Augmentation Techniques

Introduction

In machine learning and, particularly, in deep learning, data augmentation refers to the process of synthetically increasing the diversity and quantity of training data by applying label-preserving transformations to existing instances. This technique is especially useful in domains where models require large volumes of data to generalize well, such as computer vision, natural language processing, or audio processing. The main idea is that, while the number of available labeled samples may be limited, one can generate new, varied instances by applying controlled distortions or transformations that do not alter the semantic meaning of the data.

Augmentation can be used to counteract over fitting, by exposing the model to a wider range of inputs, thereby reducing its tendency to adapt to the existing training data too well. Likewise, augmentation techniques can improve generalization by simulating variations an perturbations that the model may encounter during deployment. In this sense, augmentation injects prior knowledge into the learning process, i.e. it encodes assumptions about what variations should not change the output (i.e. flipping an image of an object still contains that object).

Augmentation of Image Data

In image-based machine learning tasks, data augmentation plays an essential role in enhancing model performance and generalization. The primary objective of image augmentation is to artificially-expand the training dada set by generating new, realistic images from existing ones through a variety of label-preserving transformations. These transformations introduce variability into the data, enabling models to learn representations that are more invariant to real-world conditions, such as lighting changes, object orientation, occlusion, or background noise.

Image augmentation addresses several fundamental challenges in computer vision. Deep neural network require a large amount of data to converge and are highly-prone to overfitting, especially when trained on small data sets. Similarly to most other scenarios, augmentation techniques attempt to reduce overfitting by increasing the diversity of the training data set without requiring additional manual labeling. Also, real-world image data exhibits significant variability. Augmentation acts as a form of domain randomization, making models less

sensitive to irrelevant image-specific features and more focused on semantically-meaningful patterns.

There are two broad categories of image augmentations: photometric and geometric:

- **Photometric augmentations** alter the pixel values without changing the spatial layout of the image. Examples include brightness and contrast adjustments, color jitter, histogram equalization and the addition of Gaussian noise. These transformations simulate changes in illumination, sensor noise, or color balance.
- **Geometric augmentations** modify the spatial structure of the image. These include operations such as horizontal or vertical flipping, rotation, scaling, translation, cropping, and perspective warping. Such transformations emulate the effect of viewing the object from different angles or distances.

More sophisticated augmentation techniques attempt to simulate more complex variations. Methods like Cutout**,** MixUp, and CutMix introduce partial occlusions or combine regions of different images to encourage robustness to object localization and background context. Adversarial augmentation introduces perturbations designed to challenge the model, helping it become more resilient to edge-case scenarios. Generative approaches, such as using Generative Adversarial Networks (GANs), create entirely new images that retain the semantic content of the original class, offering augmentation that reflects underlying data distributions more faithfully.

Augmentation can be applied either offline (pre-processing and storing new images) or online (on-the-fly during training). The latter is more efficient and scalable, especially in deep learning pipelines, as it introduces stochasticity into every epoch, effectively generating a unique dataset each time the model iterates over the data. Furthermore, the choice of augmentation techniques is generally task-sensitive. For instance, horizontal flipping is beneficial for object recognition tasks but not suitable for character recognition, where orientation is important. Similarly, aggressive color distortions may hinder performance in medical imaging, where color consistency influences the diagnostic.

Traditional Augmentation

Traditional augmentation techniques are manually defined, deterministic or stochastic operations that are straightforward to implement and computationally inexpensive. These

methods became important in computer vision pipelines well before generative models or automated policy learning systems like AutoAugment.

Traditional augmentations aim to simulate realistic variations that the model may encounter during inference, such as changes in orientation, scale, lighting, or occlusion. They increase dataset diversity, reduce overfitting, and improve generalization without requiring additional labeled data. These techniques are typically applied online, during data loading, which means a single image can result in a multiple augmented versions over multiple training epochs.

Geometric Transformations

These modify the spatial layout of the image while preserving its semantic content (Fig. 1).

- Horizontal and Vertical Flipping
    - Most common for symmetric objects.
    - Horizontal flipping is safe and useful in many contexts (object classification), while vertical flipping is task-dependent (not suitable for digits or faces).
- Rotation
    - Random rotations (e.g., ±15°) help models become invariant to small pose changes.
    - Larger rotations may not be appropriate for all tasks (character recognition).
- Scaling and Zooming
    - Alters the size of the object within the image.
    - Can simulate distance from the camera or size variation.
- Translation (Shifting)
    - Moves the image content horizontally or vertically.
    - Useful for invariance to object location.
- Shearing
    - Distorts the image by slanting its axes.
    - Introduces perspective-like variation.
- Cropping
    - Random crops simulate partial views of an object.
    - Center crops are often used in evaluation to reduce variability.
- Padding and Resizing
    - Padding helps preserve information when cropping or shifting.
    - Resizing standardizes input size across the dataset.

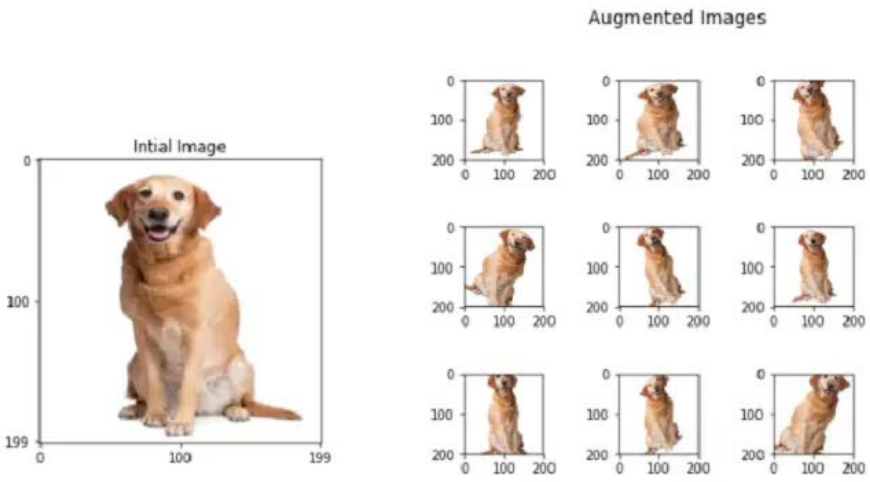

Fig. 1. Examples of image augmentation using geometric transformation (rotation, scaling, cropping)

Photometric Transformations

These adjust pixel-level color and intensity properties, simulating lighting or sensor noise (Fig. 2). Examples include, but are not limited to:

- Brightness Adjustment: Adds or subtracts a constant value to pixel intensities.
- Contrast Adjustment: Stretches or compresses the intensity distribution relative to the mean.
- Saturation Adjustment: Affects the intensity of colors; useful when working with natural scenes.
- Hue Shifting: Rotates the colors to simulate color temperature or sensor changes.
- Color Jittering: Combines multiple adjustments randomly to simulate real-world lighting variability.
- Gaussian Noise Injection: Adds zero-mean Gaussian noise to simulate sensor or transmission noise.
- Blurring and Sharpening: Applies convolutional filters (ex. Gaussian blur) to emulate out-of-focus conditions.

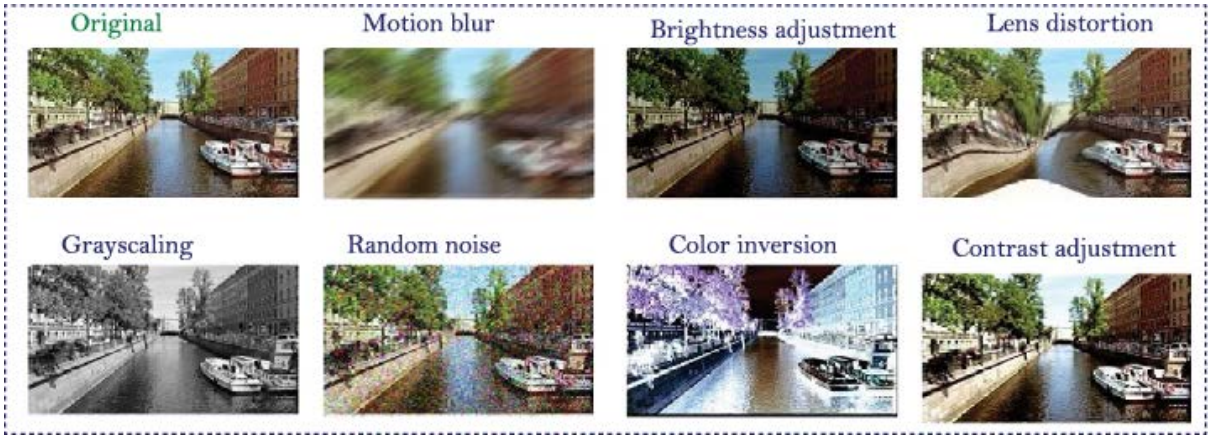

Fig. 2. Examples of image augmentation using photometric transformations, that change pixel colors and intensities.

Occlusion-Based Transformations

These techniques hide parts of the image, forcing the model to rely on global features (Fig. 3).

- Random Erasing
  - Randomly masks a rectangular region in the image.
  - Simulates occlusion or object truncation.
- Cutout
  - A fixed-size square region is erased (set to zero or a constant).
  - Helps the model become robust to partial visibility.

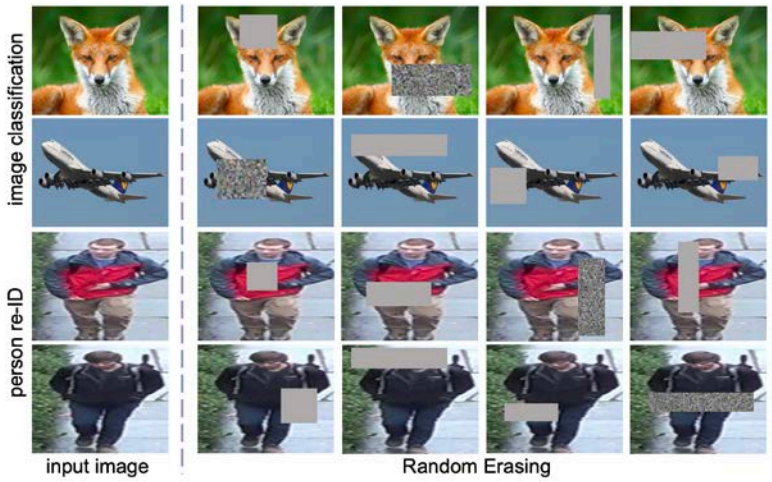

Fig. 3. Examples of image augmentation using random erasing, which involves removing or masking out part of the images.

Methods Based on Image Mixing

Among traditional and advanced augmentation techniques used in computer vision, MixUp and CutMix have been successfully used to improve generalization and robustness in deep learning models. Unlike basic augmentations such as flipping or rotating, MixUp and CutMix work by blending samples at the image and label levels, encouraging neural networks to learn smoother decision boundaries and reduce reliance on correlations such as background or texture cues.

MixUp is a linear interpolation-based augmentation technique which involves creating new training images by linearly combining two input images and their corresponding labels (Fig. 4.).

Considering two images, x_i and x_j, and their corresponding labels, y_i and y_j, a synthetic image x with a corresponding label y is generated as:

$$x = \lambda x_i + (1-\lambda)x_j$$
$$y = \lambda y_i + (1-\lambda)y_j$$

where $\lambda \in [0, 1]$ (lambda) is generated from a beta distribution Beta(α, α), α > 0.

In MixUp, the labels of the augmented instances are also obtained by linearly-interpolating the labels of the original images. During training, the loss function is also computed by linearly interpolating the loss functions corresponding to the original labels. Assuming cross-entropy loss CE is used, for an image obtained by mixing images i and j with labels y_i and y_j, for which the model prediction is $\hat{y}$, the loss is:

$$CE = \lambda CE(\hat{y}, y_i) + (1-\lambda)CE(\hat{y}, y_j)$$

MixUp has the disadvantage that blended images can be unnatural-looking, especially when the two images are semantically or structurally dissimilar.

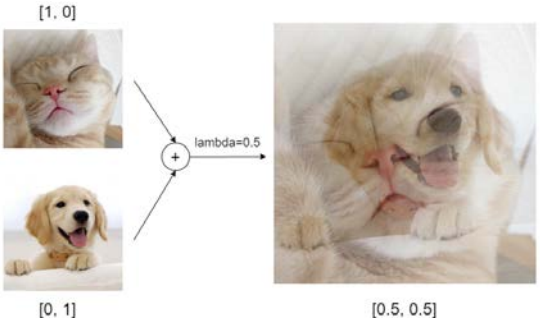

Fig. 4. Example of an image resulting from mixing two images. The one-hot-encoded labels are mixed using the same logic.

CutMix improves upon MixUp by combining patches of images rather than performing pixel-wise blending. It replaces a rectangular region of one image with a patch from another image and adjusts the label proportionally to the area of the patch (Fig. 5).
Given two images, x_i and x_j, with labels y_i and y_j, CutMix creates a new instance (x, y) as follows:

- Generate λ from a beta distribution Beta(α, α), in the same was as MixUp
- Compute the area of the patch to cut using ratio:

$$\text{cut\_ratio}=\sqrt{1-\lambda}$$

- Choose a random position in the image and define the bounding box for the patch based on the cut_ratio (i.e. a subregion of the image of size (w*cut_ratio, h*cut_ratio))
- Replace the patch in x_i with the corresponding region from x_j, resulting in x
- The loss function is computed using the same ratio λ, similarly to MixUp

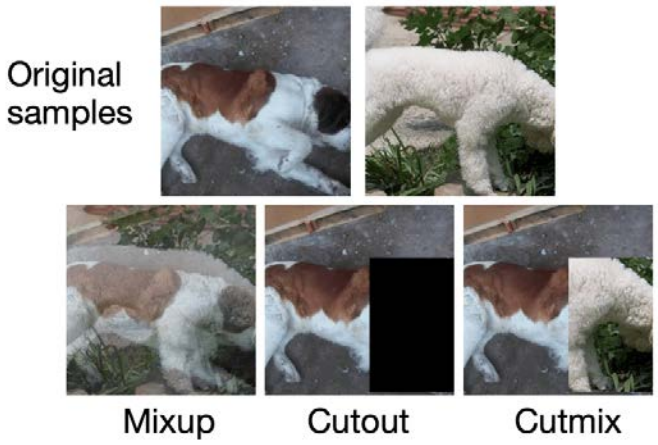

Fig. 5. Examples of mix-based augmentation. In MixUp, the resulting image is obtained by blending two images from the original data set. In CutMix, a subregion (patch) of one of the images is pasted onto the other image, creating a combination of the original instances where local features are preserved.

Tasks

For completing the tasks, the code samples that are provided with this documentation can be used as starting points.

1) Implement and test various traditional augmentation techniques on an image data set, such as:
   - Random horizontal flipping
   - Random cropping
   - Color jittering
   - Random rotations

   These can be applied in PyTorch by adding to the transformations that are carried out when loading the data set (currently, the data is converted to PyTorch tensors and normalized):

```
transform_train = transforms.Compose([
       transforms.ToTensor(),
       transforms.Normalize((0.5, 0.5, 0.5), (0.5, 0.5, 0.5))
        ])
```

   Other transformations may be added to the images in a similar fashion. For example, randomly flipping the images can be done as follows::

```
transform_train_aug = transforms.Compose([
       transforms.RandomHorizontalFlip(),
       transforms.ToTensor(),
       transforms.Normalize((0.5, 0.5, 0.5), (0.5, 0.5, 0.5)),
       ])
```

2) Implement and test the MixUp and CutMix augmentation strategies.

3) Verify how/if the various augmentation techniques improve model convergence/accuracy.