Computer Vision 1-Final project

Michael Mo
University of Amsterdam
michael.mo@uva.student.nl

Changxin Miao
University of Amsterdam
changxin.miao@uva.student.nl

1 Introduction

In the final project, we investigate two approaches for the classification of images based on certain object classes. The first approach is a classification system based on a Bag-of-Words (BoW) model. The general idea of this approach is that at first an "visual vocabulary" is created from a training set of relevant images, after which test images can then be classified based on which "visual words" they contain and the corresponding frequencies. The second approach is based on a convolutional neural network (CNN) to classify the images. Hereby the main idea is that features of images are gained by filtering the image with multiple filters after each other, and the method implemented to obtain the most appropriate filters for certain objects is to let the filter-values be trained on some corresponding training set of images. How these two approaches work and how they perform on a small dataset is investigated.

2 Part 1

Section 2.1

The idea behind the BoW approach, is to predefine a collection of "words" (visual vocabulary), so that each word represents a certain type of characteristic image patch. Therefore, the first step before defining this visual vocabulary is to find out which characteristic image patches exist in the current image collections. Descriptors extracted from the images are used to build up the visual vocabulary. They reserve the information of pixel intensities, color, texture and oriented gradients of the specific image patches. Multiple methods are experimented to find and describe those characteristic image patches:

1. key point (SIFT): SIFT key point detector implements lowe's algorithm at first to detect key points in an image. Then it divide the image patch in to 4×4 sub-patches and compute the gradient orientation for all pixels. Key point SIFT only works for gray scale images.
2. dense sampling (DSIFT): Dense SIFT differs from the key point SIFT method in the way interesting points are extracted from the image. This method uses a dense sampling to define what the key point locations are, after which the same procedure of descriptor calculation follows. Compared to key point SIFT, DSIFT is much faster and more accurate for categorization tasks.
3. RGB SIFT: The RGB SIFT descriptor run the SIFT descriptor in each color channel at first. Then it incorporates descriptors from three channels as separate descriptor. Compared to the gray scale, this method returns three times larger dimension of the original matrix, as it gathers information from three channels.
4. rgb SIFT: The rgb SIFT descriptor functions similarly as the RGB SIFT. The only difference lies in values in each color channel. Where values in each channel is normalized by the sum of three channels at first.
5. opponent SIFT: The opponent SIFT convert color values in each channel into values in the color space. Then it follows the same step as RGB SIFT method.

Submitted to 31st Conference on Neural Information Processing Systems (NIPS 2017). Do not distribute.

Because we want to be able to detect images of objects from certain categories (airplanes/cars/faces/motorbikes), we particularly want to find image patches characteristic for those type of objects. Therefore, the locating and describing is done for the training set which consists of images of those four classes. Subsequently, we build up the descriptor sets based on different descriptors. For the key point SIFT, the default setting of parameters are implemented. For gray DSIFT, RGB SIFT, rgb SIFT and opponent SIFT, the bin size and key point multiplier to 3 and 8 respectively. The initial number of images from each category is set to 100. These numbers vary in different experiment settings.

Section 2.2

After finding a whole collection of characteristic image patches from all images in the training set, the next step is to build the "visual vocabulary". The "visual vocabulary" should come with the size of 128 $\times$ number of words (gray scale) or 384 $\times$ number of words (RGB, rgb and opponent color space). It is expected that in general all descriptors should be different from another. However, similar image patches are expected to have similar types of descriptors, meaning in euclidean distance the descriptor-vectors should be close to each other. To define the visual vocabulary, we perform K-means clustering on the set of all image patches found. This gives us K clusters (vectors), and the meaning of one of those vectors can be thought of as a descriptor representative of some type of characteristic image patch. Note that because of K-means clustering, the K clusters are in general different to all the descriptors calculated from the training set. In our initial experiment setting, the number of cluster is set to 400, then we gradually increase the vocabulary size to observe its influence on the final prediction accuracy.

Section 2.3

K-means clustering results in the assignment of images patches and centers of each clusters. Subsequently, features from each image should be extracted and assigned to the nearest center. Given a visual vocabulary, it is now possible to represent any image as a collection of "visual words" from the visual vocabulary as follows:

1. Descriptors for each image are found by various SIFT descriptors.

2. Each found descriptor, is matched to the closest cluster (visual word) of the visual vocabulary.

3. The result is thus a count of which and the number of visual words are found in the given image.

This whole process of calculating the collection of visual words with their count, can be seen as a feature extraction and quantization process of the given image. Also an obvious discovery is that the collection and count can be seen as just a histogram of the visual words. Below figure displays four examples of histograms of example images1.

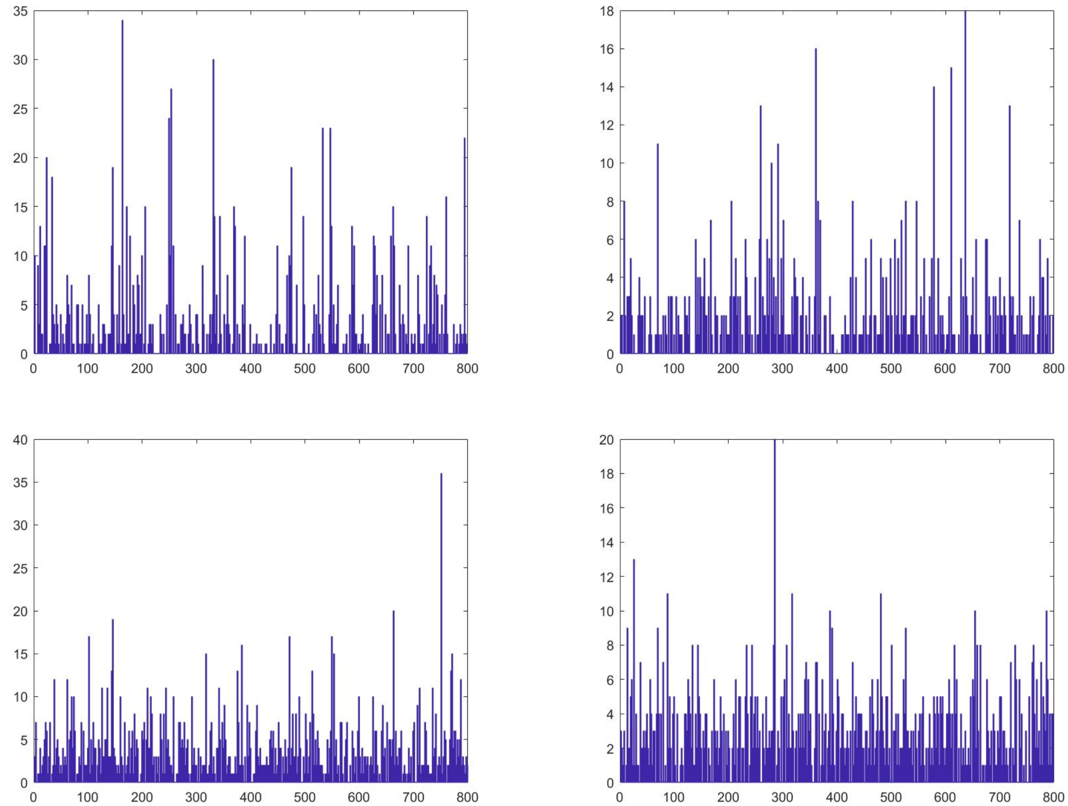

Figure 1: Histograms of example images (800 words) not normalized

Section 2.4

The feature extraction and quantization process given before will heavily depend on image size, since in general larger images is likely to include more descriptors. Therefore a process of normalization is performed. In the end the histogram of each image sums up to 1.

Section 2.5

After the feature quantization, a matrix containing histogram of every image from training set is ready for the SVM classification. The SVM process could be regarded as a binary process for classifying different images. Four SVM classifiers are built for images from these four categories.
The liblinear library is rendered for this training process. For SVM model, the automatic parameters selection feature of liblinear is chosen, it runs four iteration for classifier and saves the parameters to have the best $log2c$ values. Then it attempts to discover the best model parameters at first. Then all best parameters are plugged in the current model and implemented for training each classifier. For each classifier, it could obtain the accuracy rate of more than 90% in the end.
The model will return decision values, which we use to establish a ranking list for evaluation.

Section 2.6

Based on the decision values and real labels of test images, average precision rates are calculated for each classifier and mean average precision rates are also calculated.

2.1 Results

2.1.1 key point SIFT vs. DSIFT

With the same number of "vocabulary words " (400) and number of training image (400), as the statistics shows, DSIFT has a higher mean accuracy rate than key point SIFT (97.98% vs. 89.78%). In addition, within the top ranked images, key point SIFT tend to make more wrong predictions compared with the DSIFT. One of the reasons behind that is DSIFT could take the spacial information into account. In the $HTML$ files, it is clear that the face classifier does the best job. In general, both key point SIFT and gray SIFT descriptors perform the classification smartly (mAP >=90%).

2.1.2 gray SIFT vs. RGBSIFT vs. rgbSIFT vs. opponent SIFT

These four descriptors are based on the basic dense SIFT, general definition for each descriptor is given in section 2.1. The experiment setting is 400 clusters/word vocabulary and 400 training data points for each descriptors.

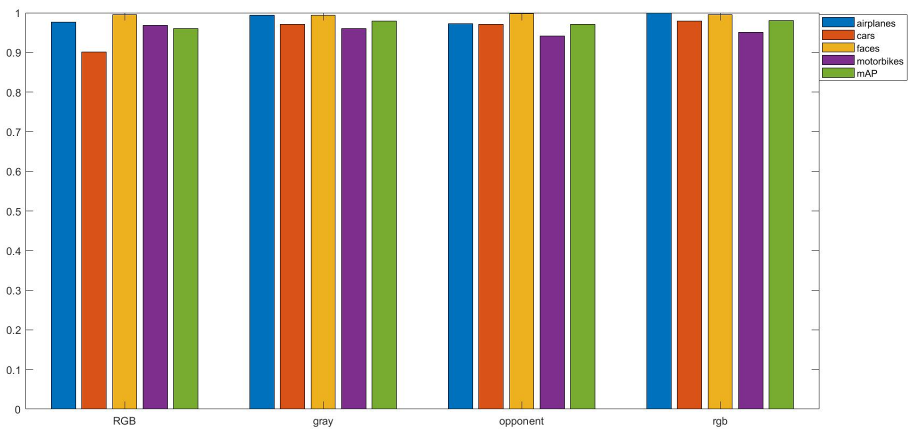

Figure 2: precision rate for different descriptors

As it is demonstrated in the image, all descriptors have a quiet high accuracy rate for images. Relatively speaking, RGB performs weaker than other descriptors. rgb has the highest mAP among all descriptors. In terms of classifiers, airplane and faces classifiers perform quiet robust with varying color descriptors. Detailed statistical information could be found in appendix.

2.1.3 number of clusters

In this experiment setting, we implement similar parameters for bin (8), descriptor (Gray DSIFT), number of training words (400) but different number of clusters. Statistics of models could be summarized in the table below: The above table indicates that as the number of cluster increases,

	airplanes classifier	cars classifier	faces classifier	motorbikes classifier
accuracy rate	99.5%	96.5%	97.5%	96.5%
average precision	99.35%	97.12%	99.37%	96.06%
mAP	97.98%			

Table 1: Evaluation for k-means with cluster number 400

there is a slight improvement in training accuracy rate, average precision and mean Average precision of the DSIFT model. This is reasonable since the number of clusters increases, more features could be implemented for training SVM. Details of images will be clearer and more information is available for the SVM classification.

	airplanes classifier	cars classifier	faces classifier	motorbikes classifier
accuracy rate	97.5%	96.5%	98%	95.5%
average precision	99.46%	97.02%	99.66%	97.22%
mAP	98.34%			

Table 2: Evaluation for k-means with cluster number 800

One limitation of this experiment setting is that as the performance of the gray DSIFT model is already highly satisfactory, it is hard to perceive the effect of changing parameters on the model.

2.1.4 number of training image

The number of training points for SVM is assumed to influence the model's accuracy. For all DSIFT models, it might be difficult to observe the significant change due to increasing number of training data points. Therefore, key point SIFT descriptors are implemented in this experiment. The number of cluster is set to be 400 and all other parameters are set as default.
The training points are 400, 800 to 1200. The effectiveness of corresponding SVM model is evaluated mainly based on the mAP. As it is displayed in the graph below, the mAP hits the peak at 800 and the it slides down. The same trend exists for all four models. This might be due to the over-fitting problem occurs when the training sample is way too large. The whole training process could be visualized as the figure below:

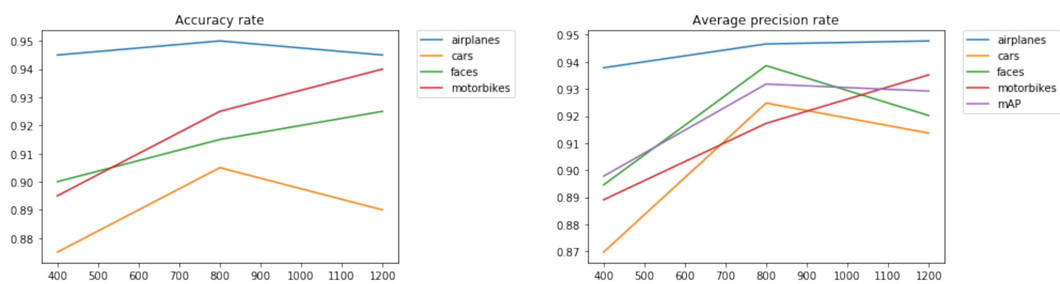

Figure 3: Various training samples for SIFT

2.1.5 SVM kernel

Liblinear implements linear SVM by default, hence, the kernel analysis is not applicable in this case.

2.1.6 Summary

According the above analysis, we could draw several conclusions.

1. In general, DSIFT performs better than SIFT for image classification.
2. rgb SIFT performs the best among all color descriptors.
3. High number of clusters could contribute to higher mAP.
4. As the number of training points increase, the mAP increases at first and then decreases.

3 Part 2

Section 3.1

We are given an already trained convolutional neural network (CNN). To check the specific architecture of the trained CNN, we can get an overview of the CNN with the function vl_simplenn_display. The result is in Figure 4.

```
>> x = load('data/pre_trained_model.mat');
>> vl_simplenn_display(x.net)
     layer|    0|     1|      2|     3|     4|     5|      6|     7|     8|      9|     10|     11|     12|     13|
      type|input|  conv|  mpool|  relu|  conv|  relu|  apool|  conv|  relu|  apool|   conv|   relu|   conv|softmxl|
      name|  n/a|layer1| layer2|layer3|layer4|layer5| layer6|layer7|layer8| layer9|layer10|layer11|layer12|layer13|
----------|-----|------|-------|------|------|------|-------|------|------|-------|-------|-------|-------|-------|
   support|  n/a|     5|      3|     1|     5|     1|      3|     5|     1|      3|      4|      1|      1|      1|
  filt dim|  n/a|     3|    n/a|   n/a|    32|   n/a|    n/a|    32|   n/a|    n/a|     64|    n/a|     64|    n/a|
filt dilat|  n/a|     1|    n/a|   n/a|     1|   n/a|    n/a|     1|   n/a|    n/a|      1|    n/a|      1|    n/a|
 num filts|  n/a|    32|    n/a|   n/a|    32|   n/a|    n/a|    64|   n/a|    n/a|     64|    n/a|     10|    n/a|
    stride|  n/a|     1|      2|     1|     1|     1|      2|     1|     1|      2|      1|      1|      1|      1|
       pad|  n/a|     2|0x1x0x1|     0|     2|     0|0x1x0x1|     2|     0|0x1x0x1|      0|      0|      0|      0|
----------|-----|------|-------|------|------|------|-------|------|------|-------|-------|-------|-------|-------|
   rf size|  n/a|     5|      7|     7|    15|    15|     19|    35|    35|     43|     67|     67|     67|     67|
 rf offset|  n/a|     1|      2|     2|     2|     2|      4|     4|     4|      8|     20|     20|     20|     20|
 rf stride|  n/a|     1|      2|     2|     2|     2|      4|     4|     4|      8|      8|      8|      8|      8|
----------|-----|------|-------|------|------|------|-------|------|------|-------|-------|-------|-------|-------|
 data size|   32|    32|     16|    16|    16|    16|      8|     8|     8|      4|      1|      1|      1|      1|
data depth|    3|    32|     32|    32|    32|    32|     32|    64|    64|     64|     64|     64|     10|      1|
  data num|    1|     1|      1|     1|     1|     1|      1|     1|     1|      1|      1|      1|      1|      1|
----------|-----|------|-------|------|------|------|-------|------|------|-------|-------|-------|-------|-------|
  data mem| 12KB| 128KB|   32KB|  32KB|  32KB|  32KB|    8KB|  16KB|  16KB|    4KB|   256B|   256B|    40B|     4B|
 param mem|  n/a|  10KB|     0B|    0B| 100KB|    0B|     0B| 200KB|    0B|     0B|  256KB|     0B|    3KB|     0B|

parameter memory|569KB (1.5e+05 parameters)|
     data memory|  313KB (for batch size 1)|
```

Figure 4: Architecture of pre-trained CNN.

Question 1

Looking at the architecture of the give trained CNN 'pre_trained_model.mat', we first see that it has 13 layers where each layer is a convolutional layer, relu layer, pooling layer or softmax layer. There are some things and patterns to note. First of all we note that in this CNN we have that the types of layers keep on interchanging: After a convolution layer, we mostly have a relu layer followed by some pooling layer (but not always like this). If we look at layers 1 to 11, we also see a clear pattern: As we move through those layers, we have that the data size (width and height) becomes smaller, but the data depth gets bigger. The memory needed for the data itself also keeps on decreasing after layer 1. At last we also note that the number of parameters (weights) for the convolutional layers becomes more except for the last one.

Question 2

The layer number 10 has the most parameters. We see for layer 10 (convolutional layer): The input is of size WxHxD=4x4x64. For a single filter of this convolution layer the support is 4, meaning the width and height of that 1 filter are both 4. The depth of the filter is given as 64. (Note the input size for this layer is exactly 4x4x64 and the padding is 0, so the filter covers exactly the entire input in this layer). This shows 1 filter will have 4x4x64=1024 parameters (weights). This layer has 64 filters in total, so in total we get 64x1024=65536 parameters for this layer. And because the type of the parameter in here is 'single', which has a size of 4 bytes, we get that the param mem is 65536x4 bytes=262144 bytes =262144/1024 KB=256 KB. This indeed matches the number specified in the bottom of the table.

Section 3.2

Suppose we want to create a new CNN for some task. After designing the architecture, it still needs to be trained on mostly lots of data. Although we do not train a CNN from scratch, at a later step we still do some training. In this step, we prepare some data so that it has some nicer structure. We prepare the data (1865 training images,200 test images) as the IMDB structure specified in the exercise. Note that the training and test images are not necessarily of the same size. Some images of the motorbike are also in gray scale. Since the CNN we are using wants as input 32x32x3 RGB images, we resize each image to 32x32 (using bi-cubic interpolation) and convert gray scale images to RGB by setting the R/G/B values equal as the gray level.

Section 3.3

We want a CNN to classify images only from the four classes "airplanes","cars","faces" and "motorbikes. However, the pre-trained CNN we are given had an architecture where the last softmax layer used 10 classes. Therefore, we change the output size of the last convolutional layer to NEW_OUTPUT_SIZE=4. A filter of the last convolutional layer was of dimensions 1x1x64, and thus we set NEW_INPUT_SIZE=64.

Section 3.4

The pre-trained CNN was trained to classify images for 10 different images, but in the previous step we changed the architecture so that it can be trained to classify four classes of images. To train all weights from scratch takes too long and needs more images. Since the pre-trained CNN was trained for image classification, the weights of especially the first few layers of the pre-trained CNN should still be very useful (because the first few layers will detect patches with more "general" things similar to gabor filters, and therefore not very specific to a certain kind of object). The weights are therefore copied. However, to make the CNN classify and specialize on images of the new four classes, we also train the weights of the new CNN on only images of the four classes (transfer learning/fine tuning).

For the training of the CNN we have some hyper-parameters. We were given default values of the learning rate as:

```
lr_prev_layers = [.2, 2] % lr weights,lr biases
lr_new_layers  = [1, 4]  % lr weights,lr biases
weightDecay = 0.0001
batchSize = 100
numEpochs = 50
```

What we noticed is that if we trained with these values, the objective function went to infinity. This seems to suggest that the learning rate is too high. We thus lowered the learning rate, and in the end found that setting the values as:

```
lr_prev_layers = [0.04,0.1] % lr weights,lr biases
lr_new_layers  = [0.05,0.5] % lr weights,lr biases
```

gave much better results. We keep those values fixed, and "fine tune" (train it on the image set of the four object classes) the pre-trained CNN with different settings for batchSize and numEpochs. We also note that an acceptable learning rate also depends on the value of batchSize (some values we tried for learning rate which worked for batchSize 100 did not work for batchSize 50), and saw that a larger batch size needs a smaller learning rate. The learning rate we used in the end however works for both of them. For all consequent experiments involving the fine-tuned CNN we always have the same learning rates as mentioned above. Since the "most optimal" learning rate depends on the other hyper parameters (i.e. batch size), we do note that different results might have been obtained if we had "optimalized" learning rates for batch size for example. However, since with the used current constant learning rate both runs with different hyper parameters worked quite nicely, we do not expect much difference if we had set different learning rates. The results are visible in Figures 5, 6, and snippets of training/validation errors are given in Appendix C.

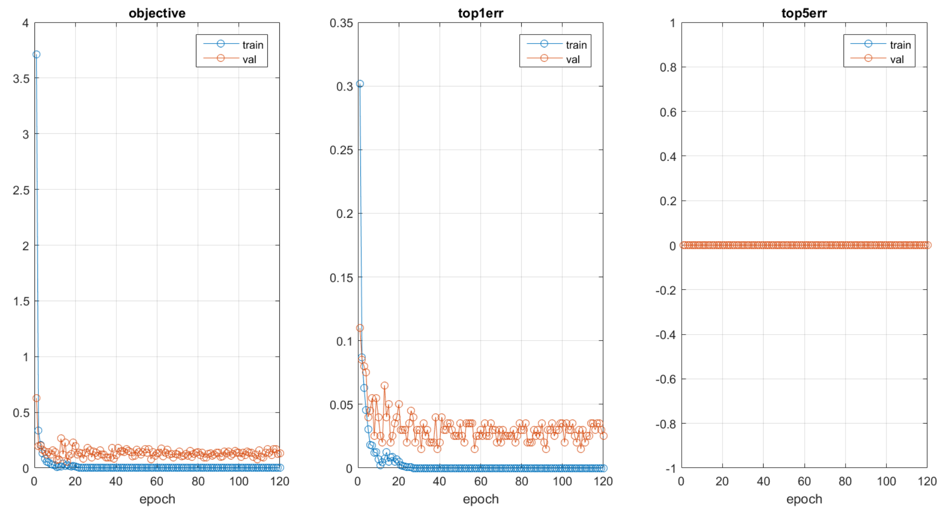

Figure 5: Objective,top1err,top5err when training with batchSize=50 and learning rates lr_prev_layers = [0.04,0.1] lr_new_layers = [0.05,0.5].

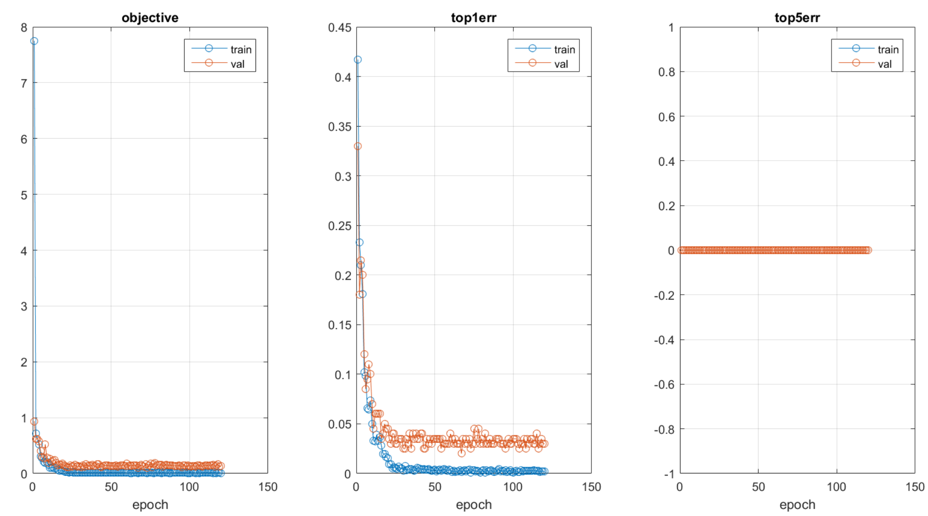

Figure 6: Objective,top1err,top5err when training with batchSize=100 and learning rates lr_prev_layers = [0.04,0.1] lr_new_layers = [0.05,0.5].

For all the six different hyper-parameters, we get that the objective function quickly goes down to 0.001 .

Note that for the "top5err" we always get 0 validation/training error. This is because we only have four different classes, and all those four are of course always in the top 5. The performance of the fine-tuned CNN is discussed in Section 4.2 and can be seen in Table 3.

Section 4.1

The CNN takes a 32x32x3 image as input, and as it goes through the layers the data gets reduces and changes sizes. After the last convolutional layer (just before the softmax) it has been reduced to a 64 dimensional vector. To try to understand how this feature space looks like, we use t-sne to project the 64-dimensional on a 2-dimensional space. There are also some parameters in t-sne. We used the default parameters:

```
no_dims = 2 (project onto 2D space),
initial_dims = 50 (first use PCA to project onto 50-dim space)
perplexity = 30
```

We investigate how t-sne projects the feature space for the pre-trained CNN and the fine-tuned CNN (see Figures 7 and 8).

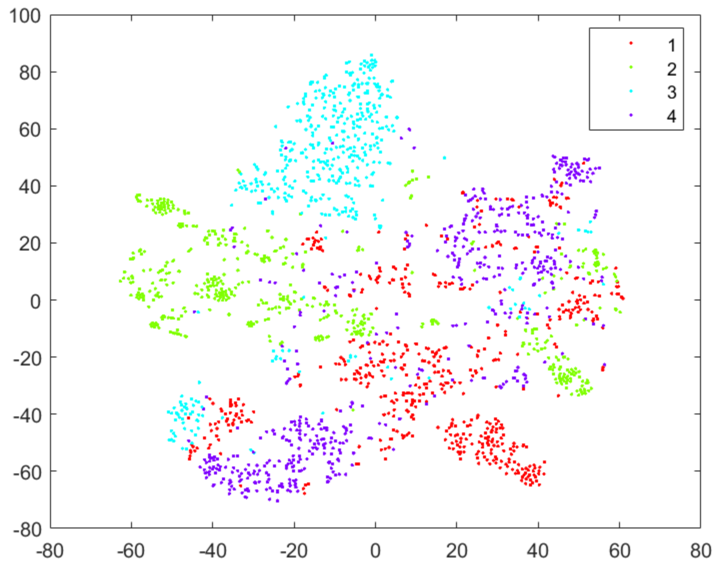

Figure 7: Visualization feature space pre-trained CNN using t-sne.

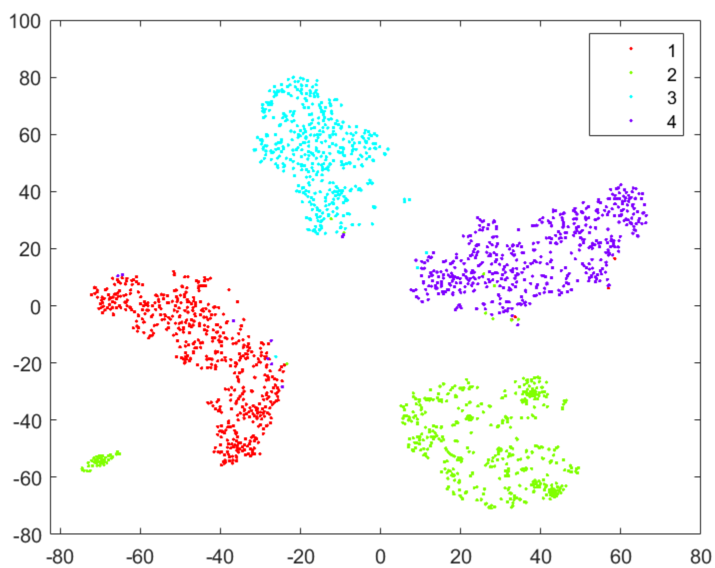

Figure 8: Visualization feature space fine-tuned CNN using t-sne.

For the visualization of the pre-trained CNN, we see that there are multiple "clouds" of projected features of the same class. The biggest cloud is of projections of features corresponding to class 3 (faces). The projections of the features corresponding to the other three classes are less together. When we look at the visualization of the feature space of the fine-tuned CNN, we see a much clearer and better distinction. Here we see four big clouds and a small cloud. Each of the four big cloud consists belongs almost solely to a single class. The small class solely belongs to class 2. So in both visualizations we see that the projections are clustered together. For the pre-trained the spreading is quite large, but in the fine-tuned case the spreading is much less. This also suggests that the SVM for the fine-tuned feature space should give very good results, since in the (very much) reduced dimensional space the features are already quite good separated. Note that even though in Figure 7 the projections are a bit mixed through each other, that does not mean a (very) good classifier can not be gained from those features, since in the original 64 dimensional space the features of each group might be more separated.

Section 4.2

We now compare some object recognition models by calculating the classification accuracy on the test set (consists of 50 images from each class).

We first investigate what effect fine-tuning the pre-trained CNN on images from the four object classes has. For each of the 6 runs of fine-tuning the CNN (with training hyper parameters as in section 3.4), we create a SVM for the resulting fine-tuned CNN respectively. The training set used consists of 500,465,400,500 images of airplanes,cars,faces,motorbikes respectively. Each image goes through the fine-tuned CNN, and the output of the last convolutional layer is the feature-vector of the image we use to train the SVM. To evaluate the performance, the classification accuracy on the test set is calculated for each of the 6 SVMs. The results are visible in Table 3. The classification error of using the (complete) fine-tuned CNN as a classifier, and a SVM trained using features from the pre-trained CNN is also shown in the table. First we note that all classification accuracies are very

FT CNN training hyper parameters	FT CNN	SVM (using features PT CNN)	SVM (using features FT CNN)
1. BS = 50, E = 40	98.00	94.50	98.00
2. BS = 100, E = 40	96.50	94.50	95.50
3. BS = 50, E = 80	98.00	94.50	98.00
4. BS = 100, E = 80	96.50	94.50	95.50
5. BS = 50, E = 120	97.50	94.50	98.00
6. BS = 100, E = 120	96.50	94.50	95.50

Table 3: Classification accuracy on test set. The learning rate is set equal for all cases as described in section 3.3. (BS=batch size, E=num epochs, PT=pre-trained, FT=fine-tuned)

high ($\geq$ 90 %). As seen from the discussion and visualization of the (projected) features (Figures 7,8), the high classification accuracy is indeed like expected. It also suggests that the since the pre-trained CNN used images not specifically from our 4 object classes, the features were still useful enough to mostly distinguish between airplanes/cars/faces/motorbikes. Furthermore, the table shows that the accuracy of the SVM when using features from the fine-tuned CNN is slightly better than when it is trained on features from the pre-trained CNN.

The fine-tuned CNN by itself can also be used as a classifier. In Table 3 we see that the best fine-tuned CNN already gave very good results on the test set (98 % classification accuracy). This is comparable with many of the binary classifiers based on BoW model (see Appendix A,B for all accuracy rates). Note however that the fine-tuned did use a larger training set then the BoW-based classifiers, but much more importantly the fine-tuned CNN is based on the pre-trained CNN. And the training of the pre-trained CNN takes a very long time and needs to have a much larger data set. This shows that if there is no pre-trained CNN available, a classifier using the BoW approach is definitely quicker to create and gives roughly same performance as a fine-tuned CNN or SVM using features from fine-tuned CNN. At last we note that the performance is based on classifying images for 4 object classes. Bigger differences in performance might be detected when we consider a larger number of classes.

4 Conclusion

Both the BoW and CNN approach return excellent results in the experiments concerning classifying objects from exactly 4 object classes. The BoW approach works with limited data sets and time. Using the CNN approach without a pre-trained CNN available, will in comparison take much longer and require more data

With BoW approach, different bin size, descriptors, vocabulary size and training data points are experimented and compared. Summary of the experiment process is explained the summary section 2.1.6. The best mAP (98.1 %) was achieved when using rgb-SIFT descriptors with vocabulary size of 400 and 400 training points. In order to build up the optimal BoW, it is vital to incorporate different parameters and descriptors.
The classification systems based on a CNN presents high predicting accuracy rate. Fine-tuning the CNN to images of the object classes gives even better performance, and in the end we managed to get a classification accuracy of 98% for the fine-tuned CNN as well as a SVM based on features from the fine-tuned CNN. Visualizing the features after the final convolution layer of the fine-tuned shows features from different object classes are quite distinctive from each other. This shows that the filters from the CNN indeed seem to manage to be trained so that they extract appropriate useful image structures from images.

Future research: Both the BoW and CNN based classification systems were for only four different object classes. How the performance for each approach changes when considering multiple classes is interesting for further research.
Furthermore, the four categories considered for training the BoW and CNN are distinctive in nature. They are quiet heterogenous and share limited common features, which lead to compelling predicting accuracy. Therefore, categories with similar features could be explored by future research, as well as comparing the performance of BoW and CNN on classifying these categories.

5 Appendix

A color descriptors

	airplanes classifier	cars classifier	faces classifier	motorbikes classifier
accuracy rate	96%	93%	98%	95%
average precision	97.60%	90.10%	99.50%	96.80%
mAP	96.00%			

Table 4: Evaluation for RGBSIFT

	airplanes classifier	cars classifier	faces classifier	motorbikes classifier
accuracy rate	97.5%	96.5%	98%	95.5%
average precision	99.36%	97.12%	99.37%	96.06%
mAP	97.98%			

Table 5: Evaluation for graySIFT

	airplanes classifier	cars classifier	faces classifier	motorbikes classifier
accuracy rate	98.5%	94.5%	97%	96%
average precision	99.9%	97.92%	99.55%	95.15%
mAP	98.10%			

Table 6: Evaluation for rgbSIFT

	airplanes classifier	cars classifier	faces classifier	motorbikes classifier
accuracy rate	95.5%	96.5%	98.5%	96%
average precision	97.22%	97.08%	99.78%	94.2%
mAP	97.07%			

Table 7: Evaluation for opponent SIFT

B training points

	airplanes classifier	cars classifier	faces classifier	motorbikes classifier
accuracy rate	94.5%	87.5%	90%	89.5%
average precision	93.78%	86.98%	89.46%	88.91%
mAP	89.78%			

Table 8: key point SIFT 400 data points

	airplanes classifier	cars classifier	faces classifier	motorbikes classifier
accuracy rate	95%	90.5%	91.5%	92.5%
average precision	94.66%	92.48%	93.86%	91.73%
mAP	93.18%			

Table 9: key point SIFT 800 data points

	airplanes classifier	cars classifier	faces classifier	motorbikes classifier
accuracy rate	94.5%	89%	92.5%	94%
average precision	94.77%	91.37%	92.02%	93.52%
mAP	92.92%			

Table 10: key point SIFT 1200 data points

C Training/validation errors

Snippets fine-tuning CNN with batch size of 50 and learning rates lr_prev_layers = [0.04,0.1] (lr weights,lr biases), lr_new_layers = [0.05,0.5] (lr weights,lr biases).

After 40 epochs:

```
train: epoch 40:   1/ 38: 204.5 (204.5) Hz objective: 0.001 top1err: 0.000 top5err: 0.000
train: epoch 40:   2/ 38: 210.5 (216.9) Hz objective: 0.002 top1err: 0.000 top5err: 0.000
train: epoch 40:   3/ 38: 207.7 (202.4) Hz objective: 0.001 top1err: 0.000 top5err: 0.000
train: epoch 40:   4/ 38: 205.3 (198.4) Hz objective: 0.001 top1err: 0.000 top5err: 0.000
train: epoch 40:   5/ 38: 202.8 (222.8) Hz objective: 0.001 top1err: 0.000 top5err: 0.000
train: epoch 40:   6/ 38: 204.7 (214.4) Hz objective: 0.001 top1err: 0.000 top5err: 0.000
train: epoch 40:   7/ 38: 206.8 (220.4) Hz objective: 0.001 top1err: 0.000 top5err: 0.000
train: epoch 40:   8/ 38: 208.2 (218.3) Hz objective: 0.001 top1err: 0.000 top5err: 0.000
train: epoch 40:   9/ 38: 209.6 (221.5) Hz objective: 0.001 top1err: 0.000 top5err: 0.000
train: epoch 40:  10/ 38: 210.9 (223.6) Hz objective: 0.001 top1err: 0.000 top5err: 0.000
train: epoch 40:  11/ 38: 211.5 (218.3) Hz objective: 0.001 top1err: 0.000 top5err: 0.000
train: epoch 40:  12/ 38: 212.6 (224.4) Hz objective: 0.001 top1err: 0.000 top5err: 0.000
train: epoch 40:  13/ 38: 212.8 (216.5) Hz objective: 0.001 top1err: 0.000 top5err: 0.000
train: epoch 40:  14/ 38: 213.4 (221.4) Hz objective: 0.001 top1err: 0.000 top5err: 0.000
train: epoch 40:  15/ 38: 213.4 (212.4) Hz objective: 0.001 top1err: 0.000 top5err: 0.000
train: epoch 40:  16/ 38: 214.0 (223.4) Hz objective: 0.001 top1err: 0.000 top5err: 0.000
train: epoch 40:  17/ 38: 214.5 (223.6) Hz objective: 0.001 top1err: 0.000 top5err: 0.000
train: epoch 40:  18/ 38: 214.8 (220.6) Hz objective: 0.001 top1err: 0.000 top5err: 0.000
train: epoch 40:  19/ 38: 214.9 (215.4) Hz objective: 0.001 top1err: 0.000 top5err: 0.000
train: epoch 40:  20/ 38: 215.3 (224.3) Hz objective: 0.001 top1err: 0.000 top5err: 0.000
train: epoch 40:  21/ 38: 215.7 (224.0) Hz objective: 0.001 top1err: 0.000 top5err: 0.000
train: epoch 40:  22/ 38: 215.7 (214.4) Hz objective: 0.001 top1err: 0.000 top5err: 0.000
train: epoch 40:  23/ 38: 215.9 (220.8) Hz objective: 0.001 top1err: 0.000 top5err: 0.000
train: epoch 40:  24/ 38: 215.3 (203.2) Hz objective: 0.001 top1err: 0.000 top5err: 0.000
train: epoch 40:  25/ 38: 215.3 (215.3) Hz objective: 0.001 top1err: 0.000 top5err: 0.000
train: epoch 40:  26/ 38: 215.2 (211.7) Hz objective: 0.001 top1err: 0.000 top5err: 0.000
train: epoch 40:  27/ 38: 215.0 (209.4) Hz objective: 0.001 top1err: 0.000 top5err: 0.000
train: epoch 40:  28/ 38: 215.0 (216.6) Hz objective: 0.001 top1err: 0.000 top5err: 0.000
train: epoch 40:  29/ 38: 215.2 (219.9) Hz objective: 0.001 top1err: 0.000 top5err: 0.000
train: epoch 40:  30/ 38: 215.6 (227.2) Hz objective: 0.001 top1err: 0.000 top5err: 0.000
train: epoch 40:  31/ 38: 215.6 (217.4) Hz objective: 0.001 top1err: 0.000 top5err: 0.000
train: epoch 40:  32/ 38: 215.9 (223.5) Hz objective: 0.002 top1err: 0.000 top5err: 0.000
train: epoch 40:  33/ 38: 216.1 (222.8) Hz objective: 0.002 top1err: 0.000 top5err: 0.000
train: epoch 40:  34/ 38: 216.3 (223.2) Hz objective: 0.002 top1err: 0.000 top5err: 0.000
train: epoch 40:  35/ 38: 216.1 (211.5) Hz objective: 0.002 top1err: 0.000 top5err: 0.000
train: epoch 40:  36/ 38: 216.3 (222.3) Hz objective: 0.002 top1err: 0.000 top5err: 0.000
train: epoch 40:  37/ 38: 216.4 (220.5) Hz objective: 0.002 top1err: 0.000 top5err: 0.000
train: epoch 40:  38/ 38: 216.5 (226.9) Hz objective: 0.002 top1err: 0.000 top5err: 0.000
val: epoch 40:   1/  4: 575.0 (575.0) Hz objective: 0.009 top1err: 0.000 top5err: 0.000
val: epoch 40:   2/  4: 568.7 (562.5) Hz objective: 0.023 top1err: 0.010 top5err: 0.000
val: epoch 40:   3/  4: 570.8 (575.0) Hz objective: 0.024 top1err: 0.007 top5err: 0.000
val: epoch 40:   4/  4: 579.6 (607.9) Hz objective: 0.118 top1err: 0.020 top5err: 0.000
```

After 80 epochs:

```
train: epoch 80:   1/ 38: 211.2 (211.2) Hz objective: 0.000 top1err: 0.000 top5err: 0.000
train: epoch 80:   2/ 38: 211.2 (211.2) Hz objective: 0.002 top1err: 0.000 top5err: 0.000
train: epoch 80:   3/ 38: 209.9 (207.2) Hz objective: 0.001 top1err: 0.000 top5err: 0.000
train: epoch 80:   4/ 38: 213.9 (226.9) Hz objective: 0.003 top1err: 0.000 top5err: 0.000
train: epoch 80:   5/ 38: 222.9 (208.4) Hz objective: 0.004 top1err: 0.000 top5err: 0.000
train: epoch 80:   6/ 38: 209.0 (159.5) Hz objective: 0.003 top1err: 0.000 top5err: 0.000
train: epoch 80:   7/ 38: 210.1 (216.9) Hz objective: 0.003 top1err: 0.000 top5err: 0.000
train: epoch 80:   8/ 38: 212.1 (226.5) Hz objective: 0.002 top1err: 0.000 top5err: 0.000
```

```
train: epoch 80:   9/ 38: 204.5 (159.0) Hz objective: 0.002 top1err: 0.000 top5err: 0.000
train: epoch 80:  10/ 38: 205.0 (210.1) Hz objective: 0.002 top1err: 0.000 top5err: 0.000
train: epoch 80:  11/ 38: 198.2 (148.8) Hz objective: 0.002 top1err: 0.000 top5err: 0.000
train: epoch 80:  12/ 38: 199.6 (215.7) Hz objective: 0.002 top1err: 0.000 top5err: 0.000
train: epoch 80:  13/ 38: 201.2 (223.6) Hz objective: 0.002 top1err: 0.000 top5err: 0.000
train: epoch 80:  14/ 38: 201.6 (207.3) Hz objective: 0.002 top1err: 0.000 top5err: 0.000
train: epoch 80:  15/ 38: 202.7 (218.1) Hz objective: 0.002 top1err: 0.000 top5err: 0.000
train: epoch 80:  16/ 38: 203.5 (217.1) Hz objective: 0.002 top1err: 0.000 top5err: 0.000
train: epoch 80:  17/ 38: 205.1 (233.8) Hz objective: 0.002 top1err: 0.000 top5err: 0.000
train: epoch 80:  18/ 38: 205.6 (214.0) Hz objective: 0.002 top1err: 0.000 top5err: 0.000
train: epoch 80:  19/ 38: 206.5 (224.3) Hz objective: 0.002 top1err: 0.000 top5err: 0.000
train: epoch 80:  20/ 38: 207.4 (227.8) Hz objective: 0.002 top1err: 0.000 top5err: 0.000
train: epoch 80:  21/ 38: 208.2 (223.6) Hz objective: 0.001 top1err: 0.000 top5err: 0.000
train: epoch 80:  22/ 38: 209.4 (238.6) Hz objective: 0.001 top1err: 0.000 top5err: 0.000
train: epoch 80:  23/ 38: 206.0 (152.3) Hz objective: 0.001 top1err: 0.000 top5err: 0.000
train: epoch 80:  24/ 38: 207.0 (231.3) Hz objective: 0.001 top1err: 0.000 top5err: 0.000
train: epoch 80:  25/ 38: 206.7 (200.2) Hz objective: 0.001 top1err: 0.000 top5err: 0.000
train: epoch 80:  26/ 38: 207.4 (228.7) Hz objective: 0.001 top1err: 0.000 top5err: 0.000
train: epoch 80:  27/ 38: 208.1 (227.3) Hz objective: 0.001 top1err: 0.000 top5err: 0.000
train: epoch 80:  28/ 38: 208.9 (231.5) Hz objective: 0.001 top1err: 0.000 top5err: 0.000
train: epoch 80:  29/ 38: 209.3 (221.5) Hz objective: 0.001 top1err: 0.000 top5err: 0.000
train: epoch 80:  30/ 38: 209.7 (224.2) Hz objective: 0.001 top1err: 0.000 top5err: 0.000
train: epoch 80:  31/ 38: 210.5 (237.4) Hz objective: 0.001 top1err: 0.000 top5err: 0.000
train: epoch 80:  32/ 38: 210.9 (224.3) Hz objective: 0.001 top1err: 0.000 top5err: 0.000
train: epoch 80:  33/ 38: 211.3 (222.7) Hz objective: 0.001 top1err: 0.000 top5err: 0.000
train: epoch 80:  34/ 38: 212.0 (237.4) Hz objective: 0.001 top1err: 0.000 top5err: 0.000
train: epoch 80:  35/ 38: 212.2 (221.3) Hz objective: 0.001 top1err: 0.000 top5err: 0.000
train: epoch 80:  36/ 38: 212.5 (221.1) Hz objective: 0.001 top1err: 0.000 top5err: 0.000
train: epoch 80:  37/ 38: 212.8 (227.4) Hz objective: 0.001 top1err: 0.000 top5err: 0.000
train: epoch 80:  38/ 38: 212.9 (220.9) Hz objective: 0.001 top1err: 0.000 top5err: 0.000
val: epoch 80:   1/  4: 648.5 (648.5) Hz objective: 0.026 top1err: 0.020 top5err: 0.000
val: epoch 80:   2/  4: 624.0 (601.3) Hz objective: 0.198 top1err: 0.030 top5err: 0.000
val: epoch 80:   3/  4: 607.0 (575.7) Hz objective: 0.165 top1err: 0.033 top5err: 0.000
val: epoch 80:   4/  4: 598.3 (573.6) Hz objective: 0.144 top1err: 0.030 top5err: 0.000
```

After 120 epochs:

```
train: epoch 120:   1/ 38: 201.4 (201.4) Hz objective: 0.000 top1err: 0.000 top5err: 0.000
train: epoch 120:   2/ 38: 199.6 (197.8) Hz objective: 0.005 top1err: 0.000 top5err: 0.000
train: epoch 120:   3/ 38: 197.9 (194.6) Hz objective: 0.004 top1err: 0.000 top5err: 0.000
train: epoch 120:   4/ 38: 197.0 (194.6) Hz objective: 0.003 top1err: 0.000 top5err: 0.000
train: epoch 120:   5/ 38: 197.0 (207.2) Hz objective: 0.002 top1err: 0.000 top5err: 0.000
train: epoch 120:   6/ 38: 198.7 (207.9) Hz objective: 0.002 top1err: 0.000 top5err: 0.000
train: epoch 120:   7/ 38: 198.5 (197.0) Hz objective: 0.002 top1err: 0.000 top5err: 0.000
train: epoch 120:   8/ 38: 199.5 (207.3) Hz objective: 0.002 top1err: 0.000 top5err: 0.000
train: epoch 120:   9/ 38: 200.7 (210.5) Hz objective: 0.002 top1err: 0.000 top5err: 0.000
train: epoch 120:  10/ 38: 201.2 (205.5) Hz objective: 0.002 top1err: 0.000 top5err: 0.000
train: epoch 120:  11/ 38: 196.7 (161.3) Hz objective: 0.002 top1err: 0.000 top5err: 0.000
train: epoch 120:  12/ 38: 189.3 (133.9) Hz objective: 0.002 top1err: 0.000 top5err: 0.000
train: epoch 120:  13/ 38: 188.3 (177.1) Hz objective: 0.002 top1err: 0.000 top5err: 0.000
train: epoch 120:  14/ 38: 189.9 (213.0) Hz objective: 0.002 top1err: 0.000 top5err: 0.000
train: epoch 120:  15/ 38: 191.4 (215.6) Hz objective: 0.001 top1err: 0.000 top5err: 0.000
train: epoch 120:  16/ 38: 192.1 (203.8) Hz objective: 0.001 top1err: 0.000 top5err: 0.000
train: epoch 120:  17/ 38: 192.9 (206.3) Hz objective: 0.001 top1err: 0.000 top5err: 0.000
train: epoch 120:  18/ 38: 193.6 (206.2) Hz objective: 0.001 top1err: 0.000 top5err: 0.000
train: epoch 120:  19/ 38: 194.3 (208.3) Hz objective: 0.001 top1err: 0.000 top5err: 0.000
train: epoch 120:  20/ 38: 195.2 (212.9) Hz objective: 0.001 top1err: 0.000 top5err: 0.000
train: epoch 120:  21/ 38: 196.1 (216.7) Hz objective: 0.001 top1err: 0.000 top5err: 0.000
train: epoch 120:  22/ 38: 196.4 (202.5) Hz objective: 0.001 top1err: 0.000 top5err: 0.000
train: epoch 120:  23/ 38: 197.2 (217.6) Hz objective: 0.001 top1err: 0.000 top5err: 0.000
```

```
train: epoch 120:  24/ 38: 197.1 (195.2) Hz objective: 0.001 top1err: 0.000 top5err: 0.000
train: epoch 120:  25/ 38: 197.6 (208.2) Hz objective: 0.001 top1err: 0.000 top5err: 0.000
train: epoch 120:  26/ 38: 198.3 (217.1) Hz objective: 0.001 top1err: 0.000 top5err: 0.000
train: epoch 120:  27/ 38: 198.8 (213.1) Hz objective: 0.001 top1err: 0.000 top5err: 0.000
train: epoch 120:  28/ 38: 199.3 (213.5) Hz objective: 0.001 top1err: 0.000 top5err: 0.000
train: epoch 120:  29/ 38: 199.7 (213.9) Hz objective: 0.001 top1err: 0.000 top5err: 0.000
train: epoch 120:  30/ 38: 200.4 (222.3) Hz objective: 0.001 top1err: 0.000 top5err: 0.000
train: epoch 120:  31/ 38: 200.8 (212.9) Hz objective: 0.001 top1err: 0.000 top5err: 0.000
train: epoch 120:  32/ 38: 201.5 (224.6) Hz objective: 0.001 top1err: 0.000 top5err: 0.000
train: epoch 120:  33/ 38: 202.0 (222.0) Hz objective: 0.001 top1err: 0.000 top5err: 0.000
train: epoch 120:  34/ 38: 197.7 (116.4) Hz objective: 0.001 top1err: 0.000 top5err: 0.000
train: epoch 120:  35/ 38: 197.5 (189.5) Hz objective: 0.001 top1err: 0.000 top5err: 0.000
train: epoch 120:  36/ 38: 197.7 (206.1) Hz objective: 0.001 top1err: 0.000 top5err: 0.000
train: epoch 120:  37/ 38: 198.1 (212.9) Hz objective: 0.001 top1err: 0.000 top5err: 0.000
train: epoch 120:  38/ 38: 198.1 (194.5) Hz objective: 0.001 top1err: 0.000 top5err: 0.000
val: epoch 120:   1/  4: 521.3 (521.3) Hz objective: 0.396 top1err: 0.060 top5err: 0.000
val: epoch 120:   2/  4: 550.0 (582.0) Hz objective: 0.250 top1err: 0.040 top5err: 0.000
val: epoch 120:   3/  4: 549.9 (549.7) Hz objective: 0.175 top1err: 0.033 top5err: 0.000
val: epoch 120:   4/  4: 536.2 (499.0) Hz objective: 0.132 top1err: 0.025 top5err: 0.000
```

Snippets fine-tuning CNN with batch size of 100 and learning rates lr_prev_layers = [0.04,0.1] (lr weights,lr biases), lr_new_layers = [0.05,0.5] (lr weights,lr biases).

After 40 epochs:

```
train: epoch 40:   1/ 19: 207.1 (207.1) Hz objective: 0.032 top1err: 0.020 top5err: 0.000
train: epoch 40:   2/ 19: 203.9 (200.7) Hz objective: 0.022 top1err: 0.010 top5err: 0.000
train: epoch 40:   3/ 19: 203.2 (202.0) Hz objective: 0.016 top1err: 0.007 top5err: 0.000
train: epoch 40:   4/ 19: 204.1 (206.7) Hz objective: 0.018 top1err: 0.007 top5err: 0.000
train: epoch 40:   5/ 19: 200.0 (177.0) Hz objective: 0.016 top1err: 0.006 top5err: 0.000
train: epoch 40:   6/ 19: 201.8 (211.1) Hz objective: 0.013 top1err: 0.005 top5err: 0.000
train: epoch 40:   7/ 19: 202.6 (208.2) Hz objective: 0.014 top1err: 0.006 top5err: 0.000
train: epoch 40:   8/ 19: 201.3 (192.2) Hz objective: 0.015 top1err: 0.007 top5err: 0.000
train: epoch 40:   9/ 19: 201.7 (205.6) Hz objective: 0.014 top1err: 0.007 top5err: 0.000
train: epoch 40:  10/ 19: 199.5 (181.4) Hz objective: 0.013 top1err: 0.006 top5err: 0.000
train: epoch 40:  11/ 19: 198.6 (189.6) Hz objective: 0.014 top1err: 0.006 top5err: 0.000
train: epoch 40:  12/ 19: 199.7 (213.0) Hz objective: 0.014 top1err: 0.006 top5err: 0.000
train: epoch 40:  13/ 19: 201.1 (219.2) Hz objective: 0.014 top1err: 0.006 top5err: 0.000
train: epoch 40:  14/ 19: 201.8 (212.3) Hz objective: 0.014 top1err: 0.006 top5err: 0.000
train: epoch 40:  15/ 19: 201.6 (198.7) Hz objective: 0.014 top1err: 0.005 top5err: 0.000
train: epoch 40:  16/ 19: 202.6 (219.0) Hz objective: 0.013 top1err: 0.005 top5err: 0.000
train: epoch 40:  17/ 19: 202.2 (194.9) Hz objective: 0.014 top1err: 0.005 top5err: 0.000
train: epoch 40:  18/ 19: 202.7 (213.3) Hz objective: 0.013 top1err: 0.005 top5err: 0.000
train: epoch 40:  19/ 19: 200.5 (152.9) Hz objective: 0.014 top1err: 0.005 top5err: 0.000
val: epoch 40:   1/  2: 551.4 (551.4) Hz objective: 0.119 top1err: 0.040 top5err: 0.000
val: epoch 40:   2/  2: 540.4 (529.8) Hz objective: 0.136 top1err: 0.035 top5err: 0.000
```

After 80 epochs:

```
train: epoch 80:   1/ 19: 202.7 (202.7) Hz  objective: 0.007 top1err: 0.000 top5err: 0.000
train: epoch 80:   2/ 19: 183.4 (167.5) Hz objective: 0.026 top1err: 0.005 top5err: 0.000
train: epoch 80:   3/ 19: 176.2 (163.4) Hz objective: 0.019 top1err: 0.003 top5err: 0.000
train: epoch 80:   4/ 19: 178.8 (186.9) Hz objective: 0.026 top1err: 0.007 top5err: 0.000
train: epoch 80:   5/ 19: 188.8 (196.7) Hz objective: 0.022 top1err: 0.006 top5err: 0.000
train: epoch 80:   6/ 19: 182.8 (157.7) Hz objective: 0.021 top1err: 0.005 top5err: 0.000
train: epoch 80:   7/ 19: 176.7 (147.1) Hz objective: 0.020 top1err: 0.006 top5err: 0.000
train: epoch 80:   8/ 19: 178.9 (196.4) Hz objective: 0.019 top1err: 0.005 top5err: 0.000
train: epoch 80:   9/ 19: 175.5 (152.7) Hz objective: 0.018 top1err: 0.006 top5err: 0.000
train: epoch 80:  10/ 19: 174.2 (163.3) Hz objective: 0.018 top1err: 0.005 top5err: 0.000
train: epoch 80:  11/ 19: 173.9 (171.0) Hz objective: 0.017 top1err: 0.005 top5err: 0.000
train: epoch 80:  12/ 19: 170.8 (142.8) Hz objective: 0.016 top1err: 0.004 top5err: 0.000
```

```
train: epoch 80:  13/ 19: 165.5 (120.2) Hz objective: 0.015 top1err: 0.004 top5err: 0.000
train: epoch 80:  14/ 19: 157.5 (97.0) Hz objective: 0.015 top1err: 0.004 top5err: 0.000
train: epoch 80:  15/ 19: 153.9 (116.3) Hz objective: 0.014 top1err: 0.003 top5err: 0.000
train: epoch 80:  16/ 19: 151.1 (118.8) Hz objective: 0.014 top1err: 0.003 top5err: 0.000
train: epoch 80:  17/ 19: 153.6 (208.3) Hz objective: 0.013 top1err: 0.003 top5err: 0.000
train: epoch 80:  18/ 19: 154.3 (167.1) Hz objective: 0.013 top1err: 0.003 top5err: 0.000
train: epoch 80:  19/ 19: 156.0 (226.2) Hz objective: 0.013 top1err: 0.003 top5err: 0.000
val: epoch 80:   1/  2: 543.4 (543.4) Hz objective: 0.188 top1err: 0.040 top5err: 0.000
val: epoch 80:   2/  2: 545.5 (547.6) Hz objective: 0.160 top1err: 0.035 top5err: 0.000
```

After 120 epochs:

```
train: epoch 120:   1/ 19: 193.7 (193.7) Hz objective: 0.022 top1err: 0.000 top5err: 0.000
train: epoch 120:   2/ 19: 193.3 (192.9) Hz objective: 0.017 top1err: 0.000 top5err: 0.000
train: epoch 120:   3/ 19: 191.9 (189.1) Hz objective: 0.013 top1err: 0.000 top5err: 0.000
train: epoch 120:   4/ 19: 183.9 (163.5) Hz objective: 0.011 top1err: 0.000 top5err: 0.000
train: epoch 120:   5/ 19: 169.1 (196.1) Hz objective: 0.010 top1err: 0.000 top5err: 0.000
train: epoch 120:   6/ 19: 173.1 (196.6) Hz objective: 0.010 top1err: 0.000 top5err: 0.000
train: epoch 120:   7/ 19: 176.5 (199.9) Hz objective: 0.010 top1err: 0.000 top5err: 0.000
train: epoch 120:   8/ 19: 177.6 (185.9) Hz objective: 0.010 top1err: 0.000 top5err: 0.000
train: epoch 120:   9/ 19: 179.8 (198.7) Hz objective: 0.009 top1err: 0.000 top5err: 0.000
train: epoch 120:  10/ 19: 179.1 (173.6) Hz objective: 0.009 top1err: 0.000 top5err: 0.000
train: epoch 120:  11/ 19: 180.9 (201.4) Hz objective: 0.011 top1err: 0.001 top5err: 0.000
train: epoch 120:  12/ 19: 181.9 (192.5) Hz objective: 0.010 top1err: 0.001 top5err: 0.000
train: epoch 120:  13/ 19: 182.9 (196.6) Hz objective: 0.010 top1err: 0.001 top5err: 0.000
train: epoch 120:  14/ 19: 183.9 (197.6) Hz objective: 0.010 top1err: 0.001 top5err: 0.000
train: epoch 120:  15/ 19: 184.0 (185.6) Hz objective: 0.011 top1err: 0.002 top5err: 0.000
train: epoch 120:  16/ 19: 184.5 (192.3) Hz objective: 0.010 top1err: 0.002 top5err: 0.000
train: epoch 120:  17/ 19: 185.2 (197.0) Hz objective: 0.010 top1err: 0.002 top5err: 0.000
train: epoch 120:  18/ 19: 185.8 (197.7) Hz objective: 0.010 top1err: 0.002 top5err: 0.000
train: epoch 120:  19/ 19: 186.1 (192.2) Hz objective: 0.011 top1err: 0.002 top5err: 0.000
val: epoch 120:   1/  2: 496.8 (496.8) Hz objective: 0.234 top1err: 0.050 top5err: 0.000
val: epoch 120:   2/  2: 510.0 (524.0) Hz objective: 0.137 top1err: 0.030 top5err: 0.000
```