# Image and Vision Computing Practical

Alun Owen: s2108808, James Ellis: s1728091

March 2021

## 1 Introduction

Classification of images has been at the forefront of image and vision computing. In the past decade, there has been a significant rise in powerful GPU computing. This has allowed for the possibility of extremely deep and powerful models that were once prohibitively expensive to train. For example, the 2012 ImageNet challenge winner, AlexNet [1], has 60 million parameters. More recent prize-winning models have now exceeded 500 million parameters [2, 3, 4]. This power has now been passed to consumers. Most modern-day smartphones can categorise photos based on high-level semantic information.

In this experiment, we train two different classifiers to group an image dataset of cats and dogs. First, we use a deep neural network. In particular, we perform transfer learning on ResNet18 [5]., a pre-trained deep convolutional network. Convolutional neural networks (CNN) are well adapted to the translational invariance of an image. Therefore, they are well suited to the classification task, given that cats or dogs could appear at different locations and angles within the images. In this transfer learning regime, we remove the last layer from ResNet18 and freeze the weights. A fully connected layer is then attached to the end of the truncated network, which performs the final classification, while ResNet18 performs feature extraction. Secondly, we use a support vector machine (SVM) using features from a bag of visual words (BoVW) [6]. SVMs perform a binary linear classification by constructing a hyperplane to separate the data points only using a few points (support vectors) to construct the boundary. It can further be extended to nonlinear classification using the kernel trick. Using raw pixel data from the images as features would be futile given the intra-class variation, i.e. the different locations and angles a cat or dog could appear in a photo. BoVW helps create a compact representation of the image, such that the SVM can classify accurately and efficiently. Drawing analogies from the bag of word representation, BoVW extracts features from the images to create a "visual vocabulary" by k-means clustering such that the frequency of "visual words" can represent images. Finally, we investigate the robustness of each classifier by performing several perturbations on a held-out set of images.

The layout of the article is as follows: In section 2, we outline the methods and perturbations used in this experiment. In section 3, we present and discuss the results. Finally, we end with some concluding remarks in section 4.

## 2 Methods

### 2.1 Dataset

Experimentation was performed on a dataset containing 1188 and 1200 pictures of cats and dogs respectively, each of $224 \times 224$ pixels in size. Some examples can be seen in Figure 1. The vast proportion of the images are colour images following the RGB colour channels. However the data contains a greyscale image; this is either preprocessed or dealt with by functions used. The data was separated into three balanced folds in preparation for cross-validation. During training, 75% of the training folds were used for training, with the remaining 25% used as a validation set.

(a) (b) (c) (d)

Figure 1: Four example images from the dataset used in the experiment

## 2.2 Transfer Learning with ResNet18

ResNet18 is a deep convolutional neural network trained on over 1.2 million images of the ImageNet dataset with 1000 categories. We use this network to perform transfer learning: freeze the weights of a truncated version of ResNet18, training only a final fully connected classification layer with two output neurons. The data used for training was normalised in the same fashion as it was for ResNet18. After the image's pixel values were shifted to the range [0,1], each channel was normalised to the mean and standard deviation of the ImageNet dataset. For R,G, and B channels, the mean values used were 0.485, 0.456, and 0.406 respectively, whilst the standard deviations used were 0.229, 0.224, and 0.225 respectively. All three folds were optimised by a random hyperparameter search (see Sec 2.4). With the optimal parameters, each fold was trained for a maximum of 40 epochs using stochastic gradient descent and the cross-entropy loss function. Early stopping was used during training to mitigate the effects of overfitting. To further help against overfitting, the training data was randomly rotated within the range $[-15°, 15°]$ and flipped horizontally with a 50% probability.

| Fold | Test Accuracy | Early Stop. Epoch |
|---|---|---|
| 1 | 99.25 | 10 |
| 2 | 98.62 | 35 |
| 3 | 99.62 | 11 |

Table 1: Test accuracy for each fold at the end of the training period. The network parameters used were those that maximised the validation accuracy. The epoch in which this occurred is show in the right hand column.

## 2.3 SVM with BoVW

To experiment with time tested machine learning methods, we implemented SVM with BOVW. This classifier will be produced on Matlab with inspiration from relevant documentation [7]. The code for implementing all operations with the SVM are contained in Section A.2. The BOVW is produced with a single line `bagOfFeatures(...)`[1]; within here, we can set out different hyperparameters for the function. SURF is chosen as both descriptor and detector since it performs its operations quicker than the common alternate, scale-invariant feature transform (SIFT) [8]. Uniquely the method can produce features that are invariant to both scale and rotation [9]. The SVM is trained by running the `trainImageCategoryClassifier(...)`[2], the options that formulate the type of SVM is done with `templateSVM(...)`[3].

[1] https://uk.mathworks.com/help/vision/ref/bagoffeatures.html
[2] https://uk.mathworks.com/help/vision/ref/trainimagecategoryclassifier.html
[3] https://uk.mathworks.com/help/stats/templatesvm.html

## 2.4 Hyperparameter Search

Prior to training each model, we employed a random hyperparameter search for each of the three folds in our cross-validation models. For each model, 200 combinations of hyperparameters were randomly selected with each fold of the model being tested on the same set. The objective was to maximise the validation accuracy. For ResNet18, we maximised the validation accuracy within 10 epochs.

For training the SVM with BoVW, the following parameters were optimised:

- The number of *visual words* was chosen randomly from {1000, 2000, 3000, 4000, 5000, 6000, 7000, 8000, 9000, 10000}
- The *proportion* of strongest features used was chosen randomly from {0.6, 0.65, 0.7, 0.75, 0.8, 0.85, 0.9, 0.95}
- The feature extraction algorithm was chosen as either SURF or U-SURF
- The type of *kernel* was chosen to be either Gaussian, linear, or polynomial
- If polynomial was randomly chosen, the *degree* of the polynomial was randomly chosen from {2, 3, 4, 5}

The hyperparameter search results can be seen in full within the appendix in Section B.2. Between the three folds, there was very little consensus of any optimum hyperparameter choice. The only hyperparameter to be equivalent for the three fold was with the usage of U-SURF. Each of the different folds achieved their optimum with a different kernel type for the SVM. With the different kernel, a complete variety for the number of visual words was deemed optimum. They ranged from 4000 visual words for one fold and 9000 for another. Both the optimum for the second and third fold were in agreement for a 80% proportion of features to use for the BOVW. Regarding the optimum of the second fold. It used a polynomial kernel with a degree of 3. The choices of hyperparameters used for each fold is labelled in the final column of the table. With the best configurations set for each fold, the models are then trained again on both the training and validation for each fold before engaging in robustness testing with the test sets.

For training the fully connected layer, the following parameters were optimised:

- The *learning rate* used during gradient descent was randomly drawn from a logarithmic uniform distribution within the range $[10^{-4}, 10^{-1}]$.
- The optimiser's *momentum* factor was randomly chosen from {0.5, 0.9, 0.99} as these are common values used in research [10]
- The *batch size* used during training was randomly chosen from {4,8,16,32}.
- The learning rate decay *step size* was randomly chosen from {3,5,7}
- The learning rate decay multiplicative factor (*gamma*) was randomly drawn from a uniform distribution in the range [0.1,0.5]

Given the similarities of the task and high performance of the source code (see Appendix A.1), the ranges for the hyperparameter search were constrained.

For each optimum configuration of hyperparameters for each fold, we tested the performance against a test set which has been augmented with several different perturbations. The perturbations were applied with increasing intensity to investigate how robust the classifiers are with different levels of distortion.

## 2.5 Perturbations

### Gaussian Noise

To apply the Gaussian noise to an image, a $H \times W \times C$ matrix $X$ is sampled from the Gaussian distribution $X \sim \mathcal{N}(0_{H,W,C}, \sigma^2 I_{H,W,C})$ where $H \times W$ denotes the dimension of the image and $C$ signifies the number of channels (i.e. the red, green and blue channels of the image). The standard deviation of the Gaussian distribution, $\sigma$, was varied within the set

$$\sigma_{noise} \in \{0, 2, 4, 6, 8, 10, 12, 14, 16, 18\}$$

This provides a control on the intensity of the pixel noise. The sampled matrix is simply added to the image matrix. A comparison of $\sigma = 0$ and $\sigma = 18$ is seen in the top-left of Figure 2.

### Gaussian Blur

To perform a Gaussian blurring on the test images, an approximate discrete Gaussian kernel, $K$, was convolved over each channel of images

$$K = \frac{1}{16}\begin{bmatrix} 1 & 2 & 1 \\ 2 & 4 & 2 \\ 1 & 2 & 1 \end{bmatrix}$$

To approximate Gaussian blurring with increasingly larger standard deviations, the kernel was applied $n$ times to the test images where $n \in [0..9]$. Convolving the kernel over the images will take a weighted mean of the pixel intensities in the neighbourhood. This will result in a blurring effect. To keep the transformed image the same size as the test image, the convolution is applied with padding. An example of the Gaussian blurring with 9 convolutions can be see in the top-middle of Figure 2.

### Image Contrast

The contrast of the image was be varied to test how robust the classifiers are to varying amounts of spread of the distribution in each channel. To emulate contrast change, every pixel will be scalar multiplied by a factor. This will either produce a wider/narrower range of colors if the scalar is larger/smaller than one. Experimentation will be done for both contrast increase and decrease. For contrast increase, the scalars, $c$, are defined

$$c \in \{1, 1.03, 1.06, 1.09, 1.12, 1.15, 1.18, 1.21, 1.24, 1.27\}$$

where as for contrast decrease are

$$c \in \{1, 0.9, 0.8, 0.7, 0.6, 0.5, 0.4, 0.3, 0.2, 0.1\}$$

An example of highest and lowest intensity contrast increase is seen in the top-right of Figure 2. Similarly, the middle-left of Figure 2 demonstrates contrast decrease.

### Image Brightness

Similarly to changing the contrast, brightness also affects the histogram of pixel intensities in an image for each of the channels but only by a shift in the distribution. To simulate this, a constant scalar, $b$, will be added or subtracted to each pixel of each channel in the image

$$b \in \{0, 5, 10, 15, 20, 25, 30, 35, 40, 45\} \text{ px}$$

For increasing and decreasing the brightness, we added and subtracted the scalar, $b$, respectively. Examples of both are shown in the middle and middle right of Figure 2.

### Hue Noise

Hue noise refers to noise in the Hue channel of an HSV image. Therefore, we first convert the RGB images into HSV format. Once the image is converted, the method is similar to Gaussian noise, expect only adding the noise matrix to the hue component of the image. Assuming the values of the hue component lie within the range $[0, 1]$, the standard deviation of the Gaussian were defined as

$$\sigma_{hue} \in \{0, 0.02, 0.04, 0.06, 0.08, 0.10, 0.12, 0.14, 0.16, 0.18\}$$

The difference between no variation and maximal variation in the hue noise is displayed in bottom-left of Figure 2.

### Saturation Noise

In a similar manner, we follow the same method in hue noise, except adding the noise matrix to the saturation component of the image. Again, we assume the values of the saturation component lie within the range $[0, 1]$. We define the values of $\sigma$ the same as for hue noise

$$\sigma_{sat} \in \{0, 0.02, 0.04, 0.06, 0.08, 0.10, 0.12, 0.14, 0.16, 0.18\}$$

An example of saturation noise for an image is in the bottom-middle of Figure 2. For testing, both hue and saturation perturbation test sets are transformed back to RGB format for classification.

### Occlusion

Finally, we tested the classifiers robustness with simulated image occlusion. To test occlusion, the images were augmented to include a black $r \times r$ square. The square is place randomly in the image and such that the entirety of the box lies within the image boundaries. The boxes were varied in size, with $r$ defined as

$$r \in \{0, 5, 10, 15, 20, 25, 30, 35, 40, 45\} \text{ px}$$

An example of an image with occlusion , refer to the bottom right pair of images in Figure 2.

---

All code for the robustness experiments is summarised in the Appendix (Sec A.1 & A.2: Figures 4 - 20).

## 3 Results & Discussion

Overall, there is a significant performance difference between the two models. On the unperturbed test set, the ResNet model achieves an average test accuracy of $(99.2 \pm 0.4)\%$ over the three folds, while the SVM with BOVW achieves $(80.7 \pm 0.7)\%$. However, this is not unexpected. ResNet18 has the advantage of having on the order of $10^7$ parameters, as well as being pre-trained on millions of high resolution images. On the other hand, SVM with BoVW relies on a hand full of vectors to delineate the two classes. In the rest of this section, we present the results from the perturbations. The graphs for each perturbation discussed in Sec. 2.5 is respectively shown in a sinistrodextral fashion in Figure 2.

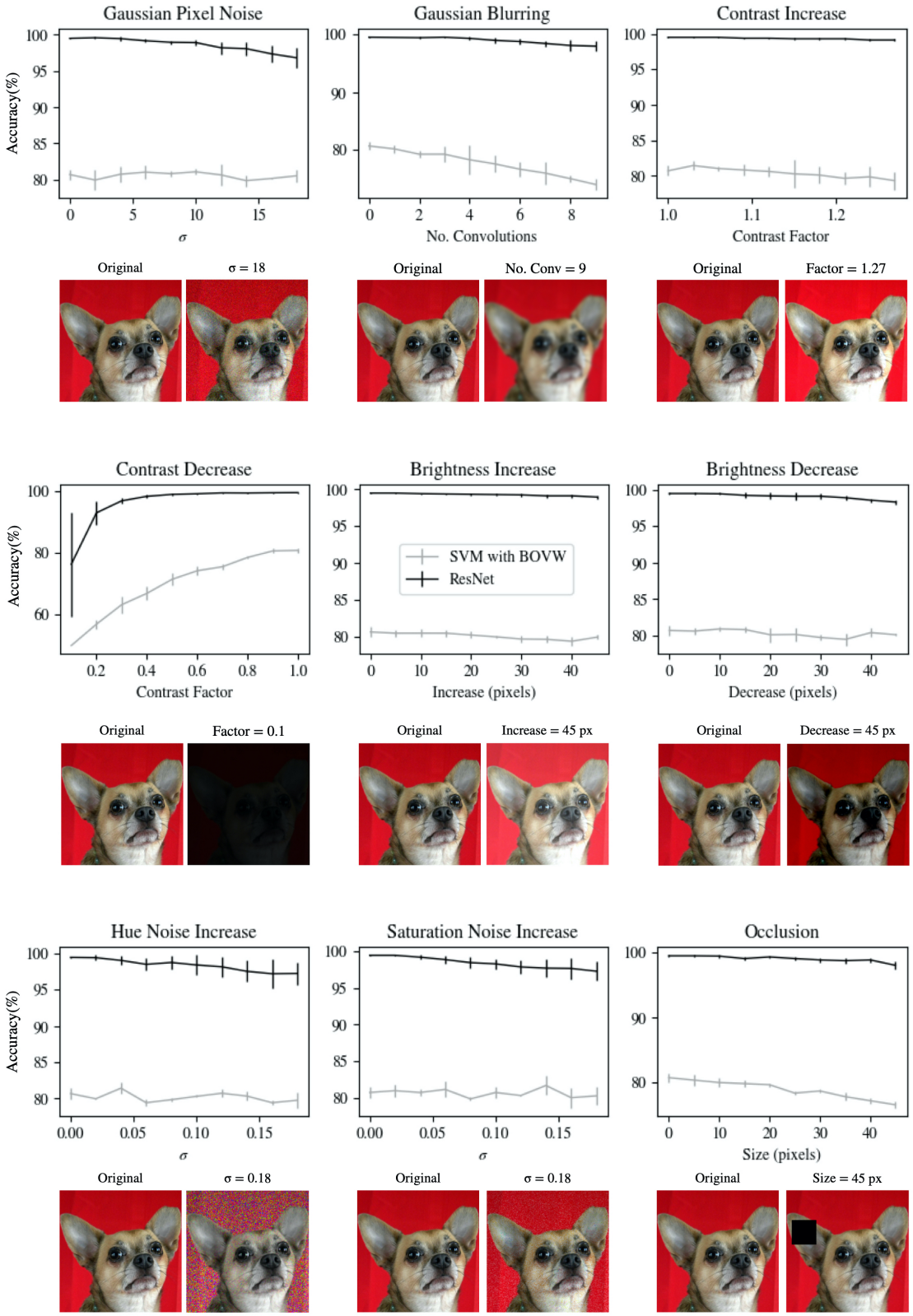


Figure 2: Results for robustness experiments

### 3.1 Gaussian Noise

As seen in the top left of Figure 2, the ResNet18 model suffers a steady decline in test accuracy with increasing levels of perturbation. Whereas the test accuracy of SVM with BoVW is approximately constant. The robustness of SVM with BoVW is due to the Gaussian kernel convolved with the image during the extraction of keypoints. This kernel will diminish mild random noise since it takes a weighted average of its neighbourhood. On the other hand, the ResNet18 model suffers due to small receptive fields ($7 \times 7$ and $3 \times 3$) in the convolutional layers. With increasing noise, more dissimilarity will be seen in the features of the images and the features the convolutional layers have been trained to search for. This will affect cases near the boundary because their classification will be sensitive to changes in activation values.

### 3.2 Gaussian Blur

Both ResNet and SVM with BoVW suffer a decline in test accuracy with increasing levels of Gaussian blur perturbation. The results can be seen in the top middle of Figure 2. As the images become more blurred, edges and corners become less pronounced making it more difficult for the classifiers to identify key points and features. This is the main cause for the premature drop in accuracy by SVM with BoVW; it detects a large proportion of key points which will be dampened due to the blurring. Unlike SVM with BoWW, ResNet18 knows what features it is looking for. Blurring may decrease the activation of these features, but on the whole they will mostly be found. The decrease in activation values could be enough to change test images already near the decision boundary, hence explaining the slight decrease in test accuracy observed.

### 3.3 Image Contrast

The results for increasing and decreasing image contrast can be seen in the top right and middle left of Figure 2 respectively. Increasing the image contrast has little affect on Resnet18 model, with near constant levels of accuracy and variance maintained with increasing levels of perturbation. Since increasing contrast creates a wider spread in each channels distribution, features can become more prominent. Similarly with the SVM, increasing the contrast will also scale the derivatives, this will be proportionate to each derivative and therefore should not produce irregularities when detecting using the second derivatives from the Hessian matrix in SURF. However, this is a very simplistic model of increasing constrast by multiplying by a scalar. The method achieves an increased spread in each channel distribution; however, features near saturation could be lost. For example, image a white dog (saturated pixels) standing by a cream wall (nearly saturated pixels). In this case, our method of contrast increase would remove important features, which is undesirable.

The decrease in performance due to contrast decrease is significant for both classifiers, but it should be noted that the magnitude of contrast decrease is much greater than that of contrast increase. Initially, Resnet18 performs steadily then experiences a sharp drop in accuracy at the largest decreases in contrast. This is likely to be due to numerous reasons. If the equivalent factors of pertubations are compared, the Resnet18 does extremely similar to the contrast increase. Our implementation of contrast adjustment (as instructed in specification) also adjusts the brightness. Therefore such a large scaling performs both strong contrast decrease and brightness decrease. This scaling is also performed on a test set which has 72% of images have median pixel intensity on the darker side of the spectrum pre-perturbation. Therefore removing color in such a way will significantly lose information related to features as seen in the example image. For similar reasoning, the SVM with BOVW also diminishes in performance but on a steadier trajectory. This is due to the contrast adjustment affecting the gradients which fundamentally affects keypoint detection for the BoVW.

### 3.4 Image Brightness

The results for increasing and decreasing image brightness can be seen in the middle and middle right of Figure 2 respectively. The ResNet18 model shows a slight negative trend with increasing levels of perturba-

tion for brightness increase and decrease, with the brightness decrease causing a larger accuracy degradation. This can be attributed to the dataset; the majority of the images' pixel values are closer to 0 than saturation. In the entire dataset, 28% of the images have a median pixel intensity of greater than 127.5. Given that pixel values are capped within the range [0..255], there will be more information loss from a brightness decrease than increase. SVM with BoVW remains largely unaffected in both cases because the information loss due to brightness increase or decrease is insufficient to remove the difference in pixel intensities from keypoints such as edges and corners.

### 3.5 HSV Hue Noise Increase

The results for increasing the amount of hue noise in the test images can be seen in the bottom left of Figure 2. We observe a relatively consistent performance from SVM with BoVW. Hue noise is typically only added to 'bright' colours such as a red background, or bright blue sky. More dull colours such as brown fur have very little noise added to them. Therefore, SURF is able to still pick up key points of the image especially since the extraction of keypoints derives from a Gaussian kernel convolved with the image which eradicates some noise. On the other hand, the ResNet18 model sees a similar pattern to what was observed in Gaussian noise, with similar reasoning in this case.

### 3.6 HSV Saturation Noise Increase

The results for increasing the amount of saturation noise can be seen in the bottom middle of Figure 2. Similarly to the other noise related perturbations, saturation noise has no effect on the SVM with BOVW. Although saturation noise appears to disproportionately affect white fur for example, the Gaussian kernel applied in SURF alleviates these effects. Similarly to hue noise, the ResNet18 model again sees a slight decrease in test accuracy with increasing perturbation.

### 3.7 Image Occlusion

The results for occlusion perturbations can be seen in the bottom right of Figure 2. The SVM with BoVW model accuracy decreases with increasing size of occlusion. This is because SURF is likely identifying key points from the square shaped occlusion which are then aggregated to visual words by nearest neighbour. This can ultimately change the histogram of visual words for the image. Changing the histogram of visual words can drastically change the interpretation of the image for the classifier by introducing randomly placed occlusions as part of the descriptor for the image. On the other hand, the ResNet18 model accuracy only slightly decreases with increasing size of occlusion. This is because feature maps won't respond to the occlusion because the convolutional layers won't respond to the occlusion. The slight decrease will merely be due to a small loss of information.

## 4 Conclusion

In this experiment, we have successfully trained and evaluated the robustness of two popular, but very different, classifiers: a transfer learning model with ResNet18, and an SVM model using BoVW as features for the classifier. As to be expected, we observed our transfer learning model to outperform SVM with BoVW due to the massive parameter and pre-training advantage that ResNet18 has. Our robustness experiments showed that, on the whole, the ResNet18 model is less robust than the SVM. We believe this is due to the pre-training of the convolutional layers. Each convolutional layer is "looking" for a specific feature in the image, and any perturbation will affect the activation values of that convolutional layer. The SVM with BoVW model can efficiently handle noise related perturbations since the descriptor applies a Gaussian kernel in the process of detecting keypoints. However, the model was least robust to Gaussian blurring and occlusion whereas the ResNet18 model was most robust to these perturbation. However, it should be noted that it is not an entirely fair test. Images of blurring and occlusion can easily be found in the ImageNet

dataset meaning that ResNet18 has an intrinsic pre-trained advantage for these perturbations. An example can be seen in Figure 3

Figure 3: Blurry example image of a dog from the ImageNet dataset

Although SVM with BoVW is more robust in our experiments, it is natural to ask which would be more robust in a real world situation. Perturbations such as blurring and occlusion are very common in real life. For example, an unfocused image or covering the lens slightly with the tip of a finger. It would be unlikely to see perturbations such as pixel noise independently unless paired with a low light situation [4]. If such scenarios of classification tasks are to be performed on the fly as they are in some smartphones, a cost benefit analysis should be performed between performance, robustness, storage space and computational power needed.

# References


[1] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. Imagenet classification with deep convolutional neural networks, 2012.

[2] Pierre Foret, Ariel Kleiner, Hossein Mobahi, and Behnam Neyshabur. Sharpness-aware minimization for efficiently improving generalization, 2021.

[3] Andrew Brock, Soham De, Samuel L Smith, and Karen Simonyan. High-performance large-scale image recognition without normalization, 2021.

[4] Hieu Pham, Zihang Dai, Qizhe Xie, Minh-Thang Luong, and Quoc V Le. Meta pseudo labels, 2021.

[5] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition, 2015.

[6] Jun Yang, Yu-Gang Jiang, Alexander G Hauptmann, and Chong-Wah Ngo. Evaluating bag-of-visual-words representations in scene classification. In *Proceedings of the international workshop on Workshop on multimedia information retrieval*, pages 197–206, 2007.

[7] Image classification with bag of visual words. https://uk.mathworks.com/help/vision/ug/image-classification-with-bag-of-visual-words.html. Accessed: 2021-03-01.

[8] Luo Juan and Oubong Gwun. A comparison of sift, pca-sift and surf. *International Journal of Image Processing (IJIP)*, 3(4):143–152, 2009.

[9] Herbert Bay, Andreas Ess, Tinne Tuytelaars, and Luc Van Gool. Speeded-up robust features (surf). *Computer vision and image understanding*, 110(3):346–359, 2008.


[4] https://en.wikipedia.org/wiki/Signal-to-noise_ratio

[10] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning*. MIT Press, 2016. http://www.deeplearningbook.org.

# A Perturbation Experiments Code

## A.1 ResNet18: Python

The code used follows a similar structure found in https://pytorch.org/tutorials/beginner/transfer_learning_tutorial.html.. Pytorch supports easy data-augmentation by providing a transform list to apply to the images. Functions used in section 4 can easily be called within this list:

```
transform_list = [
    transforms.ToTensor(),
    # Placeholder function for illustration
    Custom_Transformation_Here(perturbation=6),
    transforms.Normalize([0.485, 0.456, 0.406], [0.229, 0.224, 0.225]),
    ]
```

Figure 4: List of transforms used during inference (python)

Most functions required for section 4 required custom functions. Furthermore, all of the following sections are working with tensors in the range 0-1.

```
class AddGaussianNoise(object):
    def __init__(self, mean=0., std=1.):
        self.std = std
        self.mean = mean

    def __call__(self, tensor):
        return tensor + 1/255 * torch.randn(tensor.size()) * self.std + self.mean

    def __repr__(self):
        return self.__class__.__name__ + '(mean={0}, std={1})'.format(self.mean, self.std)
```

Figure 5: Custom gaussian noise class (python)

```
for j in range(10):

    transform_list = [
        transforms.ToTensor(),
    ]

    for k in range(j):
        # Produces the same 3x3 kernel specified in the coursework handout
        transform_list.append(transforms.GaussianBlur(3, 0.7955))

    transform_list.append(
        transforms.Normalize([0.485, 0.456, 0.406], [0.229, 0.224, 0.225])
    )
```

Figure 6: Transform list for gaussian blur (python)

```python
class AdjustContrast(object):
    def __init__(self, amount=0):
        self.amount = amount

    def __call__(self, img):
        img = img * self.amount

        return torch.clamp(img, min=0, max=1)

    def __repr__(self):
        return self.__class__.__name__ + '(amount={0})'.format(self.amount)
```

Figure 7: Transform function for increasing and decreasing the contrast (python)

```python
class AdjustBrightness(object):
    def __init__(self, amount=0):
        self.amount = amount

    def __call__(self, img):
        img = img + self.amount/255

        return torch.clamp(img, min=0, max=1)

    def __repr__(self):
        return self.__class__.__name__ + '(amount={0})'.format(self.amount)
```

Figure 8: Custom function for adjusting the brightness (python)

```python
from kornia.color import rgb_to_hsv, hsv_to_rgb

class AdjustHue(object):
    def __init__(self, std=0):
        self.std = std

    def __call__(self, img):
        img = rgb_to_hsv(img)

        noise = 2*np.pi * torch.randn(img[0].size()) * self.std
        hue = img[0]
        hue += noise

        #Hue component in range 0 - 2pi
        hue[hue>1] -= 2 * np.pi
        hue[hue<1] += 2 * np.pi

        img = hsv_to_rgb(img)
        return img

    def __repr__(self):
        return self.__class__.__name__ + '(std={0})'.format(self.std)
```

Figure 9: Custom function for increasing the hue of an RGB image (python)

```python
from kornia.color import rgb_to_hsv, hsv_to_rgb

class AdjustSaturation(object):
    def __init__(self, std=0):
        self.std = std

    def __call__(self, img):
        img = rgb_to_hsv(img)

        noise =  torch.randn(img[0].size()) * self.std
        img[1] += noise

        img[1] = torch.clamp(img[1], min=0,max=1)
        img = hsv_to_rgb(img)
        return img

    def __repr__(self):
        return self.__class__.__name__ + '(std={0})'.format(self.std)
```

Figure 10: Custom function for adjusting saturation of RGB image (python)

```python
class CutOut(object):
    def __init__(self, length=0):
        self.length = length

    def __call__(self, img):
        img_length = img.size(1)
        max_index = img_length - self.length

        start_x = int(np.round(np.random.uniform(low=0, high=max_index, size=1)))
        start_y = int(np.round(np.random.uniform(low=0, high=max_index, size=1)))

        img[:, start_x:start_x + self.length, start_y:start_y+self.length] = 0

        return img

    def __repr__(self):
        return self.__class__.__name__ + '(length={0})'.format(self.length)
```

Figure 11: Custom function for occlusion increase (python)

## A.2 SVM with BoVW: Matlab

```
imds = imageDatastore('catdog','IncludeSubfolders',true,'LabelSource','foldernames');


partitions = cvpartition(imds.Labels, 'Kfold', 3);
imd_combine_train1 = subset(imds, training(partitions,1));
[imd_train1, imd_val1] = splitEachLabel(imd_combine_train1,3/4,'randomize');
imd_test1 = subset(imds, test(partitions,1));
imd_combine_train2 = subset(imds, training(partitions,2));
[imd_train2, imd_val2] = splitEachLabel(imd_combine_train2,3/4,'randomize');
imd_test2 = subset(imds, test(partitions,2));
imd_combine_train3 = subset(imds, training(partitions,3));
[imd_train3, imd_val3] = splitEachLabel(imd_combine_train3,3/4,'randomize');
```

Figure 12: Code for splitting the data with 3 fold cross validation on matlab

```
load('imd_train.mat');
load('imd_val.mat');
load('imd_test.mat');
results = {};
results_configurations = {};
parameter1 = {1000,2000,3000,4000,5000,6000,7000,8000,9000,10000};
parameter2 = {0.6,0.65,0.7,0.75,0.8,0.85,0.9,0.95};
parameter3 = {true,false};
parameter4 = {'gaussian', 'linear', 'polynomial'};
parameter5 = {2,3,4,5};
for i = 1:200
    first = randi([1,10],1);
    second = randi([1,8],1);
    third = randi([1,2],1);
    fourth = randi([1,3],1);
    fifth = randi([1,4],1);
    if strcmp(parameter4{fourth}, 'polynomial')
        results_configurations{end + 1} = {parameter1{first}, parameter2{second}, ...
                                           parameter3{third}, parameter4{fourth}, ...
                                           parameter5{fifth}};
    else
        results_configurations{end + 1} = {parameter1{first}, parameter2{second}, ...
                                           parameter3{third}, parameter4{fourth}, 'N/A'};
    end
    if strcmp(parameter4{fourth}, 'polynomial')
        option = templateSVM('KernelFunction', parameter4{fourth}, ...
                             'PolynomialOrder', parameter5{fifth});
    else
        option = templateSVM('KernelFunction', parameter4{fourth});
    end
    bag_features1 = bagOfFeatures(imd_train1,'VocabularySize', parameter1{first} , ...
                                  'StrongestFeatures', parameter2{second}, ...
                                  'PointSelection', 'Detector', ...
                                  'Upright', parameter3{third} );
    bag_features2 = bagOfFeatures(imd_train2,'VocabularySize', parameter1{first} , ...
                                  'StrongestFeatures', parameter2{second}, ...
                                  'PointSelection', 'Detector', ...
                                  'Upright', parameter3{third} );
    bag_features3 = bagOfFeatures(imd_train3,'VocabularySize', parameter1{first} , ...
                                  'StrongestFeatures', parameter2{second}, ...
                                  'PointSelection', 'Detector', ...
                                  'Upright', parameter3{third} );
    classifier1 = trainImageCategoryClassifier(imd_train1, bag_features1, ...
                                               'LearnerOptions', option);
    classifier2 = trainImageCategoryClassifier(imd_train2, bag_features2, ...
                                               'LearnerOptions', option);
    classifier3 = trainImageCategoryClassifier(imd_train3, bag_features3, ...
                                               'LearnerOptions', option);
    results_val1 = evaluate(classifier1, imd_val1);
    results_val2 = evaluate(classifier2, imd_val2);
    results_val3 = evaluate(classifier3, imd_val3);
    results{end + 1} = {average_acc(results_val1), average_acc(results_val2), ...
                        average_acc(results_val3), (average_acc(results_val1) + ...
                        average_acc(results_val2) + average_acc(results_val3))/3};
end

function [y] = average_acc(x)
    y = (x(1,1)+x(2,2))/2;
end
```



Figure 13: This process was the random search carried out on the all three folds.

```
option = templateSVM('KernelFunction', 'gaussian');
bag_features3 = bagOfFeatures(imd_combine_train1,'VocabularySize', 4000 , ...
                              'StrongestFeatures', 0.6, ...
                              'PointSelection', 'Detector', ...
                              'Upright', true );
classifier1 = trainImageCategoryClassifier(imd_combine_train1, bag_features3, ...
                                           'LearnerOptions', option);
```

Figure 14: Here the classifier for a specific fold is trained over both training and validation set with hyperparameters set to the optimum according to the hyperparameter search. Example shown is for the first fold.

```
noise = [];
pics = readall(imd_test1);
for j=0:2:18
confusion_matrix = [0,0;0,0];
for i=1:796
   pic = cell2mat(pics(i));
   x = j * randn(size(pic));
   x = uint8(x);
   pic = pic + x;
   pred = predict(classifier1,pic);
   if pred == 1 & imd_test1.Labels(i) == 'CATS'
       confusion_matrix(1,1) = confusion_matrix(1,1) + 1;
   elseif pred == 1 & imd_test1.Labels(i) == 'DOGS'
       confusion_matrix(2,1) = confusion_matrix(2,1) + 1;
   elseif pred == 2 & imd_test1.Labels(i) == 'CATS'
       confusion_matrix(1,2) = confusion_matrix(1,2) + 1;
   elseif pred == 2 & imd_test1.Labels(i) == 'DOGS'
       confusion_matrix(2,2) = confusion_matrix(2,2) + 1;
   end
end
noise(end + 1) = (confusion_matrix(1,1)+confusion_matrix(2,2))/796;
end


%% Gaussian Blur
blur = [];
mask = [1,2,1;2,4,2;1,2,1]/16;

pics = readall(imd_test1);
for j=0:9
confusion_matrix = [0,0;0,0];
for i=1:796
   pic = cell2mat(pics(i));
   if length(size(pic)) == 2
       pic = pic(:,:,[1 1 1]);
   end
   k = 0;
   while k<j
       pic(:,:,1) = conv2(pic(:,:,1),mask,'same');
       pic(:,:,2) = conv2(pic(:,:,2),mask,'same');
       pic(:,:,3) = conv2(pic(:,:,3),mask,'same');
       k = k + 1;
   end
   pred = predict(classifier1,pic);
   if pred == 1 & imd_test1.Labels(i) == 'CATS'
       confusion_matrix(1,1) = confusion_matrix(1,1) + 1;
```

```
%% Contrast Increase
cont_incr = [];
pics = readall(imd_test1);
for j=1:0.03:1.27
confusion_matrix = [0,0;0,0];
for i=1:796
   pic = cell2mat(pics(i));
   if length(size(pic)) == 2
       pic = pic(:,:,[1 1 1]);
   end
   x = j.*ones(size(pic));
   pic = double(pic);
   pic = x.*pic;
   pic = uint8(pic);
   pred = predict(classifier1,pic);
   if pred == 1 & imd_test1.Labels(i) == 'CATS'
       confusion_matrix(1,1) = confusion_matrix(1,1) + 1;
   elseif pred == 1 & imd_test1.Labels(i) == 'DOGS'
       confusion_matrix(2,1) = confusion_matrix(2,1) + 1;
   elseif pred == 2 & imd_test1.Labels(i) == 'CATS'
       confusion_matrix(1,2) = confusion_matrix(1,2) + 1;
   elseif pred == 2 & imd_test1.Labels(i) == 'DOGS'
       confusion_matrix(2,2) = confusion_matrix(2,2) + 1;
   end
end
cont_incr(end + 1) = (confusion_matrix(1,1)+confusion_matrix(2,2))/796;
end


%% Contrast Decrease
cont_decr = [];
pics = readall(imd_test1);
for j=1:-0.1:0.1
confusion_matrix = [0,0;0,0];
for i=1:796
   pic = cell2mat(pics(i));
   if length(size(pic)) == 2
       pic = pic(:,:,[1 1 1]);
   end
   x = j.*ones(size(pic));
   pic = double(pic);
   pic = x.*pic;
   pic = uint8(pic);
   pred = predict(classifier1,pic);
   if pred == 1 & imd_test1.Labels(i) == 'CATS'
       confusion_matrix(1,1) = confusion_matrix(1,1) + 1;
   elseif pred == 1 & imd_test1.Labels(i) == 'DOGS'
       confusion_matrix(2,1) = confusion_matrix(2,1) + 1;
   elseif pred == 2 & imd_test1.Labels(i) == 'CATS'
       confusion_matrix(1,2) = confusion_matrix(1,2) + 1;
   elseif pred == 2 & imd_test1.Labels(i) == 'DOGS'
       confusion_matrix(2,2) = confusion_matrix(2,2) + 1;
   end
end
cont_decr(end + 1) = (confusion_matrix(1,1)+confusion_matrix(2,2))/796;
end
```



Figure 16: Contrast change on test set using SVM with BOVW (matlab)

```
%% Brightness Increase
bri_incr = [];
pics = readall(imd_test1);
for j=0:5:45
confusion_matrix = [0,0;0,0];
for i=1:796
   pic = cell2mat(pics(i));
   if length(size(pic)) == 2
       pic = pic(:,:,[1 1 1]);
   end
   x = j.* ones(size(pic));
   pic = double(pic);
   pic = pic + x;
   pic = uint8(pic);
   pred = predict(classifier1,pic);
   if pred == 1 & imd_test1.Labels(i) == 'CATS'
       confusion_matrix(1,1) = confusion_matrix(1,1) + 1;
   elseif pred == 1 & imd_test1.Labels(i) == 'DOGS'
       confusion_matrix(2,1) = confusion_matrix(2,1) + 1;
   elseif pred == 2 & imd_test1.Labels(i) == 'CATS'
       confusion_matrix(1,2) = confusion_matrix(1,2) + 1;
   elseif pred == 2 & imd_test1.Labels(i) == 'DOGS'
       confusion_matrix(2,2) = confusion_matrix(2,2) + 1;
   end
end
bri_incr(end + 1) = (confusion_matrix(1,1)+confusion_matrix(2,2))/796;
end


%% Brightness Decrease
bri_decr = [];
pics = readall(imd_test1);
for j=0:-5:-45
confusion_matrix = [0,0;0,0];
for i=1:796
   pic = cell2mat(pics(i));
   x = j.* ones(size(pic));
   pic = double(pic);
   pic = pic + x;
   pic = uint8(pic);
   pred = predict(classifier1,pic);
   if pred == 1 & imd_test1.Labels(i) == 'CATS'
       confusion_matrix(1,1) = confusion_matrix(1,1) + 1;
   elseif pred == 1 & imd_test1.Labels(i) == 'DOGS'
       confusion_matrix(2,1) = confusion_matrix(2,1) + 1;
   elseif pred == 2 & imd_test1.Labels(i) == 'CATS'
       confusion_matrix(1,2) = confusion_matrix(1,2) + 1;
   elseif pred == 2 & imd_test1.Labels(i) == 'DOGS'
       confusion_matrix(2,2) = confusion_matrix(2,2) + 1;
   end
end
bri_decr(end + 1) = (confusion_matrix(1,1)+confusion_matrix(2,2))/796;
end
```



Figure 17: Brightness changes on the test data for SVM with BOVW (matlab)

```
%% RGB to HSV Hue
hue = [];
pics = readall(imd_test1);
for j=0:0.02:0.18
confusion_matrix = [0,0;0,0];
for i=699:699
   pic = cell2mat(pics(i));
   if length(size(pic)) == 2
       pic = pic(:,:,[1 1 1]);
   end
   pic = rgb2hsv(pic);
   x = j.* randn(size(pic));
   x = x(:,:,1);
   pic(:,:,1) = pic(:,:,1) + x;
   for x=1:224
       for y=1:224
           if pic(y,x,1)>1
               pic(y,x,1) = pic(y,x,1) - 1;
           end
           if pic(y,x,1)<0
               pic(y,x,1) = pic(y,x,1) + 1;
           end
       end
   end
   pic = hsv2rgb(pic);
   pred = predict(classifier1,pic);
   if pred == 1 & imd_test1.Labels(i) == 'CATS'
       confusion_matrix(1,1) = confusion_matrix(1,1) + 1;
   elseif pred == 1 & imd_test1.Labels(i) == 'DOGS'
       confusion_matrix(2,1) = confusion_matrix(2,1) + 1;
   elseif pred == 2 & imd_test1.Labels(i) == 'CATS'
       confusion_matrix(1,2) = confusion_matrix(1,2) + 1;
   elseif pred == 2 & imd_test1.Labels(i) == 'DOGS'
       confusion_matrix(2,2) = confusion_matrix(2,2) + 1;
   end
end
hue(end + 1) = (confusion_matrix(1,1)+confusion_matrix(2,2))/796;
end
```

Figure 18: Hue noise on test set using SVM with BOVW (matlab)

```
%% RGB to HSV Saturation
sat = [];
pics = readall(imd_test1);
for j=0:0.02:0.18
confusion_matrix = [0,0;0,0];
for i=1:796
   pic = cell2mat(pics(i));
   if length(size(pic)) == 2
       pic = pic(:,:,[1 1 1]);
   end
   pic = rgb2hsv(pic);
   x = j.* randn(size(pic));
   x = x(:,:,2);
   pic(:,:,2) = pic(:,:,2) + x;
   for x=1:224
       for y=1:224
           if pic(y,x,2)>1
               pic(y,x,2) = 1;
           end
           if pic(y,x,2)<0
               pic(y,x,2) = 0;
           end
       end
   end
   pic = hsv2rgb(pic);
   pred = predict(classifier1,pic);
   if pred == 1 & imd_test1.Labels(i) == 'CATS'
       confusion_matrix(1,1) = confusion_matrix(1,1) + 1;
   elseif pred == 1 & imd_test1.Labels(i) == 'DOGS'
       confusion_matrix(2,1) = confusion_matrix(2,1) + 1;
   elseif pred == 2 & imd_test1.Labels(i) == 'CATS'
       confusion_matrix(1,2) = confusion_matrix(1,2) + 1;
   elseif pred == 2 & imd_test1.Labels(i) == 'DOGS'
       confusion_matrix(2,2) = confusion_matrix(2,2) + 1;
   end
end
sat(end + 1) = (confusion_matrix(1,1)+confusion_matrix(2,2))/796;
end
```

Figure 19: Saturation noise on test set using SVM with BOVW (matlab)

```
%% Occlusion
occ = [];
pics = readall(imd_test1);
for j=0:5:45
confusion_matrix = [0,0;0,0];
for i=1:796
   pic = cell2mat(pics(i));
   if length(size(pic)) == 2
       pic = pic(:,:,[1 1 1]);
   end
   size_pic = size(pic);
   top_left_x = ceil( (size_pic(2)-j) * rand(1) );
   top_left_y = ceil( (size_pic(1)-j) * rand(1) );
   pic(top_left_y:(top_left_y + j - 1),top_left_x:(top_left_x + j - 1),1) = zeros(j);
   pic(top_left_y:(top_left_y + j - 1),top_left_x:(top_left_x + j - 1),2) = zeros(j);
   pic(top_left_y:(top_left_y + j - 1),top_left_x:(top_left_x + j - 1),3) = zeros(j);
   pred = predict(classifier1,pic);
   if pred == 1 & imd_test1.Labels(i) == 'CATS'
       confusion_matrix(1,1) = confusion_matrix(1,1) + 1;
   elseif pred == 1 & imd_test1.Labels(i) == 'DOGS'
       confusion_matrix(2,1) = confusion_matrix(2,1) + 1;
   elseif pred == 2 & imd_test1.Labels(i) == 'CATS'
       confusion_matrix(1,2) = confusion_matrix(1,2) + 1;
   elseif pred == 2 & imd_test1.Labels(i) == 'DOGS'
       confusion_matrix(2,2) = confusion_matrix(2,2) + 1;
   end
end
occ(end + 1) = (confusion_matrix(1,1)+confusion_matrix(2,2))/796;
end

results1 = { noise, blur, cont_incr, cont_decr, bri_incr, bri_decr, hue, sat, occ };
```

Figure 20: Occlusion Code - SVM with BOVW (matlab)

# B Hyperparameter Search Results

In this section we provide the results for the random hyperparameter search for both ResNet18 transfer learning and SVM with BoVW. 200 random configurations of hyperparameters were selected and used for each fold. The hyperparameters that maximised the accuracy on a held-out validation set were used for that fold. The last column 'Best' indicates the best combination of hyperparameters for each fold, where F1 is the first fold etc. Any accuracy of 0 is due to numerical instability.

## B.1 ResNet18

| Learning Rate | Momentum | Step Size | Gamma | Batch Size | Acc. (Fold 1) | Acc. (Fold 2) | Acc. (Fold 3) | Best |
|---|---|---|---|---|---|---|---|---|
| 0.0018 | 0.5 | 3 | 0.1 | 32 | 0.807 | 0.864 | 0.857 | |
| 0.0993 | 0.5 | 3 | 0.259 | 32 | 0.204 | 0.118 | 0 | |
| 0.0011 | 0.99 | 5 | 0.316 | 16 | 0.819 | 0.736 | 0.505 | |
| 0.0009 | 0.5 | 3 | 0.451 | 8 | 0.899 | 0.807 | 0.824 | |
| 0.004 | 0.5 | 5 | 0.272 | 16 | 0.867 | 0.925 | 0.915 | |
| 0.0004 | 0.9 | 5 | 0.487 | 32 | 0.839 | 0.882 | 0.889 | |
| 0.0002 | 0.9 | 5 | 0.446 | 8 | 0.902 | 0.94 | 0.877 | |
| 0.0002 | 0.9 | 7 | 0.451 | 16 | 0.857 | 0.857 | 0.872 | |
| 0.006 | 0.99 | 5 | 0.265 | 32 | 0.246 | 0.078 | 0.786 | |

| Learning Rate | Momentum | Step Size | Gamma | Batch Size | Acc. (Fold 1) | Acc. (Fold 2) | Acc. (Fold 3) | Best |
|---|---|---|---|---|---|---|---|---|
| 0.0119 | 0.9 | 3 | 0.413 | 4 | 0 | 0.013 | 0.055 | |
| 0.0001 | 0.9 | 7 | 0.496 | 4 | 0.947 | 0.98 | 0.93 | |
| 0.0008 | 0.99 | 7 | 0.129 | 16 | 0.844 | 0.837 | 0.802 | |
| 0.0532 | 0.99 | 5 | 0.152 | 8 | 0.078 | 0.118 | 0.103 | |
| 0.0002 | 0.99 | 3 | 0.464 | 16 | 0.942 | 0.927 | 0.92 | |
| 0.003 | 0.9 | 3 | 0.336 | 16 | 0.92 | 0.852 | 0.922 | |
| 0.0005 | 0.9 | 5 | 0.297 | 4 | 0.96 | 0.962 | 0.975 | F3 |
| 0.0001 | 0.9 | 7 | 0.366 | 16 | 0.872 | 0.832 | 0.872 | |
| 0.0238 | 0.9 | 5 | 0.155 | 4 | 0.128 | 0.241 | 0.138 | |
| 0.0475 | 0.99 | 3 | 0.315 | 8 | 0.166 | 0.008 | 0 | |
| 0.0606 | 0.5 | 3 | 0.4 | 8 | 0.196 | 0.296 | 0 | |
| 0.0002 | 0.5 | 5 | 0.359 | 4 | 0.894 | 0.812 | 0.912 | |
| 0.0011 | 0.5 | 3 | 0.458 | 16 | 0.864 | 0.892 | 0.859 | |
| 0.0441 | 0.5 | 7 | 0.299 | 16 | 0.394 | 0.394 | 0.817 | |
| 0.0002 | 0.99 | 7 | 0.242 | 8 | 0.894 | 0.854 | 0.872 | |
| 0.0017 | 0.99 | 7 | 0.461 | 4 | 0.053 | 0.073 | 0.045 | |
| 0.0008 | 0.9 | 5 | 0.161 | 16 | 0.905 | 0.854 | 0.907 | |
| 0.0038 | 0.9 | 5 | 0.243 | 8 | 0.864 | 0.907 | 0.832 | |
| 0.0197 | 0.99 | 3 | 0.208 | 4 | 0.111 | 0 | 0.033 | |
| 0.0118 | 0.5 | 5 | 0.289 | 8 | 0.93 | 0.947 | 0.94 | |
| 0.0628 | 0.5 | 3 | 0.126 | 8 | 0.186 | 0.018 | 0 | |
| 0.0033 | 0.5 | 5 | 0.308 | 8 | 0.957 | 0.937 | 0.925 | |
| 0.0001 | 0.9 | 3 | 0.111 | 8 | 0.862 | 0.907 | 0.922 | |
| 0.0002 | 0.9 | 5 | 0.321 | 8 | 0.965 | 0.927 | 0.872 | |
| 0.0007 | 0.5 | 7 | 0.137 | 4 | 0.97 | 0.95 | 0.917 | |
| 0.0811 | 0.99 | 3 | 0.107 | 32 | 0 | 0.01 | 0 | |
| 0.0007 | 0.9 | 5 | 0.345 | 32 | 0.882 | 0.869 | 0.874 | |
| 0.039 | 0.5 | 5 | 0.322 | 16 | 0.52 | 0.201 | 0.857 | |
| 0.0012 | 0.99 | 7 | 0.475 | 32 | 0.776 | 0.796 | 0.791 | |
| 0.0002 | 0.5 | 5 | 0.385 | 8 | 0.884 | 0.899 | 0.839 | |
| 0.0092 | 0.5 | 5 | 0.487 | 4 | 0.892 | 0.925 | 0.925 | |
| 0.0007 | 0.99 | 7 | 0.475 | 32 | 0.829 | 0.824 | 0.884 | |
| 0.0813 | 0.99 | 3 | 0.219 | 4 | 0.005 | 0 | 0 | |
| 0.0072 | 0.5 | 5 | 0.36 | 32 | 0.892 | 0.902 | 0.857 | |
| 0.0029 | 0.9 | 3 | 0.313 | 16 | 0.912 | 0.935 | 0.894 | |
| 0.0055 | 0.99 | 5 | 0.32 | 4 | 0 | 0.126 | 0 | |
| 0.0561 | 0.9 | 5 | 0.406 | 16 | 0.156 | 0.113 | 0.148 | |
| 0.0077 | 0.9 | 7 | 0.401 | 4 | 0 | 0.116 | 0.018 | |
| 0.0012 | 0.99 | 3 | 0.157 | 4 | 0 | 0.121 | 0.02 | |
| 0.0083 | 0.5 | 5 | 0.349 | 32 | 0.902 | 0.905 | 0.899 | |
| 0.0002 | 0.5 | 7 | 0.463 | 16 | 0.789 | 0.802 | 0.779 | |
| 0.0115 | 0.5 | 3 | 0.391 | 16 | 0.935 | 0.899 | 0.945 | |
| 0.0703 | 0.5 | 7 | 0.323 | 8 | 0 | 0.01 | 0.005 | |
| 0.0001 | 0.9 | 7 | 0.218 | 16 | 0.844 | 0.879 | 0.844 | |
| 0.0065 | 0.9 | 7 | 0.47 | 4 | 0.176 | 0.59 | 0.462 | |
| 0.0013 | 0.99 | 3 | 0.17 | 32 | 0.877 | 0.935 | 0.894 | |
| 0.0002 | 0.5 | 7 | 0.105 | 4 | 0.917 | 0.882 | 0.962 | |
| 0.0698 | 0.99 | 5 | 0.106 | 4 | 0.269 | 0.035 | 0.058 | |
| 0.0002 | 0.5 | 5 | 0.454 | 8 | 0.902 | 0.829 | 0.829 | |
| 0.0011 | 0.99 | 3 | 0.333 | 16 | 0.807 | 0.802 | 0.57 | |
| 0.0005 | 0.99 | 7 | 0.486 | 16 | 0.884 | 0.912 | 0.706 | |
| 0.0044 | 0.99 | 3 | 0.296 | 8 | 0.106 | 0.038 | 0 | |
| 0.0009 | 0.99 | 7 | 0.187 | 8 | 0.296 | 0.399 | 0.58 | |
| 0.0264 | 0.9 | 5 | 0.457 | 4 | 0 | 0.374 | 0.013 | |
| 0.0043 | 0.9 | 3 | 0.336 | 16 | 0.882 | 0.869 | 0.925 | |
| 0.0001 | 0.5 | 3 | 0.372 | 32 | 0.43 | 0.452 | 0.44 | |
| 0.0054 | 0.9 | 5 | 0.251 | 8 | 0.824 | 0.653 | 0.817 | |
| 0.0002 | 0.9 | 7 | 0.184 | 4 | 0.96 | 0.947 | 0.955 | |
| 0.0077 | 0.9 | 5 | 0.379 | 4 | 0.038 | 0.09 | 0 | |
| 0.0376 | 0.9 | 7 | 0.379 | 16 | 0.525 | 0.284 | 0.035 | |
| 0.0001 | 0.5 | 5 | 0.244 | 4 | 0.905 | 0.864 | 0.852 | |
| 0.0017 | 0.99 | 7 | 0.227 | 8 | 0.198 | 0.191 | 0 | |
| 0.0048 | 0.5 | 3 | 0.268 | 8 | 0.96 | 0.94 | 0.95 | |
| 0.0004 | 0.99 | 7 | 0.188 | 4 | 0.389 | 0.339 | 0.234 | |
| 0.0437 | 0.5 | 7 | 0.107 | 16 | 0.847 | 0.294 | 0.455 | |
| 0.0367 | 0.99 | 5 | 0.421 | 16 | 0.05 | 0 | 0.008 | |
| 0.0002 | 0.9 | 3 | 0.278 | 16 | 0.879 | 0.892 | 0.854 | |
| 0.0051 | 0.99 | 5 | 0.355 | 8 | 0.216 | 0.04 | 0 | |
| 0.0014 | 0.99 | 7 | 0.118 | 4 | 0 | 0 | 0 | |
| 0.0273 | 0.99 | 7 | 0.364 | 32 | 0.008 | 0.035 | 0.003 | |
| 0.0003 | 0.99 | 5 | 0.238 | 8 | 0.852 | 0.937 | 0.827 | |
| 0.0348 | 0.9 | 5 | 0.156 | 16 | 0.402 | 0.095 | 0 | |
| 0.0536 | 0.5 | 7 | 0.144 | 8 | 0 | 0 | 0 | |
| 0.0028 | 0.5 | 3 | 0.104 | 4 | 0.922 | 0.915 | 0.905 | |
| 0.0006 | 0.5 | 3 | 0.266 | 32 | 0.756 | 0.786 | 0.761 | |
| 0.0065 | 0.99 | 3 | 0.306 | 4 | 0.136 | 0.113 | 0 | |
| 0.0004 | 0.5 | 7 | 0.318 | 16 | 0.854 | 0.889 | 0.867 | |
| 0.0034 | 0.9 | 3 | 0.158 | 4 | 0.294 | 0.472 | 0.349 | |
| 0.0036 | 0.5 | 3 | 0.257 | 8 | 0.942 | 0.894 | 0.877 | |
| 0.0342 | 0.9 | 3 | 0.447 | 32 | 0.173 | 0.332 | 0.254 | |
| 0.0004 | 0.99 | 7 | 0.229 | 16 | 0.912 | 0.819 | 0.729 | |
| 0.009 | 0.99 | 5 | 0.344 | 16 | 0.261 | 0.055 | 0 | |
| 0.013 | 0.9 | 3 | 0.488 | 8 | 0.327 | 0.369 | 0.357 | |
| 0.0006 | 0.9 | 5 | 0.353 | 32 | 0.905 | 0.847 | 0.874 | |
| 0.0217 | 0.99 | 7 | 0.413 | 32 | 0.372 | 0.239 | 0.013 | |
| 0.0022 | 0.9 | 3 | 0.322 | 4 | 0.905 | 0.902 | 0.869 | |
| 0.084 | 0.99 | 7 | 0.498 | 4 | 0.101 | 0.053 | 0 | |
| 0.0095 | 0.5 | 7 | 0.283 | 8 | 0.955 | 0.937 | 0.94 | |
| 0.0001 | 0.9 | 3 | 0.398 | 16 | 0.822 | 0.862 | 0.784 | |
| 0.0009 | 0.99 | 3 | 0.292 | 8 | 0.603 | 0.161 | 0.317 | |
| 0.0004 | 0.5 | 3 | 0.496 | 4 | 0.844 | 0.91 | 0.92 | |
| 0.0007 | 0.5 | 7 | 0.4 | 32 | 0.867 | 0.824 | 0.844 | |
| 0.0004 | 0.99 | 3 | 0.286 | 16 | 0.869 | 0.907 | 0.93 | |
| 0.0249 | 0.9 | 5 | 0.111 | 16 | 0.005 | 0.284 | 0.249 | |
| 0.0002 | 0.99 | 5 | 0.4 | 32 | 0.905 | 0.952 | 0.899 | |
| 0.0514 | 0.5 | 3 | 0.203 | 8 | 0.286 | 0.038 | 0 | |
| 0.0134 | 0.99 | 3 | 0.133 | 32 | 0.058 | 0.04 | 0 | |
| 0.0006 | 0.9 | 7 | 0.144 | 32 | 0.864 | 0.889 | 0.877 | |
| 0.0001 | 0.99 | 5 | 0.283 | 4 | 0.847 | 0.982 | 0.937 | F2 |
| 0.0011 | 0.9 | 3 | 0.427 | 4 | 0.907 | 0.947 | 0.915 | |
| 0.0076 | 0.5 | 7 | 0.107 | 4 | 0.92 | 0.859 | 0.819 | |

| Learning Rate | Momentum | Step Size | Gamma | Batch Size | Acc. (Fold 1) | Acc. (Fold 2) | Acc. (Fold 3) | Best |
|---|---|---|---|---|---|---|---|---|
| 0.0056 | 0.5 | 5 | 0.244 | 16 | 0.937 | 0.892 | 0.942 | |
| 0.0184 | 0.5 | 7 | 0.434 | 32 | 0.897 | 0.915 | 0.859 | |
| 0.0022 | 0.5 | 3 | 0.29 | 4 | 0.932 | 0.965 | 0.91 | |
| 0.0943 | 0.99 | 5 | 0.361 | 16 | 0.608 | 0.003 | 0 | |
| 0.0002 | 0.9 | 7 | 0.104 | 4 | 0.945 | 0.894 | 0.925 | |
| 0.0002 | 0.5 | 5 | 0.41 | 4 | 0.892 | 0.899 | 0.917 | |
| 0.0244 | 0.9 | 5 | 0.131 | 8 | 0.121 | 0.08 | 0 | |
| 0.0004 | 0.99 | 5 | 0.462 | 4 | 0.442 | 0.869 | 0.372 | |
| 0.0383 | 0.99 | 5 | 0.464 | 16 | 0 | 0.382 | 0.013 | |
| 0.0053 | 0.99 | 7 | 0.425 | 16 | 0.402 | 0 | 0.246 | |
| 0.0349 | 0.9 | 3 | 0.334 | 4 | 0.08 | 0.05 | 0.043 | |
| 0.0886 | 0.5 | 3 | 0.313 | 4 | 0 | 0.033 | 0.01 | |
| 0.0207 | 0.9 | 5 | 0.468 | 32 | 0.673 | 0.337 | 0.151 | |
| 0.0034 | 0.99 | 5 | 0.112 | 8 | 0.186 | 0 | 0 | |
| 0.0028 | 0.99 | 5 | 0.309 | 4 | 0.068 | 0.146 | 0 | |
| 0.0458 | 0.99 | 5 | 0.306 | 32 | 0.025 | 0.01 | 0 | |
| 0.0023 | 0.5 | 5 | 0.458 | 8 | 0.877 | 0.922 | 0.955 | |
| 0.0009 | 0.99 | 7 | 0.467 | 16 | 0.824 | 0.862 | 0.791 | |
| 0.0003 | 0.5 | 5 | 0.124 | 32 | 0.686 | 0.746 | 0.696 | |
| 0.0647 | 0.5 | 3 | 0.468 | 16 | 0 | 0.357 | 0.075 | |
| 0.0002 | 0.99 | 3 | 0.243 | 4 | 0.726 | 0.812 | 0.847 | |
| 0.0003 | 0.99 | 5 | 0.481 | 32 | 0.889 | 0.925 | 0.877 | |
| 0.0004 | 0.9 | 5 | 0.291 | 16 | 0.897 | 0.872 | 0.93 | |
| 0.0001 | 0.99 | 5 | 0.299 | 32 | 0.91 | 0.884 | 0.897 | |
| 0.0003 | 0.99 | 7 | 0.471 | 4 | 0 | 0.862 | 0.731 | |
| 0.0282 | 0.5 | 7 | 0.49 | 8 | 0.802 | 0.538 | 0.799 | |
| 0.0196 | 0.99 | 5 | 0.137 | 32 | 0.003 | 0.01 | 0 | |
| 0.0002 | 0.9 | 3 | 0.228 | 4 | 0.932 | 0.849 | 0.947 | |
| 0.0003 | 0.9 | 7 | 0.178 | 32 | 0.884 | 0.887 | 0.854 | |
| 0.0079 | 0.5 | 3 | 0.291 | 16 | 0.877 | 0.872 | 0.912 | |
| 0.0873 | 0.9 | 7 | 0.184 | 4 | 0.216 | 0.106 | 0.01 | |
| 0.0025 | 0.9 | 5 | 0.155 | 8 | 0.922 | 0.932 | 0.94 | |
| 0.0002 | 0.5 | 3 | 0.166 | 16 | 0.636 | 0.701 | 0.653 | |
| 0.001 | 0.5 | 7 | 0.402 | 16 | 0.849 | 0.872 | 0.889 | |
| 0.0408 | 0.9 | 3 | 0.107 | 16 | 0 | 0.133 | 0.083 | |
| 0.0105 | 0.99 | 5 | 0.259 | 8 | 0.309 | 0.337 | 0.173 | |
| 0.0036 | 0.5 | 3 | 0.105 | 8 | 0.955 | 0.94 | 0.952 | |
| 0.0124 | 0.99 | 7 | 0.154 | 8 | 0 | 0.103 | 0.008 | |
| 0.0001 | 0.5 | 7 | 0.444 | 16 | 0.721 | 0.781 | 0.623 | |
| 0.0074 | 0.99 | 5 | 0.425 | 16 | 0 | 0.06 | 0 | |
| 0.0007 | 0.5 | 5 | 0.319 | 8 | 0.95 | 0.874 | 0.844 | |
| 0.0369 | 0.99 | 7 | 0.399 | 16 | 0 | 0.003 | 0.005 | |
| 0.007 | 0.9 | 5 | 0.31 | 4 | 0.09 | 0 | 0.07 | |
| 0.0011 | 0.5 | 5 | 0.407 | 32 | 0.857 | 0.859 | 0.849 | |
| 0.085 | 0.9 | 5 | 0.189 | 32 | 0 | 0 | 0 | |
| 0.0042 | 0.99 | 5 | 0.297 | 8 | 0.015 | 0.015 | 0 | |
| 0.0001 | 0.99 | 3 | 0.37 | 32 | 0.887 | 0.905 | 0.879 | |
| 0.0118 | 0.99 | 5 | 0.374 | 8 | 0.08 | 0.06 | 0.07 | |
| 0.0565 | 0.5 | 5 | 0.322 | 8 | 0.294 | 0.211 | 0.339 | |
| 0.0026 | 0.99 | 5 | 0.347 | 8 | 0.008 | 0.008 | 0.188 | |
| 0.0131 | 0.99 | 5 | 0.25 | 4 | 0.06 | 0.003 | 0 | |
| 0.0003 | 0.99 | 5 | 0.272 | 4 | 0.749 | 0.791 | 0.902 | |
| 0.0022 | 0.9 | 7 | 0.238 | 8 | 0.965 | 0.94 | 0.915 | |
| 0.0085 | 0.5 | 5 | 0.34 | 4 | 0.93 | 0.925 | 0.932 | |
| 0.076 | 0.9 | 5 | 0.118 | 16 | 0.111 | 0.075 | 0 | |
| 0.0685 | 0.99 | 7 | 0.122 | 16 | 0.005 | 0.015 | 0 | |
| 0.0177 | 0.99 | 5 | 0.18 | 4 | 0.03 | 0 | 0.043 | |
| 0.0517 | 0.5 | 7 | 0.312 | 4 | 0 | 0.721 | 0.08 | |
| 0.0054 | 0.5 | 7 | 0.303 | 16 | 0.912 | 0.915 | 0.899 | |
| 0.0066 | 0.99 | 3 | 0.415 | 32 | 0.003 | 0.156 | 0.188 | |
| 0.0225 | 0.5 | 5 | 0.329 | 16 | 0.947 | 0.925 | 0.897 | |
| 0.0575 | 0.9 | 5 | 0.334 | 16 | 0.118 | 0.018 | 0.068 | |
| 0.0003 | 0.9 | 5 | 0.187 | 8 | 0.917 | 0.902 | 0.927 | |
| 0.0345 | 0.9 | 3 | 0.205 | 32 | 0.128 | 0.261 | 0.121 | |
| 0.0001 | 0.99 | 3 | 0.36 | 4 | 0.972 | 0.95 | 0.967 | |
| 0.0142 | 0.9 | 7 | 0.211 | 8 | 0 | 0.065 | 0.111 | |
| 0.0002 | 0.99 | 3 | 0.192 | 8 | 0.93 | 0.917 | 0.882 | |
| 0.0035 | 0.5 | 7 | 0.412 | 16 | 0.93 | 0.92 | 0.877 | |
| 0.0019 | 0.9 | 5 | 0.38 | 16 | 0.912 | 0.96 | 0.907 | |
| 0.0016 | 0.5 | 3 | 0.382 | 16 | 0.852 | 0.844 | 0.884 | |
| 0.0489 | 0.5 | 7 | 0.188 | 32 | 0.829 | 0.897 | 0.884 | |
| 0.0537 | 0.99 | 7 | 0.215 | 4 | 0.02 | 0 | 0 | |
| 0.0002 | 0.9 | 5 | 0.23 | 16 | 0.887 | 0.877 | 0.897 | |
| 0.0025 | 0.99 | 3 | 0.339 | 16 | 0.55 | 0.339 | 0.013 | |
| 0.0106 | 0.9 | 7 | 0.462 | 4 | 0 | 0 | 0.07 | |
| 0.0076 | 0.99 | 7 | 0.279 | 4 | 0 | 0.005 | 0 | |
| 0.0024 | 0.99 | 3 | 0.15 | 16 | 0.005 | 0 | 0.05 | |
| 0.0534 | 0.99 | 3 | 0.407 | 16 | 0 | 0 | 0 | |
| 0.0655 | 0.99 | 7 | 0.293 | 4 | 0 | 0.126 | 0.967 | |
| 0.0052 | 0.5 | 3 | 0.307 | 4 | 0.905 | 0.952 | 0.942 | |
| 0.0002 | 0.99 | 5 | 0.327 | 4 | 0.842 | 0.942 | 0.827 | |
| 0.0702 | 0.5 | 3 | 0.435 | 32 | 0.299 | 0.302 | 0.508 | |
| 0.0072 | 0.5 | 7 | 0.125 | 16 | 0.95 | 0.917 | 0.922 | |
| 0.0689 | 0.99 | 3 | 0.134 | 32 | 0 | 0.003 | 0 | |
| 0.0021 | 0.5 | 3 | 0.421 | 4 | 0.977 | 0.96 | 0.884 | F1 |
| 0.004 | 0.5 | 3 | 0.458 | 16 | 0.889 | 0.889 | 0.882 | |
| 0.0005 | 0.5 | 7 | 0.373 | 4 | 0.917 | 0.927 | 0.952 | |
| 0.0014 | 0.5 | 7 | 0.486 | 8 | 0.922 | 0.927 | 0.952 | |
| 0.0005 | 0.9 | 3 | 0.154 | 16 | 0.867 | 0.899 | 0.829 | |
| 0.0141 | 0.5 | 7 | 0.418 | 8 | 0.892 | 0.942 | 0.867 | |

## B.2 SVM with BoVW

| Visual Words | Proportion | U-SURF | Kernel | Degree | Fold 1 Accuracy | Fold 2 Accuracy | Fold 3 Accuracy | Best |
|---|---|---|---|---|---|---|---|---|
| 6000 | 0.75 | 1 | gaussian | N/A | 0.809 | 0.794 | 0.809 | |
| 10000 | 0.85 | 1 | polynomial | 4 | 0.839 | 0.817 | 0.829 | |
| 8000 | 0.6 | 0 | gaussian | N/A | 0.764 | 0.769 | 0.761 | |
| 9000 | 0.95 | 0 | linear | N/A | 0.733 | 0.776 | 0.734 | |

| Visual Words | Proportion | U-SURF | Kernel | Degree | Fold 1 Accuracy | Fold 2 Accuracy | Fold 3 Accuracy | Best |
|---|---|---|---|---|---|---|---|---|
| 2000 | 0.95 | 1 | gaussian | N/A | 0.821 | 0.814 | 0.804 | |
| 6000 | 0.85 | 0 | linear | N/A | 0.748 | 0.698 | 0.761 | |
| 1000 | 0.6 | 0 | polynomial | 3 | 0.736 | 0.719 | 0.779 | |
| 1000 | 0.8 | 0 | polynomial | 4 | 0.749 | 0.741 | 0.744 | |
| 7000 | 0.8 | 1 | gaussian | N/A | 0.811 | 0.779 | 0.819 | |
| 9000 | 0.6 | 0 | polynomial | 2 | 0.761 | 0.751 | 0.739 | |
| 6000 | 0.8 | 0 | linear | N/A | 0.754 | 0.746 | 0.761 | |
| 2000 | 0.85 | 1 | gaussian | N/A | 0.819 | 0.817 | 0.811 | |
| 3000 | 0.95 | 0 | polynomial | 2 | 0.746 | 0.731 | 0.739 | |
| 7000 | 0.6 | 0 | gaussian | N/A | 0.759 | 0.759 | 0.759 | |
| 4000 | 0.65 | 1 | gaussian | N/A | 0.832 | 0.784 | 0.804 | |
| 4000 | 0.7 | 0 | gaussian | N/A | 0.766 | 0.731 | 0.796 | |
| 1000 | 0.9 | 1 | gaussian | N/A | 0.821 | 0.809 | 0.791 | |
| 5000 | 0.8 | 1 | polynomial | 3 | 0.809 | 0.839 | 0.801 | F2 |
| 3000 | 0.8 | 1 | linear | N/A | 0.814 | 0.751 | 0.794 | |
| 7000 | 0.8 | 1 | gaussian | N/A | 0.821 | 0.804 | 0.791 | |
| 6000 | 0.6 | 1 | polynomial | 5 | 0.804 | 0.814 | 0.817 | |
| 4000 | 0.8 | 0 | gaussian | N/A | 0.751 | 0.724 | 0.756 | |
| 4000 | 0.95 | 1 | polynomial | 4 | 0.809 | 0.814 | 0.801 | |
| 6000 | 0.85 | 0 | polynomial | 4 | 0.761 | 0.746 | 0.769 | |
| 9000 | 0.65 | 0 | linear | N/A | 0.731 | 0.721 | 0.731 | |
| 4000 | 0.9 | 1 | polynomial | 5 | 0.814 | 0.827 | 0.794 | |
| 4000 | 0.85 | 1 | linear | N/A | 0.789 | 0.796 | 0.776 | |
| 2000 | 0.85 | 0 | gaussian | N/A | 0.716 | 0.744 | 0.761 | |
| 9000 | 0.85 | 0 | linear | N/A | 0.764 | 0.761 | 0.779 | |
| 10000 | 0.95 | 0 | linear | N/A | 0.776 | 0.739 | 0.726 | |
| 1000 | 0.65 | 0 | polynomial | 4 | 0.769 | 0.741 | 0.761 | |
| 7000 | 0.75 | 0 | polynomial | 5 | 0.761 | 0.769 | 0.754 | |
| 4000 | 0.6 | 1 | gaussian | N/A | 0.847 | 0.824 | 0.796 | F1 |
| 7000 | 0.65 | 1 | linear | N/A | 0.806 | 0.781 | 0.792 | |
| 9000 | 0.75 | 0 | polynomial | 3 | 0.754 | 0.731 | 0.726 | |
| 3000 | 0.95 | 0 | linear | N/A | 0.766 | 0.744 | 0.731 | |
| 8000 | 0.7 | 0 | polynomial | 5 | 0.731 | 0.784 | 0.769 | |
| 4000 | 0.75 | 1 | linear | N/A | 0.824 | 0.791 | 0.796 | |
| 1000 | 0.75 | 1 | linear | N/A | 0.789 | 0.789 | 0.764 | |
| 10000 | 0.85 | 1 | gaussian | N/A | 0.816 | 0.829 | 0.829 | |
| 6000 | 0.6 | 0 | gaussian | N/A | 0.716 | 0.781 | 0.764 | |
| 9000 | 0.6 | 1 | linear | N/A | 0.832 | 0.781 | 0.829 | |
| 5000 | 0.95 | 0 | polynomial | 2 | 0.724 | 0.746 | 0.774 | |
| 2000 | 0.6 | 0 | gaussian | N/A | 0.746 | 0.726 | 0.751 | |
| 10000 | 0.8 | 0 | gaussian | N/A | 0.756 | 0.754 | 0.736 | |
| 8000 | 0.9 | 0 | polynomial | 2 | 0.754 | 0.751 | 0.746 | |
| 9000 | 0.95 | 0 | polynomial | 2 | 0.726 | 0.759 | 0.749 | |
| 7000 | 0.7 | 0 | linear | N/A | 0.756 | 0.726 | 0.744 | |
| 9000 | 0.7 | 1 | gaussian | N/A | 0.824 | 0.814 | 0.804 | |
| 6000 | 0.8 | 1 | polynomial | 3 | 0.811 | 0.799 | 0.811 | |
| 9000 | 0.7 | 1 | gaussian | N/A | 0.824 | 0.799 | 0.824 | |
| 2000 | 0.95 | 0 | linear | N/A | 0.744 | 0.694 | 0.756 | |
| 5000 | 0.65 | 0 | linear | N/A | 0.761 | 0.741 | 0.736 | |
| 5000 | 0.85 | 1 | polynomial | 2 | 0.804 | 0.799 | 0.801 | |
| 1000 | 0.85 | 1 | linear | N/A | 0.811 | 0.781 | 0.784 | |
| 10000 | 0.7 | 0 | polynomial | 2 | 0.726 | 0.771 | 0.736 | |
| 7000 | 0.95 | 0 | polynomial | 3 | 0.779 | 0.736 | 0.766 | |
| 5000 | 0.95 | 1 | linear | N/A | 0.824 | 0.771 | 0.804 | |
| 9000 | 0.85 | 0 | polynomial | 5 | 0.743 | 0.739 | 0.764 | |
| 3000 | 0.75 | 0 | linear | N/A | 0.726 | 0.708 | 0.749 | |
| 1000 | 0.9 | 1 | polynomial | 3 | 0.801 | 0.799 | 0.801 | |
| 7000 | 0.85 | 0 | polynomial | 3 | 0.754 | 0.741 | 0.741 | |
| 3000 | 0.65 | 0 | gaussian | N/A | 0.744 | 0.744 | 0.749 | |
| 7000 | 0.75 | 0 | polynomial | 4 | 0.761 | 0.749 | 0.746 | |
| 6000 | 0.9 | 1 | gaussian | N/A | 0.819 | 0.824 | 0.807 | |
| 6000 | 0.65 | 1 | polynomial | 5 | 0.809 | 0.819 | 0.794 | |
| 1000 | 0.8 | 0 | linear | N/A | 0.696 | 0.719 | 0.756 | |
| 3000 | 0.65 | 0 | gaussian | N/A | 0.723 | 0.771 | 0.731 | |
| 3000 | 0.65 | 1 | gaussian | N/A | 0.821 | 0.819 | 0.791 | |
| 7000 | 0.7 | 0 | gaussian | N/A | 0.771 | 0.756 | 0.749 | |
| 6000 | 0.95 | 0 | polynomial | 2 | 0.723 | 0.744 | 0.739 | |
| 4000 | 0.9 | 0 | polynomial | 3 | 0.766 | 0.764 | 0.741 | |
| 2000 | 0.7 | 1 | polynomial | 3 | 0.801 | 0.794 | 0.789 | |
| 8000 | 0.75 | 1 | polynomial | 3 | 0.801 | 0.829 | 0.789 | |
| 9000 | 0.65 | 0 | gaussian | N/A | 0.759 | 0.812 | 0.731 | |
| 2000 | 0.75 | 0 | linear | N/A | 0.746 | 0.724 | 0.759 | |
| 8000 | 0.9 | 0 | gaussian | N/A | 0.739 | 0.729 | 0.776 | |
| 6000 | 0.6 | 0 | linear | N/A | 0.736 | 0.751 | 0.734 | |
| 7000 | 0.95 | 0 | linear | N/A | 0.754 | 0.754 | 0.741 | |
| 7000 | 0.95 | 1 | polynomial | 2 | 0.809 | 0.786 | 0.799 | |
| 8000 | 0.6 | 0 | polynomial | 5 | 0.754 | 0.779 | 0.756 | |
| 8000 | 0.75 | 1 | linear | N/A | 0.784 | 0.794 | 0.796 | |
| 8000 | 0.8 | 1 | gaussian | N/A | 0.824 | 0.812 | 0.814 | |
| 9000 | 0.9 | 0 | polynomial | 4 | 0.736 | 0.736 | 0.749 | |
| 10000 | 0.7 | 1 | gaussian | N/A | 0.814 | 0.792 | 0.794 | |
| 1000 | 0.7 | 1 | linear | N/A | 0.786 | 0.764 | 0.809 | |
| 8000 | 0.7 | 1 | linear | N/A | 0.839 | 0.794 | 0.806 | |
| 10000 | 0.8 | 0 | polynomial | 2 | 0.751 | 0.726 | 0.736 | |
| 8000 | 0.65 | 0 | gaussian | N/A | 0.741 | 0.716 | 0.746 | |
| 3000 | 0.8 | 1 | gaussian | N/A | 0.829 | 0.799 | 0.806 | |
| 5000 | 0.75 | 0 | linear | N/A | 0.746 | 0.724 | 0.718 | |
| 5000 | 0.9 | 0 | linear | N/A | 0.749 | 0.749 | 0.739 | |
| 4000 | 0.9 | 0 | gaussian | N/A | 0.729 | 0.744 | 0.764 | |
| 6000 | 0.95 | 1 | gaussian | N/A | 0.824 | 0.814 | 0.814 | |
| 2000 | 0.8 | 0 | linear | N/A | 0.754 | 0.741 | 0.744 | |
| 2000 | 0.8 | 1 | gaussian | N/A | 0.814 | 0.784 | 0.794 | |
| 7000 | 0.65 | 1 | gaussian | N/A | 0.806 | 0.794 | 0.814 | |
| 6000 | 0.95 | 1 | polynomial | 4 | 0.811 | 0.807 | 0.799 | |
| 8000 | 0.65 | 0 | polynomial | 3 | 0.741 | 0.764 | 0.761 | |
| 9000 | 0.9 | 1 | gaussian | N/A | 0.811 | 0.809 | 0.804 | |
| 7000 | 0.9 | 1 | gaussian | N/A | 0.811 | 0.824 | 0.804 | |
| 1000 | 0.9 | 0 | linear | N/A | 0.774 | 0.756 | 0.746 | |
| 9000 | 0.9 | 0 | gaussian | N/A | 0.766 | 0.764 | 0.741 | |
| 3000 | 0.8 | 0 | gaussian | N/A | 0.739 | 0.744 | 0.748 | |
| 10000 | 0.8 | 0 | gaussian | N/A | 0.749 | 0.751 | 0.736 | |

| Visual Words | Proportion | U-SURF | Kernel | Degree | Fold 1 Accuracy | Fold 2 Accuracy | Fold 3 Accuracy | Best |
|---|---|---|---|---|---|---|---|---|
| 2000 | 0.75 | 0 | polynomial | 2 | 0.728 | 0.713 | 0.754 | |
| 5000 | 0.65 | 1 | polynomial | 2 | 0.801 | 0.781 | 0.797 | |
| 3000 | 0.8 | 0 | linear | N/A | 0.744 | 0.729 | 0.766 | |
| 6000 | 0.9 | 1 | linear | N/A | 0.799 | 0.812 | 0.822 | |
| 3000 | 0.65 | 0 | linear | N/A | 0.726 | 0.749 | 0.769 | |
| 8000 | 0.7 | 1 | gaussian | N/A | 0.789 | 0.802 | 0.784 | |
| 10000 | 0.85 | 1 | linear | N/A | 0.796 | 0.794 | 0.796 | |
| 10000 | 0.7 | 0 | linear | N/A | 0.739 | 0.751 | 0.741 | |
| 7000 | 0.65 | 0 | polynomial | 3 | 0.749 | 0.774 | 0.769 | |
| 5000 | 0.85 | 0 | linear | N/A | 0.746 | 0.726 | 0.751 | |
| 10000 | 0.7 | 1 | gaussian | N/A | 0.781 | 0.802 | 0.814 | |
| 10000 | 0.9 | 1 | gaussian | N/A | 0.791 | 0.812 | 0.809 | |
| 9000 | 0.75 | 1 | polynomial | 5 | 0.821 | 0.804 | 0.794 | |
| 1000 | 0.6 | 0 | polynomial | 5 | 0.789 | 0.721 | 0.759 | |
| 3000 | 0.65 | 1 | gaussian | N/A | 0.786 | 0.819 | 0.806 | |
| 2000 | 0.8 | 0 | polynomial | 2 | 0.724 | 0.721 | 0.723 | |
| 7000 | 0.7 | 1 | polynomial | 2 | 0.832 | 0.801 | 0.801 | |
| 4000 | 0.65 | 1 | polynomial | 3 | 0.809 | 0.802 | 0.812 | |
| 1000 | 0.9 | 0 | polynomial | 5 | 0.741 | 0.736 | 0.779 | |
| 6000 | 0.75 | 0 | linear | N/A | 0.784 | 0.736 | 0.734 | |
| 4000 | 0.8 | 1 | polynomial | 4 | 0.826 | 0.809 | 0.829 | |
| 7000 | 0.9 | 1 | linear | N/A | 0.821 | 0.791 | 0.771 | |
| 4000 | 0.9 | 0 | gaussian | N/A | 0.731 | 0.766 | 0.766 | |
| 3000 | 0.7 | 1 | linear | N/A | 0.791 | 0.759 | 0.786 | |
| 4000 | 0.75 | 1 | gaussian | N/A | 0.806 | 0.807 | 0.781 | |
| 2000 | 0.75 | 1 | linear | N/A | 0.809 | 0.789 | 0.784 | |
| 1000 | 0.9 | 1 | polynomial | 2 | 0.806 | 0.817 | 0.791 | |
| 9000 | 0.8 | 0 | linear | N/A | 0.749 | 0.726 | 0.746 | |
| 2000 | 0.7 | 1 | gaussian | N/A | 0.799 | 0.784 | 0.824 | |
| 8000 | 0.75 | 0 | gaussian | N/A | 0.746 | 0.749 | 0.781 | |
| 5000 | 0.7 | 1 | linear | N/A | 0.816 | 0.799 | 0.779 | |
| 1000 | 0.85 | 0 | polynomial | 2 | 0.728 | 0.719 | 0.731 | |
| 9000 | 0.85 | 1 | polynomial | 4 | 0.836 | 0.817 | 0.789 | |
| 1000 | 0.7 | 0 | polynomial | 4 | 0.739 | 0.739 | 0.754 | |
| 5000 | 0.7 | 0 | linear | N/A | 0.761 | 0.779 | 0.749 | |
| 4000 | 0.85 | 1 | linear | N/A | 0.799 | 0.797 | 0.811 | |
| 1000 | 0.85 | 0 | linear | N/A | 0.761 | 0.711 | 0.764 | |
| 2000 | 0.95 | 1 | polynomial | 3 | 0.809 | 0.829 | 0.814 | |
| 8000 | 0.85 | 1 | linear | N/A | 0.799 | 0.824 | 0.796 | |
| 5000 | 0.65 | 0 | gaussian | N/A | 0.749 | 0.759 | 0.756 | |
| 3000 | 0.85 | 0 | linear | N/A | 0.754 | 0.734 | 0.719 | |
| 10000 | 0.9 | 0 | gaussian | N/A | 0.749 | 0.761 | 0.751 | |
| 9000 | 0.6 | 1 | gaussian | N/A | 0.834 | 0.809 | 0.799 | |
| 7000 | 0.8 | 1 | polynomial | 2 | 0.822 | 0.791 | 0.806 | |
| 6000 | 0.8 | 0 | linear | N/A | 0.749 | 0.736 | 0.764 | |
| 1000 | 0.8 | 1 | gaussian | N/A | 0.791 | 0.804 | 0.804 | |
| 10000 | 0.75 | 0 | gaussian | N/A | 0.746 | 0.696 | 0.734 | |
| 7000 | 0.85 | 0 | polynomial | 4 | 0.754 | 0.761 | 0.779 | |
| 9000 | 0.65 | 1 | gaussian | N/A | 0.804 | 0.809 | 0.804 | |
| 6000 | 0.65 | 0 | linear | N/A | 0.731 | 0.724 | 0.736 | |
| 9000 | 0.85 | 0 | gaussian | N/A | 0.761 | 0.736 | 0.736 | |
| 10000 | 0.95 | 0 | linear | N/A | 0.759 | 0.704 | 0.741 | |
| 5000 | 0.7 | 0 | polynomial | 5 | 0.741 | 0.764 | 0.776 | |
| 1000 | 0.95 | 1 | linear | N/A | 0.811 | 0.789 | 0.771 | |
| 7000 | 0.8 | 1 | linear | N/A | 0.804 | 0.774 | 0.774 | |
| 9000 | 0.75 | 0 | linear | N/A | 0.746 | 0.759 | 0.739 | |
| 8000 | 0.9 | 1 | linear | N/A | 0.804 | 0.784 | 0.809 | |
| 4000 | 0.95 | 1 | linear | N/A | 0.806 | 0.814 | 0.801 | |
| 4000 | 0.7 | 0 | polynomial | 5 | 0.754 | 0.759 | 0.779 | |
| 6000 | 0.65 | 1 | linear | N/A | 0.844 | 0.827 | 0.784 | |
| 10000 | 0.85 | 0 | linear | N/A | 0.766 | 0.749 | 0.716 | |
| 7000 | 0.75 | 0 | linear | N/A | 0.716 | 0.693 | 0.771 | |
| 9000 | 0.7 | 1 | polynomial | 4 | 0.796 | 0.794 | 0.811 | |
| 9000 | 0.85 | 0 | polynomial | 3 | 0.723 | 0.756 | 0.764 | |
| 7000 | 0.95 | 0 | polynomial | 2 | 0.723 | 0.751 | 0.726 | |
| 4000 | 0.95 | 0 | gaussian | N/A | 0.774 | 0.764 | 0.786 | |
| 6000 | 0.8 | 1 | polynomial | 2 | 0.809 | 0.806 | 0.786 | |
| 9000 | 0.7 | 1 | polynomial | 5 | 0.824 | 0.822 | 0.801 | |
| 9000 | 0.8 | 1 | gaussian | N/A | 0.834 | 0.806 | 0.816 | |
| 9000 | 0.65 | 1 | polynomial | 3 | 0.829 | 0.801 | 0.799 | |
| 9000 | 0.6 | 0 | gaussian | N/A | 0.769 | 0.769 | 0.749 | |
| 1000 | 0.65 | 0 | polynomial | 3 | 0.754 | 0.739 | 0.774 | |
| 10000 | 0.65 | 1 | polynomial | 4 | 0.819 | 0.794 | 0.779 | |
| 7000 | 0.9 | 0 | polynomial | 2 | 0.759 | 0.708 | 0.776 | |
| 7000 | 0.65 | 0 | polynomial | 2 | 0.726 | 0.731 | 0.764 | |
| 4000 | 0.6 | 1 | linear | N/A | 0.814 | 0.769 | 0.817 | |
| 7000 | 0.65 | 1 | polynomial | 2 | 0.794 | 0.807 | 0.804 | |
| 10000 | 0.7 | 1 | gaussian | N/A | 0.796 | 0.817 | 0.794 | |
| 1000 | 0.75 | 1 | linear | N/A | 0.781 | 0.779 | 0.784 | |
| 4000 | 0.95 | 1 | polynomial | 4 | 0.819 | 0.809 | 0.827 | |
| 6000 | 0.6 | 0 | linear | N/A | 0.761 | 0.776 | 0.774 | |
| 5000 | 0.95 | 1 | polynomial | 2 | 0.811 | 0.766 | 0.814 | |
| 9000 | 0.8 | 1 | linear | N/A | 0.819 | 0.817 | 0.832 | F3 |
| 7000 | 0.6 | 1 | gaussian | N/A | 0.794 | 0.832 | 0.806 | |
| 1000 | 0.7 | 0 | gaussian | N/A | 0.738 | 0.741 | 0.724 | |
| 2000 | 0.8 | 1 | linear | N/A | 0.801 | 0.799 | 0.794 | |
| 4000 | 0.6 | 0 | linear | N/A | 0.764 | 0.746 | 0.754 | |
| 1000 | 0.95 | 1 | linear | N/A | 0.736 | 0.782 | 0.756 | |
| 1000 | 0.75 | 0 | linear | N/A | 0.761 | 0.734 | 0.741 | |
| 3000 | 0.9 | 1 | gaussian | N/A | 0.821 | 0.794 | 0.809 | |
| 7000 | 0.9 | 0 | linear | N/A | 0.718 | 0.716 | 0.746 | |
| 9000 | 0.75 | 1 | polynomial | 3 | 0.796 | 0.774 | 0.819 | |
| 4000 | 0.75 | 0 | polynomial | 5 | 0.726 | 0.746 | 0.759 | |
| 10000 | 0.95 | 0 | polynomial | 3 | 0.764 | 0.756 | 0.769 | |
| 2000 | 0.95 | 0 | polynomial | 4 | 0.769 | 0.771 | 0.779 | |