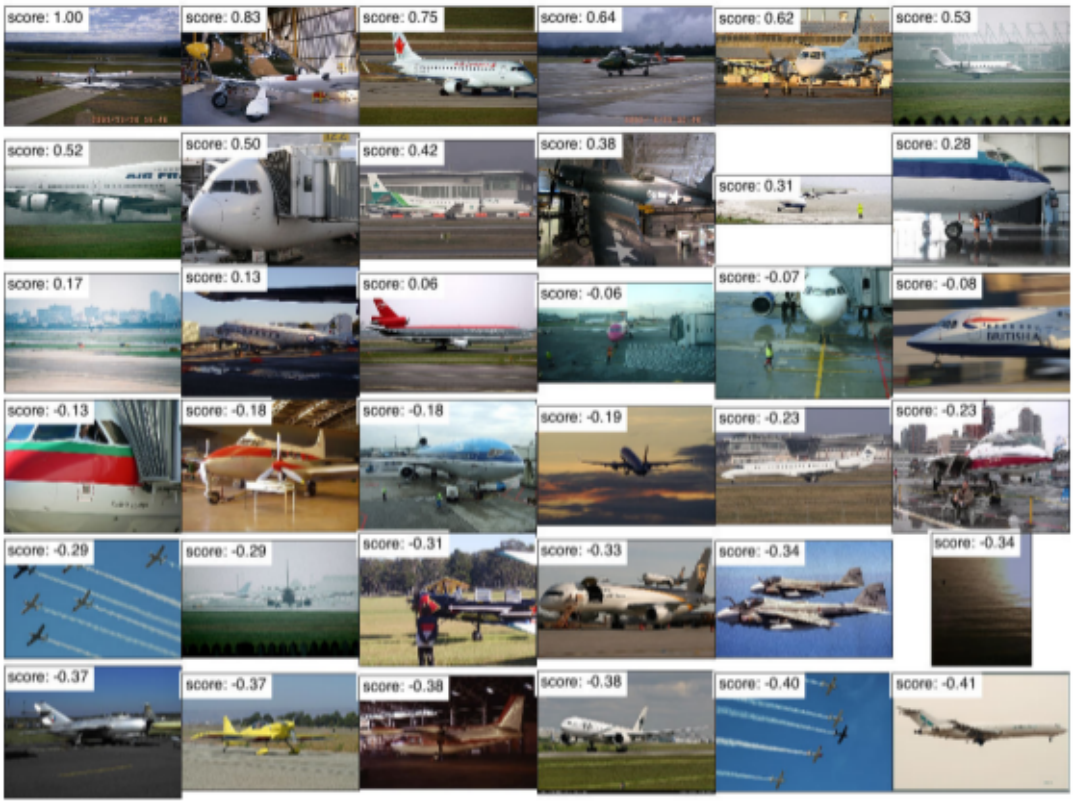

Figure 1: Ranked training images for the airplane class.

QB2: In your report, show relevant patches for the three most relevant visual words (in three separate figures) for the top ranked image. Are the most relevant visual words on the airplane or also appear on background?
The relevant patches for the first three visual words are shown in the picture bellow. We can observe that for the first visual word, a lot are found in the background. On the other hand for the second and third they are pretty well localized on the plane.

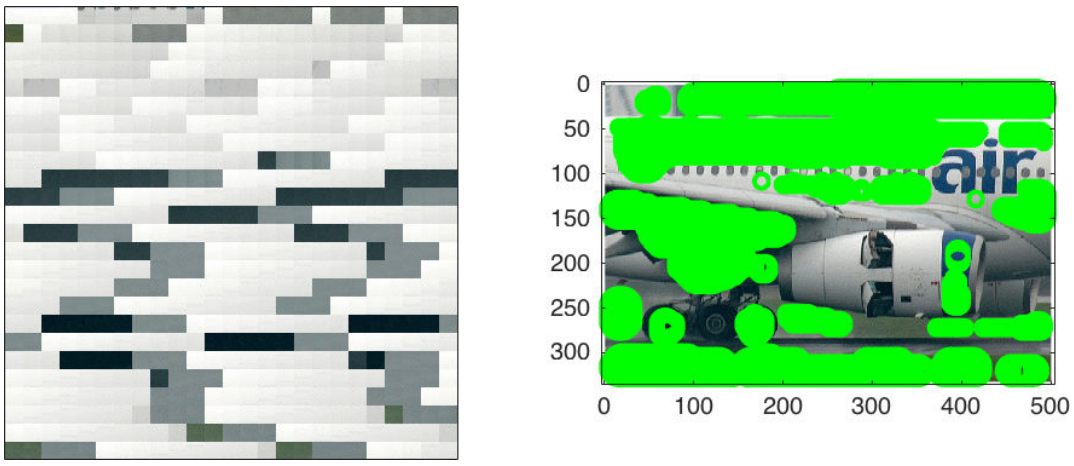

Figure 2: Relevant patches for the first visual word.

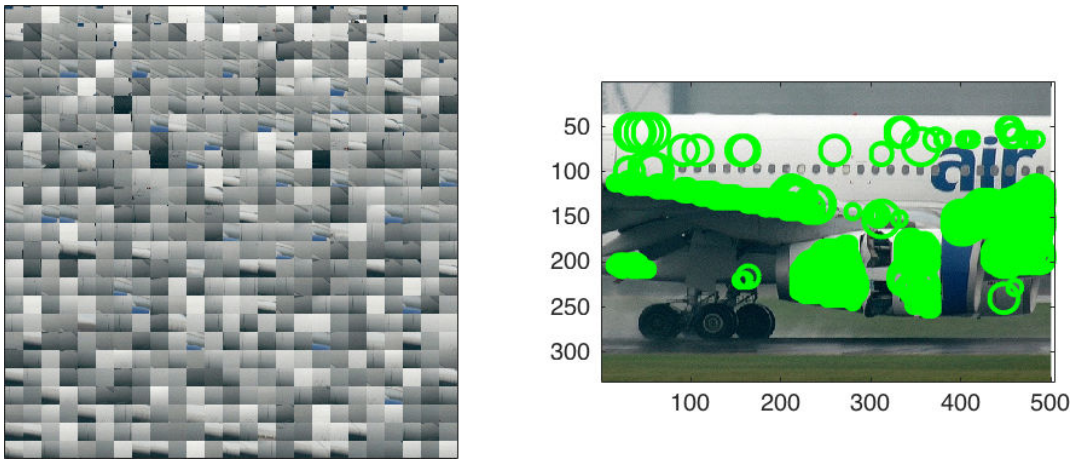

Figure 3: Relevant patches for the second visual word.

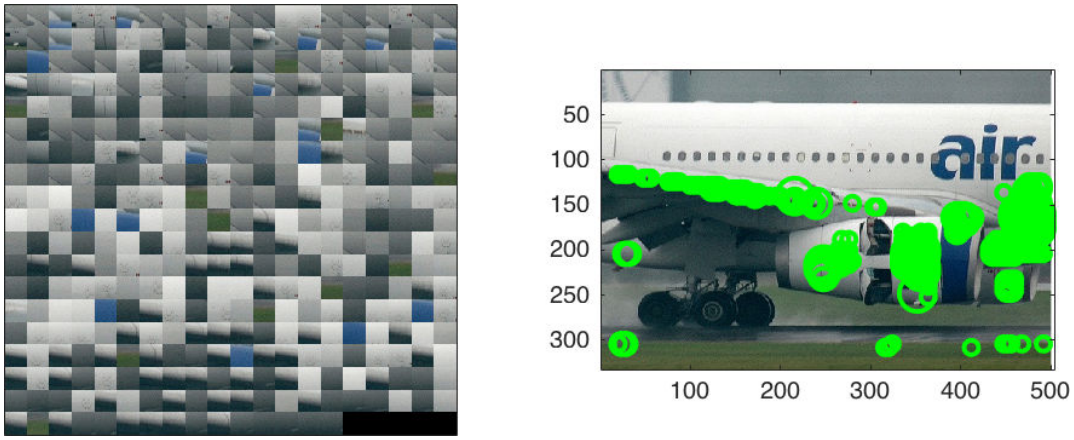

Figure 4: Relevant patches for the third visual word.

C. Classify the test images and assess the performance

QC1: Why is the bias term not needed for the image ranking?
As we just want to be able to compare images, we don't need the exact score, but only to conserve the order between images. Thus any offset doesn't change the ranking and we can omit the bias.

D. Learn a classifier for the other classes and assess its performance

QD1: In your report, show the top ranked images, precision-recall curves and APs for the test data of all the three classes (aeroplanes, motorbikes, and persons). Does the AP performance for the different classes match your expectations based on the variation of the class images?
Below are the ranked images plus the precision recall curve for all three classes: airplane, person and motorbike. We can observe that the AP performance for the person classifier seems to be better but this in part due to the fact there is 10 times person images than motorbike or airplanes.

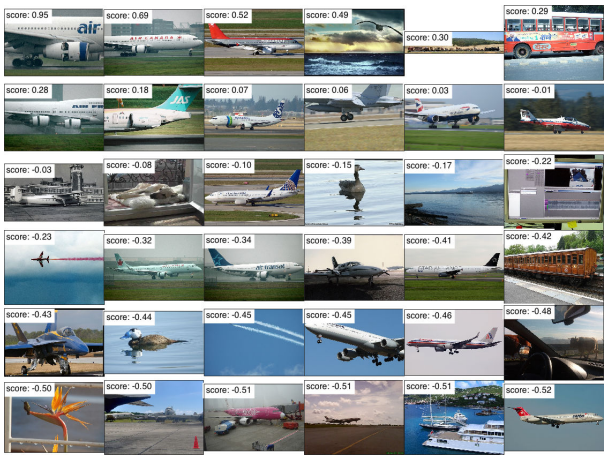

Figure 5: Ranked images for the airplane class.

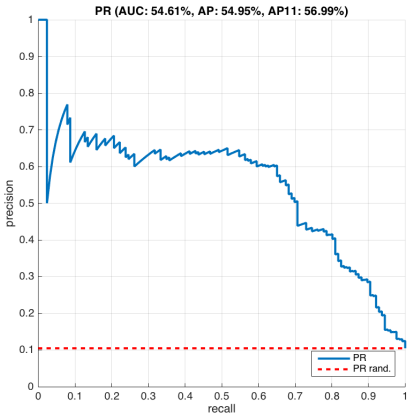

Figure 6: Precision-recall curve of the airplane class.

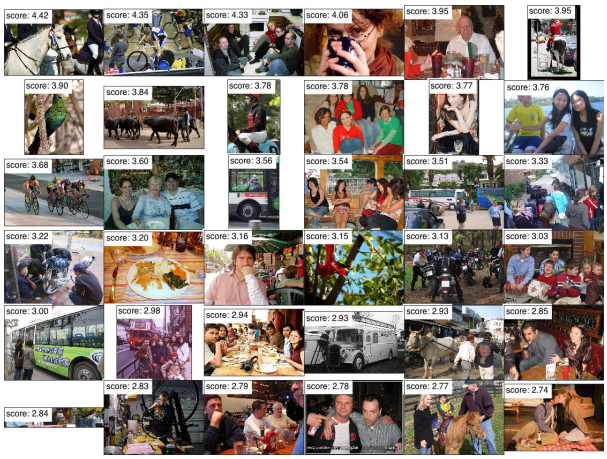

Figure 7: Ranked images for the person class.

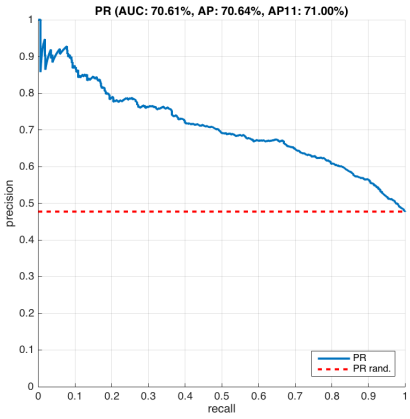

Figure 8: Precision-recall curve of the person class.

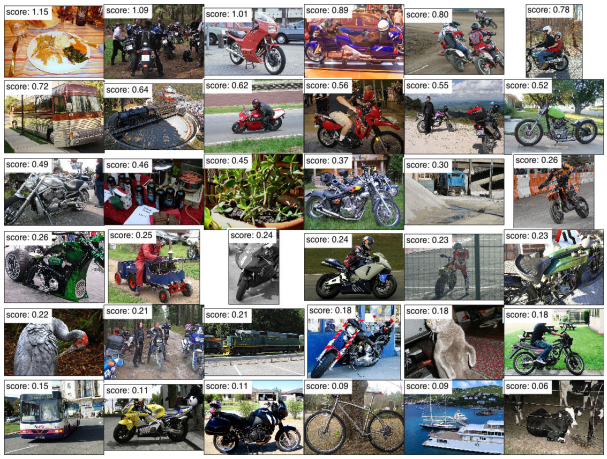

Figure 9: Ranked images for the motorbike class.

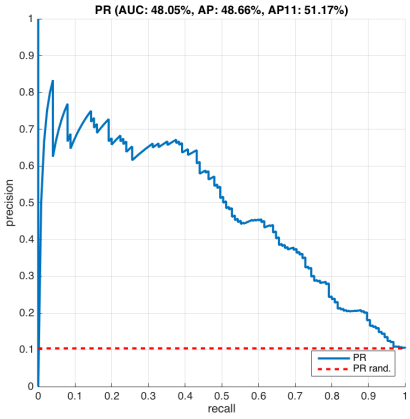

Figure 10: Precision-recall curve of the motorbike class.

QD2: For the motorbike class, give the rank of the first false positive image. What point on the precision-recall curve corresponds to this first false positive image? Give in your report the value of precision and recall for that point on the precision-recall curve.

If we take a closer look at the motorbike classifier we can see that the first returned image is actually not a motorbike. This implies that the first point of the precision-recall curve is the point $(0, 0)$ as can be shown in the Figure 10.

E. Vary the image representation

QE1: Include in your report precision recall-curves and APs, and compare the test performance to the spatially tiled representation in stage D. How is the performance changing? Why?

The precision-recall curves without spatial tiling are presented below. Removing the spatial tiling implies a diminution of the AP between 1 and 7% because we have no more information on the spatial structure of the image.

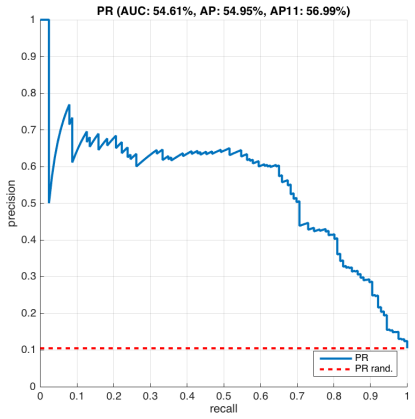

Figure 11: Precision-recall curve of the airplane class without spatial tiling.

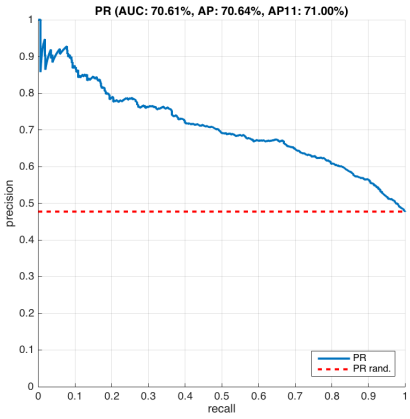

Figure 12: Precision-recall curve of the person class without spatial tiling.

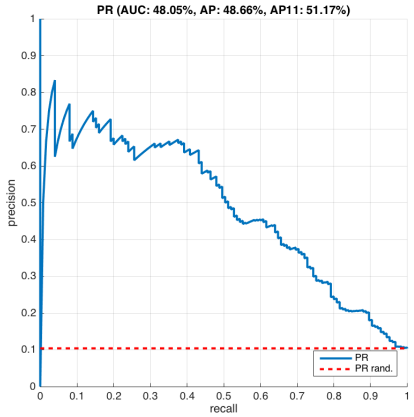

Figure 13: Precision-recall curve of the motorbike class without spatial tiling.

QE2: Modify exercise1.m to use L_1 normalization and no normalization and measure the performance change. For each class we present side by side the precision-recall curve with L_1 normalization and no normalization. The AP for all classes is summarized in the table below.

Normalization	L_2	L_1	none
Airplane	54.95%	51.51%	62.45%
Person	70.64%	56.54%	67.20%
Motorbike	48.66%	26.00%	48.73%

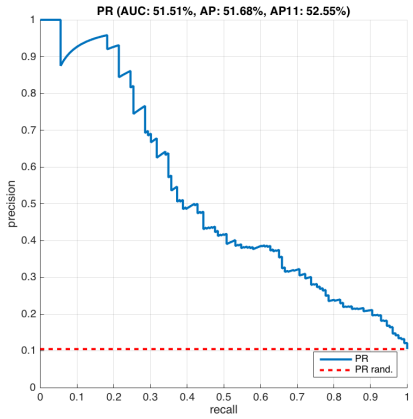

Figure 14: Precision-recall curve of the airplane class with L_1 normalization.

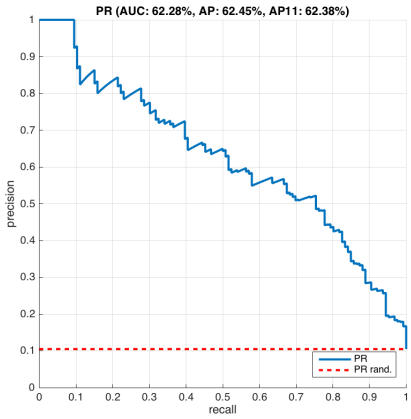

Figure 15: Precision-recall curve of the airplane class without normalization.

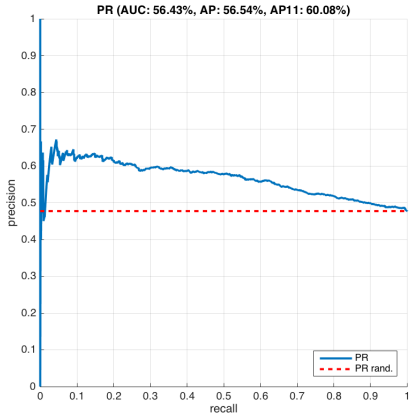

Figure 16: Precision-recall curve of the person class with L_1 normalization.

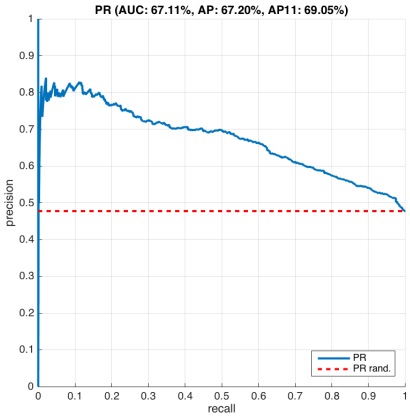

Figure 17: Precision-recall curve of the person class without normalization.

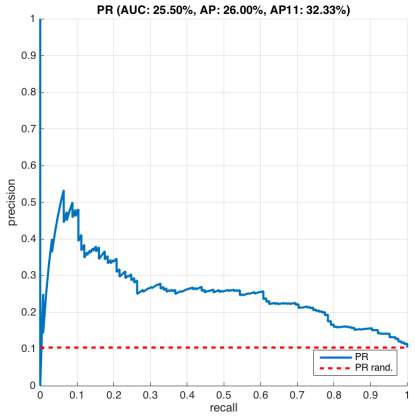

Figure 18: Precision-recall curve of the motorbike class with L_1 normalization.

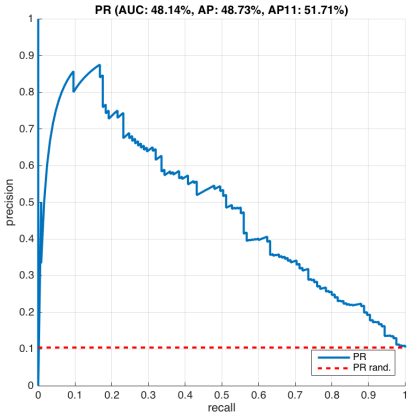

Figure 19: Precision-recall curve of the motorbike class without normalization.

QE3: What can you say about the self-similarity, K(h,h), of a BoVW histogram h that is L_2 normalized? Hint: Compare K(h,h) to the similarity, K(h,h?), of two different L_2 normalized BoVW histograms h and h?. Can you say the same for unnormalized or L_1 normalized histograms?
If h and h' are L_2 normalized then:

$$K(h,h) = \sum_{i=1}^{d} h_i^2 = 1$$

$$K(h,h') = \sum_{i=1}^{d} h_i h'_i \leq \sqrt{K(h,h)}\sqrt{K(h',h')} = 1$$

This relation does not hold for L_1 or unnormalized histograms.

QE4: Do you see a relation between the classification performance and L_2 normalization?
L_2 normalization seems to improve the performance for some classes, such as persons, but sometimes behave poorly compared to other techniques. To draw more precise conclusion we should run this test on several classes.

F. Vary the classifier

QF1: Based on the rule of thumb introduced above, how should the BoVW histograms h and h? be normalized? Should you apply this normalization before or after taking the square root?

We want the vectors that will ultimately be fed to the linear SVM to be L_2 normalized. This means that the vector equal to the square root of h should be L_2 normalized thus h should be L_1 normalized before applying the square root and feeding it into the linear SVM.

QF2: Why is this procedure equivalent to using the Hellinger kernel in the SVM classifier?
The change in feature space before applying the SVM classifier ensure that the SVM remains linear. Besides we have a bijection between our initial feature space with a Hellinger kernel and the new feature space with the linear SVM. Theoretically both are completely identical. Practically there is a big difference in implementation.

QF3: Why is it an advantage to keep the classifier linear, rather than using a non-linear kernel?
Making sure we use a linear SVM classifier yields one big improvement: optimization is much faster. As a consequence operating the feature space change increase the overall speed of the implementation while staying equivalent to the original kernel

QF2: Try the other histogram normalization options and check that your choice yields optimal performance. Summarize your finding in the report (include only mAP results, no need to include the full precision-recall curves).
We compare the Average Precision for the Hellinger kernel compared to all the techniques previously studied. The Hellinger kernel performs better on all classes.

Technique	Hellinger	L_2	L_1	none
Airplane	70.72%	54.95%	51.51%	62.45%
Person	77.39%	70.64%	56.54%	67.20%
Motorbike	63.25%	48.66%	26.00%	48.73%

G. Vary the number of training images

QG1: Report and compare performance you get with the linear kernel and with the Hellinger kernel for the different classes and proportions of training images (10%, 50% and 100%). You don?t have to report the precision-recall curves, just APs are sufficient. Plot the APs for one class into a graph, with AP on the y-axis and the proportion of training images on the x-axis. You can use the matlab function plot Plot three curves (one curve for each class) into one figure. Produce two figures, one for the linear kernel and one for the Hellinger kernel. Make sure to properly label axis (use functions xlabel and ylabel), show each curve in a different color, and have a legend (function legend) in each figure. Show the two figures in your report.

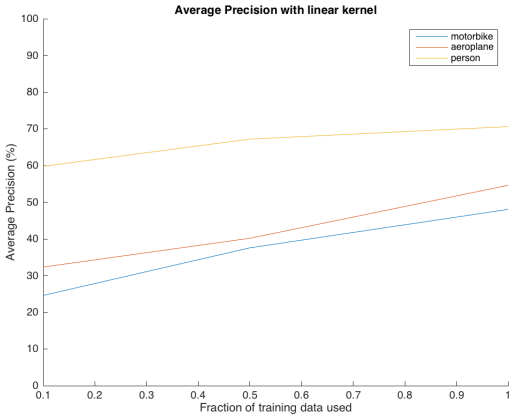

Figure 20: Average precision for the linear kernel depending on the fraction of training samples used.

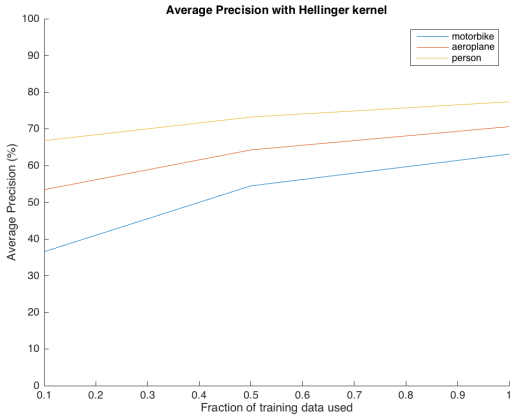

Figure 21: Average precision for the Hellinger kernel depending on the fraction of training samples used.

QG2: By analyzing the two figures, do you think the performance has 'saturated' if all the training images are used, or would adding more training images give an improvement?
Although the improvement in performance might not be as drastic as by going from a tenth to the full dataset, adding more examples should give an improvement to the average precision.

P. Training an Image Classifier for Retrieval using Internet image search

QP1: For the horse class, report the precision at rank-36 for 5 and 10 training images. Show the training images you used. Did the performance of the classifier improve when 10 images were used?
For 5 training images only one image was correctly retrieved in the top 36, by increasing the number of training images to 10, 9 more correctly identified images were found for a total of 10 images. Below are the images used for training the classifier.

Figure 22: First 5 images used to train the horse classifier.

Figure 23: Next 5 images used to train the horse classifier.

QP2: What is the best performance (measured by precision at rank-36) you were able to achieve for the horse and the car class? How many training images did you use? For each of the two classes, show examples of your training images, show the top ranked 36 images, and report the precision at rank-36. Compare the difficulty of of retrieving horses and cars.
By adding 15 more images for a total of 25 images we only achieve to retrieve 13 correct images for the horse classifier. For cars its a bit more tricky, we only achieve 3 correct images using 25 images. See below for top 36 images and sample of car training images.

Figure 24: 5 of the 25 images used to train the car classifier.

Figure 25: Top 36 returned by the horse classifier.

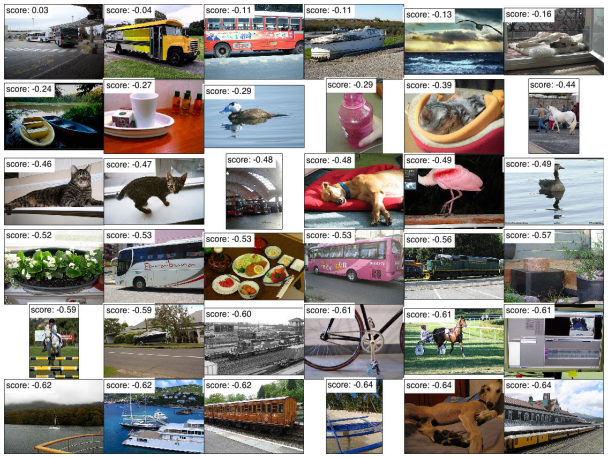

Figure 26: Top 36 returned by the car classifier.

H. First order methods

QH1: Compare the dimension of VLAD and BoVW vectors for a given value of K. What should be the relation of the K in VLAD to the K in BoVW in order to obtain descriptors of the same dimension?
We have the following relation:

$$K_{BoVW} = K_{VLAD} * D_{SIFT}$$

QH2: Replace the encoding used in exercise1 with the VLAD encoding, and repeat the classification experiments for the three classes of Part I (Both linear and Hellinger kernel). How do the results compare to the BoVW encoding? Report mAP results in a table. No need to report all precision-recall curves.
The two new Average Precision curves plus the table summarizing all the results can be found below.

Feature Space	**BoVW**		**VLAD**	
Kernel	Linear	Hellinger	Linear	Hellinger
Airplane	54.61%	70.60%	74.54%	75.48%
Person	70.61%	77.37%	75.52%	78.84%
Motorbike	48.05%	63.11%	68.72%	75.36%

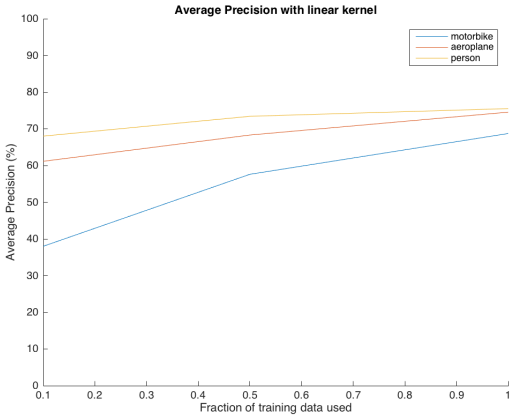

Figure 27: Average precision for the linear kernel using VLAD.

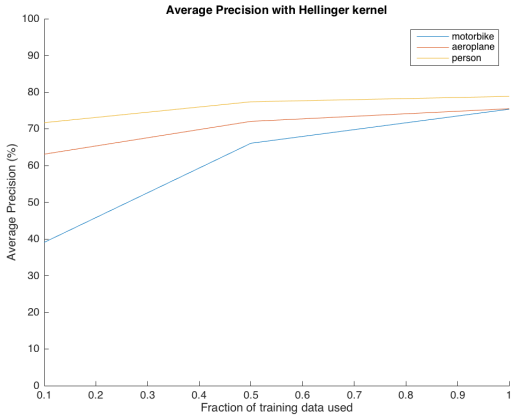

Figure 28: Average precision for the Hellinger kernel using VLAD.

I. Second order methods

QI1: Replace the encoding used in exercise1 with the FV encoding, and repeat the classification experiments for the three classes of Part I. Report the results in the same table as QH2 so that you can see the performance of the three encoding methods side by side.

Feature Space	**BoVW**		**VLAD**		**FV**	
Kernel	Linear	Hellinger	Linear	Hellinger	Linear	Hellinger
Airplane	54.61%	70.60%	74.54%	75.48%	77.42%	82.10%
Person	70.61%	77.37%	75.52%	78.84%	70.49%	78.07%
Motorbike	48.05%	63.11%	68.72%	75.36%	72.59%	81.09%

QI2: What are the advantages or disadvantages of FV compared to VLAD in terms of computation time and storage/memory footprint - especially for a large number (hundreds of millions) of images.
The overall memory footprint of the FV technique is twice bigger than for VLAD, which can lead to severe issues in implementation. In term of computation there are also drawbacks: with VLAD we just need to attribute each SIFT feature to a bin and then compute a running mean of all the residuals. With the Fisher vector we also need to compute the covariance matrices, and even assuming they are diagonal this implies a certain increase in computation operations.

Assignment 3: Neural networks

November 14, 2016

Part 1: Training a neural network by back-propagation

A. Computing gradients of the loss

QA1: In your report, derive using the chain rule the form of the gradient of the logistic loss (4) with respect to the parameters of the network W_i , W_o , B_i and B_o.

We have the following parameters $X \in \mathbb{R}^{2\times 1}, W_i \in \mathbb{R}^{5\times 2}, B_i \in \mathbb{R}^{5\times 1}, H \in \mathbb{R}^{5\times 1}, W_o \in \mathbb{R}^{1\times 5}$ and $B_o \in \mathbb{R}$ that verifies:

$$
\begin{aligned}
H &= ReLU(W_i X + B_i) \\
\overline{Y} &= W_o H + B_o \\
L &= \sum_n s(Y^n, \overline{Y}(X^n)) \\
s(Y, \overline{Y}) &= \log(1 + \exp(-Y\overline{Y}))
\end{aligned}
$$

Denoting w_i^k and b_i^k the k-th row of respectively W_i and B_i and using the chain rule we write:

$$
\begin{aligned}
\frac{\partial s}{\partial W_o} &= \frac{\partial s}{\partial \overline{Y}} \frac{\partial \overline{Y}}{\partial W_o} & \frac{\partial s}{\partial W_o} &= -Y\sigma(-Y\overline{Y})H^\intercal \\
\frac{\partial s}{\partial B_o} &= \frac{\partial s}{\partial \overline{Y}} \frac{\partial \overline{Y}}{\partial B_o} & \frac{\partial s}{\partial B_o} &= -Y\sigma(-Y\overline{Y}) \\
\frac{\partial s}{\partial w_i^k} &= \frac{\partial s}{\partial \overline{Y}} \frac{\partial \overline{Y}}{\partial H} \frac{\partial H}{\partial w_i^k} & \frac{\partial s}{\partial w_i^k} &= \begin{cases} -Y\sigma(-Y\overline{Y})w_o^k X^\intercal & \text{if } w_i^k X + b_i^k > 0 \\ 0 & \text{otherwise} \end{cases} \\
\frac{\partial s}{\partial b_i^k} &= \frac{\partial s}{\partial \overline{Y}} \frac{\partial \overline{Y}}{\partial H} \frac{\partial H}{\partial b_i^k} & \frac{\partial s}{\partial b_i^k} &= \begin{cases} -Y\sigma(-Y\overline{Y})w_o^k & \text{if } w_i^k X + b_i^k > 0 \\ 0 & \text{otherwise} \end{cases}
\end{aligned}
$$

B. Numerically verify the gradients

QB1: In your report, write down the general formula for numerically computing the approximate derivative of the loss $s(\Theta)$, with respect to the parameter Θ_i using finite differencing. Hint: use the first order Taylor expansion of loss $s(\Theta + \Delta\Theta)$ around point Θ.

Using the Taylor expansion we write:

$$
s(\Theta + \Delta\Theta) = s(\Theta) + \sum_k \left(\frac{\partial s}{\partial \Theta_k}(\Theta)\Delta\Theta_k + o(\Delta\Theta_k) \right)
$$

Thus using $\Delta\Theta = \Delta\Theta_i$ we observe that:

$$s(\Theta + \Delta\Theta_i) = s(\Theta) + \frac{\partial s}{\partial \Theta_i}(\Theta)\Delta\Theta_i + o(\Delta\Theta_i)$$
$$\frac{\partial s}{\partial \Theta_i}(\Theta) = \frac{s(\Theta + \Delta\Theta_i) - s(\Theta)}{\Delta\Theta_i} + o(1)$$

Which implies that:

$$\frac{\partial s}{\partial \Theta_i}(\Theta) = \lim_{\Theta_i \to 0} \frac{s(\Theta + \Delta\Theta_i) - s(\Theta)}{\Delta\Theta_i}$$

QB2: In your report, choose a training example (X, Y) and report the values of the analytically computed gradients: `grad_s_Wi`, `grad_s_Wo`, `grad_s_bi`, `grad_s_bo` *as well as their numerically computed counterparts:* `grad_s_Wi_approx`, `grad_s_Wo_approx`, `grad_s_bi_approx`, `grad_s_bo_approx` *Are their values similar?*

Using a precision of `eps = 1e-6` we obtain the numerical values below:

$$\texttt{X} = \begin{pmatrix} 2.0834 \\ 1.2068 \end{pmatrix}$$
$$\texttt{Y} = -1$$

$$\texttt{grad\_s\_Wi} = \begin{pmatrix} 0.2227 & 0.1290 \\ 1.5550 & 0.9007 \\ 1.2155 & 0.7041 \end{pmatrix} \qquad \texttt{grad\_s\_Wi\_approx} = \begin{pmatrix} 0.2227 & 0.1290 \\ 1.5550 & 0.9007 \\ 1.2155 & 0.7041 \end{pmatrix}$$

$$\texttt{grad\_s\_Wo} = \begin{pmatrix} 1.7320 & 2.7319 & 2.4448 \end{pmatrix} \qquad \texttt{grad\_s\_Wo\_approx} = \begin{pmatrix} 1.7320 & 2.7319 & 2.4448 \end{pmatrix}$$

$$\texttt{grad\_s\_bi} = \begin{pmatrix} 0.1069 \\ 0.7464 \\ 0.5834 \end{pmatrix} \qquad \texttt{grad\_s\_bi\_approx} = \begin{pmatrix} 0.1069 \\ 0.7464 \\ 0.5834 \end{pmatrix}$$

$$\texttt{grad\_s\_bo} = 0.9886 \qquad \texttt{grad\_s\_bo\_approx} = 0.9886$$

We also display the norm L_1 of the difference for each parameter in the table below to get a better idea of the precision:

Parameters variation	
$\lVert\texttt{grad\_s\_Wi} - \texttt{grad\_s\_Wi\_approx}\rVert_1$	`9.4728e-08`
$\lVert\texttt{grad\_s\_Wo} - \texttt{grad\_s\_Wo\_approx}\rVert_1$	`3.1116e-08`
$\lVert\texttt{grad\_s\_bi} - \texttt{grad\_s\_bi\_approx}\rVert_1$	`5.3172e-09`
$\lVert\texttt{grad\_s\_bo} - \texttt{grad\_s\_bo\_approx}\rVert_1$	`5.8735e-09`

Therefore we can guarantee that the gradients computed numerically and analytically are equal up to the precision used.

C. Training the network using backpropagation and experimenting

QC1: Include the decision hyper-plane visualization and the training and validation error plots in your report. What are the final training and validation errors? After how many iterations did the network converge?

The final training and validation error are both 0 and they were reached at iteration 14833 for the validation error and iteration 17590 for the training error. Besides the figures representing both error and the separating hyperplan can be found below:

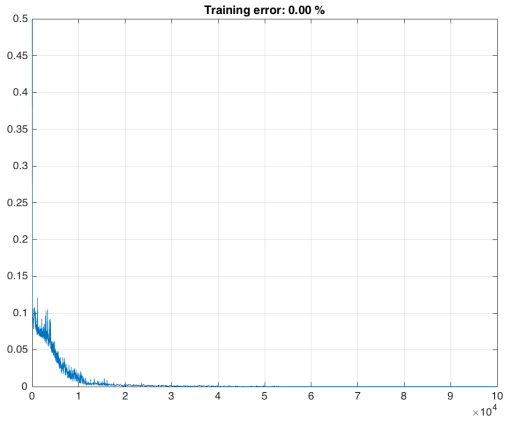

Figure 1: Neural network training error.

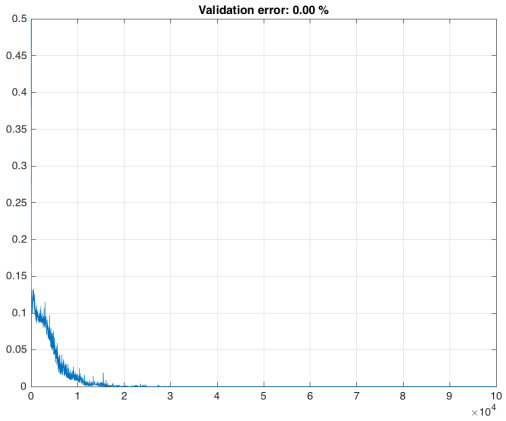

Figure 2: Neural network validation error.

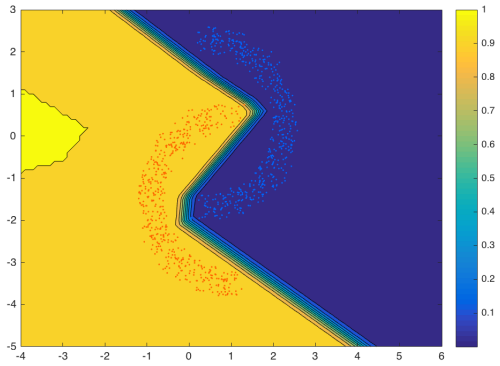

Figure 3: Neural network hyperplane separation.

QC2: Random initializations. Repeat this procedure 5 times from 5 different random initializations. Record for each run the final training and validation errors. Did the network always converge to zero training error? Note: to speed-up the training you can set the variable `visualization_step = 10000` *. This will plot the visualization figures less often and hence will speed-up the training.*

We summarize the final training and validation error for the different runs below:

Run	**Training error (%)**	**Validation error (%)**
1^{st}	0	0
2^{nd}	14.1	12.5
3^{rd}	0	0
4^{th}	0	0
5^{th}	6.1	7.6

QC3: Learning rate. Keep h=7 and change the learning rate to values lrate = 2, 0.2, 0.02, 0.002. For each of these values run the training procedure 3 times and observe the training behaviour. For each run and the value of the learning rate report: the final (i) training and (ii) validation errors, and (iii) after how many iterations the network converged (if at all). Visualize the decision hyperplane and the evolution of the training error for each value of the learning (here only one of the three runs is sufficient). Briefly discuss the different behaviour of the training for different learning rates.

We start by running the algorithm for the different values of `lrate`:

(a) `lrate = 2`

`lrate = 2`			
Run	**Training error (%)**	**Validation error (%)**	**Iterations**
1^{st}	39.5	41.1	–
2^{nd}	50.0	50.0	–
3^{rd}	41.3	42.0	–

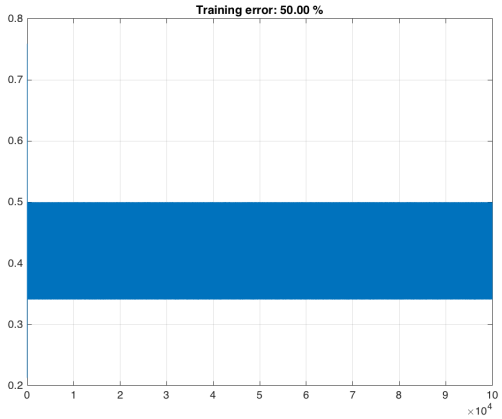

Figure 4: `lrate = 2` training error.

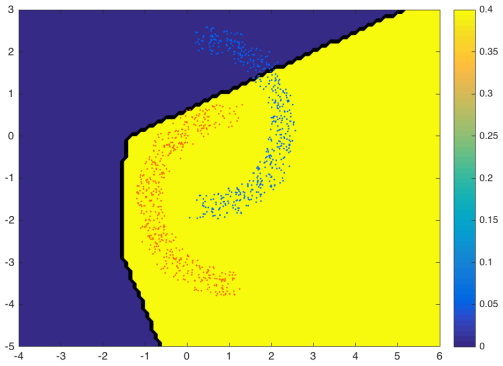

Figure 5: `lrate = 2` hyperplane separation.

(b) `lrate = 0.2`

`lrate = 0.2`			
Run	**Training error (%)**	**Validation error (%)**	**Iterations**
1^{st}	12.6	13.2	–
2^{nd}	0	0	12210
3^{rd}	0	0	3994

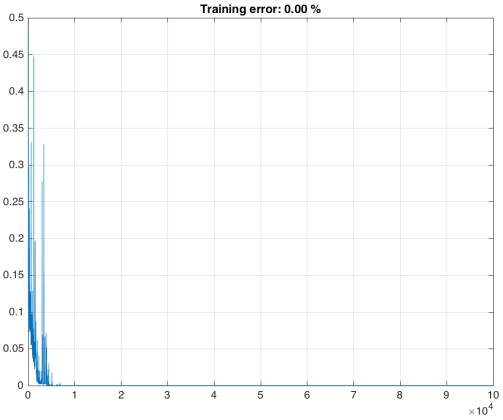

Figure 6: `lrate = 0.2` training error.

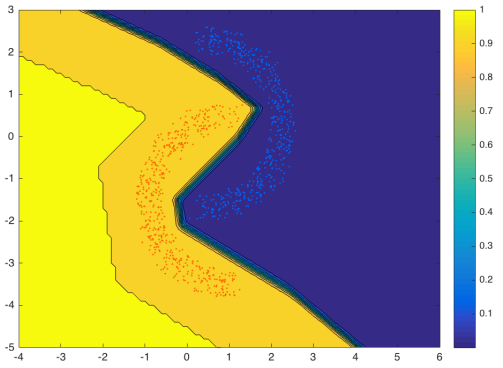

Figure 7: `lrate = 0.2` hyperplane separation.

(c) `lrate = 0.02`

`lrate = 0.02`			
Run	**Training error (%)**	**Validation error (%)**	**Iterations**
1^{st}	0	0	23592
2^{nd}	0	0	25134
3^{rd}	0	0	18563

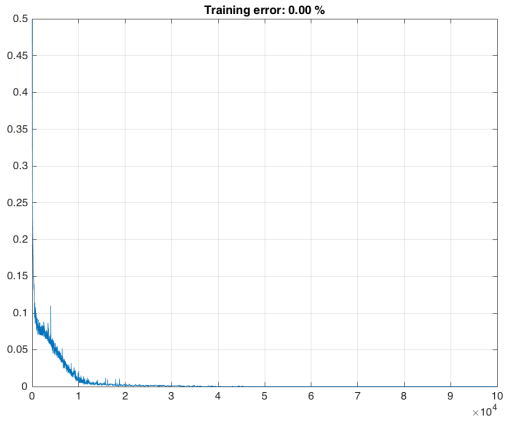

Figure 8: `lrate = 0.02` training error.

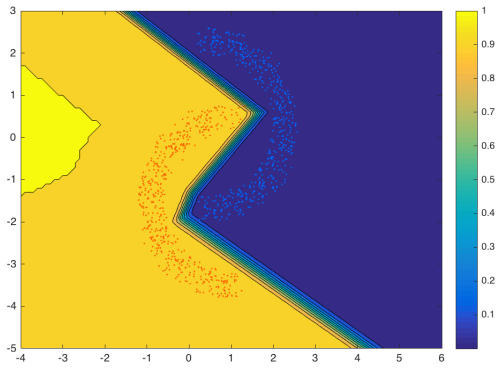

Figure 9: `lrate = 0.02` hyperplane separation.

(d) `lrate = 0.002`

`lrate = 0.002`			
Run	**Training error (%)**	**Validation error (%)**	**Iterations**
1^{st}	0.6	0.8	–
2^{nd}	0.6	0.7	–
3^{rd}	0.4	0.3	–

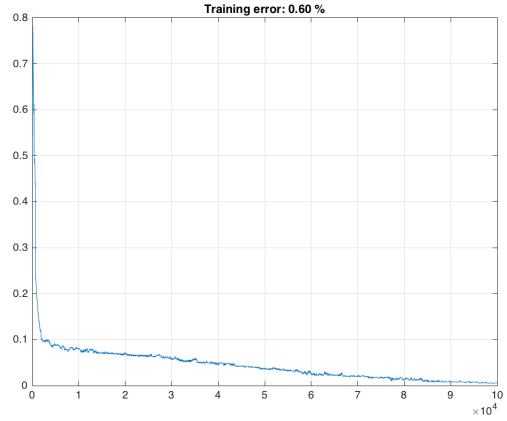

Figure 10: `lrate = 0.002` training error.

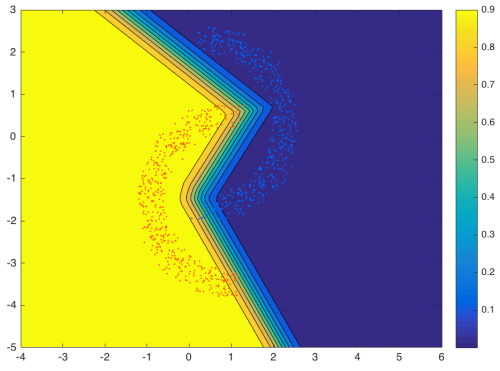

Figure 11: `lrate = 0.002` hyperplane separation.

From this we can infer 4 different regimens (although we would need more data to draw certain conclusion);

(a) `lrate` is very large: oscillations appear and the algorithm doesn't converge.

(b) `lrate` is upper middle: the algorithm behavior is unstable, it can either converge very fast or not converge at all.

(c) `lrate` is lower middle: it takes more step to converge but the algorithm seems to converge most of the time.

(d) `lrate` is very small: the steps are too small and thus the algorithm needs more steps than are allowed to converge although given more time it seems to be in the right direction.

In this example the case (c) with `lrate = 0.02` seems like a good compromise between convergence and iterations.

QC4: The number of hidden units. Set the learning rate to 0.02 and change the number of hidden units h = 1, 2, 5, 7, 10, 100. For each of these values run the training procedure 3 times and observe the training behaviour.

For each run and the value of the number of hidden units record and report: the value of the final (i) training and (ii) validation error, and (iii) after how many iterations the network converged (if at all). Show the plot of the final decision hyperplane for each value of h (only one of the three plots for each h is sufficient). Discuss the different behaviours for the different numbers of hidden units.

We start by running the algorithm for the different values of `h`:

(a) `h = 1`

`h = 1`			
Run	**Training error (%)**	**Validation error (%)**	**Iterations**
1^{st}	8.9	11.5	–
2^{nd}	8.3	10.6	–
3^{rd}	8.0	9.5	–

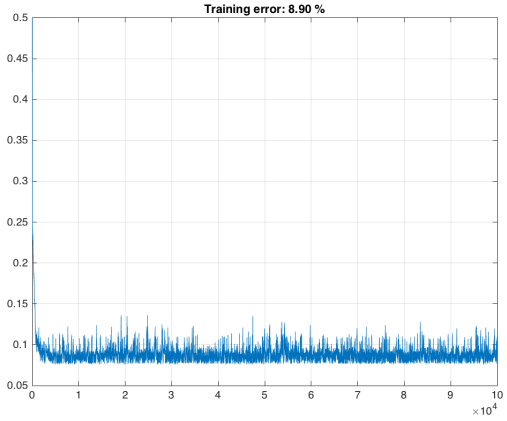

Figure 12: `h = 1` training error.

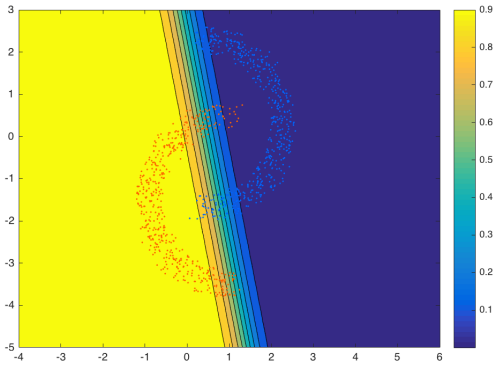

Figure 13: `h = 1` hyperplane separation.

(b) `h = 2`

`h = 2`			
Run	**Training error (%)**	**Validation error (%)**	**Iterations**
1^{st}	9.1	10.2	–
2^{nd}	7.7	9.6	–
3^{rd}	7.1	9.3	–

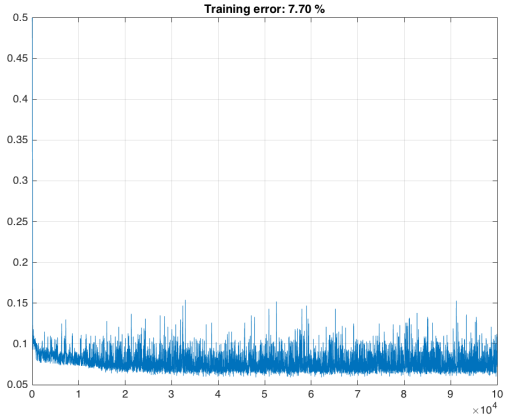

Figure 14: `h = 2` training error.

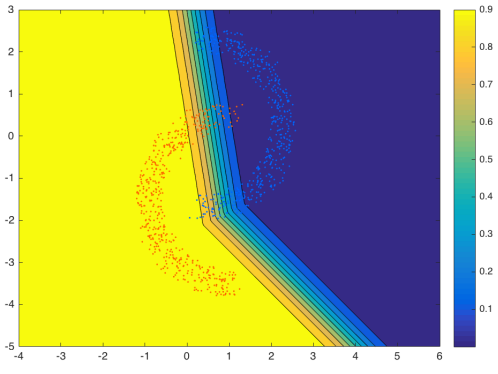

Figure 15: `h = 2` hyperplane separation.

(c) `h = 5`

`h = 5`			
Run	**Training error (%)**	**Validation error (%)**	**Iterations**
1^{st}	0	0	22596
2^{nd}	7.0	9.2	–
3^{rd}	0	0	19200

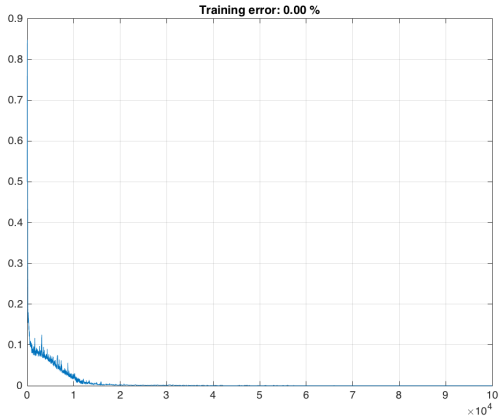

Figure 16: `h = 5` training error.

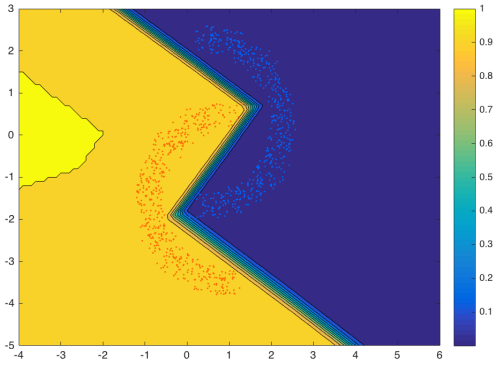

Figure 17: `h = 5` hyperplane separation.

(d) `h = 7`

`h = 7`			
Run	**Training error (%)**	**Validation error (%)**	**Iterations**
1^{st}	0	0	24816
2^{nd}	6.8	8.0	–
3^{rd}	0	0	19605

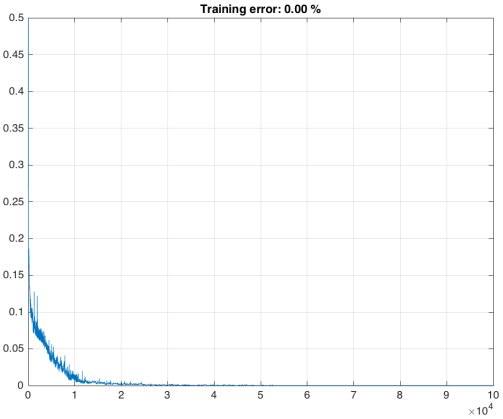

Figure 18: `h = 7` training error.

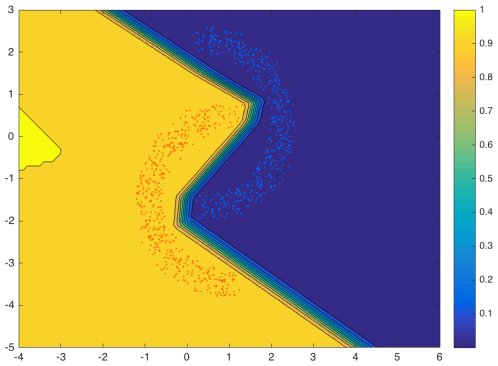

Figure 19: `h = 7` hyperplane separation.

(e) `h = 10`

`h = 10`			
Run	**Training error (%)**	**Validation error (%)**	**Iterations**
1^{st}	0	0	22327
2^{nd}	0	0	22519
3^{rd}	7.7	9.9	–

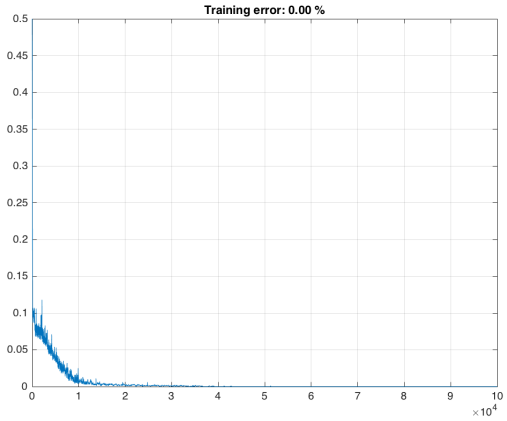

Figure 20: `h = 10` training error.

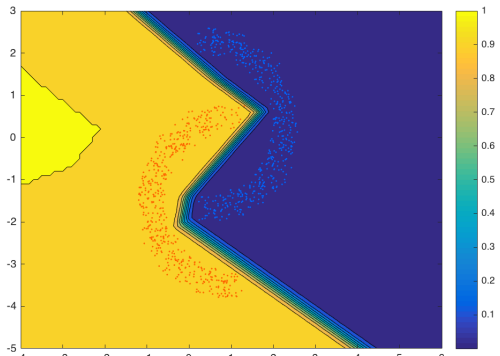

Figure 21: `h = 10` hyperplane separation.

(f) `h = 100`

`h = 100`			
Run	**Training error (%)**	**Validation error (%)**	**Iterations**
1^{st}	0	0	13417
2^{nd}	0	0	16331
3^{rd}	0	0	16697

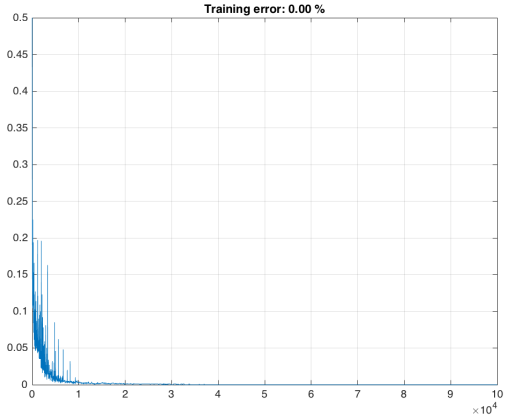

Figure 22: `h = 100` training error.

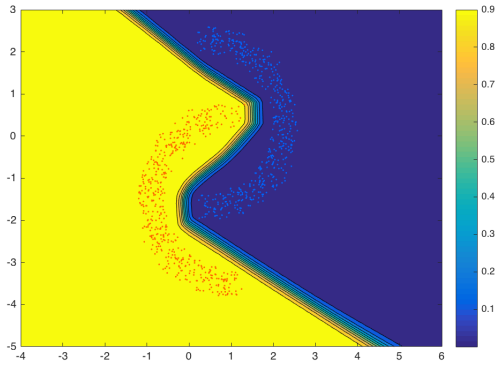

Figure 23: `h = 100` hyperplane separation.

From this it would seem that:

(a) `h` is too small: the network and too simple and can't capture the complexity of the data.

(b) `h` is middle: depending on the initialization the algorithm either converge or doesn't but the complexity of the data and the model seem to be pretty similar.

(c) `h` is very big: the network complexity is way higher than the data complexity. This doesn't induce better error result because it was already captured. On the other hand fewer iterations are needed but they need a longer time to compute. Overall this is not an efficient solution.

In this example the case (b) with `h` $\sim$ `7` seems like a good compromise between convergence and iterations.

Part 2: Image classification with CNN features

A. Computing CNN features

Q2A1: What is done by the above normalization code and why is it needed?

The normalization code consists of 5 steps that are detailed below:

(1) `net = load('imagenet-vgg-f.mat');`
Load the pre-trained neural network model in Matlab.

(2) `im = imread('data/images/000005.jpg');`
Retrieve the raw image using its filename and path.

(3) `im_ = single(im);`
Convert the image from `uint8` to `single`.

(4) `im_ = imresize(im_, net.meta.normalization.imageSize(1:2));`
Resize the image so that it's compatible with the neural network input.

(5) `im_ = im_ - net.meta.normalization.averageImage;`
Normalize the image with the same value used during the neural network training. This is done to ensure that the new image is comparable to the images used during training.

Q2A2: Look at the content of variables `net` *and* `res`*. What is their structure? What is encoded by* `net.layers{k}` *for different values of* `k`*? What do values in* `res(k).x` *represent for different values of* `k`*? For hints refer to http://www.vlfeat.org/matconvnet/matconvnet-manual.pdf*

The variable `net` corresponds to the pre-trained convolutional neural network. It is structured with a `meta` part regrouping normalization parameters, input information or classes and a second `layers` section that identify each neural network layer with its name, function, parameter value, etc.
On the other hand `res` contains the value of the different layers of the neural computed using the image `im_` during the forward pass.

B. Image classification using CNN features and linear SVM

Q2B1: Report your classification results for the three object classes by including precision-recall plots and AP values into the report. What CNN layers work best as features? Are feature normalization and SVM cross-validation important steps to obtain best results? Motivate your answer by performance numbers. ImageNet dataset contains no person class. Can you connect this fact to the difference in the performance of your person classifiers trained/tested using different layers of CNN output?

The values of the average precision for different object classes, feature space, normalization and SVM regularization parameter C are summarized in the synthetic plots below. Also we plot the precision-recall for the best feature space, norm and SVM parameter C for each object class. From this plots we can infer several observations:

(a) Fine-tuning of the SVM C parameter is crucial as it can drastically change the average precision.

(b) Feature normalization also plays a very important role and overall the L_2 norm seems to be the best candidate for this particular application.

(c) The `fc8` layer seems to be the CNN layer that yields the best result. This might be due to the fact that the `fc8` has a higher level feature representation than the `fc7` while still not being a soft decision on the class like the final softmax `prob` layer.

(d) Although the network was not trained to recognize persons (as there are no person class in the ImageNet dataset0, the performance of our classifier is surprisingly good. This could mean that the features learned by the neural network are actually partly independent from the class which basically means that they are good descriptors no matter what class they apply to.

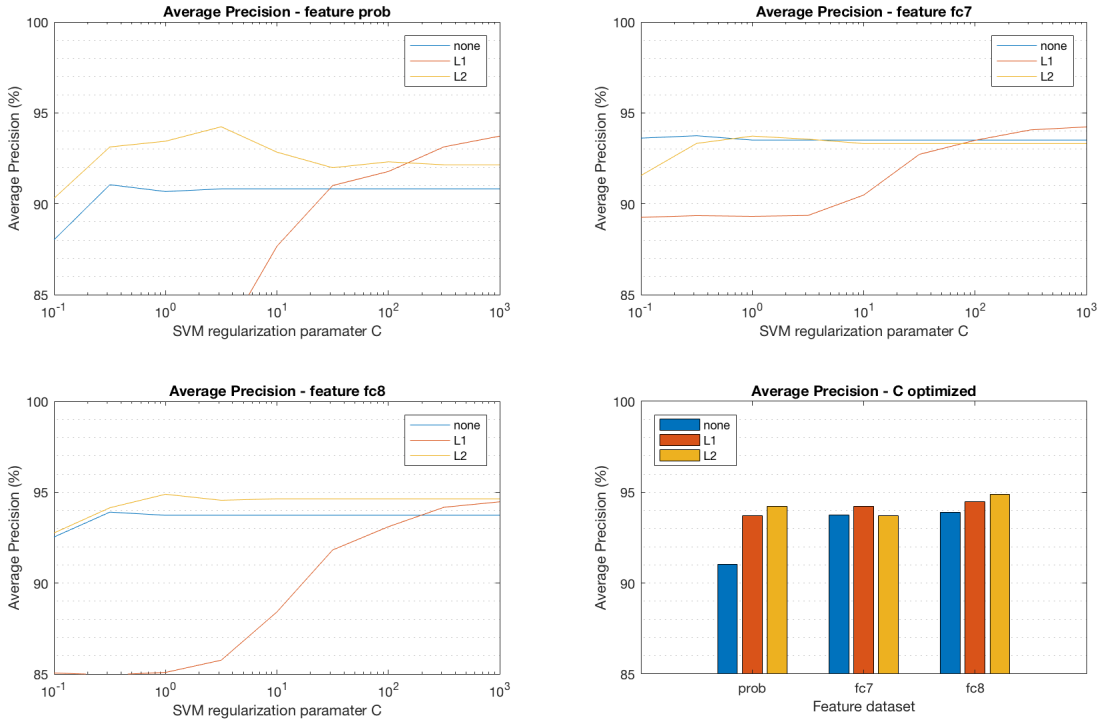

Figure 24: Aeroplane object class.

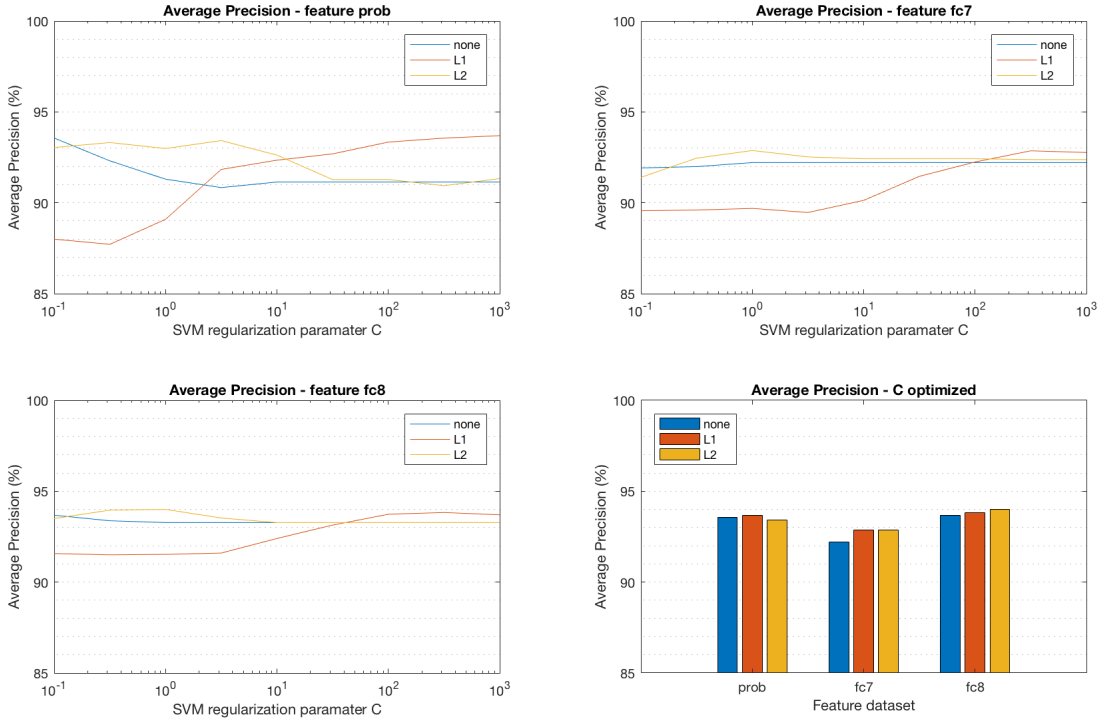

Figure 25: Motorbike object class.

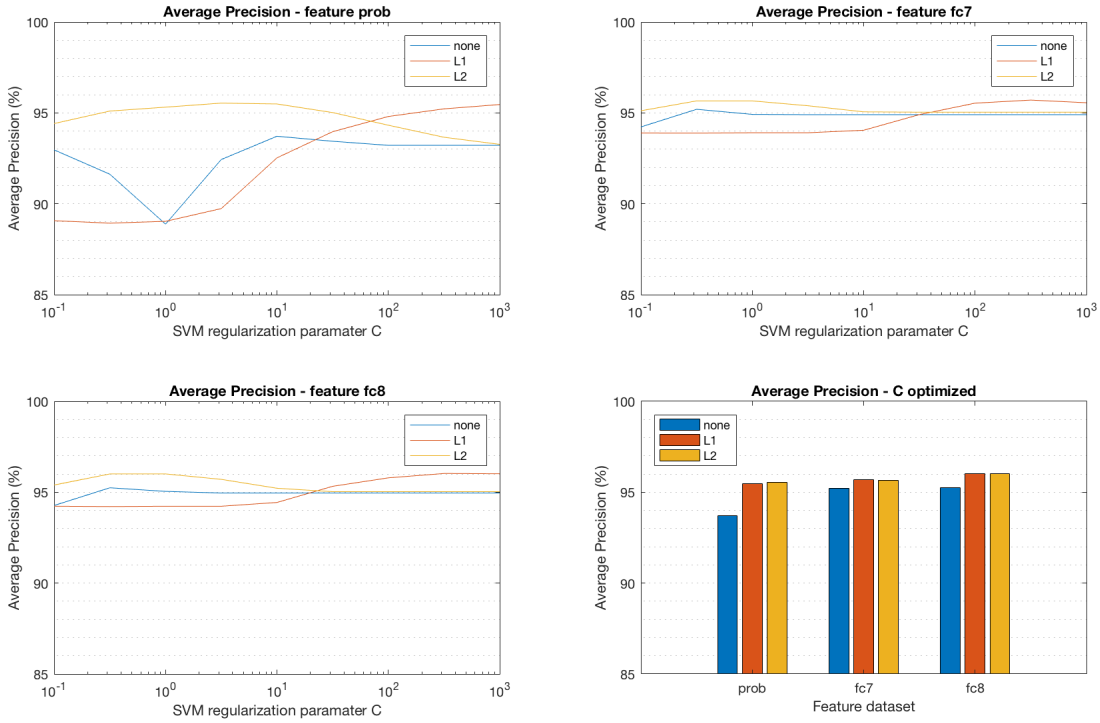

Figure 26: Person object class.

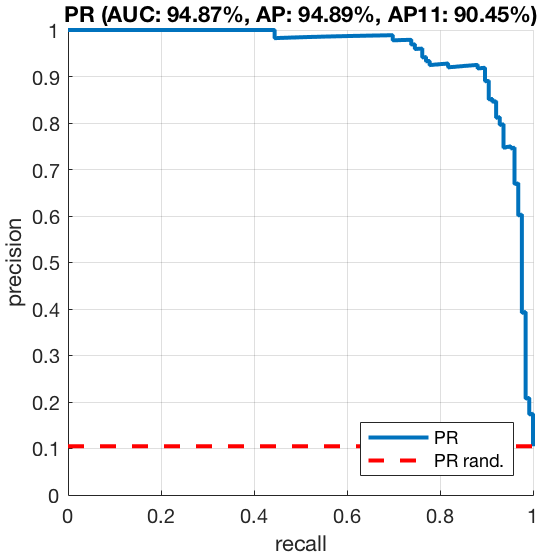

Figure 27: Precision recall of aeroplane class (best).

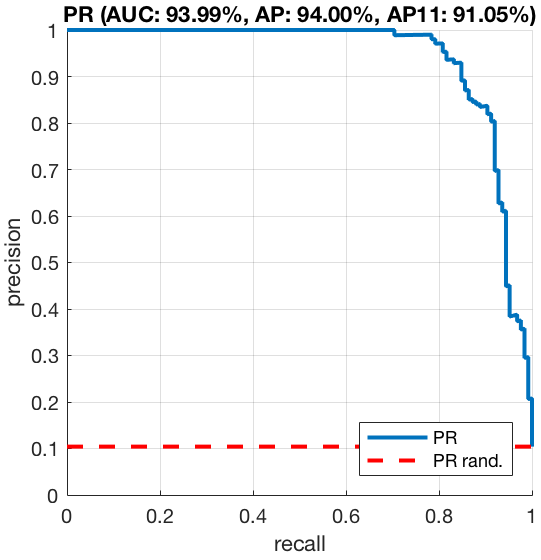

Figure 28: Precision recall of motorbike class (best).

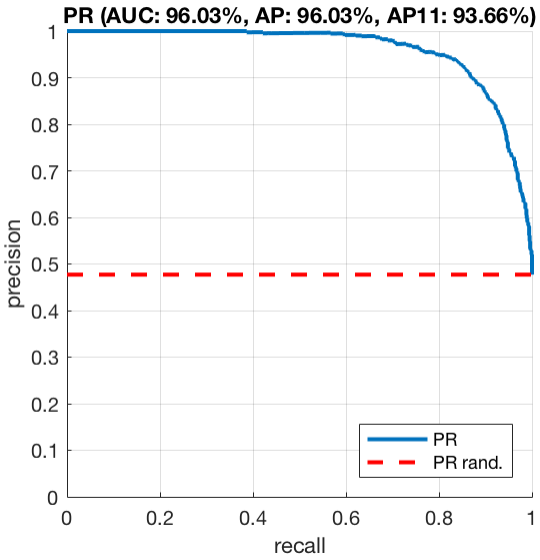

Figure 29: Precision recall of person class (best).

Q2B2: Compare your best results using CNN features to your results obtained in Assignment 2 for corresponding object classes and training/testing image subsets. What conclusions can you draw? Illustrate a few negative test samples with the highest classification scores (false positives) and a few positive test samples with the lowest classification scores (false negatives) for each class. Can you explain the errors of the classifier?

The best results for each class are summarized in the table below, we can see that the SVM trained with the output of the CNN has way better results than the methods tried in assignment 2.

	Assignment 2		**Assignment 3**	
class	**precision**	**method**	**precision**	**method**
aeroplane	82.10%	FV + Hellinger	94.89%	fc8 + L_2
person	78.84%	VLAD + Hellinger	96.03%	fc8 + L_2
motorbike	81.09%	FV + Hellinger	94.00%	fc8 + L_2

Below we show the top (up to 3 if there are that many available) false positive and false negative for each class.

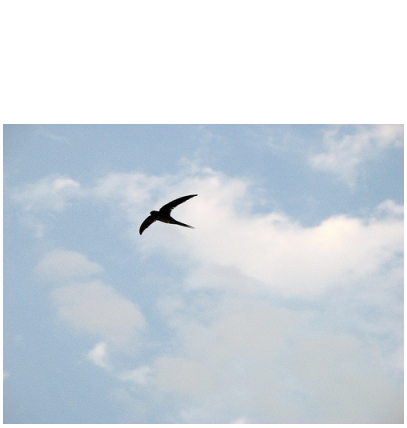

Figure 30: Top false positive for the aeroplane class.

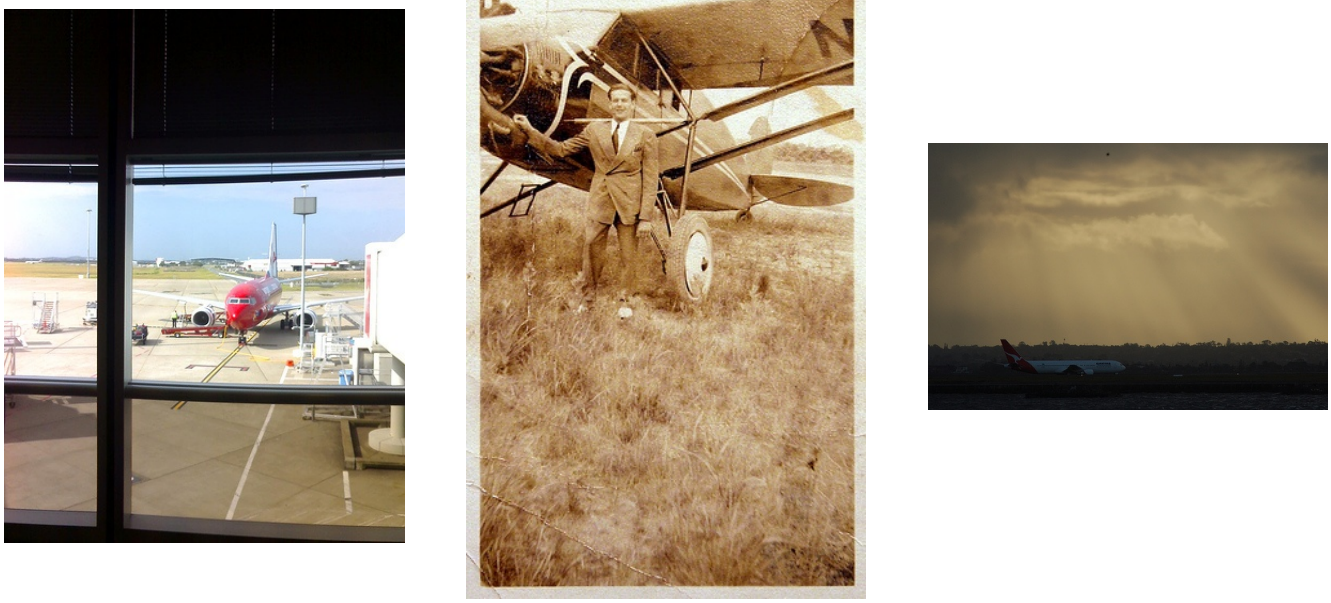

Figure 31: Top false negative for the aeroplane class.

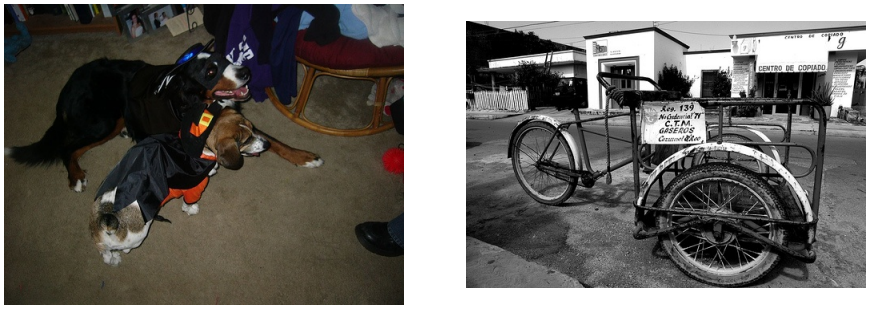

Figure 32: Top false positive for the motorbike class.

Figure 33: Top false negative for the motorbike class.

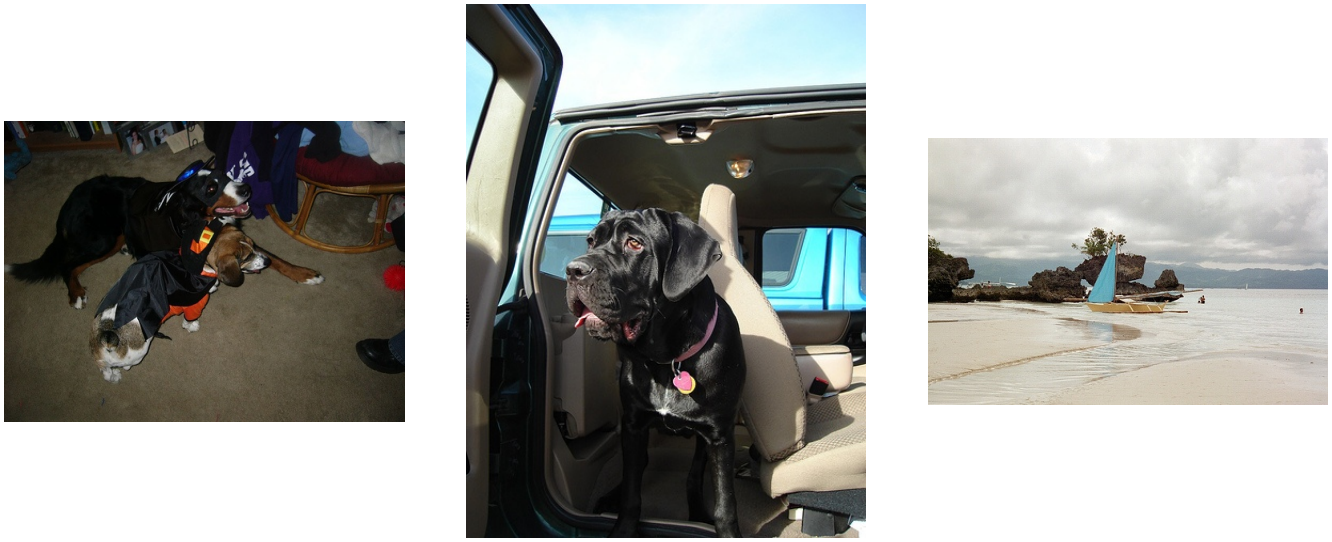

Figure 34: Top false positive for the person class.

Figure 35: Top false negative for the person class.

Overall top false positives seem to be object that are very similar, a bird and a plane, a bike and motorbike, etc. On the other hand in the false negatives image although the object is present it is not the main focus of the picture, for instance a passenger head in a train. Basically sometime it is very subjective to say that an image belongs to a class or an other because multiple classes are present in the same image.