IMAGE CLASSIFICATION

Task 1: Bag of Words

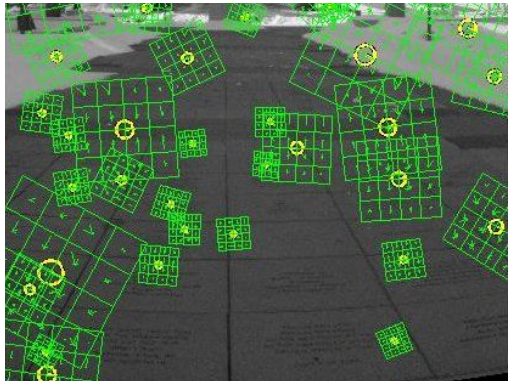
Team 5: Marc Artero Pons, Diego Hernández Antón, Bernat Medina Pérez and Pau Monserrat Llabrés

1. Local descriptors

Local descriptors

Local descriptors are numerical representations of small patches of an image around a specific keypoint. These numerical representations are the local **features** of the image. With them, it is possible to generate the codebook, necessary for the Bag of Visual Words method (BoVW). Hence, the first step for our image classification pipeline is to decide the local descriptor(s) to be used.

The ones we are going to test are SIFT, ORB, AKAZE and Dense SIFT. A further study of them is necessary to know its strengths and weaknesses.



Local descriptors

There are two different types of descriptors:

- SIFT, ORB and AKAZE are **sparse descriptors**. They detect points of interest (blobs, corners and edges) on the image to compute the local features on a patch around them.
  - SIFT is highly robust, but computationally expensive.
  - AKAZE has a good balance between speed and accuracy.
  - ORB is extremely fast, but less robust than the other two.
- Dense SIFT is a **dense descriptor**. It computes the standard SIFT descriptor at regular, fixed intervals (dense grid) across the image.
  - It computes local features over the entire image, deriving in a representation of the whole image content.

What about the descriptors' invariance?

	Scaling	Illumination	Rotation
ORB	✔	–	✔
AKAZE	✔	✔	✔
SIFT	✔	✔	✔
Dense SIFT	✗	✔	✗

Sparse descriptors are better suited for matching tasks, while dense descriptors are usually better for classification tasks.

Local descriptors

For our experimental setup, the hyperparameters of each descriptor are the default ones. Furthermore, to ensure the difference in the results comes from the descriptor used, we fix the rest of the configuration:

- **Classifier**: Logistic Regression with C = 1 (C is the inverse of the regularization strength).
- **Codebook size** (number of visual words): 100.
- **Number of features**: 1024 (not possible for AKAZE, since the number is automatically determined).
- We normalize the histogram of the visual words by its L2 norm (histogram normalization).

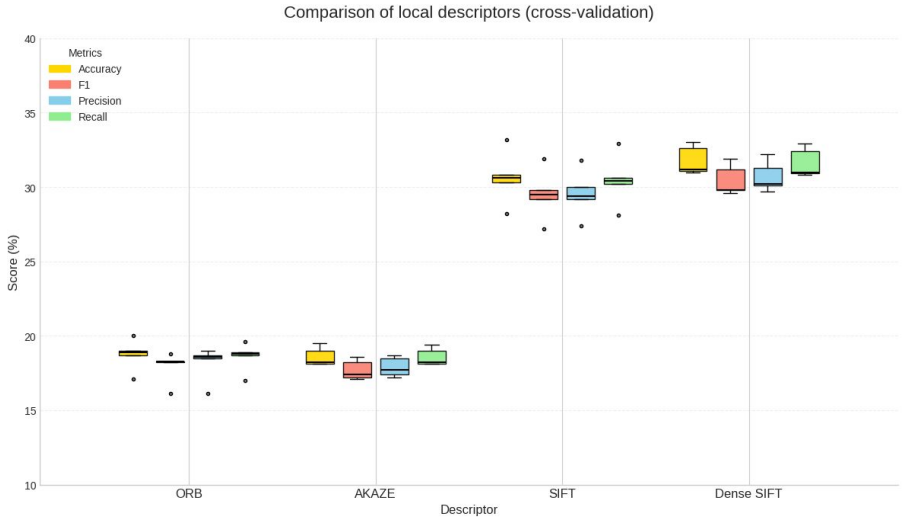
Our hypotheses for this experiment are the following:

➢ **Hypothesis 1:** Dense SIFT is the best descriptor due to its dense nature, it is suited for image classification and captures information from smooth areas.
➢ **Hypothesis 2:** Based on the previous properties discussed, the best-performing sparse descriptor is SIFT, followed by AKAZE.

The experiment (and the following ones) uses **cross-validation** with **5 folds**.

Local descriptors

- The performance of ORB and AKAZE is similar. The same happens with SIFT and Dense SIFT.
- Dense SIFT improves AKAZE's accuracy by around 64%.



Local descriptors

- **HYPOTHESIS 1:** ✔

Contrary to our expectations, Dense SIFT and SIFT perform similarly. This leads us to conclude that the feature information of the keypoints detected by SIFT is sufficient for accurate image characterisation in our context. Hence, using Dense SIFT is not as beneficial as we initially thought. However, it improves SIFT's performance by ~4%, and their results have lower variance, making it a more confident choice.

- **HYPOTHESIS 2:** ✔

As we expected, SIFT outperforms both ORB and AKAZE. However, AKAZE and ORB have similar performance, so we cannot effectively conclude one is better than the other.

Based on the results and the fact we were recommended to use it in the project presentation session, we select Dense SIFT for the rest of our experiments.

Local descriptors - Number of features

In order to determine the importance of the number of detected keypoints in classifying an image, we fix the other hyperparameters as described in the last experiment.

We study their impact using both SIFT and Dense SIFT. The number of features in SIFT is controlled by the *nfeatures* parameter, which is an upper bound. Less keypoints are returned if there are large smooth areas.

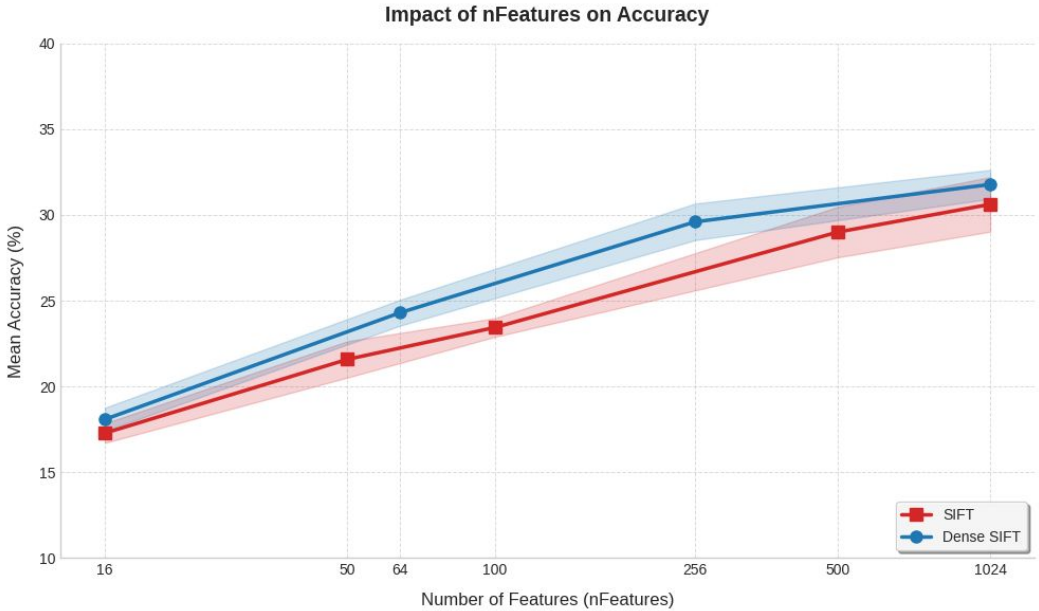
Conversely, the number of keypoints detected by Dense SIFT is always the same for all the images and it is determined by our *stride* parameter. Since all the images in the dataset are of 256x256 pixels. The total number of detected keypoints for each one of them is (256/*stride*) x (256/*stride*).

- ➢ **Hypothesis**: We believe that accuracy is directly proportional to the number of features detected: the more keypoints we have, the better the image description is.

Local descriptors - Number of features

- **HYPOTHESIS:** ✔

Accuracy is directly proportional to the number of features used in the descriptor. We can also state that Dense SIFT is slightly better than SIFT for any given number of features (at least for the tested range).



stride values tested for Dense SIFT: [8, 16, 32, 64]

* Each dot on the curves corresponds to a specific value of number of features tested.

2. Codebook

Codebook - Number of words

The codebook is the core of the BoVW method, since it determines the vocabulary of visual words used to classify images: after clustering the whole set of features of the train set, each cluster defines a visual word. Therefore, it is extremely important to define the number of visual words in the vocabulary.

We test the codebook with 4, 8, 16, 32, 64, 128, 256, 512, 1024, 2048 number of words (powers of 2).

➢ **Hypothesis:** A big number of words unnecessarily increases the number of features, and the model is likely to overfit. In contrast, models with a low number of words will lack expressivity, and will underfit.

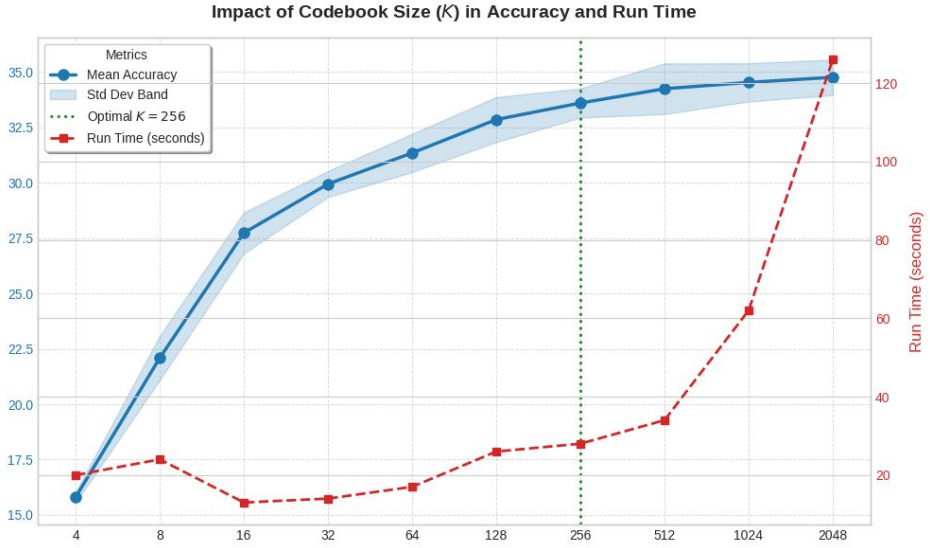
For the experimental setup, we fix the following parameters:

- **Local descriptor:** Dense SIFT with *stride* = 8 and *scale* = 2.
- **Classifier:** Logistic Regression with C = 1.
- Histogram normalization.

Codebook - Number of words

- **HYPOTHESIS: ✗**

Performance grows as the codebook size is increased. Despite that, the amount of computation needed for the larger number of words does not provide a proportional benefit in accuracy. Therefore, for us, the optimal codebook size is 256: it achieves 36% of accuracy in only 26 seconds for training/testing the model.



Efficiency trade-off: Large codebooks (K>256) give better accuracy, but with an unnecessary increase in computation.

Conclusion: We select K=256 as the codebook size for the next experiments.

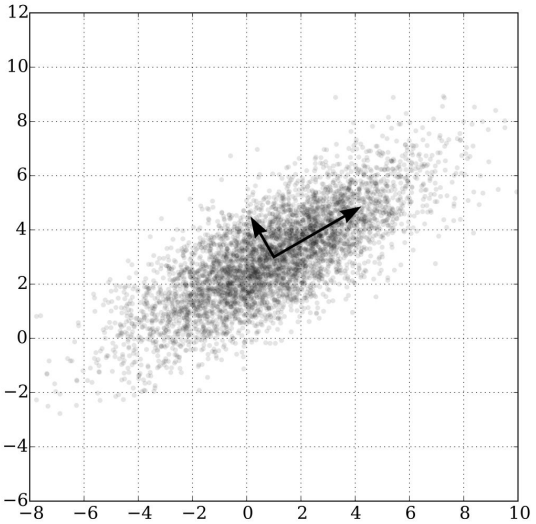
Codebook - PCA

The descriptors computed by Dense SIFT have 128 dimensions. We can reduce them using PCA, which will accelerate the K-Means computation for vocabulary generation. However, what about the accuracy?

➢ **Hypothesis**: Dimensionality reduction using PCA will translate into lower accuracy.

Experimental setup:

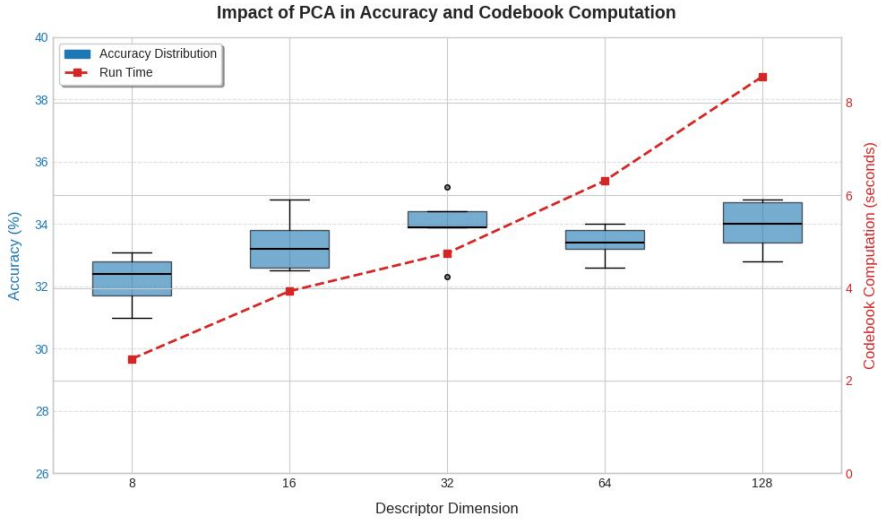
- **Classifier**: Logistic Regression with C = 1.
- **Codebook size**: 256.
- Histogram normalization.



Codebook - PCA

- **HYPOTHESIS: ✗**

Surprisingly, the mean accuracy for 32 and 128 dimensions is the same. Nonetheless, the difference in the codebook computation time is insignificant (just 4 seconds). To retain all possible discriminative power, we maintain the dimensionality of our baseline in the next experiments.



Descriptor Dimension	Mean Accuracy (%)
8	32.2
16	33.4
32	33.9
64	33.4
128	33.9

3. Classifiers

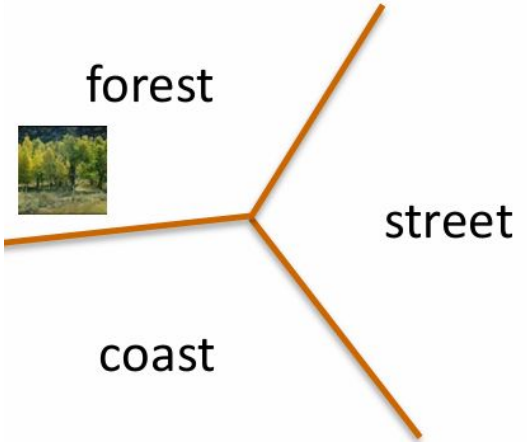
Classifiers

A classifier is an algorithm that sorts data into predefined classes based on its features. In our case, the data that we want to classify is a given set of images. Using the BoVW technique, we extract the images' features by constructing their histogram of VW, and we separate the images into classes based on these histograms.

The classifiers we are going to test are **Logistic Regression** and **Support Vector Machine** (SVM).

Experimental setup:

- **Local descriptor:** Dense SIFT with *stride* = 8 and *scale* = 2.
- **Codebook size:** 256.
- Histogram normalization.



Classifiers - Logistic Regression

One of the simplest classifiers is the **Logistic Regression**. The most important parameter that it has to be set is the C parameter, which controls the regularization penalty applied to the model's coefficients.

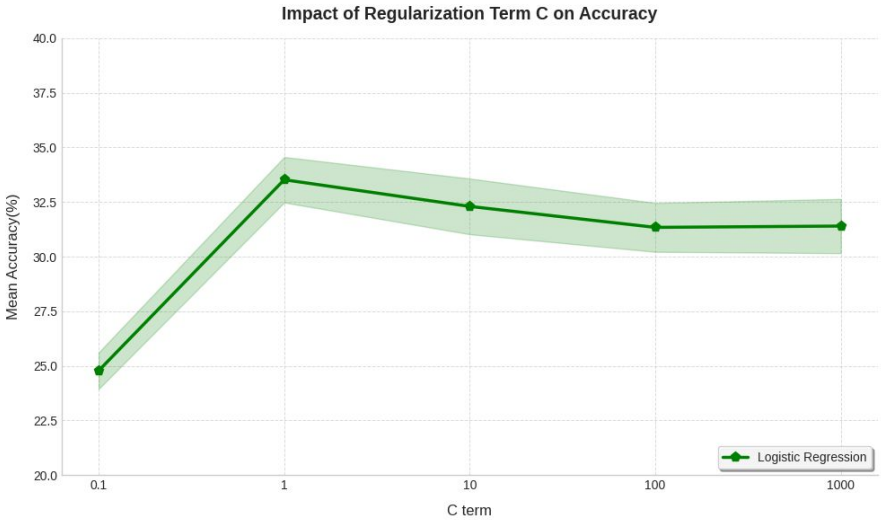
For high values (C >> 1), the optimizer focuses almost entirely on minimizing the cost function (fitting the training data perfectly). This increases the risk of **overfitting**. For low values (C << 1), the optimizer heavily penalizes large coefficients, forcing them towards zero. Hence, the model is more likely to **underfit**.

- **Hypothesis**: Neither large nor low values of C are beneficial for improving accuracy. Therefore, the optimal value is likely to be around 1, the default value.

Classifiers - Logistic Regression

- **HYPOTHESIS:** ✔

The accuracy peak is obtained when C = 1. There is a clear underfitting for C = 0.1 and the model starts overfitting for C > 1. Hence, for the next experiments we fix the value of C at 1.



The accuracy when C equals 1 is 33.5%.

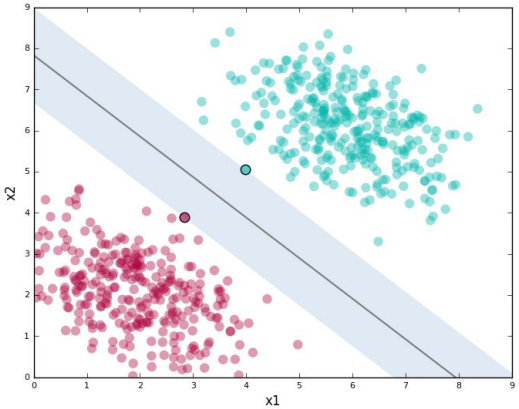
Classifiers - SVM

SVM is a more powerful classifier than Logistic Regression: it aims to find the maximum margin hyperplane and it can also handle non-linearly separable data using kernels. With them, SVMs can find a linear boundary in a high-dimensional space that corresponds to a complex, non-linear boundary in the original data space.

The kernels we test are **RBF**, **Linear** and **Histogram Intersection**.

- **Hypothesis 1**: SVM performs better than Logistic Regression.
- **Hypothesis 2:** RBF is the best kernel (it can separate non-linear data).

Same configuration as in the previous experiment, but here we fix C to 1. As it is mentioned in further slides, we have experienced that the behaviour of SVMs with respect to the C parameter is similar to the one of Logistic Regression.



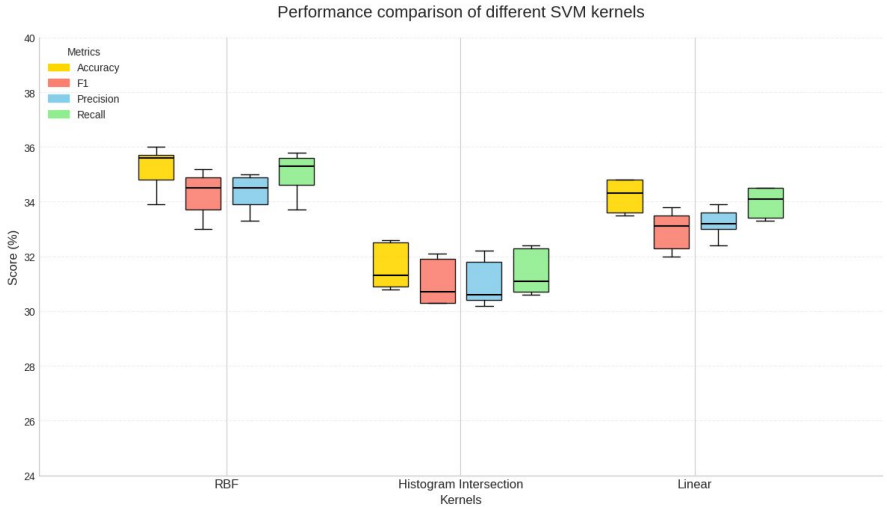
Classifiers - SVM

- **HYPOTHESIS 1:** ✔

RBF kernel outperforms the others in all metrics.

- **HYPOTHESIS 2:** ✔

The mean accuracy for RBF kernel is 35.7%, which is higher than the highest accuracy obtained using logistic regression (33.5%).



4. Spatial pyramids

Spatial pyramids

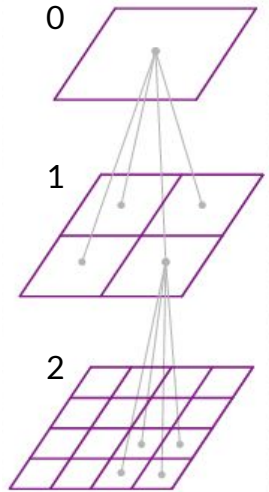
The aim of spatial pyramids is to address a major weakness of BoVW, which is the loss of spatial information (BoVW treats an image as an unordered collection of visual words). It divides the image into increasingly finer grid divisions (levels) and computes the feature histogram within each resulting region. At the end, all these histograms are concatenated.

We are going to test the spatial pyramid with 0, 1 and 2 levels.

➢ **Hypothesis**: the higher the level of the spatial pyramid, the greater the accuracy. However, we believe that there is a trade-off between accuracy and computational time, as complexity increases with each level of the spatial pyramid.

For the experimental setup, we are going to fix the following parameters:

- **Local descriptor**: Dense SIFT with *stride* = 8 and *scale* = 2
- **Codebook size**: 256
- **Classifier**: SVM with C = 1 and RBF kernel
- Histogram normalization

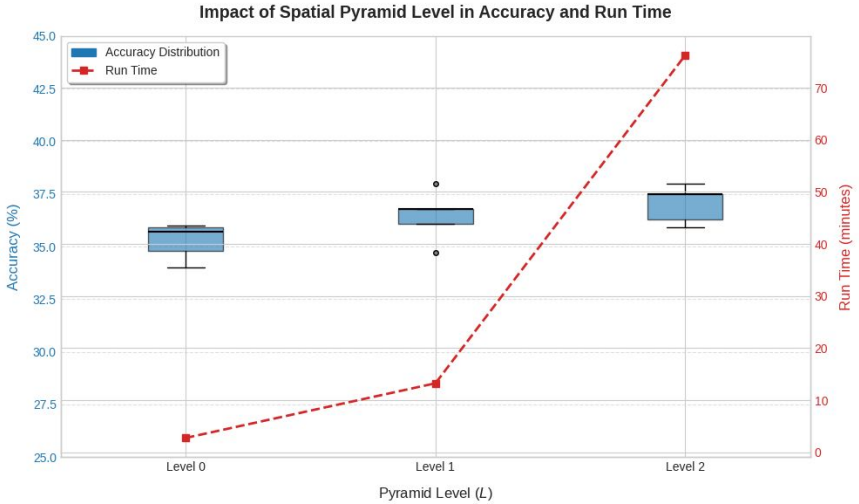


Spatial pyramids

* We conclude the **best configuration** is using the spatial pyramid up to **Level 2**. However, due to its computational cost, we run the next experiments without it. In the final results, though, we do use the optimal configuration of this experiment.

❖ **HYPOTHESIS: ✔**

Pyramid Level 2 improves the accuracy of Pyramid Level 0 by around 5%, but the runtime is also increased by 2670%. The dimensionality of the problem grows exponentially with each additional pyramid level. Nonetheless, the runtime grows exponentially, making it impossible for us to try higher levels.*



		Mean Accuracy (%)
Pyramid Level	0	35.3
	1	36.5
	2	37.0

5. Dense SIFT

Dense SIFT - Stride & Scale

In the first two experiments we compared Dense SIFT with keypoint detection methods and studied the impact of the *stride* parameter. However, the *scale* parameter (the size of the neighbourhood used for keypoint computation) was fixed to 2. Small details are captured with low values, while bigger features are captured with higher ones. Now, we want to study the impact of the *scale* on the model's performance.

➢ **Hypothesis**: Mid-values lead to higher performance: they enable to capture both small and big details.

We use the following experimental setup:

- **Classifier**: Logistic Regression with C = 1. *
- **Codebook size**: 256.
- Histogram normalization.

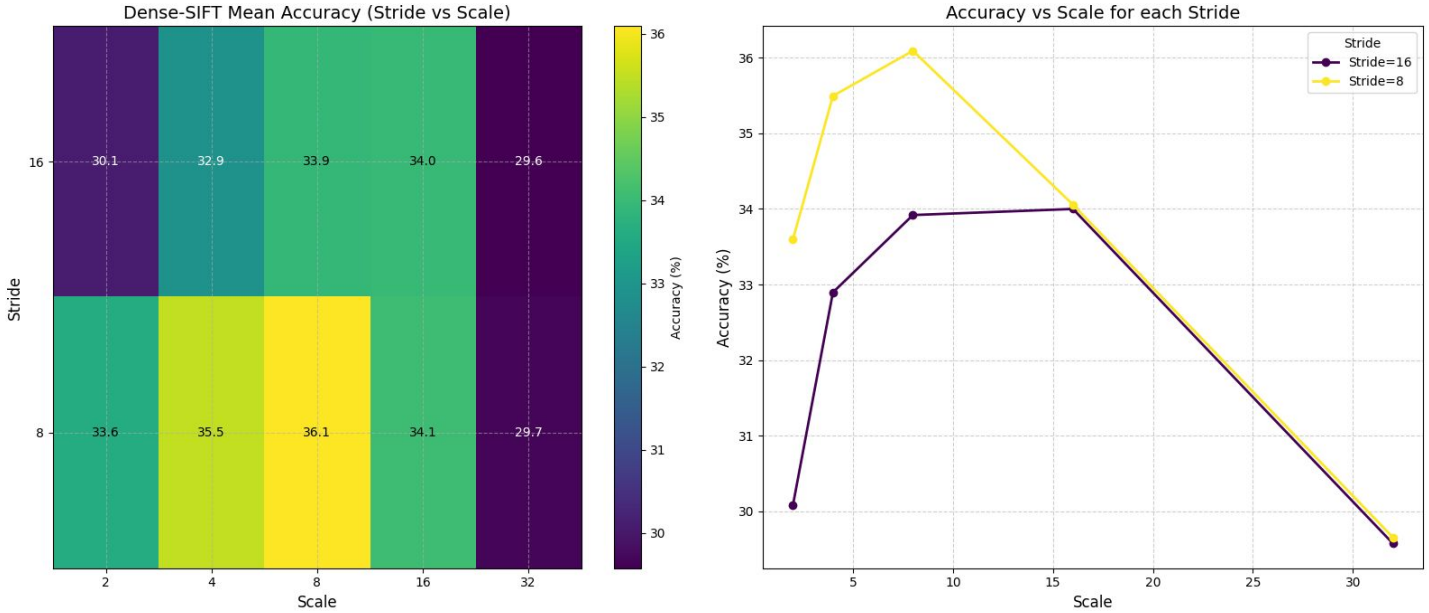
* We use Logistic Regression instead of SVM with RBF kernel (our best current classifier) to save computation. We have found that parameter settings that benefit Logistic Regression also increase the performance of the SVM classifier.

Dense SIFT - Stride & Scale

* Further reducing the *stride* parameter drastically increases the computational complexity of the classification pipeline.

❖ **HYPOTHESIS: ✔**

Increasing the *scale* yields better performance, but after a certain value, it starts to drop. Interestingly, higher accuracy is obtained for *stride* = 8*. Therefore, we keep *stride* = 8 = *scale* for the next experiments.



6. Extra

Extra - Histogram normalization

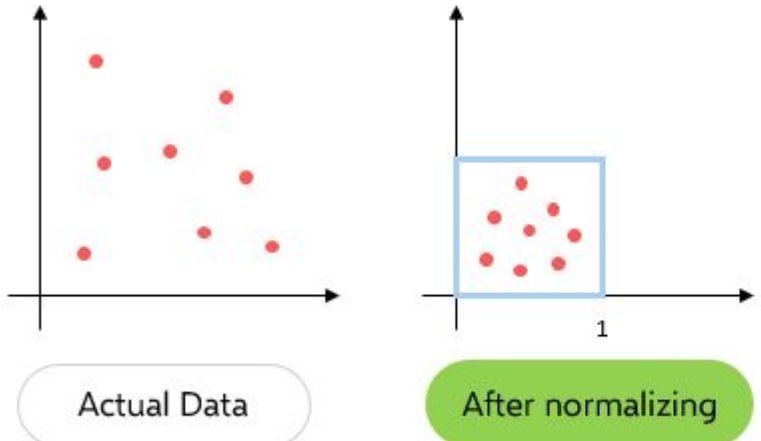
Until now, we have always normalized the histograms by their L2 norm. But, which is the real impact of it?

➢ **Hypothesis**: Histogram normalization helps the model perform better because it transforms the raw counts into relative amounts. This reduces the scale of the features used by the classifier.

We perform the experiment with the following parameters:

- **Local descriptor**: Dense SIFT with *stride* = 8 and *scale* = 8.
- **Codebook size**: 256.
- **Classifier**: Logistic Regression with C = 1 .*

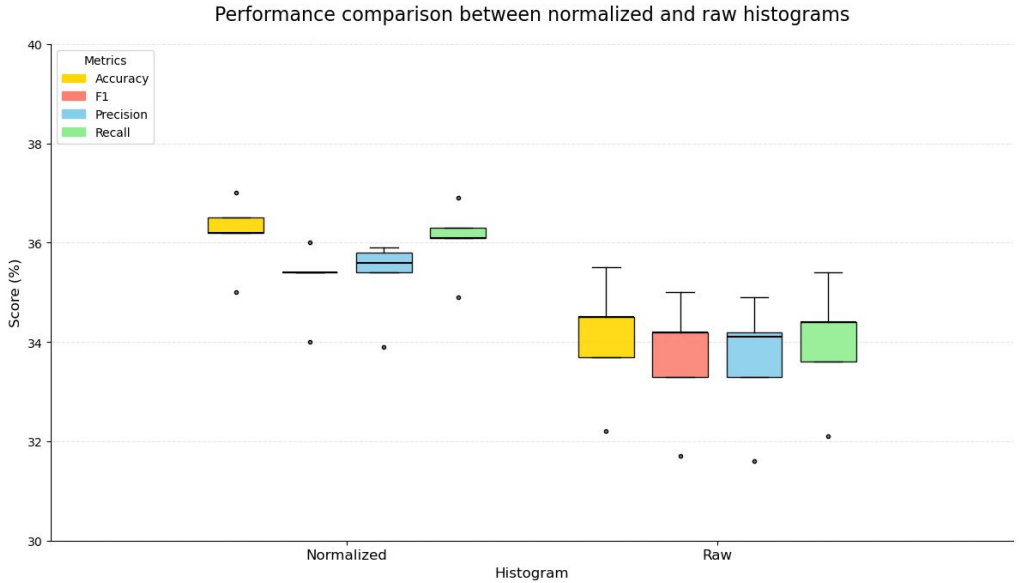
* We use Logistic Regression instead of SVM with RBF kernel (our best current classifier) to save computation. We have found that parameter settings that benefit Logistic Regression also increase the performance of the SVM classifier.



Extra - Histogram normalization

- **HYPOTHESIS: ✔**

Indeed, the classification model benefits from normalized data. Furthermore, we found the classifier struggles to converge when using the raw data. This gives us another reason to perform normalization.



Extra - Scaling

Another commonly used technique is the application of standardization (transform features to have *mean* of 0 and *std* of 1) on the descriptors before computing the vocabulary with the K-Means algorithm.

➢ **Hypothesis**: Using Standard Scaling (from *scikit-learn*) to the descriptors prior to running K-Means clustering results in higher classification accuracy: we expect it to help in constructing a more representative visual vocabulary.

For this experiment, we fix the following parameters:

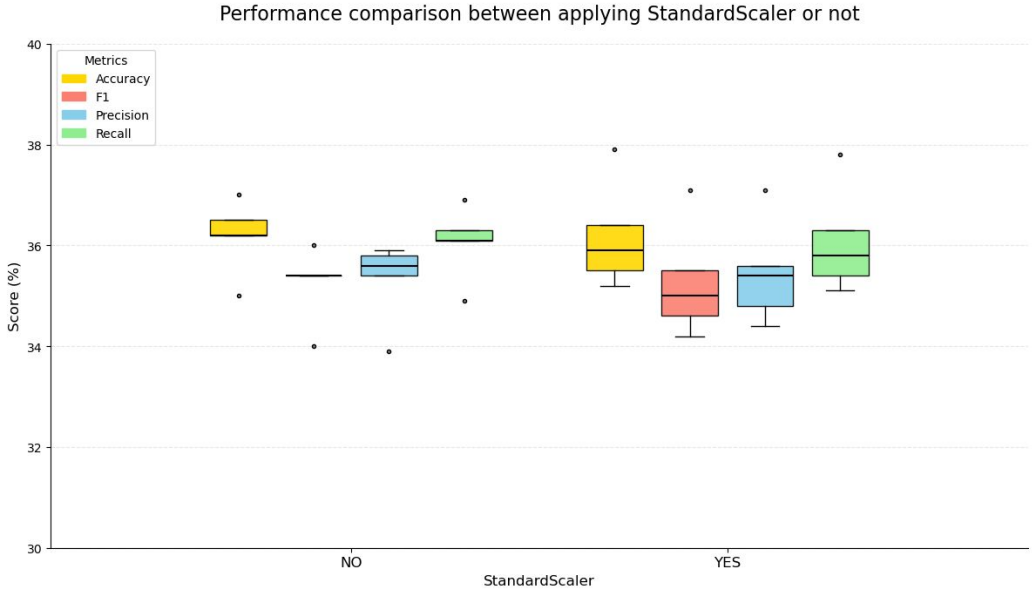
- **Local descriptor**: Dense SIFT with *stride* = 8 and *scale* = 8.
- **Codebook size**: 256.
- **Classifier**: Logistic Regression with C = 1.*
- Histogram normalization.

* We use Logistic Regression instead of SVM with RBF kernel (our best current classifier) to save computation. We have found that parameter settings that benefit Logistic Regression also increase the performance of the SVM classifier.

Extra - Scaling

- **HYPOTHESIS: ✗**

The metrics are similar when standardization is applied. Moreover, not using it provides a more confident classification (lower variance in cross-validation). Therefore, we don't use it in the next experiments.



Extra - Fisher Vectors

We explored Fisher Vectors as alternative to the K-Means codebook. Unlike BoVW, which only stores frequencies, Fisher Vectors encode second-order statistics (deviations from the mean and variance) of a Gaussian Mixture Model (GMM).

➢ **Hypothesis**: Fisher Vectors yield a richer and more discriminative image representation than BoVW, leading to higher accuracy. However, they generate very high-dimensional vectors, requiring significantly more computational power.

Dimensionality Reduction (PCA): Given our limited computational resources, we applied PCA to the local descriptors. This reduces the Dense SIFT features from 128 dimensions to 32 prior to encoding, making the training process feasible.

GMM Components: We test the number of Gaussian components (N) with values [5, 10, 25, 40].

We are going to fix the following parameters:

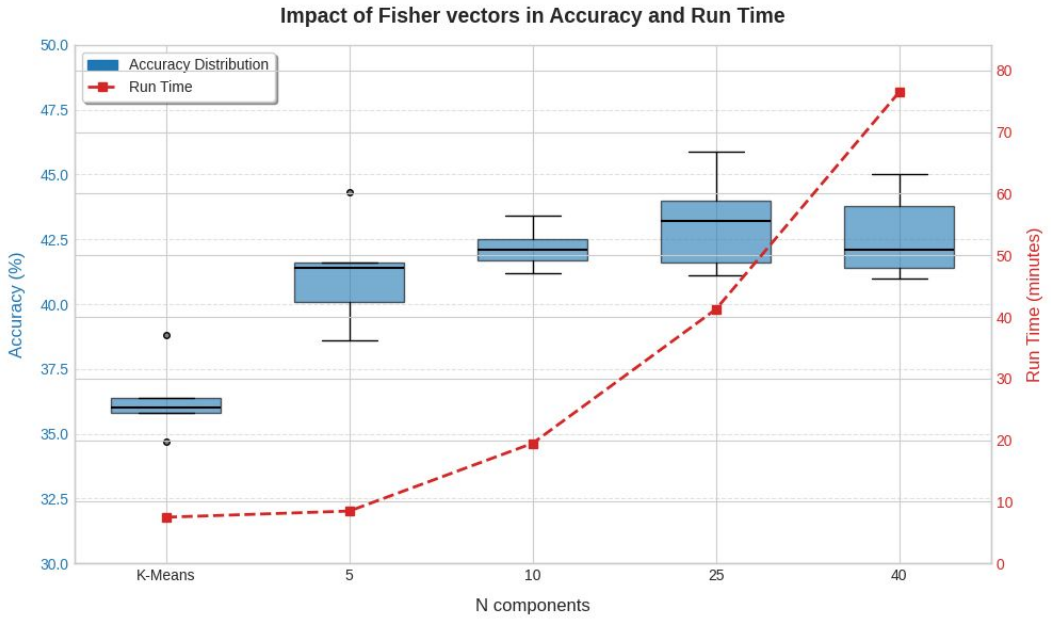
- **Local descriptor:** Dense SIFT with *stride* = 8 and *scale* = 8.
- **Classifier:** SVM with C=1 and RBF kernel.
- **Dimensionality Reduction:** PCA (32 components).

Extra - Fisher Vectors

- **HYPOTHESIS:** ✔

Fisher Vectors significantly outperform the standard K-Means approach (baseline on the left of the plot), improving accuracy by around 36% to over 44%. However, as expected, the improvement comes with a drastic increase in computation (red line), which grows exponentially with N.

We select **25 components** as the optimal configuration. This setting offers the maximum discriminative power (44.5% accuracy) before the run time becomes prohibitive. Increasing the components to 40 nearly doubles the execution time without yielding further accuracy gains.



7. Conclusions

Final model

After our experiments, the best configuration found (without accounting for Fisher vectors) is:

- **Local descriptor**: Dense SIFT with *stride* = 8 and *scale* = 8.
- **Codebook size**: 256.
- Histogram normalization.
- **Spatial pyramid** with 2 levels.
- **Classifier**: SVM with C = 1 and RBF kernel.

Metrics on the test set

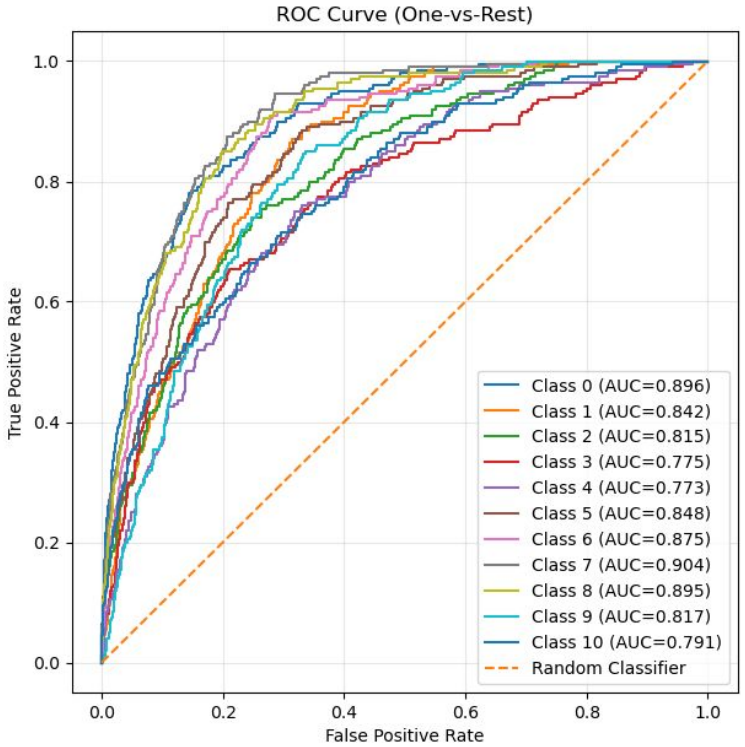
	Accuracy	F1	Precision	Recall
Score (%)	38.7	37.9	38.2	38.7

Final model

The model demonstrates good classification capability since all class curves are significantly above the diagonal.

Class 7 is the best performing class (AUC = 0.904). The model clearly identifies its visual features.

Class 4 is the worst performing class (AUC = 0.773). This suggests that the category shares visual features with others, making it difficult for the model to properly classify the images belonging to it.



Final model - Fisher vectors

After the *extra* experiments, we improved our best performing model by using Fisher vectors. The new optimal parameter configuration is the following:

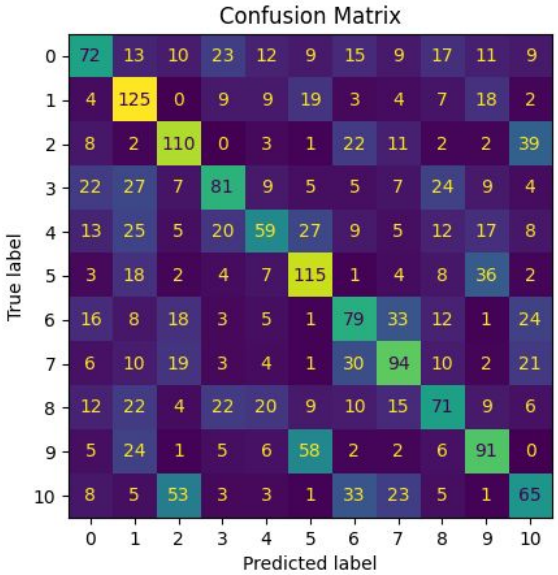
- **Local descriptor**: Dense SIFT with *stride* = 8 and *scale* = 8.
- **Codebook size**: 256.
- **Fisher vectors** with 25 components (N).
- Histogram normalization.
- **Spatial pyramid** with 0 levels.
- **Classifier**: SVM with C = 1 and RBF kernel.

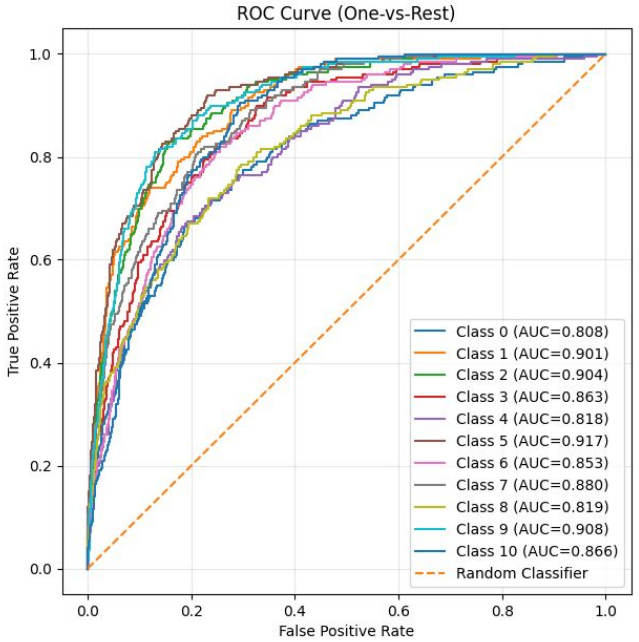
Metrics on the test set

	Accuracy	F1	Precision	Recall
Score (%)	44.5	43.9	44.0	44.5

Final model - Fisher vectors

AUC is higher for the classes with poor performance in the previous model. However, class 4 is still the one with the worst performance (AUC = 0.818). Now, though, the performance for class 5 is the best (AUC = 0.917).





Qualitative analysis

True: commercial buildings
Pred: shopping and dining (0.24)
restaurant_patio_Places365_val_00005009.jpg

Figure a

True: sports and leisure
Pred: home or hotel (0.57)
locker_room_Places365_val_00014380.jpg

Figure b

True: mountains_hills_desert_sky
Pred: sports_fields (0.22)
rock_arch_Places365_val_00019914.jpg

Figure c

True: industrial and construction
Pred: mountains_hills_desert_sky (0.39)
water_tower_Places365_val_00031351.jpg

Figure d

Some failures of the model are logical:

- For example, in case a), we predicted shopping and dining because the image appeared in an alley with restaurants, which can be found in the "shopping and dining" category.
- Case b) is from a room containing clothes that looks like a changing room, which is also found in the "Home or Hotel" category.
- Case C is a mountain with a sand texture and is predicted as "sports fields." Upon checking the category, we found bullrings, which also have a sand texture.
- Example d) is an image from the "industrial and construction" category that was predicted as "mountains, hills, desert, and sky." This is another case where we can find images like this in the predicted class.

Based on this analysis and our review of the class images, we conclude that the dataset's classes need to be more specific to address this issue. This is because there are images from different categories with similar textures and compositions.