CS838 Homework Assignment 1

Akshata Bhat (akshatabhat@cs.wisc.edu)

1 October 2019

1 Introduction

Training deep learning neural network models on more data may result in improved performance. Data augmentation techniques are used to artificially create new training example from existing data. The idea is to generate various versions of an image, where all these variations belong to the same class. This process can enable the models to generalise better. These transformations/variations are chosen such that they are plausible in real-life images.

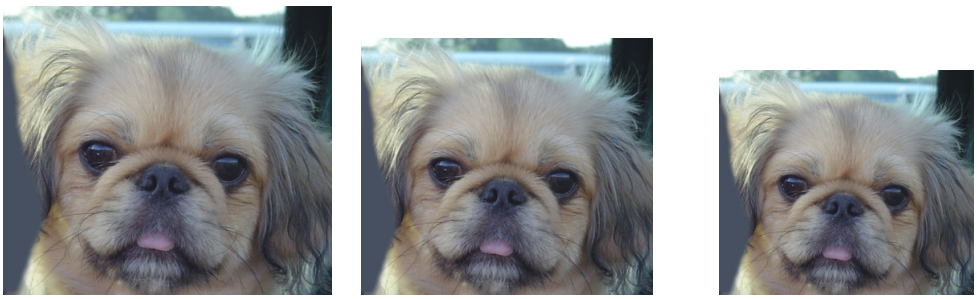
Goal of this assignment is to :

- Implement basic image processing functions.
- Assemble these functions into a data augmentation pipeline.

2 Image Processing

2.1 Image Resizing

The numpy array is resized to the required pre-specified size using adaptive resizing. If the pre-specified size is a sequence like (w, h), output size is matched to this. If size is an int, smaller edge of the image is matched to this number. Hence, if height > width, then image will be rescaled to (size, size * height / width) and vice-versa.



(a) (b) (c)

Figure 1: Image Resizing a) Original Image b) Image resized to size = 320 c) Image resized to size = 280

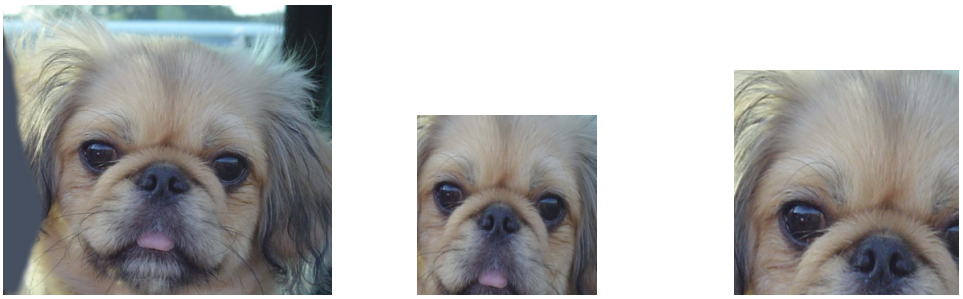
2.2 Image Cropping

Given a numpy array, a crop of image is made by selecting a region randomly. The size of the region is drawn from a uniform distribution of a given range of image areas and the aspect ratio is drawn from a pre-specified interval. (x,y) start pixels are also drawn randomly from the image. Given the target area and the aspect ratio, the new dimensions of the image are calculated as follows. (Note there are two possibilities):-

$$height_1 = \sqrt{\frac{target\ area}{aspect\ ratio}}\ width_1 = \frac{target\ area}{height_1}$$

$$width_2 = \sqrt{\frac{target\ area}{aspect\ ratio}}\ height_2 = \frac{target\ area}{width_2}$$

Given the randomness in the start pixel, and the crop size, there is a possibility that the region doesn't entirely lie within the image. Hence, this operation is tried for a number of trials, after which a fallback mechanism is used where the input image itself is the cropped image. After the image is cropped, it is resized. If pre-specified size is a sequence like (w, h), output size is matched to this. If size is an int, output size is (size, size).



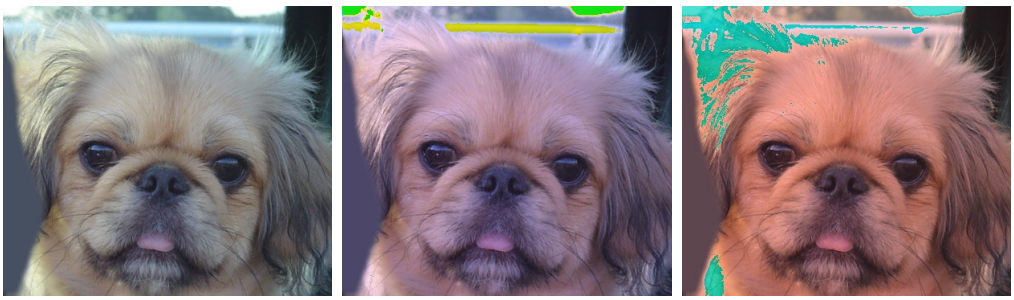
(a) (b) (c)

Figure 2: Image Cropping. a) Original Image b) Image resized to size = 244 c) Image resized to size = 280

Design Choice : Since the image is converted to numpy array, the image is cropped efficiently using indexing(start and end pixels for x and y axis).

2.3 Color Jitters

A small perturbation in the input image can lead to images with drastically different pixels, which the machine learning model could perceive very differently, but are perceptually realistic to a human eye. To make the model robust, for a given image, the color channels are perturbed. α is uniformly sampled from U(-r, r) with $r \in (0, 1)$ and $1 + \alpha$ is multiplied to a color channel. This procedure is done independently for each channel.



(a) (b) (c)

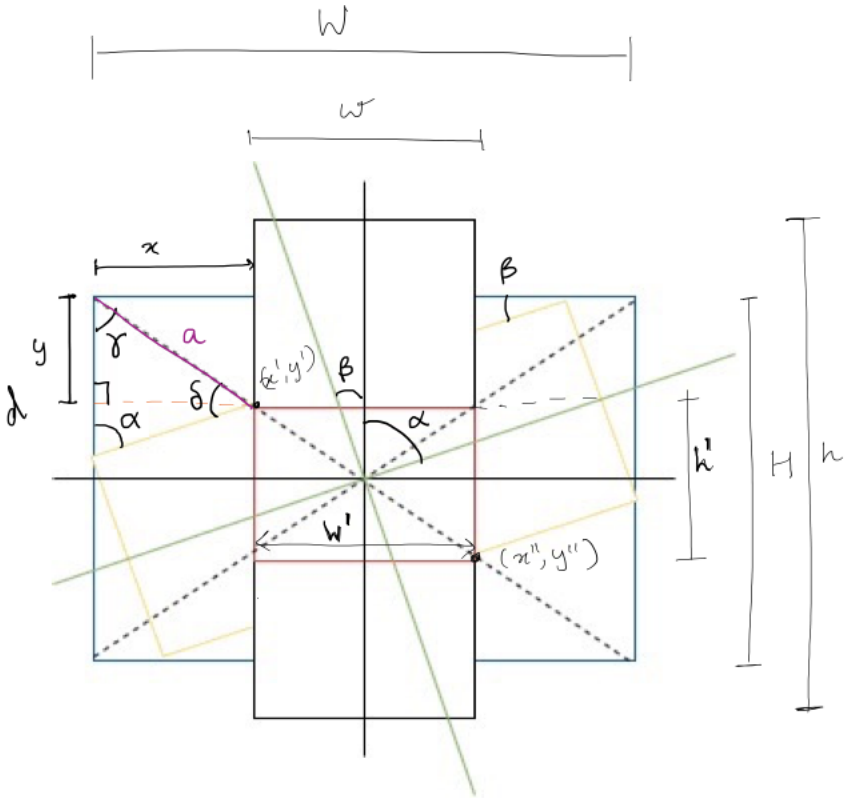
Figure 3: Color Jitter. a) Original Image b) Perturbed image with $\alpha = 0.15$ c) Perturbed image with $\alpha = 0.4$

Design Choice : Given n pixels, multiplying $1 + \alpha$ with each pixel is an inefficient solution with a complexity of O(n). However, instead of using a for-loop, numpy nd-array can directly be multiplied with a scalar value. This approach is used in the implementation and it speeds up the operation drastically. Since the operation is performed multiple times on multiple images, the overall performance boost is huge.

2.4 Rotation

The given numpy array is rotated by a random degree. The degree value is uniformly randomly sampled from a pre-specified range. *rotate* function is used from the *PIL* library to perform the rotation. However,

this function creates empty black pixels. To filter these pixels, the largest rectangular region is cropped from the rotated image. The key idea is that the largest rectangle is proportional to the outer resulting rectangle, and its two opposing corners lie at the intersection of the diagonals of the outer rectangle with the longest side of the rotated rectangle. After computing the width and height of the largest rectangle, the image (containing black pixels) is cropped around the center using these dimensions as given in.



Compute quadrant : {0, 1, 2, 3}

α' = (quadrant & 1) == 0 ? angle : 180° − angle

α = (α' % 180° + 180°) % 180°

$\gamma = \tan^{-1}\left(\frac{W}{H}\right)$ if $w < H$, else $\gamma = \tan^{-1}\left(\frac{H}{W}\right)$

$\delta = 90 - \alpha + 90 - \alpha = 180 - \alpha - \gamma$,

$d = h\cos\alpha$ if $w < h$, else $d = w\cos\alpha$

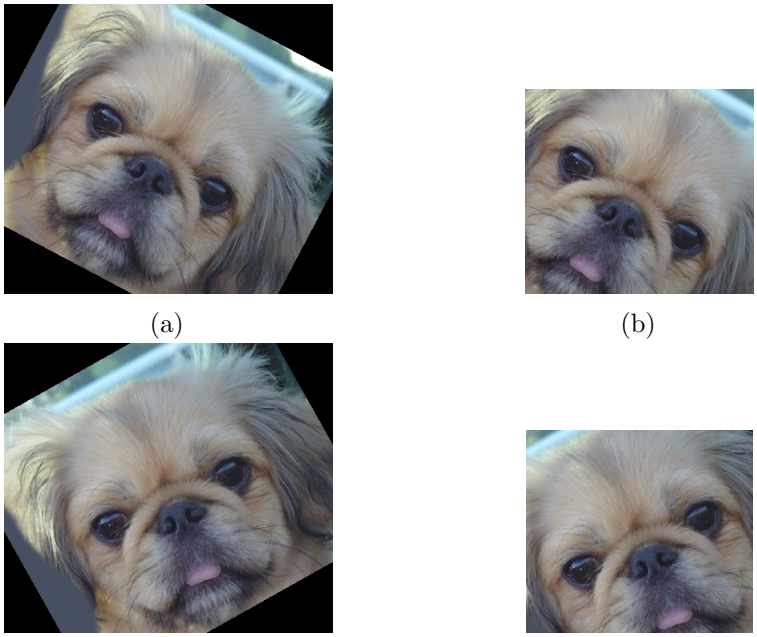
$\frac{d}{\sin\delta} = \frac{a}{\sin\alpha} \Rightarrow \boxed{a = \frac{d\sin\alpha}{\sin\delta}}$

$y = a\cos\gamma$, $x = y\tan\gamma$

cropped width, $w' = W - 2 * x$

cropped height, $h' = H - 2 * y$

Figure 4: Finding the largest rectangle in a rotate image.



(a) (b)

(c) (d)

Figure 5: Image Rotation. a) Rotated image (with rotate range = 30) with empty black pixels and b) is its corresponding image without empty black pixels. c) Rotated image (with rotate range = 50) with empty black pixels and d) is its corresponding image without empty black pixels.

2.5 Horizontal Flip

The given image is randomly(with a probability of 0.5) flipped horizontally using the function *flip* from *opencv*

(a) (b)

Figure 6: Horizontal Flip. a) Original Image b) Horizontally flipped image

3 Transformed Images

For each image, transformations are applied in this order - resizing, random horizontal flip, color jitters, random rotation and random cropping.

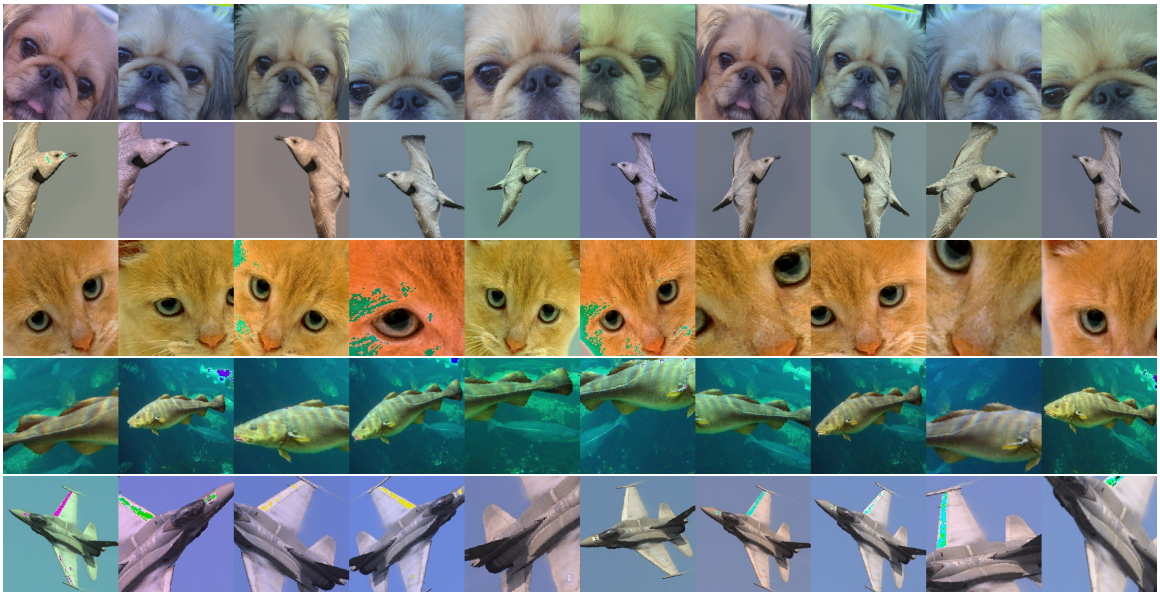


Figure 7: Transformations applied to 5 images

4 Data Augmentation and Input Pipeline

4.1 Composition of Image Transformations

A series of transformations are composed and applied to the input images. The transformations applied in this assignment are affine transformations. An affine transformation preserves the following properties:-

- Collinearity between points - three or more points that lie on the same line continue to be co-linear even after the transformation.
- Parallelism - two or more lines that are parallel continue to be parallel after the transformation.

Hence the ordering of these transformations dosen't matter.

4.2 Data Input

The training of models require loading multiple image files at the same time. Following factors could also have the speed :-

- Certain image formats could be better. (.bmp could be better than .jpg)
- Resizing the image before performing other transformations may boost the performance in cases where the raw images are too huge. Applying other transformations on the resized image could speed up the process as the numpy array sized decrease.
- Pre-fetching data can enable faster loading of data.
- Converting numpy arrays to tensors (pytorch or tensorflow tensors) is also necessary.
- It is necessary to shuffle the data, to have variety of examples in each batch.
- The batch size should be selected carefully, depending on the available memory.

5 Conclusion

Simple transformations can be used to perform data augmentation, which can aid in improving the performance of a deep learning model.