School of Informatics

Image and Vision Computing
Image classification using ResNet18 and Bag of Visual Words and robustness exploration

March 2021

Date: Thursday 25th March, 2021

1 Introduction

This report covers the results and analyses of Image and Vision Computing practical. Two classifiers, one based on ResNet18 and other on Bag Of Visual Words and Support Vector Machines (BoVW/SVM classifier) respectively, are trained to classify the images of cats and dogs with 3-fold cross-validation. Our task is to compare the robustness of two classifiers as progressively stronger perturbations are applied.

In this report, we first discuss the dataset as well as the structure and training of two classifiers respectively. Then we discuss the robustness of the two models to 9 different perturbations based on the changes in accuracy score. All experiments are implemented in Python language.

2 Dataset

Dataset for this task contains 1188 cat and 1200 dog face images all of which are 224 by 224 pixels. Each class has 12 subclasses (breeds) with approximately 100 examples, thus, the dataset is balanced. One of the images in the cat class (cat_6_32.png) was found to be grayscale. In order to handle it, all three colour channels were set to grayscale value.

When training our model, we split the dataset into training, validation and test sets. For this purpose, we defined class called `CatDogDataset` whose exact implementation varies slightly for each classifier. It returns valid image-label pairs for training, validation and test. In case of ResNet18, it inherits methods and properties from `torch.utils.data.Dataset`.

However, for both classifiers we split the data in exact same way ensuring as fair comparison as possible. According to the requirements of the assignment, we applied 3-fold cross-validation to our training process. We divided the image dataset into 3 equal parts (denoting as A, B and C). Two parts (e.g. A and B) formed the training and validation sets while the one left (e.g. C) became the test set in each cross-validation session. To improve the quality of training and testing, we kept the same distribution of classes and subclasses (breeds) in each part of the image dataset. More specifically, each part of the image dataset contains one-third of images from each subclass of both cats and dogs resulting in balanced splits of dataset.

3 ResNet18 Classifier

The implementation of the ResNet18 classifier is based on Pytroch [1] [2].

3.1 Loading Dataset

We imported `torch.utils.data.DataLoader`. It was used to process and schedule the dataset object created by CatDogDataset class, namely setting the batch size of the images and shuffling the images in each batch. We defined a function called `load_data` to create dataset objects for training, validation and testing respectively and used provided these three datasets in batches.

3.2 Data Augmentation

We imported `torchvision.transforms` which was used to achieve data augmentation. For the training set, it performed random horizontal and vertical flips. Afterwards, the images

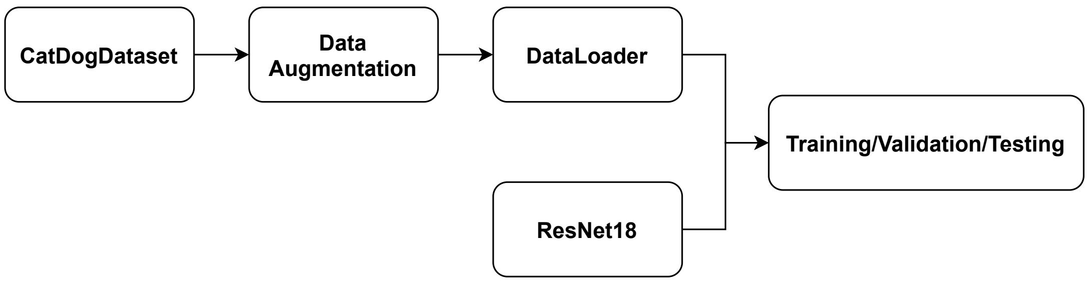

Figure 1: Components in the ResNet18 classifier

were rotated randomly from -90 degree to +90 degree. The images for validation and testing did not perform the augmentation above to reduce the noise. The images in all sets were normalised. According to a link in the assignment sheet, all pre-trained models expect input images normalised in the same way with $mean = [0.485, 0.456, 0.406]$ and $std = [0.229, 0.224, 0.225]$.

3.3 Network Model

The ResNet18 model [3] was imported as a pre-trained model from `torchvision.models.resnet18`. We "froze" its convolutional layers (no parameters updated during training) and switched the original fully connected layer (`nn.Linear(512, 1000)`) to a new fully connected layer with output size 2 (`nn.Linear(512, 2)`). The new fully connected layer was the only layer trained.

3.4 Classifier Training

Because we applied 3-fold cross-validation to our training process, three ResNet18 models were trained. Each model was trained on a different train-validation-test split.

During each model training, the aforementioned four components were connected to provide the data and model shown in Figure 1. At the last layer of the neural network, we applied the softmax function to the output and computed negative log likelihood loss against the ground truth. To update the parameters in the neural network, we adopted Adam Optimization Algorithm. The rest of hyperparameters used in the training are:

- learning rate = 1e-3
- weight decay = 1e-2
- batch size = 16
- number of epochs = 25

Because we used the pre-trained ResNet18 model, the validation accuracy reached its peak within 25 epochs, a low number by usual deep learning standards.

In the forward propagation, a batch of images were fed into the classifier to generate predicted labels each time. Then the loss between predicted labels and ground-true labels was calculated. The loss was backpropagated to compute gradients only for parameters in the fully connected layer and update them.

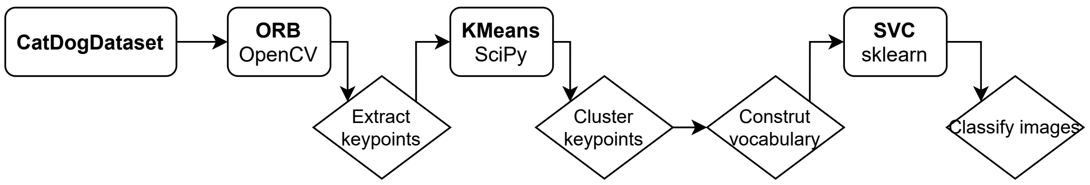

Figure 2: Components in the SVM classifier

4 Bag of Visual Words and SVM Classifier

Bag of Words (BoW) has demonstrated great success in natural language processing. Based on this success, it has become a very popular way to represent images in classification task [4]. Images are represented as a set of small patches that algorithms detect as "points of interest in the image". These features are then clustered and histogram is built which is then used to classify the images. That way, the location of interest points does not matter just as in text classification [5].

4.1 Classifier structure

The SVM based image classifier consists of several key steps, namely: extracting features from image, clustering features to produce visual vocabulary and classifying images based on this vocabulary. Our implementation was created based on several tutorials that exist online [5] [6] [7] [8]. Implementation code is provided in the appendix and figure 2 shows the structure of the model.

First of all, keypoints need to be extracted from the image that are used to describe the image. OpenCV [9] is a very popular open-source Computer Vision library in Python that provides multiple feature detectors. We intended to use SIFT (Scale-Invariant Feature Transform), however, it is patented and thus not available for free use. Instead, we use ORB (Oriented FAST and Rotated BRIEF) that claims to be equivalent in performance [10].

The obtained image descriptors (vectors) are then passed to clustering algorithm. In our implementation, we use KMeans algorithm implementation in the `SciPy` library. The algorithm then assigns each descriptor to the cluster. Then, each image is represented as a histogram of the codewords where each codeword describes how many image features are in particular cluster. Finally, the codewords are normalised to zero mean and unit variance and passed onto support vector machine (SVM) classifier. We used SVC from `sklearn` library [11].

4.2 Hyperparameter tuning

In order to tune the model and achieve as high score as possible on the dataset, an extensive model hyperparameter search has been performed. As the model consists of three major components, there are a lot of tunable parameters. Due to time and computational constraints, we attempted to select the ones that could performance the most based on existing similar work. Initially, Grid Search over the following parameters was conducted:

Component	Hyperparameter	Values
ORB *OpenCV*	nfeatures	250, 500, 1000, 5000, 10000
KMeans *SciPy*	k_or_guess	2, 10, 20, 40
SVC *sklearn*	kernel	linear, poly, rbf

In total, 60 combinations of hyperparamters were tried. We selected hyperparameters based on model performance on validation set. Presented accuracy scores are an average over 3 folds, i.e. for every fold same set of hyperparameters were used at each run. The accuracy scores ranged widely from 0.479 (worse than random guessing) to 0.733 (it was achieved using 1000 features, 40 clusters and rbf kernel). Full results are provided in the appendix B.2 that are sorted by validation set performance.

By default, ORB keypoint detector retains up to 500 features, however, it was not sufficient for this dataset and model performs better when twice as many features are retained. Any further increase seems to result in detecting insignificant features and overfitting the model. The more clusters there were, the better performance in general seemed to be. Rbf kernel for SVM classifier performed significantly better than other kernels and linear kernel seemed to be worst overall. Interestingly, polynomial kernel had much more significant difference between training and validation set performance than other kernels and achieved highest score on training set (0.899). It was observed that kernel choice for SVM and number of clusters had the greatest impact on accuracy score. The best set of hyperparameters were run on test set which yielded accuracy score of 0.693 (1000, 40, rbf).

However, after conducting the hyperparameter search, the results were still quite disappointing. They were significantly lower than those of ResNet18. After some research it was found that the choice of clustering algorithm could be the causing factor. First of all, it was suspected that number of clusters could be too small. Therefore, additional tests were carried out with previously found best hyperparameters: 1000 features in ORB detector and rbf kerne. 400 and 1000 clusters were tested and results were better:

Number of clusters	Training set accuracy	Validation set accuracy
400	0.945	0.764
1000	0.965	0.761

Accuracy on validation set with 1000 clusters decreased slightly even though train set performance increase was quite significant suggesting that model was too complex. Thus, for the final model that was used in its robustness testing, 400 clusters, rbf kernel and 1000 features were selected. It achieved 0.737 on test set.

In addition to that, late into the project it was discovered that KMeans clustering algorithm performance varies significantly based on its implementation [12] [13]. It is widely agreed that KMeans implementation by sklearn is superior to SciPy implementation. These results suggest that scores can vary widely depending of the model implementation. To our surprise, after hyperparameter search using KMeans implementation by `sklearn`, results on validation set did not improve and it was decided to stick to existing implementation. It is evident that further tests should be carried out with different set of hyperparameters and potentially different implementations of algorithms in order to achieve higher performance and make interesting discoveries.

5 Robustness Explorations

This section covers the results and analyses of robustness exploration experiments. 9 kinds of perturbations were applied to the images. For each kind of perturbation, 9 increasing levels of perturbation were applied to the three test sets. Two classifiers trained on "clean dataset" were applied to these perturbed test sets respectively. We recorded the mean and standard deviation of the accuracy on three test sets under each of the perturbations for each classifier. Figure 3 shows an example image without any perturbation. For each perturbation, we show an example image with the corresponding maximal perturbation (left), ResNet18 and BoVW/SVM results (right). We then briefly state the perturbation algorithm and our analysis on the robustness of the 2 classification approaches.

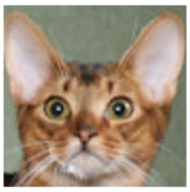

Figure 3: Original image (cat_1_0.png)

5.1 Gaussian pixel noise

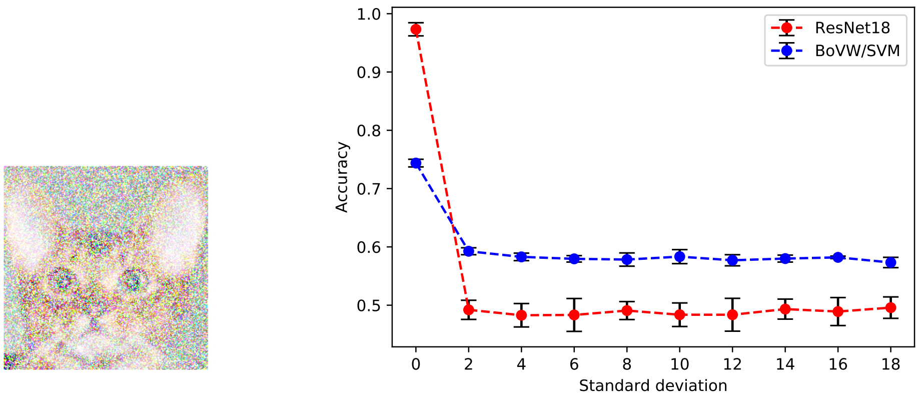

Figure 4: Gaussian pixel noise

To apply the Gaussian pixel noise perturbation, we generated a 3-dimensional array with each pixel value sampled from a Gaussian distribution and added it to the original image. The level of perturbation is represented by the standard deviation of the Gaussian distribution. Figure 4 shows the accuracy of both models when increasing the level of perturbation.

Both classifiers suffered a drastic decrease in accuracy after the lowest-level perturbation was applied. BoVW/SVM is relatively more robust in this case due to its smaller decrease in accuracy (around 15%) compared to that of ResNet18 (around 40%). The performance of the ResNet18 classifier is close to random guessing.

The images after the perturbation are still recognizable to the naked eye. But we believe the Gaussian pixel noise has a relatively big impact on convolution and interest point detection because the random changes in the local areas disturb feature extraction of both methods. This demonstrates how algorithms are prone to Gaussian noise attacks. However, contrary to expectation, the strength of Gaussian noise almost did not alter how much the performance itself changed.

5.2 Gaussian blurring

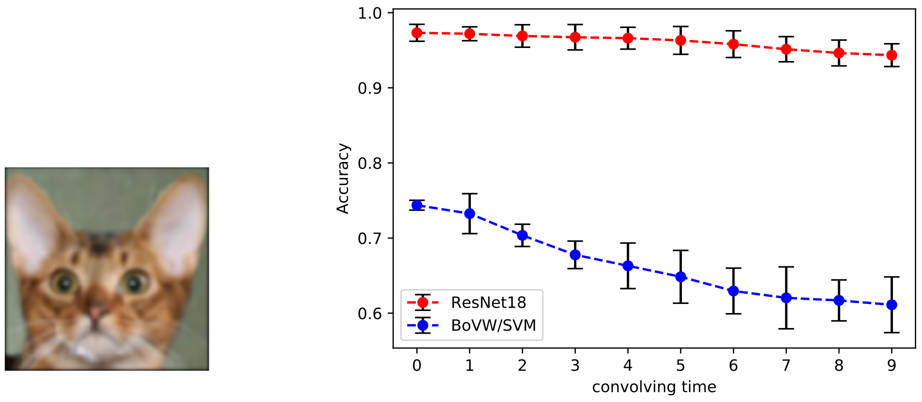

Figure 5: Gaussian blurring

We convolved the original images with the 3×3 mask specified in the assignment sheet to achieve Gaussian blurring. With an increasing perturbation level, we increased the times of convolution to approximate Gaussian blurring with increasingly larger standard deviations.

Compared with the previous Gaussian pixel noise perturbation, Gaussian blurring has less impact on the accuracy of both ResNet18 (3% decrease) and BoVW/SVM (9% decrease). While convolving with a mask makes the image blurry, the local features are preserved. Therefore, Gaussian blurring will not affect convolution or interest point detection too much. We think the ResNet18 classifier is more robust against Gaussian Blurring because its convolutional layers have the same nature of convolving a kernel (mask) with the image.

5.3 Image contrast increase and decrease

To increase the contrast of an image, we multiplied each pixel value of the image by an increase factor (a value greater than 1). All pixel values were capped at 255. To decrease the contrast of an image, we multiplied each pixel value of the image by an decrease factor (a positive value smaller than 1).

When the contrast of images increases, the accuracy of both ResNet18 and BoVW/SVM is relatively stable. The ResNet18 classifier is slightly more robust with less fluctuation in terms of accuracy. It is worth noting that the accuracy of BoVW/SVM has a downward trend when the perturbation level is high.

The decrease in image contrast has little effects on the ResNet18 classifier. However, the accuracy of the ResNet18 classifier plunged by 9% when the highest perturbation was applied.

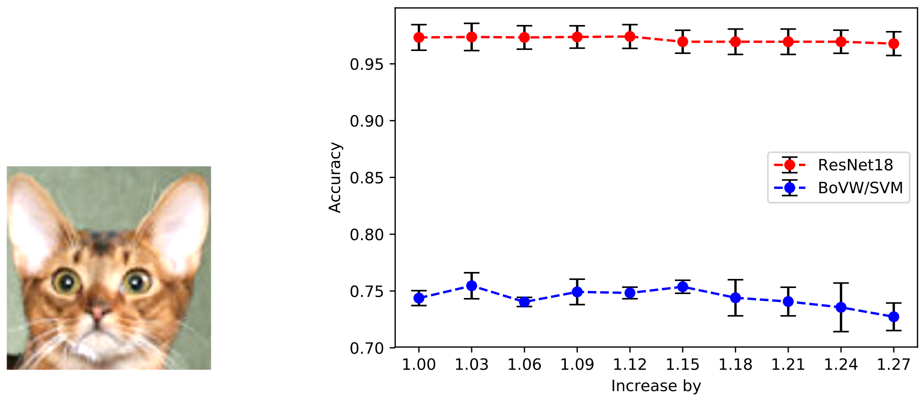

Figure 6: Image contrast increase

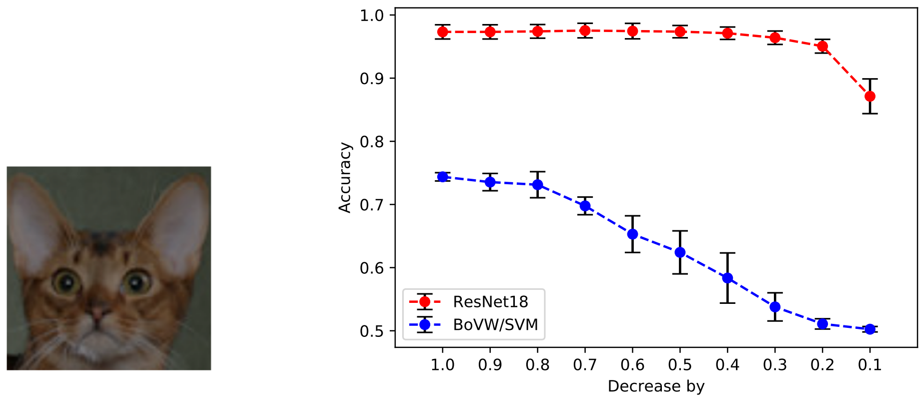

Figure 7: Image contrast decrease

The accuracy for BoVW/SVM classifier witnessed a steady decline of 25%. This means the ResNet18 classifier is more robust under the low brightness and low contrast condition.

Also note that increasing the contrast of an image will potentially distort the image because all the pixel values are capped at 255, which surprisingly does not affect the accuracy that much. Another reason for this behaviour could be that images in dataset in general have less contrast and less change in contrast decrease is necessary for the performance of models to reduce.

5.4 Image brightness increase and decrease

To increase the brightness of an image, we added a positive value to each pixel value of the image. All pixel values were capped at 255. To decrease the brightness of an image, we subtracted a positive value from each pixel value of the image. All negative pixel values after the subtraction were changed to 0.

Both image brightness increase and decrease have little impact on the classification accuracy. Increasing brightness or decreasing brightness directly changes the pixel values by a certain amount. Because the pixel value ranges from 0 - 255, it is likely that the different pixel values receive different changes. This distortion of the image does not affect the classification accuracy

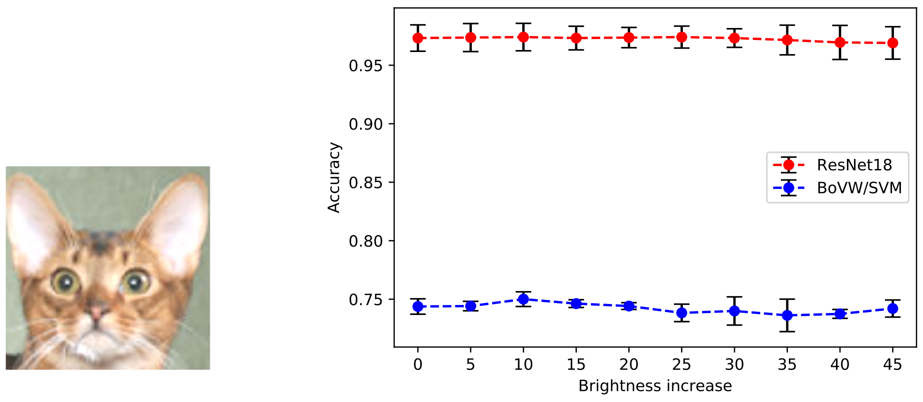

Figure 8: Image brightness increase

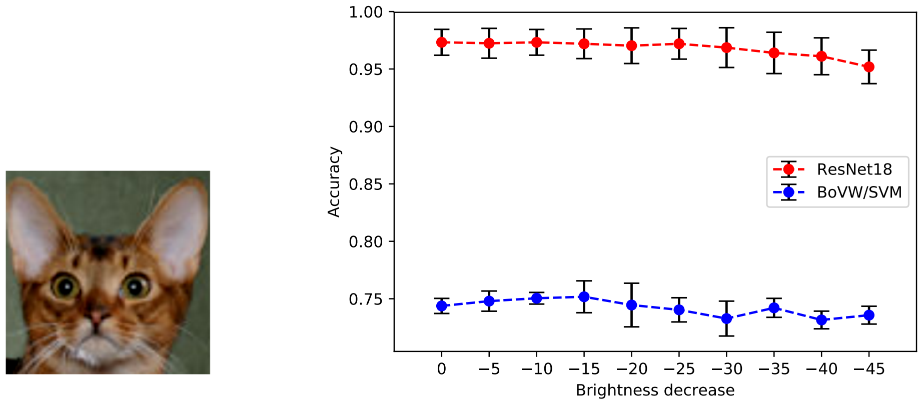

Figure 9: Image brightness decrease

that much similar to what we saw in the image contrast increase. Decreasing the brightness of images did not lower the overall brightness as much as the image contrast decrease in the last section. It might be the reason why the accuracy in the low brightness is relatively stable in this section.

5.5 HSV Hue noise increase

We first converted an image from RGB to HSV. We then generated a 2d-array with each value sampled from a Gaussian distribution and added it to the fist channel of the HSV array (Hue). Next, the HSV array was converted back to RGB.

BoVW/SVM is more robust than ResNet18 when Hue noise increasing. The accuracy of ResNet18 classifier declines by around 3% during the course of increasing noise while the accuracy of BoVW/SVM fluctuates around 0.74.

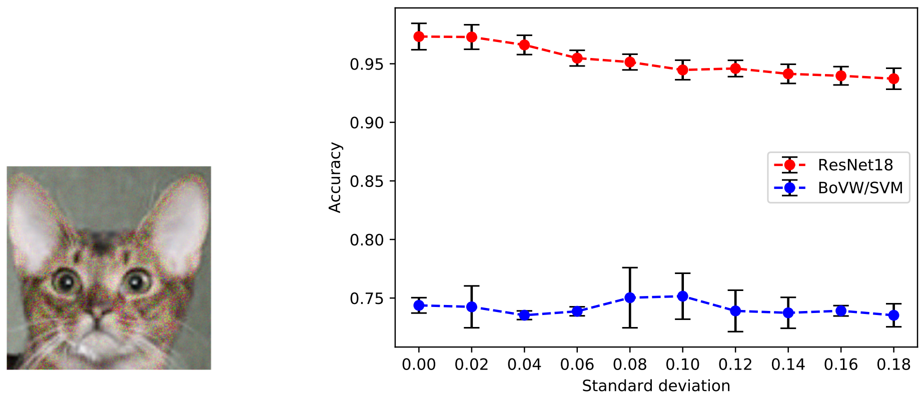

Figure 10: HSV Hue noise increase

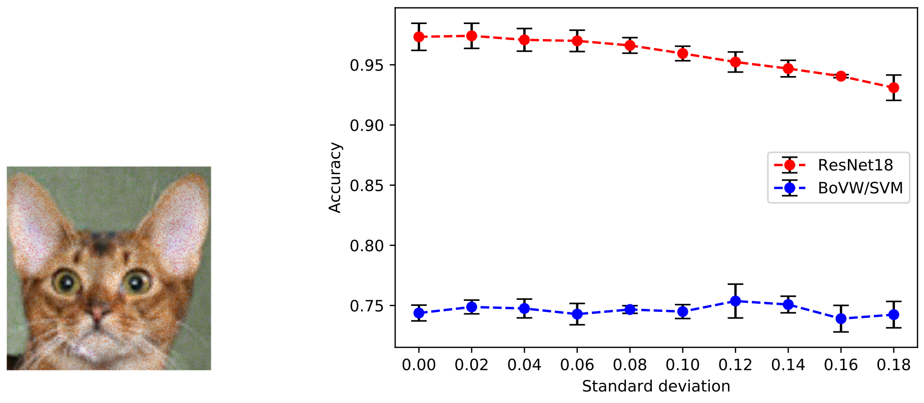

Figure 11: HSV Saturation noise increase

5.6 HSV Saturation and hue noise increase

We first converted an image from RGB to HSV. We then generated a 2d-array with each value sampled from a Gaussian distribution and added it to the second channel of the HSV array (Saturation). Next, the HSV array was converted back to RGB.

BoVW/SVM is more robust than ResNet18 when Saturation noise increasing. The accuracy of ResNet18 classifier declines by around 4% during the course of increasing noise while the accuracy of BoVW/SVM fluctuates around 0.74. It seems that BoVW/SVM is more robust against perturbation on Hue and Saturation in general.

5.7 Occlusion of the image increase

To implement occlusion of the image, We randomly chose the position of a square region in an image and set all the pixel values in that region to 0. We increased the edge length of the square region to represent a larger scale of occlusion.

The increase of occlusion does not affect the accuracy of both classifiers too much. Their robustness against the occlusion means than both ResNet18 and BoVW rely on features from

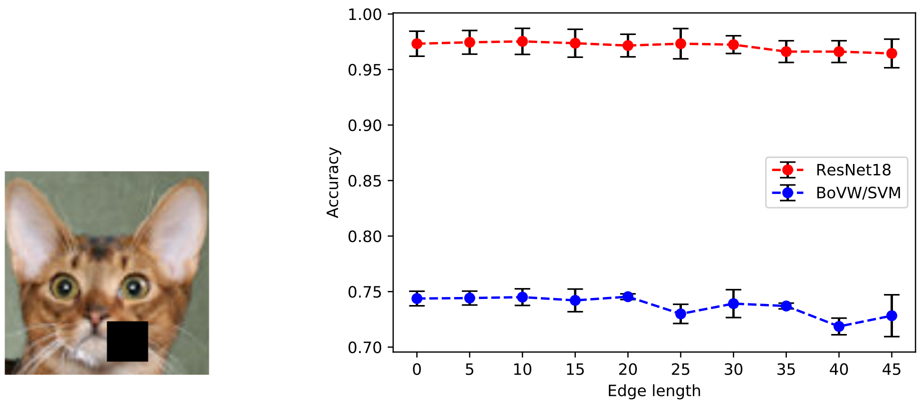

Figure 12: Occlusion of the image increase

different parts of the image that are unaffected by this perturbation. Removing some part of images will not drastically hurt the classification performance. Should significantly larger part of image be removed, the performance would likely be affected more strongly.

6 Summary

These experiments highlight similarities and differences between ResNet18 and BoVW/SVM classifiers. ResNet18 achieves significantly higher performance on the unmodified dataset as well as when perturbations of varying intensity are added. This means ResNet18 could extract more useful features under most of tested conditions. The only exception is adding Gaussian pixel noise to the images where ResNet18 is virtually guessing and BoVW/SVM classifier manages to achieve accuracy of around 0.6 with very low standard deviation. The ResNet18 classifier achieves higher performance because we believe it is better at feature extraction. Deep convolutional neural networks like ResNet18 have the ability to approximate more complex function to represent the data and hence it is easy to classify them. BoVW, however, cannot represent data in a way that it is as easy for SVM to classify them. However, this claim needs to be backed up by carrying out further experiments to test BoVW and ResNet18's ability of feature extraction.

References

[1] PyTorch. Transfer learning for computer vision tutorial.

[2] PyTorch. Pytorch framework. `https://pytorch.org/`, 2020.

[3] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. *CoRR*, abs/1512.03385, 2015.

[4] Zhi-Hua Zhou Yin Zhang, Rong Jin. Understanding bag-of-words model: a statistical framework. 2010.

[5] Sachin Mohan. Image classification using bag of visual words model.

[6] Aybüke Yalçıner. Bag of visual words (bovw).

[7] No author. Bag-of-words model with sift descriptors.

[8] Sreenivas Bhattiprolu. Train bov. https://github.com/bnsreenu/python_for_microscopists/blob/master/069a-Train_BOVW_V1.0.py, 2020.

[9] OpenCV. Opencv library. https://opencv.org/releases/, 2020.

[10] OpenCV. Orb (oriented fast and rotated brief). https://opencv-python-tutroals.readthedocs.io/en/latest/py_tutorials/py_feature2d/py_orb/py_orb.html#orb, 2011.

[11] sklearn.svm.svc. https://scikit-learn.org/stable/modules/generated/sklearn.svm.SVC.html, 2021.

[12] Are there advantages of using sklearn kmeans versus scipy kmeans? https://stackoverflow.com/questions/37210795/are-there-advantages-of-using-sklearn-kmeans-versus-scipy-kmeans, 2016.

[13] Scipy kmeans vs sklearn kmeans performance. https://github.com/scikit-learn-contrib/hdbscan/issues/285, 2019.

Appendices

A ResNet18

```
import torch
import torch.nn as nn
import torch.optim as optim
import numpy as np
from torch.utils.data import Dataset, DataLoader
from torchvision import transforms, utils
import os, os.path
import random
from skimage import io, transform
import torchvision.models as models
from PIL import Image
import time
import copy

# Label: cat == 0; dog == 1

# the number of cat/dog pics for each breed
NUM_CATS = [99, 98, 100, 100, 100, 92, 100, 100, 99, 100, 100, 100]
NUM_DOGS = [100 for i in range(12)]


# define a dataset class to streamline the training, validation and testing
# 3-fold cross validation
# A&B -> C, then repeat for A&C -> B and B&C -> A (according to an email from Bob)
# data is split into 3 equal parts: A, B, C
# training and validation set is split 75%/25%

class CatDogDataset(Dataset):
    def __init__(self, root_dir, set_name, transform=None):
        # root directory of the dataset is usually '../catdog/'
        self.root_dir = root_dir
        # set_name is used to identify which part (A, B or C) is testing set
        # and determine the output of the dataset is training/validation/test
        self.set_name = set_name
        # apply transform if needed
        self.transform = transform

        # rng (random number generator) is used to generate random indices for part A, B and C
        # we use a seed to make sure the split is fixed in each run
        rng = np.random.RandomState(seed=0)

        # initialise empty lists to store names for different image sets
        # we will use the names of images to retrieve images later
        cat_A, cat_B, cat_C, dog_A, dog_B, dog_C = [], [], [], [], [], []

        # we use for loop to sample images from each breed of dogs and cats
        # to make sure set A, B and C have the same data distribution
        for b in range(1, 13):
            # here we sample image indices for each breed
            # for each breed (around 100 images), sample 66 images for set catA and catB
            catAncatB = rng.choice(a=[i for i in range(NUM_CATS[b - 1])], size=66, replace=False)

            # catA samples 33 images from combined catAncatB set
            catA = rng.choice(a=catAncatB, size=33, replace=False)

            # the rest of images in combined catAncatB set will go to catB
            catB = [i for i in catAncatB if i not in catA]

            # the rest of images in the original set will go to catC
            catC = [i for i in range(NUM_CATS[b - 1]) if i not in catAncatB]

            # use the image indices to build the name/path of the images in each set
            # 0 is the label for cats
            cat_A += [['CATS/cat_' + str(b) + '_' + str(i) + '.png', 0] for i in catA]
            cat_B += [['CATS/cat_' + str(b) + '_' + str(i) + '.png', 0] for i in catB]
            cat_C += [['CATS/cat_' + str(b) + '_' + str(i) + '.png', 0] for i in catC]

            # it is the same process but for dogs
            # 1 is the label for dogs
            dogAndogB = rng.choice(a=[i for i in range(NUM_DOGS[b - 1])], size=66, replace=False)
            dogA = rng.choice(a=dogAndogB, size=33, replace=False)
            dogB = [i for i in dogAndogB if i not in dogA]
            dogC = [i for i in range(NUM_DOGS[b - 1]) if i not in dogAndogB]

            dog_A += [['DOGS/dog_' + str(b) + '_' + str(i) + '.png', 1] for i in dogA]
            dog_B += [['DOGS/dog_' + str(b) + '_' + str(i) + '.png', 1] for i in dogB]
            dog_C += [['DOGS/dog_' + str(b) + '_' + str(i) + '.png', 1] for i in dogC]

        # combine cat and dog images to form complete set A, B and C, and reshuffle
        catdogA = cat_A + dog_A
        catdogB = cat_B + dog_B
        catdogC = cat_C + dog_C
        # use 0 as the seed to make sure the reproducibility
        random.Random(0).shuffle(catdogA)
        random.Random(0).shuffle(catdogB)
        random.Random(0).shuffle(catdogC)

        # return a dataset depending on the set_name
        # first uppercase letter represents the test dataset in the current split
        # train/val/test represents the dataset returned in the current split

        if self.set_name == 'C_train':
            # when set C is the test set, set A and B combined will be training and validation set
            self.data = catdogA + catdogB
            random.Random(1).shuffle(self.data)
            # with 75%/25% split, first 1200 images will be training set
            self.data = self.data[:1200]
        elif self.set_name == 'C_val':
            self.data = catdogA + catdogB
            random.Random(1).shuffle(self.data)
            # the rest of it will be validation set
            self.data = self.data[1200:]
        elif self.set_name == 'C_test':
            self.data = catdogC
        elif self.set_name == 'B_train':
            self.data = catdogA + catdogC
            random.Random(1).shuffle(self.data)
            self.data = self.data[:1200]
        elif self.set_name == 'B_val':
            self.data = catdogA + catdogC
            random.Random(1).shuffle(self.data)
            self.data = self.data[1200:]
        elif self.set_name == 'B_test':
            self.data = catdogB
        elif self.set_name == 'A_train':
            self.data = catdogB + catdogC
            random.Random(1).shuffle(self.data)
            self.data = self.data[:1200]
        elif self.set_name == 'A_val':
            self.data = catdogB + catdogC
            random.Random(1).shuffle(self.data)
            self.data = self.data[1200:]
        elif self.set_name == 'A_test':
            self.data = catdogA

    def __len__(self):

        return len(self.data)

    def __getitem__(self, index):

        image_name, target = self.data[index][0], self.data[index][1]
        img = io.imread(self.root_dir + image_name)

        # there is a grayscale image in the cat images
        # we will convert it to a colour image by duplicating it across RGB channels
        if len(img.shape) == 2:
            img = np.repeat(img[:, :, np.newaxis], 3, axis=2)

        img = Image.fromarray(img)

        if self.transform:
            img = self.transform(img)
        return img, np.array([target])


# Data augmentation is limited to translation, rotation, flipping and scale variations
data_transforms = {
    'train': transforms.Compose([
        # apply random flips to make networks more robust
        transforms.RandomHorizontalFlip(),
        transforms.RandomVerticalFlip(),
        transforms.RandomRotation(degrees=(-90, 90)),
        transforms.ToTensor(),
        # normalise the images
        # the first tuple is the mean in each channel, the second one is the std
        # these numbers are pre-calculated in
        # https://pytorch.org/docs/stable/torchvision/models.html
        transforms.Normalize((0.485, 0.456, 0.406), (0.229, 0.224, 0.225))
    ]),
    'val': transforms.Compose([
        transforms.ToTensor(),
        transforms.Normalize((0.485, 0.456, 0.406), (0.229, 0.224, 0.225))
    ]),
    'test': transforms.Compose([
        transforms.ToTensor(),
        transforms.Normalize((0.485, 0.456, 0.406), (0.229, 0.224, 0.225))
    ])
}


def load_data(setname, batch_size=4, root_dir='../catdog/'):
    image_datasets = {x: CatDogDataset(root_dir=root_dir,
                                       set_name=setname + '_' + x,
                                       transform=data_transforms[x])
                      for x in ['train', 'val', 'test']}

    dataloaders = {x: DataLoader(image_datasets[x],
                                 batch_size=batch_size,
                                 shuffle=True,
                                 num_workers=2)
                   for x in ['train', 'val', 'test']}

    dataset_sizes = {x: len(image_datasets[x]) for x in ['train', 'val', 'test']}

    return dataloaders, dataset_sizes


def train_model(model, criterion, optimizer, model_path, num_epochs=50):
    since = time.time()

    best_model_wts = copy.deepcopy(model.state_dict())

    best_acc = 0.0
    best_running_corrects = 0

    acc_list_train = []
```

```
acc_list_val = []

for epoch in range(num_epochs):
    print('Epoch {}/{}'.format(epoch, num_epochs - 1))
    print('-' * 10)

    # Each epoch has a training and validation phase
    for phase in ['train', 'val']:
        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.
        for inputs, labels in dataloaders[phase]:
            inputs = inputs.to(device)
            labels = labels.to(device)

            labels = labels.squeeze(1)
            # zero the parameter gradients
            optimizer.zero_grad()

            with torch.set_grad_enabled(phase == 'train'):
                outputs = model(inputs)
                _, preds = torch.max(outputs, 1)
                loss = criterion(outputs, labels)

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

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

        epoch_loss = running_loss / dataset_sizes[phase]
        epoch_acc = running_corrects.double() / dataset_sizes[phase]

        if phase == 'train':
            acc_list_train.append(epoch_acc)
        elif phase == 'val':
            acc_list_val.append(epoch_acc)

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

        if phase == 'val' and epoch_acc > best_acc:
            best_acc = epoch_acc
            best_running_corrects = running_corrects
            best_model_wts = copy.deepcopy(model.state_dict())

    print()
    time_elapsed = time.time() - since
    print('Training complete in {:.0f}m {:.0f}s'.format(
        time_elapsed // 60, time_elapsed % 60))
    print('Best val Acc: {:4f}'.format(best_acc))

    torch.save(best_model_wts, model_path)

    model.load_state_dict(best_model_wts)

    return model, best_acc, best_running_corrects, \
           dataset_sizes['val'], acc_list_train, acc_list_val


def test_model(model):
    model.eval()
    running_loss = 0.0
    running_corrects = 0
    for inputs, labels in dataloaders['test']:
        inputs = inputs.to(device)
        labels = labels.to(device)

        labels = labels.squeeze(1)

        with torch.set_grad_enabled(False):
            outputs = model(inputs)
            _, preds = torch.max(outputs, 1)
            loss = criterion(outputs, labels)

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

    epoch_loss = running_loss / dataset_sizes['test']
    epoch_acc = running_corrects.double() / dataset_sizes['test']

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

    return epoch_acc, running_corrects


# hyper-parameters
learning_rate = 1e-3
weight_decay = 1e-2
batch_size = 16
num_epochs = 25

# use gpu if it is available
device = torch.device("cuda:0" if torch.cuda.is_available() else "cpu")

# three-fold validation

# use different set (A, B or C) as test set
testsets = ['A', 'B', 'C']
```

```
corrects_val = []
val_sizes = []
corrects_test = []

for testset in testsets:

    print('#' * 20)
    print('Use ' + testset + ' as test set')
    print('#' * 20)

    dataloaders, dataset_sizes = load_data(setname=testset, batch_size=batch_size)

    resnet18 = models.resnet18(pretrained=True)

    # ‘freezing’ the convolutional layers except for the fully connected layer
    for param in resnet18.parameters():
        param.requires_grad = False

    # when assigning a new layer, requires_grad will be set to True automatically
    resnet18.fc = nn.Linear(512, 2)

    criterion = nn.CrossEntropyLoss()
    optimizer = torch.optim.Adam(resnet18.parameters(),
                                 lr=learning_rate, weight_decay=weight_decay)
    model_path = './models/' + testset + '_astest'

    resnet18 = resnet18.to(device)

    # training
    resnet18, acc_val, correct_val, val_size, acc_list_train, acc_list_val = \
        train_model(resnet18, criterion, optimizer,
                    model_path=model_path,
                    num_epochs=num_epochs)

    # test
    acc_test, correct_test = test_model(resnet18)

    corrects_val.append(correct_val)
    corrects_test.append(correct_test)
    val_sizes.append(val_size)

average_acc_val = sum(corrects_val).double() / sum(val_sizes)
average_acc_test = sum(corrects_test).double() / 2388

print('val average acc {:.4f} '.format(average_acc_val))
print('test average acc {:.4f} '.format(average_acc_test))
```

B SVM

B.1 Grid Search of hyperparameters

```
import cv2
import numpy as np
import os
import random
import pandas as pd
from skimage import io, transform
from PIL import Image
import matplotlib
from matplotlib import pyplot as plt
from scipy.cluster.vq import kmeans, vq
from sklearn.cluster import KMeans
from sklearn.preprocessing import StandardScaler
from sklearn.svm import LinearSVC
from sklearn.svm import SVC
import joblib
from sklearn.metrics import confusion_matrix, accuracy_score #sreeni
from scipy import signal

# Three-fold validation

# Use different set (A, B or C) as test set
testsets = ['A', 'B', 'C']

# Tested hyperparameter values
features_set = [250, 500, 1000, 5000, 10000]
number_clusters_set = [2, 10, 20, 40]
kernel_set = {'linear', 'poly', 'rbf'}

mean_accuracies_train = []
mean_accuracies_val = []

for nfeatures in nfeatures_set:
    for number_clusters in number_clusters_set:
        for kernel in kernel_set:

            accuracies_train = []
            accuracies_val = []

            print('\n', '#' * 40)
            print('Hyperparameters:')
            print('nfeatures=', nfeatures)
            print('number_of_clusters=', number_clusters)
            print('kernel=', kernel,'\n')

            for testset in testsets:

                """
                Train
                """
                # Find keypoints in images
                descriptors_list = []
                orb = cv2.ORB_create(nfeatures=nfeatures)

                image_datasets, dataset_sizes = load_data(setname=testset)

                for image in image_datasets['train']:
                    im = cv2.imread(image[0])
                    kp = orb.detect(im,None)
                    keypoints, descriptor = orb.compute(im, kp)
                    descriptors_list.append((image,descriptor))

                # Stack all the descriptors vertically in a numpy array
                descriptors = descriptors_list[0][1]
                for image_path, descriptor in descriptors_list[1:]:
                    descriptors = np.vstack((descriptors, descriptor))


                # kmeans works only on float, so convert integers to float
                descriptors_float = descriptors.astype(float) # maybe don't need

                # K-Means clustering
                # Perform k-means clustering and vector quantization

                k = number_clusters
                voc, variance = kmeans(descriptors_float, k, 1)

                # Calculate the histogram of features and represent them as vector
                #vq Assigns codes from a code book to observations.
                im_features = np.zeros((len(image_datasets['train']), k), "float32")
                for i in range(len(image_datasets['train'])):
                    words, distance = vq(descriptors_list[i][1],voc)
                    for w in words:
                        im_features[i][w] += 1


                # Get target class labels
                image_classes_train = [x[1][0] for x in image_datasets['train']]
                image_classes_train = np.array(image_classes_train)

                # Scaling the words
                # Normalization - Standardize features by removing the
                # mean and scaling to unit variance.
                stdSlr = StandardScaler().fit(im_features)
                im_features = stdSlr.transform(im_features)


                # Train an algorithm to discriminate vectors corresponding to positive
                # and negative training images.
                clf = SVC(kernel=kernel)
                clf.fit(im_features, np.array(image_classes_train))

                true_classes_train=[]
                for i in image_classes_train:
                    if i==0:
                        true_classes_train.append("Cat")
                    else:
                        true_classes_train.append("Dog")

                predict_classes_train=[]
                for i in clf.predict(im_features):
                    if i==0:
                        predict_classes_train.append("Cat")
                    else:
```

```
                predict_classes_train.append("Dog")

        accuracy_train = accuracy_score(true_classes_train, predict_classes_train)
        accuracies_train.append(accuracy_train)


        """
        Validation
        """

        image_classes_val = [x[1][0] for x in image_datasets['val']]
        image_classes_val = np.array(image_classes_val)

        description_list_val=[]
        for image in image_datasets['val']:
            im = cv2.imread(image[0])
            kp = orb.detect(im,None)
            keypoints_test, descriptor_val = orb.compute(im, kp)
            description_list_val.append((image,descriptor_val))

        test_features = np.zeros((len(image_datasets['val']), k), "float32")
        for i in range(len(image_datasets['val'])):
            words, distance = vq(description_list_val[i][1],voc)
            for w in words:
                test_features[i][w] += 1

        test_features = stdSlr.transform(test_features)

        true_classes_val=[]
        for i in image_classes_val:
            if i==0:
                true_classes_val.append("Cat")
            else:
                true_classes_val.append("Dog")

        predict_classes_val=[]
        for i in clf.predict(test_features):
            if i==0:
                predict_classes_val.append("Cat")
            else:
                predict_classes_val.append("Dog")

        accuracy_val = accuracy_score(true_classes_val, predict_classes_val)
        accuracies_val.append(accuracy_val)

    mean_accuracy_train = np.mean(accuracies_train)
    mean_accuracies_train.append(mean_accuracy_train)
    print('Mean train accuracy: ', mean_accuracy_train)
    mean_accuracy_val = np.mean(accuracies_val)
    mean_accuracies_val.append(mean_accuracy_val)
    print('Mean validation accuracy: ', mean_accuracy_val)
```

B.2 Hyperparameter search summary

Unnamed: 0	Features to detect	Clusters	SVM Kernel	Train accuracy	Validation accuracy
34	1000	40	rbf	0.7994439999999999	0.733007
46	5000	40	rbf	0.763333	0.719197
58	10000	40	rbf	0.7525	0.70907
22	500	40	rbf	0.819167	0.703862
43	5000	20	rbf	0.714167	0.694444
10	250	40	rbf	0.836111	0.69284
31	1000	20	rbf	0.7375	0.688447
33	1000	40	poly	0.814167	0.678399
16	500	10	rbf	0.7061109999999999	0.674795
35	1000	40	linear	0.688333	0.674058
47	5000	40	linear	0.6752779999999999	0.671533
19	500	20	rbf	0.761389	0.668534
28	1000	10	rbf	0.6838890000000001	0.666535
59	10000	40	linear	0.675833	0.663431
7	250	20	rbf	0.772778	0.654488
55	10000	20	rbf	0.699444	0.653777
23	500	40	linear	0.691389	0.651989
52	10000	10	rbf	0.6675	0.650963
21	500	40	poly	0.8688889999999999	0.650305
40	5000	10	rbf	0.668611	0.646596
9	250	40	poly	0.898611	0.644492
44	5000	20	linear	0.6286109999999999	0.641519
11	250	40	linear	0.665	0.638468
30	1000	20	poly	0.724722	0.638336
4	250	10	rbf	0.707778	0.636995
32	1000	20	linear	0.623333	0.635785
20	500	20	linear	0.626667	0.631366
56	10000	20	linear	0.615278	0.630419
15	500	10	poly	0.679167	0.62913
6	250	20	poly	0.789167	0.626526
18	500	20	poly	0.770278	0.621344
45	5000	40	poly	0.7011109999999999	0.618424
27	1000	10	poly	0.651389	0.614662
53	10000	10	linear	0.614167	0.605955
8	250	20	linear	0.640556	0.602562
25	1000	2	rbf	0.5847220000000001	0.601904
41	5000	10	linear	0.613611	0.600773
57	10000	40	poly	0.687778	0.597485
29	1000	10	linear	0.6013890000000001	0.593987
17	500	10	linear	0.6286109999999999	0.593382
13	500	2	rbf	0.583333	0.58841
3	250	10	poly	0.665833	0.58699
5	250	10	linear	0.608889	0.579598
37	5000	2	rbf	0.5755560000000001	0.573995
49	10000	2	rbf	0.578056	0.573127
1	250	2	rbf	0.584167	0.570733
54	10000	20	poly	0.610556	0.54577
42	5000	20	poly	0.5955560000000001	0.537405
39	5000	10	poly	0.566667	0.522017
51	10000	10	poly	0.568889	0.515125
12	500	2	poly	0.5344439999999999	0.514652
0	250	2	poly	0.525556	0.506866
26	1000	2	linear	0.526389	0.505524
24	1000	2	poly	0.5236109999999999	0.504419
2	250	2	linear	0.516111	0.49760600000000005
14	500	2	linear	0.526111	0.49642299999999995
36	5000	2	poly	0.521944	0.49502799999999997
48	10000	2	poly	0.5222220000000001	0.49331899999999995
38	5000	2	linear	0.515	0.48213900000000004
50	10000	2	linear	0.515556	0.47953500000000004

C Perturbation classes (extending CatDogDataset)

```python
# Gaussian pixel noise
class GPN(CatDogDataset):
    def __init__(self, root_dir, set_name, transform, std):
        super().__init__(root_dir, set_name, transform)
        self.std = std

    def __getitem__(self, index):

        image_name, target = self.data[index][0], self.data[index][1]
        img = io.imread(self.root_dir + image_name)

        if len(img.shape) == 2:
            img = np.repeat(img[:, :, np.newaxis], 3, axis=2)

                noise = np.rint(np.random.normal(0, self.std, (224, 224, 3))).astype(np.uint8)
                mask = (255 - img) < noise
                img = np.where(mask, 255, img + noise)
        img = Image.fromarray(img)

        if self.transform:
            img = self.transform(img)

        return img, np.array([target])

# Gaussian blurring
class GB(CatDogDataset):
    def __init__(self, root_dir, set_name, transform, convolving_times):
        super().__init__(root_dir, set_name, transform)
        self.convolving_times = convolving_times

    def __getitem__(self, index):
        image_name, target = self.data[index][0], self.data[index][1]
        img = io.imread(self.root_dir + image_name)

        if len(img.shape) == 2:
            img = np.repeat(img[:, :, np.newaxis], 3, axis=2)

        mask = 1/16 * np.array([[1, 2, 1],
                                [2, 4, 2],
                                [1, 2, 1]])

        for n in range(self.convolving_times):
            for c in range(3):
                img[:, :, c] = signal.correlate2d(img[:,:,c], mask,
                                                  boundary='fill', mode='same')

        img = np.clip(img, 0, 255)
        img = Image.fromarray(img)

        if self.transform:
            img = self.transform(img)

        return img, np.array([target])
```

```
# Image contrast increase/decrease
class ICI(CatDogDataset):
    def __init__(self, root_dir, set_name, transform, contrast):
        super().__init__(root_dir, set_name, transform)
        self.contrast = contrast

    def __getitem__(self, index):
        image_name, target = self.data[index][0], self.data[index][1]
        img = io.imread(self.root_dir + image_name)

        if len(img.shape) == 2:
            img = np.repeat(img[:, :, np.newaxis], 3, axis=2)

                if self.contrast > 1.0:
                        mask = (255 - img) < img * (self.contrast - 1.0)
                        img = np.where(mask, 255, img * self.contrast)
                        img = np.rint(img).astype(np.uint8)
                else:
                        img = np.rint(self.contrast * img).astype(np.uint8)
        img = Image.fromarray(img)

        if self.transform:
            img = self.transform(img)

        return img, np.array([target])

# Image brightness increase/decrease
class IB(CatDogDataset):
    def __init__(self, root_dir, set_name, transform, brightness_increase):
        super().__init__(root_dir, set_name, transform)
        self.brightness_increase = brightness_increase

    def __getitem__(self, index):
        image_name, target = self.data[index][0], self.data[index][1]
        img = io.imread(self.root_dir + image_name)

        if len(img.shape) == 2:
            img = np.repeat(img[:, :, np.newaxis], 3, axis=2)

                if self.brightness_increase > 0:
                        mask = (255 - img) < self.brightness_increase
                        img = np.where(mask, 255, img + self.brightness_increase)
                else:
                        mask = img < -self.brightness_increase
                        img = np.where(mask, 0, img + self.brightness_increase)
                        img = img.astype(np.uint8)
        img = Image.fromarray(img)

        if self.transform:
            img = self.transform(img)

        return img, np.array([target])

# HSV Hue noise increase
```

```
class HNI(CatDogDataset):
    def __init__(self, root_dir, set_name, transform, std):
        super().__init__(root_dir, set_name, transform)
        self.std = std

    def __getitem__(self, index):
        image_name, target = self.data[index][0], self.data[index][1]
        img = io.imread(self.root_dir + image_name)

        if len(img.shape) == 2:
            img = np.repeat(img[:, :, np.newaxis], 3, axis=2)

        img = matplotlib.colors.rgb_to_hsv(img/255)

        noise = np.random.normal(0, self.std, (224, 224))
        img[:, :, 0] = img[:, :, 0] + noise

        img[:, :, 0] = np.where(img[:, :, 0] > 1, img[:, :, 0] - 1,  img[:, :, 0])
        img[:, :, 0] = np.where(img[:, :, 0] < 0, img[:, :, 0] + 1,  img[:, :, 0])

        img = matplotlib.colors.hsv_to_rgb(img)
        img = np.rint(img * 255).astype(np.uint8)
        img = Image.fromarray(img)

        if self.transform:
            img = self.transform(img)

        return img, np.array([target])

# HSV Saturation noise increase
class SNI(CatDogDataset):
    def __init__(self, root_dir, set_name, transform, std):
        super().__init__(root_dir, set_name, transform)
        self.std = std

    def __getitem__(self, index):
        image_name, target = self.data[index][0], self.data[index][1]
        img = io.imread(self.root_dir + image_name)

        if len(img.shape) == 2:
            img = np.repeat(img[:, :, np.newaxis], 3, axis=2)

        img = matplotlib.colors.rgb_to_hsv(img/255)

        noise = np.random.normal(0, self.std, (224, 224))
        img[:, :, 1] = img[:, :, 1] + noise

        img = np.clip(img, 0.0, 1.0)
        img = matplotlib.colors.hsv_to_rgb(img)
        img = np.rint(img * 255).astype(np.uint8)
        img = Image.fromarray(img)

        if self.transform:
            img = self.transform(img)
        return img, np.array([target])

# Occlusion of the image increase
class OII(CatDogDataset):
    def __init__(self, root_dir, set_name, transform, edge_length):
        super().__init__(root_dir, set_name, transform)
        self.edge_length = edge_length

    def __getitem__(self, index):
        image_name, target = self.data[index][0], self.data[index][1]
        img = io.imread(self.root_dir + image_name)

        if len(img.shape) == 2:
            img = np.repeat(img[:, :, np.newaxis], 3, axis=2)

        # randomly choose the top-left pixel of the square region

        x = random.randint(0, 224-self.edge_length)
        y = random.randint(0, 224-self.edge_length)

        img[x:x+self.edge_length, y: y+self.edge_length, :] = 0
        img = Image.fromarray(img)

        if self.transform:
            img = self.transform(img)

        return img, np.array([target])
```