Universitat Politècnica de Catalunya

Machine Learning II

Image Classification Using Kernels

Authors:

Benjamí Parellada
Sígrid Vila

Fall Semester 2021/2022

Contents

1 Introduction

Image recognition is the part of computer vision which uses machine learning to identify and detect objects or features in digital visual formats, such as image or video. The main application of image recognition is the classification and detection of objects, patterns, or properties of objects, in digital images. With a wide array of applications such as image retrieval, facial recognition, computer aided design, robotics, and surveillance.

A fundamental element of computer vision and image recognition is the extraction of features from images. There are many types of features such as edges, corners, blobs, moments, etc. Most features, like edges, corners, color histograms, are local features. Local features are useful for object detection and classification because they are highly invariant to affine transformations, such as rotation and scaling.

In this project, we build and test models for the multinomial task of classifying images using kernel methods. Concretely, we extract various features from the images and using Kernalized Support Vector Machines (KSVM) as a classifier we determine how accurate the classification is with each kernel. The focus of this project is to study and delve deeper into kernels, as such we use different kernels in order to solve the proposed problem. We implement the Log kernel from Boughorbel et al. [2], the Generalized Histogram Intersection kernel from Boughorbel et al. [3] and Mixture kernels from Tian et al. [11].

To test the effectiveness of the SVM with the different kernels, we have used a subset of the COREL database for content based image retrieval [10]. The results from the various kernelized SVM are contrasted with state of the art convolutional neural networks (CNN) and a simpler models such as K-Nearest Neighbors (KNN).

The results returned from the Log and Mixture kernels are not as good as the ones presented in the literature. However, we obtain similar results using the Polynomial, Gaussian RBF, and the Generalized Histogram Intersection. The results hold even though we have used more classes, and classes that have similar color distribution profile which are harder to distinguish than the ones presented in the papers.

The remainder of the paper is organized as follows. Section 2 presents the previous state of the art work done in relation to solving this problem. Section 4 presents the COREL database and the subset of images we are using to test our algorithms. Section 3 presents the mathematical background needed to understand the corresponding design done, and the kernels used. Section 5 presents the results with the approach presented. Finally, Section 6 draws the conclusions of the paper.

2 Previous work

While image retrieval and recognition has been recently dominated by Neural Networks, with accuracies of over 90% this year with CoAtNet-3 [4] and ViT-G/14 [12]. The performances seems to be reaching a stand-still and the computational resources and number of parameters needed is staggering. Hence, simpler models with similar performances are always welcome.

Kernel based methods such as KSVM have been studied in many contexts. The main computational advantage of kernel-based methods is that they do not require explicit feature mapping or feature engineering and are able to implicitly capture the inherent structure of the input space, allowing them to achieve superior generalization performance. Despite its success, however, the performance of a kernel-based classifier depends crucially on the selection of an appropriate kernel function. A large collection of kernel functions are available, each with their own strengths and weaknesses.

In 2005, Boughorbel published various works on image recognition using KSVM, our particular interests comes from Conditionally Positive Definite Kernels [2] in which a rigorous comparison was made on the performance of different kernels on the task of image classification with KSVM's. Using a dataset of six classes of one hundred images in RGB, they extracted the color histograms of each image and used SVM with different kernels in order to predict the classes. The particular kernels used were Gaussian Radial Basis Function (RBF), Laplace RBF, Polynomial, and they introduced a new conditionally positive definite kernel they named *Log*.

The use of color histograms brings the question in how to compare histograms adequately, as there are many metrics to compare distributions of histograms [7]. Thankfully for us, Boughorbel's group, also presented another kernel named *Generalized Histogram Intersection* (GHI) at the same year they presented the Log kernel. This new kernel was based on the histogram intersection for image retrieval and showed good results [3].

Kernel	**CV Error**	**Test Error**
RBF	24.38 ± 0.54	24.33 ± 1.06
Laplace	23.50 ± 0.82	24.66 ± 0.89
Power	21.88 ± 0.15	21.44 ± 2.16
Log	21.77 ± 0.20	21.12 ± 1.70
GHI	21.11 ± 1.29	20.33 ± 1.70

Table 1: Summary of the results presented by Boughorbel et al. in [2, 3] using the COREL Database.

More recently, mixture models have proven to be quite useful in modeling complex den-

sities and are very flexible, as using a small number of parameters, one can model very complex distributions. Tian et al. [11] have applied kernel mixture models on image classification and achieved higher classification accuracy than conventional SVM with any single kernels.

Seeing these results, we embark in a mission to try and replicate them, while also presenting new mixture kernels that we have developed.

3 Methodology

In this section we shall describe the implementation of the algorithm that detects and classifies images from a subset of classes. The pipeline that our computer vision system follows is represented in Figure 1. Concretely, from an input image, there is a preprocessing applied, after which we extract features for each image. These features are fed to a machine learning model which will predict the class of the input image. Following, we explain step by step each of the points in the diagram to explain the overall classifier.

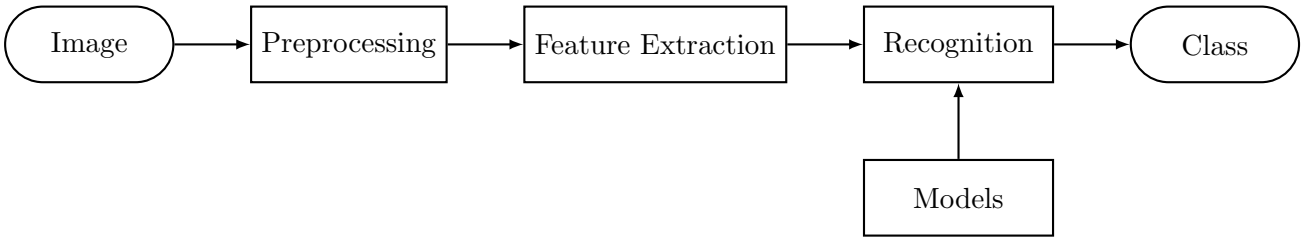


Figure 1: Flux diagram of our image classification system.

3.1 Images

Before we can start any sort of processing or modeling, we need images to build our discriminant model which will classify the input images. These images are split into 2 parts: training and test. The images used in the training set represent 90% of the images we have and will be used to construct the machine learning models. Once these models have been tuned, we process each image in the test partition the same way the images as the train set to get an unbiased generalized error of the classifier.

The specific information on the image database used is described in section 4.

3.2 Preprocessing

Whether the image is train or validation, they all will have to follow the same processing. In the case of this project, the processing is minimal, we only test different color spaces and color normalization. Concretely, the processing we do is:

- **RGB**: the default 3D space most images are in. We can normalize each image in order to reduce to a 2D space:

  $$r = \frac{R}{R+G+B} \qquad g = \frac{G}{R+G+B} \qquad b = \frac{B}{R+G+B}$$

  Since $r + g + b = 1$, we can eliminate one dimension by using this transformation. Other than reducing the dimension, we achieve invariance towards illumination with this transformation.

There are other representations of color, such as HSV and CIELAB which have their benefits. In HSV, the Hue component represents the chrome directly, which is ideal for working on image classes that are very reliant on color. However, there are some downsides to this representation, the Hue is cyclic therefore 0° and 360° represent the same color, and you either map it to polar coordinates or use some programming hack to make them the same. Moreover, the Hue value of black can be random and we do not have invariance towards illumination.

The CIELAB, is also a 3D space where the main benefit is that changes of color represent a uniform change, however, while the values of a* and b* tend to be around [-128, 127] there is no guarantee and the computation from RGB to L*a*b* is quite costly.

Evaluating the costs of each of these color spaces, the easiest representation seems to be the RGB color space with the chromatic normalization. We have the benefit of reducing by one dimension the image while maintaining all the relevant information.

We also tested with histogram equalization which is a technique for adjusting image intensities to enhance contrast, however the results were worse than not using it.

3.3 Feature Extraction

After reading and processing the images, we need to extract features from them. At first, we tried to flatten each image an compare pixel by pixel, however this approach lent a myriad of problems. First, the computational complexity of this approach was unfeasible. Just by flattening one RGB image of 320×320 pixels would produce 307200 features. Second, not all images are 320×320, as such we had to pad these images and center

them correctly. Third, it is not translation nor rotational invariant, just one shift of one pixel to each row of the image would make our classifier fail. Hence, the need for better descriptors was needed.

The main descriptor used is the color histograms, which is basically the distribution of colors in each of the dimensions in the color space. So, since we normalized the RGB color space, the corresponding color histogram for an image will be two vectors, each representing a component, i.e. r and g. Here we see the benefit of this color representation, with HSV or CIELAB, we would have needed to use three vectors to represent each of the components of the color space, however using normalization in RGB there is one color that can be lineally computed from the other two.

We decided to use color histograms for two reasons: the first was that in [2, 3, 11], they used it as a local feature. Secondly, there is usually a relationship between color and the object represented. Especially in our database, for example elephants are mostly gray, beaches mostly blue, etc. Hence, using color histograms as local features describing the image will produce descriptors that are invariant to geometric transformations and appropriate for our use case.

While this approach has quite a few benefits, we have to be careful when we define the number of bins of the histograms. We want them to be the same for each image, and one would be quick to set 256 or 64 as they did in [2, 3]. However, fixing the number of bins does not guarantee that the bins match between histograms, as such two very different color distributions could have the same number of counts in their histogram due to the bins not representing the same values. The correct way to proceed is to homogeneously map all the input space to a fixed number of bins, which we equally spaced.

Summarizing, for each image we concatenate the counts of the two normalized color histograms by homogeneously mapping [0, 1] into 128 bins. We also add the appropriate label into a matrix.

3.4 Models

With the feature descriptors obtained from the images described in the previous section, we train a model that will use be used to recognize the different classes that can appear in the image. Since we have more than one class, we have to use multi-class classifiers.

3.4.1 K-Nearest Neighbours

K-Nearest Neighbors (KNN) [6] is an instance-based learning method which does not attempt to construct a general internal model, but simply stores instances of the training

data. Classification is computed from a simple majority vote of the K nearest neighbors of each point.

As we are using color histograms as descriptors, we are comparing normalized values against each other. As such, we are just comparing distances between the histograms which should be in Euclidean Space. Hence, we can use KNN without worry.

We have one hyperparameter to tune: K, which is the number of neighbors that will vote on each instance. We test with a wide array of odd values using k-fold cross-validation. We particularly use odd-values since it is impossible to have a tie when voting.

3.4.2 Support Vector Machine

The objective of Support Vector Machine (SVM) [1] is to find a hyperplane in an N-dimensional space that distinctly classifies the data points.

The SVM can be reduced to a quadratic minimization problem:

$$W(\boldsymbol{\alpha}) = -\sum_{i=1}^{l} \alpha_i + \frac{1}{2} \sum_{i,j=1}^{l} \alpha_i \alpha_j y_i y_j K(\boldsymbol{x}_i, \boldsymbol{x}_j)$$

with respect to Khun-Tucker coefficients $\boldsymbol{\alpha}$, under que equilibirum constraint:

$$\sum_{i=1}^{l} \alpha_i y_i = 0$$

As long as $K(\boldsymbol{x}_i, \boldsymbol{x}_j)$ is positive definite [2], the $W(\boldsymbol{\alpha})$ is convex with respect to $\boldsymbol{\alpha}$, therefore the minimization is achieved at a unique minimum. In Boughorbel [2], they prove how conditionally positive definite can be used for the SVM.

This addition allows to modify the classic SVM which performs linear classification, to efficiently perform a non-linear classification using what is called the kernel trick, implicitly mapping their inputs into high-dimensional feature spaces.

The kernels we tested are:

- **Linear** is the simplest of all kernel functions. It would equate to the classic SVM without kernel, the main inconvenient is that we need numerical data as input.

$$K_L(\boldsymbol{x}, \boldsymbol{x}') = \langle \boldsymbol{x}, \boldsymbol{x}' \rangle$$

- **Gaussian Radial Basis Function** (RBF) or Gaussian kernel. A general purpose

kernel typically used when no prior knowledge is available about the data.

$$K_{RBF}(\boldsymbol{x}, \boldsymbol{x}') = \exp\left(-\sigma||\boldsymbol{x} - \boldsymbol{x}'||^2\right)$$

Intuitively, the σ parameter defines how far the influence of a single training example reaches, with low values meaning 'far' and high values meaning 'close'. The gamma parameters can be seen as the inverse of the radius of influence of samples selected by the model as support vectors.

Increasing σ leads to a better fit to training data (danger of ending up overfitting), and decreasing σ (towards zero) leads to a worse fit to training data (danger of ending up underfitting)

- **Laplace radial basis kernel** very similar to the classical Gaussian RBF, a general purpose kernel used when no prior knowledge is available.

$$K_{LRBF}(\boldsymbol{x}, \boldsymbol{x}') = \exp\left(-\sigma||\boldsymbol{x} - \boldsymbol{x}'||\right)$$

- **Polynomic** represents the similarity between two vectors. Not only does it take into consideration the similarity in one dimension, but in multiple. It basically counts the interaction between characteristics. It is quite useful in image classification [2, 3]. The p should be a positive integer, θ represents the scale of the interaction and c an offset.

$$K_P(\boldsymbol{x}, \boldsymbol{x}') = (\theta\langle \boldsymbol{x}, \boldsymbol{x}' + c\rangle)^p$$

- **Logarithmic** kernel as presented by Boughorbel et al. [2]. A conditionally positive kernel, if the parameter is $0 < \beta \leq 2$.

$$K_{Log}(\boldsymbol{x}, \boldsymbol{x}') = -\log\left(1 + ||\boldsymbol{x} - \boldsymbol{x}'||^\beta\right)$$

- **Generalized Intersection Histogram** kernel as presented by Boughorbel et al. [3]. A positive definite kernel, with one parameter $\beta \geq 0$.

$$K_{GHI}(\boldsymbol{x}, \boldsymbol{x}') = \sum_{i=1}^{m} \min\{|x_i|^\beta, |x_i'|^\beta\}$$

- **Mixture** kernels, take use of the property that the sum of kernels is kernel. As such, we can define kernels by adjusting the weight of each kernel. Being K_1 and K_2 kernels we can define a mixture kernel as:

$$K_{Mix}(\boldsymbol{x}, \boldsymbol{x}') = \lambda K_1(\boldsymbol{x}, \boldsymbol{x}') + (1 - \lambda)K_2(\boldsymbol{x}, \boldsymbol{x}')$$

As such, we define the following kernels, the first one and the idea to produce these kernels comes from [11].

$$
\begin{aligned}
K_{RBF-P}(\boldsymbol{x}, \boldsymbol{x}') &= \lambda K_{RBF}(\boldsymbol{x}, \boldsymbol{x}') + (1-\lambda) K_{P}(\boldsymbol{x}, \boldsymbol{x}') \\
K_{RBF-Log}(\boldsymbol{x}, \boldsymbol{x}') &= \lambda K_{RBF}(\boldsymbol{x}, \boldsymbol{x}') + (1-\lambda) K_{Log}(\boldsymbol{x}, \boldsymbol{x}') \\
K_{Log-P}(\boldsymbol{x}, \boldsymbol{x}') &= \lambda K_{Log}(\boldsymbol{x}, \boldsymbol{x}') + (1-\lambda) K_{P}(\boldsymbol{x}, \boldsymbol{x}') \\
K_{RBF-GIH}(\boldsymbol{x}, \boldsymbol{x}') &= \lambda K_{RBF}(\boldsymbol{x}, \boldsymbol{x}') + (1-\lambda) K_{GIH}(\boldsymbol{x}, \boldsymbol{x}') \\
K_{Log-GIH}(\boldsymbol{x}, \boldsymbol{x}') &= \lambda K_{Log}(\boldsymbol{x}, \boldsymbol{x}') + (1-\lambda) K_{GIH}(\boldsymbol{x}, \boldsymbol{x}') \\
K_{P-GIH}(\boldsymbol{x}, \boldsymbol{x}') &= \lambda K_{P}(\boldsymbol{x}, \boldsymbol{x}') + (1-\lambda) K_{GIH}(\boldsymbol{x}, \boldsymbol{x}')
\end{aligned}
$$

The SVM is formulated for binary classification, in our case we have a classification problem which includes more than two classes, hence we opt for a one-against-one method. This method constructs $\binom{k}{2}$ classifiers where each one is trained on the data from the two respective classes. Prediction is done by voting, where each classifier gives a prediction and the class which is predicted more often wins, similar to KNN.

Finally, other than the parameters from each kernel, the SVM used in classification uses one hyperparameter C. C behaves as a regularization parameter in the SVM and trades off correct classification of training examples against maximization of the decision function's margin. For larger values of C, a smaller margin will be accepted if the decision function is better at classifying all training points correctly. A lower C will encourage a larger margin, therefore a simpler decision function, at the cost of training accuracy.

3.4.3 Convulational Neural Networks

For the the purpose of this project, we create a Convolutional Neural Networks (CNN) similar to the VGG-16 [9], the following is a description of the blocks used:

1. **Convolutional Blocks** which consists of a padding layer to ensure we can stride over the edges, a convolutional layer of stride 3, a rectification (ReLU) non-linearity activation function, and finally a max pooling layer.

2. **Flatten** which flattens a matrix to a 1-Dimensional vector.

3. **Dropout** which randomly zeroes some of the elements of the input tensor to reduce overfitting.

4. **Fully connected layers** which consist of densely connected layers with a ReLU or a log soft-max for the final layer.

We combine four blocks of 1), after which we flatten the image 2), and apply dropout 3), after the network has been flattened, we densly connect layers twice 4) with the second one being a log soft-max. All this is trained for 30 epochs in which we use the negative log likelihood loss which is useful for multi-class classification problems.

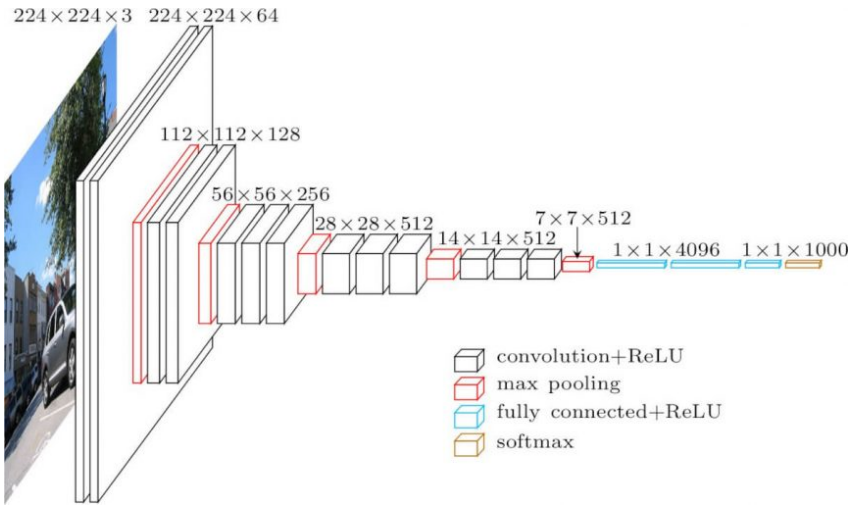


Figure 2: Summary of the architecture of the VGG-16 architecture, which is similar to ours. Image from neurohive, 2018.

3.5 Experimentation

The objective of this work is to recognize what appears in an image from a subset of possible classes. To this end, we shall try the different models used on the preprocessed features extracted from the images. The images are partitioned into train and test. For the KNN and SVM models we use 5-fold cross validation in order to find the best hyperparameters. Once the parameters are tuned, we use cross validation with the training partition in order to return an accurate estimate of the training error. Afterwards, we use the test partition to return an unbiased generalization error of a sample the model has never seen. For the CNN, the procedure is similar, however we are training and testing it on every epoch, and we use the raw images instead of the features extracted.

KNN	$k \in \{3, 5, 7, 9, 13, 15, 17, 21, 35, 51\}$
SVM	$C \in \{0.001, 0.01, 0.1, 1, 10, 100, 1000\}$
RBF	$\sigma \in \{0.00001, 0.0001, 0.001, 0.01, 0.1, 1, 10, 100, 1000\}$
P	$p \in \{2, 3, 4, 5, 6, 7\}$
Log/GIH	$\beta \in \{0.1, 0.2, 0.25, 0.3, 0.4, 0.7, 0.9\}$
Mix	$\lambda \in \{0.2, 0.4, 0.5, 0.6, 0.8\}$

Table 2: Different hyperparameters tuned and the ranges checked.

Take into consideration, that for the mixed kernels, we only tune the mix λ parameter, taking the best hyperparameters for the different individual kernels.

4 Case study

In order to apply the work described above we use the COREL database [10]. However, for the scope of this project, the entire database would be impossible to train and compute, since it contains 10800 images which can be divided into 80 different classes, e.g., autumn, aviation, bus, castle, cloud, dog, elephant, primates, ship, tiger, train, waterfall, etc.

The original papers from Boughorbel [2] used 6 distinct classes: castle, sun rises, grasses, mountains, birds and waterfalls. To validate the results in the paper, we decide to use 10 different classes with more variation than the ones presented in the paper. The classes we use are: beaches, buses, dinosaurs, elephants, flowers, foods, horses, monuments, mountains and snow, and people in villages in Africa. A sample of the images used can be found in Figure 3. Each class contains 100 RGB images, 90 will form the training set and the 10 remaining the test set.

Figure 3: Each row in each block represents on of the 10 classes used for the experiments. Left block: beaches, buses, dinosaurs, elephants, flowers; right block: food, horses, monuments, mountains, people in Africa.

5 Results

We compute first the average validation error using 5-fold cross validation for the different hyperparameters. Once these best parameters have been found, we repeat a 5-fold CV to give an accurate measurement of the metrics in the train partition. Afterwards, we

evaluate the data on the test partition for an accurate generalized error. As explained in section 3 we try each kernel for the SVM with different values of regularization. For the KNN, we do the same process, however only with the number of neighbors that vote, K.

Since this project is more focused on studying kernels, we present the full results that the kernalized SVM has achieved in order to draw a comparison between the kernels on our specific problems. Once we have decided which of these combinations of hyperparameters and kernels is the best for the SVM, we choose the one that returns the least validation error and compare it with the CNN and KNN.

5.1 Kernalized SVM

After cross validating to find the best hyperparameter for each value of C, we compute the validation error using these parameters. The following table represents the mean of the 5-fold CV done once we found the best hyperparameter for each kernel and regularization.

Kernel	0.0001	0.001	0.01	0.1	1	10	100	1000
K_L	0.3156	**0.2978**	0.3067	0.3189	0.3300	0.3178	0.3244	0.3411
K_{GRBF}	0.9289	0.9200	0.9244	0.5667	0.3033	**0.2678**	0.2711	0.2800
K_P	0.2778	0.2922	0.2856	0.3044	**0.2689**	0.2900	0.2989	0.2833
K_{GIH}	0.7956	0.2622	**0.2056**	0.2844	0.2811	0.3233	0.2778	0.2789
K_{Log}	0.8878	0.9322	0.9100	0.5967	0.3367	**0.3011**	0.3022	0.3067
K_{LRBF}	0.9189	0.9278	0.8889	0.9344	0.8344	0.4933	0.2789	**0.2556**
K_{RBF-P}	0.4156	0.3900	**0.3400**	0.4178	0.4078	0.3744	0.3933	0.4444
$K_{RBF-Log}$	0.9389	0.9278	0.9167	0.5989	**0.2978**	0.4078	0.3222	0.3178
K_{Log-P}	0.3944	0.3933	0.3767	0.3656	0.4344	0.4067	**0.3711**	0.4289
$K_{RBF-GIH}$	0.8900	0.2756	**0.2144**	0.2189	0.2889	0.3400	0.2989	0.2911
$K_{Log-GIH}$	0.9211	0.2689	**0.2189**	0.2356	0.2889	0.3544	0.2944	0.2767
K_{P-GIH}	0.3778	0.3933	0.3622	0.3544	0.3911	0.3489	**0.3467**	0.3822

Table 3: Validation error for different kernels and values of C. Marked with in bold the lowest error for each kernel.

The following table presents the winning kernel hyperparameters, the full table with each of the best parameters can be found in Table 8 at the Annex.

K_{GRBF}	K_P	K_{GIH}	K_{Log}	K_{LRBF}	K_{RBF-P}	$K_{RBF-Log}$	K_{Log-P}	$K_{RBF-GIH}$	$K_{Log-GIH}$	K_{P-GIH}
0.0001	2	0.25	0.2	0.0001	0.8	0.8	0.2	0.2	0.2	0.2

Table 4: The hyperparameters that returned the best test validation error (the ones marked in bold in the previous table).

We see the GIH kernel is the one that returns the best results, similar to the ones in [3]. Moreover, the kernels that are right behind it are combinations of it: RBF-GIH and Log-GIH. Checking the mixture parameter, λ, we can see how the mixture is not really working that well as it is clearly moving towards the model which returns better results individually. In other words, it gives more weight to the better individual kernel (i.e. the GIH). All the other kernels get results that are far worse in comparison to the GIH kernel and its combinations. The Log kernel, seems to be quite far to the results presented in [2], returning on of the worst performances in the set.

Finally, with each of the best hyperparameters found, we evaluate the results on the test partition:

Kernel	0.0001	0.001	0.01	0.1	1	10	100	1000
K_L	0.3100	**0.2400**	**0.2400**	0.2600	0.2600	0.2600	0.2600	0.2600
K_{GRBF}	0.4400	0.4400	0.4700	0.3900	0.2700	**0.2400**	0.2600	0.2600
K_P	0.2700	**0.2500**	**0.2500**	**0.2500**	0.2700	**0.2500**	**0.2500**	0.2700
K_{GIH}	**0.2000**	**0.2000**	**0.2000**	0.2400	**0.2000**	0.2400	0.2300	0.2400
K_{Log}	0.3500	0.3600	0.3600	0.3400	0.3200	**0.3100**	**0.3100**	**0.3100**
K_{LRBF}	0.4200	0.4200	0.4200	0.4200	0.4200	0.3500	0.2400	**0.2100**
K_{RBF-P}	0.4100	0.3600	**0.3200**	0.3300	0.4200	0.3700	0.3800	0.4000
$K_{RBF-Log}$	0.4400	0.4400	0.3400	0.3700	**0.2700**	0.3400	0.3200	0.3000
K_{Log-P}	0.4100	0.3400	0.3500	0.3200	0.3300	**0.3100**	0.4000	0.4400
$K_{RBF-GIH}$	0.2000	0.2000	**0.1900**	0.2000	0.2700	0.3000	0.2200	0.2300
$K_{Log-GIH}$	0.2000	0.2000	**0.1900**	0.2200	0.2700	0.3600	0.2200	0.3100
K_{P-GIH}	0.4000	0.3700	0.3300	0.3300	0.4000	0.3700	**0.3100**	0.3700

Table 5: Test error for the different kernels tested and various values of C. Marked in bold the lowest error for each kernel.

Just like in the validation error, the kernels with the best test error are the GIH and the combinations with it. Nevertheless, the results are very tight to conclude that one is better than the other to perform image classification. And, the mixture seemed to now work better than the individual classifier, even if it is just one more image correctly classified.

However, we would rather use the GIH as the final model for the SVM. The small performance we win with the mixture models does not seem to outweigh the increase of complexity. Furthermore, checking Table 8, we can observe how most of the time, the mixture does not stay in the middle, but strongly weighs in the direction of the better individual kernel.

The following table represents the confusion matrix for the GIH kernel on the test partition, which we considered is the best performing kernel on our data.

Ref. \ Pred.	Beach	Bus	Dino	Eleph	Flwr	Food	Hors	Monu	Moun	Peop
Beaches	6	1	-	-	-	1	-	-	1	1
Bus	-	10	-	-	-	-	-	-	-	-
Dinosaurs	-	-	10	-	-	-	-	-	-	-
Elephants	2	-	-	8	-	-	-	-	-	-
Flowers	-	-	-	-	10	-	-	-	-	-
Foods	1	1	-	-	-	7	-	-	1	-
Horses	-	-	-	1	-	-	9	-	-	-
Monuments	1	-	-	1	-	-	-	6	2	-
Mountains	5	1	-	-	-	-	-	-	4	-
People	-	-	-	-	-	-	-	-	-	10

Table 6: Confusion matrix for the multiclass SVM with the GIH kernel with C = 0.01 and $\beta = 0.25$.

We can see that the GIH kernel distinguishes perfectly the buses, dinosaurs, flowers and people living in Africa. However, it has difficulties distinguishing beaches, mountains, and monuments. The reason for this is quite clear, these classes have quite similar color profiles, especially the beaches and mountains as we can see in Figure 3. If we only use the color for image classification, these limitations are bound to happen.

Overall, it is clear that the GIH kernel and their combinations resulted with the best performances by far, even a bit better than the original paper [3]. The polynomial kernel and Gaussian RBF also seem comparable to the results presented in the paper. However, the Log kernel seems to miss the mark by achieving quite worse results than the original literature [2]. The mixture kernels, also do not seem to live up to the expectations. While returning adequate results, it seems they are mostly mixing towards the best individual kernel.

5.2 Final Model and Discussion of the Results

Once we have seen which kernel and regularization parameter combination returns the best result for the SVM, we compare this method with more state of the art approaches such as a CNN. The overall results of our application can be found in Table 7.

The CNN and SVM seem to be quite comparable in their results. While the CNN has achieved better results overall, the difference between the two is quite small and depending on our devices embedding, we might decide to go for the SVM.

Model	Validation Error	Test Error
KNN	0.3767	0.3100
SVM	0.2056	0.2000
CNN	0.1690	0.1800

Table 7: Summary of the results using the methodology presented on the COREL Database. For the SVM the hyperparameters were set was set at $C = 0.01$ and a Generalized Intersection Kernel (GIH) with $\beta = 0.25$; for the KNN the best K achieved in cross validation was $K = 9$.

Nonetheless, it is clear that the kernelized SVM's are a much better option to perform image classification than the KNN, as its error is significantly worse. However, we must take into account that the KNN are simpler and take less time to train.

6 Conclusion

In this paper we have presented a methodology to solve an image classification problem exploring and testing many different kernels and some state of the art methods. To this end, we created a subset of images from the COREL [10] database which amounted to 10 distinct classes each containing 100 images in RGB.

From these images, various preprocessing techniques have been tested, such as histogram equalization, changing the color space to HSV, CIELAB and chromatic normalization; however, the only relevant preprocessing ended up being RGB normalization to reduce the number of features of the dataset.

From this preprocessed images, we feature extract color histograms in order to distinguish the different classes. These descriptors have the property of being invariant to geometric transformations. The reduction of the dimensionality, as well as the qualities of color histograms and the nature of the problem, has brought us to believe that these descriptors are adequate to determine which class to classify each image as.

Once the color histograms have been obtained, the dataset is split into train and test to construct machine learning models. The proposed models are: K-nearest neighbors (KNN), kernalized support vector machines (KSVM), and convolutional neural networks (CNN). The CNN, do not need the feature extraction described earlier, and will be trained and tested on the images directly. Once the models have been trained, we validate the results on the test partition.

For the KNN and SVM, we search which hyperparameters are the best for each model. The main focus of this paper was exploring kernel methods, as such an extensive search and development of kernels has been proposed. The best kernel for our problem has been

the Generalized Intersection Kernel [3], followed by the Gaussian Radial Basis Function, and the Polynomial kernel. In contrast to the literature read, the mixutre kernels [11] do not return better results than their independent parts, and the Log kernel [2] does not achieve great performances on image classification.

Looking at the overall results, we believe that the kernalized SVM can return comparable results to CNN in simple image classification problems. The results presented here are not comparable to the immense neural networks presented in recent state of the art implementations of this problem, such as the ones presented in [4] and [12], which have millions of parameters. However, compared to a not so fine-tuned neural network, kernalized SVM and CNN seem comparable.

Without doubt, we have learned a lot about the difficulties of creating a pattern matching computer vision model: from which color space to choose, to the selection of feature extraction and preprocessing, to selecting which models are the best, to selecting and researching different kernels to use. We have seen the flexibility and the power that kernel methods bring in adapting difficult problems by just switching the kernel. It is a laborious work that requires both human and computational time to develop a correct experimentation technique to present valid results, as we have done here.

7 Future Work

There are some limitations in our paper and places we wish to further this research.

- Color spaces: we briefly tested with other color spaces such as HSV and CIELAB, which in principal have good properties. However, the results given in intermediate results were not as good. While the proposed color space in section 3 was invariant to illumination and 2D, we found no literature talking how to achieve the same properties using HSV and CIELAB. A formal comparison between the different color spaces could have been produced, even if it was 2D color histograms versus 3D.
- Feature extraction: while color histograms seemed to work pretty well, there surges problems when classes have similar colors, as we saw in the confusion matrix. Hence, it makes sense to search for other features from the images to correctly classify whenever images have similar color distributions. At first we tried a few, the moments of the histogram, contours, etc. however, since one of the proposed kernels was based on histogram comparison, we decided to just use the color histograms.
- Kernalized distance metrics: while reading [7], the results seemed obvious that the Earth Movers Distance (EMD) seems to work very well on image retrieval

problems in comparing histogram distributions. Being a metric, it should be easy to propose a kernel from this distance, and they did in [5]. We tried applying the kernel, and while there are some EMD implementations in R, the original algorithm has complexity $O(n^3 \log n)$, which makes it unfeasible to train-test with various hyperparameters in the given time frame. While there are approximations in linear time [8], we feel it is out of the scope of this project to research and implement.

– Tuning hyperparameter: while satisfied with the tuning grid search, there are still some improvements to be made in the tuning of kernels, such as the power kernel and mixture kernels. For the power kernel, we decided to fix the scale to $\theta = 1$ and the offset to $c = 0$ in order to simplify both our code and the grid search. For the mixture kernels, we used the best hyperparameters found for both models independently and tuned the mixing coefficient (λ). We are assuming the best hyperparameters for each independent kernel will be the best for the mixture, however this assumption might not be accurate as there might exist interaction between the hyperparameters.

  Finally, combinations of more than 2 kernels could be tested in the same manner, using mixtures of mixtures. However, the complexity of these models might not achieve better results than simply the original kernels, as what happened in most of our results in section 5.

– Images without a class: while the images we tested all belong to a class, in real world applications this assumption does not need to hold. Hence, future work should be done to test the presented models with images they have never seen, i.e. images which belong to no class or images that contain multiple classes. Currently, the way the model is implemented, it returns the probabilities of belonging to each class and it assigns the image to the one with the highest probability. In order to take into consideration images that do not have a class, a probability threshold could be set so as to discriminate if the image belongs to the subset of images viewed or if it is a new class not belonging to the subset of known classes.

  Future work could be done in searching and setting this threshold which would probably require another machine learning model to minimize the number of miss-classifications. The framework is quite similar to the one presented in our code. We have decided against implementing it for two reasons. First, because the literature presented only took into consideration the subset of classes used. Secondly, this would strive the project away from studying kernels. However, future work could be done in implementing this.

8 References

[1] Christopher M Bishop. "Pattern recognition". In: *Machine learning* 128.9 (2006).

[2] Sabri Boughorbel, J-P Tarel, and Nozha Boujemaa. "Conditionally positive definite kernels for svm based image recognition". In: *2005 IEEE International Conference on Multimedia and Expo*. IEEE. 2005, pp. 113–116.

[3] Sabri Boughorbel, J-P Tarel, and Nozha Boujemaa. "Generalized histogram intersection kernel for image recognition". In: *IEEE International Conference on Image Processing 2005*. Vol. 3. IEEE. 2005, pp. III–161.

[4] Zihang Dai et al. "CoAtNet: Marrying Convolution and Attention for All Data Sizes". In: *arXiv preprint arXiv:2106.04803* (2021).

[5] Yulia Dodonova et al. "Kernel classification of connectomes based on earth mover's distance between graph spectra". In: *arXiv preprint arXiv:1611.08812* (2016).

[6] Brian D Ripley. *Pattern recognition and neural networks*. Cambridge university press, 2007.

[7] Yossi Rubner, Carlo Tomasi, and Leonidas J Guibas. "The earth mover's distance as a metric for image retrieval". In: *International journal of computer vision* 40.2 (2000), pp. 99–121.

[8] Sameer Shirdhonkar and David W Jacobs. "Approximate earth mover's distance in linear time". In: *2008 IEEE Conference on Computer Vision and Pattern Recognition*. IEEE. 2008, pp. 1–8.

[9] Karen Simonyan and Andrew Zisserman. "Very deep convolutional networks for large-scale image recognition". In: *arXiv preprint arXiv:1409.1556* (2014).

[10] D Tao. *The corel database for content based image retrieval*. 2009.

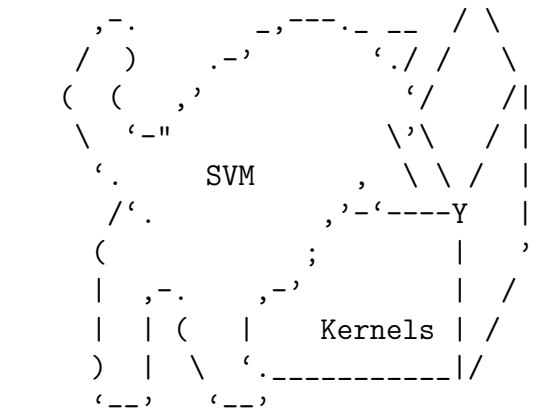
[11] Dongping Tian, Xiaofei Zhao, and Zhongzhi Shi. "Support vector machine with mixture of kernels for image classification". In: *International Conference on Intelligent Information Processing*. Springer. 2012, pp. 68–76.

[12] Xiaohua Zhai et al. "Scaling vision transformers". In: *arXiv preprint arXiv:2106.04560* (2021).

9 Annex

Kernel	0.001	0.01	0.1	0.5	1	2	10	
K_{GRBF}	0.000100	0.000100	0.000001	0.000100	0.000100	0.000100	0.000010	0.000001
K_P	2.000000	3.000000	3.000000	3.000000	2.000000	3.000000	3.000000	2.000000
K_{GIH}	0.700000	0.700000	0.250000	0.700000	0.700000	0.150000	0.700000	0.700000
K_{Log}	0.400000	0.100000	0.200000	0.700000	0.400000	0.200000	0.200000	0.100000
K_{LRBF}	0.000001	0.000100	0.000100	0.000010	0.000100	0.000100	0.000100	0.000100
K_{RBF-P}	0.800000	0.500000	0.800000	0.200000	0.500000	0.400000	0.400000	0.800000
$K_{RBF-Log}$	0.800000	0.400000	0.500000	0.800000	0.800000	0.800000	0.200000	0.500000
K_{Log-P}	0.200000	0.600000	0.200000	0.400000	0.500000	0.600000	0.400000	0.800000
$K_{RBF-GIH}$	0.200000	0.200000	0.200000	0.800000	0.600000	0.500000	0.200000	0.200000
$K_{Log-GIH}$	0.800000	0.200000	0.200000	0.800000	0.200000	0.400000	0.600000	0.200000
K_{P-GIH}	0.200000	0.800000	0.400000	0.800000	0.500000	0.200000	0.200000	0.500000

Table 8: Hyperparameters that returned the best CV error found for each of the C in the SVM (results of table 3), each value represents the main hyperparameter as described in Section 3.


SVM
Kernels