

Evaluating the robustness of ResNet18 and Bag of Visual Words

Federico Arenas López and Tiago Lé

INTRODUCTION

Research in pattern recognition and computer vision was catalysed by the introduction of ImageNet (Deng et al., 2009), a large-scale data set for image classification. However, multiple image recognition algorithms have been studied since the field of computer vision was born (Huang, 1996), and more recently, with the advent of Deep Neural Networks and Convolutional Neural Networks (CNNs), these have become the norm for image recognition tasks (Voulodimos et al., 2018).

Nonetheless, before CNNs, various algorithms proved highly performing for representing images for image recognition tasks. These include SIFT (Lowe, 1999), SURF (Bay et al., 2008), ORB (Rublee et al., 2011), and many others (Nixon and Aguado, 2019). To this day, some of these methods have been proven more computationally efficient and faster at recognition than CNNs (LeCun et al., 1995).

During this study, we focus on comparing the performance of a ResNet18 CNN architecture with the performance of a Bag of Visual Words (BoVW) method that uses Support Vector Machines (SVMs) for a binary image classification task. The binary classification task consists in correctly classifying an image from a provided dataset that contains 1888 images of dogs, and 1200 images of cats. We will train and validate both models using a 3-fold cross-validation strategy, from which we will fine-tune our models and report the mean accuracy of both models on the 3 unseen test sets that resulted from the 3-fold split of our data.

To further our study, we analyse the impact of adding noise to the test images on the performance of both models. This is formalised by performing 9 different types of noise perturbations on each test set, and doing this for 10 increasing levels of noise. Both models are tested on the noisy images, and we record the average accuracy for each level of noise, for each type of noise perturbation. This in turn allows us to understand the sensitivity of both models to given types of perturbations and their intensities.

We conclude our study by analysing and comparing the results from both models under different noise perturbations, and by suggesting possible improvements to make to our models to increase robustness under such perturbations.

DATA SET AND TASK

We will perform a classification task on a data set consisting of images of cats and dogs. This data set contains a total of 2388 images of 224×224 pixels in 3 channels (RGB) split into 2 classes: cats (1188 images) and dogs (1200 images). Each class is split into 12 breeds with different number of images. The number of images for each breed is presented in Table 1.

The data set is not completely balanced, since the number of images for each class is different. In addition to this, the number of images per breed also varies. However, this imbalance is not significant enough to cause bias in the models. Regardless, in the following Section we will explain how we dealt with this minor class imbalance problem.

METHODOLOGY

In this section we go over the different models used for this task as well as the methods used to train them, considering the small class imbalance problem.

ResNet18

ResNets were introduced by He et al. (2016) as a method to tackle the difficulties of training deep neural networks, more specifically, the *vanishing gradients problem* (for certain activation functions, gradients are small, and therefore the weights remain with very small values, disrupting the learning process) and the *degradation problem* (with increasing

SubClass	Cats	n	Dogs	n
	Breeds		Breeds	
1	Abyssinian	99	American Bulldog	100
2	Bengal	98	American Pitbull Terrier	100
3	Birman	100	Basset Hound	100
4	Bombay	100	Beagle	100
5	British Shorthair	100	Boxer	100
6	Egyptian Mau	92	Chihuahua	100
7	Maine Coon	100	English Cocker Spaniel	100
8	Persian	100	English Setter	100
9	Ragdoll	99	German Shorthaired	100
10	Russian Blue	100	Great Pyrenees	100
11	Siamese	100	Havanese	100
12	Sphynx	100	Japanese Chin	100
Total		1188		1200

Table 1. Number of images per breed.

network depth, the accuracy gets saturated and degrades rapidly). They hypothesised that it would be easier to optimise the residual mapping than to optimise the original, unreferenced mapping.

The main idea is to explicitly let groups of stacked layers fit a residual mapping, instead of expecting them to fit the desired underlying mapping. This can be stated more formally: denoting the desired underlying mapping as $\mathcal{H}(\mathbf{x})$, the stacked non-linear layers now fit $\mathcal{F}(\mathbf{x}) = \mathcal{H}(\mathbf{x}) - \mathbf{x}$ and the original mapping is recast into $\mathcal{F}(\mathbf{x}) + \mathbf{x}$. This formulation of the original mapping can be realized by feedforward neural networks with shortcut connections that skip one or more layers and perform identity mapping and their outputs are added to the outputs of the stacked layers. The identity shortcut connections do not add extra parameters or computational complexity.

The ResNet Architecture is based on the VGG Architecture, first introduced by Simonyan and Zisserman (2014). It is composed of k stages, each composed by b convolutional blocks. Each convolutional block is composed of two convolution "same" operations each followed by a non-linearity. Each stage S_k is interfaced by one convolutional block with a pooling layer that halves the size of the inputs from S_k , and feeds them to the following stage S_{k+1} . There is one convolutional layer for the input sample, and one fully connected layer for the output from this sample.

For this task, we used the Pytorch ¹ implementation of [ResNet18](#). This model has 5 layers and expects mini-batches of 3-channel RGB images of shape $(3 \times 224 \times 224)$ since it was pre-trained on ImageNet. In order to adapt the model to this task, we had to re-train it so that it classified only cats and dogs. This was done by freezing all the parameters in the model as this guarantees that the gradients are not computed and back propagated. No data augmentation was performed.

Bag of Visual Words

Bag of Visual Words (BoVW) is an image category classification process in which the images are represented by histograms of visual word occurrences that are used to train the classifier.

Using the training set, a visual vocabulary is created by extracting feature descriptors from the images in each category using the SIFT feature detection algorithm and clustering the descriptors using the K-Means clustering algorithm. Each of the created cluster centers represents a visual word. Finally, for each image, a frequency histogram of the vocabularies and their frequency in the image is created. These histograms correspond to the BoVW and are then used to train a Support Vector Machine (SVM) classifier. Figure 1 and 2 summarise this process using a sample image.

Cross-validation

In order to completely eliminate the class imbalance problem and to conduct the hyperparameter search, we performed 3-fold cross-validation.

¹PyTorch is an optimised tensor library for deep learning using GPUs and CPUs, introduced by Paszke et al. (2019)

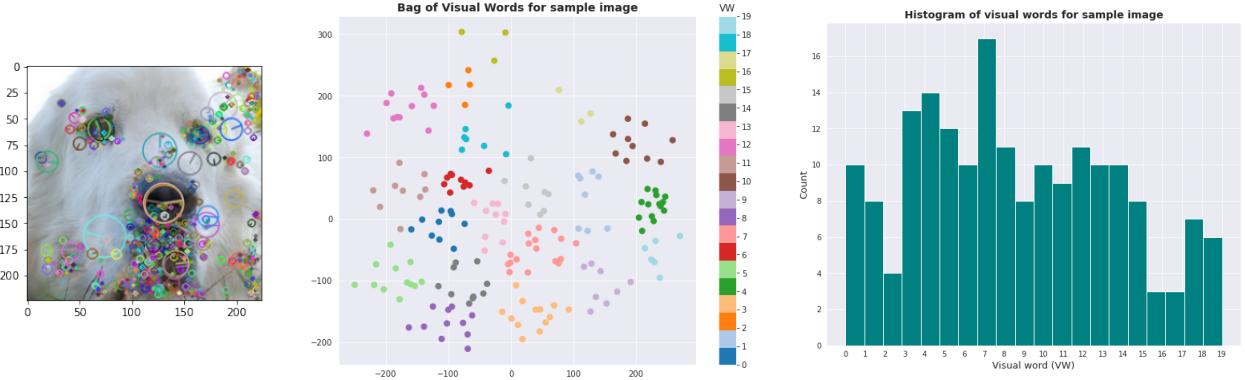


Figure 1. BoVW: *Left.* Visualization of SIFT features on a sample image. *Center.* Scatter plot of the features projected into a 2D space, and then clustered using a K-means algorithm for 20 clusters. *Right.* Histogram representation of the SIFT descriptors of the image on the left, this histogram representation can now be used with a classification algorithm such as SVM, along with the histogram representation of the rest of the images in the data set.

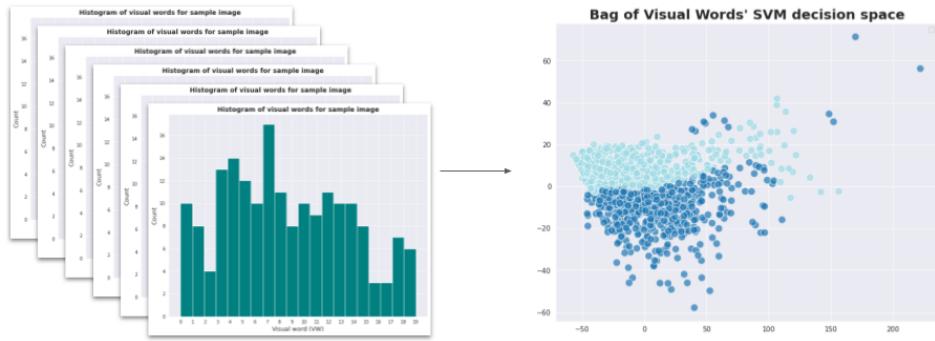


Figure 2. BoVW: All images are turned into histogram Bag of Visual Words histogram representations (left) and then projected into a vector space (right). The Support Vector Machine model then creates a decision boundary in the vector space. The figure on the right corresponds to the classification of an SVM trained on the histograms of the left. We can see that there is a clear decision boundary, where the light points correspond to cats, and the dark points correspond to dogs.

First, the data was split into 3 equal parts (A, B and C), taking into account the number of images in each class, as well as the number of images in each breed, and created three splits of the form (train set, test set): (A+B, C), (A+C, B) and (B+C, A). Then, in order to conduct hyperparameter search, each train set was split into 75% training and 25% validation, resulting in train split and validation split. For each set of hyperparameters, a model was trained and validated on each split and the ones that lead to the best validation performance across all splits were picked. Using these parameters, three models were trained on the original train sets (A+B, A+C and B+C). Finally, we test these models on the corresponding test set and consider the average testing loss and accuracy on all splits.

EXPERIMENTS

In order to study the effectiveness of the two aforementioned methods, we began by conducting a hyperparameter search by training and validating on each split. We then trained the models on each split with the best set of hyperparameters. To test the effectiveness of each method, we tested the trained models on the corresponding split's test set. Finally, to study the robustness of the trained models, we applied a variety of perturbations to the images in the test sets and evaluated the models' performances. This is explained in detail in the rest of the section.

Hyperparameter search

We use the gridsearch method to conduct our hyperparameter search for both the ResNet18 and the BoVW models.

Since the ResNet18 was a pre-trained model that only needed to be fine-tuned for this task, there was no need to conduct an extensive hyperparameter search. For this reason, we simply varied the initial learning rate of the models to take the values $\{0.01, 0.05, 0.1\}$. The validation loss on each split for each learning rate is presented in Figure 3. The validation loss and accuracy averaged on all splits for each learning rate is presented in Figure 4. We also represent the standard deviation to better illustrate the different behaviour of the three models.

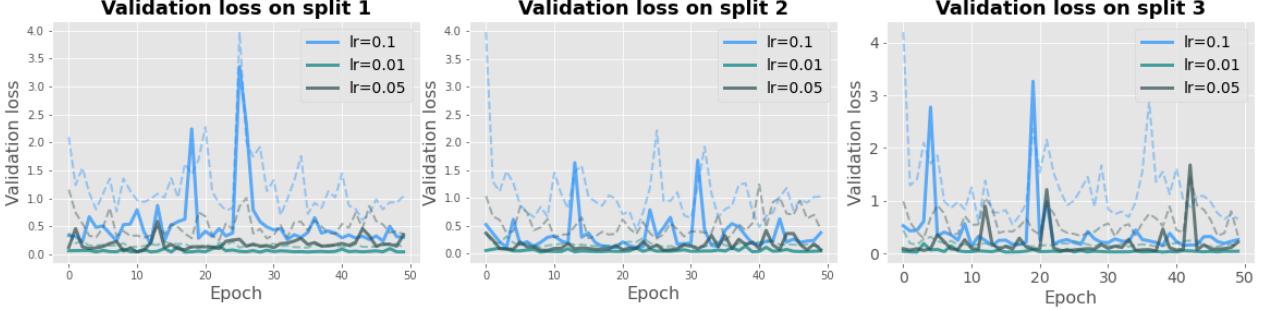


Figure 3. ResNet18: Validation loss from the three splits during training for 50 epochs. The dashed, lighter line corresponds to the training results during 50 epochs. From the three plots we can see that the models with $lr=0.01$ consistently outperform the rest of the models.

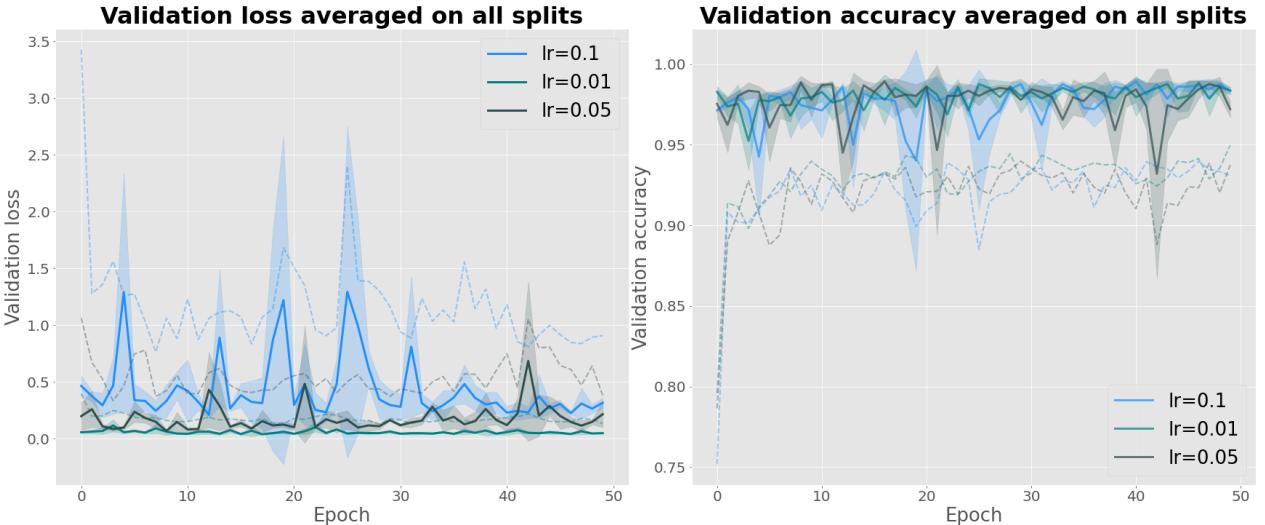


Figure 4. ResNet18: Validation loss and accuracy averaged over the three splits sampled for 50 epochs of training. The dashed line corresponds to the loss and accuracy results obtained during training of the models, for the three learning rates. The shadowed contour corresponds to the standard deviation of the average. We can see a significant variation for high learning rates, consistent with the effect of a large learning step.

From these results, we concluded that the best initial learning rate was 0.01. The test results obtained from training the three full training sets using this learning rate are presented in Table 2. We note that the validation loss is significantly higher than the training loss. This is due to the significantly smaller size of the test set, compared to the training set, and the fact that the ResNet18 model has been pre-trained on a much larger dataset.

On the other hand, the BoVW models were trained from scratch. This motivated a more exhaustive hyperparameter search: the number of clusters k produced by the K-Means algorithm, the type of kernel being used by the SVM classifier and the value of the regularisation parameter c being used by SVM classifier. The different values for each of the hyperparameters are presented in Table 3. The results for different values of k and c using the rbf kernel on the train and validation sets are presented in Figure 5.

Split No.	Training Loss	Validation Loss	Training Accuracy	Test Accuracy	Epoch
1	0.17	0.021	0.94	0.99	47
2	0.17	0.021	0.94	0.99	48
3	0.17	0.046	0.93	0.99	40
Avg.	0.17	0.029	0.94	0.99	

Table 2. ResNet18: Test accuracy and loss final results for the three ResNet18 models for the three splits with an initial learning rate of 0.01.

Hyperparameter	Values
k	50, 100, 150, 200, 250, 300
$kernel$	<i>linear, poly, rbf</i>
c	$0.5 + 2i, i \in \{0, 1, \dots, 99\}$

Table 3. BoVW: Values taken by each hyperparameter in the gridsearch.

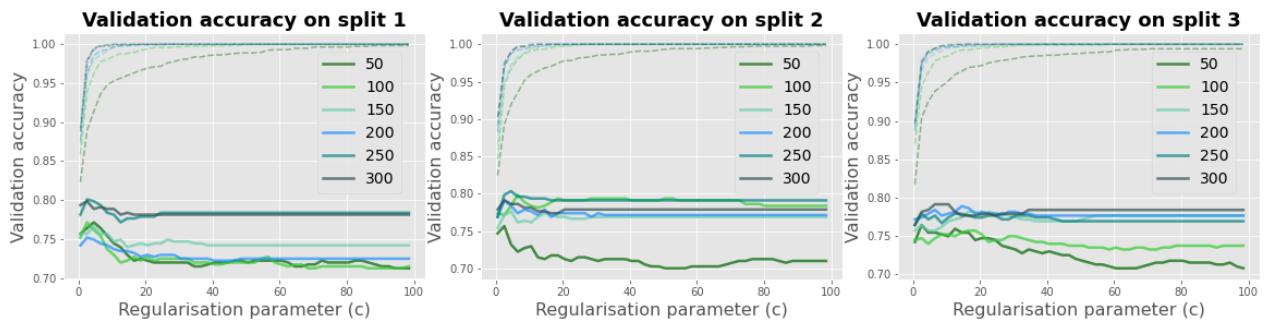


Figure 5. BoVW: Validation loss from the three splits for different values of c and using the rbf kernel in the SVM classifier. Each line corresponds to the results obtained using a different k .

We concluded that the best set of hyperparameters are $k = 250$, rbf kernel and $c = 2.5$. The test results obtained from training the three full training sets using these hyper parameters are presented in Table 4.

Split No.	Training Accuracy	Validation Accuracy	Test Accuracy
1	0.98	0.80	0.81
2	0.98	0.80	0.80
3	0.98	0.78	0.83
Avg.	0.98	0.79	0.81

Table 4. BoVW: Train, validation and test accuracy final results for the three BoVW models for the three splits with $k = 250$, rbf kernel and $c = 2.5$.

Robustness

In order to evaluate the robustness of the trained models to different types and levels of noise in the testing data, we use the best set of hyperparameters we discovered for each model.

In order to obtain a baseline performance, we evaluated the test performance on testing data with no noise. The average test accuracy for the ResNet18 models was 99% and for the BoVW models was 81%.

The different perturbations introduced to the data are described below.

Gaussian pixel noise

To each pixel in each colour channel, we added a Gaussian distributed random number with 10 increasing standard deviations from $\{0, 2, 4, 6, 8, 10, 12, 14, 16, 18\}$. We clipped the values to be in the range $0..255$ ².

²Negative numbers were replaced by 0 and values > 255 were replaced by 255.

The effects of this perturbation are presented in Figure 6.



Figure 6. Effects of applying Gaussian pixel noise with $std = \{0, 10, 18\}$.

Gaussian blurring

Each image was convolved [0 – 10] times with the following 3×3 mask:

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

This corresponds to an approximation of Gaussian blurring with increasingly larger standard deviations. The effects of this perturbation are presented in Figure 7.

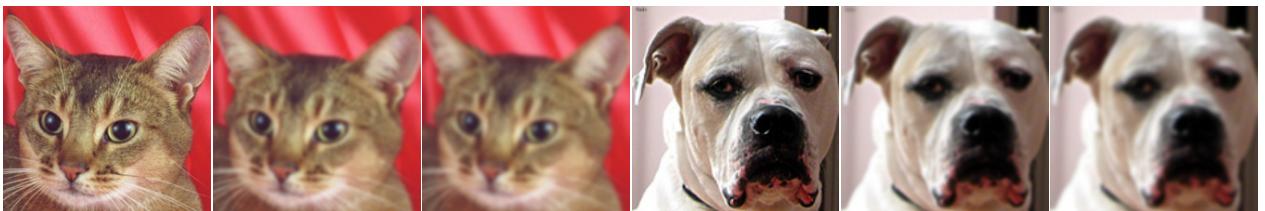


Figure 7. Effects of convolving the image $\{0, 5, 10\}$ times.

Image contrast increase

Each channel was multiplied by $\{1.0, 1.03, 1.06, 1.09, 1.12, 1.15, 1.18, 1.21, 1.24, 1.27\}$ and the values were clipped to be in the range 0..255.

Since we are increasing the difference between pixel values, this corresponds to increasing the image contrast. The effects of this perturbation are presented in Figure 8.

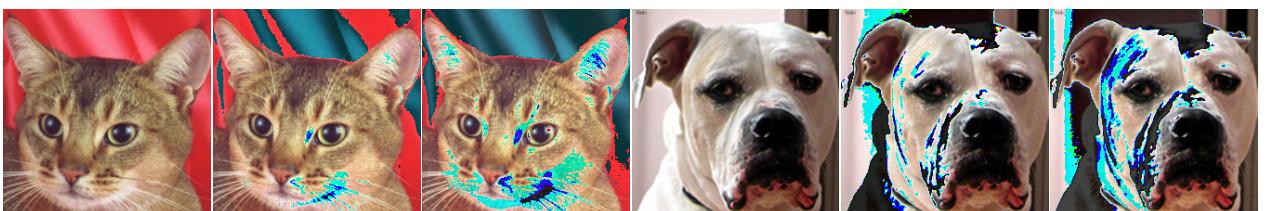


Figure 8. Effects of increasing the image contrast by a factor of $\{1.0, 1.15, 1.27\}$.

Image contrast decrease

Each channel was multiplied by $\{1.0, 0.90, 0.80, 0.70, 0.60, 0.50, 0.40, 0.30, 0.20, 0.10\}$ and the values were clipped to be in the range 0..255.

Since we are decreasing the difference between pixel values, this corresponds to decreasing the image contrast. The effects of this perturbation are presented in Figure 9.



Figure 9. Effects of decreasing the image contrast by a factor of $\{1.0, 0.5, 0.1\}$.

Image brightness increase

To each channel we added $\{0, 5, 10, 15, 20, 25, 30, 35, 40, 45\}$ and clipped the values to be in the range 0..255.

Since we are making all pixel values closer to 255 (white), this corresponds to increasing the image brightness. The effects of this perturbation are presented in Figure 10.



Figure 10. Effects of increasing the image brightness with intensity $\{0, 25, 45\}$.

Image brightness decrease

To each channel we subtracted $\{0, 5, 10, 15, 20, 25, 30, 35, 40, 45\}$ and clipped the values to be in the range 0..255.

Since we are making all pixel values closer to 0 (black), this corresponds to decreasing the image brightness. The effects of this perturbation are presented in Figure 11.



Figure 11. Effects of decreasing the image brightness with intensity $\{0, 25, 45\}$.

HSV Hue noise increase

The images were converted from RGB to HSV. We then added Gaussian random noise with zero mean and standard deviation $\{0.00, 0.02, 0.04, 0.06, 0.08, 0.10, 0.12, 0.14, 0.16, 0.18\}$ to the Hue component (first channel). Since the HSV values were in a 0..255 scale, the Gaussian noise was multiplied by 255 and the values were clipped to remain in the range 0..255. Finally, the images were converted back from HSV to RGB.

The effects of this perturbation are presented in Figure 12.

HSV Saturation noise increase

The images were converted from RGB to HSV. We then added Gaussian random noise with zero mean and standard deviation $\{0.00, 0.02, 0.04, 0.06, 0.08, 0.10, 0.12, 0.14, 0.16, 0.18\}$ to the Saturation component (second channel). Since the HSV values were in a 0..255 scale, the Gaussian noise was multiplied by 255 and the values were clipped to remain in the range 0..255. Finally, the images were converted back from HSV to RGB.

The effects of this perturbation are presented in Figure 13.



Figure 12. Effects of applying Gaussian noise with $std = \{0, 0.10, 0.18\}$ to the Hue channel.



Figure 13. Effects of applying Gaussian noise with $std = \{0, 0.10, 0.18\}$ to the Saturation channel.

Occlusion of the image increase

We replaced a randomly placed square region of each image with black pixels with square edge length of $\{0, 5, 10, 15, 20, 25, 30, 35, 40, 45\}$

The effects of this perturbation are presented in Figure 14.



Figure 14. Effects of occluding square regions with square edge length of $\{0, 25 \text{ and } 45\}$.

Robustness Experiments Results

The average testing accuracy for the ResNet18 and BoVW models on the perturbed testing data is presented in Figure 15 and on Table 5.

On the ResNet18 models, the Gaussian blurring and image occlusion lead to accuracy drops of only 1%, approximately. Changing the image contrast, increasing the image brightness and adding noise to the Hue channel lead to less than a 15% drop in accuracy. Adding Gaussian pixel noise, decreasing the image brightness and adding noise to the Saturation channel resulted in a drop in performance of more than 25%.

On the BoVW models, the addition of noise to the Hue channel and image occlusion lead to accuracy drops of less than 1%, approximately. Adding Gaussian pixel noise, increasing the image contrast, increasing the image brightness and adding noise to the Saturation channel lead to less than a 10% drop in the accuracy of the models. Gaussian blurring and the decrease of the image's brightness caused the accuracy to drop below 60%. Finally, the most noticeable performance drop was when the image contrast was decreased, causing the accuracy to drop to approximately 50%, reaching 39% when the contrast was decreased to 20%.

DISCUSSION

Overall, the ResNet18 models outperform the BoVW models. This was expected as the ResNet18 models were pre-trained on a larger and more complex data set and required only some fine-tuning, whereas the BoVW models

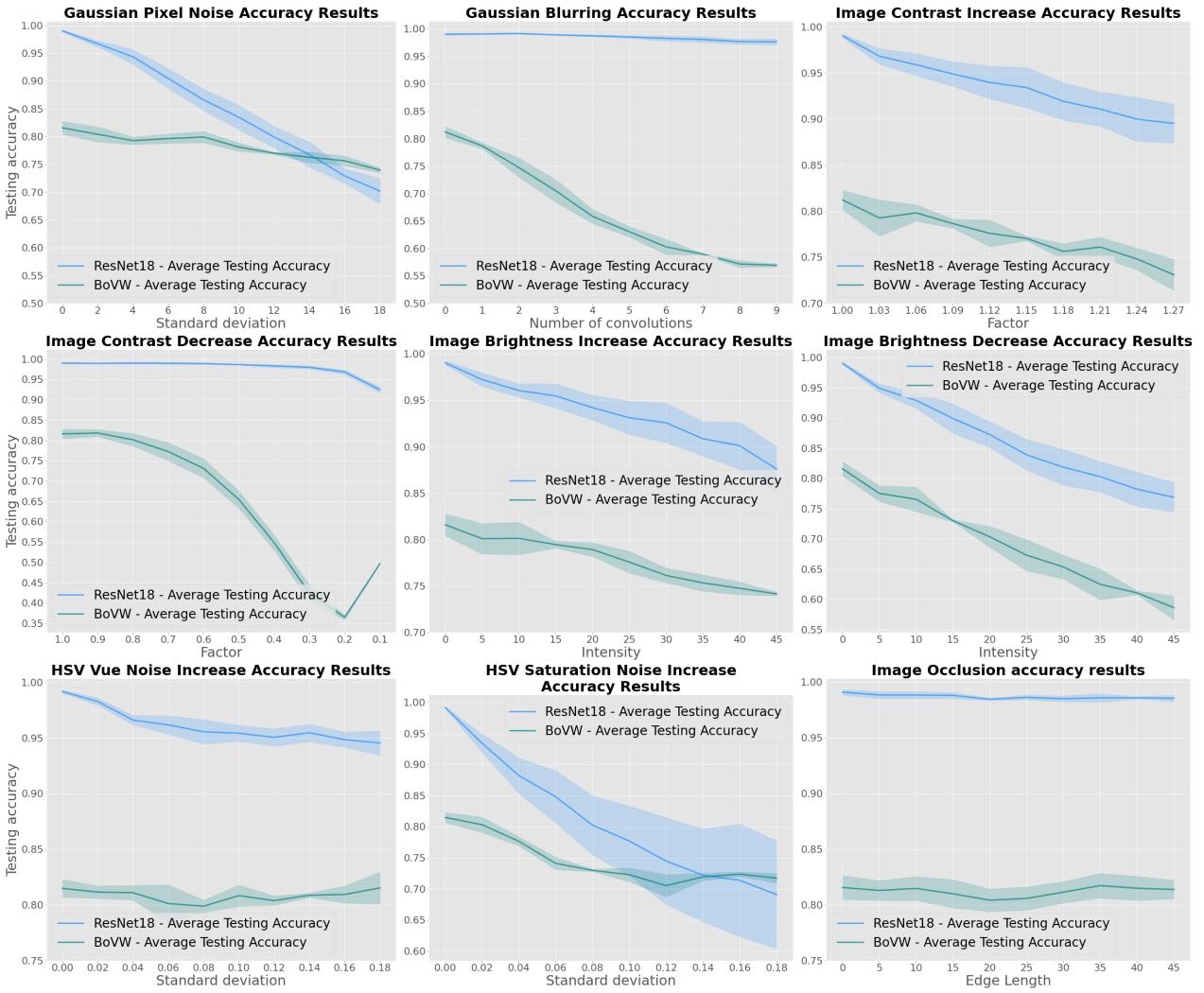


Figure 15. Average test accuracy of the ResNet18 and BoVW models on test sets incrementally perturbed by a given noise. From these results we can see that the model’s accuracy is highly affected by gaussian pixel noise. On the contrary, increasing the contrast of the image appears to improve the model’s performance at detecting cats and dogs.

had to be trained from scratch. This was visible in the better generalisation performance on unseen data achieved by the ResNet18 models. On the contrary, since the BoVW was trained only on this task’s data set, the validation and test performances were significantly hampered. This was to be expected, since Deep Learning methods currently comprehend the state-of-the-art for image recognition tasks Voulodimos et al. (2018).

This performance difference was also reflected on the robustness tests, where the ResNet18 models consistently achieved better results than the BoVW models, with two exceptions in Gaussian pixel noise and the HSV Saturation noise. With these perturbations, the performance drop was more intense in the ResNet18 models and ultimately lead to the BoVW outperforming them on the last two perturbation levels.

In general, we expected the performance to drop as the intensity of the perturbations increased. This was the case for all models in all types of perturbation with exception of the BoVW models on the last level of image contrast decrease, where we observe a small performance boost from the penultimate to the last level.

On the BoVW models, the most visible performance drop was caused by the decrease of image contrast. This was to be expected as the SIFT features are extracted by computing the image’s gradients. Since all the pixel values are now closer to 0, the gradients will be smaller and, therefore, closer to each other, making it harder for the classifier to

Perturbation	Method	0	1	2	3	4	5	6	7	8	9
<i>Gaussian pixel noise</i>	ResNet18	0.99	0.97	0.94	0.90	0.87	0.83	0.80	0.77	0.73	0.70
	BoVW	0.82	0.80	0.79	0.80	0.80	0.78	0.77	0.76	0.76	0.74
<i>Gaussian blurring</i>	ResNet18	0.99	0.99	0.99	0.99	0.99	0.99	0.98	0.98	0.98	0.98
	BoVW	0.81	0.79	0.75	0.71	0.66	0.63	0.60	0.59	0.57	0.57
<i>Image contrast increase</i>	ResNet18	0.99	0.97	0.96	0.95	0.94	0.93	0.92	0.91	0.90	0.90
	BoVW	0.81	0.79	0.80	0.79	0.78	0.77	0.76	0.76	0.75	0.73
<i>Image contrast decrease</i>	ResNet18	0.99	0.99	0.99	0.99	0.99	0.99	0.98	0.98	0.97	0.92
	BoVW	0.81	0.82	0.80	0.77	0.73	0.65	0.55	0.42	0.36	0.50
<i>Image brightness increase</i>	ResNet18	0.99	0.97	0.96	0.95	0.94	0.93	0.93	0.91	0.90	0.88
	BoVW	0.82	0.80	0.80	0.79	0.79	0.78	0.76	0.75	0.75	0.74
<i>Image brightness decrease</i>	ResNet18	0.99	0.95	0.93	0.90	0.87	0.84	0.82	0.80	0.78	0.77
	BoVW	0.81	0.78	0.77	0.73	0.70	0.67	0.65	0.62	0.61	0.59
<i>HSV Hue noise</i>	ResNet18	0.99	0.98	0.97	0.96	0.96	0.95	0.95	0.95	0.95	0.95
	BoVW	0.81	0.81	0.81	0.80	0.80	0.81	0.80	0.81	0.81	0.82
<i>HSV Saturation noise</i>	ResNet18	0.99	0.93	0.88	0.85	0.80	0.78	0.74	0.72	0.71	0.69
	BoVW	0.81	0.80	0.78	0.74	0.73	0.72	0.71	0.72	0.72	0.72
<i>Image occlusion</i>	ResNet18	0.99	0.99	0.99	0.99	0.98	0.99	0.98	0.99	0.99	0.99
	BoVW	0.81	0.81	0.81	0.81	0.80	0.81	0.81	0.82	0.81	0.81

Table 5. Average test accuracy of the ResNet18 and BoVW models on test sets for the different levels of the different perturbations introduced in the data.

distinguish the images later in the classification process.

Another aspect worth mentioning is the training, validation and testing times. As expected, training time was significantly higher for the ResNet18 models, as they were fine-tuned for 50 epochs on the task's data set during training time. On the contrary, and surprisingly, the validation and testing times were higher for the BoVW models, since the SIFT features for each image had to be extracted, clustered with K-Means, where the distance to each point needs to be calculated in order to find the image's cluster, and the histograms need to be created. Finally, the images still need to be classified with the SVM. This process is undoubtedly slower than the ResNet18's testing process. This is because for testing, we can easily load the pretrained ResNet18 models and directly evaluate it on the complete test set.

CONCLUSION

The best models on our cats and dogs task were the ResNet18 models fine-tuned for 50 epochs with a learning rate of 0.01. Despite the longer training time, its validation and testing times were lower than for the BoVW models and with better results. In addition to this, the ResNet18 models were more robust to the perturbations on the test set, except when noise was added to the Vue and Saturation channels.

The lower performance of the BoVW was expected, as we merely adopted a vanilla implementation of the SIFT algorithm. Using other feature extraction algorithms, such as Dense Scale Invariant Feature Transform (DSIFT) or ORB, should improve the models' performances and reduce training, validation and testing times.

In general, both the ResNet18 and BoVW models were robust to the perturbations on the testing data, with accuracy dropping less than 10% at the last level of most perturbations. Applying data augmentation in the training data should lead to an increase in generalisation performance, as well as an improvement in their robustness.

REFERENCES

- Bay, H., Ess, A., Tuytelaars, T., and Van Gool, L. (2008). Speeded-up robust features (surf). *Computer vision and image understanding*, 110(3):346–359.
- Deng, J., Dong, W., Socher, R., Li, L.-J., Li, K., and Fei-Fei, L. (2009). Imagenet: A large-scale hierarchical image database. In *2009 IEEE conference on computer vision and pattern recognition*, pages 248–255. Ieee.
- He, K., Zhang, X., Ren, S., and Sun, J. (2016). Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 770–778.
- Huang, T. (1996). Computer vision: Evolution and promise.
- LeCun, Y., Bengio, Y., et al. (1995). Convolutional networks for images, speech, and time series. *The handbook of brain theory and neural networks*, 3361(10):1995.
- Lowe, D. G. (1999). Object recognition from local scale-invariant features. In *Proceedings of the seventh IEEE international conference on computer vision*, volume 2, pages 1150–1157. Ieee.
- Nixon, M. and Aguado, A. (2019). *Feature extraction and image processing for computer vision*. Academic press.
- Paszke, A., Gross, S., Massa, F., Lerer, A., Bradbury, J., Chanan, G., Killeen, T., Lin, Z., Gimelshein, N., Antiga, L., et al. (2019). Pytorch: An imperative style, high-performance deep learning library. *arXiv preprint arXiv:1912.01703*.
- Rublee, E., Rabaud, V., Konolige, K., and Bradski, G. (2011). Orb: An efficient alternative to sift or surf. In *2011 International conference on computer vision*, pages 2564–2571. Ieee.
- Simonyan, K. and Zisserman, A. (2014). Very deep convolutional networks for large-scale image recognition. *arXiv preprint arXiv:1409.1556*.
- Voulodimos, A., Doulamis, N., Doulamis, A., and Protopapadakis, E. (2018). Deep learning for computer vision: A brief review. *Computational intelligence and neuroscience*, 2018.

APPENDIX

Directory Tree

```
├── README.md
├── catdog
│   ├── CATS
│   ├── DOGS
│   └── catdogs.csv
├── catdogs.csv
├── data
│   ├── full_split_1
│   ├── full_split_2
│   ├── full_split_3
│   ├── robustness
│   ├── split_1
│   ├── split_2
│   └── split_3
├── environment.yml
├── figs
│   ├── bovw
│   └── resnet18
├── full_split_val_1.csv
├── full_split_val_2.csv
├── helpers
│   ├── data_loader.py
│   └── perturb_images.py
├── models
│   ├── bovw.py
│   ├── notebooks
│   ├── resnet18.py
│   ├── run_bovw.py
│   └── run_resnet18.py
├── output
│   ├── bovw
│   └── resnet18
│       ├── resnet_data_analyser.ipynb
│       ├── resnet_robustness_tests.ipynb
│       ├── bovw_data_analyser.ipynb
│       ├── bovw_notebook.ipynb
│       └── split_data.ipynb
```

Figure 16. Directory tree of the project, containing the data, the models, and helper functions.

ResNet18 Models' Code

Listing 1. resnet18.py

```
1 # Library imports
2 from __future__ import print_function
3 from __future__ import division
4 import torch
5 import torch.nn as nn
6 import torch.optim as optim
7 import numpy as np
8 import pandas as pd
9 import torchvision
10 from torchvision import datasets, models, transforms
11 import matplotlib.pyplot as plt
12 import time
13 import os
14 import copy
15 #from tqdm.notebook import tqdm
16 from tqdm import tqdm
17
18 # Define input feature map size
19 INPUT_SIZE = 224
20
21 class ResNet18:
22     def __init__(self, on_split, input_data_dir = "./data",
23                  output_dir = "./output/resnet18",
24                  batch_size = 50,
25                  num_epochs = 50,
26                  num_classes = 2,
27                  training = True,
28                  pretrained_model = None,
29                  feature_extract = True,
30                  full_split = False,
31                  perturbation = ""):
32         ...
33
34     The ResNet18 class gets the data directory as input, as well as the hyperparameters for
35     training, and outputs the training progress and the trained model as a .pth file.
36     """
37     # Initialize global variables
38     self.full_split = full_split
39     if self.full_split:
40         self.data_dir = input_data_dir+"/full_split_"+str(on_split)+"/"
41         self.output_dir = output_dir+"/full_split_"+str(on_split)+"/"
42     else:
43         self.data_dir = input_data_dir+"/split_"+str(on_split)+"/"
44         self.output_dir = output_dir+"/split_"+str(on_split)+"/"
45     self.training = training
46     self.num_classes = num_classes
47     self.batch_size = batch_size
48     self.num_epochs = num_epochs
49     self.input_size = INPUT_SIZE
50
51     # Whether to use pretrained weights from ImageNet or not
52     self.feature_extract = feature_extract
53     self.pretrained_model = pretrained_model
54
55     def set_parameter_requires_grad(self, model, feature_extracting):
56         """
57         Receives the model and a the boolean feature_extracting variable, which if
58         set to True, uses the pretrained weights from ImageNet and updates the parameters of
59         the model accordingly.
60         """
61         if feature_extracting:
62             for param in model.parameters():
63                 param.requires_grad = False
64
65     def initialize_model(self, num_classes, feature_extract):
66         """
```

```

67     Receives the output number of classes , feature_extract , and the use_pretrained flag
68     and outputs the initialized model.
69     '',
70     # Initialize input size and model variable
71     model_ft = None
72     input_size = 0
73     # Initialize model
74     model_ft = models.resnet18(pretrained = feature_extract)
75
76     self.set_parameter_requires_grad(model_ft , feature_extract)
77     num_ftrs = model_ft.fc.in_features
78     model_ft.fc = nn.Linear(num_ftrs , num_classes)
79
80     return model_ft
81
82 def get_dataloaders(self , data_dir , bs , input_size):
83     '',
84     Takes the data directory as input , the batch size and the input size , and outputs
85     the torch dataloader containing the transformed images in batch of bs size.
86     '',
87     # Define the data transformations
88     data_transforms = {
89         'train': transforms.Compose([
90             transforms.RandomResizedCrop(input_size),
91             transforms.RandomHorizontalFlip(),
92             transforms.ToTensor(),
93             transforms.Normalize([0.485 , 0.456 , 0.406] , [0.229 , 0.224 , 0.225])
94         ]),
95         'val': transforms.Compose([
96             transforms.Resize(input_size),
97             transforms.CenterCrop(input_size),
98             transforms.ToTensor(),
99             transforms.Normalize([0.485 , 0.456 , 0.406] , [0.229 , 0.224 , 0.225])
100        ]),
101    }
102
103 if self.training:
104     # Create training and validation datasets
105     image_datasets = {x: datasets.ImageFolder(os.path.join(data_dir , x) , data_transforms[x]
106         ) for x in ['train' , 'val']}
107     # Create training and validation dataloaders
108     loader = {x: torch.utils.data.DataLoader(image_datasets[x] , batch_size=bs , shuffle=
109             True , num_workers=4) for x in ['train' , 'val']}
110 else:
111     # Create training and validation datasets
112     image_datasets = {x: datasets.ImageFolder(os.path.join(data_dir , x) , data_transforms[x]
113         ) for x in ['val']}
114     # Create training and validation dataloaders
115     loader = {x: torch.utils.data.DataLoader(image_datasets[x] , batch_size=bs , shuffle=
116             True , num_workers=4) for x in ['val']}
117
118     return loader
119
120 def train_model(self , model , dataloaders , criterion , optimizer , device , num_epochs=25):
121     '',
122     Takes a pretrained model , the dataloaders , the momentum criterium , the optimizer , the
123     torch device , the number
124     of epochs and outputs the trained model with the progress information.
125     '',
126     # Start counting the training time
127     since = time.time()
128
129     # Initialize variables that will store the training information
130     train_acc_history = []
131     train_loss_history = []
132     val_acc_history = []
133     val_loss_history = []
134
135     # Get the best model weights from the inputted model
136     best_model_wts = copy.deepcopy(model.state_dict())

```

```

132     best_acc = 0.0
133
134     # Loop through epochs to start training
135     for epoch in range(num_epochs):
136         print('Epoch {}/{},'.format(epoch, num_epochs - 1))
137         print('-' * 10)
138
139         # Each epoch has a training and validation pass
140         if self.training:
141             phases = {'train': "Training network...", 'val': "Evaluating network..."}
142         else:
143             phases = {'val': "Evaluating network..."}
144
145         for phase in phases.keys():
146             if phase == 'train':
147                 model.train()          # Set model to training mode
148             else:
149                 model.eval()          # Set model to evaluate mode
150
151             running_loss = 0.0
152             running_corrects = 0
153
154             # Iterate over data.
155             print(phases[phase])
156             print(f"Looking at data in {self.data_dir}")
157             for inputs, labels in tqdm(dataloaders[phase]):
158                 inputs = inputs.to(device)
159                 labels = labels.to(device)
160
161                 # zero the parameter gradients
162                 optimizer.zero_grad()
163
164                 # forward
165                 # track history if only in train
166                 with torch.set_grad_enabled(phase == 'train'):
167                     # Get model outputs and calculate loss
168                     outputs = model(inputs)
169                     loss = criterion(outputs, labels)
170
171                     _, preds = torch.max(outputs, 1)
172
173                     # Backpropagation + optimize only if in training phase
174                     if phase == 'train':
175                         loss.backward()
176                         optimizer.step()
177
178                     # statistics
179                     running_loss += loss.item() * inputs.size(0)
180                     running_corrects += torch.sum(preds == labels.data)
181
182             epoch_loss = running_loss / len(dataloaders[phase].dataset)
183             epoch_acc = running_corrects.double() / len(dataloaders[phase].dataset)
184
185             print('{} Loss: {:.4f} Acc: {:.4f}'.format(phase, epoch_loss, epoch_acc))
186
187             # deep copy the model
188             if phase == 'val' and epoch_acc > best_acc:
189                 best_acc = epoch_acc
190                 best_model_wts = copy.deepcopy(model.state_dict())
191
192             if phase == 'val':
193                 val_acc_history.append(epoch_acc.numpy())
194                 val_loss_history.append(epoch_loss)
195             else:
196                 train_acc_history.append(epoch_acc.numpy())
197                 train_loss_history.append(epoch_loss)
198
199             # Calculate total training time
200             time_elapsed = time.time() - since
201             print('\nTraining complete in {:.0f}m {:.0f}s'.format(time_elapsed // 60, time_elapsed % 60))

```

```

201     #print('Best val Acc: {:.4f}'.format(best_acc))
202
203     # load best model weights
204     model.load_state_dict(best_model_wts)
205
206     return model, val_acc_history, val_loss_history, train_acc_history, train_loss_history,
207             best_acc.numpy(), time_elapsed
208
209 def run(self, lr=0.01, momentum=0.9, hyp_name = ""):
210     print("Initializing Datasets and Dataloaders ...")
211     # Get cross-validation data
212     loader = self.get_dataloaders(self.data_dir, self.batch_size, self.input_size)
213
214     # Detect if we have a GPU available
215     device = torch.device("cuda:0" if torch.cuda.is_available() else "cpu")
216
217     # Initialize the model for this run
218     model_ft = self.initialize_model(self.num_classes, self.feature_extract)
219
220     # Send the model to GPU
221     model_ft = model_ft.to(device)
222
223     # Initialize optimizer
224     params_to_update = model_ft.parameters()
225
226     if self.feature_extract:
227         params_to_update = []
228         for name, param in model_ft.named_parameters():
229             if param.requires_grad == True:
230                 params_to_update.append(param)
231     else:
232         for name, param in model_ft.named_parameters():
233             if param.requires_grad == True:
234                 pass
235
236     # If testing load information from pretrained model
237     if not self.training:
238         model_ft.load_state_dict(self.pretrained_model)
239
240     # Define optimizer
241     optimizer_ft = optim.SGD(params_to_update, lr=lr, momentum=0.9)
242
243     # Setup the loss function
244     criterion = nn.CrossEntropyLoss()
245
246     # Train model
247     model_ft, val_acc_history, val_loss_history, train_acc_history, train_loss_history,
248             best_acc, time_elapsed = self.train_model(
249                 model_ft, loader, criterion, optimizer_ft, device, num_epochs=self.num_epochs)
250
251     # Record data
252     training_output = {}
253
254     # Output data
255     root_output_dir = self.output_dir
256     if self.training:
257         training_output['train_loss'] = train_loss_history
258         training_output['train_acc'] = train_acc_history
259         training_output['val_loss'] = val_loss_history
260         training_output['val_acc'] = val_acc_history
261         training_output['best_acc'] = [best_acc]*self.num_epochs
262         training_output['runtime(s)'] = [time_elapsed]*self.num_epochs
263         df_name = root_output_dir+"progress/train-progress"+hyp_name+".csv"
264         print(f"Outputting data to {df_name}...")
265         pd.DataFrame.from_dict(training_output).to_csv(df_name, index=False)
266         output_w_dir = root_output_dir+"weights/trained_model"+hyp_name+".pth"
267         torch.save(model_ft.state_dict(), output_w_dir)
268         print(f"Saving final trained model to {output_w_dir}")
269     else:
270         training_output['val_loss'] = val_loss_history

```

```

269     training_output['val_acc'] = val_acc_history
270     training_output['best_acc'] = [best_acc]*self.num_epochs
271     training_output['runtime(s)'] = [time_elapsed]*self.num_epochs
272     df_name = root_output_dir+"progress/test_progress.csv"
273     #print(f"Outputting data to {df_name}...")
274     #pd.DataFrame.from_dict(training_output).to_csv(df_name, index=False)
275     print(f"Test accuracy of {best_acc}")
276     return best_acc
277
278
279 #-----#
280 #                                     MAIN FUNCTION
281 #-----#
282
283 if __name__ == "__main__":
284     # Load pre-trained model
285     pretrained = torch.load("./output/resnet18/split_1/weights/trained_model.pth")
286     # Instantiate network
287     resnet = ResNet18(1, "./data", "./output/resnet18/",
288                       batch_size=50, num_epochs=1,
289                       num_classes=2, training=False, pretrained_model=pretrained,
290                       feature_extract=True)
291
292     # Train ResNet18
293     resnet.run()

```

Listing 2. run_resnet18.py

```

1 from resnet18 import *
2
3 # Fine-tune models with learning rate
4 def train_hyperparameters():
5     lrs = [0.01, 0.05, 0.1]
6     for split in range(1, 4):
7         print(f"##### On split {split} #####")
8         for lr in lrs:
9             print(f"Learning rate = {lr}")
10            # Instantiate network
11            resnet = ResNet18(split, "./data", "./output/resnet18/",
12                              batch_size=50, num_epochs=50,
13                              num_classes=2, training=True, pretrained_model=None,
14                              feature_extract=True)
15
16            # Train ResNet18
17            lr_name = "_lr_" + str(lr).replace('.', '_')
18            resnet.run(lr=lr, hyp_name=lr_name)
19
20    def train_full_splits():
21        # Training on lr=0.01 for all full splits
22        for split in range(1, 4):
23            print(f"##### On split {split} #####")
24            # Instantiate network
25            resnet = ResNet18(split, "./data", "./output/resnet18",
26                              batch_size=50, num_epochs=50,
27                              num_classes=2, training=True, pretrained_model=None, feature_extract=
28                              True, full_split=True)
29
30            # Train ResNet18
31            resnet.run(lr=0.01)
32
33    def evaluate_test_splits():
34        # Load pre-trained model
35        for split in range(1, 4):
36            print(f"## On Split {split} ##")
37            pretrained = torch.load("./output/resnet18/full_split_" + str(split) + "/weights/trained_model
38 .pth")
39            # Instantiate network
40            input_data_dir = "./data/robustness/5_3/3"
41            resnet = ResNet18(1, input_data_dir, "./output/resnet18",
42                             batch_size=50, num_epochs=1,

```

```

40             num_classes=2, training=False, pretrained_model=pretrained,
41             feature_extract=True, full_split=True)
42
43     # Train ResNet18
44     best_acc = resnet.run()
45
46     print(f"From Split {split}, the best accuracy was {best_acc}")
47
48 def create_dir(directory):
49     if not os.path.exists(directory):
50         os.makedirs(directory)
51
52 def create_placeholder_dirs():
53     # Create root directory
54     robustness_output_path = "./output/resnet18/robustness"
55     create_dir(robustness_output_path)
56     # Create list of sub-directory names
57     perturb_ids = list(range(1,10))
58     # Create all directories
59     print("Creating directories ...")
60     for perturb_id in tqdm(perturb_ids):
61         for split in [1,2,3]:
62             # Create perturbation id folder
63             full_path = robustness_output_path+ "/5_"+str(perturb_id)+"/full_split_"+str(split)+"/"
64             create_dir(full_path)
65
66 def evaluate_robustness():
67     # Create list of sub-directory names
68     perturb_ids = list(range(1,10))
69     # Perturbation levels
70     perturb_levels = list(range(1,11))
71     # Load pre-trained model
72     for split in tqdm(range(3,4)):
73         print(f"### On Split {split} ###")
74         pretrained = torch.load("./output/resnet18/full_split_"+str(split)+"/weights/trained_model.pth")
75         # Instantiate network
76         if split == 3:
77             perturbations = [3]
78         else:
79             perturbations = perturb_ids
80         for perturb_id in perturbations:
81             # Initialize list to store accuracies
82             best_accs = []
83             results_dict = {}
84             # Get id folder name
85             id_fname = "/5_"+str(perturb_id)
86             output_data_dir = "./output/resnet18/robustness/"+id_fname
87             for pertur_level in perturb_levels:
88                 print(f"On perturbation level {pertur_level}, of perturbation id {perturb_id}")
89                 input_data_dir = "./data/robustness"+id_fname+"/"+str(pertur_level)
90                 resnet = ResNet18(split, input_data_dir, output_data_dir,
91                                   batch_size=50, num_epochs=1,
92                                   num_classes=2, training=False, pretrained_model=pretrained,
93                                   feature_extract=True, full_split=True)
94
95                 # Train ResNet18
96                 best_acc = resnet.run()
97                 best_accs.append(best_acc)
98
99                 # Store results
100                results_dict['perturb_level'] = perturb_levels
101                results_dict['best_accs'] = best_accs
102                results_df = pd.DataFrame.from_dict(results_dict)
103                # Ouput results to .csv file
104                csv_fname = output_data_dir+"/full_split_"+str(split)+"/robustness_results_5_"+str(
105                  perturb_id)+".csv"
106                results_df.to_csv(csv_fname, index=False)

```

```

107
108 | if __name__ == "__main__":
109 |     evaluate_robustness()

```

BoVW Models' Code

Listing 3. boww.py

```

1 # Library import
2 import numpy as np
3 import cv2
4 import os
5 import pandas as pd
6 import matplotlib.pyplot as plt
7 from scipy import ndimage
8 from scipy.spatial import distance
9 from sklearn.cluster import KMeans
10 from sklearn.utils import shuffle
11 from sklearn.svm import SVC
12 from sklearn.pipeline import make_pipeline
13 from sklearn.preprocessing import StandardScaler
14 from tqdm import tqdm
15
16 # Define BoVW class
17 class BoVW:
18     def __init__(self, split_n, data_dir, full_split = False):
19         """
20             This class defines the Bag of Visual Words model
21         """
22         self.split_n = split_n
23         self.full_split = full_split
24         if self.full_split:
25             self.data_dir = data_dir + "/full_splits"
26         else:
27             self.data_dir = data_dir + "/split_"
28
29     def get_splits(self, split_n):
30         """
31             Receives the number of the split split_n and outputs the training and validation
32             pandas dataframes.
33         """
34         if self.full_split:
35             path = self.data_dir + str(split_n) + '/full_splits' + str(split_n) + ".csv"
36         else:
37             path = self.data_dir + str(split_n) + '/split_' + str(split_n) + ".csv"
38
39         train_df = pd.read_csv(path + 'train.csv')
40         val_df = pd.read_csv(path + 'val.csv')
41
42         return train_df, val_df
43
44     def get_valid_splits(self, split_n):
45         """
46             Receives the number of the split split_n and outputs the validation
47             pandas dataframes.
48         """
49         if self.full_split:
50             path = self.data_dir + str(split_n) + '/full_splits_val' + str(split_n) + ".csv"
51         else:
52             path = self.data_dir + str(split_n) + '/split_' + str(split_n) + ".csv"
53
54         val_df = pd.read_csv(path)
55
56         return val_df
57
58     def image_reader(self, dataframe):
59         """
60             Receives the dataframe that points to the images, and outputs a dictionary
61             with all images loaded with OpenCV.

```

```

62     """
63     image_dict = {}
64     file_locations = list(dataframe['image_id'])
65     labels = list(dataframe['label'])
66     category_0 = []
67     category_1 = []
68     for i in range(len(file_locations)):
69         image = cv2.imread(file_locations[i], cv2.COLOR_RGB2BGR)
70         try:
71             image = cv2.cvtColor(image, cv2.COLOR_RGB2BGR)
72         except: # if the image is gray
73             image = cv2.cvtColor(image, cv2.COLOR_GRAY2RGB)
74
75         if labels[i] == 0:
76             category_0.append(image)
77         else:
78             category_1.append(image)
79
80     image_dict[0] = category_0
81     image_dict[1] = category_1
82
83     return image_dict
84
85 def sift_features(self, images):
86     """
87     Creates descriptors using sift. Takes one parameter that is images dictionary. Return an
88     array whose first
89     index holds the descriptor_list without an order and the second index holds the
90     sift_vectors dictionary which
91     holds the descriptors but this is seperated class by class.
92     """
93     sift_vectors = {}
94     descriptor_list = []
95     sift = cv2.xfeatures2d.SIFT_create()
96     for key, value in images.items():
97         features = []
98         for img in value:
99             kp, des = sift.detectAndCompute(img, None)
100            try:
101                descriptor_list.extend(des)
102                features.append(des)
103            except:
104                print(f"Faulty descriptor found : {des}")
105            sift_vectors[key] = features
106
107     return descriptor_list, sift_vectors
108
109 def kmeans(self, k, descriptor_list):
110     """
111     A k-means clustering algorithm who takes 2 parameter which is number
112     of cluster(k) and the other is descriptors list(unordered 1d array)
113     Returns an array that holds central points.
114     """
115     kmeans = KMeans(n_clusters = k, n_init=10, verbose=0)
116     kmeans.fit(descriptor_list)
117     vwords = kmeans.cluster_centers_
118
119     return kmeans, vwords
120
121 def find_index(self, image, center):
122     """
123     Receives an image and a cluster center and returns to which image does the center belong
124     as and index
125     """
126     count = 0
127     ind = 0
128     for i in range(len(center)):
129         if(i == 0):
130             count = distance.euclidean(image, center[i])
131         else:

```

```

130         dist = distance.euclidean(image, center[i])
131         if(dist < count):
132             ind = i
133             count = dist
134     return ind
135
136 # Extract visual words from descriptors
137 def get_histograms_dictio(self, sift_vectors, kmeans_centers):
138     """
139     Receives the keypoints and the cluster centers and returns
140     a dictionary with all the histograms, divided by class
141     """
142     dict_feature = {}
143     for key,value in tqdm(sift_vectors.items()):
144         print(f"Getting histograms of class {key}...")
145         category = []
146         for img in tqdm(value):
147             histogram = np.zeros(len(kmeans_centers))
148             for each_feature in img:
149                 ind = self.find_index(each_feature, kmeans_centers)
150                 histogram[ind] += 1
151                 category.append(histogram)
152
153         dict_feature[key] = category
154     return dict_feature
155
156 # Map dictionary of histograms to np arrays
157 def get_histogram_arrays(self, sift_vectors, kmeans_centers):
158     """
159     Receives the keypoints and cluster centers and returns the np.array of the
160     training data and the training labels
161     """
162     histogram_dictio = self.get_histograms_dictio(sift_vectors, kmeans_centers)
163     X = []
164     Y = []
165     print("Converting to np.arrays...")
166     for key in histogram_dictio.keys():
167         for value in histogram_dictio[key]:
168             X.append(value)
169             Y.append(key)
170
171     return np.array(X), np.array(Y)
172
173 def train_svm(self, visual_words, labels, c, k_):
174     """
175     Takes the visual words, the labels, the complexity ter c, and the type of
176     kernel and outputs a trained svm classifier
177     """
178     # Get training data
179     X_train = visual_words
180     Y_train = labels
181
182     # Initialize SVM classifier
183     svc_classifier = make_pipeline(StandardScaler(), SVC(C=c, kernel=k_, gamma='auto'))
184     svc_classifier.fit(X_train, Y_train)
185
186     return svc_classifier
187
188 def get_train_val_dict(self, split_n):
189     """
190     # Get split data
191     print('Convert dataframe to dictionary of images')
192     split_n = self.split_n
193     train_splits, val_splits = self.get_splits(split_n)
194
195     # Get split images
196     print(f'Get split {split_n} data')
197     train_dict = self.image_reader(train_splits)
198     val_dict = self.image_reader(val_splits)
199
200     return train_dict, val_dict

```

```

200
201
202     def get_all_histograms(self, K_clusters, train_dict, val_dict):
203         """
204             Receives the number of clusters to perform kmeans on the data, and returns nothing
205             but outputs a .npy file with the training and validation histograms for the current
206             data split
207         """
208         # Get descriptors and keypoints from split
209         print('Get full sift features for training data')
210         train_descriptor_list, train_sift_vectors = self.sift_features(train_dict)
211         print('Get full sift features for validation data... ')
212         val_descriptor_list, val_sift_vectors = self.sift_features(val_dict)
213
214         # Perform kmeans training to get visual words
215         K = K_clusters
216         print('Perform clustering on training data... ')
217         train_kmeans, train_centers = self.kmeans(K_clusters, train_descriptor_list)
218
219         # Get histograms and save them
220         print("Get histograms from kmeans clustering")
221         train_histograms, train_classes = self.get_histogram_arrays(train_sift_vectors,
222                         train_centers)
223         print("Get validation histograms from kmeans clustering")
224         val_histograms, val_classes = self.get_histogram_arrays(val_sift_vectors, train_centers)
225         if not self.full_split:
226             np.save("output/boww/split_"+str(self.split_n)+"/histograms/train/
227                 train_visual_words_k_"+str(K_clusters)+".npy", train_histograms)
228             np.save("output/boww/split_"+str(self.split_n)+"/histograms/train/train_classes_k_"+
229                 str(K_clusters)+".npy", train_classes)
230             np.save("output/boww/split_"+str(self.split_n)+"/histograms/val/val_visual_words_k_"+
231                 str(K_clusters)+".npy", val_histograms)
232             np.save("output/boww/split_"+str(self.split_n)+"/histograms/val/val_classes_k_"+
233                 str(K_clusters)+".npy", val_classes)
234
235         else:
236             np.save("output/boww/full_split_"+str(self.split_n)+"/histograms/train/
237                 train_visual_words_k_"+str(K_clusters)+".npy", train_histograms)
238             np.save("output/boww/full_split_"+str(self.split_n)+"/histograms/train/
239                 train_classes_k_"+str(K_clusters)+".npy", train_classes)
240             np.save("output/boww/full_split_"+str(self.split_n)+"/histograms/train/
241                 train_centers_k_"+str(K_clusters)+".npy", train_centers)
242             np.save("output/boww/full_split_"+str(self.split_n)+"/histograms/val/
243                 val_visual_words_k_"+str(K_clusters)+".npy", val_histograms)
244             np.save("output/boww/full_split_"+str(self.split_n)+"/histograms/val/val_classes_k_"+
245                 str(K_clusters)+".npy", val_classes)

```

Listing 4. run_boww.py

```

1  # Import boww class
2  from boww import *
3  from tqdm import tqdm
4
5  def retrieve_histograms():
6      # Define number of clusters and split to go through
7      clusters = [50, 100, 150, 200, 250, 300]
8      splits = [1,2,3]
9
10     for split in tqdm(splits):
11         print(f"Retrieving histograms from Split {split}...")
12         boww = BowW(split, "./data")
13         print("Getting data... ")
14         train_dict, val_dict = boww.get_train_val_dict(split)
15         if split == 1:
16             print("On left overs from split 1...")
17             for c in [250]:
18                 print("On left overs from Split 1, cluster", c)
19                 # Get all histograms for all clusters
20                 boww.get_all_histograms(c, train_dict, val_dict)
21             continue

```

```

22     for cluster in clusters:
23         print(f"On cluster {cluster}...")
24         # Initialize BovW model
25         # Get all histograms for all clusters
26         bovw.get_all_histograms(cluster, train_dict, val_dict)
27
28 def retrieve_accuracies():
29     # Define hyperparameter's value range
30     c_range = np.arange(0.5, 100.5, 2)
31     clusters = [50, 100, 150, 200, 250, 300]
32     kernels = ['linear', 'poly', 'rbf']
33     # Run through splits and clusters
34     for split in tqdm(range(1,4)):
35         # Loop through kernels
36         for k in tqdm(kernels):
37             # Define place holder for accuracie values
38             df_train_dict = {}
39             df_val_dict = {}
40             # Loop through clusters
41             for cluster in tqdm(clusters):
42                 # Load data
43                 train_histograms = np.load("output/boww/split_"+str(split)+"/histograms/train/
44                                         train_visual_words_k_"+str(cluster)+".npy")
45                 train_classes = np.load("output/boww/split_"+str(split)+"/histograms/train/
46                                         train_classes_k_"+str(cluster)+".npy")
47                 val_histograms = np.load("output/boww/split_"+str(split)+"/histograms/val/
48                                         val_visual_words_k_"+str(cluster)+".npy")
49                 val_classes = np.load("output/boww/split_"+str(split)+"/histograms/val/
50                                         val_classes_k_"+str(cluster)+".npy")
51                 # Cluster accuracies
52                 train_accuracies = []
53                 val_accuracies = []
54                 # Loop through c values
55                 c_vals = []
56                 for c in tqdm(c_range):
57                     c_vals.append(c)
58                     # Get SVM classifier
59                     bovw = BovW(split, "./data")
60                     svm_classifier = bovw.train_svm(train_histograms, train_classes, c, k)
61                     # Get train and val accuracies
62                     train_acc = svm_classifier.score(train_histograms, train_classes)
63                     val_acc = svm_classifier.score(val_histograms, val_classes)
64                     # Append accuracy values
65                     train_accuracies.append(train_acc)
66                     val_accuracies.append(val_acc)
67
68                     # Store accuracies in dictionary
69                     df_train_dict['c'] = c_vals
70                     df_train_dict[str(cluster)] = train_accuracies
71                     df_val_dict['c'] = c_vals
72                     df_val_dict[str(cluster)] = val_accuracies
73
74                     # Store dataframes
75                     df_train = pd.DataFrame.from_dict(df_train_dict)
76                     df_val = pd.DataFrame.from_dict(df_val_dict)
77
78                     # Save dataframes
79                     df_train.to_csv("./output/boww/split_"+str(split)+"/svm_results/train_acc_"+str(k)+".csv")
80                     df_val.to_csv("./output/boww/split_"+str(split)+"/svm_results/val_acc_"+str(k)+".csv")
81
82 def train_full_splits(best_cluster_n):
83     # Define number of clusters and split to go through
84     splits = [1,2,3]
85
86     for split in tqdm(splits):
87         print(f"Retreiving histograms from Split {split}...")
88         bovw = BovW(split, "./data", full_split=True)
89         print("Getting data...")
90         train_dict, val_dict = bovw.get_train_val_dict(split)

```

```

87     print("Getting histograms...")
88     bovw.get_all_histograms(best_cluster_n, train_dict, val_dict)
89
90 def evaluate_robustness(best_c, best_kernel):
91     # Create list of sub-directory names
92     perturb_ids = list(range(1,10))
93     # Perturbation levels
94     perturb_levels = list(range(1,11))
95     # Load pre-trained model
96     for split in tqdm(range(1,4)):
97         print(f"### On Split {split} ###")
98         # Get trained histograms
99         train_histograms = np.load("./output/bovw/full_split_"+str(split)+"/histograms/train/" +
100             "train_visual_words_k_250.npy")
101         train_classes = np.load("./output/bovw/full_split_"+str(split)+"/histograms/train/" +
102             "train_classes_k_250.npy")
103         train_centers = np.load("./output/bovw/full_split_"+str(split)+"/histograms/train/" +
104             "train_centers_k_250.npy")
105
106         # Instantiate network
107         # Skip level 5_4 since something is happening with Nones, we can debug it ad the end
108         for perturb_id in perturb_ids[4:]:
109             #for perturb_id in [4]:
110                 # Initialize list to store accuracies
111                 best_accs = []
112                 results_dict = {}
113                 # Get id folder name
114                 id_fname = "/5_"+str(perturb_id)
115                 output_data_dir = "./output/bovw/robustness/"+id_fname+"/"
116                 # Loop through perturbation levels
117                 for pertur_level in perturb_levels:
118                     #for pertur_level in [7]:
119                         print(f"On perturbation level {pertur_level}, of perturbation id {perturb_id}")
120                         input_data_dir = "./data/robustness"+id_fname+"/"+str(pertur_level)
121                         bovw = BoW(split, input_data_dir, full_split=True)
122                         # Get validation information from perturbed images
123                         val_df = bovw.get_valid_splits(split)
124                         print("Getting validation images...")
125                         val_dict = bovw.image_reader(val_df)
126                         val_descriptor_list, val_sift_vectors = bovw.sift_features(val_dict)
127                         val_histograms, val_classes = bovw.get_histogram_arrays(val_sift_vectors,
128                             train_centers)
129                         # Train SVM classifier on perturbed images
130                         svm_classifier = bovw.train_svm(train_histograms, train_classes, best_c,
131                             best_kernel)
132                         # Get train and val accuracies
133                         train_acc = svm_classifier.score(train_histograms, train_classes)
134                         val_acc = svm_classifier.score(val_histograms, val_classes)
135                         # Append accuracy values
136                         print(f"Accuracy of {val_acc} on 5-{perturb_id} of level {pertur_level}.")
137                         best_accs.append(val_acc)
138
139                         # Store results
140                         results_dict['perturb_level'] = perturb_levels
141                         results_dict['best_accs'] = best_accs
142                         results_df = pd.DataFrame.from_dict(results_dict)
143                         # Ouput results to .csv file
144                         csv_fname = output_data_dir+"full_split_"+str(split)+"/robustness_results_5_"+
145                             str(perturb_id)+".csv"
146                         print(f"Outputting appending data to {csv_fname}...")
147                         results_df.to_csv(csv_fname, index=False)
148
149 if __name__=="__main__":
150     evaluate_robustness(2.5, 'rbf')

```

Data Loader Code

Listing 5. data_loader.py

```

1 # Library imports
2 import os.path
3 from os import path
4 import numpy as np
5 import pandas as pd
6 from PIL import Image
7 import matplotlib.pyplot as plt
8
9 import torch
10 from torch.utils.data import Sampler
11 from torch.utils.data import Dataset
12 from torch.utils.data import DataLoader
13 from torch.utils.data import BatchSampler
14 from torchvision.transforms import transforms
15 from sklearn.model_selection import StratifiedKFold
16 from sklearn.model_selection import StratifiedShuffleSplit
17 from PIL import Image
18 from tqdm import tqdm
19 from shutil import copyfile
20 from sklearn.utils import shuffle
21
22 # Define data paths
23 DOGS_PATH = "./catdog/DOGS/"
24 CATS_PATH = "./catdog/CATS/"
25 DATA_PATH = "./catdogs.csv"
26
27
28 # Create csv file of all data paths
29 def create_csv():
30     dog_imgs = np.array([[ "./catdog/DOGS/" + name, 0] for name in os.listdir(DOGS_PATH) if os.path.isfile(os.path.join(DOGS_PATH, name))])
31     cat_imgs = np.array([[ "./catdog/CATS/" + name, 1] for name in os.listdir(CATS_PATH) if os.path.isfile(os.path.join(CATS_PATH, name))])
32     imgs = np.concatenate([dog_imgs, cat_imgs])
33
34     df = pd.DataFrame({"image_id": imgs[:, 0], "abc": np.full(imgs.shape[0], ""), "label": imgs[:, 1]})
35     df = df[df.image_id != DOGS_PATH + '.DS_Store'] # Remove the .DS_Store file
36     df = df[df.image_id != CATS_PATH + '.DS_Store'] # Remove the .DS_Store file
37
38     # Split each breed into 3 folds (A, B and C) with balanced breeds
39     classes = [('dog', DOGS_PATH), ('cat', CATS_PATH)]
40     breeds = np.arange(1, 13)
41     for animal, path in classes:
42         for breed in breeds:
43             breed_df = df[df.image_id.str.startswith(path + animal + '-{}-'.format(breed))]
44             num_images = breed_df.shape[0]
45             # Send first third of the breed to the A fold
46             for img_id in breed_df[:num_images//3].image_id:
47                 df.loc[df.image_id == img_id, 'abc'] = 'A'
48             # Send second third of the breed to the B fold
49             for img_id in breed_df[num_images//3:num.images * 2//3].image_id:
50                 df.loc[df.image_id == img_id, 'abc'] = 'B'
51             # Send last third of the breed to the C fold
52             for img_id in breed_df[num_images * 2//3:].image_id:
53                 df.loc[df.image_id == img_id, 'abc'] = 'C'
54
55     df.to_csv('catdogs.csv', index=False) # Save DataFrame as a csv file
56
57
58 def split(dataframe, full_split=False):
59     train_folds = [[ 'A', 'B'], [ 'A', 'C'], [ 'B', 'C']]
60
61     splits = {}
62     for i, train in enumerate(train_folds):
63         split = {}
64         split['train'] = dataframe[dataframe.abc.isin(train)]
65         split['test'] = dataframe[~dataframe.abc.isin(train)]
66
67         if full_split:

```

```

68         splits['full_split_{}'.format(i+1)] = split
69     else:
70         splits['split_{}'.format(i+1)] = split
71
72     return splits
73
74 # Divide each split's training data into train (75%) and validation (25%) with balanced breeds
75 def divide_train_val_test(dataframe, splits):
76     classes = [('dog', DOGS.PATH), ('cat', CATS.PATH)]
77     breeds = np.arange(1, 13)
78     for i in range(1, 4):
79         split_train = pd.DataFrame(
80             { "image_id": splits['split_{}'.format(i)]['train'].image_id,
81               "train_val_test": np.full(splits['split_{}'.format(i)]['train'].shape[0], ""),
82               "label": splits['split_{}'.format(i)]['train'].label
83             })
84         for animal, path in classes:
85             for breed in breeds:
86                 breed_df = split_train[split_train.image_id.str.startswith(path + animal + '_{}'.
87                     format(breed))]
88                 num_images = breed_df.shape[0]
89                 # 75% training data
90                 for img_id in breed_df[:num_images*3//4].image_id:
91                     split_train.loc[dataframe.image_id == img_id, 'train_val_test'] = 'train'
92                 # 25% validation data
93                 for img_id in breed_df[num_images*3//4:].image_id:
94                     split_train.loc[dataframe.image_id == img_id, 'train_val_test'] = 'val'
95
96         splits['split_{}'.format(i)]['train'] = split_train
97         splits['split_{}'.format(i)]['test'] = pd.DataFrame(
98             { "image_id": splits['split_{}'.format(i)]['test'].image_id,
99               "train_val_test": np.full(splits['split_{}'.format(i)]['test'].shape[0], "test"),
100              "label": splits['split_{}'.format(i)]['test'].label
101            })
102
103     return splits
104
105 # Changes test data to validation to be compatible with the Resnet's dataloaders
106 # There is no test set
107 def divide_train_val(dataframe, splits):
108     for i in range(1, 4):
109         splits['full_split_{}'.format(i)]['train'] = pd.DataFrame(
110             { "image_id": splits['full_split_{}'.format(i)]['train'].image_id,
111               "train_val_test": np.full(splits['full_split_{}'.format(i)]['train'].shape[0], "train"),
112               "label": splits['full_split_{}'.format(i)]['train'].label
113             })
114         splits['full_split_{}'.format(i)]['test'] = pd.DataFrame(
115             { "image_id": splits['full_split_{}'.format(i)]['test'].image_id,
116               "train_val_test": np.full(splits['full_split_{}'.format(i)]['test'].shape[0], "val"),
117               "label": splits['full_split_{}'.format(i)]['test'].label
118             })
119
120     return splits
121
122 # Create three splits ([train=A+B, test=C], [train=A+C, test=B], [train=B+C, test=A])
123 def create_splits(dataframe, full_split=False):
124     # Create splits
125     splits = split(dataframe, full_split)
126
127     if full_split:
128         splits = divide_train_val(dataframe, splits)
129     else:
130         splits = divide_train_val_test(dataframe, splits)
131
132     return splits

```

```

133 # Create directory if it does not exist
134 def create_dir(directory):
135     if not os.path.exists(directory):
136         os.makedirs(directory)
137
138
139 # Create directories to store the new splits in
140 def create_dirs(root_dir, split_names, classes):
141     # Create root directory
142     create_dir(root_dir)
143
144     for split_name in split_names:
145         # Create split directory
146         create_dir(root_dir+split_name)
147
148         # Create full_split directory
149         create_dir(root_dir+'full_'+split_name)
150
151         # Create train directory for the current split
152         create_dir(root_dir+split_name+'/train/')
153         # Create validation directory for the current split
154         create_dir(root_dir+split_name+'/val/')
155         # Create test directory for the current split
156         create_dir(root_dir+split_name+'/test/')
157
158         # Create train directory for the current full_split
159         create_dir(root_dir+'full_'+split_name+'/train/')
160         # Create validation directory for the current split
161         create_dir(root_dir+'full_'+split_name+'/val/')
162
163     for c in classes:
164         # Create class directories for the current split
165         create_dir(root_dir+split_name+'/train/'+c+'/')
166         create_dir(root_dir+split_name+'/val/'+c+'/')
167         create_dir(root_dir+split_name+'/test/'+c+'/')
168
169         # Create class directories for the current full_split
170         create_dir(root_dir+'full_'+split_name+'/train/'+c+'/')
171         create_dir(root_dir+'full_'+split_name+'/val/'+c+'/')
172
173 # Copy files from catdog directory to splits directory
174 def copy_files(splits, full_split=False):
175     # Initial information definition
176     root_dir = './data/'
177     split_names = splits.keys()
178     classes = ['dog', 'cat']
179     # Create directories, if necessary
180     if not os.path.exists(root_dir):
181         print("Creating data directories...")
182         create_dirs(root_dir, splits, classes)
183
184     print("Going through each split")
185     for split_name in tqdm(split_names):
186         # Copy files to train and validation sets
187         train_df_dict = {}
188         train_files = []
189         train_labels = []
190         val_df_dict = {}
191         val_files = []
192         val_labels = []
193         test_df_dict = {}
194         test_files = []
195         test_labels = []
196
197         for row in tqdm(splits[split_name]['train'].itertuples()):
198             # row = (index, image_id, train_val_test, label)
199             old_path = row[1]
200             file_name = row[1].split('/')[-1]
201             data_set = row[2]
202             class_name = classes[row[3]]

```

```

203     new_path = root_dir + split_name + '/' + data_set + '/' + classes[row[3]] + '/' +
204         file_name
205 #print(old_path, '\t\t', new_path)
206 copyfile(old_path, new_path)
207
208 # Store file names and labels
209 if data_set == 'train':
210     train_files.append(new_path)
211     train_labels.append(row[3])
212 else:
213     val_files.append(new_path)
214     val_labels.append(row[3])
215 for row in tqdm(splits[split_name]['test'].itertuples()):
216     # row = (index, image_id, train_val_test, label)
217     old_path = row[1]
218     file_name = row[1].split('/')[-1]
219     data_set = row[2]
220     class_name = classes[row[3]]
221     new_path = root_dir + split_name + '/' + data_set + '/' + classes[row[3]] + '/' +
222         file_name
223 #print(old_path, '\t\t', new_path)
224 copyfile(old_path, new_path)
225
226 # Store file names and labels
227 test_files.append(new_path)
228 test_labels.append(row[3])
229
230 # Create .csv with training file names and labels
231 train_df_dict['image_id'] = train_files
232 train_df_dict['label'] = train_labels
233 file_name = root_dir+'/'+split_name+'/'+split_name+'_'+train+'.csv'
234 pd.DataFrame.from_dict(train_df_dict).to_csv(file_name, index=False)
235
236 # Create .csv with validation file names and labels
237 if not full_split:
238     val_df_dict['image_id'] = val_files
239     val_df_dict['label'] = val_labels
240     file_name = root_dir+'/'+split_name+'/'+split_name+'_'+val+'.csv'
241     pd.DataFrame.from_dict(val_df_dict).to_csv(file_name, index=False)
242
243 # Create .csv with test file names and labels
244 test_df_dict['image_id'] = test_files
245 test_df_dict['label'] = test_labels
246 if full_split:
247     file_name = root_dir+'/'+split_name+'/'+split_name+'_'+val+'.csv'
248     pd.DataFrame.from_dict(test_df_dict).to_csv(file_name, index=False)
249 else:
250     file_name = root_dir+'/'+split_name+'/'+split_name+'_'+test+'.csv'
251     pd.DataFrame.from_dict(test_df_dict).to_csv(file_name, index=False)
252
253 # Create data splits
254 if __name__ == '__main__':
255     # Create .csv file with data location
256     create_csv()
257
258     # Retrieve data locations dataframe
259     print("Loading data...")
260     dataset = pd.read_csv('catdogs.csv')
261
262     # Create splits
263     print("Creating splits...")
264     splits = create_splits(dataset)
265     full_splits = create_splits(dataset, full_split=True)
266
267     # Create directories and copy files to directories
268     print("Copying files...")
269     copy_files(splits)
270     copy_files(full_splits, full_split=True)

```

Image Noise Adder Code

Listing 6. perturb_images.py

```
1 import numpy as np
2 from numpy.random import normal, randint
3 import cv2
4 import os
5 import pandas as pd
6 import matplotlib.pyplot as plt
7 from scipy import ndimage
8 from scipy.spatial import distance
9 from sklearn.cluster import KMeans
10 from sklearn.utils import shuffle
11 from sklearn.svm import SVC
12 from sklearn.pipeline import make_pipeline
13 from sklearn.preprocessing import StandardScaler
14 from tqdm import tqdm
15 from shutil import copyfile
16
17
18 class Noises:
19     def __init__(self):
20         """
21             This class processes the images from a file location
22             to output the same, noisier images in an output file location
23         """
24
25     def gaussian_pixel_noise(self, img, std):
26         """
27             This function takes an image and a standard deviation as input and outputs
28             the same image with noise added to it.
29         """
30         gauss = normal(0, std, img.shape).astype('uint8')
31         img_gauss = img+gauss
32         image = np.clip(img_gauss, 0, 255)
33
34     return image
35
36     def gaussian_blurring(self, image, n_convs):
37         """
38             This function takes an image as an input, and the number of convolutions
39             to apply to the image, and outputs a blurred image
40         """
41         self.convolution_kernel = 1/16 * np.array([[1, 2, 1], [2, 4, 2], [1, 2, 1]])
42         for conv in range(n_convs):
43             image = cv2.filter2D(image, -1, self.convolution_kernel)
44         return image
45
46     def increase_contrast(self, image, factor):
47         """
48             This function takes an image and a factor as an input, and outputs a contrasted
49             image according to the given factor
50         """
51         n_rows, n_columns, n_channels = image.shape
52         for channel in range(n_channels):
53             image[:, :, channel] = image[:, :, channel] * factor
54
55         image = np.clip(image, 0, 255)
56
57     return image
58
59     def decrease_contrast(self, image, factor):
60         """
61             Takes an image and a factor as input, and outputs an image with the contrast decreased
62             by the given factor
63         """
64         n_rows, n_columns, n_channels = image.shape
65         for channel in range(n_channels):
66             image[:, :, channel] = image[:, :, channel] * factor
```

```

67     image = np.clip(image, 0, 255)
68
69     return image
70
71 def increase_brightness(self, image, factor):
72     """
73     Takes an image and a factor as input, and outputs the same image with the brightness
74     increased
75     """
76     n_rows, n_columns, n_channels = image.shape
77     for channel in range(n_channels):
78         image[:, :, channel] += factor
79         image = np.clip(image, 0, 255)
80     return image
81
82 def decrease_brightness(self, image, factor):
83     """
84     Takes an image and a factor as input, and outputs the same image with the brightness
85     decreased
86     """
87     n_rows, n_columns, n_channels = image.shape
88
89     for channel in range(n_channels):
90         image[:, :, channel] -= factor
91         image = np.clip(image, 0, 255)
92     return image
93
94 def increase_hue_noise(self, image, std):
95     """
96     Takes an image as input and a standard deviation and returns an image with
97     it's hue noise increased
98     """
99     image = cv2.cvtColor(image, cv2.COLOR_BGR2HSV)
100    hue_noise = normal(0, std, image.shape[:-1])*255
101    image[:, :, 0] = image[:, :, 0] + hue_noise
102    image = np.clip(image, 0, 255)
103    hued_image = cv2.cvtColor(image, cv2.COLOR.HSV2BGR)
104    return hued_image
105
106 def increase_saturation_noise(self, image, std):
107     """
108     Receives an image as input and outputs the same image with increased
109     saturation
110     """
111     image = cv2.cvtColor(image, cv2.COLOR_BGR2HSV)
112     sat_noise = normal(0, std, image.shape[:-1])*255
113     image[:, :, 1] = image[:, :, 1] + sat_noise
114     image = np.clip(image, 0, 255)
115     sat_image = cv2.cvtColor(image, cv2.COLOR.HSV2BGR)
116     return sat_image
117
118 def image_occlusion(self, image, edge, print_center=False):
119     """
120     Receives an image and an edge and returns the same image
121     with occlusion
122     """
123     n_rows, n_columns, _ = image.shape
124     center = np.array([randint(0, n_rows+1), randint(0, n_columns+1)])
125     start_point = center - edge // 2
126     start_point = tuple((start_point[0], start_point[1]))
127     end_point = center + edge // 2
128     end_point = tuple((end_point[0], end_point[1]))
129     colour = (0, 0, 0)
130     thickness = -1
131     if print_center:
132         print(f"Center of the square: [{center[0]}, {center[1]}]")
133     image = cv2.rectangle(image, start_point, end_point, colour, thickness)
134
135     return image
136

```

```

137 | class PerturbImages(Noises):
138 |     def __init__(self, directories_created=True, source_path = "./data/full_split_"):
139 |         self.directories_created = directories_created
140 |         self.source_path = source_path # Path for images, add number according to split
141 |         self.robust_path = "./data/robustness"
142 |         self.source_path = source_path
143 |
144 |         self.perturb_ids = list(range(1,10))
145 |         self.perturb_levels = list(range(1,11))
146 |         self.splits = list(range(1,4))
147 |         self.classes = ['dog', 'cat']
148 |
149 |     def create_dir(self, directory):
150 |         if not os.path.exists(directory):
151 |             os.makedirs(directory)
152 |
153 |     def create_placeholder_dirs(self):
154 |         # Create root directory
155 |         self.create_dir(self.robust_path)
156 |         # Create list of sub-directory names
157 |
158 |         # Create all directories
159 |         print("Creating directories...")
160 |         for perturb_id in tqdm(self.perturb_ids):
161 |             self.create_dir(self.robust_path)
162 |             for perturb_level in self.perturb_levels:
163 |                 for split in self.splits:
164 |                     for class_ in self.classes:
165 |                         # Create perturbation id folder
166 |                         full_path = self.robust_path + '/5_1' + str(perturb_id) + '/' + str(perturb_level) +
167 |                         '/full_split_' + str(split) + '/val/' + class_
168 |                         self.create_dir(full_path)
169 |
170 |     def perform_perturbations(self):
171 |         # If placeholders have not been created, create placeholders
172 |         if not self.directories_created:
173 |             self.create_placeholder_dirs()
174 |
175 |         for split in tqdm(self.splits):
176 |             csv_path = self.source_path + str(split) + "/full_split_" + str(split) + "_val.csv"
177 |             split_df = pd.read_csv(csv_path)
178 |             image_paths = list(split_df['image_id'])
179 |             image_ids = list(split_df['label'])
180 |             print(f"Loading {len(image_paths)} images...")
181 |             for i, im_path in tqdm(enumerate(image_paths)):
182 |                 # Load current image
183 |                 original_image = cv2.imread(im_path)
184 |                 image_name = im_path.split('/')[-1]
185 |                 curr_class = self.classes[image_ids[i]]
186 |
187 |                 # Start perturbations of current image
188 |                 # Add gaussian noise
189 |                 #print(f"Adding noise...")
190 |                 self.add_all_gaussian_noise(original_image, "5_1", split, curr_class, image_name)
191 |                 # Add gaussian blur
192 |                 self.add_all_gaussian_blur(original_image, "5_2", split, curr_class, image_name)
193 |                 #print(f"Done with image {im_path}...")
194 |                 # Increase contrast
195 |                 self.add_all_increase_contrast(original_image, "5_3", split, curr_class,
196 |                                             image_name)
197 |                 #print(f"Done with image {im_path}...")
198 |                 # Decrease contrast
199 |                 self.add_all_decrease_contrast(original_image, "5_4", split, curr_class,
200 |                                             image_name)
201 |                 #print(f"Done with image {im_path}...")
202 |                 # Increase brightness
203 |                 self.add_all_increase_brightness(original_image, "5_5", split, curr_class,
204 |                                             image_name)
205 |                 #print(f"Done with image {im_path}...")
206 |                 # Decrease brightness

```

```

203     #self.add_all_decrease_brightness(original_image, "5_6", split, curr_class,
204         image_name)
205     #print(f"Done with image {im_path}...")
206     # Add hue noises
207     #self.add_all_hue_noises(original_image, "5_7", split, curr_class, image_name)
208     #print(f"Done with image {im_path}...")
209     # Add sat noises
210     #self.add_all_sat_noises(original_image, "5_8", split, curr_class, image_name)
211     # Add occlusions
212     #self.add_all_image_occlusions(original_image, "5_9", split, curr_class,
213         image_name)

214
215     def add_all_gaussian_noise(self, original_image, id_path, split, curr_class, image_name):
216         self.stds = np.arange(0, 19, 2)
217         for l, std in enumerate(self.stds):
218             image = np.array(original_image).copy()
219             image = self.gaussian_pixel_noise(image, std)
220             if curr_class == 'dog':
221                 new_im_path = self.robust_path+"/"+id_path+"/"+str(l+1)+"/full_split_"+str(split)+
222                     "/val/dog/"+image_name
223                 cv2.imwrite(new_im_path, image)
224                 print(f"Saving image to {new_im_path}")
225                 #copyfile(old_path, new_im_path)
226             else:
227                 new_im_path = self.robust_path+"/"+id_path+"/"+str(l+1)+"/full_split_"+str(split)+
228                     "/val/cat/"+image_name
229                 cv2.imwrite(new_im_path, image)
230                 print(f"Saving image to {new_im_path}")
231                 #copyfile(old_path, new_im_path)

232     def add_all_gaussian_blurr(self, original_image, id_path, split, curr_class, image_name):
233         num_convs = np.arange(10)
234         for l, n_convs in enumerate(num_convs):
235             image = np.array(original_image).copy() # This allows us to apply the same convolution
236                 to the blurred image
237             image = self.gaussian_blurring(image, n_convs)
238             if curr_class == 'dog':
239                 new_im_path = self.robust_path+"/"+id_path+"/"+str(l+1)+"/full_split_"+str(split)+
240                     "/val/dog/"+image_name
241                 cv2.imwrite(new_im_path, image)
242                 print(f"Saving image to {new_im_path}")
243                 #copyfile(old_path, new_im_path)
244             else:
245                 new_im_path = self.robust_path+"/"+id_path+"/"+str(l+1)+"/full_split_"+str(split)+
246                     "/val/cat/"+image_name
247                 cv2.imwrite(new_im_path, image)
248                 print(f"Saving image to {new_im_path}")
249                 #copyfile(old_path, new_im_path)

250     def add_all_increase_contrast(self, original_image, id_path, split, curr_class, image_name):
251         factors = np.linspace(1.0, 1.27, 10)
252         for l, factor in enumerate(factors):
253             image = np.array(original_image).copy()
254             image = self.increase_contrast(image, factor)
255             if curr_class == 'dog':
256                 new_im_path = self.robust_path+"/"+id_path+"/"+str(l+1)+"/full_split_"+str(split)+
257                     "/val/dog/"+image_name
258                 cv2.imwrite(new_im_path, image)
259                 #print(f"Saving image to {new_im_path}")
260                 #copyfile(old_path, new_im_path)
261             else:
262                 new_im_path = self.robust_path+"/"+id_path+"/"+str(l+1)+"/full_split_"+str(split)+
263                     "/val/cat/"+image_name
264                 cv2.imwrite(new_im_path, image)
265                 #print(f"Saving image to {new_im_path}")
266                 #copyfile(old_path, new_im_path)

267     def add_all_decrease_contrast(self, original_image, id_path, split, curr_class, image_name):
268         factors = np.arange(1.0, 0.0, -0.1)
269         for l, factor in enumerate(factors):

```

```

264     image = np.array(original_image).copy()
265     image = self.decrease_contrast(image, factor)
266     if curr_class == 'dog':
267         new_im_path = self.robust_path+"/"+id_path+"/"+str(l+1)+"/full_split_"+str(split)+"/val/dog/"+image_name
268         cv2.imwrite(new_im_path,image)
269         #print(f"Saving image to {new_im_path}")
270         #copyfile(old_path, new_im_path)
271     else:
272         new_im_path = self.robust_path+"/"+id_path+"/"+str(l+1)+"/full_split_"+str(split)+"/val/cat/"+image_name
273         cv2.imwrite(new_im_path,image)
274         #print(f"Saving image to {new_im_path}")
275         #copyfile(old_path, new_im_path)
276
277     def add_all_increase_brightness(self, original_image, id_path, split, curr_class, image_name):
278         :
279         factors = np.arange(0, 50, 5)
280         for l,factor in enumerate(factors):
281             image = np.array(original_image).copy()
282             image = self.increase_brightness(image, factor)
283             if curr_class == 'dog':
284                 new_im_path = self.robust_path+"/"+id_path+"/"+str(l+1)+"/full_split_"+str(split)+"/val/dog/"+image_name
285                 cv2.imwrite(new_im_path,image)
286                 #print(f"Saving image to {new_im_path}")
287                 #copyfile(old_path, new_im_path)
288             else:
289                 new_im_path = self.robust_path+"/"+id_path+"/"+str(l+1)+"/full_split_"+str(split)+"/val/cat/"+image_name
290                 cv2.imwrite(new_im_path,image)
291                 #print(f"Saving image to {new_im_path}")
292                 #copyfile(old_path, new_im_path)
293
294     def add_all_decrease_brightness(self, original_image, id_path, split, curr_class, image_name):
295         :
296         factors = np.arange(0, 50, 5)
297         for l,factor in enumerate(factors):
298             image = np.array(original_image).copy()
299             image = self.decrease_brightness(image, factor)
300             if curr_class == 'dog':
301                 new_im_path = self.robust_path+"/"+id_path+"/"+str(l+1)+"/full_split_"+str(split)+"/val/dog/"+image_name
302                 cv2.imwrite(new_im_path,image)
303                 #print(f"Saving image to {new_im_path}")
304                 #copyfile(old_path, new_im_path)
305             else:
306                 new_im_path = self.robust_path+"/"+id_path+"/"+str(l+1)+"/full_split_"+str(split)+"/val/cat/"+image_name
307                 cv2.imwrite(new_im_path,image)
308                 #print(f"Saving image to {new_im_path}")
309                 #copyfile(old_path, new_im_path)
310
311     def add_all_hue_noises(self, original_image, id_path, split, curr_class, image_name):
312         stds = np.arange(0.0, 0.19, .02)
313         for l,std in enumerate(stds):
314             image = np.array(original_image).copy()
315             print(f"Standard deviation: {std}")
316             image = self.increase_hue_noise(image, std)
317             if curr_class == 'dog':
318                 new_im_path = self.robust_path+"/"+id_path+"/"+str(l+1)+"/full_split_"+str(split)+"/val/dog/"+image_name
319                 cv2.imwrite(new_im_path,image)
320                 #print(f"Saving image to {new_im_path}")
321                 #copyfile(old_path, new_im_path)
322             else:
323                 new_im_path = self.robust_path+"/"+id_path+"/"+str(l+1)+"/full_split_"+str(split)+"/val/cat/"+image_name
324                 cv2.imwrite(new_im_path,image)
325                 #print(f"Saving image to {new_im_path}")

```

```

324         #copyfile(old_path , new_im_path)
325         #
326
327     def add_all_sat_noises(self , original_image , id_path , split , curr_class , image_name):
328         stds = np.arange(0.0, 0.19, .02)
329         for l, std in enumerate(stds):
330             image = np.array(original_image).copy()
331             #print(f"Standard deviation: {std}")
332             image = self.increase_saturation_noise(image , std)
333             if curr_class == 'dog':
334                 new_im_path = self.robust_path+"/"+id_path+"/"+str(l+1)+"/full_split_"+str(split)+"/val/dog/"+image_name
335                 cv2.imwrite(new_im_path,image)
336                 #print(f"Saving image to {new_im_path}")
337                 #copyfile(old_path , new_im_path)
338             else:
339                 new_im_path = self.robust_path+"/"+id_path+"/"+str(l+1)+"/full_split_"+str(split)+"/val/cat/"+image_name
340                 cv2.imwrite(new_im_path,image)
341                 #print(f"Saving image to {new_im_path}")
342                 #copyfile(old_path , new_im_path)
343                 #
344
345     def add_all_image_occlusions(self , original_image , id_path , split , curr_class , image_name):
346         edges = np.arange(0, 50, 5)
347         for l,edge in enumerate(edges):
348             image = np.array(original_image).copy()
349             image = self.image_occlusion(image , edge , print_center=True)
350             if curr_class == 'dog':
351                 new_im_path = self.robust_path+"/"+id_path+"/"+str(l+1)+"/full_split_"+str(split)+"/val/dog/"+image_name
352                 cv2.imwrite(new_im_path,image)
353                 print(f"Saving image to {new_im_path}")
354                 #copyfile(old_path , new_im_path)
355             else:
356                 new_im_path = self.robust_path+"/"+id_path+"/"+str(l+1)+"/full_split_"+str(split)+"/val/cat/"+image_name
357                 cv2.imwrite(new_im_path,image)
358                 print(f"Saving image to {new_im_path}")
359                 #copyfile(old_path , new_im_path)
360
361     def create_csv_files(self):
362         # Create root directory
363         # Create all directories
364         print("Creating csvs...")
365         for perturb_id in tqdm(self.perturb_ids):
366             print(f"On perturbation level {perturb_id}...")
367             for perturb_level in self.perturb_levels:
368                 for split in self.splits:
369                     # Create perturbation id folder
370                     full_path = self.robust_path+'5_'+str(perturb_id)+"/"+str(perturb_level)+"/full_split_"+str(split)+"/"
371                     full_path_dog = full_path+"val/dog/"
372                     full_path_cat = full_path+"val/cat/"
373                     dog_imgs = np.array([[full_path_dog + name , 0] for name in os.listdir(full_path_dog) if os.path.isfile(os.path.join(full_path_dog , name))])
374                     cat_imgs = np.array([[full_path_cat + name , 1] for name in os.listdir(full_path_cat) if os.path.isfile(os.path.join(full_path_cat , name))])
375                     imgs = np.concatenate([dog_imgs , cat_imgs ])
376
377                     df = pd.DataFrame({"image_id": imgs[:, 0] , "label": imgs[:, 1] })
378                     df.to_csv(full_path+"full_split_val_"+str(split)+".csv")
379
380
381 if __name__ == "__main__":
382     # Create robustness folder placeholder
383     perturbimages = PerturbImages()
384     #perturbimages.create_placeholder_dirs()
385     #perturbimages.create_csv_files()
386     perturbimages.perform_perturbations()

```
