Practical Machine Learning - Project 2

- Unsupervised Scene Classification, 2022-2023 -

Ionescu Andrei, 407

andreei.ionescu@gmail.com

Abstract

The scope of this paper is to analyze the performance of unsupervised methods in the context of Scene Classification (Nitisha, 2018) and investigate different representations that can be used as data features. The unsupervised algorithms considered are DBSCAN (Deng, 2020) and AffinityPropagation (Frey & Dueck, 2007), and they use a clustering approach in order to group similar patterns in the data.

1 Dataset

In order to investigate the efficacy of unsupervised approaches in classification tasks a supervised baseline is needed in order to be able to quantify if the procedure gives satisfiable results or not. In order to do so, the dataset considered in the following experiments is **Scene Classification** (Nitisha, 2018), which consists of 17.000 labeled RGB images of size 150×150, and 7000+ test images that have no labels. The images are split into **6 classes**: Buildings, Forests, Mountains, Glacier, Street and Sea. In the following sections, only the training subset is considered in order to be able to evaluate the performance of the algorithms as mentioned.

1.1 Dataset Preprocessing

An initial inquiry of the dataset integrity showed that it did not contain any duplicated or missing image-class associations, however some of the images had **varying resolutions** which is a problem because algorithms and preprocessing steps usually expect a fixed amount of features. To overcome this issue, I analyzed the amount of images that violated this assumption and because their number was small relative to the size of the dataset I removed them.

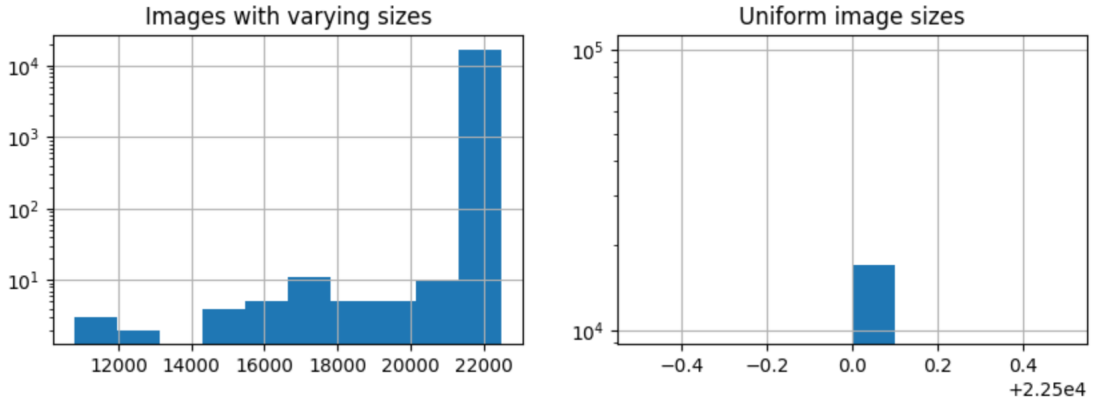

Figure 1: Removal of smaller images

The left histogram shows, using a logarithmic scale, that there are very few images with smaller sizes so that they can be easily removed with no impact.

As a next step I investigated whether the data is balanced or not, as this can skew the results towards more representative classes and would lead to bad clustering of data features. According to Figure 2, there are classes which have a visible amount of additional samples that are eliminated in order to have uniform distribution and reduce the possibility of introducing bias into the models. Additional preprocessing steps are particular to each feature extraction procedure and are described in Section 2.

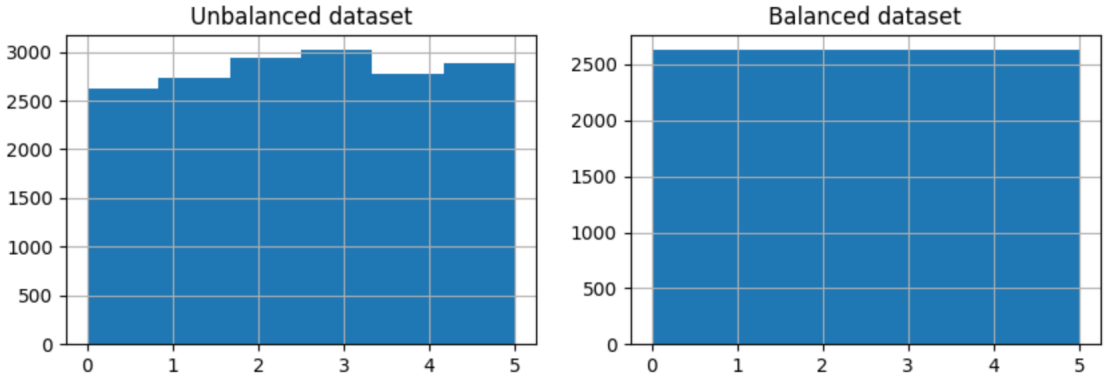

Figure 2: Dataset Balancing Using Undersampling
The classes with more samples were reduced to have an equal amount of data points.

2 Feature Extraction

An essential step in image processing is to extract **relevant characteristics**, which can give a richer representation, that can be then used by various algorithms on particular tasks. While deeplearning approaches allow end-to-end training on datasets with minimal manual intervention on the input, classical clustering models, such as those considered in this paper, can suffer from the lack of better representations and may miss important relationships found in the data. Motivated by this observation, I considered 4 feature extraction techniques that are discussed individually below. Each of the presented methods is considered as a **black-box feature extractor** that is applied on the raw data, and the resulting features are given to the selected clustering algorithms for the classification task.

2.1 Normalized Image Pixel Space

A naive approach is to use the pixel space of the images as the data features, by first applying **data standardization** per-channel with the mean and standard deviation values computed on the training subset. As an observation, because the clustering methods use distance functions to accept or reject samples into their clusters, this computation is done per-pixel. As a consequence, the *channel*, *height* and *width* dimensions can be collapsed, and this would not change the results of the distance functions. On the other hand, one weakness of this approach is that the distances only consider isolated pixel distances and neglect that neighbouring pixels are usually similar.

Another consideration can be taken with regards to the *channel axis*, with the possibility of transforming the images to grayscale. While this step may be applied it can result into a loss of information, because colors are important in order to decide that certain features such as leaves and grass may actually be related and represent a *forest scene*. Because of this, the color channel is kept as an important characteristic.

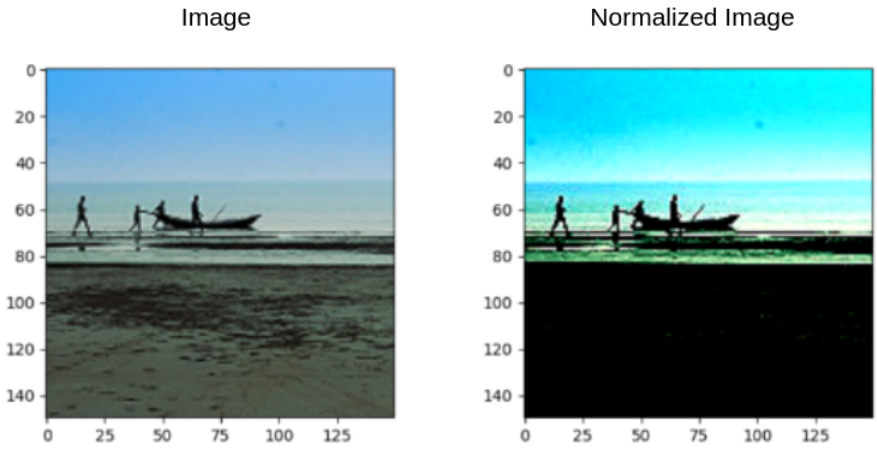

Figure 3: Image Normalization

2.2 Color Histograms

Rather than using directly the pixel space of the images, computing the **per-channel color histograms** might be able to give a more informative representation as the amount of color between related images should be similar. In order to obtain this representation, a histogram is computed for each color channel of an image, their value is normalized using the maximum count, and finally, they are concatenated. The histograms bins can be 256 or less, mapping nuances in the same interval for a coarse representation. An example unconcatenated histograms can be seen in Figure 4, which are then concatenated internally resulting in a feature vector of size $128 \times 3 = 384$.

In comparison to the previous technique, this approach allows the clustering algorithms to map images closer which have similar color palettes but placed in different regions. While this might be an improvement, it is also a weakness since the spatial information is completely lost. For example, one sea scene might be very similar to a glacier scene as both can have great amounts of blue nuances, but structurally they differ.

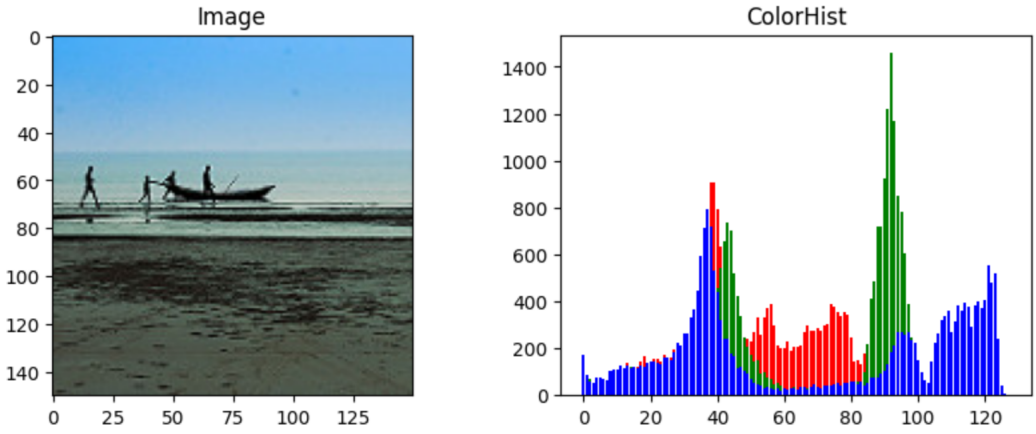

Figure 4: Per-Channel Color Histograms
The unconcatenatd per-channel color histograms of a 150x150 RGB image with 128 bins.

2.3 Bag of Visual Words Using HOG

A better approach is to extract relevant patterns / objects in the image using a **image descriptor** (such as HOG), cluster them in order to obtain **prototypes** that can better generalize similar patterns, and build histograms using the selected prototypes for each image. This representation is richer in the sense that clustering histograms of visual words will, in a sense, assign objects that have similar contents (textures, patches, objects, edges, etc.) closer to each other. The practical steps taken in order to build this representation consist of:

1. Apply **Histograms of Oriented Gradients** (Dalal & Triggs, 2005) over each image and obtain $N \times 16 \times 16 \times 3 \times 3 \times 9$ features in total. The dimensions indicate the amount of samples (N), the number of blocks (16×16), the number of cells inside each block (3×3) and a histogram with 9 bins of oriented gradients. The dimensions of the descriptors are then collapsed building an array of size $N \times 256 \times 81$.

2. The first two axes are then collapsed resulting in $(256 * N) \times 81$ elements. These array represents the concatenation of all descriptors resulting from all images.

3. In order to obtain prototypes for generalization, an *SKLearn MiniBatchKMeans* (n.d.) is applied with a given *n_clusters* value that is used for finetuning in a later stage. The resulting centroids represent the *bag of visual words vocabulary* (bovw).

4. The centroids are then used in the *inference stage* to assign an image's *descriptor features representation* to clusters. The cluster assignments are then counted using a histogram format, with the same length as the number of prototypes. Finally, this histogram is normalized using the maximum frequency count in it.

5. The end result is that the images are mapped to normalized histograms of size *n_clusters* and are used as features from this point forward.

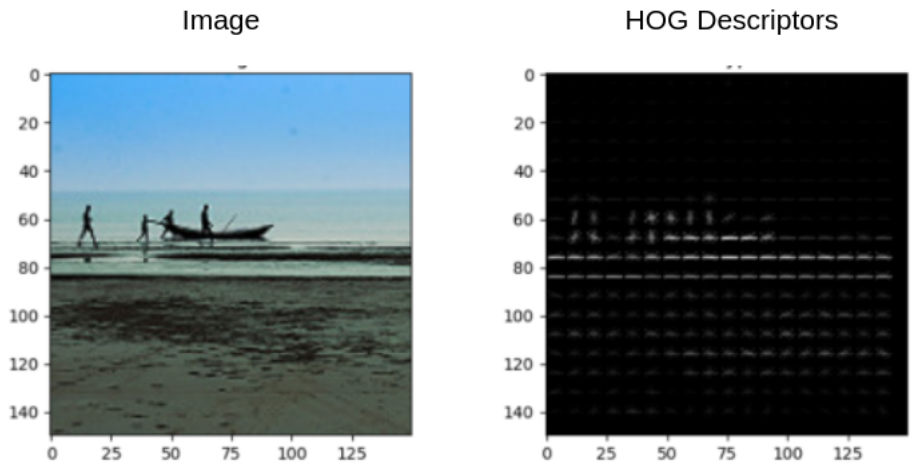

Figure 5: HOG Descriptors for an Image

Another image descriptor that I have tried to apply is **Scale-invariant feature transform (SIFT)** but had some issues when trying to perform grid searching.

2.4 Transfer Learning - Pretrained Feature Extraction

A more novel approach is to use a deeplearning as a backend for feature extraction by leveraging a pretrained vision classification model, and extract its **intermediary activations** as features for the clustering algorithm. This should offer good results because the model has already built high-level abstract representions, using a large dataset, and should be able to give insights for the current dataset because it contains similar images. In practical terms, the steps that I applied are:

1. Use a VGG11 Model (Simonyan & Zisserman, 2014) pretrained on ImageNet as the backend model.
2. Preprocess the images using the pipeline given by the model (normalization, scaling, etc.)
3. Freeze all layers because no training is performend.
4. Pass mini-batches of images through the model and extract the second-to-last activations which are arrays of size 4096 elements.

3 Unsupervised Approach

The clustering algorithms considered are DBSCAN and AffinityPropagation, both which infer the number of clusters. They are trained over the outputs given by the already detailed *feature extractors* and by clustering can build on top of them better generalizing representations:

- **Normalized Image Pixel Space** - Images with close pixel values, in each location, are more similar.
- **Color Histograms** - Images with similar color palettes should be closer to one another.
- **BOVW + HOG** - Images that contain similar keypoints / content (texture, patches, objects, edges, etc.) are brought together.
- **Pretrained Features** - Images that have close high-level representations, learnt before by a vision model, should have more similar semantic value.

4 Training and Results

In order to finetune the models I applied *SKLearn GridSearch* (n.d.) and the data was split as follows:

- 80% Train Data (12609 samples), of which is further split for hyperparam tuning:
  - 80% Train Data (10087 samples)
  - 20% Validation Data (2522 samples)
- 20% Test Data (3153 samples)

4.1 AffinityPropagation Algorithm

Feature Extractor	Valid Accuracy	Sillhouette
BOVW (10)	0.581	0.076
BOVW (128)	0.657	-0.023
BOVW (512)	0.628	-0.030
Activations	0.877	-0.031
ColorHist (256)	0.507	-0.035
ColorHist (128)	0.512	-0.023
ColorHist (64)	0.508	-0.029
ImageNorm	0.462	-0.022

Table 1: AffinityPropagation Hyperparam Tuning using GridSearch
Finetuning on the *damping* hyperparam led to very similar results, so they are omitted. On the other hand, the feature extraction is tuned to better fit the algorithm.

4.2 DBSCAN Algorithm

I tried to finetune the hyperparameters using GridSearch but poor performance was obtained regardless of the parameters used. The algorithm would either fail to find clusters or group all data into a single one. The hyperparams considered:

- p = 1, 2
- eps = 0.1, 0.15, 0.25, 0.5, 0.75
- min_samples = 5, 7, 15, 20
- feature_extractor

The results are as follows for the algorithm:

- 0.0 accuracy - no clusters
- 0.176 accuracy - single cluster
- 0.243 accuracy - manual finetuning (indicated by table below)

4.3 Test Results

For the supervised approach I considered the same VGG11 used in feature extraction, and finetuned its final two layers for a classification task on the current dataset with 6 classes. Adam optimizer was used with 2×10^{-4} *learning rate*.

Algorithm	Test Accuracy	Sillhouette	Params
DBSCAN	0.243	-0.075	p=2, e=0.1, m=5, fe=BOVW_HOG (512)
AffinityPropagation	0.884	-0.027	damping=0.5, fe=Activations
Supervised	0.94	-	epochs=20, batch_size=64
RandomChance	0.166	-	-

Table 2: Accuracy obtained on the test set

References

Dalal, N., & Triggs, B. (2005). Histograms of oriented gradients for human detection. , *1*, 886-893 vol. 1. DOI: 10.1109/CVPR.2005.177

Deng, D. (2020). Dbscan clustering algorithm based on density. , 949-953. DOI: 10.1109/IFEEA51475.2020.00199

Frey, B. J., & Dueck, D. (2007). Clustering by passing messages between data points. *Science*, *315*(5814), 972–976. DOI: 10.1126/science.1136800

Nitisha, B. (2018, Dec). *Kaggle scene classification.* Retrieved 2023-01-25, from https://www.kaggle.com/datasets/nitishabharathi/scene-classification

Simonyan, K., & Zisserman, A. (2014). *Very deep convolutional networks for large-scale image recognition.* arXiv. Retrieved from https://arxiv.org/abs/1409.1556 DOI: 10.48550/ARXIV.1409.1556

Sklearn gridsearch. (n.d.). Retrieved 2023-01-25, from https://scikit-learn.org/stable/modules/generated/sklearn.model_selection.GridSearchCV.html

Sklearn minibatchkmeans. (n.d.). Retrieved 2023-01-25, from https://scikit-learn.org/stable/modules/generated/sklearn.cluster.MiniBatchKMeans.html