Image and Vision Computing

INFR11140

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), ORB (Rublee et al., 2011), SURF (Bay et al., 2008), 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 formalized 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 balanced, since the number of images for each class is different. In addition to this, the number of images per breed also varies. In the following Section we will explain how we dealt with the 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 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 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.

	Cats		Dogs	
SubClass	Breeds	n	Breeds	n
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.

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 $\mathscr{H}(\mathbf{x})$, the stacked non-linear layers now fit $\mathscr{F}(\mathbf{x}) = \mathscr{H}(\mathbf{x}) - \mathbf{x}$ and the original mapping is recast into $\mathscr{F}(\mathbf{x}) + \mathbf{x}$. 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. 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 summarize this process using a sample image.

Cross-validation

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

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,

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

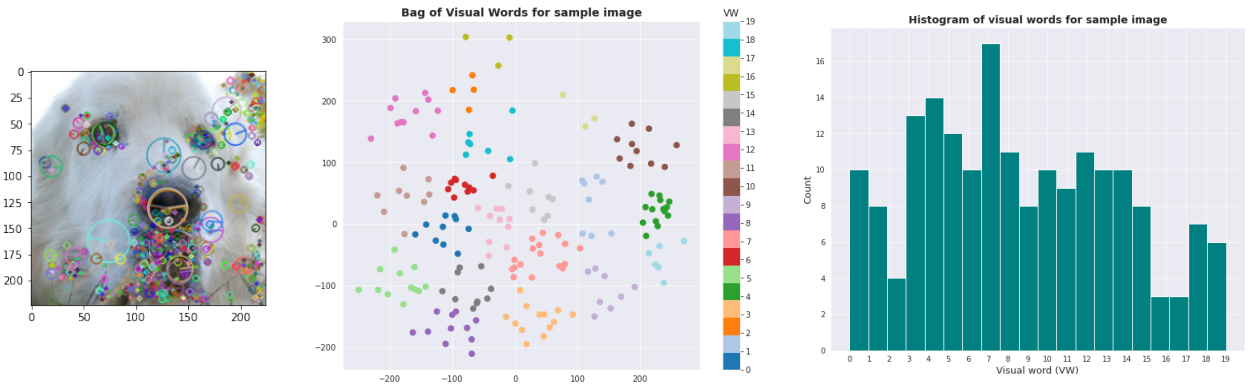

Figure 1. BoVW: *Left.* Visualization of SIFT features on a sample image. *Center.* Scatter plot of the features projected into a 2D space, and then clustered using a K-means algorithm of 20 clusters. *Right.* Histogram representation of the SIFT descriptors of the image on the left, this histogram representation 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.

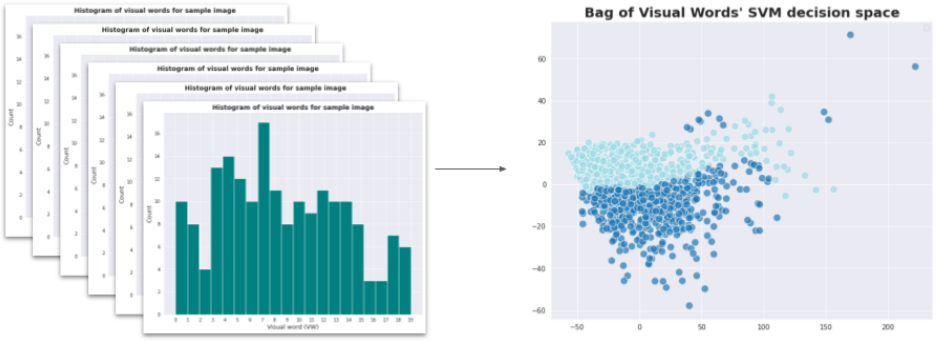

Figure 2. BoVW: All images are turned into histogram Bag of Visual Words histogram representations (left) and then projected into a vector space (right). The Support Vector Machine model then creates a decision boundary in the vector space. The figure on the right corresponds to the classification of 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.

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 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.

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.

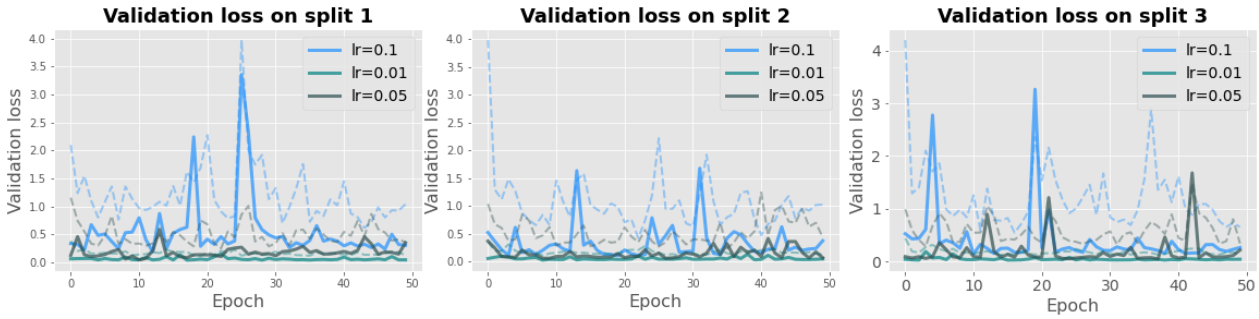

Figure 3. ResNet18: Validation loss from the three splits during training for 50 epochs. The dashed, lighter line corresponds to the training results during 50 epochs. From the three plots we can see that the models with lr=0.01 consistently outperform the rest of the models.

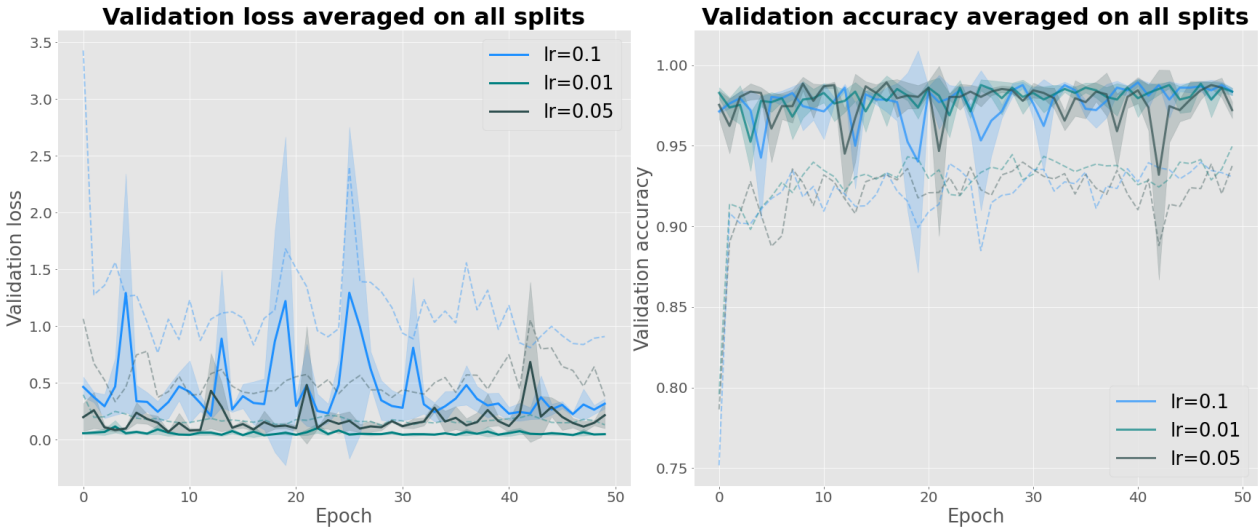

Figure 4. ResNet18: Validation loss and accuracy averaged over the three splits sampled for 50 epochs of training. The dashed line corresponds to the loss and accuracy results obtained during training of the models, for the three learning rates. The shadowed contour corresponds to the standard deviation 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.

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.

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.

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

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

We concluded that the best set of hyperparameters are $k = 250$, rbf kernel and $c = 2.5$. The results for different values of k and c using the rbf kernel are presented in Figure 5.

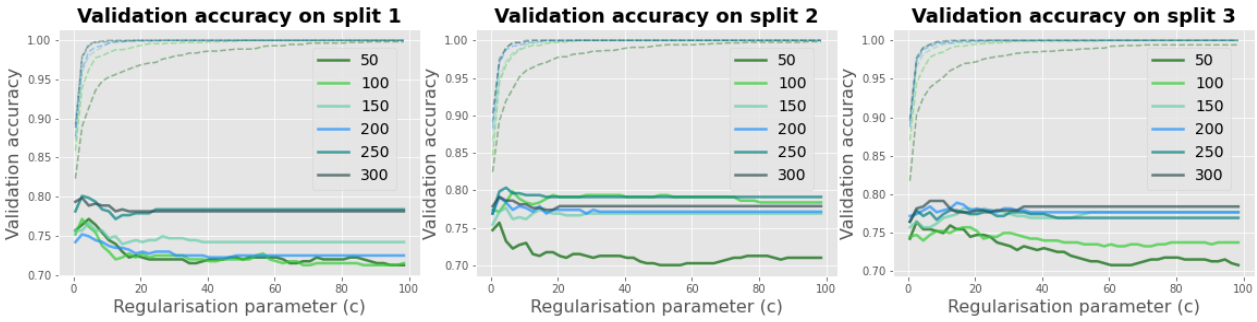

Figure 5. BoVW: Validation loss from the three splits for different values of c and using the rbf kernel in the SVM classifier. Each line corresponds to the results obtained using a different k.

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 80% [2].

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 [3].

The effects of this perturbation are presented in Figure 6.

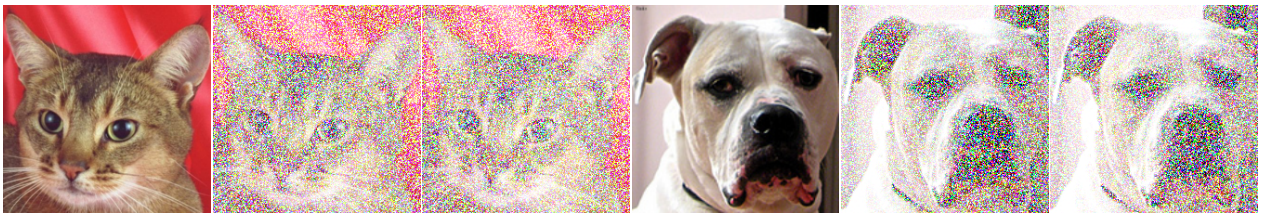

Figure 6. Effects of applying Gaussian pixel noise with $std = \{0, 10, 18\}$.

[2] This is an estimate from the test results found for Split 1 and Split 2. The results for Split 3 are at the moment being calculated.
[3] Negative numbers were replaced by 0 and values > 255 were replaced by 255.

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.

This corresponds to increasing the image contrast. The effects of this perturbation are presented in Figure 7.

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.

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.

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.

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.

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

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.

Occlusion of the image increase

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

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

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

The effects of this perturbation are presented in Figure 14.

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

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 Vue channel lead to less than a 15% drop in accuracy. Decreasing the image brightness and and adding noise to the Saturation channel resulted in a drop in performance of more than 25%. Adding Gaussian pixel noise brought the performance of the classifier to about 50%, even on the first level of perturbation, meaning that the model had the almost the same performance as a random classifier.

TODO: DESCRIBE BOVW MODELS' PERFORMANCE

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 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. In general, we expected the performance to drop as the intensity of the perturbations increased. This was the case for most perturbations, except for the introduction of Gaussian pixel noise to the images, which led to an instant drop in performance, instead of the expected continuous decrease. A possible explanation to this is that noise was added to each pixel, across all channels, whereas all other perturbations change the pixels in the different channels in a similar way or only add noise to one of the channels, allowing the model to still learn from the other two.

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 suprisingly, 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

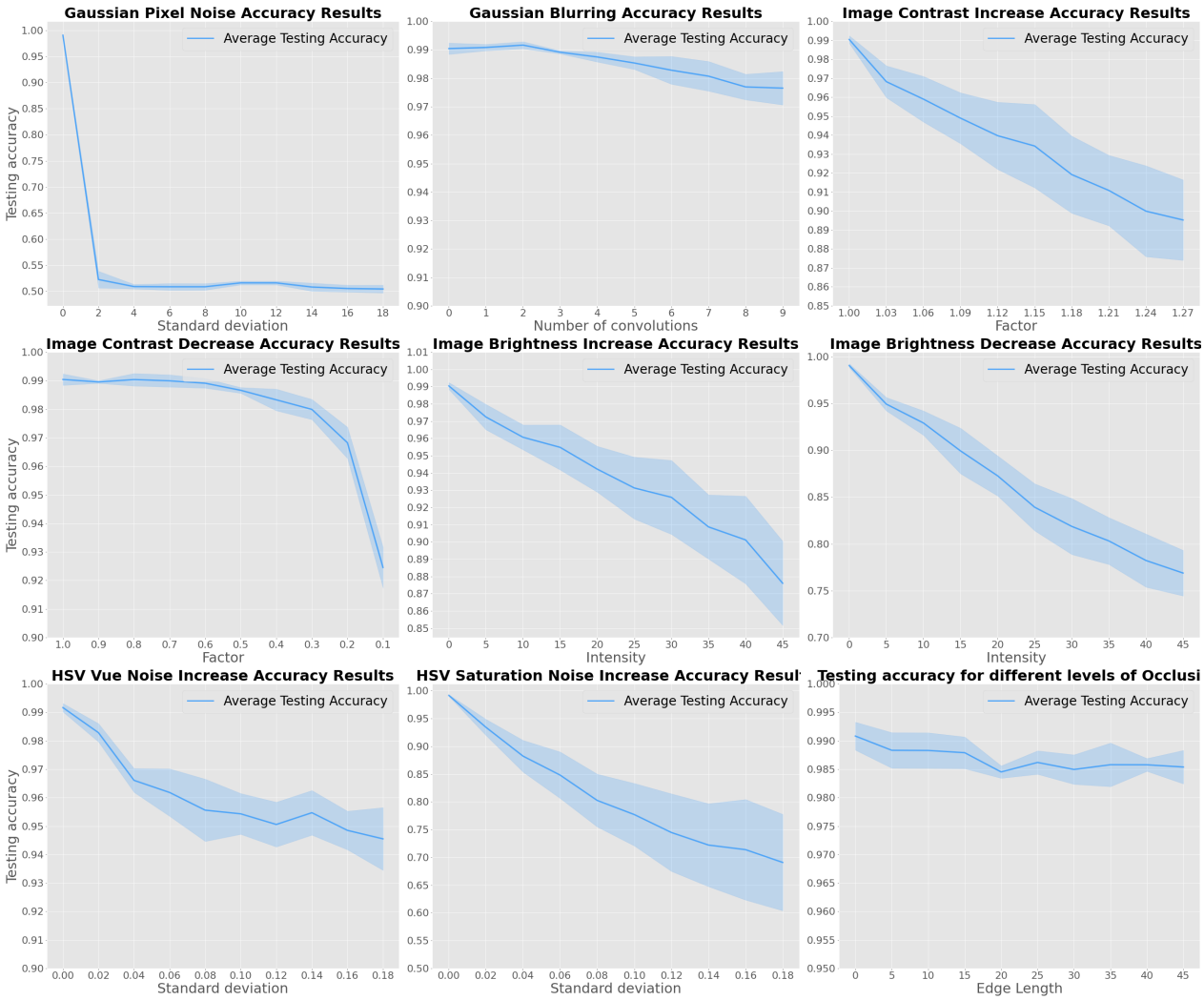

Figure 15. ResNet18: Average test accuracy of the ResNet18 model on test sets incrementally perturbed by a given noise. From these results we can see that the model's accuracy is highly affected by gaussian pixel noise. On the contrary, increasing the contrast of the image appears to improve the model's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

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.

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

```
# Library imports
from __future__ import print_function
from __future__ import division
import torch
import torch.nn as nn
import torch.optim as optim
import numpy as np
import pandas as pd
import torchvision
from torchvision import datasets, models, transforms
import matplotlib.pyplot as plt
import time
import os
import copy
#from tqdm.notebook import tqdm
from tqdm import tqdm

# Define input feature map size
INPUT_SIZE = 224

class ResNet18:
    def __init__(self, on_split, input_data_dir = "./data",
                 output_dir = "./output/resnet18",
                 batch_size = 50,
                 num_epochs = 50,
                 num_classes = 2,
                 training = True,
                 pretrained_model = None,
                 feature_extract = True,
                 full_split = False,
                 perturbation = ""
                 ):
        '''
        The ResNet18 class gets the data directory as input, as well as the hyperparameters for
        training, and outputs the training progress and the trained model as a .pth file.
        '''
        # Initialize global variables
        self.full_split = full_split
        if self.full_split:
            self.data_dir = input_data_dir+"/full_split_"+str(on_split)+"/"
            self.output_dir = output_dir+"/full_split_"+str(on_split)+"/"
        else:
            self.data_dir = input_data_dir+"/split_"+str(on_split)+"/"
            self.output_dir = output_dir+"/split_"+str(on_split)+"/"
        self.training = training
        self.num_classes = num_classes
        self.batch_size = batch_size
        self.num_epochs = num_epochs
        self.input_size = INPUT_SIZE

        # Whether to use pretrained weights from ImageNet or not
        self.feature_extract = feature_extract
        self.pretrained_model = pretrained_model

    def set_parameter_requires_grad(self, model, feature_extracting):
        '''
        Receives the model and a the boolean feature_extracting variable, which if
        set to True, uses the pretrained weights from ImageNet and updates the parameters of
        the model accordingly.
        '''
        if feature_extracting:
            for param in model.parameters():
                param.requires_grad = False

    def initialize_model(self, num_classes, feature_extract):
        '''
```

```
        Receives the output number of classes, feature_extract, and the use_pretrained flag
        and outputs the initialized model.
        '''
        # Initialize input size and model variable
        model_ft = None
        input_size = 0
        # Initialize model
        model_ft = models.resnet18(pretrained = feature_extract)

        self.set_parameter_requires_grad(model_ft, feature_extract)
        num_ftrs = model_ft.fc.in_features
        model_ft.fc = nn.Linear(num_ftrs, num_classes)

        return model_ft

    def get_dataloaders(self, data_dir, bs, input_size):
        '''
        Takes the data directory as input, the batch size and the input size, and outputs
        the torch dataloader containing the transformed images in batch of bs size.
        '''
        # Define the data transformations
        data_transforms = {
            'train': transforms.Compose([
                transforms.RandomResizedCrop(input_size),
                transforms.RandomHorizontalFlip(),
                transforms.ToTensor(),
                transforms.Normalize([0.485, 0.456, 0.406], [0.229, 0.224, 0.225])
            ]),
            'val': transforms.Compose([
                transforms.Resize(input_size),
                transforms.CenterCrop(input_size),
                transforms.ToTensor(),
                transforms.Normalize([0.485, 0.456, 0.406], [0.229, 0.224, 0.225])
            ]),
        }

        if self.training:
            # Create training and validation datasets
            image_datasets = {x: datasets.ImageFolder(os.path.join(data_dir, x), data_transforms[x
                ]) for x in ['train', 'val']}
            # Create training and validation dataloaders
            loader = {x: torch.utils.data.DataLoader(image_datasets[x], batch_size=bs, shuffle=
                True, num_workers=4) for x in ['train', 'val']}
        else:
            # Create training and validation datasets
            image_datasets = {x: datasets.ImageFolder(os.path.join(data_dir, x), data_transforms[x
                ]) for x in ['val']}
            # Create training and validation dataloaders
            loader = {x: torch.utils.data.DataLoader(image_datasets[x], batch_size=bs, shuffle=
                True, num_workers=4) for x in ['val']}

        return loader

    def train_model(self, model, dataloaders, criterion, optimizer, device, num_epochs=25):
        '''
        Takes a pretrained model, the dataloaders, the momentum criterium, the optimizer, the
            torch device, the number
        of epochs and outputs the trained model with the progress information.
        '''
        # Start counting the training time
        since = time.time()

        # Initialize variables that will store the training information
        train_acc_history = []
        train_loss_history = []
        val_acc_history = []
        val_loss_history = []

        # Get the best model weights from the inputted model
        best_model_wts = copy.deepcopy(model.state_dict())
```

```
        best_acc = 0.0

        # Loop through epochs to start training
        for epoch in range(num_epochs):
            print('Epoch {}/{}'.format(epoch, num_epochs -1))
            print('-'*10)

            # Each epoch has a training and validation pass
            if self.training:
                phases = {'train':"Training network...", 'val':"Evaluating network..."}
            else:
                phases = {'val':"Evaluating network..."}

            for phase in phases.keys():
                if phase == 'train':
                    model.train()       # Set model to training mode
                else:
                    model.eval()        # Set model to evaluate mode

                running_loss = 0.0
                running_corrects = 0

                # Iterate over data.
                print(phases[phase])
                print(f"Looking at data in {self.data_dir}")
                for inputs, labels in tqdm(dataloaders[phase]):
                    inputs = inputs.to(device)
                    labels = labels.to(device)

                    # zero the parameter gradients
                    optimizer.zero_grad()

                    # forward
                    # track history if only in train
                    with torch.set_grad_enabled(phase == 'train'):
                        # Get model outputs and calculate loss
                        outputs = model(inputs)
                        loss = criterion(outputs, labels)

                        _, preds = torch.max(outputs, 1)

                        # Backpropagation + optimize only if in training phase
                        if phase == 'train':
                            loss.backward()
                            optimizer.step()

                    # statistics
                    running_loss += loss.item() * inputs.size(0)
                    running_corrects += torch.sum(preds == labels.data)

                epoch_loss = running_loss / len(dataloaders[phase].dataset)
                epoch_acc = running_corrects.double() / len(dataloaders[phase].dataset)

                print('{} Loss: {:.4f} Acc: {:.4f}'.format(phase, epoch_loss, epoch_acc))

                # deep copy the model
                if phase == 'val' and epoch_acc > best_acc:
                    best_acc = epoch_acc
                    best_model_wts = copy.deepcopy(model.state_dict())
                if phase == 'val':
                    val_acc_history.append(epoch_acc.numpy())
                    val_loss_history.append(epoch_loss)
                else:
                    train_acc_history.append(epoch_acc.numpy())
                    train_loss_history.append(epoch_loss)

        # Calculate total training time
        time_elapsed = time.time() - since
        print('\nTraining complete in {:.0f}m {:.0f}s'.format(time_elapsed // 60, time_elapsed %
            60))
```

```
        #print('Best val Acc: {:4f}'.format(best_acc))

        # load best model weights
        model.load_state_dict(best_model_wts)

        return model, val_acc_history, val_loss_history, train_acc_history, train_loss_history,
            best_acc.numpy(), time_elapsed

    def run(self, lr=0.01, momentum=0.9, hyp_name = ""):
        print("Initializing Datasets and Dataloaders...")
        # Get cross-validation data
        loader = self.get_dataloaders(self.data_dir, self.batch_size, self.input_size)

        # Detect if we have a GPU available
        device = torch.device("cuda:0" if torch.cuda.is_available() else "cpu")

        # Initialize the model for this run
        model_ft = self.initialize_model(self.num_classes, self.feature_extract)

        # Send the model to GPU
        model_ft = model_ft.to(device)

        # Initialize optimizer
        params_to_update = model_ft.parameters()

        if self.feature_extract:
            params_to_update = []
            for name, param in model_ft.named_parameters():
                if param.requires_grad == True:
                    params_to_update.append(param)
        else:
            for name, param in model_ft.named_parameters():
                if param.requires_grad == True:
                    pass

        # If testing load information from pretrained model
        if not self.training:
            model_ft.load_state_dict(self.pretrained_model)

        # Define optimizer
        optimizer_ft = optim.SGD(params_to_update, lr=lr, momentum=0.9)

        # Setup the loss function
        criterion =  nn.CrossEntropyLoss()

        # Train model
        model_ft, val_acc_history, val_loss_history, train_acc_history, train_loss_history,
            best_acc, time_elapsed = self.train_model(
            model_ft, loader, criterion, optimizer_ft, device, num_epochs=self.num_epochs)

        # Record data
        training_output = {}

        # Output data
        root_output_dir = self.output_dir
        if self.training:
            training_output['train_loss'] = train_loss_history
            training_output['train_acc'] = train_acc_history
            training_output['val_loss'] = val_loss_history
            training_output['val_acc'] = val_acc_history
            training_output['best_acc'] = [best_acc]*self.num_epochs
            training_output['runtime(s)'] = [time_elapsed]*self.num_epochs
            df_name = root_output_dir+"progress/train_progress"+hyp_name+".csv"
            print(f"Outputting data to {df_name}...")
            pd.DataFrame.from_dict(training_output).to_csv(df_name, index=False)
            output_w_dir = root_output_dir+"weights/trained_model"+hyp_name+".pth"
            torch.save(model_ft.state_dict(), output_w_dir)
            print(f"Saving final trained model to {output_w_dir}")
        else:
            training_output['val_loss'] = val_loss_history
```

```
            training_output['val_acc'] = val_acc_history
            training_output['best_acc'] = [best_acc]*self.num_epochs
            training_output['runtime(s)'] = [time_elapsed]*self.num_epochs
            df_name = root_output_dir+"progress/test_progress.csv"
            #print(f"Outputting data to {df_name}...")
            #pd.DataFrame.from_dict(training_output).to_csv(df_name, index=False)
            print(f"Test accuracy of {best_acc}")
            return best_acc


#-------------------------------------------------------------------------------------------
#                                        MAIN FUNCTION
#-------------------------------------------------------------------------------------------

if __name__ == "__main__":
    # Load pre-trained model
    pretrained = torch.load("./output/resnet18/split_1/weights/trained_model.pth")
    # Instantiate network
    resnet = ResNet18(1, "./data", "./output/resnet18/",
                      batch_size=50, num_epochs=1,
                      num_classes=2, training=False, pretrained_model=pretrained,
                      feature_extract=True)

    # Train ResNet18
    resnet.run()
```

Listing 2. run_resnet18.py

```
from resnet18 import *

# Fine-tune models with learning rate
def train_hyperparameters():
    lrs = [0.01, 0.05, 0.1]
    for split in range(1, 4):
        print(f"#####  On split {split} ######")
        for lr in lrs:
            print(f"Learning rate = {lr}")
            # Instantiate network
            resnet = ResNet18(split, "./data", "./output/resnet18/",
                              batch_size=50, num_epochs=50,
                              num_classes=2, training=True, pretrained_model=None,
                                  feature_extract=True)

            # Train ResNet18
            lr_name = "_lr_"+str(lr).replace('.','_')
            resnet.run(lr=lr, hyp_name=lr_name)

def train_full_splits():
    # Training on lr=0.01 for all full splits
    for split in range(1, 4):
        print(f"#####  On split {split} ######")
        # Instantiate network
        resnet = ResNet18(split, "./data", "./output/resnet18",
                          batch_size=50, num_epochs=50,
                          num_classes=2, training=True, pretrained_model=None, feature_extract=
                              True, full_split=True)

        # Train ResNet18
        resnet.run(lr=0.01)

def evaluate_test_splits():
    # Load pre-trained model
    for split in range(1,4):
        print(f"### On Split {split} ###")
        pretrained = torch.load("./output/resnet18/full_split_"+str(split)+"/weights/trained_model
            .pth")
        # Instantiate network
        input_data_dir = "./data/robustness/5_3/3"
        resnet = ResNet18(1, input_data_dir, "./output/resnet18/",
                          batch_size=50, num_epochs=1,
```

```
                        num_classes=2, training=False, pretrained_model=pretrained,
                        feature_extract=True, full_split=True)

        # Train ResNet18
        best_acc = resnet.run()

        print(f"From Split {split}, the best accuracy was {best_acc}")

def create_dir(directory):
    if not os.path.exists(directory):
        os.makedirs(directory)

def create_placeholder_dirs():
    # Create root directory
    robustness_output_path = "./output/resnet18/robustness"
    create_dir(robustness_output_path)
    # Create list of sub-directory names
    perturb_ids = list(range(1,10))
    # Create all directories
    print("Creating directories...")
    for perturb_id in tqdm(perturb_ids):
        for split in [1,2,3]:
            # Create perturbation id folder
            full_path = robustness_output_path+'/5_'+str(perturb_id)+'/full_split_'+str(split)+'/progress'
            create_dir(full_path)

def evaluate_robustness():
    # Create list of sub-directory names
    perturb_ids = list(range(1,10))
    # Perturbation levels
    perturb_levels = list(range(1,11))
    # Load pre-trained model
    for split in tqdm(range(3,4)):
        print(f"### On Split {split} ###")
        pretrained = torch.load("./output/resnet18/full_split_"+str(split)+"/weights/trained_model.pth")
        # Instantiate network
        if split == 3:
            perturbations = [3]
        else:
            perturbations = perturb_ids
        for perturb_id in perturbations:
            # Initialize list to store accuracies
            best_accs = []
            results_dict = {}
            # Get id folder name
            id_fname = "/5_"+str(perturb_id)
            output_data_dir = "./output/resnet18/robustness/"+id_fname
            for pertur_level in perturb_levels:
                print(f"On perturbation level {pertur_level}, of perturbation id {perturb_id}")
                input_data_dir = "./data/robustness"+id_fname+"/"+str(pertur_level)
                resnet = ResNet18(split, input_data_dir, output_data_dir,
                                  batch_size=50, num_epochs=1,
                                  num_classes=2, training=False, pretrained_model=pretrained,
                                  feature_extract=True, full_split=True)

                # Train ResNet18
                best_acc = resnet.run()
                best_accs.append(best_acc)

            # Store results
            results_dict['perturb_level'] = perturb_levels
            results_dict['best_accs'] = best_accs
            results_df = pd.DataFrame.from_dict(results_dict)
            # Ouput results to .csv file
            csv_fname = output_data_dir+"/full_split_"+str(split)+"/robustness_results_5_"+str(perturb_id)+".csv"
            results_df.to_csv(csv_fname, index=False)

```

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

BoVW Models' Code

Listing 3. bovw.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_split_"
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_split_'+str(split_n)+"_"
36         else:
37             path = self.data_dir+str(split_n)+'/split_'+str(split_n)+"_"
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_split_val_'+str(split_n)+".csv"
51         else:
52             path = self.data_dir+str(split_n)+'/split_'+str(split_n)+"_"
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 dictionnary
61         with all images loaded with OpenCV.
```

```
        '''
        image_dict = {}
        file_locations = list(dataframe['image_id'])
        labels = list(dataframe['label'])
        category_0 = []
        category_1 = []
        for i in range(len(file_locations)):
            image = cv2.imread(file_locations[i], cv2.COLOR_RGB2BGR)
            try:
                image = cv2.cvtColor(image, cv2.COLOR_RGB2BGR)
            except: # if the image is gray
                image = cv2.cvtColor(image,cv2.COLOR_GRAY2RGB)

            if labels[i] == 0:
                category_0.append(image)
            else:
                category_1.append(image)

        image_dict[0] = category_0
        image_dict[1] = category_1

        return image_dict

    def sift_features(self, images):
        '''
        Creates descriptors using sift. Takes one parameter that is images dictionary. Return an array whose first
        index holds the decriptor_list without an order and the second index holds the sift_vectors dictionary which
        holds the descriptors but this is seperated class by class.
        '''
        sift_vectors = {}
        descriptor_list = []
        sift = cv2.xfeatures2d.SIFT_create()
        for key,value in images.items():
            features = []
            for img in value:
                kp, des = sift.detectAndCompute(img,None)
                try:
                    descriptor_list.extend(des)
                    features.append(des)
                except:
                    print(f"Faulty descriptor found : {des}")
            sift_vectors[key] = features

        return descriptor_list, sift_vectors

    def kmeans(self, k, descriptor_list):
        '''
        A k-means clustering algorithm who takes 2 parameter which is number
        of cluster(k) and the other is descriptors list(unordered 1d array)
        Returns an array that holds central points.
        '''
        kmeans = KMeans(n_clusters = k, n_init=10, verbose=0)
        kmeans.fit(descriptor_list)
        vwords = kmeans.cluster_centers_

        return kmeans, vwords

    def find_index(self, image, center):
        '''
        Receives an image and a cluster center and returns to which image does the center belong as and index
        '''
        count = 0
        ind = 0
        for i in range(len(center)):
            if(i == 0):
                count = distance.euclidean(image, center[i])
            else:
```

```
                dist = distance.euclidean(image, center[i])
                if(dist < count):
                    ind = i
                    count = dist
        return ind

    # Extract visual words from descriptors
    def get_histograms_dictio(self, sift_vectors, kmeans_centers):
        '''
        Receives the keypoints and the cluster centers and returns
        a dictionary with all the histograms, divided by class
        '''
        dict_feature = {}
        for key,value in tqdm(sift_vectors.items()):
            print(f"Getting histograms of class {key}...")
            category = []
            for img in tqdm(value):
                histogram = np.zeros(len(kmeans_centers))
                for each_feature in img:
                    ind = self.find_index(each_feature, kmeans_centers)
                    histogram[ind] += 1
                    category.append(histogram)

            dict_feature[key] = category
        return dict_feature

    # Map dictionary of histograms to np arrays
    def get_histogram_arrays(self, sift_vectors, kmeans_centers):
        '''
        Receives the keypoints and cluster centers and returns the np.array of the
        training data and the training labels
        '''
        histogram_dictio = self.get_histograms_dictio(sift_vectors, kmeans_centers)
        X = []
        Y = []
        print("Converting to np.arrays...")
        for key in histogram_dictio.keys():
            for value in histogram_dictio[key]:
                X.append(value)
                Y.append(key)

        return np.array(X), np.array(Y)

    def train_svm(self, visual_words, labels, c, k_):
        '''
        Takes the visual words, the labels, the complexity ter c, and the type of
        kernel and outputs a trained svm classifier
        '''
        # Get training data
        X_train = visual_words
        Y_train = labels

        # Initialize SVM classifier
        svc_classifier = make_pipeline(StandardScaler(), SVC(C=c, kernel=k_, gamma='auto'))
        svc_classifier.fit(X_train, Y_train)

        return svc_classifier

    def get_train_val_dict(self, split_n):
        # Get split data
        print('Convert dataframe to dictionary of images')
        split_n = self.split_n
        train_splits, val_splits = self.get_splits(split_n)

        # Get split images
        print(f'Get split {split_n} data')
        train_dict = self.image_reader(train_splits)
        val_dict = self.image_reader(val_splits)

        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_k_means, 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,
                train_centers)
222         print("Get validation histograms from kmeans clustering")
223         val_histograms, val_classes = self.get_histogram_arrays(val_sift_vectors, train_centers)
224         if not self.full_split:
225             np.save("output/bovw/split_"+str(self.split_n)+"/histograms/train/
                    train_visual_words_k_"+str(K_clusters)+".npy", train_histograms)
226             np.save("output/bovw/split_"+str(self.split_n)+"/histograms/train/train_classes_k_"+
                    str(K_clusters)+".npy", train_classes)
227             np.save("output/bovw/split_"+str(self.split_n)+"/histograms/val/val_visual_words_k_"+
                    str(K_clusters)+".npy", val_histograms)
228             np.save("output/bovw/split_"+str(self.split_n)+"/histograms/val/val_classes_k_"+str(
                    K_clusters)+".npy", val_classes)
229
230         else:
231             np.save("output/bovw/full_split_"+str(self.split_n)+"/histograms/train/
                    train_visual_words_k_"+str(K_clusters)+".npy", train_histograms)
232             np.save("output/bovw/full_split_"+str(self.split_n)+"/histograms/train/
                    train_classes_k_"+str(K_clusters)+".npy", train_classes)
233             np.save("output/bovw/full_split_"+str(self.split_n)+"/histograms/train/
                    train_centers_k_"+str(K_clusters)+".npy", train_centers)
234             np.save("output/bovw/full_split_"+str(self.split_n)+"/histograms/val/
                    val_visual_words_k_"+str(K_clusters)+".npy", val_histograms)
235             np.save("output/bovw/full_split_"+str(self.split_n)+"/histograms/val/val_classes_k_"+
                    str(K_clusters)+".npy", val_classes)
```

Listing 4. run_bovw.py

```
1  # Import bovw class
2  from bovw import *
3  from tqdm import tqdm
4
5  def retreive_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"Retreiving histograms from Split {split}...")
12         bovw = BovW(split, "./data")
13         print("Getting data...")
14         train_dict, val_dict = bovw.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, cluser", c)
19                 # Get all histograms for all clusters
20                 bovw.get_all_histograms(c, train_dict, val_dict)
21             continue
```

```
        for cluster in clusters:
            print(f"On cluster {cluster}...")
            # Initialize BovW model
            # Get all histograms for all clusters
            bovw.get_all_histograms(cluster, train_dict, val_dict)

def retrieve_accuracies():
    # Define hyperparameter's value range
    c_range =np.arange(0.5, 100.5, 2)
    clusters = [50, 100, 150, 200, 250, 300]
    kernels = ['linear', 'poly', 'rbf']
    # Run through splits and clusters
    for split in tqdm(range(1,4)):
        # Loop through kernels
        for k in tqdm(kernels):
            # Define place holder for accuracie values
            df_train_dict = {}
            df_val_dict = {}
            # Loop through clusters
            for cluster in tqdm(clusters):
                # Load data
                train_histograms = np.load("output/bovw/split_"+str(split)+"/histograms/train/
                    train_visual_words_k_"+str(cluster)+".npy")
                train_classes = np.load("output/bovw/split_"+str(split)+"/histograms/train/
                    train_classes_k_"+str(cluster)+".npy")
                val_histograms = np.load("output/bovw/split_"+str(split)+"/histograms/val/
                    val_visual_words_k_"+str(cluster)+".npy")
                val_classes = np.load("output/bovw/split_"+str(split)+"/histograms/val/
                    val_classes_k_"+str(cluster)+".npy")
                # Cluster accuracies
                train_accuracies = []
                val_accuracies = []
                # Loop through c values
                c_vals = []
                for c in tqdm(c_range):
                    c_vals.append(c)
                    # Get SVM classifier
                    bovw = BovW(split, "./data")
                    svm_classifier = bovw.train_svm(train_histograms, train_classes, c, k)
                    # Get train and val accuracies
                    train_acc = svm_classifier.score(train_histograms, train_classes)
                    val_acc = svm_classifier.score(val_histograms, val_classes)
                    # Append accuracy values
                    train_accuracies.append(train_acc)
                    val_accuracies.append(val_acc)

                # Store accuracies in dictionary
                df_train_dict['c'] = c_vals
                df_train_dict[str(cluster)] = train_accuracies
                df_val_dict['c'] = c_vals
                df_val_dict[str(cluster)] = val_accuracies

            # Store dataframes
            df_train = pd.DataFrame.from_dict(df_train_dict)
            df_val = pd.DataFrame.from_dict(df_val_dict)

            # Save dataframes
            df_train.to_csv("./output/bovw/split_"+str(split)+"/svm_results/train_acc_"+str(k)+".
                csv")
            df_val.to_csv("./output/bovw/split_"+str(split)+"/svm_results/val_acc_"+str(k)+".csv")

def train_full_splits(best_cluster_n):
    # Define number of clusters and split to go through
    splits = [1,2,3]

    for split in tqdm(splits):
        print(f"Retreiving histograms from Split {split}...")
        bovw = BovW(split, "./data", full_split=True)
        print("Getting data...")
        train_dict, val_dict = bovw.get_train_val_dict(split)
```

```
        print("Getting histograms...")
        bovw.get_all_histograms(best_cluster_n, train_dict, val_dict)

def evaluate_robustness(best_c, best_kernel):
    # Create list of sub-directory names
    perturb_ids = list(range(1,10))
    # Perturbation levels
    perturb_levels = list(range(1,11))
    # Load pre-trained model
    for split in tqdm(range(1,4)):
        print(f"### On Split {split} ###")
        # Get trained histograms
        train_histograms = np.load("./output/bovw/full_split_"+str(split)+"/histograms/train/
            train_visual_words_k_250.npy")
        train_classes = np.load("./output/bovw/full_split_"+str(split)+"/histograms/train/
            train_classes_k_250.npy")
        train_centers = np.load("./output/bovw/full_split_"+str(split)+"/histograms/train/
            train_centers_k_250.npy")

        # Instantiate network
        # Skip level 5_4 since something is happening  with Nones, we can debug it ad the end
        for perturb_id in perturb_ids[4:]:
        #for perturb_id in [4]:
            # Initialize list to store accuracies
            best_accs = []
            results_dict = {}
            # Get id folder name
            id_fname = "/5_"+str(perturb_id)
            output_data_dir = "./output/bovw/robustness/"+id_fname+"/"
            # Loop through perturbation levels
            for pertur_level in perturb_levels:
            #for pertur_level in [7]:
                print(f"On perturbation level {pertur_level}, of perturbation id {perturb_id}")
                input_data_dir = "./data/robustness"+id_fname+"/"+str(pertur_level)
                bovw = BovW(split, input_data_dir, full_split=True)
                # Get validation information from perturbed images
                val_df = bovw.get_valid_splits(split)
                print("Getting validation images...")
                val_dict = bovw.image_reader(val_df)
                val_descriptor_list, val_sift_vectors = bovw.sift_features(val_dict)
                val_histograms, val_classes = bovw.get_histogram_arrays(val_sift_vectors,
                    train_centers)
                # Train SVM classifier on perturbed images
                svm_classifier = bovw.train_svm(train_histograms, train_classes, best_c,
                    best_kernel)
                # Get train and val accuracies
                train_acc = svm_classifier.score(train_histograms, train_classes)
                val_acc = svm_classifier.score(val_histograms, val_classes)
                # Append accuracy values
                print(f"Accuracy of {val_acc} on 5_{perturb_id} of level {pertur_level}.")
                best_accs.append(val_acc)

            # Store results
            results_dict['perturb_level'] = perturb_levels
            results_dict['best_accs'] = best_accs
            results_df = pd.DataFrame.from_dict(results_dict)
            # Ouput results to .csv file
            csv_fname = output_data_dir+"full_split_"+str(split)+"/robustness_results_5_"+str(
                perturb_id)+".csv"
            print(f"Outputting appending data to {csv_fname}...")
            results_df.to_csv(csv_fname, index=False)

if __name__=="__main__":
    evaluate_robustness(2.5, 'rbf')
```

Data Loader Code

Listing 5. data_loader.py

```python
# Library imports
import os.path
from os import path
import numpy as np
import pandas as pd
from PIL import Image
import matplotlib.pyplot as plt

import torch
from torch.utils.data import Sampler
from torch.utils.data import Dataset
from torch.utils.data import DataLoader
from torch.utils.data import BatchSampler
from torchvision.transforms import transforms
from sklearn.model_selection import StratifiedKFold
from sklearn.model_selection import StratifiedShuffleSplit
from PIL import Image
from tqdm import tqdm
from shutil import copyfile
from sklearn.utils import shuffle

# Define data paths
DOGS_PATH = "./catdog/DOGS/"
CATS_PATH = "./catdog/CATS/"
DATA_PATH = "./catdogs.csv"


# Create csv file of all data paths
def create_csv():
    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))])
    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))])
    imgs = np.concatenate([dog_imgs, cat_imgs])

    df = pd.DataFrame({"image_id": imgs[:, 0], "abc": np.full(imgs.shape[0], ""), "label": imgs[:,
         1]})
    df = df[df.image_id != DOGS_PATH + '.DS_Store'] # Remove the .DS_Store file
    df = df[df.image_id != CATS_PATH + '.DS_Store'] # Remove the .DS_Store file

    # Split each breed into 3 folds (A, B and C) with balanced breeds
    classes = [('dog', DOGS_PATH), ('cat', CATS_PATH)]
    breeds = np.arange(1, 13)
    for animal, path in classes:
        for breed in breeds:
            breed_df = df[df.image_id.str.startswith(path + animal + '_{}_'.format(breed))]
            num_images = breed_df.shape[0]
            # Send first third of the breed to the A fold
            for img_id in breed_df[:num_images//3].image_id:
                df.loc[df.image_id == img_id, 'abc'] = 'A'
            # Send second third of the breed to the B fold
            for img_id in breed_df[num_images//3:num_images*2//3].image_id:
                df.loc[df.image_id == img_id, 'abc'] = 'B'
            # Send last third of the breed to the C fold
            for img_id in breed_df[num_images*2//3:].image_id:
                df.loc[df.image_id == img_id, 'abc'] = 'C'

    df.to_csv('catdogs.csv', index=False) # Save DataFrame as a csv file


def split(dataframe, full_split=False):
    train_folds = [['A', 'B'], ['A', 'C'], ['B', 'C']]

    splits = {}
    for i, train in enumerate(train_folds):
        split = {}
        split['train'] = dataframe[dataframe.abc.isin(train)]
        split['test'] = dataframe[~dataframe.abc.isin(train)]

        if full_split:
```

```
            splits['full_split_{}'.format(i+1)] = split
        else:
            splits['split_{}'.format(i+1)] = split

    return splits

# Divide each split's training data into train (75%) and validation (25%) with balanced breeds
def divide_train_val_test(dataframe, splits):
    classes = [('dog', DOGS_PATH), ('cat', CATS_PATH)]
    breeds = np.arange(1, 13)
    for i in range(1, 4):
        split_train = pd.DataFrame(
                                {"image_id": splits['split_{}'.format(i)]['train'].image_id,
                                 "train_val_test": np.full(splits['split_{}'.format(i)]['train'].
                                     shape[0], ""),
                                 "label": splits['split_{}'.format(i)]['train'].label
                                })
        for animal, path in classes:
            for breed in breeds:
                breed_df = split_train[split_train.image_id.str.startswith(path + animal + '_{}_'.
                    format(breed))]
                num_images = breed_df.shape[0]
                # 75% training data
                for img_id in breed_df[:num_images*3//4].image_id:
                    split_train.loc[dataframe.image_id == img_id, 'train_val_test'] = 'train'
                # 25% validation data
                for img_id in breed_df[num_images*3//4:].image_id:
                    split_train.loc[dataframe.image_id == img_id, 'train_val_test'] = 'val'

        splits['split_{}'.format(i)]['train'] = split_train
        splits['split_{}'.format(i)]['test'] = pd.DataFrame(
                                {"image_id": splits['split_{}'.format(i)]['test'].image_id,
                                 "train_val_test": np.full(splits['split_{}'.format(i)]['test'].
                                     shape[0], "test"),
                                 "label": splits['split_{}'.format(i)]['test'].label
                                })

    return splits

# Changes test data to validation to be compatible with the Resnet's dataloaders
# There is no test set
def divide_train_val(dataframe, splits):
    for i in range(1, 4):
        splits['full_split_{}'.format(i)]['train'] = pd.DataFrame(
                                {"image_id": splits['full_split_{}'.format(i)]['train'].image_id,
                                 "train_val_test": np.full(splits['full_split_{}'.format(i)]['
                                     train'].shape[0], "train"),
                                 "label": splits['full_split_{}'.format(i)]['train'].label
                                })
        splits['full_split_{}'.format(i)]['test'] = pd.DataFrame(
                                {"image_id": splits['full_split_{}'.format(i)]['test'].image_id,
                                 "train_val_test": np.full(splits['full_split_{}'.format(i)]['
                                     test'].shape[0], "val"),
                                 "label": splits['full_split_{}'.format(i)]['test'].label
                                })

    return splits

# Create three splits ([train=A+B, test=C], [train=A+C, test=B], [train=B+C, test=A])
def create_splits(dataframe, full_split=False):
    # Create splits
    splits = split(dataframe, full_split)

    if full_split:
        splits = divide_train_val(dataframe, splits)
    else:
        splits = divide_train_val_test(dataframe, splits)

    return splits

```

```
# Create directory if it does not exist
def create_dir(directory):
    if not os.path.exists(directory):
        os.makedirs(directory)


# Create directories to store the new splits in
def create_dirs(root_dir, split_names, classes):
    # Create root directory
    create_dir(root_dir)

    for split_name in split_names:
        # Create split directory
        create_dir(root_dir+split_name)

        # Create full_split directory
        create_dir(root_dir+'full_'+split_name)

        # Create train directory for the current split
        create_dir(root_dir+split_name+'/train/')
        # Create validation directory for the current split
        create_dir(root_dir+split_name+'/val/')
        # Create test directory for the current split
        create_dir(root_dir+split_name+'/test/')

        # Create train directory for the current full_split
        create_dir(root_dir+'full_'+split_name+'/train/')
        # Create validation directory for the current split
        create_dir(root_dir+'full_'+split_name+'/val/')

        for c in classes:
            # Create class directories for the current split
            create_dir(root_dir+split_name+'/train/'+c+'/')
            create_dir(root_dir+split_name+'/val/'+c+'/')
            create_dir(root_dir+split_name+'/test/'+c+'/')

            # Create class directories for the current full_split
            create_dir(root_dir+'full_'+split_name+'/train/'+c+'/')
            create_dir(root_dir+'full_'+split_name+'/val/'+c+'/')

# Copy files from catdog directory to splits directory
def copy_files(splits, full_split=False):
    # Initial information definition
    root_dir = './data/'
    split_names = splits.keys()
    classes = ['dog', 'cat']
    # Create directories, if necessary
    if not os.path.exists(root_dir):
        print("Creating data directories...")
        create_dirs(root_dir, splits, classes)

    print("Going through each split")
    for split_name in tqdm(split_names):
        # Copy files to train and validation sets
        train_df_dict = {}
        train_files = []
        train_labels = []
        val_df_dict = {}
        val_files = []
        val_labels = []
        test_df_dict = {}
        test_files = []
        test_labels = []

        for row in tqdm(splits[split_name]['train'].itertuples()):
            # row = (index, image_id, train_val_test, label)
            old_path = row[1]
            file_name = row[1].split('/')[-1]
            data_set = row[2]
            class_name = classes[row[3]]
```

```
            new_path = root_dir + split_name + '/' + data_set + '/' + classes[row[3]] + '/' +
                file_name
            #print(old_path, '\t\t', new_path)
            copyfile(old_path, new_path)

            # Store file names and labels
            if data_set == 'train':
                train_files.append(new_path)
                train_labels.append(row[3])
            else:
                val_files.append(new_path)
                val_labels.append(row[3])
        for row in tqdm(splits[split_name]['test'].itertuples()):
            # row = (index, image_id, train_val_test, label)
            old_path = row[1]
            file_name = row[1].split('/')[-1]
            data_set = row[2]
            class_name = classes[row[3]]
            new_path = root_dir + split_name + '/' + data_set + '/' + classes[row[3]] + '/' +
                file_name
            #print(old_path, '\t\t', new_path)
            copyfile(old_path, new_path)

            # Store file names and labels
            test_files.append(new_path)
            test_labels.append(row[3])

        # Create .csv with training file names and labels
        train_df_dict['image_id'] = train_files
        train_df_dict['label'] = train_labels
        file_name = root_dir+'/'+split_name+'/'+split_name+'_train.csv'
        pd.DataFrame.from_dict(train_df_dict).to_csv(file_name, index=False)

        # Create .csv with validation file names and labels
        if not full_split:
            val_df_dict['image_id'] = val_files
            val_df_dict['label'] = val_labels
            file_name = root_dir+'/'+split_name+'/'+split_name+'_val.csv'
            pd.DataFrame.from_dict(val_df_dict).to_csv(file_name, index=False)

        # Create .csv with test file names and labels
        test_df_dict['image_id'] = test_files
        test_df_dict['label'] = test_labels
        if full_split:
            file_name = root_dir+'/'+split_name+'/'+split_name+'_val.csv'
            pd.DataFrame.from_dict(test_df_dict).to_csv(file_name, index=False)
        else:
            file_name = root_dir+'/'+split_name+'/'+split_name+'_test.csv'
            pd.DataFrame.from_dict(test_df_dict).to_csv(file_name, index=False)

# Create data splits
if __name__ == '__main__':
    # Create .csv file with data location
    create_csv()

    # Retrieve data locations dataframe
    print("Loading data...")
    dataset = pd.read_csv('catdogs.csv')

    # Create splits
    print("Creating splits...")
    splits = create_splits(dataset)
    full_splits = create_splits(dataset, full_split=True)

    # Create directories and copy files to directories
    print("Copying files...")
    copy_files(splits)
    copy_files(full_splits, full_split=True)
```

Image Noise Adder Code

Listing 6. perturb_images.py

```
import numpy as np
from numpy.random import normal, randint
import cv2
import os
import pandas as pd
import matplotlib.pyplot as plt
from scipy import ndimage
from scipy.spatial import distance
from sklearn.cluster import KMeans
from sklearn.utils import shuffle
from sklearn.svm import SVC
from sklearn.pipeline import make_pipeline
from sklearn.preprocessing import StandardScaler
from tqdm import tqdm
from shutil import copyfile


class Noises:
    def __init__(self):
        '''
        This class processes the images from a file location
        to output the same, noisier images in an output file location
        '''

    def gaussian_pixel_noise(self, img, std):
        '''
        This function takes an image and a standard deviation as input and outputs
        the same image with noise added to it.
        '''
        gauss = normal(0, std, img.shape).astype('uint8')
        img_gauss = cv2.add(img,gauss)
        image = np.clip(img_gauss, 0, 255)

        return image

    def gaussian_blurring(self, image, n_convs):
        '''
        This function takes an image as an input, and the number of convolutions
        to apply to the image, and outputs a blurred image
        '''
        self.convolution_kernel = 1/16 * np.array([[1, 2, 1],[2, 4, 2],[1, 2, 1]])
        for conv in range(n_convs):
            image = cv2.filter2D(image, -1, self.convolution_kernel)
        return image

    def increase_contrast(self, image, factor):
        '''
        This function takes an image and a factor as an input, and outputs a contrasted
        image according to the given factor
        '''
        n_rows, n_columns, n_channels = image.shape
        for channel in range(n_channels):
            image[:, :, channel] = image[:, :, channel] * factor

        image = np.clip(image, 0, 255)

        return image

    def decrease_contrast(self, image, factor):
        '''
        Takes an image and a factor as input, and outputs an image with the contrast decreased
        by the given factor
        '''
        n_rows, n_columns, n_channels = image.shape
        for channel in range(n_channels):
            image[:, :, channel] = image[:, :, channel] * factor
```

```
        image = np.clip(image, 0, 255)

        return image

    def increase_brightness(self, image, factor):
        '''
        Takes an image and a factor as input, and outputs the same image with the brightness
        increased
        '''
        n_rows, n_columns, n_channels = image.shape
        for channel in range(n_channels):
            image[:, :, channel] += factor
            image = np.clip(image, 0, 255)
        return image

    def decrease_brightness(self, image, factor):
        '''
        Takes an image and a factor as input, and outputs the same image with the brightness
        decreased
        '''
        n_rows, n_columns, n_channels = image.shape

        for channel in range(n_channels):
            image[:, :, channel] -= factor
            image = np.clip(image, 0, 255)
        return image

    def increase_hue_noise(self, image, std):
        '''
        Takes an image as input and a standard deviation and returns an image with
        it's hue noise increased
        '''
        image = cv2.cvtColor(image, cv2.COLOR_BGR2HSV)
        hue_noise = normal(0, std, image.shape[:-1])*255
        image[:,:,0] = image[:,:,0] + hue_noise
        image = np.clip(image, 0, 255)
        hued_image = cv2.cvtColor(image, cv2.COLOR_HSV2BGR)
        return hued_image

    def increase_saturation_noise(self,image, std):
        '''
        Receives an image as input and outputs the same image with increased
        saturation
        '''
        image = cv2.cvtColor(image, cv2.COLOR_BGR2HSV)
        sat_noise = normal(0, std, image.shape[:-1])*255
        image[:,:,1] = image[:,:,1] + sat_noise
        image = np.clip(image, 0, 255)
        sat_image = cv2.cvtColor(image, cv2.COLOR_HSV2BGR)
        return sat_image

    def image_occlusion(self,image, edge, print_center=False):
        '''
        Receives an image and an edge and returns the same image
        with occlusion
        '''
        n_rows, n_columns, _ = image.shape
        center = np.array([randint(0, n_rows+1), randint(0, n_columns+1)])
        start_point = center - edge // 2
        start_point = tuple((start_point[0], start_point[1]))
        end_point = center + edge // 2
        end_point = tuple((end_point[0], end_point[1]))
        colour = (0, 0, 0)
        thickness = -1
        if print_center:
            print(f"Center of the square: [{center[0]}, {center[1]}]")
        image = cv2.rectangle(image, start_point, end_point, colour, thickness)

        return image

```

```
class PerturbImages(Noises):
    def __init__(self, directories_created=True, source_path = "./data/full_split_"):
        self.directories_created = directories_created
        self.source_path = source_path # Path for images, add number according to split
        self.robust_path = "./data/robustness"
        self.source_path = source_path

        self.perturb_ids = list(range(1,10))
        self.perturb_levels = list(range(1,11))
        self.splits = list(range(1,4))
        self.classes = ['dog', 'cat']

    def create_dir(self, directory):
        if not os.path.exists(directory):
            os.makedirs(directory)

    def create_placeholder_dirs(self):
        # Create root directory
        self.create_dir(self.robust_path)
        # Create list of sub-directory names

        # Create all directories
        print("Creating directories...")
        for perturb_id in tqdm(self.perturb_ids):
            self.create_dir(self.robust_path)
            for perturb_level in self.perturb_levels:
                for split in self.splits:
                    for class_ in self.classes:
                        # Create perturbation id folder
                        full_path = self.robust_path+'/5_'+str(perturb_id)+'/'+str(perturb_level)+
                            '/full_split_'+str(split)+'/val/'+class_
                        self.create_dir(full_path)

    def perform_perturbations(self):
        # If placeholders have not been created, create placeholders
        if not self.directories_created:
            self.create_placeholder_dirs()

        for split in tqdm(self.splits):
            csv_path = self.source_path+str(split)+"/full_split_"+str(split)+"_val.csv"
            split_df = pd.read_csv(csv_path)
            image_paths = list(split_df['image_id'])
            image_ids = list(split_df['label'])
            print(f"Loading {len(image_paths)} images...")
            for i, im_path in tqdm(enumerate(image_paths)):
                # Load current image
                original_image = cv2.imread(im_path)
                image_name = im_path.split('/')[-1]
                curr_class = self.classes[image_ids[i]]

                # Start perturbations of current image
                # Add gaussian noise
                #print(f"Adding noise...")
                #self.add_all_gaussian_noise(original_image, "5_1", split, curr_class, image_name)
                # Add gaussian blurr
                #self.add_all_gaussian_blurr(original_image, "5_2", split, curr_class, image_name)
                #print(f"Done with image {im_path}...")
                # Increase contrast
                #self.add_all_increase_contrast(original_image, "5_3", split, curr_class,
                    image_name)
                #print(f"Done with image {im_path}...")
                # Decrease contrast
                # self.add_all_decrease_contrast(original_image, "5_4", split, curr_class,
                    image_name)
                #print(f"Done with image {im_path}...")
                # Increase brightness
                #self.add_all_increase_brightness(original_image, "5_5", split, curr_class,
                    image_name)
                #print(f"Done with image {im_path}...")
                # Decrease brightness
```

```
203                 #self.add_all_decrease_brightness(original_image, "5_6", split, curr_class,
                        image_name)
204                 #print(f"Done with image {im_path}...")
205                 # Add hue noises
206                 #self.add_all_hue_noises(original_image, "5_7", split, curr_class, image_name)
207                 #print(f"Done with image {im_path}...")
208                 # Add sat noises
209                 #self.add_all_sat_noises(original_image, "5_8", split, curr_class, image_name)
210                 # Add occlusions
211                 self.add_all_image_occlusions(original_image, "5_9", split, curr_class, image_name
                        )
212
213     def add_all_gaussian_noise(self, original_image,  id_path, split, curr_class, image_name):
214         self.stds = np.arange(0, 19, 2)
215         for l, std in enumerate(self.stds):
216             image = np.array(original_image).copy()
217             image = self.gaussian_pixel_noise(image, std)
218             if curr_class == 'dog':
219                 new_im_path = self.robust_path+"/"+id_path+"/"+str(l+1)+"/full_split_"+str(split)+
                        "/val/dog/"+image_name
220                 cv2.imwrite(new_im_path,image)
221                 #print(f"Saving image to {new_im_path}")
222                 #copyfile(old_path, new_im_path)
223             else:
224                 new_im_path = self.robust_path+"/"+id_path+"/"+str(l+1)+"/full_split_"+str(split)+
                        "/val/cat/"+image_name
225                 cv2.imwrite(new_im_path,image)
226                 #print(f"Saving image to {new_im_path}")
227                 #copyfile(old_path, new_im_path)
228
229     def add_all_gaussian_blurr(self, original_image,  id_path, split, curr_class, image_name):
230         num_convs = np.arange(10)
231         for l, n_convs in enumerate(num_convs):
232             image = np.array(original_image).copy() # This allows us to apply the same convolution
                      to the blurred image
233             image = self.gaussian_blurring(image, n_convs)
234             if curr_class == 'dog':
235                 new_im_path = self.robust_path+"/"+id_path+"/"+str(l+1)+"/full_split_"+str(split)+
                        "/val/dog/"+image_name
236                 cv2.imwrite(new_im_path,image)
237                 print(f"Saving image to {new_im_path}")
238                 #copyfile(old_path, new_im_path)
239             else:
240                 new_im_path = self.robust_path+"/"+id_path+"/"+str(l+1)+"/full_split_"+str(split)+
                        "/val/cat/"+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
245     def add_all_increase_contrast(self, original_image,  id_path, split, curr_class, image_name):
246         factors = np.linspace(1.0, 1.27, 10)
247         for l,factor in enumerate(factors):
248             image = np.array(original_image).copy()
249             image = self.increase_contrast(image, factor)
250             if curr_class == 'dog':
251                 new_im_path = self.robust_path+"/"+id_path+"/"+str(l+1)+"/full_split_"+str(split)+
                        "/val/dog/"+image_name
252                 cv2.imwrite(new_im_path,image)
253                 #print(f"Saving image to {new_im_path}")
254                 #copyfile(old_path, new_im_path)
255             else:
256                 new_im_path = self.robust_path+"/"+id_path+"/"+str(l+1)+"/full_split_"+str(split)+
                        "/val/cat/"+image_name
257                 cv2.imwrite(new_im_path,image)
258                 #print(f"Saving image to {new_im_path}")
259                 #copyfile(old_path, new_im_path)
260
261     def add_all_decrease_contrast(self, original_image,  id_path, split, curr_class, image_name):
262         factors = np.arange(1.0, 0.0, -0.1)
263         for l,factor in enumerate(factors):
```

```
            image = np.array(original_image).copy()
            image = self.decrease_contrast(image, factor)
            if curr_class == 'dog':
                new_im_path = self.robust_path+"/"+id_path+"/"+str(l+1)+"/full_split_"+str(split)+
                    "/val/dog/"+image_name
                cv2.imwrite(new_im_path,image)
                #print(f"Saving image to {new_im_path}")
                #copyfile(old_path, new_im_path)
            else:
                new_im_path = self.robust_path+"/"+id_path+"/"+str(l+1)+"/full_split_"+str(split)+
                    "/val/cat/"+image_name
                cv2.imwrite(new_im_path,image)
                #print(f"Saving image to {new_im_path}")
                #copyfile(old_path, new_im_path)

    def add_all_increase_brightness(self, original_image, id_path, split, curr_class, image_name)
        :
        factors = np.arange(0, 50, 5)
        for l,factor in enumerate(factors):
            image = np.array(original_image).copy()
            image = self.increase_brightness(image, factor)
            if curr_class == 'dog':
                new_im_path = self.robust_path+"/"+id_path+"/"+str(l+1)+"/full_split_"+str(split)+
                    "/val/dog/"+image_name
                cv2.imwrite(new_im_path,image)
                #print(f"Saving image to {new_im_path}")
                #copyfile(old_path, new_im_path)
            else:
                new_im_path = self.robust_path+"/"+id_path+"/"+str(l+1)+"/full_split_"+str(split)+
                    "/val/cat/"+image_name
                cv2.imwrite(new_im_path,image)
                #print(f"Saving image to {new_im_path}")
                #copyfile(old_path, new_im_path)

    def add_all_decrease_brightness(self, original_image, id_path, split, curr_class, image_name)
        :
        factors = np.arange(0, 50, 5)
        for l,factor in enumerate(factors):
            image = np.array(original_image).copy()
            image = self.decrease_brightness(image, factor)
            if curr_class == 'dog':
                new_im_path = self.robust_path+"/"+id_path+"/"+str(l+1)+"/full_split_"+str(split)+
                    "/val/dog/"+image_name
                cv2.imwrite(new_im_path,image)
                #print(f"Saving image to {new_im_path}")
                #copyfile(old_path, new_im_path)
            else:
                new_im_path = self.robust_path+"/"+id_path+"/"+str(l+1)+"/full_split_"+str(split)+
                    "/val/cat/"+image_name
                cv2.imwrite(new_im_path,image)
                #print(f"Saving image to {new_im_path}")
                #copyfile(old_path, new_im_path)

    def add_all_hue_noises(self, original_image, id_path, split, curr_class, image_name):
        stds = np.arange(0.0, 0.19, .02)
        for l,std in enumerate(stds):
            image = np.array(original_image).copy()
            print(f"Standard deviation: {std}")
            image = self.increase_hue_noise(image, std)
            if curr_class == 'dog':
                new_im_path = self.robust_path+"/"+id_path+"/"+str(l+1)+"/full_split_"+str(split)+
                    "/val/dog/"+image_name
                cv2.imwrite(new_im_path,image)
                #print(f"Saving image to {new_im_path}")
                #copyfile(old_path, new_im_path)
            else:
                new_im_path = self.robust_path+"/"+id_path+"/"+str(l+1)+"/full_split_"+str(split)+
                    "/val/cat/"+image_name
                cv2.imwrite(new_im_path,image)
                #print(f"Saving image to {new_im_path}")
```

```
                #copyfile(old_path, new_im_path)
                #

    def add_all_sat_noises(self, original_image,  id_path, split, curr_class, image_name):
        stds = np.arange(0.0, 0.19, .02)
        for l,std in enumerate(stds):
            image = np.array(original_image).copy()
            #print(f"Standard deviation: {std}")
            image = self.increase_saturation_noise(image, std)
            if curr_class == 'dog':
                new_im_path = self.robust_path+"/"+id_path+"/"+str(l+1)+"/full_split_"+str(split)+
                    "/val/dog/"+image_name
                cv2.imwrite(new_im_path,image)
                #print(f"Saving image to {new_im_path}")
                #copyfile(old_path, new_im_path)
            else:
                new_im_path = self.robust_path+"/"+id_path+"/"+str(l+1)+"/full_split_"+str(split)+
                    "/val/cat/"+image_name
                cv2.imwrite(new_im_path,image)
                #print(f"Saving image to {new_im_path}")
                #copyfile(old_path, new_im_path)
                #

    def add_all_image_occlusions(self, original_image,  id_path, split, curr_class, image_name):
        edges = np.arange(0, 50, 5)
        for l,edge in enumerate(edges):
            image = np.array(original_image).copy()
            image = self.image_occlusion(image, edge, print_center=True)
            if curr_class == 'dog':
                new_im_path = self.robust_path+"/"+id_path+"/"+str(l+1)+"/full_split_"+str(split)+
                    "/val/dog/"+image_name
                cv2.imwrite(new_im_path,image)
                print(f"Saving image to {new_im_path}")
                #copyfile(old_path, new_im_path)
            else:
                new_im_path = self.robust_path+"/"+id_path+"/"+str(l+1)+"/full_split_"+str(split)+
                    "/val/cat/"+image_name
                cv2.imwrite(new_im_path,image)
                print(f"Saving image to {new_im_path}")
                #copyfile(old_path, new_im_path)

    def create_csv_files(self):
        # Create root directory
        # Create all directories
        print("Creating csvs...")
        for perturb_id in tqdm(self.perturb_ids):
            print(f"On perturbarbation level 5_{perturb_id}...")
            for perturb_level in self.perturb_levels:
                for split in self.splits:
                    # Create perturbation id folder
                    full_path = self.robust_path+'/5_'+str(perturb_id)+'/'+str(perturb_level)+'/
                        full_split_'+str(split)+"/"
                    full_path_dog = full_path+'val/dog/'
                    full_path_cat = full_path+'val/cat/'
                    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))])
                    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))])
                    imgs = np.concatenate([dog_imgs, cat_imgs])

                    df = pd.DataFrame({"image_id": imgs[:, 0], "label": imgs[:, 1]})
                    df.to_csv(full_path+"full_split_val_"+str(split)+".csv")


if __name__ == "__main__":
    # Create robustness folder placeholder
    perturbimages = PerturbImages()
    #perturbimages.create_placeholder_dirs()
    perturbimages.create_csv_files()
```