1 Image Categorization

1.1 Pipeline

We start off by creating a codebook, this is done by iterating over all positive training images and computing a set of grid points. The grid_points method was implemented using the integrated matlab function meshgrid. We then compute descriptors for each of these local features using the descriptor_hog function. Our histogram of oriented gradients method uses the integrated atan2 function to compute the angles of the gradients which were provided, we then split them into eight bins and return a $N \times 128$ vector where N is the number of grid points in the image and $h_{cell} * w_{cell} * \#_{bins} = 4 * 4 * 8 = 128$ dimensions. These descriptors together with patches for visualizing the features, are appended to vFeatures and vPatches in the create_codebook function. The resulting mosaic of patches can be seen in Figure 4. We then cluster the features using k-means.

The kmeans function chooses k centers randomly from the features, it then repeats the two main steps of k-means: assign each feature to its closest center, move cluster center to center of mass by taking the average over all features assigned to that cluster. The nearest neighbor is implemented in the findnn method and simply iterates over all features and finds the closest cluster center through euclidean distance.

We now create bag-of-words histograms for each image separated into positive and negative training samples. For this we repeat much of the same process as before, we first compute a set of grid points per image, use the descriptors_hog function to generate descriptors over these local features. We then use the bow_histogram method to obtain a bag-of-words vector which we append to the other bag-of-words vectors and which we then return as either the positive or negative bag-of-words matrix, depending on whether we used the positive or negative training data. The bow_histogram method finds the nearest cluster center of each feature and subsequently counts how many features have been assigned to each cluster, this results in a $1 \times k$ vector where k is the number of clusters. This same procedure is repeated to generate bag-of-words histograms for the test data. Once this is complete the classifier is called.

For the nearest neighbor classification we go through each image and classify their histogram, if the closest cluster center is in the positive bag-of-words vector we assume a car is present and return a true label, else we return a false label.

For the bayesian classifier we compute the parameters of a normal distribution for both the positive and negative bag-of-words histograms we then use the integrated normpdf function to sample the propability density function of the normal distribution at each position contained in the histogram vector. The resulting value is $p(U(i)|N(\mu(i), \sigma(i)))$, where i is the current histogram index. This is repeated for both the positive and negative normal distributions and using the hint provided is then summed up over the logarithm of the returned probabilities. Using hint c) the equation then simplifies to $(p_{tot}^{pos} > p_{tot}^{neg})$.

1.2 Performance

The following table illustrated the percentage of correctly classified images through five runs. On average the classification for this set is stable at classifying correctly around 93% for the nearest neighbor and 88% for the bayesian classifier. We can see that the bayesian classifier never outperforms the nearest neighbor classifier, this most likely has to do with the assumptions of naïve bayes which are not met. We cannot assume that the histogram counts are gaussian distributed and they are not independent of the labels. However, the codebook has the biggest impact on performance, since this is generated in a non-deterministic manner the performance can vary strongly between runs.

run	nn	bayes
1	0.92929	0.88889
2	0.9697	0.87879
3	0.93939	0.88889
4	0.91919	0.91919
5	0.90909	0.84848
avg	0.933332	0.884848

Figure 1: codebook visualization using 200 clusters.

class.	k=10	k=50	k=100	k=300	k=500
nn	0.91919	0.93939	0.9596	0.86869	0.90909
	0.88889	0.90909	0.9596	0.9697	0.86869
	0.9596	0.91919	0.9697	0.93939	0.9596
bayes	0.94949	0.80808	0.93939	0.90909	0.84848
	0.84848	0.81818	0.76768	0.80808	0.86869
	0.87879	0.79798	0.87879	0.80808	0.81818

For the different cluster experiments we ran each cluster setup 3 times and selected cluster sizes 10, 50, 100, 300 and 500. The resulting classification scores can be seen in the table above. We note that when we have a small cluster size, we do have enough features to tell if there is a car in the image and hence the performance is not optimal. If we increase the cluster size we obtain better results, however once we reach a certain point where the features in our codebook do not add any new information the performance decreases again, at this point we only add noise. On average we obtain the most stable results with $k = 50$ for both methods, nearest neighbor overall achieved the highest score with $k = 100$ clusters which seems to indicate a good cluster size in terms of information content in the codebook for this particular data set.

1.3 Bonus: Controller Database

For our own database we took 50 images and tried to classify those that contained an xbox or playstation controller. Of these 50 images we took 26 images which contained either some other peripherals like a computer mouse or nothing at all and 24 positive images containing a controller. The images were shot with different backgrounds and different lighting. To test our classification we picked around 25% of the images from both the positive and negative set. A positive example can be seen in Figure 2 and a negative example in Figure 3. We used the same default setup as for the cars, i.e. $k = 200$ clusters.

The performance was better than chance but did not get anywhere close to the car dataset classification, we assume this has to do with the bigger set of images and the more uniform lighting conditions. We ran the classification ten times and the resulting average for the nearest neighbor classification was 63% with a deviation of around 4, for the bayesian classification the average was around 59% with a similar deviation. The codebook (seen in Figure 4) shows a lot of background patches are being considered for classification, which is a reason for the bad performance, this could be solved by using more images.

(a) (b)

Figure 2: example positive images

(a) (b)

Figure 3: example negative images

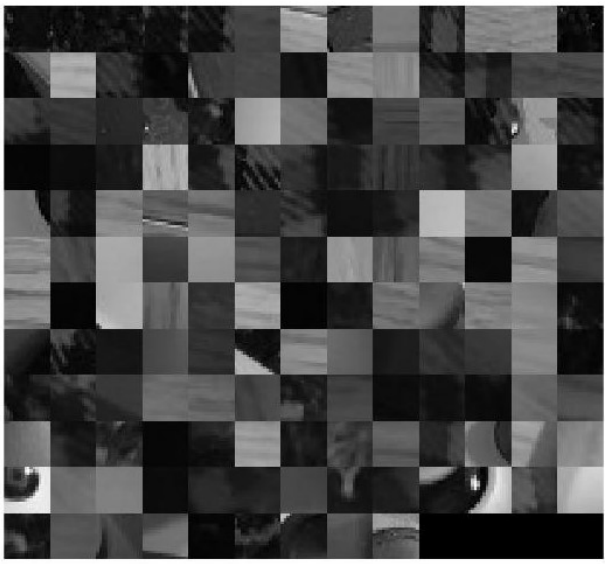

Figure 4: codebook visualization of the controllers using 200 clusters.