UNIVERSITY OF AMSTERDAM

Final Assignment Report

Luca Simonetto - 11413522
Heng Lin - 11392533
Computer Vision 1

April 5, 2017

Bag-of-Words based Image Classification

This section describes the workflow for the task of image classification using the bag-of-words approach. This method revolves around a number of steps, each one explained in the following subsections.

Descriptor extraction

The first step of the BoW approach to extract descriptors for every image that will be used for the vocabulary creation, in order to transform each one in a set of "words" (each word consisting of a descriptor). The descriptor type to use was initially given, the Scale-Invariant Feature Transform (SIFT). In order to extract SIFT descriptors from every image, the included library VL_FEAT has been used. As different approaches on descriptor extraction exist depending on the color space of choice, multiple alternative experimental setups have been created. The SIFT descriptor can be calculated as:

- **intensity SIFT**: calculated solely on the grayscale version of the input image, it produces a matrix of $n \times 128$ elements, where n is the number of descriptors extracted and *128* is the size of each descriptor.
- **color (RGB) SIFT**: in this case the keypoint locations are first computed on the grayscale image, in order to use them to extract descriptors for each color channel at the same location. As RGB images have three channels, the result is created by concatenating the descriptors for each channel, resulting in a $n \times 384$ matrix.
- **normalized RGB (rgb) SIFT**: the descriptors are calculated in the same way as the RGB SIFT descriptors are, but the color channels are normalized R, G and B.
- **opponent SIFT**: as above, but the descriptors are calculated using the o1, o2 and o3 channels (opponent color channels).
- **dense SIFT variants**: for each of the above SIFT types, the dense version is also computed. Dense descriptor positions are extracted from a grid subdivision of the image, and each descriptor has a fixed size, unlike normal SIFT. In this project only one value for spacing and size has been used, in order to reduce computation time and experiments number. For each image, 1000 descriptors on average are extracted, while normal SIFT produces around 400 descriptors.

- **LIOP**: Along with SIFT descriptors, it has been chosen to also use the Local Intensity Order Pattern (LIOP) descriptor[1]. Differently from SIFT, the LIOP descriptor by default computes only one descriptor per image, that has to be square and with an odd size length. The output for a LIOP descriptor extraction is a $m \times 144$, where m is the number of images and 144 is the size of the descriptor.

Having explained how each descriptor type is extracted, the next step can take place, the visual vocabulary creation.

Visual vocabulary creation

Given a pre-defined number of images to extract descriptors from, a dataset of "words" is created, by simple concatenation of the descriptors of every image. This dataset can then be used to compute the final visual vocabulary, by the usage of k-means clustering: defined a number of clusters to determine (vocabulary size), the k-means algorithm can be applied in order to find the centroids matrix that will be needed later. Clustering can be a very memory intensive and time consuming task, so an efficient implementation has been used. This improvement, along with other optimizations, is explained in the *Optimizations and performance improvements* subsection.

Training dataset creation

After having calculated the centroid matrix, the next step is to build the dataset that will be used to train the classifier. This task has been accomplished by loading in memory every image in the training dataset *excluding the ones used to build the visual vocabulary* in order to extract relevant descriptors, used to create an histogram of the image: each histogram is created by taking all the descriptors of an image and determining for each one the nearest visual vocabulary word (k-means cluster centroid). This allows to build an histogram composed of *vocabulary_size* bins, where each bin value defines the number of times that the cluster in that position has been determined to be the nearest one for a certain descriptor. The training dataset is then created by concatenating each histogram, in order to create a matrix of $m \times$ *vocabulary_size* histograms, where m is the number of training images.

Classifier training

The classifier used in this project is composed by four different SVM classifiers trained in a binary class fashion, where each one can only separate one single class from all the others. The training procedure is simple and straightforward: first a set of training labels composed by ones (relevant examples) and zeros (non relevant examples) is created, one for each classifier. The resulting pair *(dataset_elements, one-hot_encoding)* is then used to train the classifiers in order to do binary classification. It is important to note that a parameter of the classifier is the kernel type (linear, rbf etc.) and like vocabulary size, it has to be defined prior to the execution of the algorithm.

Testing dataset creation

As with the training dataset building, in the testing dataset building task first every test image is loaded in memory, the descriptors are then extracted and the histograms computed. The histograms are then concatenated in order to create the testing dataset.

[1] `http://zhwang.me/publication/liop/LIOP_Paper.pdf`

Classifier testing

The testing phase is mostly similar to the training phase, as labels are computed and fed together with the testing dataset to the SVM, in order to extract predicted labels and probability estimates. Testing labels are computed in order to easily retrieve the model's accuracy. In order to do be able to compare the results of the different experiments, the Mean Average Precision (MAP) is calculated, as the mean of the Average Precisions (AP) of every model. Average precision is calculated as

$$AP = \frac{1}{m_c} \sum_{i=1}^{n} \frac{f_c(x_i)}{i}$$

and the MAP is calculated as

$$MAP = \frac{AP_{airplanes} + AP_{cars} + AP_{faces} + AP_{motorbikes}}{4}$$

Determined the MAP for a given choice of parameters (vocabulary size, kernel type and descriptor type), comparison between experiments can be done by analyzing the respective scores.

Experimental setup

As this project part gives a comparison between different parameters values, a pipeline has been created, in order to cycle through each combination and save the relative results. The ranges chosen for each parameter are:

- **Descriptor type**: every descriptor type has been used to run the experiments, in order to be able to isolate increases in performance given by the sole descriptor choice. The descriptors used are mentioned in the *Descriptor extraction* subsection.
- **Vocabulary size**: as the size of the vocabulary can create a high variance in results, it has been decided to run experiments under different values for this parameter, **400**, **800**, **1600** and **2000** when using SIFT, and **200** when using LIOP. The LIOP experiments have been done only with a vocabulary size of 200 because for every image only one descriptor is extracted, hence 200 descriptors in total. The clustering phase in this case is unnecessary but has been kept in order to use the existing project structure and pipeline. As the clustering phase can be very time consuming, experiments with a vocabulary of size 4000 have been skipped.
- **SVM kernel type**: different kernel types have been compared, namely **linear**, **rbf** and **polynomial**. Matlab SVM implementation has been used, as the kernel parameters can be set to automatically adjust based on the input data to classify.
- **histogram type**: initial experiments have been done in order to determine which histogram construction method is better, resulting in the choice of using a **probabilistic histogram** approach. The chosen construction method, along with the reasons behind the choice, can be found in subsection *Optimization and performance improvements.*
- **Scoring**: in order to give a qualitative score to every experiment, two different scoring functions have been used: **mean average precision (MAP)** and **average accuracy**. Accuracy has been used in order to compare the experiments against the results of the convolutional neural network in the second part of the project and the k-nearest neighbors classifier presented in subsection *K-nearest neighbors classification.*

Having chosen all the necessary parameter ranges, an exhaustive search of the parameters space has been done. An analysis of the results is given in the following subsection.

Results

In this section the experimental results are presented and analyzed, in order to detect if a particular hyper-parameters choice is to be preferable compared to others.

Table 1 shows the results gathered using the SIFT descriptor, using different sizes for the vocabulary and different kernel types for the SVM classifiers. A brief look at the values gives a immediate picture on the general performance of the system: every experiment results in scores above 95% mean average precision, and quite often above 98%. Taking into consideration that the experiments used a very small dataset of few thousand images, the final scores are quite surprising.

Looking at the scores associated with each vocabulary size for the same descriptor, it can be seen how the SVM scores are directly proportional to the increase in number of words in the vocabulary used. Comparing results using the lowest and highest value for the vocabulary size, shows that the average increase in performance is of 2% MAP, that is moderately high considering that the values are in the 97% to 99% range.

Descriptor type	Vocabulary size	MAP linear	MAP polynomial	MAP rbf
intensity-SIFT	400	0.95824	**0.97479**	0.97302
	800	0.97609	**0.98727**	0.97847
	1600	0.98152	**0.99116**	0.97848
	2000	0.98231	**0.9917**	0.98243
RGB-SIFT	400	0.95179	0.97905	**0.97983**
	800	0.97202	**0.98957**	0.98309
	1600	0.97964	**0.99175**	0.98723
	2000	0.98066	**0.99173**	0.98494
rgb-SIFT	400	0.97495	0.98873	**0.99103**
	800	0.98245	0.99115	**0.99267**
	1600	0.98641	**0.99408**	0.99026
	2000	0.98558	**0.99117**	0.98932
opponent-SIFT	400	0.9526	0.96318	**0.9742**
	800	0.9706	0.97982	**0.98302**
	1600	0.98017	0.98237	**0.98537**
	2000	0.9816	0.98523	**0.98538**

Table 1: Comparison of classification scores using SIFT descriptors

Table 2 shows the results for the dense variant of the SIFT descriptor, using the same settings for the vocabulary size and kernel type. Scores now are considerably higher, ranging from 98% to 100%, meaning perfect classification performance. In this case, increases in vocabulary sizes are moderately influencing like previous experiments, as the increase in MAP is on average 3%. Interestingly, it can be noted how the only classifiers that reached 100% accuracy used the intensity-SIFT descriptor.

The results suggest that, when using the non-dense SIFT version, there doesn't seem to be a clear descriptor type that performs all the others, and when using dense versions, intensity-SIFT seems to give slightly better results, even though very marginal. Taking into consideration the variance existing between different runs with the same parameters combination, there doesn't seem to be a clear "best" descriptor type. Increases in vocabulary size better helps the final performance of the experiments at the expense of computation times, that are increased mostly by the clustering and descriptor extraction phases. Looking at the differences between kernel type, it can be seen that rbf and polynomial kernels are to be preferred over the linear one, as it managed to get the highest scores only in 2 experiments over the total 96. Overall, the rbf kernel seems to give the highest number of best scores, but considering the variance existing in this methodology, it cannot be decided which kernel is to be considered the highest performing one.

When using the LIOP descriptor, as shown in Table 3, the scores are considerably lower: the highest

Descriptor type	Vocabulary size	MAP linear	MAP polynomial	MAP rbf
intensity-SIFT dense	400	0.99774	**0.99962**	0.99846
	800	0.99889	**0.99981**	0.99827
	1600	0.99971	**1**	0.99845
	2000	0.99971	**1**	0.9983
RGB-SIFT dense	400	0.99794	0.99872	**0.99952**
	800	0.99877	**0.99916**	0.99812
	1600	**0.99919**	0.9992	0.99796
	2000	**0.99962**	0.99927	0.99873
rgb-SIFT dense	400	0.99288	0.98113	**0.99819**
	800	0.99575	0.98977	**0.99844**
	1600	0.99616	0.99343	**0.99834**
	2000	0.99687	0.99293	**0.99842**
opponent-SIFT dense	400	0.99563	0.98932	**0.99843**
	800	0.99716	0.9946	**0.99873**
	1600	0.99883	0.99629	**0.99942**
	2000	0.9998	0.9988	**0.99981**

Table 2: Comparison of classification scores using dense SIFT descriptors

performance achieved is 96%, with minimums at 74%[2]. Considering that only one descriptor is extracted by default from each image (compared with 300-500 for SIFT and 1000 for dense-SIFT), much lower scores are to be expected. Unlike SIFT, the LIOP descriptor gives much lower results in normalized rgb color space, producing the three lowest scores achieved. Compared to SIFT, LIOP ensures much lower time spent in extracting features, at the expense of final performance.

Descriptor type	Vocabulary size	MAP linear	MAP polynomial	MAP rbf
intensity-LIOP	200	**0.90756**	0.88577	0.90573
RGB-LIOP	200	**0.93153**	0.92139	0.92806
rgb-LIOP	200	0.75548	0.74958	**0.77305**
opponent-LIOP	200	0.94894	0.94858	**0.96288**

Table 3: Comparison of classification scores using LIOP descriptors

Looking at the scores associated with the various kernel types, it seems that polynomial kernel produces the lowest scores. As the number of experiments in this case is very low, no kernel type can be excluded from the analysis.

K-nearest neighbors classification

Similarly to the SVM classification, a k-nearest neighbors classifier has been trained and evaluated, using the built in Matlab methods. Contrary from the previous model, composed of four different binary SVM classifiers, the knn classifier is composed of only one model, trained in order to accomplish full classification singularly, evaluated using the accuracy scoring function creating the results shown in Table 4. The results are for a vocabulary size of 1600 for SIFT and 200 for LIOP, comparing the knn classifier with k of 10 (neighbors number) against the SVM one, using rbf kernel for the latter.

Results show that in the most cases, the SVM classifier has better performance compared to knn, although when using LIOP, knn shows higher scores. Taking into account that the accuracies are based upon 50 images on average (as we only consider true positives), the knn classifier seems to be a relatively good alternative, if the user is not interested in ranking (as knn results are only labels) and doesn't require top performance in

[2]All experiments have been done multiple times, and results are consistent across the various runs.

Descriptor type	knn accuracy	SVM accuracy	knn accuracy (dense)	SVM accuracy (dense)
intensity-SIFT	0.775	**0.85**	0.96	**0.975**
RGB-SIFT	0.83	**0.88**	0.955	**0.965**
rgb-SIFT	0.895	**0.925**	0.97	0.97
opponent-SIFT	0.885	**0.935**	0.97	**0.975**
intensity-LIOP	**0.8**	0.77	-	-
RGB-LIOP	0.825	0.825	-	-
rgb-LIOP	**0.665**	0.565	-	-
opponent-LIOP	**0.865**	0.85	-	-

Table 4: Comparison of classification scores using knn classifier, vocabulary size of 1600 for SIFT, 200 for LIOP, compared to the SVM scores (rbf kernel).

terms of accuracy. Knn can be a viable alternative, as the training and testing times of the model are much faster than the SVM one.

Impact of training set size

As a relatively small dataset has been provided, a qualitative analysis on the training set size can be done, in order to determine if a bigger dataset would bring proportionally higher benefits to the final model. Figure 1 shows the incidence in MAP scores associated with the increase in training set size, from 5 images per class to 330 (the maximum as not all classes have the same number of images) with increments of 25. It can be seen that the increase in MAP is rapid when the training set is small and very small when having more images for the training.

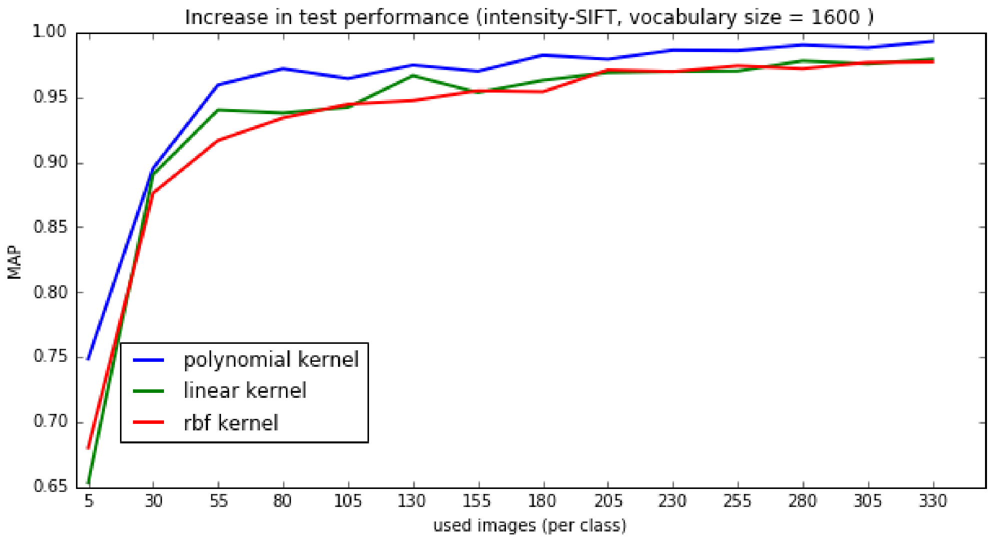


Figure 1: Variation of model performance with increase in training set size

Figure 2 shows the same results when using the dense version of the used SIFT descriptor: in this case, apart from the sudden increase in MAP when using 30 images, it can be seen that no substantial improvements are present when increasing the training set size.

Considering this results, it can be concluded that a big increase in training set size would not bring high benefits, as the curve has already starter to be asymptotic. A bigger dataset would be useful when peak performance is mandatory, and when computational resources are not an issue for the user.

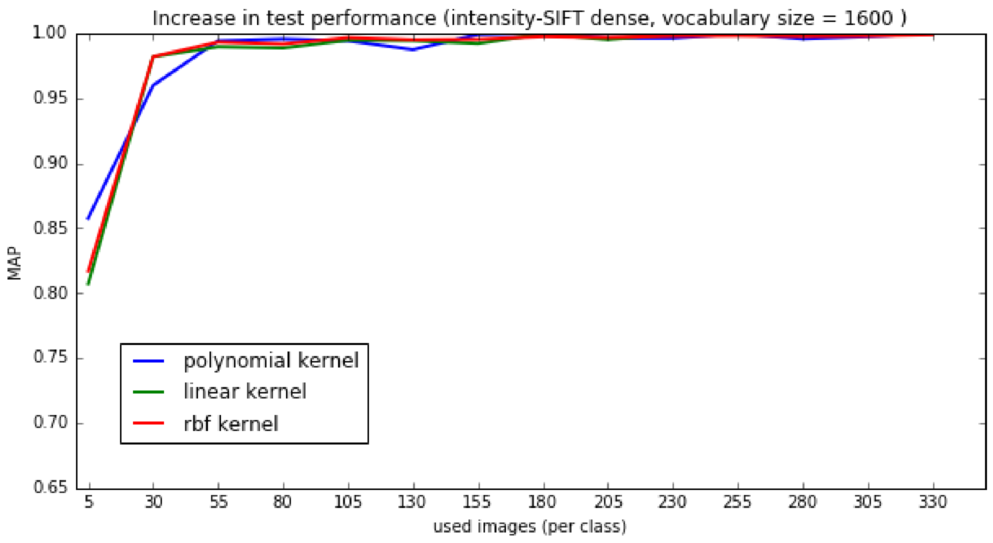


Figure 2: Variation of model performance with increase in training set size (dense SIFT)

Optimizations and performance improvements

In order to reduce time elapsed for each experiment, along with increasing the overall performance of the system, we used a number of expedients and improvements that are listed below.

histogram calculation

In order to determine the histogram associated with the descriptors of a certain image, two main strategies can be adopted: namely discrete and probabilistic histogram construction.

The first technique simply adds, for every descriptor, one count for the bin associated with the nearest descriptor, in order to output an histogram in which every bar represents the number of times a certain cluster has been found to be the nearest to one of the image descriptors. After the creation of the histogram, its values are normalized by the number of descriptors, in order to take into account the fact that different images will probably have a different number of extracted descriptors. This method underestimates a problem that has been noticed to be happening a considerable amount of times: the distance between the descriptor and the nearest cluster, compared to the second (in most cases also third and fourth) is marginally smaller, indicating that the point is situated almost halfway the two.

This issue has been resolved by using a probabilistic histogram creation: for each descriptor, instead of finding just the nearest one, the nearest n clusters are found, and their contribution (the inverse of the distance normalized) is stored instead of just the contribution of the closer one. At the end, like the other approach, the histogram is normalized by the number of descriptors.

k-means clustering

Experiments using the Matlab built in k-means clustering have been revealed to be very slow and memory intensive, causing problems and degradation in performance especially with dense sift descriptors, as the data loaded in memory started being moved to the swap memory, hugely decreasing performance. The adopted solution was to use the k-means algorithm implemented in the VL-FEAT library: the results changed from tens of minutes to less than three minutes on average, with a reduction in ram usage from 7-8 GB to 5 GB or even less. This optimization gave a noticeable boost in the experiment throughput.

Fast dense SIFT extraction

Extraction of dense SIFT descriptors takes more than double the amount of time compared to normal SIFT, and worsens the computation times of the system even more when rgb or opponent color spaces are used. This issue has been resolved using the *fast* version of the dense SIFT extraction, resulting in less the amount of time for the normal descriptor type. Fast version of dense SIFT uses a flat windowing function (compared to Gaussian), breaking exact SIFT equivalence but resulting in faster extraction[3].

[3] `http://www.vlfeat.org/matlab/vl_dsift.html`

Convolutional Neural Networks for Image Classification

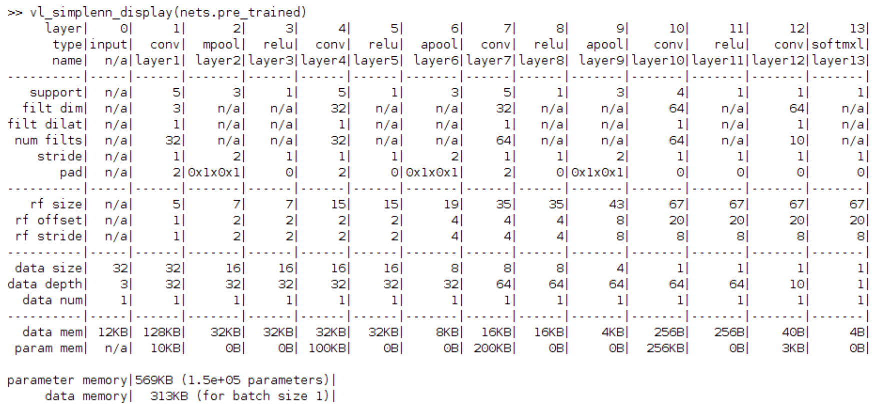
In this part of the project, image classification was achieved with Convolutional Neural Networks, rather than image descriptors like SIFT in the previous part. In order to utilize the resources available like pre-trained models, a method called Transfer Learning was adopted, to build a model on top another one that had its hyper-parameters trained on a different image dataset. Furthermore, the model architecture was modified due to the fact that the pre-trained model was designed for the CIFAR-10 dataset that has 10 categories of objects. This because the dataset used in this project, Caltech-101, only has 4 object classes. Hence, the output layer of the network architecture was modified in order to fit the number of object classes in the dataset, and re-train the hyper-parameters of the network.

Experimental Setup

- **Defining Network Architecture**: In order to conduct transfer learning based on the pre-trained network model, the network architecture was modified in order to generate the number of object classes needed. The network architecture is modified in a way that the size of the last convolution layers is changed from 10 outputs to 4, in order to satisfy the Caltech-101 dataset.
- **Data Preprocessing**: In the data preprocessing stage, the input images were resized to 32x32, and converted to single precision values in order to fit the network architecture. The input images had their mean subtracted, in order to reduce and stabilize the gradient during back propagation.

Network Architecture

The architecture of the network is shown in Figure 3: the main pattern observed from the network architecture is that the network consists of multiple block of layers, each includes a convolution, Relu, and a pooling layer. The rationale behind this architecture is because the Relu layer creates the non-linearity in the response of the convolution layer that extract the features from a number of filters. This is followed by a pooling layer that utilize the advantage of image input, such that the dimension of the input can be reduced across the network to produce a representative response of object classes.



```
>> vl_simplenn_display(nets.pre_trained)
     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 3: Convolutional Neural Network Architecture

In the network architecture, when comparing the different layers, it can be seen that the first layer has the biggest size. Although both of layer 0 and layer 1 have a size of 32, this only considers two dimensions of the volume. The depth of layer 0 and layer 1 is 3 and 32 respectively. Therefore, layer 1 is the largest layer due to the fact that it has the size of 32^3. On the other hand, layer 10 has highest number of hyper-parameters that can be inferred from the fact that it has the largest memory consumption of 256KB for the parameters. This can also be proven by calculating the number of parameters used in each layer. For example, the number of parameters used for each filter in layer 10 can be calculated by $4 \cdot 4 \cdot 64 + 1$, then multiplied by the number of filters, 64, which will in turn be the largest number among the network architecture.

Experiment with different parameter setting

In this part of the experiment the model parameters were adjusted based on empirical approach, such that a number of parameter settings were tested in order to determine the parameters that yield the least errors and objective function values. The objective function and top-1 error were used to evaluate the convergence behavior of different parameter settings. The experiments included parameters such as learning rate for different layers in the network architecture, the batch size, the number of epochs, and the weight decay of the network. The results are shown in Table 5.

Weight Decay	Batch Size	Epoch	Objective	Top-1 Error
	lr_prev = [0.01, 0.02], lr_new = [0.05, 1.0]			
0.0001	50	40	0.079	0.02
	50	**80**	**0.017**	**0.01**
	50	120	0.184	0.025
	100	40	0.102	0.01
	100	80	0.163	0.04
	100	120	0.044	0.015
	lr_prev = [0.005, 0.03], lr_new = [0.05, 1.0]			
0.0001	50	40	0.199	0.06
	50	80	0.221	0.035
	50	120	0.091	0.01
	100	40	0.168	0.04
	100	80	0.135	0.045
	100	120	0.066	0.025
	lr_prev = [0.005, 0.03], lr_new = [0.1, 0.6]			
0.0001	50	40	0.109	0.045
	50	80	0.171	0.04
	50	120	0.221	0.05
	100	40	0.109	0.035
	100	80	0.227	0.045
	100	120	0.505	0.065

Table 5: Experiment of different parameter settings

From the result in Table 5,it can be seen that larger learning rate in the previous layers enables the model to converge faster. For example, comparing the result of higher learning rate in section 1 of the result table to lower learning rate in section 2, the objective function in the section 2 required more Epochs before reaching the same objective function value and the Top-1 error. This indicates that larger learning rate enables the model to converge faster. On the other hand, the learning rate of the new layer does not make significant difference in the convergence behavior as shown in section 2 and 3 of Table 5.

The best objective function value was determined to be 0.017 with top-1 error of 0.01 given a batch size of 50, epochs of 80, early layer learning rate of (0.01, 0.02), new layer learning rate of (0.05, 1.0), and weight decay of 0.0001. Given this parameter setting, a further experiment was performed to check the effect of weight decay, and results were recorded in Table 6. It can be seen that the minimum objective function value and the minimum top-1 error was also achieved under the same parameter setting. The convergence behavior of the objective and top-1 error throughout the training can be seen in Figure 4.

The parameters that yield the best minimum objective function value and top-1 error are then used to determine the accuracy of the model, which is explained in the section **Evaluation Accuracy**.

Weight Decay	**Batch Size**	**Epoch**	**Objective**	**Top-1 Error**
	lr_prev = [0.01, 0.02], lr_new = [0.05, 1.0]			
0.00001	50	40	0.201	0.03
	50	80	0.133	0.03
	50	120	0.103	0.015
0.0001	50	40	0.079	0.02
	50	**80**	**0.017**	**0.01**
	50	120	0.184	0.025
0.001	50	40	0.119	0.02
	50	80	0.171	0.01
	50	120	0.084	0.01

Table 6: Experiment of different parameter settings

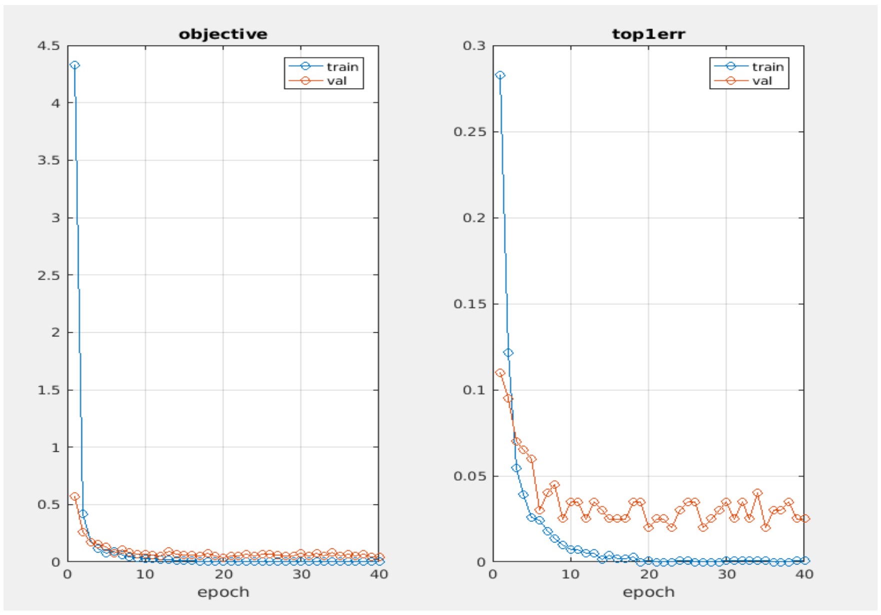


Figure 4: Training objective and top-1 error.

Feature Space Visualization

In this section, the feature space of images extracted with both pre-trained model and the fine-tuned model is visualized by converting the 64-dimensional features extracted at layer 10, into 2-dimension space with T-SNE dimensionality reduction. The visualization of features for the training image and testing image can be seen in Figure 5 and Figure 6, where the plot on the left hand side is the feature space extracted from the pre-trained model, and the plot on the right hand side is extracted from the fine-tuned model. In the plot, the color represents the label of the images.

The major differences of the feature space between the two model is the distribution of features of different labels. It can be seen that the features extracted by the fine-tuned model is more discriminative in terms of the labels. The features of the same object class tends to be clustered together. On the other hand, the feature space generated by the pre-trained model shows a result that many features of different object classes are spread among the feature space, such that many features of different object classes appeared to be in the same region in the feature space. This is due to the fact that the hyper-parameters in the fine-tuned model is adjusted to differentiate the four object classes that are interested, whereas the pre-trained model generated ten classes to represent the dataset that has only four object class, which means the class responses does

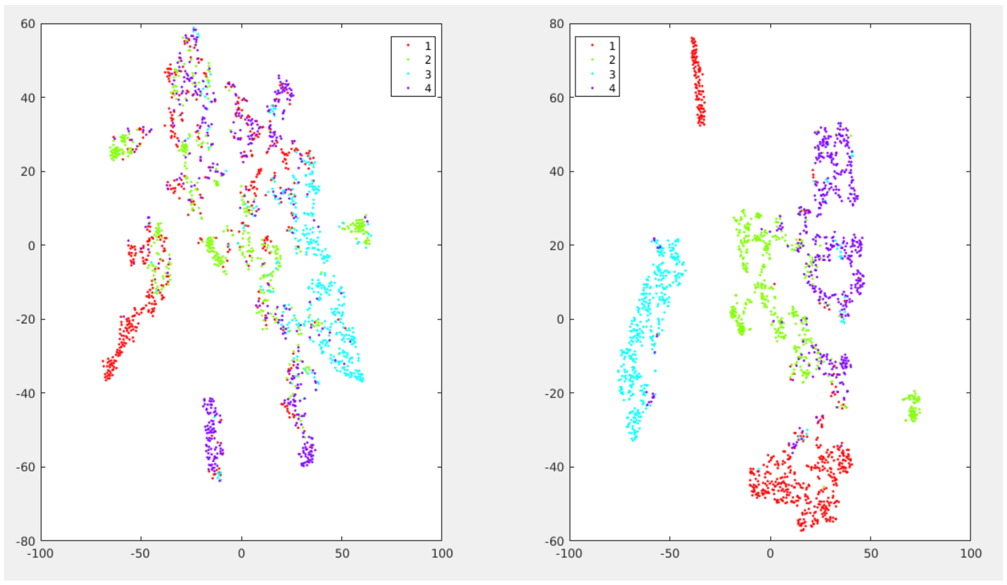


Figure 5: Feature Space Visualization - Training data

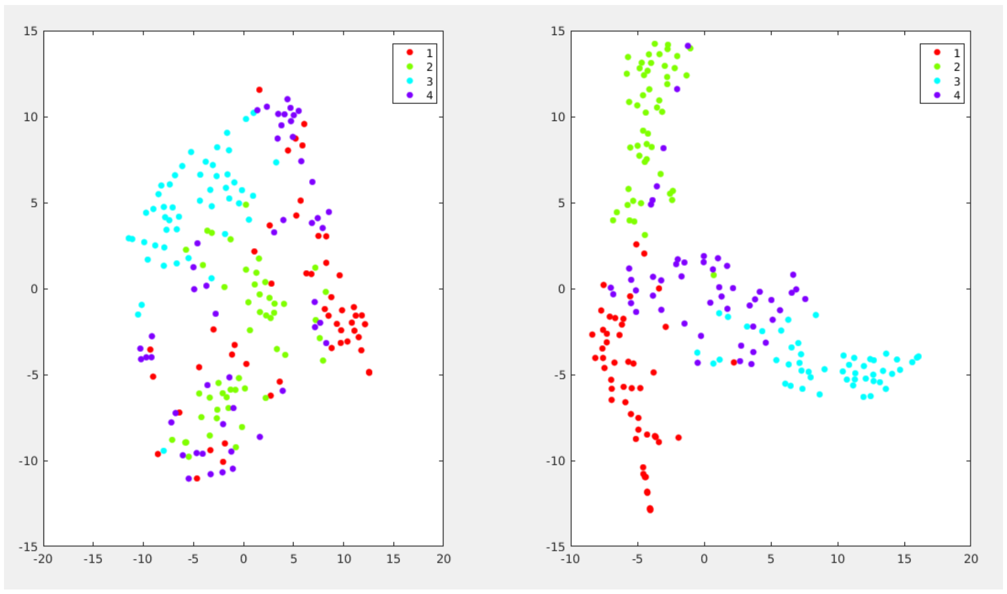


Figure 6: Feature Space Visualization - Testing data

not necessarily representative for a specific image label. Therefore, the pre-trained model does not produce a feature space that is as discriminative as the feature space generated by the fine-tuned model.

Accuracy evaluation and comparison with BoW approach

After the best set of parameters were determined by the minimum objective function value and top-1 error, the accuracy of the fine-tuned CNN model was determined, as well the accuracy of the SVM models. The SVM models were trained based on the features extracted from layer 10 of the network, which provides 64-dimensional features representation of the image, for both pre-trained and fine-tuned network.

From the result shown in Table 7, it can be seen that the accuracy of the CNN fine-tuned model, SVM pre-trained, and SVM fine-tuned is 99%, 95%, and 97.5% respectively. This result shows that the accuracy of SVM classifier is improved by 2.5% with fine-tuning the CNN model that is used to extract the features from the images. This result corresponds with the feature space illustrated in the previous section, such that the fine-tuned CNN is able to extract features that are more exclusive to a specific object class, which is the potential reason that enables the SVM classifier to predict more accurately with the fine-tuned than the pre-trained model. The accuracy of the SVM fine-tuned is slightly lower than the fine-tuned CNN, however, these are two different approaches for image classification and cannot really be compared.

As a result of conducting experiment with both the Bag-of-Words and Convolutional Neural Network approach, it was found that the CNN approach achieved an accuracy of 99%, which is slightly higher than 97.5% from the Bag-of-Words approach. Although the difference between the result of two approaches are small, the CNN approach provides an end-to-end solution for image classification, without the need of extensive processing pipelines like extracting SIFT descriptors, and building the visual vocabulary with clustering, then finally training the SVM classifier.

Method	**CNN Fine-tuned**	**SVM Pre-trained**	**SVM Fine-tuned**
CNN	0.99	0.95	0.975
	SVM - Dense, opponent SIFT		
BoW	0.975		

Table 7: Accuracy of CNN and SVM.

Optimizations and improvements

Data Augmentation

In this section, the experiment regarding the implementation of data augmentation was performed based on the parameter setting as shown alongside the result in Table 8. The rationale of not using the parameter setting that yielded the minimum objective function and top-1 error is because the accuracy is already at 99%, which potentially makes the observation of minor influences due to data augmentation unnoticeable.

The effect of data augmentation was evaluated by comparing the result against the result generated with removal of original basic data augmentation, the random image flipping.

- **Rotation**: The first data augmentation technique applied was rotating 50% of the training images. The angle of rotation is determined randomly between 0-359, 0-20, and 0-10 degrees, for three experiment respectively. The result is shown in 8, which indicates that the accuracy is reduced from 97% to 94% when the images are rotated between 0 and 359 degrees, whereas the accuracy maintained close to the model trained without data augmentation, approximately 98%. The potential explanation for this behavior is that the images in the dataset are rather coherent in terms of rotation, such that substantial rotation in the training data only create unnecessary noise.
- **Gaussian Noise**: The second data augmentation technique applied was adding Gaussian noise to 50% of the training images. The result shown in Table 8 indicates that the effect of adding Gaussian noise to the training image is limited, only slightly decreased the accuracy to 95%, which can be potentially due to randomness. In addition, it was found that the variance of the Gaussian noise distribution does not affect the accuracy, such that the three experiment of different variance of 0.01, 0.1, and 1.0 result in the same accuracy for the fine-tuned CNN model.

Freezing Early Layers

In this section, the parameters in the early layers of the network was frozen by setting the learning rate to 0, so that the parameters are solely transferred from the old model and will not be adjusted according to the new object classes during the training. Therefore only the hyper-parameters in the last layer of the network will be tuned. The result of freezing early layers is shown in Table 9. It can be seen that the accuracy of the fine-tuned CNN is reduced to approximately 84%, from 99% with the optimal parameters determined in the previous section. The rationale behind the reason of why the accuracy of the fine-tuned CNN is much lower than the SVM pre-trained and SVM fine-tuned is because the features extracted from the CNN does not provide enough discriminative capacity over the object classes in the image, and it relied on the SVM to compensate such deficiency. This can be seen in the illustration of feature space in Figure 7, such that there is no obvious clusters that discriminates the features according to their labels effectively, comparing to the difference between features extracted with pre-trained and fine-tuned model as illustrated feature space shown in Figure 5, when the entire network hyper-parameters were tuned.

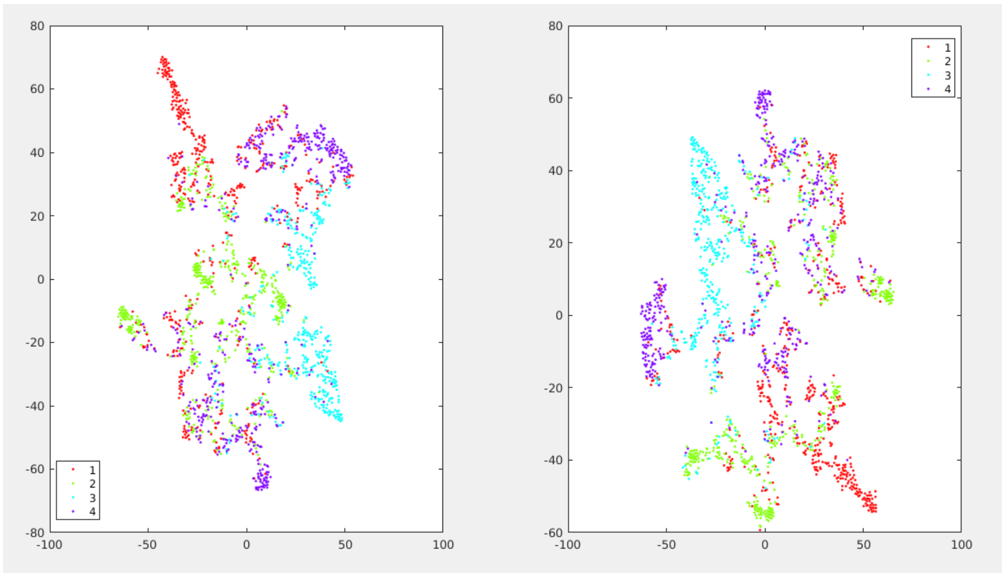


Figure 7: Illustration of feature space for freezing early layers implementation. Pre-trained network on the left and fine-tuned network on the right.

Technique	**Parameters**	**CNN Fine-tuned**	**SVM Pre-trained**	**SVM Fine-tuned**
	lr_prev = [0.01, 0.02], lr_new = [0.05, 1.0], weight decay = 0.0001			
Baseline	NA	0.97	0.95	0.98
Rotation	$\theta = [0, 359]$	0.94	0.945	0.975
	$\theta = [0, 20]$	0.98	0.935	0.98
	$\theta = [0, 10]$	0.98	0.935	0.985
Gaussian Noise	$\mu = 0, \sigma = 0.01$	0.95	0.935	0.985
	$\mu = 0, \sigma = 0.1$	0.95	0.95	0.96
	$\mu = 0, \sigma = 1.0$	0.95	0.95	0.955

Table 8: Data Augmentation Result

Weight Decay	**Batch Size**	**Epoch**	**CNN Fine-tuned**	**SVM Pre-trained**	**SVM Fine-tuned**
	lr_prev = [0.0, 0.0], lr_new = [0.05, 1.0]				
0.0001	100	40	0.83	0.945	0.945
	200	80	0.84	0.95	0.95
	200	120	0.84	0.945	0.945
	lr_prev = [0.0, 0.0], lr_new = [0.1, 1.0]				
0.0001	100	40	0.81	0.95	0.95

Table 9: Accuracy achieved with freezing early layers