# Multi-Class Image Classification

Dana Kianfar - Jose Gallego Posada
*11391014 - 11390689*
`{dana.kianfar, jose.gallegoposada}@student.uva.nl`

## 1 Introduction

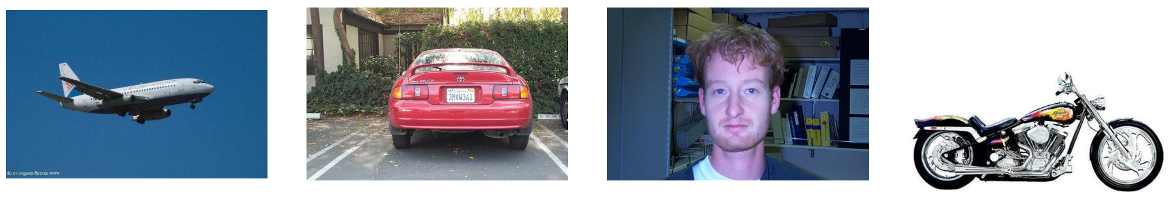
In this project we explore the Bag of Words (BoW) and Convolutional Neural Network (CNN) methods for performing image classification. We are provided with a subset of the CalTech 101 dataset with four classes representing airplanes, faces, cars and motorbikes. Under different parameter settings, we train BoW models and fine-tune a pre-trained CNN in a transfer learning setup. We compare and visualize the accuracy of both methods and discuss our best performing models.

Our results indicate that it is possible to obtain similar performance between both methodologies, reaching accuracies of 99.5% for the CNN and 97.25% for BoW.

This report is structured as follows: in Section 2 we describe the data used in this work, Section 3 provides details regarding the used techniques and the executed experiments. In Sections 4 and 5 we display our results and additional experiments. Finally, Section 6 contains the main conclusions of this project.

## 2 Dataset

Our dataset consists of 1865 training and 200 testing images with roughly the same distribution over each class. Figure 1 provides an example for each class.



(a) Airplane (b) Car (c) Face (d) Motorbike

Figure 1: Example images from each class

We note that the images in each class vary in backgrounds, texture, shape, lighting and color. They are mostly consistent with respect to pose, occlusion and location. Additionally original image sizes may vary.

## 3 Methodology

### 3.1 Bag-of-Visual-Words

Image classification using the BoW technique consists of the two major steps, namely the creation of a visual vocabulary and the actual classification. In our approach, we randomly assign 50% of the training set (roughly 1000 images) to each task, and will refer to them as the clustering and classification training sets.

We begin by building a visual vocabulary. This is accomplished by extracting feature descriptors from each image in the clustering training set using SIFT [1] and clustering descriptors from all images using K-means. We obtain $K$ centroids as representatives of the descriptor space. Each centroid is a visual word, and we use the set of $K$ visual words to represent images in a standard and comparable format. To achieve this, we extract SIFT descriptors from each image and attribute each descriptor to a visual word (for instance, by choosing the closest centroid with the Euclidean distance). Note that the SIFT descriptors must be extracted under the same setting as the clustering model, i.e. using the same colorspace and sampling density. The final representation of an image is a histogram (or BoW) of the visual word attributions, where we count the number of times each centroid was attributed to a descriptor in an image. Note that this is an orderless representation, and all spatial information is lost.

To perform classification, each image in the classification training set is represented as a BoW with respect to the visual words. As this representation provides us with a fixed-length vector of size $K$, we can train a classifier such as a support vector machine (SVM).

**Feature Extraction** We use VL_FEAT's `vl_sift` and `vl_dsift` implementations for keypoint and dense sampling respectively. We use the *RGB*, *HSV*, *rgb*, *grayscale*, *opponent* color spaces for extracting features from each image. Dense sampling is performed with a *block-size* of 3 and a *step* of 10 pixels which, depending on the image size, results in roughly 1350 keypoints per image. The block-size and step parameters were chosen empirically to avoid computational overhead. Keypoint sampling is performed on the *grayscale* version of an image to obtain a set of keypoints. Descriptors for each colorspace are extracted at the locations of these keypoints. If a colorspace has more than one channel, we extract SIFT features from each channels separately and then concatenate corresponding descriptors in all channels horizontally to obtain a descriptor vector of size $3 \times 128 = 384$.

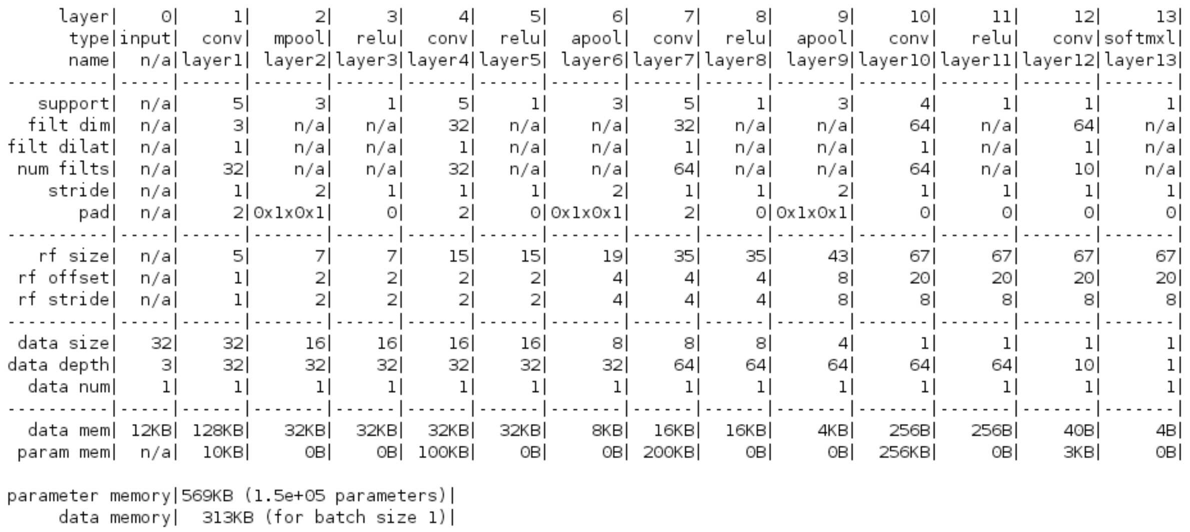
**Clustering** As SIFT descriptors are of `uint8` type, we use the integer K-means `vl_ikmeans` implementation in VL_FEAT, with 800 maximum iterations and Elkan's algorithm [2]. For the number of clusters $K$, we perform a grid search on the values $[400, 800, 1600, 2000, 4000]$. Due to limited computing resources, we do not train use the entire clustering training set and instead randomly subsample 400 images (100 from each class).

**Classification** We use SVM classifiers with *linear, RBF* and *polynomial* kernels. For the linear kernel SVM, we use LIBLINEAR [3]. For the RBF and polynomial kernels, we use MATLAB's `fitcsvm` implementation. A binary classifier is trained for each class. The entire classification training set is used to train the classifiers. Due to limited computational resources, we do not perform cross-validation to obtain the best (hyper)parameters for the SVMs.

Overall, we perform 150 experiments, where each experiment uses 400 images for training the clustering model and 932 images for training a classifier. The results for each experiment is presented in Table 4 of the Appendix. We provide a summary of the best performing models in Section 4. For efficiency we store intermediate results at each stage of the training process to disk.

## 3.2 Convolutional Neural Networks

CNNs are a Multi-Layer Perceptron (MLP) architecture that exploits a weight-sharing and local connectivity paradigm towards learning filters that can be used as feature extractors based on which the classification task is tackled using standard SVM or MLP techniques. For the current work, we are provided with a CNN that has been previously trained for a classification task using the CIFAR-10 dataset. The structure of the network is displayed in Figure 2.



```
     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 2: Architecture of the pre-trained network.

It is possible to observe that the network is formed by a repeating structure of:

- image filtering via convolution layers,

- dimensionality reduction/subsampling via average or max pooling layers,
- non-linear activation functions via RELU layers,

additionally, across the network the size of the filters increase, while the image itself (i.e. subsequent activation maps) decreases.

Even though the network has a large number of parameters (around 150.000) this is much lower than the number of weights that a fully connected architecture would have, and this is one of the main advantage of the paradigms mentioned earlier. These parameters are precisely concentrated in the convolutional layers of the network, as they form the filters that will be applied to the successive images/activation maps in the forward-propagation stage. Note that the employed subsampling and activation functions do not require the specification of further parameters. Finally, observe that layer 12 indicates that the pre-trained network was designed to be executed in a 10-class setting.

The main goal in this section is to used this pre-trained model as a starting point towards building a system that is specifically designed for the classification of the mentioned 4 classes. This approach is known as transfer learning. It is expected that the pre-trained model can exhibit acceptable performance given that the nature of the data on which it has been trained is similar to the intended use of the classifier. Note that the CIFAR-10 dataset includes airplanes and automobiles but does not contain faces or motorbikes. Of course, it is also expected that a fine-tuned system achieves better performance.

We execute 6 experiments by changing the batch size and number of epochs for used in the network training. Additionally, we perform dimensionality reduction using TSNE on the extracted features of the images as well as a visualization of the learned filters in the first layer for both our best fine-tuned network and the AlexNet [4] model from MATCONVNET.

In order to use the provided architecture, we scale all the images to be 32x32. Images with grayscale values are replicated on RGB channels.

## 4 Results

### 4.1 Bag-of-Words

| **id** | **K** | **Density** | **Colorspace** | **Classifier** | **Airplane** | **Car** | **Face** | **Bike** | **mAP** |
|---|---|---|---|---|---|---|---|---|---|
| **1** | 800 | dense | HSV | linear | **0.999** | 0.988 | 0.993 | **0.984** | **0.991** |
| **2** | 400 | dense | Grayscale | linear | 0.986 | **0.990** | 0.997 | 0.973 | 0.986 |
| **3** | 1600 | dense | Grayscale | linear | 0.984 | 0.974 | **0.998** | 0.978 | 0.983 |

Table 1: Best BoW models per class

We display the best model for each class in Table 1, where the parameters of each model is presented alongside the mean average precision (mAP) and per-class average precision

(AP). The best mAP was obtained using model 1, which also achieves the best AP for the Airplane and Motorbike classes. We present the Receiver Operating Characteristic (ROC) curves for models 1 and 2 in Figure 3. Model 1 performs better with an average area-under-the-curve (AUC) of 0.9970, while model 2 has an average AUC of 0.9950.

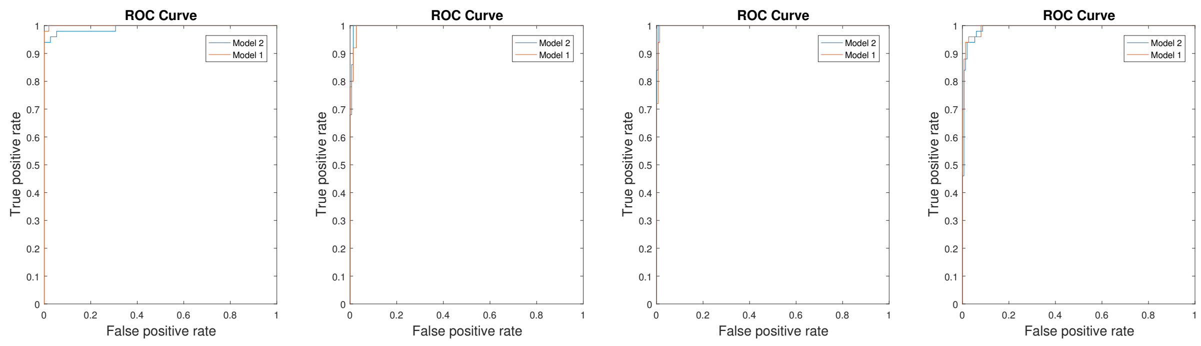


Figure 3: Per class ROC curve for models 1 and 2. Left to right: airplanes, cars, faces, motorbikes.

### 4.1.1 Evaluation

In this section we evaluate the effect of each parameter. We will denote the mean and standard deviation of the mAP values obtained under a parameter settings as $\mu$ and $\sigma$. Additionally, we use $p$ to denote the p-value of a one-sided sign binomial test of two mAP value distributions. These tests are conducted based on the ratio of mAP estimates of two parameter settings, therefore an element-wise division of two mAP vectors. The ratio is taken with the intention of evaluating whether the numerator is larger in magnitude than the denominator. The test is thresholded at 1 with the significance set at 0.05.

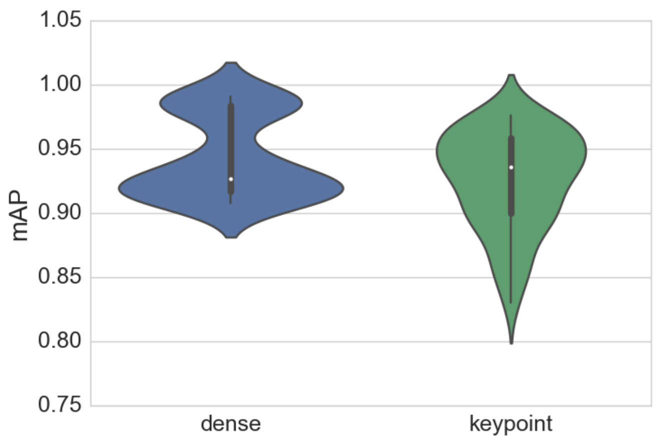


Figure 4: Distribution of mAP values across dense and keypoint sampling, where we have $N = 75$ points per distribution.

**Feature Point Sampling** In our experiments we observed that dense feature sampling performs better on average than keypoint sampling, as expected. We attribute this finding to having more feature points per image, which results in a more accurate description of the images and more robust classification models. We visualize the two distributions in Figure 4. For dense models, $\mu = 0.944$ and $\sigma = 0.0314$, and for keypoint models $\mu = 0.926$ and $\sigma = 0.0373$. With $p = 0.00003$, the null hypothesis is rejected in favor of dense sampling.

**Vocabulary size $K$** In our experiments, we did not observe a trend preferring any value of $K$ over others. As shown in Figure 5, for increasing values of $K$ the variance of the distribution decreases while the median increases. While our best models have $K \in [400, 800]$, models with $K = 4000$ has the highest mean and least variance, with $\mu = 0.944$ and $\sigma = 0.0296$. However, none of our sign-binomial tests conducted against all pairs of values for $K$ was able to reject the null hypothesis.

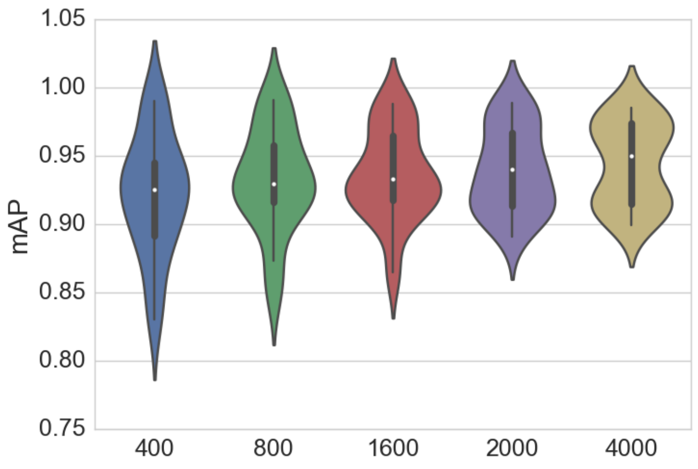


Figure 5: Distribution of mAP values across different vocabulary sizes $K$, where we have $N = 30$ points per distribution.

We postulate that the vocabulary size should be proportionate to the number of training samples available, such that larger values of $K$ should be trained with larger datasets to obtain a representative vocabulary in a high-dimensional space. Similarly, more training samples for larger vocabularies may improve the discriminative power of the classifier. Note that the performance of the clustering algorithm is also sensitive to its initialization, and to find a good value for $K$ it is necessary to train several models. Within the limits of our computational resources, we were not able to train our K-means models several times.

**Colorspace** Across our experiments, we did not have a clear winner among the colorspace. As illustrated in Figure 6, we note that the *HSV* color space is the most robust with $\mu = 0.94$ and the smallest standard deviation $\sigma = 0.0295$ among the colorspaces. Both *rgb* and *RGB* have a higher mean, but also higher variance. However, in a sign-binomial test, the null hypothesis could only be rejected in favor of HSV against the *grayscale* and *opponent* colorspaces with p-values of 0.049 and 0.021 respectively. However, we note that as the grayscale colorspace has the highest variance, it has the highest mAPs among the other colorspaces. Indeed models 2 and 3 are both trained using the grayscale colorspace.

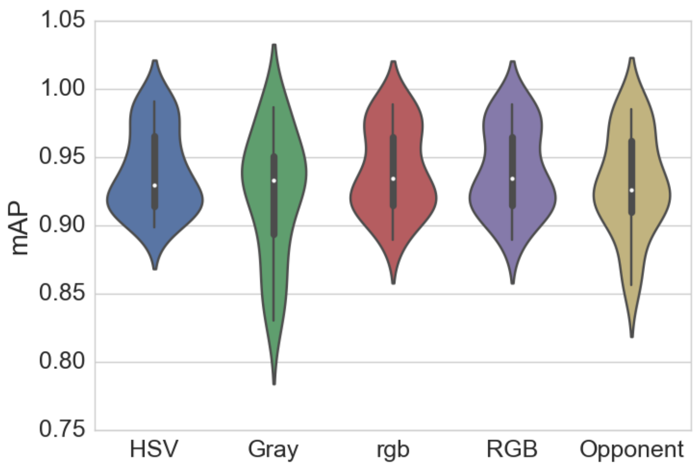


Figure 6: Distribution of mAP values across different colorspaces, where we have $N = 30$ points per distribution.

**Classifier** In our main experiments, we use three SVM classifiers with linear, RBF and polynomials kernels. With the exception of the linear SVM, within the scope of this project we did not tune the parameters of our SVM classifiers. Across our experiments, we obtained the best performance with our linear SVM. We attribute this to having performed a simple search to find an optimal $C$ value for the linear kernel, whereas no parameter search was performed for the RBF or polynomial SVM.

## 4.2 Convolutional Neural Networks

Table 2 displays the result of the hyperparameter selection experiments. We denote by FT or PT, fine-tuned or pre-trained features, respectively. For instance, PT-SVM stands for the SVM classifier with pre-trained features.

| **Batch Size** | **Epochs** | **FT-CNN** | **PT-SVM** | **FT-SVM** |
|---|---|---|---|---|
| 50 | 40 | 99.5 | 90.5 | 98.5 |
| 50 | 80 | 99.5 | 90.5 | 99.0 |
| **50** | **120** | **99.5** | **90.5** | **99.5** |
| 100 | 40 | 99.5 | 90.5 | 99.0 |
| **100** | **80** | **99.5** | **90.5** | **99.5** |
| 100 | 120 | 99.5 | 90.5 | 98.5 |

Table 2: CNN hyperparameter tuning.

As can be seen in the table, the best results are obtained for the models trained with batch size 50 and 120 epochs, and batch size 100 and 80. Both achieve 99.5% accuracy on the test set. Given that the test set is formed by 200 images, this means that only 1 of the images is being misclassified. Given the similar accuracy on both parameter configurations, we chose batch size 100 and epochs 80 to be the best model given that the increased batch size allows for more stable gradients, while having a reduced number of epochs. The reader should be aware of the additional trade-off between larger batch size and computational time per batch.

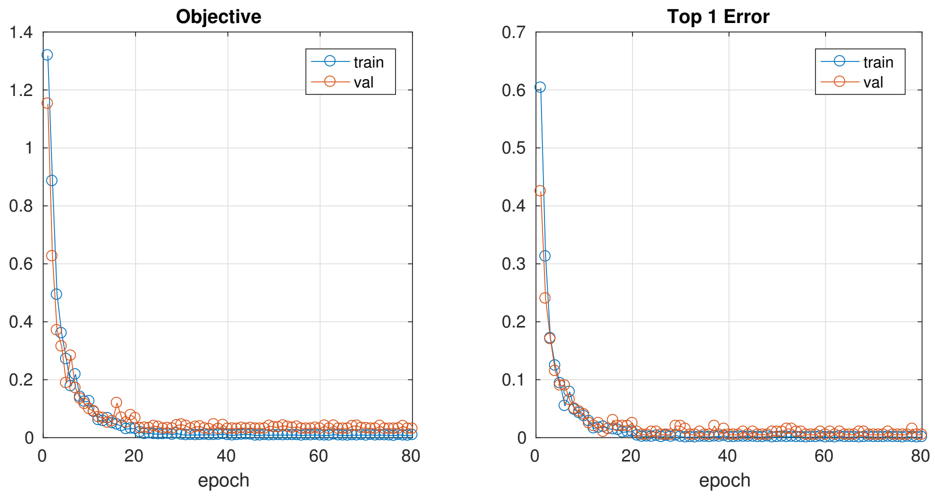


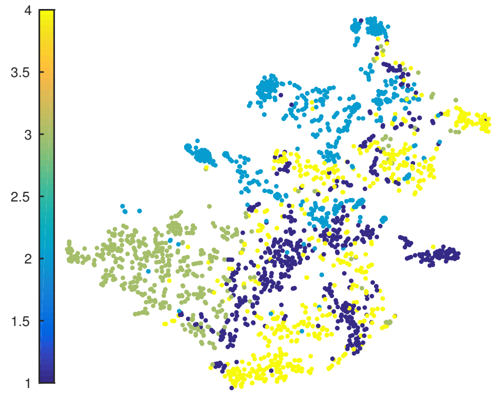
Figure 7: Training results for CNN with batch size 50 and 80 epochs.

For brevity, only the training results of the selected model is displayed in Figure 7. The reader can find all the models and their training logs in the source code of the project. It is possible to note that both the objective and top-1-error converge towards zero, with a better performance on the training set versus the validation error, as expected.

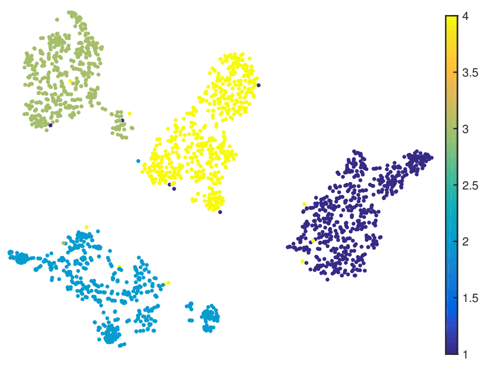
In order to analyse the characteristics of the features extracted by the CNN, we executed dimensionality reduction via TSNE for the pre-trained and fine-tuned extracted features. The results are shown in Figure 8. The color of the point represents the predicted class.

It is evident that there exists a clearer separation in the feature spaced of the fine-tuned network as compared to the pre-trained one. Nevertheless, in both maps it is possible to observe that there is a trend towards clustering together points of the same predicted class. This shows that, indeed, the fine-tuning procedure is allowing the network to focus on the task of separating the four classes of interest.

However, this separation is not perfect and we decided to investigate some of the errors present in Figure 8(b). We present two of these cases in Figure 9. Note that these images are already pre-processed: compressed size and mean shifted. Figure 9(a) shows what could be classified as a hard classification instance, while 9(b) shows an examples that was correctly classified by the CNN. Explaining why this is happening would require a depper understanding of TSNE and, for that reason it is out of the scope of this project.

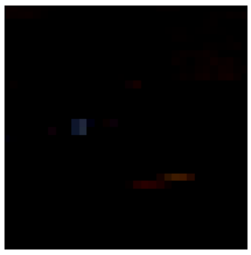


(a) Pre-trained features

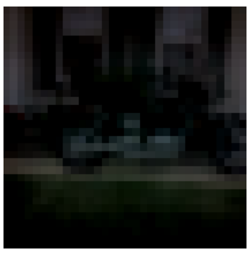


(b) Fine-tuned features

Figure 8: Results of TSNE dimensionality reduction.



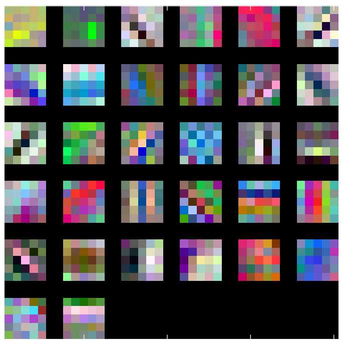
(a) Hard instance



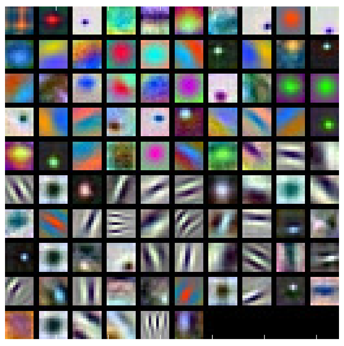
(b) Right classification, wrong cluster

Figure 9: Abnormal points in TSNE results.

Additionally, Figure 10 provides a visualization of the filters on the first convolutional layer for both the fine-tuned CNN and the AlexNet model. Note that most of these filters show patterns that resemble edge and blob detectors, as well as some texture and colors.



(a) Fine-tuned first layer filters



(b) AlexNet first layer filters

Figure 10: Visualization of learned filters.

# 5 Bonus

## 5.1 Bag-of-Words

We implemented and experimented with *Gaussian Mixture Models* for clustering, and *HOG features.* Within the time frame of the project, we were not able to achieve useful results. We include the code in our submission.

We used the **HSV colorspace** in our main experiment and obtained promising results that were discussed in Section 4. The HSV colorspace was the most robust colorspace with respect to model parameters.

We implemented the **K-Nearest Neighbors** (KNN) algorithm as a classifier in a restricted set of experiments. Due to limited computational resources, we limited other parameters to those shown in Table 3. We experimented with differnet $k$ values for the KNN algorithm ranging within $[1, 5, 10, 20]$. Despite the simple setup, the KNN obtained good performance. The best value for $K$ is 1 where the mAP was 0.939.

| **Vocab. Size** | **Density** | **Colorspace** | **KNN K** | **Airplane** | **Car** | **Face** | **Bike** | **mAP** |
|---|---|---|---|---|---|---|---|---|
| 400 | dense | Grayscale | KNN-1 | **0.947** | **0.992** | 0.940 | **0.876** | **0.939** |
| 400 | dense | Grayscale | KNN-5 | 0.907 | 0.966 | **0.986** | 0.835 | 0.924 |
| 400 | dense | Grayscale | KNN-10 | **0.947** | 0.966 | **0.986** | 0.834 | 0.933 |
| 400 | dense | Grayscale | KNN-20 | 0.903 | 0.961 | 0.931 | 0.753 | 0.887 |

Table 3: KNN models performance

## 5.2 Convolutional Neural Networks

A common way to increase the available number of instances for CNN training is to artificially **augment** the training set by applying transformations to the original images. For this, we modified the training procedure such that with certain probability the image would be flipped horizontally or rotated a random number of degrees between -10 and +10, blurred with a Gaussian filter, or stay untransformed. Then we trained the network using batch size 50 and 40 epochs, and obtained 99.5% accuracy for both FT-CNN and FT-SVM.

Another method that is often used during the fine-tuning of CNNs is to **freeze** the early layers of the network and only update those closest to the output layer. This is generally used when the network is very deep and full gradient updates would be too computationally expensive. Using batch size 50 and 40 epochs we obtained accuracy FT-CNN: 57% and FT-SVM: 80%. We think this might be due to the network overfitting the last layers to the training set.

As an additional experiment, we decided to drastically **change the activation function** which the network used and replace the original RELU with a a sigmoid activation. Then we trained the neural network for 40 epochs with batch size 50. This generated a decrease in the performance of the system for both FT-CNN and FT-SVM, which obtained 91.5% and 94.5% accuracy, respectively. However, PT-SVM got a slight increase to 91% accuracy. We find very surprising that the parameters of the network are robust enough to compensate for the change in the activation functions, while still retaining an admissible performance above 90%.

A common regularization technique used for CNNs is **dropout**. According to the authors, "the key idea is to randomly drop units (along with their connections) from the neural network during training. This prevents units from co-adapting too much" [5]. We included a dropout layer before the last convolutional layer in the network and trained it for 40 epochs with batch size 50. The accuracy of FT-CNN is 98.5%, while FT-SVM achieves 99.0%. Note that there is a low decrease in the performance of the system, which might be due to a very strong regularization.

## 6 Conclusions

The test accuracy obtained by the best models was 99.5% for the CNN and 97.25% for BoW. This shows that both methods were successful in tackling the proposed classification task, with a slight advantage to the neural models.

The transfer learning setup provided good results in comparison to the pre-trained model. However, we consider that the current 4-class task was too simple to show significant differences in the performance of the methods. For instance, no exploration has been made in this work regarding the robustness of the models with respect to changes in illumination, pose or occlusions.

The BoW model was executed with a reduced sample of the training set towards the construction of the visual vocabulary. It might be possible to build a better visual vocabulary by using a larger dataset and thus achieve higher performance. Additionally, due to time constraints, it was not possible to execute k-Means multiple times in order to improve over with respect to locally optimal solutions. Similarly, we were not able to tune parameters for our SVM classifiers. We note that by using cross-validation to find good parameters for the SVMs, the performance of the models could be improved substantially.

We discovered that, although in general the models exhibit an acceptable performance, a correct tuning of hyperparameter combinations can generate improvements in performance of around 10% for both methods. For future work, using spatial pyramids may increase the performance of the model, regardless of the regularity of images in our dataset.

## References

[1] D. G. Lowe, "Distinctive image features from scale-invariant keypoints," *International journal of computer vision*, vol. 60, no. 2, pp. 91–110, 2004.

[2] C. Elkan, “Using the triangle inequality to accelerate k-means,” 2003.

[3] R.-E. Fan, K.-W. Chang, C.-J. Hsieh, X.-R. Wang, and C.-J. Lin, “LIBLINEAR: A library for large linear classification,” *Journal of Machine Learning Research*, vol. 9, pp. 1871–1874, 2008.

[4] A. Krizhevsky, I. Sutskever, and G. E. Hinton, “Imagenet classification with deep convolutional neural networks,” in *Advances in Neural Information Processing Systems 25*, F. Pereira, C. J. C. Burges, L. Bottou, and K. Q. Weinberger, Eds. Curran Associates, Inc., 2012, pp. 1097–1105. [Online]. Available: http://papers.nips.cc/paper/4824-imagenet-classification-with-deep-convolutional-neural-networks.pdf

[5] N. Srivastava, G. E. Hinton, A. Krizhevsky, I. Sutskever, and R. Salakhutdinov, “Dropout: a simple way to prevent neural networks from overfitting.” *Journal of Machine Learning Research*, vol. 15, no. 1, pp. 1929–1958, 2014.

## Appendix

| K | Density | Colorspace | Classifier | Airplane | Car | Face | Bike | mAP |
|---|---|---|---|---|---|---|---|---|
| * 400 | dense | Grayscale | linear | 0.986 | **0.990** | 0.997 | 0.973 | 0.986 |
| 400 | dense | Grayscale | poly | 0.960 | 0.959 | 0.954 | 0.930 | 0.951 |
| 400 | dense | Grayscale | rbf | 0.961 | 0.959 | 0.944 | 0.915 | 0.945 |
| 400 | dense | RGB | linear | 0.997 | 0.985 | 0.988 | 0.983 | 0.988 |
| 400 | dense | RGB | poly | 0.975 | 0.850 | 0.963 | 0.908 | 0.924 |
| 400 | dense | RGB | rbf | 0.975 | 0.850 | 0.963 | 0.870 | 0.914 |
| 400 | dense | HSV | linear | 0.997 | 0.989 | 0.993 | 0.981 | 0.990 |
| 400 | dense | HSV | poly | 0.967 | 0.895 | 0.944 | 0.900 | 0.927 |
| 400 | dense | HSV | rbf | 0.968 | 0.900 | 0.938 | 0.869 | 0.919 |
| 400 | dense | Opp | linear | 0.991 | 0.977 | 0.988 | 0.980 | 0.984 |
| 400 | dense | Opp | poly | 0.969 | 0.946 | 0.948 | 0.917 | 0.945 |
| 400 | dense | Opp | rbf | 0.969 | 0.947 | 0.934 | 0.889 | 0.935 |
| 400 | dense | rgb | linear | 0.997 | 0.985 | 0.988 | 0.983 | 0.988 |
| 400 | dense | rgb | poly | 0.975 | 0.850 | 0.963 | 0.908 | 0.924 |
| 400 | dense | rgb | rbf | 0.975 | 0.850 | 0.963 | 0.870 | 0.914 |
| 400 | key | Grayscale | linear | 0.947 | 0.920 | 0.936 | 0.929 | 0.933 |
| 400 | key | Grayscale | poly | 0.851 | 0.873 | 0.702 | 0.897 | 0.831 |
| 400 | key | Grayscale | rbf | 0.852 | 0.873 | 0.707 | 0.897 | 0.832 |
| 400 | key | RGB | linear | 0.945 | 0.947 | 0.919 | 0.924 | 0.934 |
| 400 | key | RGB | poly | 0.920 | 0.907 | 0.829 | 0.901 | 0.889 |
| 400 | key | RGB | rbf | 0.919 | 0.907 | 0.831 | 0.901 | 0.890 |
| 400 | key | HSV | linear | 0.992 | 0.917 | 0.934 | 0.945 | 0.947 |
| 400 | key | HSV | poly | 0.946 | 0.878 | 0.853 | 0.918 | 0.899 |
| 400 | key | HSV | rbf | 0.946 | 0.877 | 0.858 | 0.918 | 0.900 |
| 400 | key | Opp | linear | 0.970 | 0.899 | 0.954 | 0.955 | 0.945 |

| | | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| 400 | key | Opp | poly | 0.878 | 0.871 | 0.751 | 0.925 | 0.856 |
| 400 | key | Opp | rbf | 0.879 | 0.872 | 0.753 | 0.926 | 0.857 |
| 400 | key | rgb | linear | 0.945 | 0.947 | 0.919 | 0.924 | 0.934 |
| 400 | key | rgb | poly | 0.920 | 0.907 | 0.829 | 0.901 | 0.889 |
| 400 | key | rgb | rbf | 0.919 | 0.907 | 0.831 | 0.901 | 0.890 |
| 800 | dense | Grayscale | linear | 0.988 | 0.979 | 0.993 | 0.977 | 0.984 |
| 800 | dense | Grayscale | poly | 0.959 | 0.959 | 0.969 | 0.911 | 0.950 |
| 800 | dense | Grayscale | rbf | 0.959 | 0.959 | 0.969 | 0.878 | 0.941 |
| 800 | dense | RGB | linear | 0.997 | 0.986 | 0.991 | 0.981 | 0.989 |
| 800 | dense | RGB | poly | 0.977 | 0.958 | 0.942 | 0.855 | 0.933 |
| 800 | dense | RGB | rbf | 0.977 | 0.960 | 0.942 | 0.810 | 0.922 |
| * 800 | dense | HSV | linear | **0.999** | 0.988 | 0.993 | **0.984** | **0.991** |
| 800 | dense | HSV | poly | 0.979 | 0.932 | 0.934 | 0.858 | 0.926 |
| 800 | dense | HSV | rbf | 0.979 | 0.935 | 0.935 | 0.821 | 0.918 |
| 800 | dense | Opp | linear | 0.990 | 0.984 | 0.993 | 0.974 | 0.985 |
| 800 | dense | Opp | poly | 0.970 | 0.961 | 0.946 | 0.863 | 0.935 |
| 800 | dense | Opp | rbf | 0.971 | 0.962 | 0.946 | 0.825 | 0.926 |
| 800 | dense | rgb | linear | 0.997 | 0.986 | 0.991 | 0.981 | 0.989 |
| 800 | dense | rgb | poly | 0.977 | 0.958 | 0.942 | 0.855 | 0.933 |
| 800 | dense | rgb | rbf | 0.977 | 0.960 | 0.942 | 0.810 | 0.922 |
| 800 | key | Grayscale | linear | 0.951 | 0.926 | 0.947 | 0.932 | 0.939 |
| 800 | key | Grayscale | poly | 0.882 | 0.889 | 0.709 | 0.920 | 0.850 |
| 800 | key | Grayscale | rbf | 0.882 | 0.890 | 0.715 | 0.920 | 0.852 |
| 800 | key | RGB | linear | 0.976 | 0.956 | 0.964 | 0.962 | 0.965 |
| 800 | key | RGB | poly | 0.940 | 0.907 | 0.892 | 0.927 | 0.916 |
| 800 | key | RGB | rbf | 0.939 | 0.908 | 0.892 | 0.927 | 0.917 |
| 800 | key | HSV | linear | 0.975 | 0.956 | 0.965 | 0.940 | 0.959 |
| 800 | key | HSV | poly | 0.955 | 0.931 | 0.829 | 0.919 | 0.908 |
| 800 | key | HSV | rbf | 0.955 | 0.932 | 0.836 | 0.921 | 0.911 |
| 800 | key | Opp | linear | 0.968 | 0.944 | 0.966 | 0.944 | 0.955 |
| 800 | key | Opp | poly | 0.898 | 0.904 | 0.755 | 0.935 | 0.873 |
| 800 | key | Opp | rbf | 0.898 | 0.904 | 0.767 | 0.936 | 0.876 |
| 800 | key | rgb | linear | 0.976 | 0.956 | 0.964 | 0.962 | 0.965 |
| 800 | key | rgb | poly | 0.940 | 0.907 | 0.892 | 0.927 | 0.916 |
| 800 | key | rgb | rbf | 0.939 | 0.908 | 0.892 | 0.927 | 0.917 |
| * 1600 | dense | Grayscale | linear | 0.984 | 0.974 | **0.998** | 0.978 | 0.983 |
| 1600 | dense | Grayscale | poly | 0.957 | 0.956 | 0.973 | 0.862 | 0.937 |
| 1600 | dense | Grayscale | rbf | 0.957 | 0.956 | 0.974 | 0.823 | 0.927 |
| 1600 | dense | RGB | linear | 0.997 | 0.982 | 0.992 | 0.975 | 0.986 |
| 1600 | dense | RGB | poly | 0.977 | 0.944 | 0.975 | 0.779 | 0.919 |
| 1600 | dense | RGB | rbf | 0.977 | 0.944 | 0.975 | 0.756 | 0.913 |
| 1600 | dense | HSV | linear | 0.997 | 0.985 | 0.994 | 0.976 | 0.988 |
| 1600 | dense | HSV | poly | 0.977 | 0.956 | 0.963 | 0.780 | 0.919 |
| 1600 | dense | HSV | rbf | 0.977 | 0.956 | 0.963 | 0.755 | 0.913 |
| 1600 | dense | Opp | linear | 0.988 | 0.973 | 0.994 | 0.978 | 0.983 |

| | | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| 1600 | dense | Opp | poly | 0.969 | 0.940 | 0.977 | 0.804 | 0.923 |
| 1600 | dense | Opp | rbf | 0.970 | 0.943 | 0.977 | 0.779 | 0.917 |
| 1600 | dense | rgb | linear | 0.997 | 0.982 | 0.992 | 0.975 | 0.986 |
| 1600 | dense | rgb | poly | 0.977 | 0.944 | 0.975 | 0.779 | 0.919 |
| 1600 | dense | rgb | rbf | 0.977 | 0.944 | 0.975 | 0.756 | 0.913 |
| 1600 | key | Grayscale | linear | 0.945 | 0.930 | 0.959 | 0.931 | 0.941 |
| 1600 | key | Grayscale | poly | 0.904 | 0.890 | 0.740 | 0.926 | 0.865 |
| 1600 | key | Grayscale | rbf | 0.904 | 0.893 | 0.742 | 0.926 | 0.866 |
| 1600 | key | RGB | linear | 0.979 | 0.967 | 0.975 | 0.939 | 0.965 |
| 1600 | key | RGB | poly | 0.955 | 0.921 | 0.930 | 0.939 | 0.936 |
| 1600 | key | RGB | rbf | 0.955 | 0.921 | 0.930 | 0.939 | 0.936 |
| 1600 | key | HSV | linear | 0.986 | 0.979 | 0.976 | 0.938 | 0.970 |
| 1600 | key | HSV | poly | 0.958 | 0.935 | 0.879 | 0.944 | 0.929 |
| 1600 | key | HSV | rbf | 0.958 | 0.936 | 0.884 | 0.945 | 0.931 |
| 1600 | key | Opp | linear | 0.963 | 0.975 | 0.975 | 0.943 | 0.964 |
| 1600 | key | Opp | poly | 0.918 | 0.923 | 0.819 | 0.947 | 0.902 |
| 1600 | key | Opp | rbf | 0.918 | 0.924 | 0.827 | 0.947 | 0.904 |
| 1600 | key | rgb | linear | 0.979 | 0.967 | 0.975 | 0.939 | 0.965 |
| 1600 | key | rgb | poly | 0.955 | 0.921 | 0.930 | 0.939 | 0.936 |
| 1600 | key | rgb | rbf | 0.955 | 0.921 | 0.930 | 0.939 | 0.936 |
| 2000 | dense | Grayscale | linear | 0.991 | 0.986 | 0.992 | 0.971 | 0.985 |
| 2000 | dense | Grayscale | poly | 0.966 | 0.957 | 0.978 | 0.834 | 0.934 |
| 2000 | dense | Grayscale | rbf | 0.966 | 0.958 | 0.978 | 0.802 | 0.926 |
| 2000 | dense | RGB | linear | 0.998 | 0.982 | 0.993 | 0.977 | 0.987 |
| 2000 | dense | RGB | poly | 0.979 | 0.937 | 0.968 | 0.769 | 0.913 |
| 2000 | dense | RGB | rbf | 0.979 | 0.937 | 0.968 | 0.750 | 0.909 |
| 2000 | dense | HSV | linear | 0.998 | 0.984 | 0.994 | 0.979 | 0.989 |
| 2000 | dense | HSV | poly | 0.981 | 0.954 | 0.979 | 0.753 | 0.917 |
| 2000 | dense | HSV | rbf | 0.981 | 0.954 | 0.979 | 0.738 | 0.913 |
| 2000 | dense | Opp | linear | 0.988 | 0.977 | 0.993 | 0.972 | 0.983 |
| 2000 | dense | Opp | poly | 0.967 | 0.933 | 0.974 | 0.791 | 0.916 |
| 2000 | dense | Opp | rbf | 0.967 | 0.935 | 0.974 | 0.768 | 0.911 |
| 2000 | dense | rgb | linear | 0.998 | 0.982 | 0.993 | 0.977 | 0.987 |
| 2000 | dense | rgb | poly | 0.979 | 0.937 | 0.968 | 0.769 | 0.913 |
| 2000 | dense | rgb | rbf | 0.979 | 0.937 | 0.968 | 0.750 | 0.909 |
| 2000 | key | Grayscale | linear | 0.947 | 0.959 | 0.961 | 0.950 | 0.954 |
| 2000 | key | Grayscale | poly | 0.918 | 0.923 | 0.778 | 0.945 | 0.891 |
| 2000 | key | Grayscale | rbf | 0.918 | 0.924 | 0.782 | 0.946 | 0.892 |
| 2000 | key | RGB | linear | 0.991 | 0.973 | 0.982 | 0.950 | 0.974 |
| 2000 | key | RGB | poly | 0.961 | 0.914 | 0.951 | 0.944 | 0.942 |
| 2000 | key | RGB | rbf | 0.961 | 0.914 | 0.953 | 0.944 | 0.943 |
| 2000 | key | HSV | linear | 0.982 | 0.974 | 0.975 | 0.938 | 0.967 |
| 2000 | key | HSV | poly | 0.959 | 0.936 | 0.924 | 0.942 | 0.940 |
| 2000 | key | HSV | rbf | 0.959 | 0.939 | 0.924 | 0.942 | 0.941 |
| 2000 | key | Opp | linear | 0.966 | 0.976 | 0.976 | 0.942 | 0.965 |

| | | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| 2000 | key | Opp | poly | 0.930 | 0.943 | 0.824 | 0.939 | 0.909 |
| 2000 | key | Opp | rbf | 0.930 | 0.943 | 0.825 | 0.939 | 0.909 |
| 2000 | key | rgb | linear | 0.991 | 0.973 | 0.982 | 0.950 | 0.974 |
| 2000 | key | rgb | poly | 0.961 | 0.914 | 0.951 | 0.944 | 0.942 |
| 2000 | key | rgb | rbf | 0.961 | 0.914 | 0.953 | 0.944 | 0.943 |
| 4000 | dense | Grayscale | linear | 0.983 | 0.977 | 0.995 | 0.970 | 0.981 |
| 4000 | dense | Grayscale | poly | 0.952 | 0.968 | 0.992 | 0.797 | 0.927 |
| 4000 | dense | Grayscale | rbf | 0.953 | 0.968 | 0.992 | 0.776 | 0.922 |
| 4000 | dense | RGB | linear | 0.997 | 0.974 | 0.992 | 0.965 | 0.982 |
| 4000 | dense | RGB | poly | 0.978 | 0.943 | 0.980 | 0.739 | 0.910 |
| 4000 | dense | RGB | rbf | 0.978 | 0.943 | 0.981 | 0.730 | 0.908 |
| 4000 | dense | HSV | linear | 0.996 | 0.981 | 0.996 | 0.968 | 0.985 |
| 4000 | dense | HSV | poly | 0.982 | 0.964 | 0.973 | 0.734 | 0.913 |
| 4000 | dense | HSV | rbf | 0.981 | 0.964 | 0.973 | 0.724 | 0.910 |
| 4000 | dense | Opp | linear | 0.992 | 0.974 | 0.996 | 0.979 | 0.986 |
| 4000 | dense | Opp | poly | 0.980 | 0.960 | 0.983 | 0.769 | 0.923 |
| 4000 | dense | Opp | rbf | 0.981 | 0.960 | 0.982 | 0.757 | 0.920 |
| 4000 | dense | rgb | linear | 0.997 | 0.974 | 0.992 | 0.965 | 0.982 |
| 4000 | dense | rgb | poly | 0.978 | 0.943 | 0.980 | 0.739 | 0.910 |
| 4000 | dense | rgb | rbf | 0.978 | 0.943 | 0.981 | 0.730 | 0.908 |
| 4000 | key | Grayscale | linear | 0.960 | 0.976 | 0.946 | 0.949 | 0.958 |
| 4000 | key | Grayscale | poly | 0.938 | 0.949 | 0.767 | 0.944 | 0.900 |
| 4000 | key | Grayscale | rbf | 0.938 | 0.950 | 0.770 | 0.945 | 0.901 |
| 4000 | key | RGB | linear | 0.988 | 0.987 | 0.979 | 0.950 | 0.976 |
| 4000 | key | RGB | poly | 0.973 | 0.950 | 0.962 | 0.966 | 0.963 |
| 4000 | key | RGB | rbf | 0.974 | 0.950 | 0.963 | 0.966 | 0.963 |
| 4000 | key | HSV | linear | 0.988 | 0.984 | 0.985 | 0.945 | 0.975 |
| 4000 | key | HSV | poly | 0.970 | 0.960 | 0.912 | 0.955 | 0.949 |
| 4000 | key | HSV | rbf | 0.970 | 0.961 | 0.916 | 0.955 | 0.950 |
| 4000 | key | Opp | linear | 0.966 | 0.981 | 0.981 | 0.951 | 0.970 |
| 4000 | key | Opp | poly | 0.946 | 0.967 | 0.834 | 0.959 | 0.926 |
| 4000 | key | Opp | rbf | 0.946 | 0.966 | 0.834 | 0.959 | 0.926 |
| 4000 | key | rgb | linear | 0.988 | 0.987 | 0.979 | 0.950 | 0.976 |
| 4000 | key | rgb | poly | 0.973 | 0.950 | 0.962 | 0.966 | 0.963 |
| 4000 | key | rgb | rbf | 0.974 | 0.950 | 0.963 | 0.966 | 0.963 |

Table 4: BoW Results