Augmentation

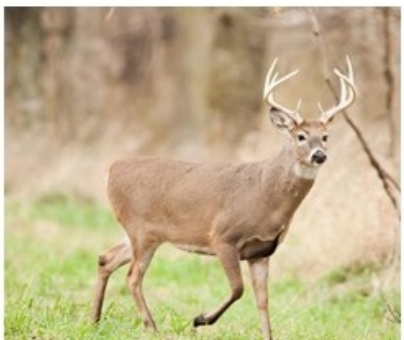
Discover how data augmentation works and its benefits for training a CNN.

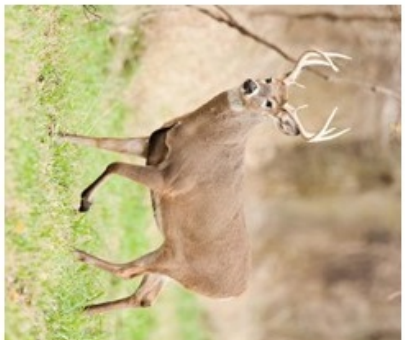
Chapter Goals:

- Learn about data augmentation and its purpose
- Apply data augmentation to image data
- Standardize the image data

A. Data augmentation

Data augmentation is the process of artificially enlarging a dataset through *image transformations*. No matter how good a model is, it cannot perform at its fullest potential unless we have a large enough dataset. Although CIFAR-10 contains 50,000 training images, it would benefit our model if it were trained on more data. Rather than going through the process of obtaining new images and resizing them, we can instead perform transformations on the images (e.g. rotating or cropping) to manufacture "new" image data.





The transformations used are (from left to right): crop, horizontal flip, and rotation.

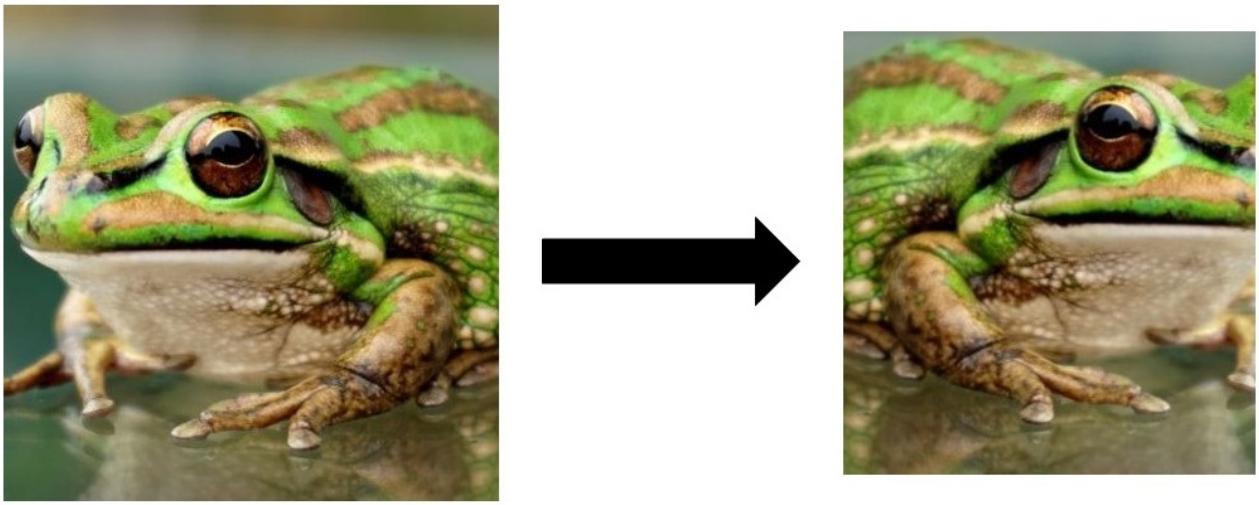
Using data augmentation, we create a variation of each image in a batch of input data. Since we train our model on hundreds of epochs (full run-throughs) of the CIFAR-10 dataset, data augmentation constructs hundreds of variations for each image in the training set.

B. Image transformation

In our code, the image transformation we use is a random crop followed (potentially) by a horizontal flip. We don't use rotations because it is unrealistic to see most of the CIFAR-10 images rotated in real life.

We first apply a random crop of the image to a height and width of `resize_dim`. This means that we crop the original image data to a randomly chosen submatrix of the data with height and width equal to `resize_dim`. For the CIFAR-10 dataset, we set `resize_dim = 24`.

Then, with a probability of 0.5, we will flip the cropped image horizontally. This results in approximately half of our augmented data being flipped and the other half not flipped, which is the optimal distribution for data variety.



Random crop and flip applied to a frog input image.

Since we only apply data augmentation when training our model, we don't use any image transformations when the model is not training. However, because we apply a random crop while training, the input to our model layers must

now have height and width equal to `resize_dim`. Therefore, we need to resize the image data even during model evaluation. For consistency between

training and evaluation, we choose to use a central crop (submatrix centered in the middle) of the original image to resize the data.

C. Data standardization

The CIFAR-10 image data is still pixel integers between 0-255, so we need to convert the data to a more suitable range of values. Rather than normalize to a range of 0.0 to 1.0 (like the MNIST data), we do per image *standardization*. What this means is that we linearly scale the pixel data of an image so that the new data has zero mean and unit variance. To accomplish this, we use the function `tf.image.per_image_standardization`, which takes in image data and returns the standardized data.

Time to Code!

In this chapter you'll be completing the `image_preprocessing` function, which preprocesses image data to augment the dataset.

Specifically, you'll be creating the helper function `random_crop_and_flip`, for cropping and flipping an image during training.

We'll first apply a random crop and random horizontal flip to the image data. We use `tf.random_crop` for the random crop, with a new cropped shape of `[self.resize_dim, self.resize_dim, 3]`.

Set `crop_image` equal to `tf.random_crop` with first argument `float_image` and second argument the new cropped shape.

We then pass the cropped image as the required argument for `tf.image.random_flip_left_right`, which performs the random flip.

Set `updated_image` equal to `tf.image.random_flip_left_right` applied with required argument `crop_image`.
Then return `updated_image`.

```
import tensorflow as tf

class SqueezeNetModel(object):
    # Model Initialization
    def __init__(self, original_dim, resize_dim, output_size):
        self.original_dim = original_dim
```

```
        self.resize_dim = resize_dim
        self.output_size = output_size

    # Random crop and flip
    def random_crop_and_flip(self, float_image):
        # CODE HERE
        pass

    # Data Augmentation
    def image_preprocessing(self, data, is_training):
        reshaped_image = tf.reshape(data, [3, self.original_dim, self.original_dim])
        transposed_image = tf.transpose(reshaped_image, [1, 2, 0])
        float_image = tf.cast(transposed_image, tf.float32)
        if is_training:
            updated_image = self.random_crop_and_flip(float_image)
        else:
            updated_image = tf.image.resize_image_with_crop_or_pad(float_image, self.resize_d
        standardized_image = tf.image.per_image_standardization(updated_image)
        return standardized_image
```