Data Augmentation

Marcus Bloice

21st March 2017

Introduction to Data Augmentation

Data Augmentation

What is data Augmentation?

- It is the **generation of artificial data through the expansion of an original dataset**
- The idea is that you **use the data you have to create more data**

Why is it used?

- Large neural networks **require lots of data** so augmentation is now basically part of **most deep learning projects**
- It is also used to address issues with **class imbalance**
- It is a cheap and relatively easy way to get more data, which will almost certainly improve the accuracy of a trained model

Data Augmentation

Wait, isn't this the era of big data?

- Experiments or clinical trials end, data is expensive to gather, more data is impossible to get, etc.
- Even in this area of "big data", it doesn't help if you personally do not have much data...

How does data augmentation help?

- It improves **model generalisation**, **model accuracy**, and can **control overfitting**
- It does this by generating **real-world feasible data** that your algorithm can learn from, that it would **not have seen in your original training set**!

Data Augmentation

Data augmentation can be **essential** for dealing with issues such as **class imbalance**.

- Class imbalance occurs in classification problems, where you have one class with very few samples compared to other classes in your dataset
    - Example: in a binary classification problem where Class A has 100 samples, while Class B has 10,000 samples
- Neural networks have real difficulty learning good, unbiased models using datasets with high class imbalance
    - This relates to how neural network weights are updated when learning

As mentioned, it is a **cheap and relatively easy way to get more data** that can make a model more robust to unseen data. In many situations, data augmentation is simply a **no-brainer**.

Image Augmentation

Data augmentation is a general term and is used for many types of data. For this seminar we will, however, concentrate on on **image** augmentation in particular.

Image augmentation is actually the most commonly seen form of augmentation.

Why?

- Text based augmentation is much harder
- Deep learning often deals with image data, and deep learning requires very large datasets

In the largest neural networks and very deep learning, image augmentation is **almost a requirement** as neural networks get more difficult to train due to ever increasing numbers of neurons that need to be adjusted when learning.

Image Augmentation

So how is image augmentation generally performed?

In principle, image augmentation is done by making **label-preserving transformations** to the original images in your dataset.

- Transformations include:
    - Rotations: arbitrary rotations, left or right
    - Zooming: zooming by arbitrary factors
    - Cropping: crop centre, crop random region
    - Transpose: flip through horizontal or vertical axis
    - Perspective Skewing: skew by arbitrary angles

Or anything else that makes sense for the particular machine learning problem you are trying to solve!

Label-Preserving Transformations

It is key that you generate **label preserving** data. But what does that actually mean?

- In supervised learning algorithms deal with data which is labelled
- This means that every image fed as input to the algorithm has an associated label

Image	Label
	8
	6

Any transformations should not be so severe that the label is no longer valid, or that the new sample is not real-world feasible

Label-Preserving Transformations

However, transforms can indeed be so dramatic that the label is no longer valid.

For example, rotation transforms:

- = "6"
- = "6" after 21 degree rotation
- != "6" after 180 degree rotation
- However: + 180 degree rotation gives

Hence we cannot discard all 180 degree rotations! **The types of augmentation that you perform are dependent on the original dataset and the problem you are trying to solve.**

Pre-processing

Related to image augmentation is image pre-processing. Pre-processing techniques are often performed alongside augmentation techniques during the pre-learning phase of a machine learning project.

- Pre-processing includes operations such as **histogram equalisation**, **colour adjustments**, or **converting images to greyscale**
- Pre-processing is typically performed on every image in your dataset, normally directly **before** or **after** your augmentation!
- The key distinction is that pre-processing **generally does not create new data**! It is **applied to all data samples**.

For example, histogram equalisation is **useless** as a method if it is **not** applied to all samples in your dataset.

Augmentation Example

In machine learning projects, it is often the case that **several augmentation techniques are applied one after another to a set of images**.

To demonstrate this, we will go through an example.

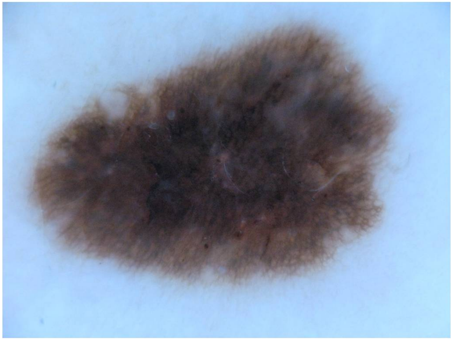
So to augment an image dataset we might:

1. First rotate each image in a dataset
2. Then zoom every image by a certain factor
3. Then crop a random region from each image
4. Then finally resize each image to the same dimensions

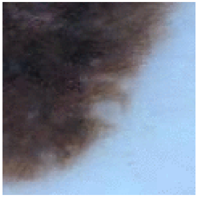
Resizing is important as neural networks often **require a uniform image size** as input.

Augmentation Example

Rotate 90° → Rotate 270° → Flip Vertical → Flip Horizontal → Crop Random → Resize → repeat × 50:



→



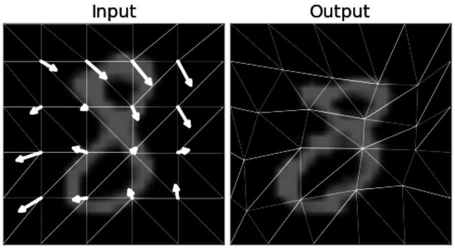
Skin lesion image obtained from the ISIC Archive[1].

[1]Download from `https://isic-archive.com:443/api/v1/image/5436e3abbae478396759f0cf/download`

Advanced Augmentation Techniques

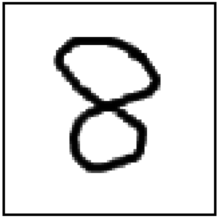
Of course, augmentation can be more advanced that just some rotations and translations through axes.

For example, **random elastic distortions**, where the centre of an image is **distorted without affecting the size or aspect ratio of the image**:

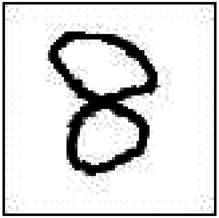


Advanced Augmentation Techniques

Using elastic distortions you can create many images based on a single original image:



$\rightarrow$



The original image has a 1 pixel black border to emphasise that **you can achieve rotation-like distortions without changing the size of the image**.

Augmentor

The Augmentor Python Package

Augmentor is a software package for Python that we have written.

It aims to make image augmentation simple, using a standalone library that is open source, free, and MIT licensed.

- With Augmentor a **pipeline-based approach to image augmentation** is used
  - This means that **operations**, such as zoom or rotate, are added to a pipeline by the user, and these are then applied to your images
- Augmentor then applies the operations to images **probabilistically**
- Augmentor is highly parametric, offering fine grain control over the generation and augmentation of images

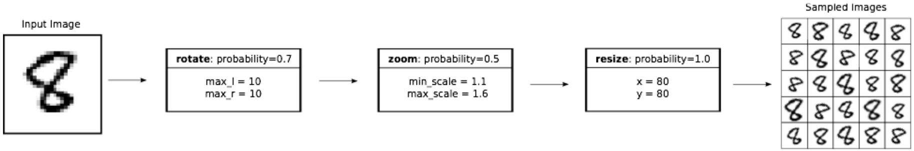
Also, Augmentor has been designed to be easily extensible and modular.

Pipeline Approach

Augmentor's pipeline approach uses a **stochastic, pipeline-based API** for generating images.

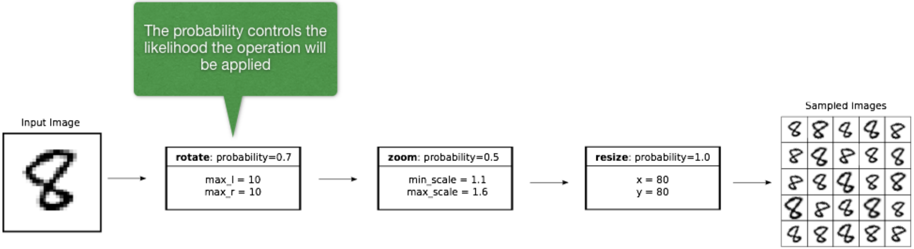
- Your first instantiate a `Pipeline` object, and then add **operations** to the pipeline
- Each operation also has its own **associated probability parameter** which defines the chance that the operation is applied to each image
- Each operation also has its **own operation specific parameters**, e.g. the `Rotate` operation allows you define the rotation in degrees
- These parameters can also be supplied as ranges, where the value applied is chosen randomly within this range each time it is called
- When a pipeline has been defined, you can **sample** from this pipeline. **Because it is stochastic, each time you pass an image through the pipeline, you get a different image**.

Pipeline Example



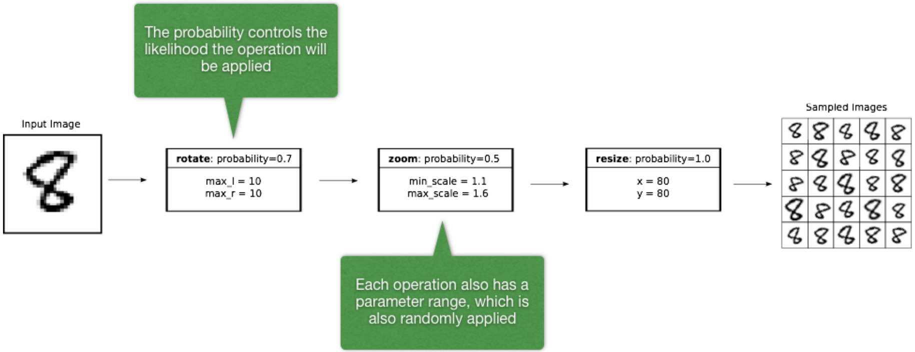
A single image can be passed through the pipeline many times. **This is because operations are applied stochastically, and each pass through the pipeline results in a new, slightly different image**.

Pipeline Example



A single image can be passed through the pipeline many times. **This is because operations are applied stochastically, and each pass through the pipeline results in a new, slightly different image**.

Pipeline Example



A single image can be passed through the pipeline many times. **This is because operations are applied stochastically, and each pass through the pipeline results in a new, slightly different image**.

Sample Code

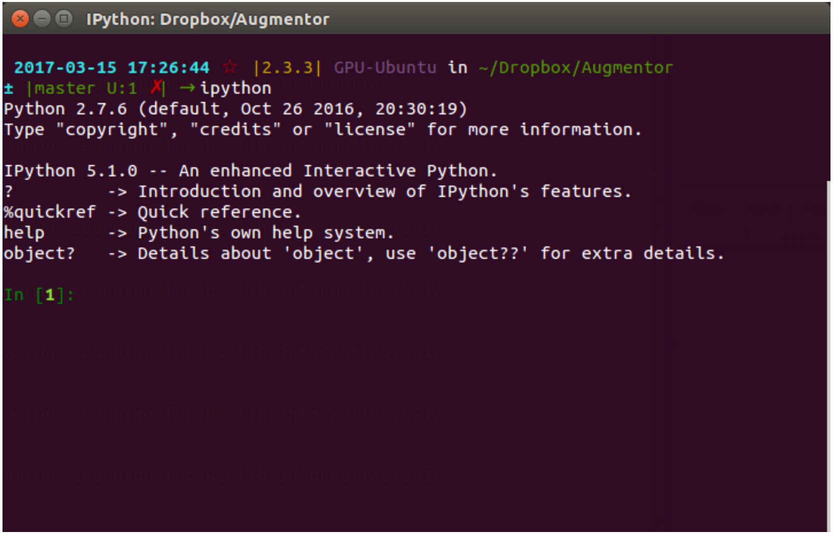
```
In [1]: import Augmentor
In [2]: p = Augmentor.Pipeline("/home/user/data")

In [3]: p.rotate90(probability=0.5)
In [4]: p.rotate270(probability=0.5)
In [5]: p.flip_left_right(probability=0.8)
In [6]: p.flip_top_bottom(probability=0.3)
In [7]: p.crop_random(probability=1, percentage_area=0.5)
In [8]: p.resize(probability=1.0, width=120, height=120)

In [9]: p.sample(100)
```

Demo

We will now make a quick demo using the IPython interactive shell



```
 2017-03-15 17:26:44 ☆ |2.3.3| GPU-Ubuntu in ~/Dropbox/Augmentor
± |master U:1 ✗| →ipython
Python 2.7.6 (default, Oct 26 2016, 20:30:19)
Type "copyright", "credits" or "license" for more information.

IPython 5.1.0 -- An enhanced Interactive Python.
?         -> Introduction and overview of IPython's features.
%quickref -> Quick reference.
help      -> Python's own help system.
object?   -> Details about 'object', use 'object??' for extra details.

In [1]:
```

Assignments

Introduction to Assignments

For this course, practical assignments have to be completed for the VU.

As mentioned previously, Augmentor has been designed to highly modular, and can be easily extended.

The assignment for this course, therefore, is to:

1. **Extend Augmentor with a new operation**
2. Perform an **empirical analysis of your new operation** (in other words, benchmark it[2])

The project is intended as **group work**, for groups of **up to four**.

[2]Several open benchmarking datasets exist, for example MNIST: `http://yann.lecun.com/exdb/mnist/`

Introduction to Assignments

Because most standard features are already implemented, I would recommend that you:

- See what has already been implemented on the Augmentor GitHub repository
- Read the current literature (arXiv is a good source for machine learning papers)
- Find the newest techniques that are currently in use
- Pay close attention to competition websites such as Kaggle

When you have decided what operation you want to implement, contact me. Not everything will be feasible, desirable, or useful :-)

Augmentor API

To extend Augmentor, you will need to know about its structure.

Augmentor consists of two main modules:

1. The `Pipeline` module
2. The `Operations` module

The user interacts with the `Pipeline` module. This is how the user defines the pipeline.

The `Operations` module contains operation classes. Operation classes are things like `Zoom`, `Rotate`, `Distort`, and so on.

Note, in Python, modules are merely files. The `Operations` module therefore is a file named `Operations.py`.

Augmentor API

So, to extend Augmentor and add a new operation, you must **add a new class to the `Operations` module** (`Operations.py`).

- The `Operations` module contains an abstract base class called `Operation`.
- To add a new operation, extend the `Operation` base class and overload its methods.

When an a new operation has been added, the user can add the operation to a pipeline using the `Pipeline` class.

Augmentor API

To add a custom operations, you must inherit from the `Operation` base class, and implement your own `__init__()` and `perform_operation()` functions.

```
# The Operation Abstract Base Class
class Operation(object):
    def __init__(self, probability):
        self.probability = probability

    def perform_operation(self, image):
        ...
        return image
```

Augmentor API

For example, this is the rotate operation's implementation:

```
class Rotate(Operation):
    def __init__(self, probability, rotation):
        Operation.__init__(self, probability)
        self.rotation = rotation

    def perform_operation(self, image):
        return image.rotate(self.rotation,
                            expand=True)
```

Augmentor API

```
# Extend Operation class
class Rotate(Operation):
    # Define your own constructor.
    # At a minimum you need to accept a probability
    # parameter,  and then as many custom parameters
    # as required
    def __init__(self, probability, rotation):
        # Call the super class's constructor, which
        # requires the probability parameter
        Operation.__init__(self, probability)
        # Save all your custom parameters as member
        # variables
        self.rotation = rotation
    ...
```

Augmentor API

```
...
# Override Operation's perform_operation function
# Ensure you accept an Image type
def perform_operation(self, image):
    # Perform your custom code here
    # and ensure you return an Image
    # type
    return image.rotate(self.rotation,
                        expand=True)
```

Custom Functions

Currently, most of the image transformations are performed by the Pillow library[3].

- Pillow is a fork of PIL, the **Python Imaging Library**
- Pillow provides functionality such as rotate, zoom, etc.
- Augmentor uses Pillow where possible

It is not possible to use Pillow for all operations, however.

[3] See `http://pillow.readthedocs.io`

Custom Functions

For custom operations you may be required to work on the matrix form of the images and apply transformations mathematically.

For example, you can rotate an image by mapping source coordinates x, y to destination coordinates x', y' through θ degrees about the origin as follows[4]:

$$\begin{bmatrix} x' \\ y' \end{bmatrix} = \begin{bmatrix} \cos(\theta) & sin(\theta) \\ -\sin(\theta) & cos(\theta) \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix}$$

This is for a **clockwise** rotation, otherwise swap $\sin(\theta)$ and $-\sin(\theta)$. See `https://en.wikipedia.org/wiki/Transformation_matrix`

[4] Actually, you would really use the *three shears* method, see `http://datagenetics.com/blog/august32013/index.html`

Custom Functions

```
import numpy
from PIL import Image
im = Image.open("im_file.png")

im_array = numpy.asarray(im).astype('uint8')

print(im_array)
[[3, -3, 8, 1, 19, 18, -20, -17],
 [-13, 1, -9, -5, 1, 8, -13, -19],
 [16, -15, -2, 19, -5, 6, 6, -6],
 ..., ]

# Do your transformations on the matrix here

new_image = Image.fromarray(im_array)
```

Example Projects

Here are a few potential projects that you could work on:

- **HSV Shifting**: involves random shifting in the HSV (Hue, Saturation, Value) colour space to generate images with different lighting (adjusting Hue shift)
- **Gaussian Random Elastic Distortions**: Currently the distortions that were demonstrated earlier are from a uniform distribution, a Gaussian distortion would also be useful—note, this does not mean merely sampling from a Gaussian distribution!
- **Random Noise**: Adding random noise to images is done sometimes. Offers a lot of scope for parametrisation.

Winners of **Kaggle** competitions are often interviewed, see `http://blog.kaggle.com/category/winners-interviews/` — their augmentation techniques are generally mentioned.

Benchmarking

Once you have a new operation, you should benchmark it. This is done by empirical analysis.

1. Use a **known, open dataset**, for example the MNIST hand written digit dataset
2. Split the dataset into a **training set** and a **test set**
3. Use any algorithm and benchmark it **before augmentation/pre-processing**
4. Apply your **new operation to the training set to augment it**
5. Use the **same algorithm on the augmented training set to judge the effectiveness** of your new operation

Use any algorithm which is appropriate for image data. In Python almost all algorithms are implemented in SciKit-Learn, while deep learning neural networks can be be created using Keras.

Why Python?

You might ask why use Python for machine learning?

- Python is a general purpose programming language that is used in many fields. Along with R, it is the most popular data science and machine learning language
- Compared to R, it has a cleaner, more modern syntax
- Python comes with “batteries included”: its standard library is very large which makes development fast, and its syntax is less verbose than Java or C++
- It is truly general purpose, and is used for game development, server-side scripting, data science, web development, etc.
- Many libraries for data science are available: SciKit-Learn, Keras, SciKit-Image, Numpy, Pandas

If you have any questions about Python, just ask.

Recap

So, to recap, for the assignment for this course, you should:

1. Get into groups of three or four
2. Read literature, arXiv, Kaggle, etc. and see what techniques are modern and in use
3. Contact me and tell me what you would like to do for your assignemnt
4. Code your extension and make a pull request on the GitHub repository
5. Perform a benchmark of your new operation to judge its effectiveness

Any questions regarding the assignment?

Resources

Resources regarding the Augmentor project:

- Augmentor on GitHub: `https://github.com/mdbloice/Augmentor`
- Augmentor documentation: `http://augmentor.readthedocs.io`
- Gitter chat: `https://gitter.im/Augmentor`

Deadlines:

- Submission of group work project proposal: 4th of April 2017 23:59:59
- Submission of project: end of the semester.

Contact me: `marcus.bloice@medunigraz.at`

Any questions? Ask now!