Computer Vision Final Project

By Jiaming Hu

March 15th, 2022

General Procedure

We are given 3 sets of texture images in 47 different categories. We have the training set, the validation set and the test set. Our goal is to use the training set and the validation set to train a model such that given an image in test set, we can correctly recognize its category. The way we do it, in short, is by converting each image into a vector representation, and then use some multi-class classifier to assign each image a category.

The first step is to convert each image into a vector representation. To do that, **1)** We first choose a set of image filters that we call filter bank. In our project we used an existing filter bank, called LM filter banks, which consists of 48 filters of size 49×49. We will use all those 48 filters to do convolution on each image in all 3 sets, which will for each pixel produce a vector in $\mathbb{R}^{48}$. Take the training set for example, consider the first image, which is of size 640×480. If we do convolution on this image with all 48 filters, then we would have $640 \times 480 = 307200$ of vectors in $\mathbb{R}^{48}$. We repeat this for all the images in the training set. **2)** Then we take all those vectors and apply KMeans on them with a desired choice of k. In our project we take $k = 128$. This will give us 128 of $\mathbb{R}^{48}$ vectors as cluster center which could be used to represent each images later. Those are our bag of words. **3)** Now, we go back to the the training set. Again take first image for example, for all each of 307200 vectors representing each pixel, we can use our KMeans model to assign one of the clusters as its label, which gives us 307200 integers between 1 and 128. Note that we need to normalize this $\mathbb{R}^{128}$ vector, because each image has different image size hence different number of pixels. Normalization would solve this problem. We do this for each images in the training set. **4)** We now can compute a histogram of frequencies of each number's appearance based on those integers, we will use as a representation of the image. This would produce for each image a $\mathbb{R}^{128}$ representation.We do the same thing for validation set and training set.

Now what we have is 1880 of $\mathbb{R}^{128}$ vectors for each set. We then use some classifier to assign each image in validation set a label. The purpose of validation set is that for some classifier, such as random forest and ridge regression, we are able to choose a good parameter for them. Then we can get our result from the test set, and that would be the accuracy of our method.

Difficulties

In this section we will discuss the difficulties that I encountered and how I solved them.

Data Size of the Problem

The first difficulty that I encountered was the number of pixels we need to deal with. This is argubaly the most difficult part of this project for me.

Remember that each pixel in our problem will produce a vector in $\mathbb{R}^{48}$. Now for each set we have 1880 images, so we have 5640 images in total, with each image being approximately $600 \times 500 = 300,000$ pixels, so in total we have about 1.7 trillion of 48 dimensional vectors. This is way more than what we can deal with. It is true that if we spend days that we could compute all those vectors, but it will take approximately 1.5 TB of storage to store them, let alone later we will use them to train our model. There is no computer that has that much RAM, at least within my access.

The way I dealt with it was by selecting every 5^{th} pixel, both alone x-axis and y-axis. We can reduce the size of data by a factor of 25 in this way, i.e. we are using only 4% of the whole dataset. This reduce the amount of RAM required to about 20 GB for one set of data. Even later we need to use all 3 sets, 60 GB of RAM is not that much, and we can always use smarter ways to reduce the amount of RAM in any given time. In fact, if we use numpy.load() function to store results of convolution, it only requires about 20 GB RAM to do all the calculations. Note that it is not possible if we use all data points.

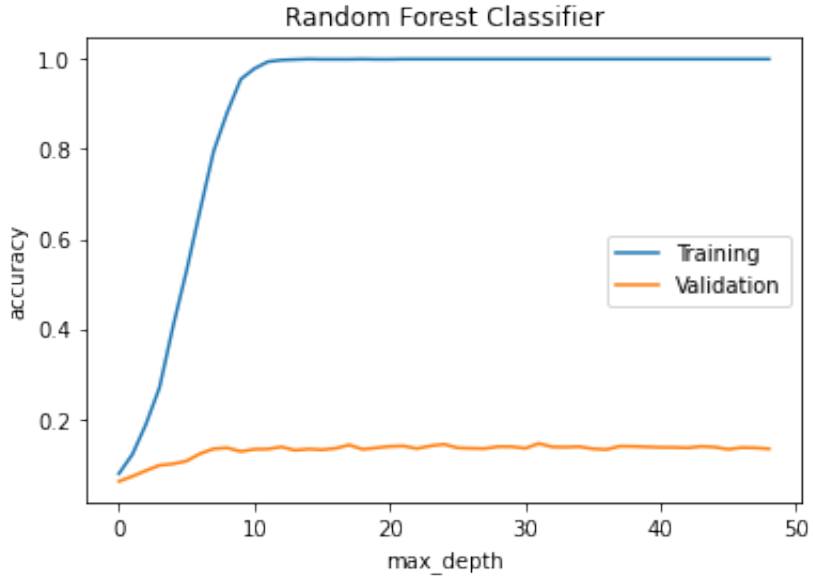
I used my own convolution function to do all the calculations, which took me about 30 hours in total. One could definitely come up with a better strategy to shorten the computation time, but mine would suffice for this project since I have 3 different PCs, so it took me about 10 hours to get all the data I need. The code that generate data points (all those $\mathbb{R}^{48}$ vectors) has been attached to this project.

Convergence of KMeans

If we just apply KMeans with $k = 128$ on the whole dataset (though we only have 4% of the whole dataset), it will take hours to converge. I chose $k = 128$ because it is more than enough to represent our data since we only have 1880 images. I used what was suggested in the assignment PDF. Namely, I first use 1% of the data and compute first set of k clusters (128 of $\mathbb{R}^{48}$ vectors). Then I use these k clusters as initial guess to calculate a KMeans model on 3% of the dataset. Then I repeat the same procedure for $5\%, 10\%, 20\%, ..., 100\%$ of the data. It only took me about 6 minutes to let the algorithm converge.

Choice of Classifier

After we use KMeans to calculate vector representation of each image, we need to choose a classifier to assign each image in the validation set a category. This step is actually non-trivial. For the same data, different classifiers bring very different accuracy. I tried a number of classification methods including K-nearest neighbors, Random Forest, SVM, Logistics Regression and Ridge Regression. It turns out that Random Forest gives the best accuracy.



We can see that a good choice of max depth in random forest would be some integer greater than 10. For best accuracy I chose max depth being 31, which gives accuracy in validation set being 14.6% and in test set being 14.1%.

Here I give the accuracy for other classifiers, all calculated on the test set. KNN has an accuracy of 7.6%. SVM has an accuracy of 9.6%. Logistics regression has an accuracy of 6.5%. Ridge regression has an accuracy of 8.8%.

Result

After we follow the steps in procedure section and use random forest in classification step, we would get an accuracy of 14.1% in test set. This number might seem low but it was the best I could do. Consider the fact that we have in total 47 categories, so by random guessing we would get an accuracy of about 2% and our method is 7 times better than guessing. Consider the fact that we only used 4% of the data, this is not as bad.

The most confused category is pleated, which has an overall accuracy 0, which means we never assign images in this category to its correct label.

One improvement my method could use would be calculate more pixels, because again we only used 4% of the total data. The accuracy should be improve had we used more pixels. Or one could in some way apply only a window of size 100×100 (or any other size) selections of pixels to do convolution, but that window of pixels has to be distinct enough to represent the image. The good side about this method is that all the pixels are continuous, whereas in our method the pixels we chose were discontinuous in a sense that they are not adjacent to each other.

Discussion

The files I used to store my computed data are in total have size about 20 GB, so I will not submit them as attachments. But I show my code of how I compute them. One just have to download the folder that contains image and everything will work.

I tried other different ways but the one that I presented yields the best result.

I have tried to use all the data points, but it ended up always kill my kernel and I could do nothing about it, which makes the procedure impossible to continue. One should always remember to store calculated data into local machines because the data we are dealing with is rather big, and it is possible that kernel would die and one could lose everything.

Also in the step of constructing cluster centers for KMeans, I tried to compute 128 for each category, which gives me $128 \times 47 = 6016$ features in our bag of words. This gave pretty good result but since our method only requires 128 features, it is better than compute features for each category and put them together.