Computer Vision 1: Final Project

Bogdan Teleaga, 11413565
Mircea Mironenco, 11164182

April 6, 2017

1 Bag-of-Words based Image Classification

1.1 Introduction

We have constructed an image classification pipeline based on the bag-of-words approach where a visual vocabulary is built using clustered image descriptors extracted from a training set using a feature detection algorithm. The following sections describe what dataset we are utilizing, what the architecture of the pipeline is and what parts can be adapted.

We run several experiments in an attempt to understand what hyperparameter settings would be ideal given the specified classification problem. All experiment results can be found in the folder **htmls**. The images in the HTML files assume that the *Caltech4* folder resides in the parent directory.

To run a specific experiment, use the **run_experiment.m** function. To run a grid search series of experiments, run the **main.m** function.

1.2 Data

Our dataset is comprised of 4 classes of images, taken from the Caltech Vision Group [1]. In total there are approximately 1800 training images and 200 test images, i.e. 50 test images per class. The classified objects are **airplanes**, **motorbikes**, **faces** or **cars**.

In our experiments the number of training images used for constructing the visual vocabulary and for training are hyperparameters, and several options are tested.

1.3 Pipeline

Feature descriptors are extracted from the images, using the SIFT algorithm([2]). The color space of the images is a hyperparameter (i.e. the image is transformed into the specified color space before extracting descriptors from it), and several options are considered: **RGB**, **normalized RGB**, **opponent** colors, **grayscale**, and as a bonus **HSV**.

Keypoint and Dense SIFT([3]) variants are used, where the dense variant is fast algorithm for the calculation of a large number of SIFT descriptors of densely sampled features

of the same scale and orientation. More specifically, features are identified for each color channel and stacked together.

Keypoint SIFT works by first converting the image to grayscale, finding key-point features in that color space, and then identifying and concatenating descriptors for each color channel of the original image.

We construct a visual vocabulary by running the K-Means clustering algorithm to obtain our centroids. Each image is then represented as a collection of visual words, by extracting descriptors (again using SIFT) and assigning them to the closes words in the vocabulary. Images are finally represented as a histogram of their visual words.

These will be our input feature to a linear classifier. We train and use an SVM classifier, using the kernel type as a hyperparameter, to classify our test images.

The **experiments** performed, involve comparing different settings of parameters when constructing the pipeline. The following are studied, with their results being illustrated in the following section:

- KeypointSIFT vs DenseSIFT.
- Type of color channel for SIFT: RGB, normalized RGB, opponent, grayscale, HSV.
- Linear vs RBF Kernel.
- Vocabulary size.
- Number of training images used for vocabulary construction, and as training input to the linear classifiers.

1.4 Results

Results are illustrated in the tables that follow. The collumns correspond to:

- V - Size of the vocabulary
- Type - DenseSIFT or KP-SIFT
- Classifier - SVM kernel used.
- Colorspace - The color space the image is transformed to before extracting descriptors.
- Airplane, Bike, Face, Car - Average precision over individual classes.
- MAP - Mean Average Precision.
- # images - Number of training images used for constructing the vocabulary, number of images used for training the SVM classifier.

MAP and AP / class for several experiments are presented in Table 1 and 2. Respectively, a clear comparison is made between several **SIFT feature extractor types**, SVM kernels, and Dense/Keypoint SIFT variants respectively.

We can observe that in all instances, the Dense SIFT variant outperforms the keypoint variant. The dense sampling over several color spaces aiding the learning process and producing more robust features. It is clear that the KP SIFT variant does not produce features on manifolds that are as easy to separate as the Dense variant.

We further observe that the **Grayscale** and **RGB** color spaces produce the higher mean average precision, across all parameter spaces.

With respect to the type of kernel used by the SVM classifier, the **linear** one performs considerably better compared to the**radial basis function** alternative.

Table 1: Results for Dense SIFT with 400 Vocabulary words.

id	V	Type	Classifier	Colorspace	Airplane	Bike	Face	Car	MAP	# images
1	400	Dense	Linear	Grayscale	0.993	0.952	0.984	0.970	0.975	200 - 200
2	400	Dense	RBF	Grayscale	0.149	0.985	0.352	0.264	0.437	200 - 200
3	400	Dense	Linear	nRGB	0.939	0.969	0.909	0.939	0.939	200 - 200
4	400	Dense	RBF	nRGB	0.574	0.986	0.600	0.316	0.619	200 - 200
5	400	Dense	Linear	opponent	0.902	0.789	0.971	0.949	0.903	200 - 200
6	400	Dense	RBF	opponent	0.715	0.806	0.476	0.295	0.573	200 - 200
7	400	Dense	Linear	RGB	0.984	0.918	0.997	0.985	0.971	200 - 200
8	400	Dense	RBF	RGB	0.175	0.928	0.392	0.236	0.433	200 - 200

Table 2: Results for Dense SIFT with 400 Vocabulary words.

id	V	Type	Classifier	Colorspace	Airplane	Bike	Face	Car	MAP	# images
9	400	KP	Linear	Grayscale	0.712	0.730	0.692	0.567	0.675	200 - 200
10	400	KP	RBF	Grayscale	0.143	0.970	0.344	0.200	0.414	200 - 200
11	400	KP	Linear	nRGB	0.716	0.728	0.586	0.668	0.675	200 - 200
12	400	KP	RBF	nRGB	0.199	0.985	0.312	0.194	0.423	200 - 200
13	400	KP	Linear	opponent	0.703	0.644	0.795	0.835	0.744	200 - 200
14	400	KP	RBF	opponent	0.143	0.922	0.328	0.200	0.398	200 - 200
15	400	KP	Linear	RGB	0.648	0.701	0.801	0.724	0.718	200 - 200
16	400	KP	RBF	RGB	0.143	0.970	0.337	0.200	0.413	200 - 200

In Table 3, we see the results of testing the impact of the **vocabulary size**, only on the Keypoint variant, due to the high training time cost. Given that the Keypoint variant was not able to learn a clear separation between the classes (we have a 1:3 dataset representation, and as such any result with MAP ¡ 0.75 is hard to interpret), there is no visible improvement seen when increasing the vocabulary size.

Finally, our last experiments test the impact of the **number of training examples** used for constructing the vocabulary size, and for training the classifier. We perform these

Table 3: Same settings as in table 2, with a vocabulary size of 800.

id	V	Type	Classifier	Colorspace	Airplane	Bike	Face	Car	MAP	# images
17	800	KP	Linear	Grayscale	0.704	0.641	0.659	0.674	0.669	200 - 200
18	800	KP	RBF	Grayscale	0.140	1.00	0.351	0.194	0.421	200 - 200
19	800	KP	Linear	nRGB	0.726	0.519	0.552	0.588	0.597	200 - 200
20	800	KP	RBF	nRGB	0.139	1.00	0.312	0.192	0.411	200 - 200
21	800	KP	Linear	opponent	0.682	0.581	0.783	0.787	0.708	200 - 200
22	800	KP	RBF	opponent	0.140	1.00	0.316	0.194	0.412	200 - 200
23	800	KP	Linear	RGB	0.703	0.597	0.683	0.649	0.658	200 - 200
24	800	KP	RBF	RGB	0.193	1.00	0.331	0.192	0.416	200 - 200

experiments using the Dense SIFT variant, which performs considerably better. Furthermore, we introduce a bonus experiment using the **HSV** color space, which outperforms all other classifiers in terms of MAP, even when using only 50/55 training examples.

Note: All Dense SIFT experiments were conducted with a *step size* of 10 and a *block size* of 5.

Table 4: Results for less training images, HSV & RGB.

id	V	Type	Classifier	Colorspace	Airplane	Bike	Face	Car	MAP	# images
25	400	Dense	Linear	RGB	0.921	0.861	0.960	0.879	0.905	100 - 55
26	400	Dense	Linear	HSV	0.942	0.857	0.954	0.982	0.933	100 - 55
27	400	Dense	Linear	RGB	0.960	0.906	0.970	0.951	0.947	50 - 50
28	400	Dense	Linear	HSV	0.969	0.972	0.992	0.982	0.979	50 - 50

1.5 Conclusion

We have seen that on average the RGB and HSV color spaces perform better than their alternatives. Vocabulary size doesn't have a conclusive impact on the classification accuracy, so a default value of 400 is sufficient in most cases.

Dense SIFT is computationally expensive, but has the largest impact of all the pipeline 'parameters' we can tune. Lastly, the linear SVM kernel outperformed RBF, even with the highly non-linear nature of the dataset.

2 Convolutional Neural Networks for Image Classification

2.1 Network Architecture

Convolutional Neural Networks are architectures that make the explicit assumption that our inputs are images. As with normal neural networks they are composed of layers that transform volumes of activations to another through differentiable functions. In

convolutional architectures the neurons in a layer are connected only to a small portion of the previous layer. We construct these intermediate layers using a *filter* which we *convolve* on the previous layer (taking also the volume into account) while sharing the parameters. The filters parameters of our network, that are initially randomly initialized and that we learn as the network trains. With them we perform dot product operations with the values of the previous layers and obtain an activation map for each filter. This process allows the network to construct higher level representations (learned later in the network) from spatially combining lower level representations (learned earlier in the network).

1. The network essentially compresses the data into features hierarchically. Across the layers we can notice how the data size goes down and the filter sizes go up. This means we start with local features(small filters) and use them to increasingly build up more important features(bigger filters) until we reach a fully connected layer with a softmax classifier at the end. The initial filters of the network learn to identify basic shapes, and lines. The further layers build on top of the previously learned filter and identify increasingly more complex structures (shapes, faces, etc.).

2. Layer 10 of the network is the one that has the largest number of parameters. It is the layer that does the final convolution and applies 64 filters of size 64.

2.2 Preparing the data

We used the inbuilt `imageDatastore` to implement the `getCaltchIMDB` function. To deal with grayscale images in the dataset we transformed them into RGB by replicating the grayscale information across the three channels. For this first step the only transformation applied to the data is the one that was already in the file, namely mean shifting.

2.3 Updating the network architecture

The last layer of the given network is of size *64x10*. Thus, to use it for our task we needed to convert that to *64x4*, which meant setting `NEW INPUT SIZE` to 64 and `NEW OUTPUT SIZE` to 4.

2.4 Hyperparameters

For this section, we tested the network with different hyperparameters:

- **batchSize**: [50, 100]
- **numEpochs**: [40, 80, 120]

As can be seen in figure 1, which illustrates the P@1 for 40 epochs using both batchSizes, the network converges within 30 epochs. Thus we did not include plots for the other cases since they are rather similar. The results of evaluating the networks on the test set can be seen in Table 5. As expected from the plots, the results are very similar for all combinations.

Noticing that the network converges within 30 epochs, we have tried lowering the learning rate either by a factor of two or ten, however without any improvement. Most of our trials resulted in a test error of 3% or lower, but we could not reach an accuracy of 1, usually getting one example wrong. In some pathological cases the lowering the learning rate resulted in the network getting stuck too early and achieving an accuracy of 74%.

Table 5: P@1 for different combinations of hyperparameters.

batchSize \ numEpochs	40	80	120
50	99%	99.5%	99.5%
100	99%	99.5%	99.5%

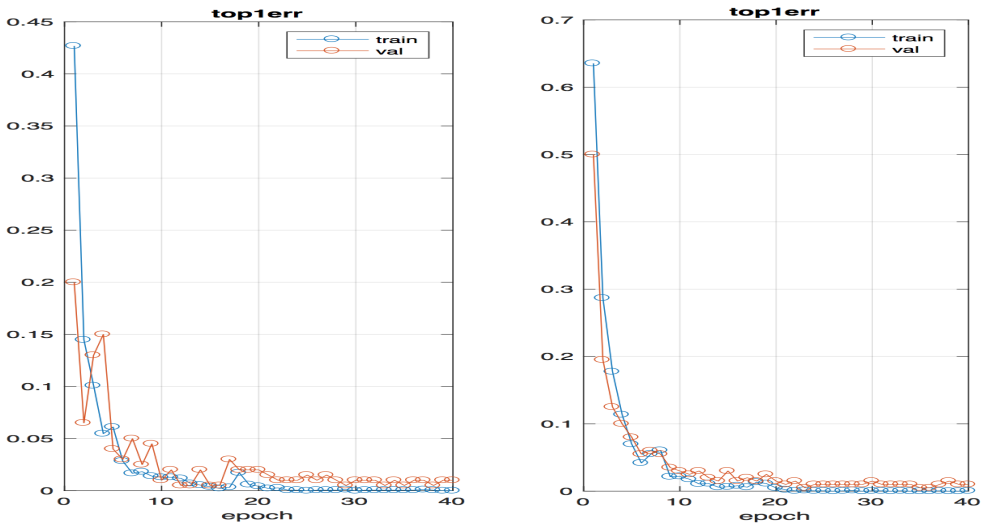

Figure 1: P@1 for batch size 50 (left) and batch size 100 (right)

2.5 Additional experiments

2.5.1 Data augmentation

We have tried different methods for data augmentation. Firstly, we have normalized the data by dividing using the standard deviation. When sampling one batch, besides the given `fliplr` a number of other transformations are applied randomly.

- **Rotation and Cropping**: When read from the disk the image is resized to be bigger than the input size to the network. It is then rotated by a degree chosen randomly from [0, 90] and cropped with a window that is the same size as the input to the network. The cropping window is placed in one of the 4 corners of the image or in the center.

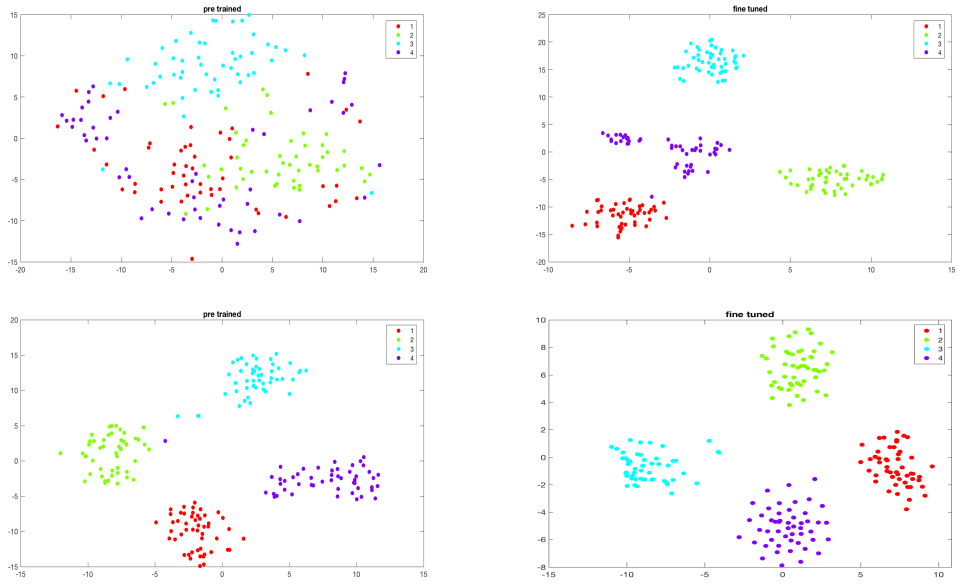

Figure 2: tSNE for different feature sets. Pretrained networks(left) vs finetuned(right) and LeNet(top) vs AlexNet(bottom)

- **Color perturbation**: Fancy PCA is applied by using the method described in the original AlexNet paper [4].

2.5.2 Deeper networks

We have tried using AlexNet and finetuning it to improve our performance. The model was downloaded from Matconvnet's website and is the one from `beta16` imported from Caffe. This is also one of the reasons we chose the augmentation steps above, which are both similar to what was used for AlexNet.

2.6 Results

For the regular experiments we have used a batch size of 50 and a number of epochs of 120, since those provided the best results for both LeNet and AlexNet. Furthermore, the data augmentation steps and freezing have been applied to both the provided net and AlexNet.

2.6.1 Visualization

For the visualization we have used tSNE. We can notice in Figure 2 how on LeNet the pretrained features cannot be used to cleanly separate the data, while after fine tuning we have a clear separation. However, for AlexNet we can see how both the pretrained network and the finetuned network can achieve a clean separation of the data.

Table 6: Accuracy for different network configurations

	CNN	SVM Pretrained	SVM Finetuned
LeNet	99.5%	89.5%	99.5%
LeNet with extraPreprocessing	98.5%	94%	99%
LeNet frozen	76%	89.5%	89.5%
AlexNet	100%	100%	100%
AlexNet with extraPreprocessing	98.5%	100%	100%
AlexNet frozen with extraPreprocessing	99%	100%	100%

These findings were confirmed by our experiments with freezing the networks and trying to train a classifier only using the features from the last layer as well as by the SVM experiments shown in the next section.

2.6.2 Accuracy

For accuracy we present the results obtained by the:

- Network
- SVM using pretrained features
- SVM using finetuned features

As we can see in table 6, for LeNet the pretrained accuracy is suboptimal which is expected given the output from tSNE. This is further enforced by the results using the frozen network. As far as the data augmentation step goes, we have found it to increase the variance in the dataset too much for this particular network leading to a decrease in performance as opposed to the expected increase. However, we can still see how the normalization step increased the performance of the SVM using pretrained features.

For AlexNet we can observe how the pretrained SVM already achieves a perfect accuracy, which again confirms our findings from the previous subsection. Again we found it hard to tune the network when applying data augmentation and training the network without any data augmentation at all resulted in a better performance than with it. We believe this is due to a wrong choice of learning rate, which we took to be the same as for LeNet, but training AlexNet is rather costly and we did not have the time to try a lot of different parameters. This is also the reason why we have not trained AlexNet only frozen without applying the data augmentation step.

Unfortunately, due to the size of the matrices we cannot send the weights+data matrices through email. We can provide them if needed. However, running LeNet should be doable and is on the order of minutes, while AlexNet can prove to be more tricky and generally takes at least a couple of hours.

References

[1] http://www.vision.caltech.edu/.

[2] David Lowe. Distinctive image features from scale-invariant keypoints. 2004.

[3] Koen van de Sande, Theo Gevers, and Cees G. M. Snoek. Evaluating color descriptors for object and scene recognition.

[4] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. Imagenet classification with deep convolutional neural networks. In *Advances in neural information processing systems*, pages 1097–1105, 2012.