Data Preparation and Image Processing

Transform Class

Responsible of setting the correct image dimensions & normalization.

- IMGaug library: Efficient image manipulation functionalities on GPU
- **Image augmentation is a technique used when we only have limited samples and we need hundreds or thousands of samples to be created from these limited samples. Look into these transformations below, but also look into image augmentation as a whole and see what this library gives**.
- Initialize class with dimensions and transformation parameters and when called it should return the transformed image.
- Look into the following image transformations:
  - Resize: different width transforms (look into what the width parameter of the function does)
  - Rotate
  - Crop to fixed size
  - Gaussian Blur
  - Sometimes (image augmentor function)
  - Multiply Brightness

Milestone #1

- Your able to read images in the dataset (without using pytorch), to read the images use matplotlib.
- Have a graphic showing the original image and then next to it the transformed image.

Create Dataset Class

- Class that holds the paths of each item in the dataset and holds the logic to get each item given its path and return the image, the bboxes, and the transcripts
- Technical:
  - Create initialization method which takes in the data IDs and Labels and sets them as attributes.
  - Define Len method which returns the length of the labels array.
  - Define a getitem method which takes in an id, loads the data and label for that id and returns them.
  - Create a dictionary defining the train/test split with ids and another with each id and its respective label.
  - The collate function specifies how the data's structure will be transformed.

```
>>> partition
{'train': ['id-1', 'id-2', 'id-3'], 'validation': ['id-4']}
```

```
>>> labels
{'id-1': 0, 'id-2': 1, 'id-3': 2, 'id-4': 1}
```

Example

- Instantiation takes data and assigns it to the class.
- The getitem method takes in an index to a datapoint using the specified path it loads that data and returns it for use.

```python
import torch

class Dataset(torch.utils.data.Dataset):
  'Characterizes a dataset for PyTorch'
  def __init__(self, list_IDs, labels):
        'Initialization'
        self.labels = labels
        self.list_IDs = list_IDs

  def __len__(self):
        'Denotes the total number of samples'
        return len(self.list_IDs)

  def __getitem__(self, index):
        'Generates one sample of data'
        # Select sample
        ID = self.list_IDs[index]

        # Load data and get label
        X = torch.load('data/' + ID + '.pt')
        y = self.labels[ID]

        return X, y
```

Details

- One of the methods of your transform class if you research data augmentation will be to crop your image. If you call the transform class, then you will randomly crop the image.
- We need to have a function called “checkBoxCrop” which will check that there is an entire bbox contained in the crop.

Milestone #2

- Show the transform used with the dataset class obtaining the images instead of a direct path

Dataloader

- Parameters:
  - Dataset class
  - Batch size
  - Number of workser
  - Collate function
  - Shuffle function
- Adding 1 more layer of complexity, a function that gets our dataset class, calls n workers, gets the data, uses the collate function to assemble a tensor, and then returns that tensor

Pytorch Script

- Creating a script to load the data:
- Instantiate the dataset class for the train and test data.
- Use the pytorch dataloader and pass in training set and a parameter dictionary containing batch size, shuffle bool, and number of workers to obtain mini batches and their labels.
- Iterate through epochs transferring the batches to the GPU to do the computations. Also do the same with the test labels.

Script Example

- Partition and labels are dictionaries with the data IDs and their type (test/train) and IDs and their label.
- Dataloader produces array with local batches and their labels.
- Disable local gradient calculations with .set_grad_enabled.

```python
use_cuda = torch.cuda.is_available()
device = torch.device("cuda:0" if use_cuda else "cpu")
torch.backends.cudnn.benchmark = True

# Parameters
params = {'batch_size': 64,
          'shuffle': True,
          'num_workers': 6}
max_epochs = 100

# Datasets
partition = # IDs
labels = # Labels

# Generators
training_set = Dataset(partition['train'], labels)
training_generator = torch.utils.data.DataLoader(training_set, **params)

validation_set = Dataset(partition['validation'], labels)
validation_generator = torch.utils.data.DataLoader(validation_set, **params)

# Loop over epochs
for epoch in range(max_epochs):
    # Training
    for local_batch, local_labels in training_generator:
        # Transfer to GPU
        local_batch, local_labels = local_batch.to(device), local_labels.to(device)

        # Model computations
        [...]

    # Validation
    with torch.set_grad_enabled(False):
        for local_batch, local_labels in validation_generator:
            # Transfer to GPU
            local_batch, local_labels = local_batch.to(device), local_labels.to(device)
```

Decoding

- Some txt files have UTF-8-sig encoding which have a BOM (byte order mark) at the beginning of the file to state the encoding type.
- The IDCAR dataset uses this encoding, so at the beginning of all files it has the hex sequence 0xEF,0xBB,0xBF which results in the symbols ï»¿.
- To get rid of this BOM the following decoding method is used when the file is opened.

```
file = open((("labels/gt_img_{}.txt").format(ID)), "r",encoding='utf-8-sig')
```

Milestone #3

- Show the transform used with the dataloader class obtaining the images instead of a direct path
- Instead of printing 1 image, you print a batch of 5 images.

Python Classes

Classes

- Classes provide a means of bundling data and functionality together. Creating a new class creates a new type of object, allowing new instances of that type to be made. Classes have a state and each class instance can have attributes attached to it for maintaining its state.
- Classes provide a new space where functions and variables can be defined, and they become accessible with className.parameter. So myClass.i would return 12345. And myClass.f() would be a function call to f, which could have some set inputs. In this case ‘self’ is not an input but a reference to the object iself. Just like ‘this’ in react.

```
class MyClass:
    """A simple example class"""
    i = 12345

    def f(self):
        return 'hello world'
```

Namespaces

- Namespaces are the different spaces available where object names identify different spaces in memory.
- As can be seen in this example, the local, nonlocal, and global namespaces all have different effects on setting a variable inside a function.
- The local assignment only changed the variable definition in the function, nonlocal changed it in the main space and the function, global changed it in the function, main space, and the global scope (other function spaces)

```python
def scope_test():
    def do_local():
        spam = "local spam"

    def do_nonlocal():
        nonlocal spam
        spam = "nonlocal spam"

    def do_global():
        global spam
        spam = "global spam"

    spam = "test spam"
    do_local()
    print("After local assignment:", spam)
    do_nonlocal()
    print("After nonlocal assignment:", spam)
    do_global()
    print("After global assignment:", spam)

scope_test()
print("In global scope:", spam)
```

```
After local assignment: test spam
After nonlocal assignment: nonlocal spam
After global assignment: nonlocal spam
In global scope: global spam
```

Class Initialization

- Class instantiation uses function notation. The class object is a function that returns a new instance of the class. So an instance of a class can be assigned to a local variable like so.
- A class usually has an initialization method, which takes inputs and sets the state (self) to those inputs.
- The initialization method is defined with __init__, there are many other methods like this, called special methods.

```
x = MyClass()
```

```
>>> class Complex:
...     def __init__(self, realpart, imagpart):
...         self.r = realpart
...         self.i = imagpart
...
>>> x = Complex(3.0, -4.5)
>>> x.r, x.i
(3.0, -4.5)
```

Special Methods

- Special methods are what give python classes object attributes.
- This example gives novel a __str__ attribute and a __len__ attribute, so when the object is printed or its length is checked for the respective attributes are shown.
- An object can be deleted with del Name, and a __del__ method can be specified to execute some logic when the object gets deleted.

```
>>> class Novel():
        def __init__(self, title, author, pages):
            self.title = title
            self.author = author
            self.pages = pages


        def __str__(self):
            return ("The novel {} is written by
{}".format(self.title, self.author))


        def __len__(self):
            return (self.pages)


>>> my_novel = Novel("Habits", "Chandra", 200)


>>> print(my_novel)
The novel Habits is written by Chandra


>>> len(my_novel)
200
```

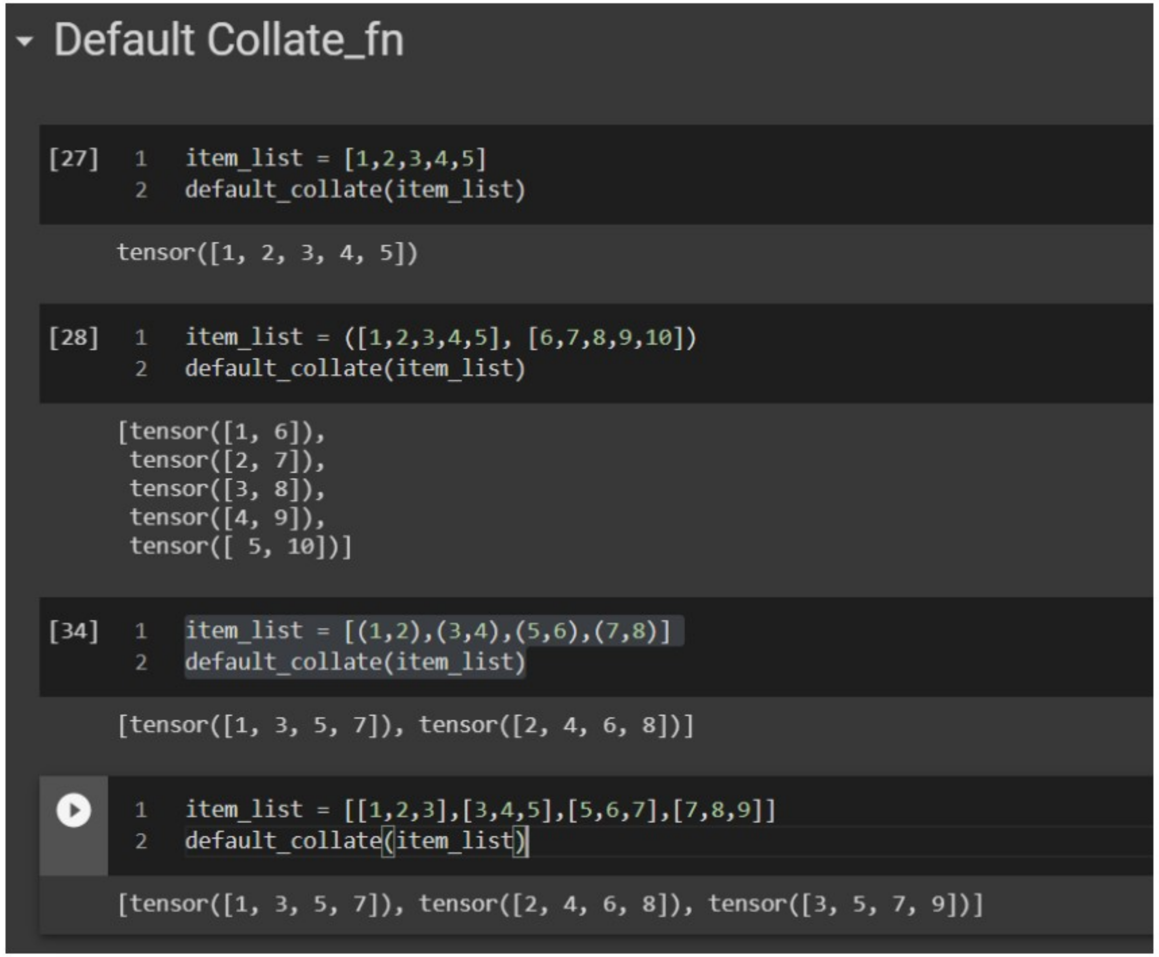
Useful Methods

Method	Signature	Explanation
Length	`__len__(self)`	`len(x)` invokes `x.__len__()`
Get Item	`__getitem__(self, key)`	`x[key]` invokes `x.__getitem__(key)`
Set Item	`__setitem__(self, key, item)`	`x[key] = item` invokes `x.__setitem__(key, item)`
Contains	`__contains__(self, item)`	`item in x` invokes `x.__contains__(item)`
Iterator	`__iter__(self)`	`iter(x)` invokes `x.__iter__()`
Next	`__next__(self)`	`next(x)` invokes `x.__next__()`

- There's also the __init__(self,inputs) method which is called when an object is initialized. This method generally sets the inputs onto the self 'identity' parameter, which is just a reference to the specified obj.

Collate Function

- The collate function is the function that the dataloader uses to transform the data batch into the desired tensor with a specific data structure.
- If the images are m by n, then the data is initially an array with X m by n objects, this function turns it into a single m by n by X tensor.
- The default function's behavior is the one seen on the right.



Dataloader Errors

```
RuntimeError: DataLoader worker (pid(s) 21656, 304) exited unexpectedly
```

- Dataloader errors don’t usually show the real problem, as seen above. To find the underlying issue set the number of workers to 0.
- For the dataloader to work, the images must be a writeable array. This means that they have an allocated slot in memory , so reading them is not enough.
- To fix this, I opened the images with matplotlib, read them as numpy arrays, and then created a copy of the array to save it in memory.
- To check if the array is writeable the image flags can be used as such.

```
# Check if images are writeable arrays
print(image.flags['WRITEABLE'])
```

Img Aug

- To do multiple augmentations we create a sequence which is just a series of steps that are essentially function calls.
- The augmentations are meant to be random for Machine learning, so the sometimes(p, fn) function takes in a probability and a function that it will or will not execute on the image input into the sequence with that probability.
- Using the lambda function definition the sometimes function can be defined as Sometimes with a set probability (50%).

```python
from imgaug import augmenters as iaa

seq = iaa.Sequential([
    iaa.Crop(px=(0, 16)), # crop images from each side by 0 to 16px (randomly chosen)
    iaa.Fliplr(0.5), # horizontally flip 50% of the images
    iaa.GaussianBlur(sigma=(0, 3.0)) # blur images with a sigma of 0 to 3.0
])

for batch_idx in range(1000):
    # 'images' should be either a 4D numpy array of shape (N, height, width, channels)
    # or a list of 3D numpy arrays, each having shape (height, width, channels).
    # Grayscale images must have shape (height, width, 1) each.
    # All images must have numpy's dtype uint8. Values are expected to be in
    # range 0-255.
    images = load_batch(batch_idx)
    images_aug = seq(images=images)
    train_on_images(images_aug)
```

```python
sometimes = lambda aug: iaa.Sometimes(0.5, aug)
```

```python
sometimes(iaa.Crop(percent=(0, 0.1))),
```

More Img Aug Functions

- The SomeOf operator takes in arrange specifying how many of the functions in its body it must execute, in this case its going to execute 0 to 5 of the functions ins the array of functions passed in as a second element.
- The OneOf operator is similar but it only executes one of the functions in its body.
- The random_order specification at the end of an operator causes the order of its operations to be random.

```
iaa.SomeOf((0, 5),
    [
                    iaa.Superpixels(
                    p_replace=(0, 1.0),
                    n_segments=(20, 200)
                ), . . .
    ]
```

```
iaa.OneOf([
    iaa.Dropout((0.01, 0.1), per_channel=0.5),
    iaa.CoarseDropout(
        (0.03, 0.15), size_percent=(0.02, 0.05),
        per_channel=0.2
    ),
]),
```

```
random_order=True
```

Resize Augmento

- The resize augmentor changes the scale of the image, it has different types of inputs as can be seen

```
>>> aug = iaa.Resize({"height": 32})
```

Resize all images to a height of `32` pixels and keeps the original width.

```
>>> aug = iaa.Resize({"height": 32, "width": 48})
```

Resize all images to a height of `32` pixels and a width of `48`.

```
>>> aug = iaa.Resize({"height": 32, "width": "keep-aspect-ratio"})
```

Resize all images to a height of `32` pixels and resizes the x-axis (width) so that the aspect ratio is maintained.

```
>>> aug = iaa.Resize(
>>>     {"shorter-side": 224, "longer-side": "keep-aspect-ratio"})
```

```
>>> aug = iaa.Resize(32)
```

Resize all images to `32x32` pixels.

```
>>> aug = iaa.Resize(0.5)
```

Resize all images to `50` percent of their original size.

```
>>> aug = iaa.Resize((16, 22))
```

Resize all images to a random height and width within the discrete interval `[16..22]` (uniformly sampled per image).

```
>>> aug = iaa.Resize((0.5, 0.75))
```

Resize all any input image so that its height (`H`) and width (`W`) become `H*v` and `W*v`, where `v` is uniformly sampled from the interval `[0.5, 0.75]`.

```
>>> aug = iaa.Resize([16, 32, 64])
```

Resize all images either to `16x16`, `32x32` or `64x64` pixels.

More Img Aug Functior

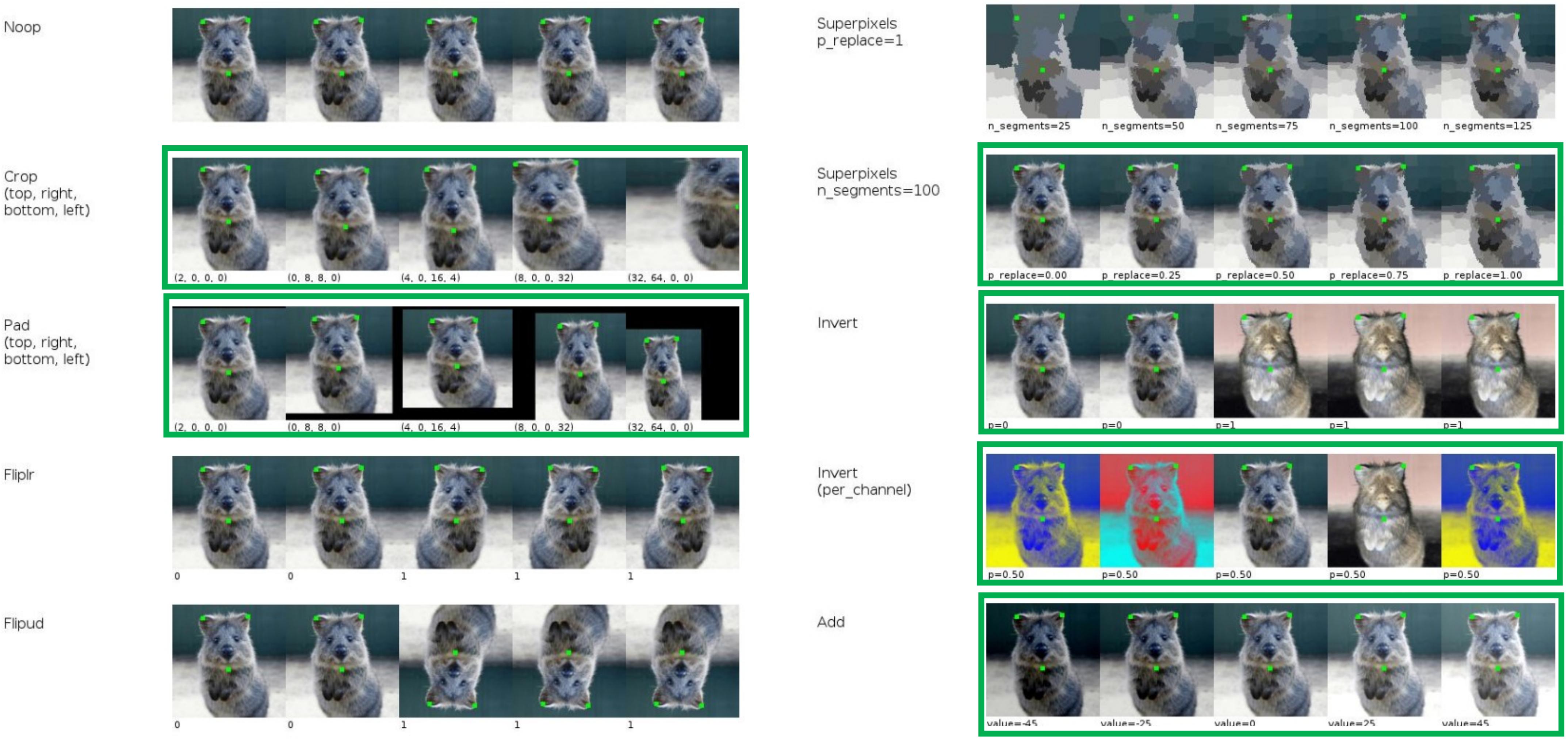
- Multiply Brightness: This function converts the images to a colorspace with a brightness related channel and it multiplies that channel by a factor between the input values then converts back to the original colorspace. There are also functions to add or add and multiply brightness.
- The crop function can take in a percent to crop or a fixed pixel crop.
- There are several blurring functions, the gaussian blur takes in a range for the value of sigma (spread of the blur)

```
aug = iaa.MultiplyBrightness((0.5, 1.5))
```

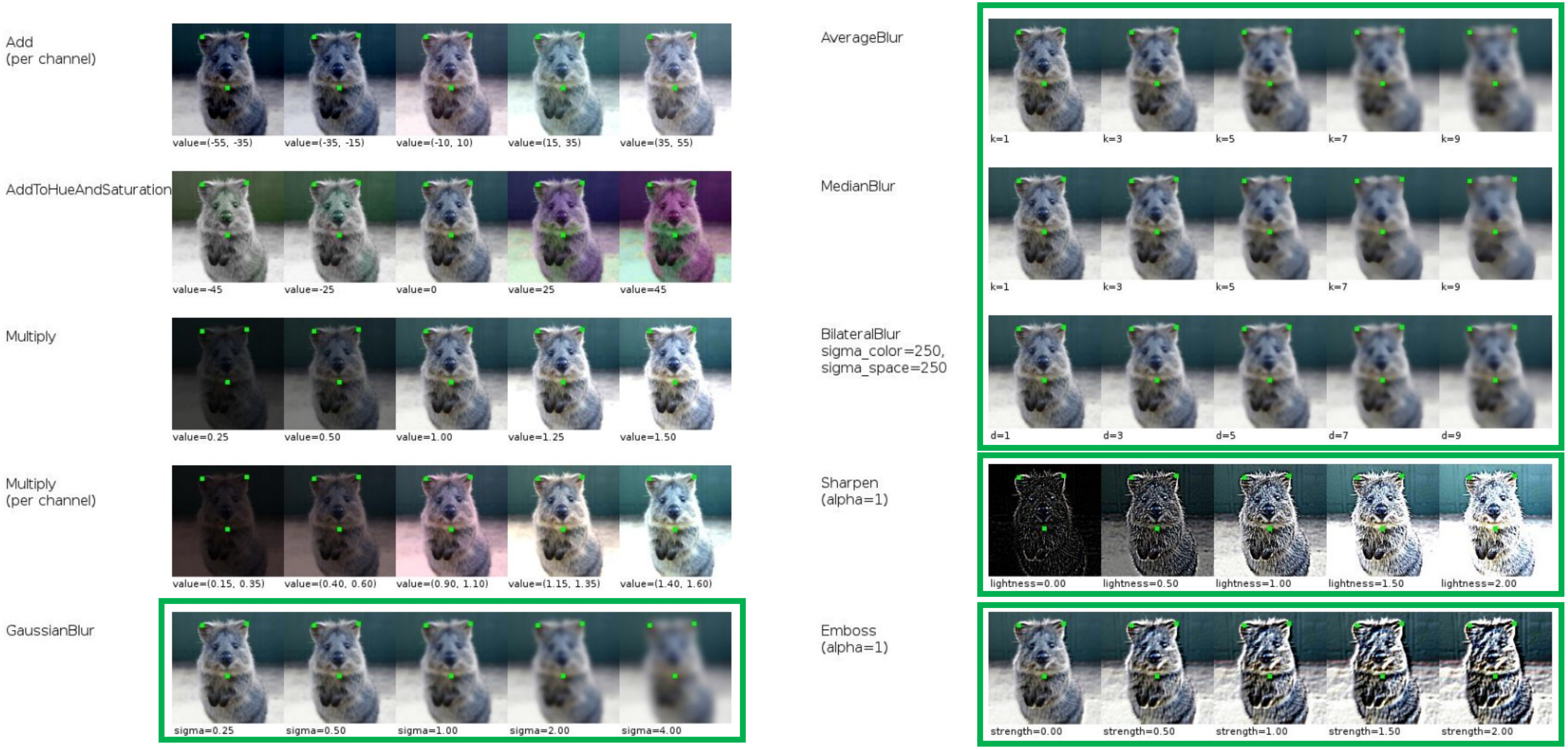
```
# Crop some of the images by 0-10% of their height/width
sometimes(iaa.Crop(percent=(0, 0.1))),

# Crop some of the images by a fixed 2px on each side
sometimes(iaa.Crop(2))
```

```
iaa.OneOf([
    iaa.GaussianBlur((0, 3.0)),
    iaa.AverageBlur(k=(2, 7)),
    iaa.MedianBlur(k=(3, 11)),
]),
```

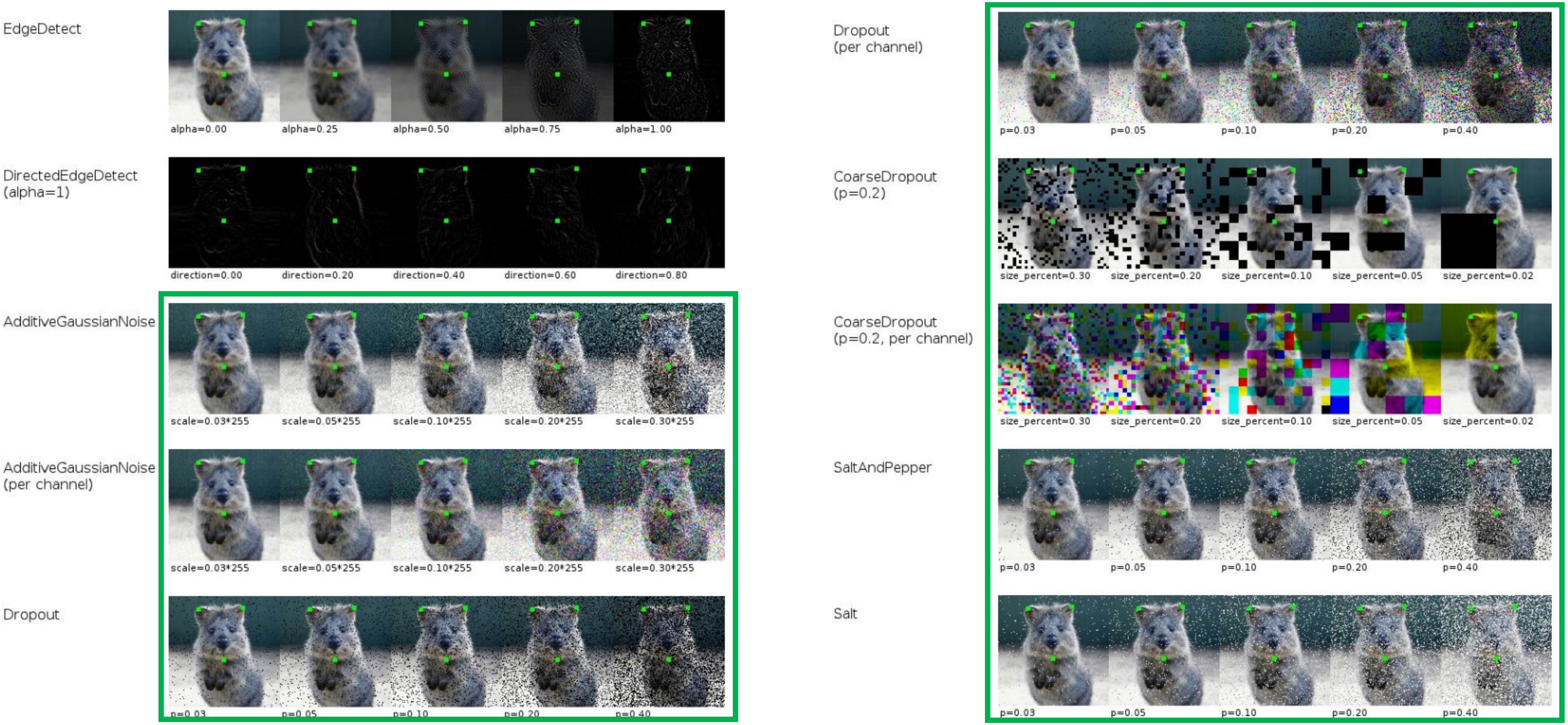
Augmentation Techniques



Available Transformations

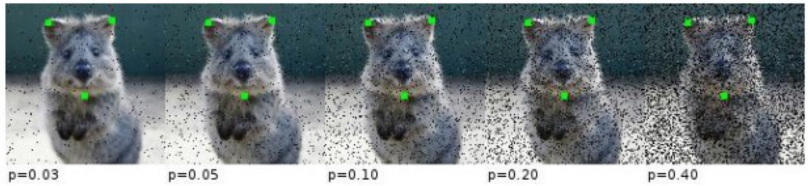


Available Transformations

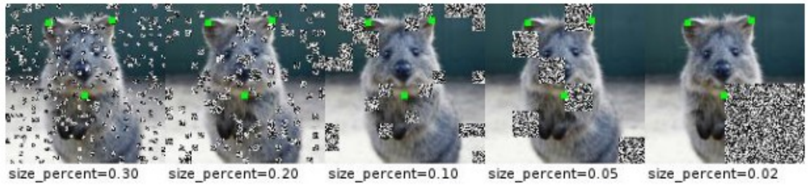


Available Transformations

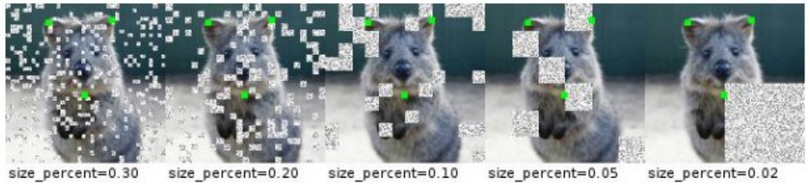
Pepper



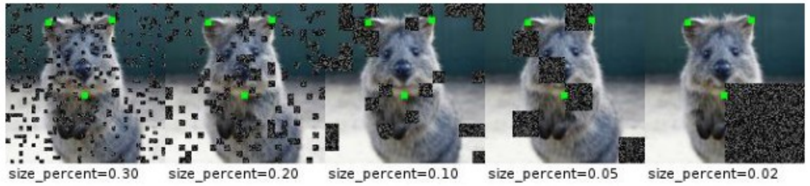
CoarseSaltAndPepper
(p=0.2)



CoarseSalt
(p=0.2)

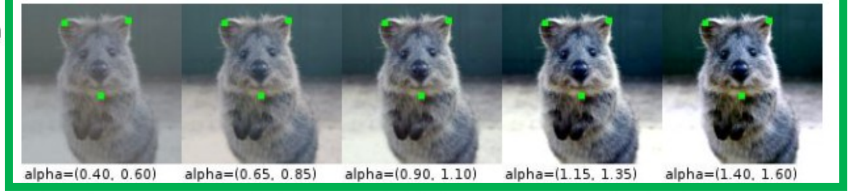


CoarsePepper
(p=0.2)



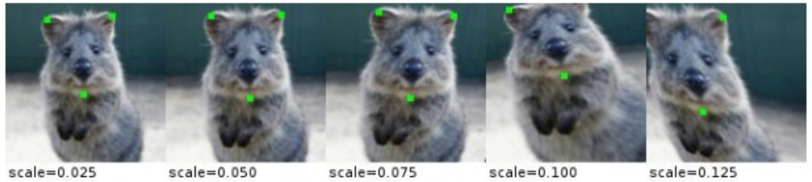
ContrastNormalization

ContrastNormalization
(per channel)



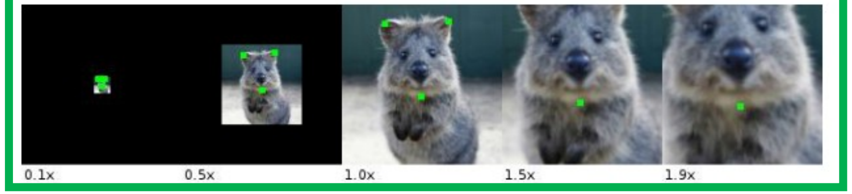
Grayscale

PerspectiveTransform

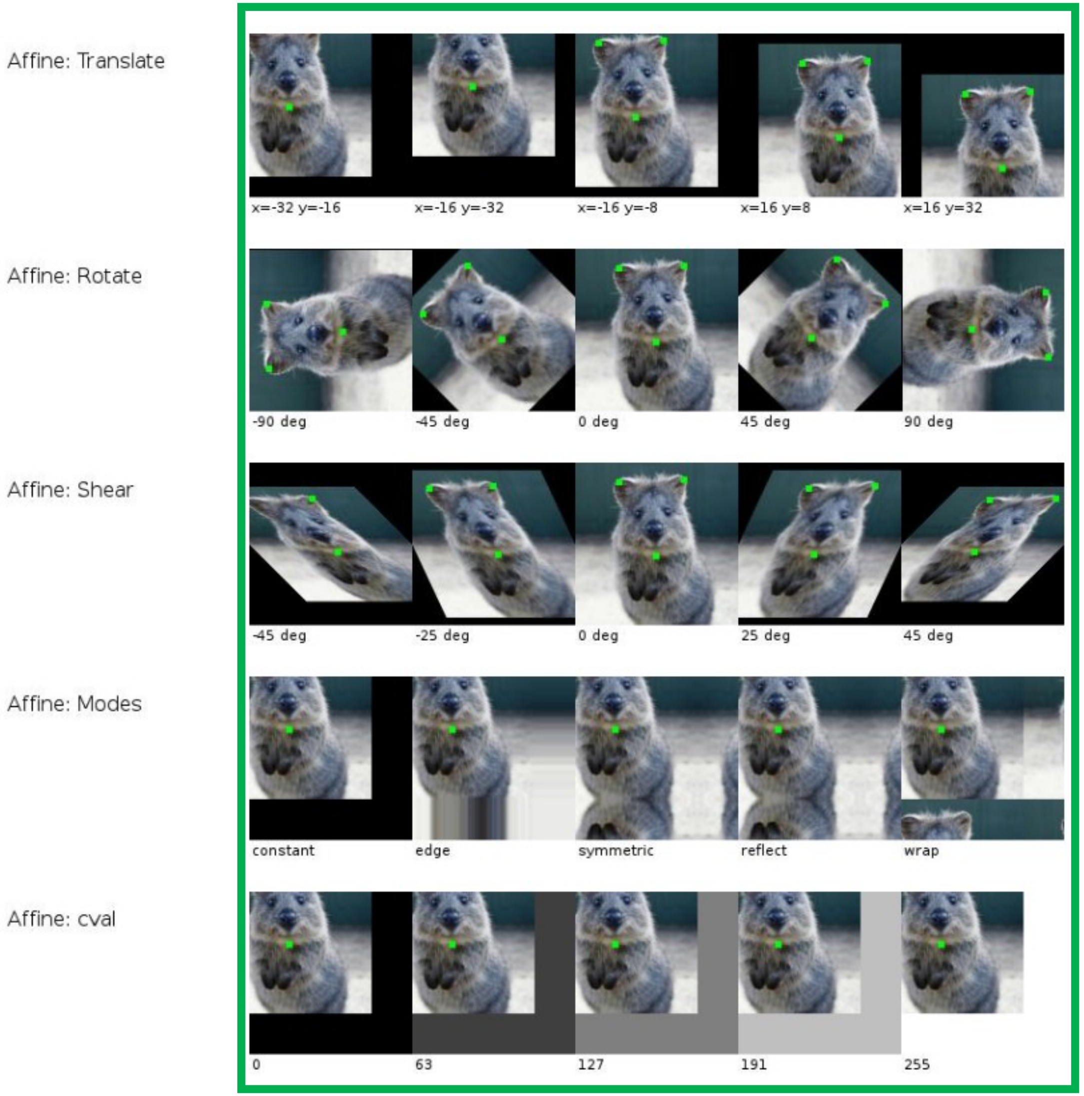


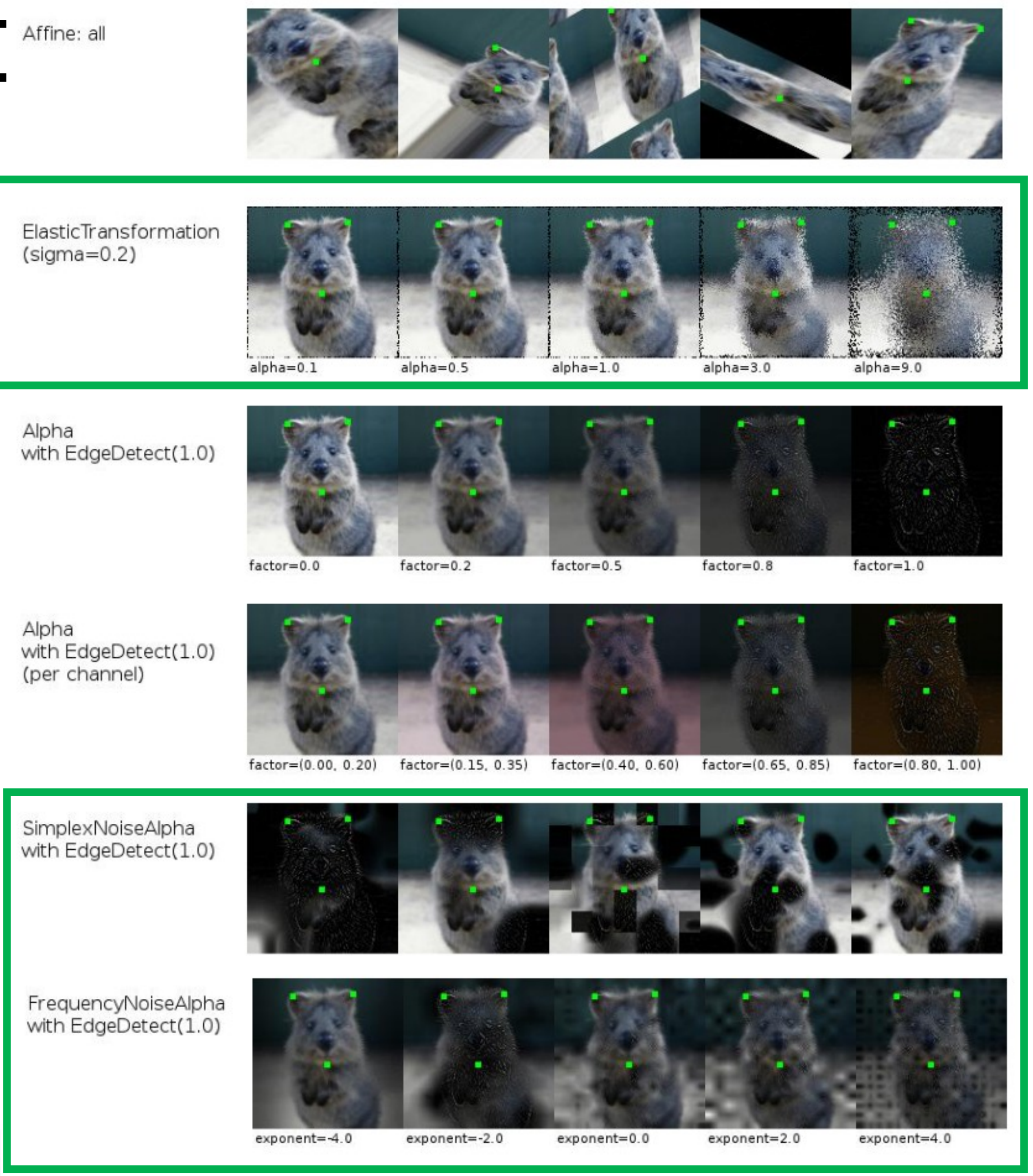
PiecewiseAffine

Affine: Scale



Available Transformat





Bounding Boxes

- The ICDAR dataset follows the following format for labels:

The text files are comma separated files, where each line corresponds to one text block in the image and gives its bounding box coordinates (four corners, clockwise) and its transcription in the format:

x1, y1, x2, y2, x3, y3, x4, y4, transcription

- IMGaug offers support for bboxes by passing them into the defined sequence with the following format.

```
bbs = BoundingBoxesOnImage([
    BoundingBox(x1=65, y1=100, x2=200, y2=150),
    BoundingBox(x1=150, y1=80, x2=200, y2=130)
], shape=image.shape)
```

```
# Augment BBs and images.
image_aug, bbs_aug = seq(image=image, bounding_boxes=bbs)
```

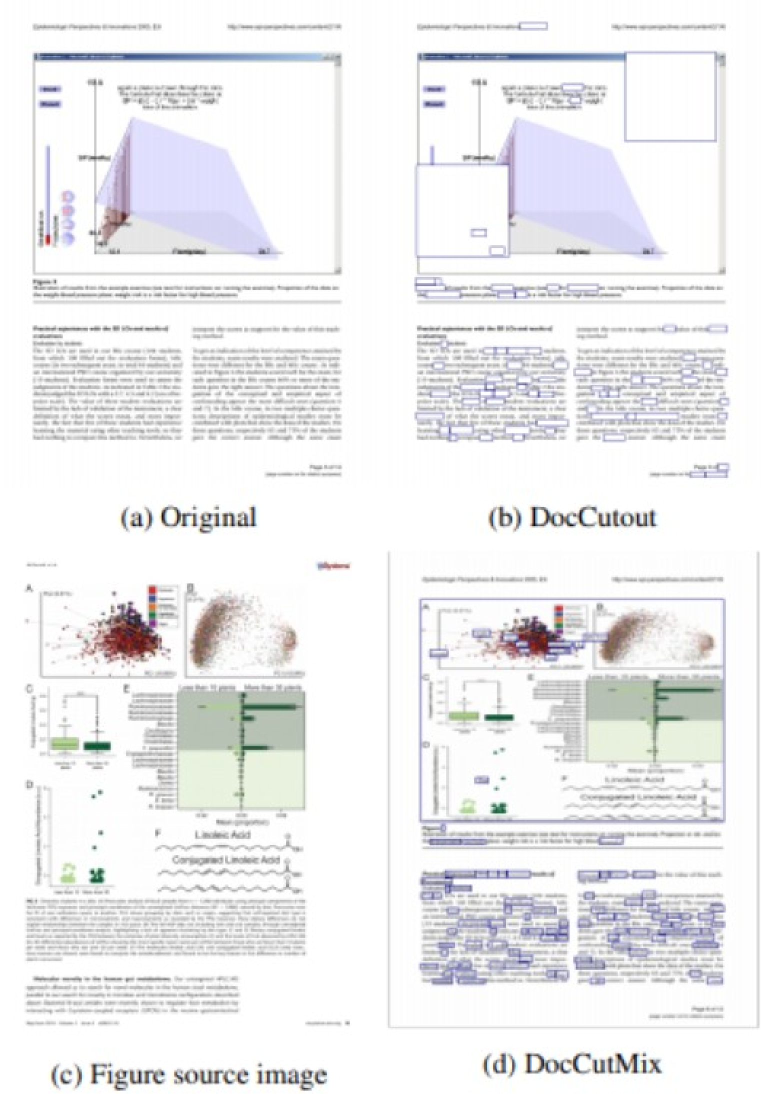
- The sequence then outputs them and they can be illustrated as follows.

```
# image with BBs before/after augmentation (shown below)
image_before = bbs.draw_on_image(image, size=2)
image_after = bbs_aug.draw_on_image(image_aug, size=2, color=[0, 0, 255])
```

Paper 1:Available Transforma

- This paper has a proposed alternative to the original cutout and cutmix methods with code that is made specifically for document augmentation purposely designed for OCR.

http://ceur-ws.org/Vol-2831/paper20.pdf



Proposed OCR alternatives

Original Cutmix and cutout with others

Paper 1 Results

- The average precision of the vanilla trained models show that when the proposed DocCutout and DocCutMix image augmentation methods are used, the model is more precise that when noise and jitter effects are used. This is a very significant improvement to the regular cutout and cutmix methods.

Augmentation	AP-figure	AP-heading	AP-listitem	AP-table	AP-text	AP
Baseline	91.32	78.99	72.96	92.54	91.87	85.07
Colorjitter	91.56	81.31	73.40	92.92	92.42	86.21
Gaussnoise	90.66	80.88	**74.14**	92.79	92.42	86.14
Affine	92.02	81.25	73.06	93.34	92.54	86.32
DocCutout (proposed)	**92.49**	**81.90**	73.99	93.52	**92.77**	**86.84**
DocCutMix (proposed)	91.88	81.63	73.02	**93.62**	**92.77**	86.33

Table 1: Comparison of various augmentations in *PubMed* document object detection
AP is Average Precision at [0.50:0.05:0.95]

Paper 2: Technique comparisons – On Historical Texts

- The most effective combination of techniques is by far the affine augmentation with the MNIST mask. The other augmentations are at least 15% lower.
- Paper

https://www.researchgate.net/publication/322239286_Methods_of_data_augmentation_for_palimpsest_character_recognition_with_Deep_Neural_Network/link/5c1bb998458515a4c7ec66fa/download

- Code

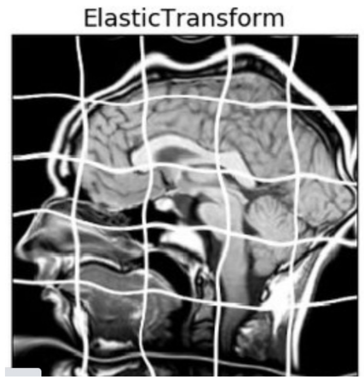
https://github.com/as3297/Image-generator/blob/master/image.py

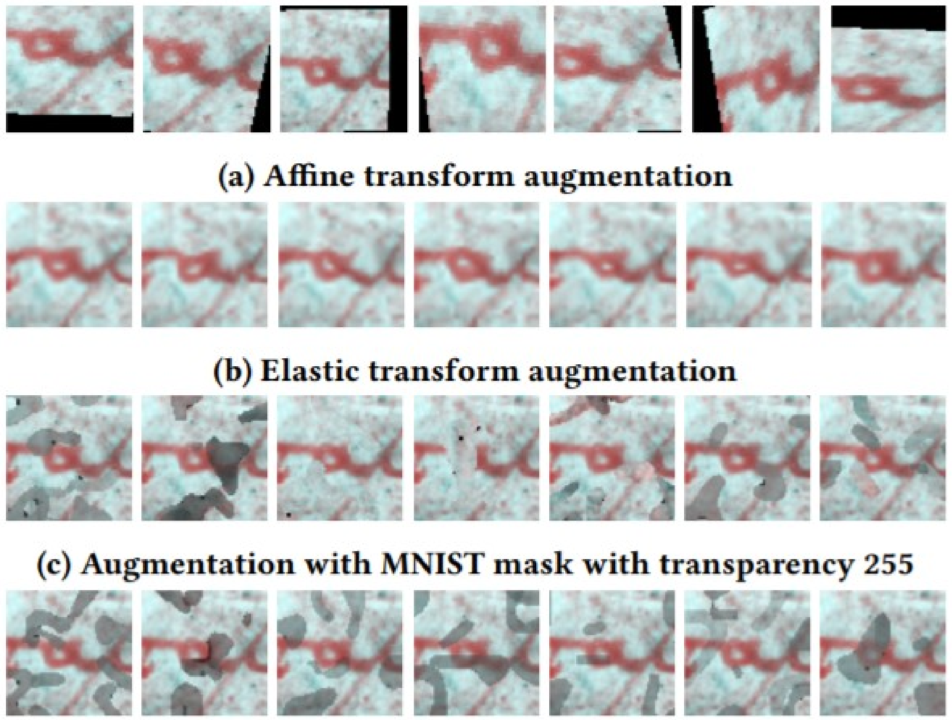
Table 1: Network performance for different augmentation techniques

Data augmentation	non-obscured accuracy	obscured accuracy
Affine augmentation, Clean characters	67%	64%
Affine augmentation, Clean+Obscured characters	62%	–
Elastic+rotation+shift augmentation	36%	27%
MNIST mask+affine augmentation, mask transparency 255	80%	72,5%
MNIST mask+affine augmentation, mask transparency 150	**82,5%**	**78,5%**

Methods

- Affine transformations include image scale, rotation, skew, and vertical and horizontal shifts, this was implemented in the paper using Keras.
- Elastic transform augmentation consists of adding zero-mean Gaussian noise to each pixel coordinate to make a small pixel displacement that can imitate uncontrolled hand jittering during writing. (They didn't really archieve this effect right as seen above).
- The MNIST





(a) Affine transform augmentation

(b) Elastic transform augmentation

(c) Augmentation with MNIST mask with transparency 255

(d) Augmentation with MNIST mask with transparency 150

MNIST mask extraction – image blending

- The mask is extracted from a randomly selected handwritten character from the MNIST dataset enlarged to the character image size. The mask was then cut into a specified number of equal parts, which were flipped, rotated, and collected in a random order. This procedure provides sufficient randomness to the mask shape.
- The mask is then filled with background pixels and blended into the original image of the character. A Ga ussian blur is applied to the mask after blending to create a more natural appearance. The opaqueness of the mask is a maximum at 255 and invisible at 0, values of 255 and 150 were chosen, the best results were obtained by 255

Transformation

- An affine transformation is the combination of a linear transformation with a translation operation (Rotating, stretching and translating).

Notes

```
num_words = word_bboxes.shape[0]
transcripts = [word for line in transcripts for word in line.split()]
```

- Good solution