University of South Wales

Module Title: Computational Applications Of Artificial Intelligence
Module Code: CS4S773
Module Leader/Tutor: Dr. Mabrouka Abuhmida
Assignment 2 — Practical Coursework

Measuring the Impact of Image Augmentation Techniques

Student Name: Salma Javid
Student ID: 30107961

Table of Contents

Abstract

This report evaluates the impact of image augmentation techniques on the performance of the given Cats vs Dogs dataset. I will experiment with a minimum of seven image processing techniques, both separately and combined (mix and match) to optimize model performance.

As asked in the assignment, the Blackbox model will be trained for three different scenarios:

- Only with the original data
- Only with the augmented data
- On both original data and augmented data

Note: The evaluation of model performance is based on accuracy metrics from Blackbox.

In the last chapter, I will explain the rationale behind my chosen augmentation methods, and discuss their impact on the model performance. I will also discuss the results and try to present insights into the effectiveness of data augmentation in enhancing model performance.

1 — Introduction

The Objectives: To measure the impact and determine how data augmentation techniques can affect the performance of a baseline model.

Dataset Details:
Name: cats_vs_dogs (given with the assignment)

It has 2 folders:
1. Cat with 12,501 images
2. Dog with 12,501 images

Dataset Size: 786.68 MiB /root/tensorflow_datasets/cats_vs_dogs/4.0.0...
Note: An image batch size is set to 32, so each batch has 32 images. In total we have 727 batches of images.

Blackbox Model: Given with the assignment.
Observation: 1738 images in the dataset were found corrupted and so were skipped from all processes.

2 — Training the Blackbox with Original Data

Step — 1: Importing all required libraries, modules and loading the TensorFlow Dataset — "Cats vs. Dogs".

Step — 2: Dataset Visualization — Using the iterator function/tool to retrieve a set of 7 images and display them with their corresponding labels.

Result: 7 images are displayed — 5 dogs + 2 cats

Observation: All image sizes are different. For example, the size of the first dog image is (262, 350, 3) while the second is (409, 336, 3).

Blackbox — 1: Training Model with Original Data

Result

```
Epoch 1/7
727/727 [==============================] - 97s 116ms/step - loss: 0.6461 - accuracy: 0.6197
Epoch 2/7
727/727 [==============================] - 84s 115ms/step - loss: 0.5068 - accuracy: 0.7488
Epoch 3/7
727/727 [==============================] - 82s 112ms/step - loss: 0.4173 - accuracy: 0.8105
Epoch 4/7
727/727 [==============================] - 84s 115ms/step - loss: 0.3442 - accuracy: 0.8504
Epoch 5/7
727/727 [==============================] - 84s 115ms/step - loss: 0.2765 - accuracy: 0.8844
Epoch 6/7
727/727 [==============================] - 83s 114ms/step - loss: 0.2114 - accuracy: 0.9148
Epoch 7/7
727/727 [==============================] - 83s 114ms/step - loss: 0.1658 - accuracy: 0.9350
```

Please see attached file: 1_[Assignment_2_1]_Blackbox_with_Original_Data for reference.

3 — Image Preprocessing: Resizing and Rescaling

Preprocessing is an important step, a stage where we apply certain techniques and remove the inconsistencies found in an image dataset before it is used in a blackbox of a computer vision model. Image preprocessing may also help decrease model training time and increase model inference speed (*...explained in detail later*).

For image preprocessing I am using two techniques — Resizing and Rescaling

```
#1 – Resizing Images to a target size of 224x224 pixels using TensorFlow
    image = tf.image.resize(image, [224, 224], antialias=True)
    image = tf.cast(image, tf.float32) / 255.0

    return image, label
```

Blackbox — 2: Training Model After Resizing and Rescaling — Result

```
Epoch 1/7
727/727 [==============================] - 58s 62ms/step - loss: 0.6411 - accuracy: 0.6240
Epoch 2/7
727/727 [==============================] - 45s 62ms/step - loss: 0.4911 - accuracy: 0.7641
Epoch 3/7
727/727 [==============================] - 44s 61ms/step - loss: 0.4010 - accuracy: 0.8170
Epoch 4/7
727/727 [==============================] - 44s 60ms/step - loss: 0.3200 - accuracy: 0.8608
Epoch 5/7
727/727 [==============================] - 44s 60ms/step - loss: 0.2355 - accuracy: 0.9018
Epoch 6/7
727/727 [==============================] - 44s 61ms/step - loss: 0.1709 - accuracy: 0.9323
Epoch 7/7
727/727 [==============================] - 44s 61ms/step - loss: 0.1195 - accuracy: 0.9538
```

4 — Image Augmentation

In the world of artificial intelligence “Image Augmentation” belongs to the field of computer vision. The need and importance of augmentation in enhancing model performance is unquestionable.

WHAT is image augmentation? In computer vision tasks image augmentation is the process (or techniques) used to generate new transformed versions of existing images artificially. This not only increases the size and diversity of the training dataset but also makes it robust. Technically the augmentation techniques manipulate the given images and create copies with varied transformations, making the model more robust to the many variations it might encounter during real-world scenarios.

WHY is it required? This is done to expose our model to a wider array of examples. The necessity of IA Techniques is also due to image scarcity in certain cases which can limit the work of computer vision engineers. IATs resolve this issue by increasing a dataset’s diversity by providing more image models (copies of the original) that can help understand the data better and improve its performance in terms of object detection, classification and/or segmentation, and helps in formulating conclusive deductions and making informed decisions.

HOW is it done? By applying various transformation techniques using functions/methods to the existing images.

4.1: Salt & Pepper Noisy Image Augmentation

```
# Take one example from the dataset
example = dataset.take(1)

# Extract the image and label
for data in example:
    image_data, label = data['image'], data['label']

# Convert image to NumPy array
image_array = tf.image.convert_image_dtype(image_data, dtype=tf.uint8).numpy()

# Function to add Salt and Pepper noise to an image
def add_salt_and_pepper_noise(image, salt_prob, pepper_prob):
    noisy_image = np.copy(image)
    total_pixels = image.size

    # Add salt noise
    salt_pixels = int(total_pixels * salt_prob)
    salt_coordinates = [np.random.randint(0, i - 1, salt_pixels) for i in image.shape]
    noisy_image[salt_coordinates[0], salt_coordinates[1], :] = 255

    # Add pepper noise
    pepper_pixels = int(total_pixels * pepper_prob)
    pepper_coordinates = [np.random.randint(0, i - 1, pepper_pixels) for i in image.shape]
    noisy_image[pepper_coordinates[0], pepper_coordinates[1], :] = 0

    return noisy_image

# Add Salt and Pepper noise to the image
salt_and_pepper_noisy_image = add_salt_and_pepper_noise(image_array, salt_prob=0.02, pepper_prob=0.02)

# Display the original and noisy images
plt.subplot(1, 2, 1), plt.imshow(image_array)
plt.title("Original Image"), plt.axis("off")
plt.subplot(1, 2, 2), plt.imshow(salt_and_pepper_noisy_image)
plt.title("Salt and Pepper Noisy Image"), plt.axis("off")
plt.show()
```

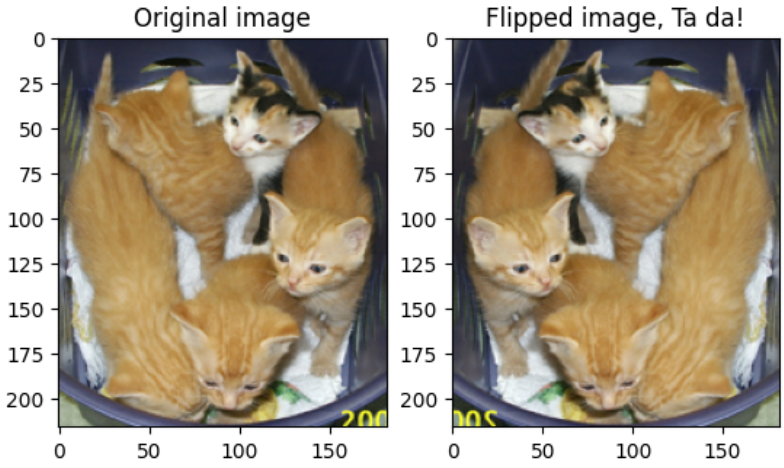
Blackbox — 3: Train the Model on Salt & Pepper Noisy Image

```
Epoch 1/7
1/1 [==============================] - 3s 3s/step - loss: 0.6583 - accuracy: 1.0000
Epoch 2/7
1/1 [==============================] - 0s 18ms/step - loss: 2.8165e-04 - accuracy: 1.0000
Epoch 3/7
1/1 [==============================] - 0s 18ms/step - loss: 9.0933e-10 - accuracy: 1.0000
Epoch 4/7
1/1 [==============================] - 0s 15ms/step - loss: 1.5053e-16 - accuracy: 1.0000
Epoch 5/7
1/1 [==============================] - 0s 15ms/step - loss: 2.4638e-24 - accuracy: 1.0000
Epoch 6/7
1/1 [==============================] - 0s 19ms/step - loss: 8.1939e-33 - accuracy: 1.0000
Epoch 7/7
1/1 [==============================] - 0s 15ms/step - loss: 0.0000e+00 - accuracy: 1.0000
<keras.src.callbacks.History at 0x7977941235e0>
```

4.2: Position Manipulation or Geometric transformation

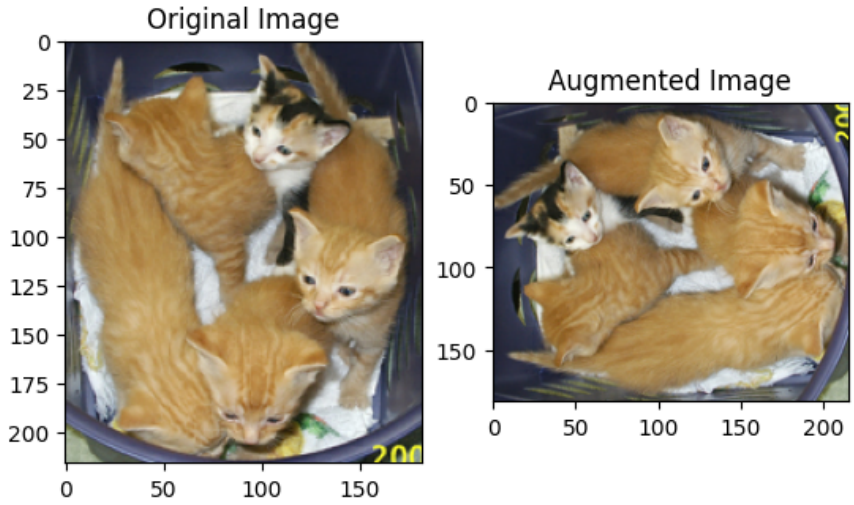
— Flipping Image

```
flipped = tf.image.flip_left_right(image)
visualize(image, flipped)
```



— Rotate an Image

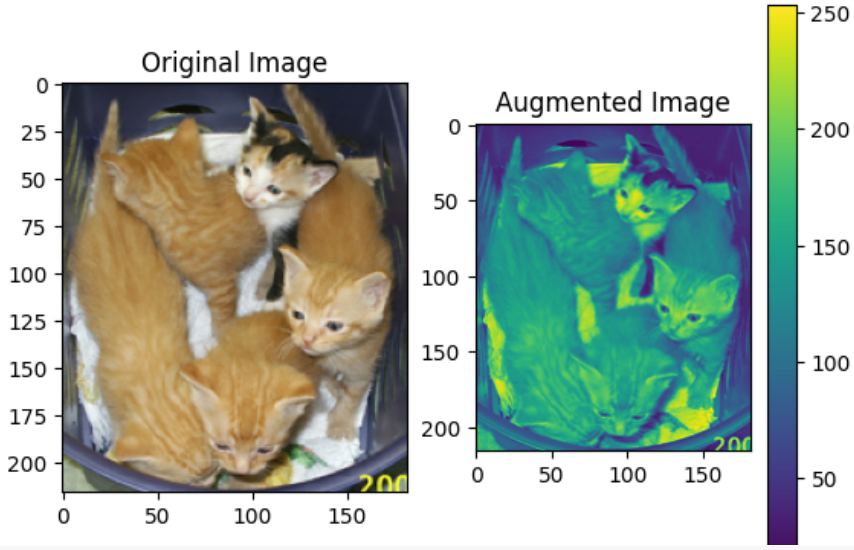
```
rotated =
tf.image.rot90(image)
visualize(image, rotated)
```



4.3: Color Changes

— Image Gray scaling

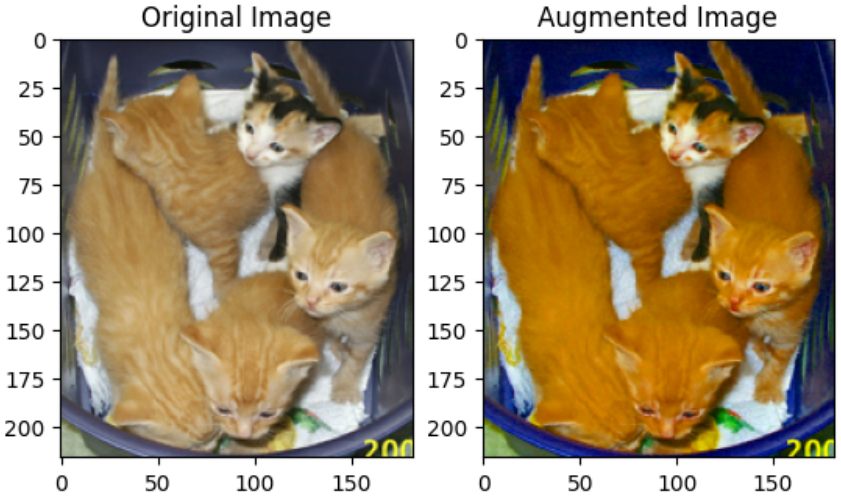
```
grayscaled =
```




```
tf.image.rgb_to_grayscale(image)
visualize(image, tf.squeeze(grayscaled))
_ = plt.colorbar()
```

— Image Saturation

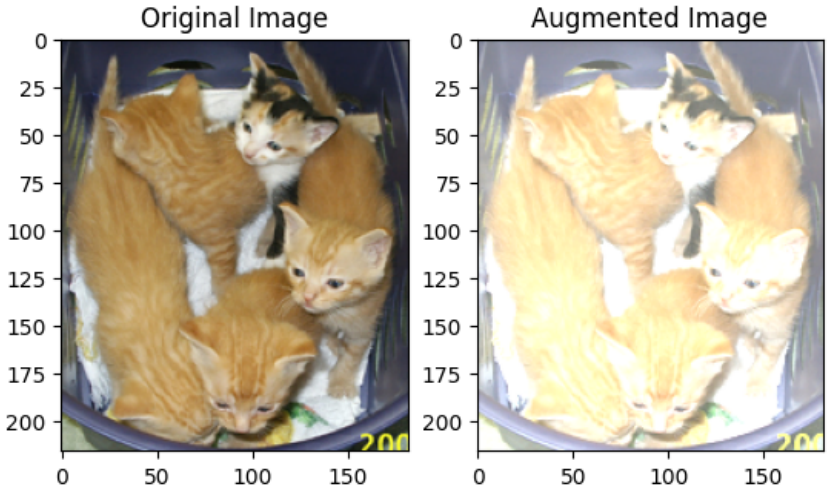
```
saturated =
tf.image.adjust_saturation(image,
visualize(image, saturated)
```




```
3)
```

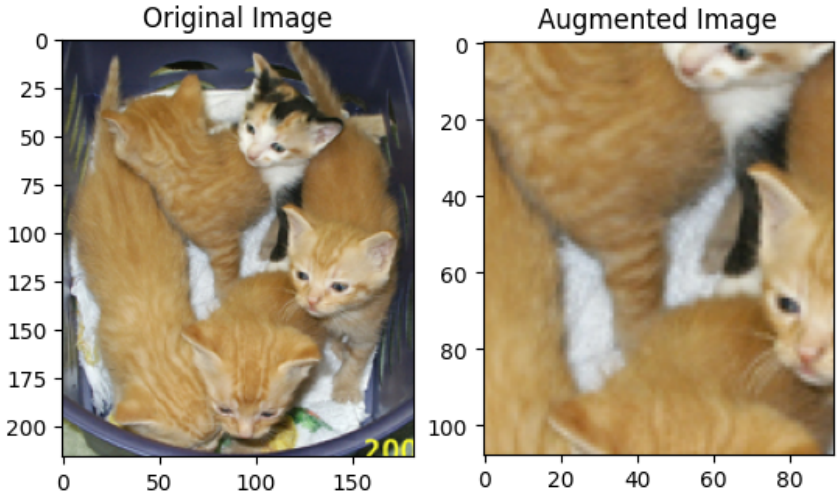
— Image Brightness Setting / Adjustment

```
bright = tf.image.adjust_brightness(image, 0.4)
visualize(image, bright)
```



4.4: Image Cropping

```
cropped = tf.image.central_crop(image, central_fraction=0.5)
visualize(image, cropped)
```

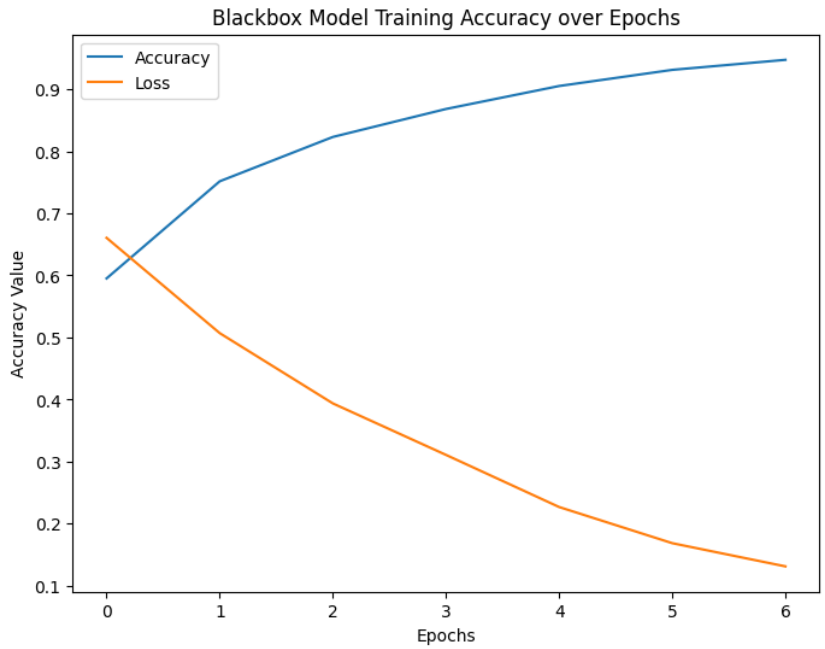


Blackbox — 4: Training Model After Multiple Augmentations

Now we're training the blackbox with the augmented [Resizing + Rescaling + Flipping + Gray scaling + Saturation] dataset — Result:

```
Epoch 1/7
727/727 [==============================] - 61s 65ms/step - loss: 0.6417 - accuracy: 0.6177
Epoch 2/7
727/727 [==============================] - 48s 67ms/step - loss: 0.4918 - accuracy: 0.7633
Epoch 3/7
727/727 [==============================] - 47s 64ms/step - loss: 0.3949 - accuracy: 0.8204
Epoch 4/7
727/727 [==============================] - 47s 64ms/step - loss: 0.3224 - accuracy: 0.8594
Epoch 5/7
727/727 [==============================] - 47s 65ms/step - loss: 0.2435 - accuracy: 0.8992
Epoch 6/7
727/727 [==============================] - 47s 64ms/step - loss: 0.1800 - accuracy: 0.9261
Epoch 7/7
727/727 [==============================] - 48s 66ms/step - loss: 0.1517 - accuracy: 0.9402
```

5. Line Plot Representing Blackbox Accuracy Score



6. Mix & Match — Image Augmentations

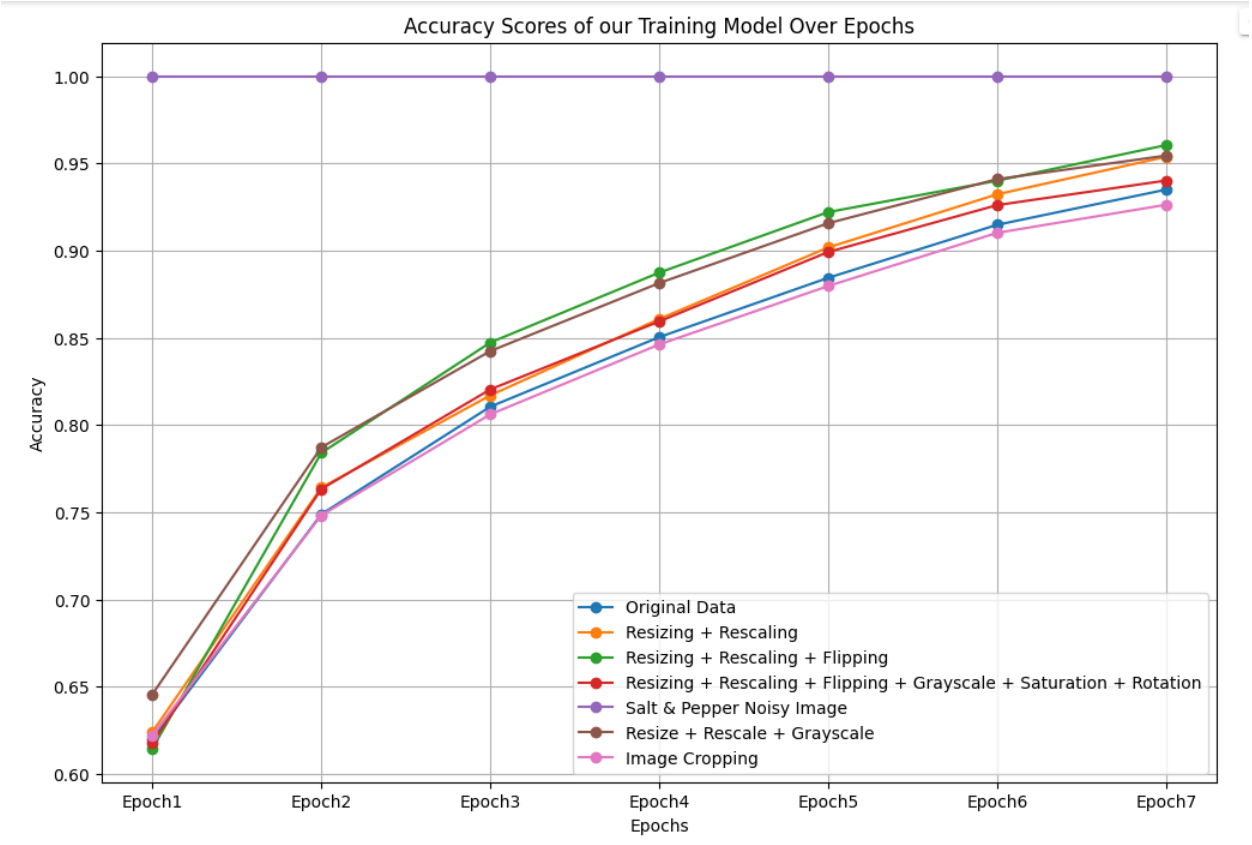
Please have a look below at the table to compare the blackbox score after 7 Mix & Match scenarios:

Accuracy Scores of Blackbox	Epoch1	Epoch2	Epoch3	Epoch4	Epoch5	Epoch6	Epoch7
Original Data	0.6197	0.7488	0.8105	0.8504	0.8844	0.9148	0.9350
Resizing + Rescaling	0.6240	0.7641	0.8170	0.8608	0.9018	0.9323	0.9538
Resizing + Rescaling + Flipping	0.6145	0.7840	0.8473	0.8874	0.9221	0.9401	0.9605

Resizing + Rescaling + Flipping + Grayscale + Saturation + Rotation	0.6177	0.7633	0.8204	0.8594	0.8992	0.9261	0.9402
Salt & Pepper Noisy Image	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000
Resize + Rescale + Grayscale	0.6456	0.7872	0.8424	0.8814	0.9157	0.9410	0.9545
Image Cropping	0.6221	0.7482	0.8061	0.8462	0.8797	0.9102	0.9263

Please refer to my jupyter files for all details. If I include all the scripts, the report length will increase further.

Graph — Accuracy Scores of our Training Model Over Epochs



Mix and Match: I have mixed and matched various augmentation techniques. The goal is to explore various data augmentation and preprocessing techniques to train my model on them and optimize its performance. Combining various data augmentation techniques, diverse strategies and preprocessing methods enhance the overall performance of a machine learning model.

7. Discussion, Evaluation & Conclusion

Discussion

I was given a large database of pet images as a dataset, of which 12,501 are of dogs and the same number is of cats.

The image augmentation techniques I have used to train my model are basically based on testing the Cats and Dogs images by cropping them, sprinkling them with Salt and Pepper augmentation technique, by changing their saturation levels and color — gray scaling it, by flipping and rotating them, etc. **These methods are suitable to this dataset.**

Let's see WHY? Certain animal parts are very distinctive (they stand out) like the shape of ears, eyes, shape of snout and tail in a dog… and cropping them out or rotating them will challenge the model to use its training and predict accuracy accordingly. Salt and Pepper is used to trick the model to identify the animal behind the sprinkle.
Similarly color changing (grayscaling & saturation) are used to train the model in a simple way. Grayscale usually has a higher score of accuracy as it removes out all color and makes it easy for the model to differentiate between the object and the background.

Although height could alter the Blackbox accuracy score (but it did not!) because in the dataset we do have dogs sitting down and standing up as well... so the model was well trained to identify different sitting, standing and positions of the dog too. For cats the distinct feature could be the round face with ear sticking out, the whiskers and of course the tail.

Common features in both the animals include 4 legs and a tail... but height difference — A cat is short and a dog is comparatively tall — **could be a deciding factor**. It might have depended upon the facial features for identification and accuracy.

Confusing cases: Like in the image above, in certain cases a cat may look or pose like a dog or vice versa. This is where the accuracy of the model is really tested.

My Choice and Rationale

Let me explain why I chose the above image augmentation techniques and how they impact the dataset:

— Resizing and Rescaling

Earlier we had observed how all image sizes were different. To present an example: The size of the first dog image was (262, 350, 3) while the second was (409, 336, 3) — clearly, they're not of the same size. If the images are not the same in size, the blackbox model and accuracy score may or may not perform as expected. To resolve this issue normalization is done.

1. The RESIZE function resizes all input images to the specified fixed dimension of 224, 224. This ensures consistency in input sizes.
2. RESCALING changes the pixel values of all images to tf.float32 data type and then scales them by dividing them by 255.0. This step normalizes the pixel values to the range [0, 1] which helps in training models faster and improves the training stability of models to be trained and tested.

— Salt and Pepper Noise Image Augmentation Technique

Salt and Pepper is a type of image augmentation technique that introduces noise — technically speaking disturbance in an image — which gives the effect of Salt (bright pixels) and Pepper (dark pixels) sprinkled on an image. Simply put, the technique produces a random pixel-level noise to an image.

I have intentionally chosen this to challenge the model, to train it to identify the image in spite of the augmentation used. This augmented image helps train a model to improve its performance under noisy conditions. It is a useful tool to augment images for training models because when noise is added to image data it manipulates the appearance of it. **It could create a bias as well**. Introducing Noise allows room for algorithm performance to be tested and improved. This technique can improve the predictive accuracy and general performance of our model.

— Flipping Technique

This is one of the most easy and common image augmentation techniques, used to flip a given image (or entire dataset), horizontally or vertically and then create copies of this flipped version. This method artificially increases the diversity of a training dataset by creating mirror images of existing examples. The trick is to flip the images, and run them through models to generalize better to variations in object orientations.

Models trained with flip augmentation are often more capable of handling variations in object orientation, contributing to better generalization performance on unseen data. Also, by exposing the model to different versions of the same object, flip augmentation can help prevent overfitting to specific orientations present in the original dataset. The purpose is to train our model to be invariant to these transformations. Additionally, it expands the dataset and introduces variability.

— Rotate

This technique rotates an image by a certain specified or a random angle. I have chosen this so that I can rotate the dataset of images of cats and dogs, change their orientation and feed it to the blackbox for checking accuracy levels. Once our model is well-trained with it, it helps improve our model's accuracy metric for all kinds of rotated images. This enables the model to identify the image in different orientations, and angles.

— Image Gray Scaling

Gray scale refers to converting color images to grayscale. Grayscale compressors an image to its barest minimum pixel and enhances easy visualization. Doing this reduces computational complexity and can simplify models. This helps in simplifying algorithms and as well eliminates the complexities related to computational requirements. It makes room for easier learning for those who are new to image processing. Grayscale images only store values in a single array (black and white).

— Image Saturation

If you wish to play with colors while doing image augmentation then Saturation is for you. Okay! Coming back to business, for our dataset adjusting the saturation of an image can help our model perform better. This situation might arise when you have images captured in low light or at night or in a fog, etc. Saturation technically tackles the amount of brightness a color appears in an image. The opposite of saturation is a grayscale or black-and-white photo, while a full-color image might be highly saturated.

— Image Brightness Setting / Adjustment

This refers to adjusting the brightness of images — changing the brightness of the image to varying lighting conditions. The purpose is to enhance model robustness.

— Image Cropping

This is done to **confuse** the model. Cropping cuts out parts of an image. For example, the identifying feature of a cat is its short height, short legs and ears sticking out of a round face. How if we crop any of those deciding factors? This image manipulation method introduces variations in the composition of the image which helps to train our model well by highlighting certain sections of the image clearly. Cropping can be random or specific. It can be used to focus on the region of interest or wherever analysis is required. This image manipulation method introduces variations in the composition of the image which helps to train our model well by highlighting certain sections of the image clearly.

Evaluation

Blackbox — 1: Training Model with Original Data

The accuracy after the 1st epoch was 0.6197 which rose to 0.9350 by the 7th epoch.

Blackbox — 2: Training Model After Resizing and Rescaling

The accuracy after the 1st epoch was 0.6240 which rose to 0.9538 by the 7th epoch.

Blackbox — 3: Train the Model on Salt & Pepper Noisy Image

The accuracy consistently recorded to be 1.0000 from 1st to 7th epoch!

Blackbox — 4: Training Model After Multiple Augmentations

Here the accuracy rose from 0.6177 from 1st epoch to 9402 in the 7th

If you see the **Mix & Match — Image Augmentations** results, the combination of "Resizing + Rescaling + Flipping" achieved the highest accuracy of 0.9605 while Image Cropping recorded the lowest score of 0.9263.

Conclusion

My findings show that the accuracy of almost every model was about 60% for the first epoch (except the Salt & Pepper model which scored a 100% for all epochs, beginning from the first!). 60% indicates that the model is not performing well and should be further optimized by running more epochs. More epochs ensure better training and data preprocessing which help improve the accuracy.

I found **Epoch 7** to be the optimal measure of epochs for all my training models as it fetches the accuracy score above 90%. This metric can be used to measure the performance of a model in real-world scenarios.

I found the accuracy metric to be very low in almost all of the first epochs. This indicates that the accuracy metric level is not reliable or effective for first epochs and therefore should not be used to make decisions. It is important to optimally analyze the accuracy of a model to ensure that it is performing well, and has become robust and reliable.

Epoch 7 appears to be optimal, more reliable and trustworthy in the sense that blackbox accuracy goes beyond 90% with it. This metric is important for the credibility of the results.

References

Data augmentation | TensorFlow Core ([no date]). Available at: https://www.tensorflow.org/tutorials/images/data_augmentation (Accessed: 27 November 2023).

A Complete Guide to Data Augmentation ([no date]). Available at: https://www.datacamp.com/tutorial/complete-guide-data-augmentation (Accessed: 28 November 2023).

JAN 26, J.N. and Read, 2020 8 Min (2020) *Why should I do pre-processing and augmentation on my computer vision datasets?* Available at: https://blog.roboflow.com/why-preprocess-augment/ (Accessed: 28 November 2023).

Pai, P. (2020) 'Data Augmentation Techniques in CNN using Tensorflow'. *YML Innovation Lab* 3 November. Available at: https://medium.com/ymedialabs-innovation/data-augmentation-techniques-in-cnn-using-tensorflow-371ae43d5be9 (Accessed: 28 November 2023).

Le, J. (2021) 'How to easily build a Dog breed Image classification model'. *NanoNets* 10 August. Available at: https://medium.com/nanonets/how-to-easily-build-a-dog-breed-image-classification-model-2fd214419cde (Accessed: 28 November 2023).