Image Clustering Report

Danila Kurganov

December 2022

1 Task

The task of this problem is to cluster images that share scene information given a large series of images. I further consider that there are limited computational resources as well as drastic lighting changes in the images.

2 Approach

My general approach to solving this problem was to, given a dataset, extract features from images, construct low-dimensional representations of these features, and then cluster these lower-dimensional representations.

Choice of dataset which considered drastic lighting changes and same-scene information was hard to find - I wish I had been able to spend more time to find a better-suited dataset. In the end, the Caltech-101 dataset (hyperlink) was used. This image-classification dataset provides images of 101 objects taken from varying light sources, distances, orientations, and occlusions. For computational reasons, I consider the first 45 image classes available in alphabetical order. This results in considering 4700 images belonging to 45 classes, each of (downscaled) size $96 \times 96 \times 3$.

To extract features from images, SIFT, SURF, HOG, and neural networks were considered.

SIFT and SURF offer a good generation of features due to creating scale, rotation, and uniform lighting change invariant features. The features are also high-dimensional, and so robust to noise or some occlusions. Finding features is also relatively fast, especially with SURF. In tests, however, due to different images giving different feature vectors - unless matching was done for same-scene images to collect shared found features, each image would produce a differently sized vector, and so further clustering or dimensionality reduction was infeasible.

HOG is another feature detector considered due to its ability to detect objects invariant to lighting changes, rotations, scaling, and translations. To detect objects in space - subject to changes from a fixed reference frame, I found this to be suitable to the task at hand. Further, given fixed image sizes and fixed HOG parameters, the output features would be of even size - and therefore comparable for clustering algorithms. One further feature of HOG is that it more severely discerns images of different scene types - perfect for the problem posed.

Neural networks were also considered feature detectors. My motivation was that if a classification neural network was trained on a broad dataset involving changes in lighting, object translation, rotations, noise, and occlusions, then these learnt symmetries would carry over to discerning unseen data too, further disregarding the symmetries previously learnt. With these priors, and with interest in keeping computational costs low, I considered a pre-trained $MobileNetV2$ architecture. This net was published specifically for use by low-powered mobile devices, pre-trained on ImageNet, a dataset of 1.2 million images from 1024 classes. The specific net used generates feature vectors by disregarding its last SoftMax layer. By evaluating how correlated an input image is to each of the 1024 classes, a 1024-sized feature descriptor for an image is generated.

Feature Descriptor	**Compute Time** ($\mu \pm \sigma$)
MobileNetV2	16.1 ± 1.320 seconds
HOG	4.06 ± 0.531 seconds

Figure 1: Feature descriptor compute time for 4700 $96 \times 96 \times 3$ RBG images.

Low-dimensional embeddings are considered mainly for computational performance speedups but also because the current embeddings may have many unnecessary components. For example, in the context of PCA, many components may have so little variance that data points can't be discerned from these values. Therefore, it would

be beneficial to remove these dimensions as they could only provide noise. Many dimensionality reduction techniques exist. Due to needing clustering later in this analysis, along with the possibility that the data manifold from which the features are generated is of a non-convex shape, I consider several techniques, even if they are not optimal. By choosing one which worked best I would be overfitting my analysis to the Caltech-101 dataset.

Method	**Descriptor** $(n \times p)$	**Complexity**	**Compute Time** $(\mu \pm \sigma)$
PCA (50)	HOG (4700×1764)	$O(p^2n + p^3)$	0.265 ± 0.015 seconds
PCA (50)	MobileNetV2 (4700×1280)	·	0.164 ± 0.002 seconds
PCA (50)	HOG + MobileNetV2 (4700×3044)	·	0.378 ± 0.009 seconds
Spectral (50)	HOG (4700×3044)	$O(plog(k)nlog(n)) + O(pnk^3) + O(50n^2)$	20.3 ± 1.490 seconds
Spectral (50)	MobileNetV2 (4700×3044)	·	19.3 ± 1.780 seconds
Spectral (50)	HOG + MobileNetV2 (4700×3044)	·	22.3 ± 0.997 seconds
LLE (50)	HOG (4700×3044)	$O(plog(k)nlog(n)) + O(pnk^3) + O(50n^2)$	6.33 ± 0.774 seconds
LLE (50)	MobileNetV2 (4700×3044)	·	7.90 ± 1.260 seconds
LLE (50)	HOG + MobileNetV2 (4700×3044)	·	8.99 ± 0.921 seconds

Figure 2: Low-dimensional embeddings as applied to image descriptors. Data is standardised $(\mu = 0, \sigma = 1)$ before being reduced. n is the number of data points, p the input dimensionality, k the number of neighbours considered. All embeddings reduced the feature sizes to 50 dimensions. This was a hand-picked value. Spectral and LLE embeddings considered default neighbours considered as given by the 'sklearn' library: 470 for Spectral, 5 for LLE.

With low-dimensional embeddings generated, clustering is done. Although not shown here, I tried clustering directly from the non-reduced features for some of the examples and got worse clustering evaluation results. Therefore, I was more confident in reducing dimensionality without losing too much information before doing clustering. Ideally, I should have considered eigenvalue sizes when using PCA to determine the number of clusters to use using an elbow or the number of non-negative eigenvalues. This could further tell me about the non-euclidean-ness of the data in the feature space. Below I consider KMeans clustering on the data priorly reduced. A distinction was made between using just HOG features and Neural net features just in case one produced irrelevant features.

Data	**Elbow**	**Calinski**
PCA_{HOG} (4700×50)	25	25
$PCA_{MobileNetV2}$ (4700×50)	33	45
$PCA_{HOG+MobileNetV2}$ (4700×100)	28	40
LLE_{HOG} (4700×50)	49	50
$LLE_{MobileNetV2}$ (4700×50)	50	47
$LLE_{HOG+MobileNetV2}$ (4700×100)	46	46
$Spectral_{HOG}$ (4700×50)	45	40
$Spectral_{MobileNetV2}$ (4700×50)	38	40
$Spectral_{HOG+MobileNetV2}$ (4700×100)	42	40

Figure 3: Best number of clusters predicted according to the Elbow and Calinski metrics. 2 to 100 clusters were considered, with the Calinski-Harabasz index computed ±15 clusters to the optimal elbow value. Elbow values were found in an automated way.

One hyper-parameter of clustering models is in the number of clusters to make. For this analysis, I consider using the 'elbow' method and Calinski-Harabasz metrics. The elbow method calculates each cluster's distortion: the sum of square distances from each point to its assigned centre. The inflexion is a supposed good rule of thumb for the number of clusters to consider. To not overfit one metric, I further consider the Calinski-Harabasz index. This index measures the ratio of the sum of between-clusters dispersion and inter-cluster dispersion for all clusters. The higher the value the more dispersed and identifiable the clusters are. After analysis, however, neither the Calinski-Harabasz nor Silhouette scores provided useful information. I, therefore, omit implementing them, and stick with using the elbow. I believe considering the elbow is a good performance metric, as I can directly compare it with the number of clusters I expect to see given the dataset I used.

Visually, looking at the elbows of PCA, I would have considered clusters greater in number than that chosen by an automated algorithm. Nevertheless, doing non-linear dimensionality reduction via LLE or Spectral provided better convex regions for KMeans to find clusters with.

Of note: there wasn't a significant difference between using HOG, MobileNetV2, or both. Therefore, for purposes of computational efficiency in the context of a device with low-compute power, I considered just using HOG features.

After much investigation, a good resulting pipeline was found:

$$Data \rightarrow HOG \rightarrow LLE(50) \rightarrow KMeans(Elbow).$$

3 Future Improvements

There were not a lot of contexts given for this problem, therefore many of the improvements or considerations made above were due to some model of trying to optimise for a space-craft with low-compute power needing to do clustering of a large series of images of objects with different lighting, rotation, distance, translations from the camera's coordinate frame. This resulted in narrowing algorithm choice to methods of low-compute power, invariant to image changes. Nevertheless, some ideas of interest had to be skipped for reasons of time.

The data set to consider, and how to pre-process images could be improved. A data set more resembling the problem would be useful to use - especially one considering several images of the same scene taken from varying angles. I could only pseudo-represent this by using different images of the same named object - hoping that the finer details didn't mess up the analysis. Further, since the task was to discern images with the same scene information, it could be more beneficial to first apply COCO or another semantic mask extracting the foreground from the background. Then use a similar pipeline as above except doing feature extractions of only the background of images.

Dimensionality reduction could also be done with an Image to Words model; this way the feature space is ensureably more dispersed, and thus a better clustering could be done. Alternatively fancier deep neural network models like autoencoders could be considered for dimensionality reduction purposes. I believe the above method is sufficient, but these are worth trying out.

Other improvements, in no particular order:

- I used fairly default HOG detector hyperparameters; studying the effects of each one would be an interesting lead.

- In doing any of the embeddings (features to low-dim features), I occasionally tried concatenating parts of the input features to construct other low-dim features. Although my analysis didn't prove useful, I would like to consider the effects of concatenation on low-dimensional embeddings, along with clustering in a more controlled manner. In U-nets this concatenation is done, so I wonder if there are benefits I have missed out on having not tried more concatenation methods.

- The dataset considered was only one, using more datasets to not overfit my results on one dataset would be a good next step.

- There are many dimensionality-reducing schemes, I only considered 3, however it would be good to read the literature on theoretical differences between each one to get a better picture of the suitability of each.

- Considering eigenvalue tests and other fancy measures to see if the linear-reducing scheme was suitable. It may be that the data can be reduced linearly, and if a test exists before needing to compute non-linear reduction schemes, computational efficiency along with clustering performance can be improved.

- Suitability of K-means as a clustering algorithm. Many other clustering algorithms exist, especially ones considering non-convex clusters. I would like to read the literature on the theory of different algorithms and consider other ones. I stayed away from these in this analysis as they had further hyperparameter tuning that I did not understand.

- In my code, I would like to have implemented more visualisation functions to see clustering in action, an easier-to-use dataset loader function, more neural network models that could be used instead of the chosen MobileNet, and more clustering metrics considered. If the training dataset is labelled; more clustering accuracy metrics can be used.

- For time reasons I ended up using libraries to implement parts of my code. Some things, like the HOG descriptor, can be hardcoded fairly easily. The Same with PCA. Doing these would be beneficial for running an efficient set of codes to do clustering with.