Augraphy: An Augmentation Pipeline API for Modern Document Images

Anonymous ICDAR 2023 submission

No Institute Given

Abstract. This paper introduces Augraphy, a Python library for constructing data augmentation pipelines for producing distortions commonly seen in real-world document image datasets. Augraphy stands apart from other data augmentation tools by providing many different strategies to produce augmented versions of clean document images that appear as if they have been altered by standard office operations, such as printing, scanning, and faxing through old or dirty machines, degradation of ink over time, and handwritten markings. This paper discusses the Augraphy tool, and shows how it can be used both as a data augmentation tool for producing diverse training data for tasks such as document denoising, and generating challenging test data for evaluating model robustness on document image modeling tasks.

1 Introduction and Motivation

Daily life in the modern world involves a plethora of tasks requiring the handling of unstructured data. Often, this data is in the form of documents, which may appear very different from when they were originally produced, having been physically altered by printing, scanning, or photocopying processes. Such real-world phenomena may introduce many types of distortions: for instance, folds, wrinkles, or tears in a page can cause color changes and shadows in a scanned document image; low or high printer ink settings may cause some regions of a document to be lighter or darker; and human-made annotations like highlighting or pencil marks can add noise to the page.

Many machine learning operations involving documents are impacted by the presence of noise, while high-level tasks like document classification and information extraction are frequently expected to perform well even on noisily-scanned document images. For instance, the RVL-CDIP document classification corpus [9] consists of scanned document images, many of which have substantial amounts of scanner-induced noise, as does the FUNSD form understanding benchmark [14]. Other intermediate-level tasks like optical character recognition (OCR) and page layout analysis may perform optimally if noise in a document image is minimized [4,24,26]. The lower-level task of document denoising tackles the document noise problem directly, by attempting to remove noise from a document image [3,8,19,22,23]. All of these tasks benefit from copious amounts of training data, and one way of generating large amounts of training data with noise-like artefacts is to use data augmentation.

For this reason we introduce *Augraphy*, an open-source Python-based data augmentation library for generating versions of document images that contain realistic noise

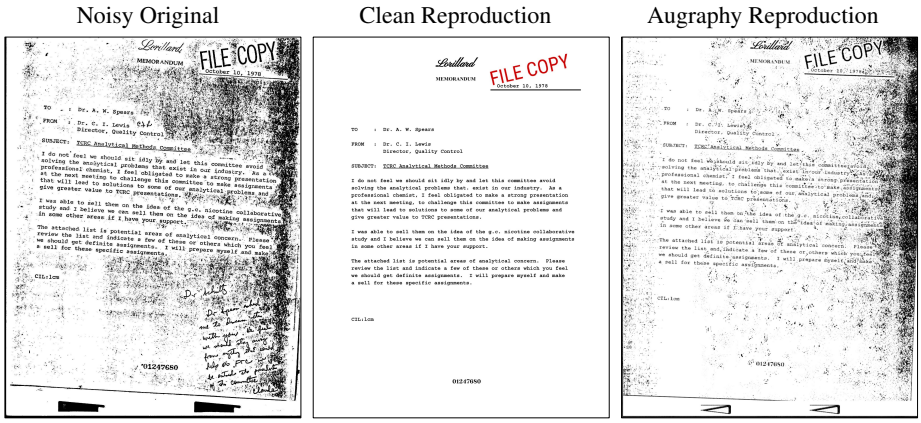

Fig. 1. *Augraphy* can be used to introduce noisy perturbations to document images like the noise seen in (a), which is a real-life sample from RVL-CDIP. We re-created a clean version of (a) in (b). We then create a noisy version of (b) by applying several augmentations to it to produce (c).

artefacts commonly introduced via scanning, photocopying, and other office procedures. *Augraphy* differs from most image data augmentation tools by specifically targeting the types of alterations and degradations seen in document images. *Augraphy* offers *26* individual augmentation methods out-of-the-box across three "phases" of augmentations, and these individual phase augmentations can be composed together along with a "paper factory" step where different paper backgrounds can be added to the augmented image. The resulting document images are realistic, noisy versions of clean documents, as evidenced in Figure 1, where we apply several *Augraphy* augmentations to a clean document image in order to mimic the types of noise seen in a real-world noisy document image from RVL-CDIP. This paper provides an overview of the *Augraphy* library, and demonstrates how it can be used both as a training data augmentation tool and as an effective means for producing data for robustness testing.

Table 1. Comparison of various image-based data augmentation libraries. Number of augmentations is a rough count, and many augmentations in other tools are what Augraphy calls Utilities. Further, many single augmentations in Augraphy — geometric transforms, for example — are represented by multiple classes in other libraries.

Library	Number of Augmentations	Document Centric	Pipeline Based	Python	License
Augmentor [1]	27	✗		✓	MIT
Albumentations [2]	216	✗	✓	✓	MIT
imgaug [16]	168	✗	✓	✓	MIT
Augly [25]	34	✗		✓	MIT
DocCreator [15]	12	✓	✗	✗	LGPL-3.0
Augraphy (ours)	*26*	✓	✓	✓	MIT

2 Related Work

This section discusses prior work related to data augmentation and robustness testing, especially as it relates to document understanding and processing tasks.

2.1 Data Augmentation

A wide variety of data augmentation tools and pipelines exist for machine learning tasks, including natural language processing (e.g., [7,6,30]), audio and speech processing (e.g., [18,20,21]), and computer vision and image processing. In the latter realm, popular tools include *Augly* [25], *Augmentor* [1], *Albumentations* [2], *DocCreator* [15], and *imgaug* [16]. Augmentation strategies from these image-centric libraries are typically general-purpose, and include transformations like rotations, warps, and color modifications. Table 1 compares *Augraphy* with other image augmentation libraries and tools. As can be seen, these other data augmentation libraries do not specifically provide support for imitating the corruptions commonly seen in document analysis corpora.

A notable exception to this is the *DocCreator* image synthesizing tool [15], which is targeted towards creating synthetic images that mimic common corruptions seen in document collections. *DocCreator* differs from *Augraphy* in several crucial ways. The first difference is that *DocCreator*'s augmentations are meant to imitate those seen in historical (e.g., ancient or medieval) documents, while *Augraphy* is intended to replicate noise caused by office room procedures. *DocCreator* was also written in the C++ programming language as a monolithic what-you-see-is-what-you-get tool, and does not have a scripting or API interface to enable use in a broader machine learning pipeline. *Augraphy*, in contrast, is written in Python and can be easily integrated into machine learning model development and evaluation pipelines, alongside other Python packages like PyTorch [**?**].

2.2 Robustness Testing

The introduction of noise-like corruptions and other modifications to image data can be used as a way of estimating and evaluating model robustness. Prior work in this space includes the use of image blurring, contrast and brightness changes, color alterations, partial occlusions, geometric transformations, pixel-level noise (e.g., salt-and-pepper noise, impulse noise, etc.), and compression artefacts (e.g., JPEG) to evaluate image classification and object detection models (e.g., [5,10,11,12,17,28,29]). More specific to the document understanding field, recent prior work has used basic noise-like corruptions to evaluate the robustness of document classifiers trained on RVL-CDIP [27]. Our paper also uses robustness testing as a way to showcase the effectiveness of *Augraphy*, but rather than general image modifications like those described above, we use document-centric modifications to produce human-readable distortions which challenge OCR models.

3 Document Distortion, Theory & Technique

Many approaches exist for adding features to an image, and many types of feature can be generated. The most common types of features added are Gaussian noise, blurring,

geometric transformations like scaling, rotating, translating, and cropping, downsampling, font weighting, and so on.

These types of feature are certainly useful in general image analysis and understanding, but bear little relation to the types of features commonly found in real-world documents.

A sheet of paper out in the world begins its life as wood pulp, bleached, drained, and pressed flat by a long series of rollers. These are cut to size and stacked, then bound in reams and sent out for sale and use. This is the last time the sheet is clean in its useful lifetime, and even at this point, manufacturing defects can lead to variations in the paper, even between two pages in the same ream. At the point of use, these pages are loaded into a printer where they are stamped or dusted in toner and burned with lasers or sprayed with ink. Any of these processes may alter the local texture or global topology of the sheet. The pages may receive handwritten marks at any point before or after printing, and may subsequently be folded, creased, crumpled, flattened, burned, stained, soaked, or generally be subject to any of a million other operations. Any secretary can describe hundreds of different document distortion features; any schoolteacher, thousands.

Augraphy's suite of augmentations was designed to faithfully reproduce this level of complexity in the document lifecycle. Every feature just listed already has a direct implementation either within the library or on the development roadmap, with many more planned.

Some techniques exist for introducing these features into images of documents, including but not limited to the following:

1. Text can be generated independently of the paper texture, and can be overlaid onto the "paper" by a number of blending functions, allowing a variety of paper textures to be used. The NoisyOffice team did this.
2. Similarly, any markup features may be generated and overlaid by the same methods.
3. Documents can be digitized with a commercial scanner, or converted to a continuous analog signal and back with a fax machine.
4. The finished document image can be used as a texture and attached to a 3D mesh, then projected back to 2 dimensions to simulate physical deformation. DocCreator has a function to do this.

Augraphy already has a story for the first three, with the fourth in planning.

4 Augraphy

This section dives into detail about the *Augraphy* library. We first provide a high-level overview of the library, then discuss the various augmentations supported out-of-the-box, and finally discuss details of the structure of the library.

4.1 Overview

Augraphy is a lightweight Python package for applying realistic perturbations to document images. It is registered on the Python Package Index (PyPI) and can be installed simply using `pip install augraphy`.

Augraphy requires only a few commonly-used Python scientific computing or image handling packages, such as NumPy [**?**], and has been tested on Windows, Linux, and Mac computing environments and supports recent major versions of Python 3. Below is a basic out-of-the-box *Augraphy* pipeline demonstrating its usage:

```
import augraphy; import cv2
pipeline = augraphy.default_augraphy_pipeline()
img = cv2.imread("image.png")
data = pipeline.augment(img)
augmented_img = data["output"]
```

Listing 1.1. Augmenting a document image with *Augraphy*.

Modern frameworks for machine learning like *fastai* [13] aim to simplify the data handling requirements, and concordantly, the *Augraphy* development team takes great pains to ensure our library's ease-of-use and compatibility.

Augraphy is designed to be immediately useful with little effort, especially as part of a preprocessing step for training machine learning models, so great care was taken to establish good defaults. The default *Augraphy* pipeline (shown in the code snippet above) makes use of all of the augmentations available in *Augraphy*, with starting parameters selected after manual visual inspection of several thousand images.

Augraphy provides *26* unique augmentations, which may be sequenced into pipeline objects which carry out the image manipulation. Users of the library can define directed acyclic graphs of images and their transformations via the AugraphyPipeline API, representing the passage of a document through real-world alterations.

Augraphy attempts to decompose the lifetime of features accumulating in a document by separating the pipeline into three phases: ink, paper, and post. The ink phase exists to sequence effects which specifically alter the printed ink — like bleedthrough from too much ink on page, extraneous lines or regions from a mechanically faulty printer, or fading over time — and transform them prior to "printing".

The paper phase applies transformations of the underlying paper on which the ink gets printed; here, a `PaperFactory` generator creates a random texture from a set of given texture images, as well as effects like random noise, shadowing, watermarking, and staining. After the ink and paper textures are separately computed, they are merged in the manner of Technique 1 from the previous section, simulating the printing of the document. After "printing", the document enters the post phase, wherein it may undergo alterations that would affect an already-printed document out in the world. Augmentations are available here which simulate the printed page being folded along multiple axes, marked by handwriting or color highlighter, faxed, photocopied, scanned, photographed, burned, stained, and so on. Figure 2 shows the individual phases of an example pipeline combining to produce a noised document image.

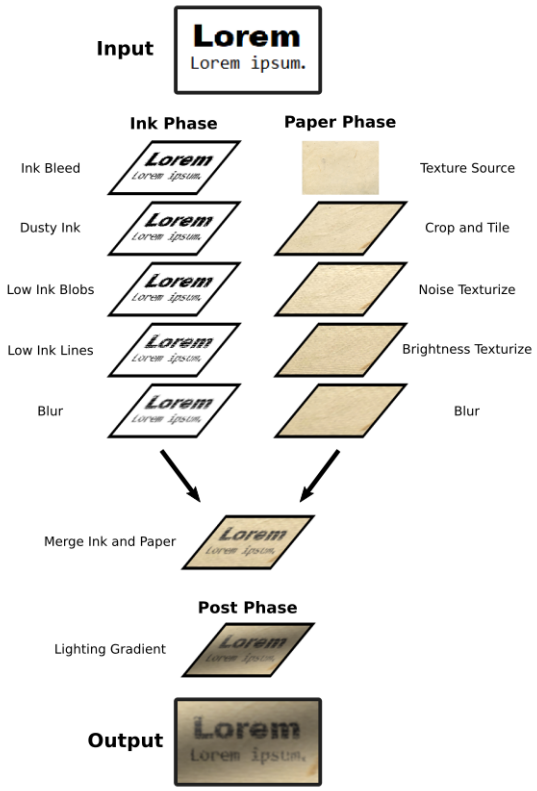

Fig. 2. Visualization of an Augraphy pipeline, showing the composition of several image augmentations together with a specific paper background

4.2 Augraphy Augmentations

Augraphy provides *26* out-of-the-box augmentations. These augmentations are listed in Table 2. As mentioned before, Ink Phase augmentations include those that imitate noisy processes that occur in a document's life cycle when ink is printed on paper. These augmentations include `BleedThrough`, which imitates what happens when ink bleeds through from the opposite side of the page. Another, `LowInkLines`, produces a streaking behavior common to printers running out of ink.

Augmentations provided by the Paper Phase include `BrightnessTexturize`, which introduces random noise in the brightness channel to emulate paper textures, and `Watermark`, which imitates watermarks in a piece of paper. Finally, the Post Phase includes augmentations that imitate noisy-processes that occur after a document has been created. These include `BadPhotoCopy`, which uses added noise to generate an effect of dirty copier, and `BookBinding`, which creates an effect with shadow and curved lines to imitate how a page from a book might appear after capture by a flatbed scanner.

Other general-purpose augmentation libraries already exist for adding basic effects like blur, scaling, and rotation, but *Augraphy* includes these types of augmentations for completeness and utility. Descriptions of each augmentation are available online, along with the motivation for their development. [1]

Table 2. Individual *Augraphy* augmentations for each augmentation phase, in suggested position within the pipeline. Augmentations that work well in more than one phase are listed in the last column.

Ink Phase	Paper Phase	Post Phase	Multiple
BleedThrough	ColorPaper	BadPhotoCopy	BrightnessTexturize
LowInkLines	Watermark	BindingsAndFasteners	DirtyDrum
InkBleed	Gamma	BookBinding	DirtyRollers
Letterpress	LightingGradient	Folding	Dithering
	SubtleNoise	JPEG	Geometric
		Markup	NoiseTexturize
		Faxify	PencilScribbles
		PageBorder	

4.3 The Augraphy Library

Augraphy is a Python-based library, allowing for maximal accessibility for practitioners, and is designed with an object-oriented structure, with concerns divided across a class hierarchy. When composed, augmentations from the library interact to produce complex document image transformations, generating realistic new synthetically-augmented document images.

There are four "main sequence" classes in the *Augraphy* codebase, which together provide the bulk of the library's functionality.

We now discuss each of these:

`Augmentation`. The `Augmentation` class is the most basic class in the project, and essentially exists as a thin wrapper over a probability value in the interval [0,1]. Every augmentation contained in a pipeline has its own chance of being applied during that pipeline's execution.

`AugmentationResult`. After an augmentation is applied, the output of its execution is stored in an `AugmentationResult` object and passed forward through the pipeline. These objects also record a full copy of the augmentation runtime data, as well as any metadata that might be relevant for debugging or other advanced use.

`AugmentationSequence`. A list of `Augmentations` — together with the intent to apply those Augmentations in sequence — determines an `AugmentationSequence`, which is itself both an `Augmentation` and callable. In practice, these are the model for the pipeline phases discussed previously; they are essentially lists of `Augmentation` constructor calls which produce callable `Augmentation` objects of the various flavors explored in `AugmentationSequences` are applied to the image during each of the

[1] `https://augraphy.readthedocs.io/en/latest/`

`AugmentationPipeline` phases, and in each case yield the image, transformed by some of the Augmentations in the sequence.

`AugmentationPipeline`. The bulk of the heavy lifting in *Augraphy* resides in the Augmentation pipeline, which is our abstraction over one or more events in a physical document's life.

Consider the following sequence of events:

> The initial printing of the document when ink adhered to the paper material, or several weeks later when the document was adhered to a public board, annotated, defaced, and torn away from its securing staples. Fifty years later, our protagonist page resurfaces in the library archive during routine preservation-scanning efforts. Conservationists use delicate tools to gently position and record an image of the document, storing this in a public repository.

An `AugmentationPipeline` can model this entire sequence of events, or any individual event within.

Realistically reproducing effects in document images requires rethinking how those effects are produced in the real world. Many issues, like the various forms of misprint, only affect text and images on the page. Others, like a coffee spill, change properties of the paper itself. Further still, there are transformations like physical deformations which alter the geometry and topology of both the page material and the graphical artifacts on it. Effectively capturing processes like these in reproducible augmentations means separating our model of a document augmentation pipeline into ink, paper, and post-processing layers, each containing some augmentations that modify the document image as it passes through. Producing realistically noisy document images can now be reduced to the definition and application of one or more Augraphy pipelines to some clean document images.

The value added by the `AugraphyPipeline` class over a bare list of functions mapped over an image is principally in the collection of runtime metadata: the output of an `AugraphyPipeline` application is a Python dictionary which contains not only the final image, but copies of every intermediate image, as well as information about the object constructors and their parameters that were used for each augmentation. This allows for easy inspection and fine-tuning of the pipeline definition to achieve outputs with desired features, facilitating (re)production of documents as in Figure 1 1.

There are also two classes that provide additional functionality in order to round out the *Augraphy* base library:

`OneOf`. To model the possibility that a document image has undergone one and only one of a collection of augmentations, we use `OneOf`, which simply selects one of those augmentations from the given list, and uses this to modify the image.

`PaperFactory`. We often print on multiple sizes and kinds of paper, and out in the world we certainly *encounter* such diverse documents. We introduce this variation into the `AugmentationPipeline` by including `PaperFactory` in the `paper` phase of the pipeline. This augmentation checks a local directory for images of paper to crop, scale, and use as a background for the document image. The pipeline contains edge detection logic for lifting only text and other foreground objects from a clean image, greatly simplifying the "printing" onto another "sheet", and capturing in a reproducible

way the construction method used to generate the NoisyOffice database. Taken together, `PaperFactory` makes it trivial to re-print a document onto other surfaces, like hemp paper, cardboard, or wood.

Interoperability and flexibility are core requirements of any data augmentation library, so *Augraphy* includes several utility classes designed to improve developer experience:

`ComposePipelines`. This class provides a means of composing two pipelines into one, allowing for the construction of complex pipeline algebras.

`Foreign`. This class can be used to wrap augmentations from external projects like *Albumentations* and *imgaug*.

`ImageOverlay`. This class uses various blending algorithms to fuse foreground and background images together, which is useful for simulating "printing".

Augraphy pipelines, then, are constructed from three sequences of augmentations, to be applied one after the other in each of those phases.

After the ink and paper textures are computed separately, they are merged in the manner of Technique 1 from the previous section, simulating the printing of the document. After "printing", the document enters the post phase, wherein it may undergo alterations that would affect an already-printed document out in the world. Augmentations are available here which simulate the printed page being folded along multiple axes, marked by handwriting or highlighter, faxed, photocopied, scanned, photographed, burned, stained, and so on. Figure 2 shows the individual phases of an example pipeline combining to produce a noised document image.

5 Deep Learning with Augraphy

Augraphy aims to facilitate rapid dataset creation, advancing the state of the art for document image analysis tasks. This section describes a brief experiment using Augraphy to augment the NoisyOffice set, producing a corpus that is used to train a denoising convolutional neural network which outperforms an identically-structured model trained on only the provided NoisyOffice data. We perform the same experiment twice, with different model designs. We continue to return to the NoisyOffice database when testing our model training pipelines and new architectures, and felt it an appropriate jumping-off point for analyzing Augraphy.

5.1 Model Architecture

To evaluate Augraphy, we trained two simple models: the first was a UNet convolutional neural network, taken directly from a submission to the NoisyOffice Kaggle competition [2]. We selected this one for its simplicity and the clarity of its exposition, and use it with few changes. The second model is an off-the-shelf Nonlinear Activation-Free Network (NAFNet), [**?**]; we again made only minor changes to the model's training hyperparameters, increasing the batch size to fit our data.

[2] `https://www.kaggle.com/code/michalbrezk/denoise-images-using-autoencoders-tf-keras/notebook`

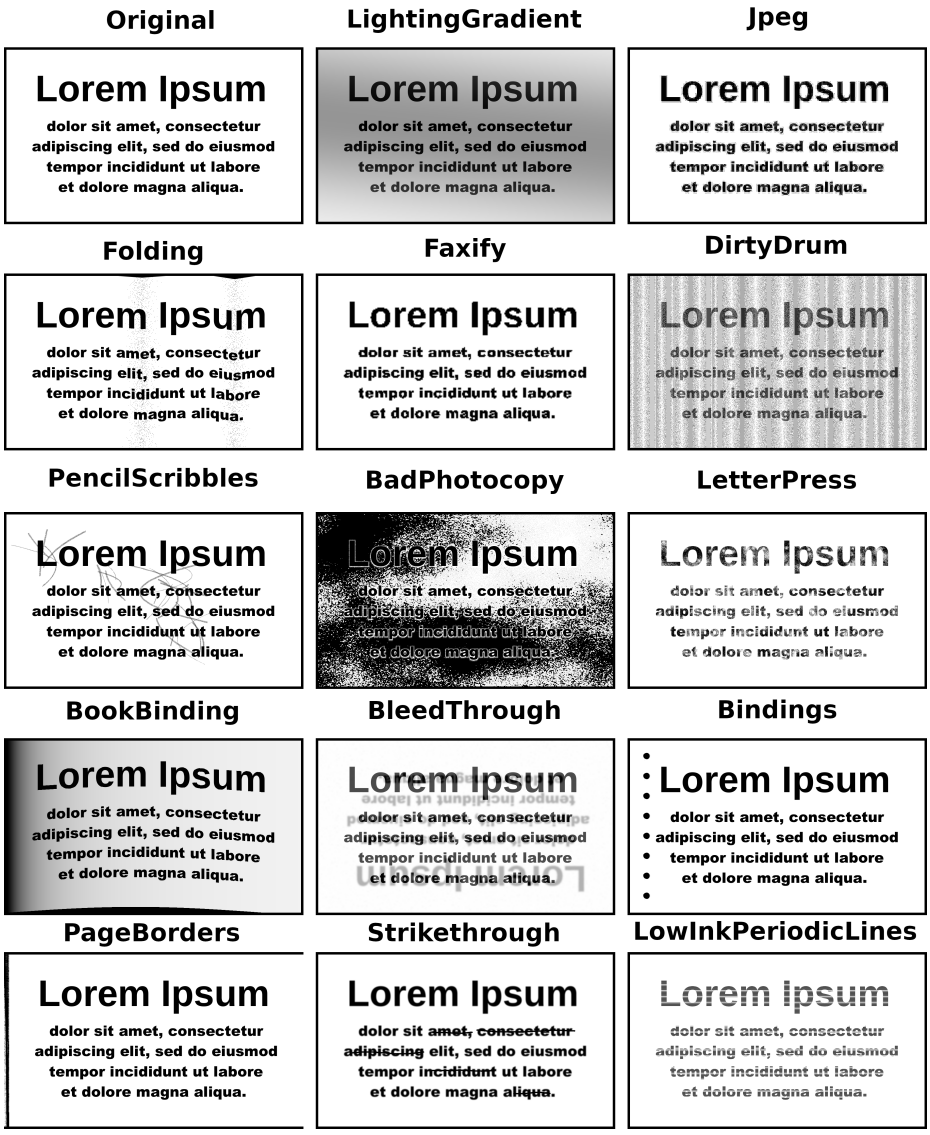

Fig. 3. Examples of some Augraphy augmentations.

5.2 Data Generation

Despite recent techniques [Training Vision Transformers with Only 2040 Images, Vision Transformer for Small-Size Datasets, Training a Vision Transformer from scratch in less than 24 hours with 1 GPU] for reducing the volume of input data required to train models, data remains king; feeding a model more data during training can help ensure better latent representations of more features, improving robustness of the model and increasing its ability to generalize.

The NoisyOffice data provided by Kaggle contains 144 ground truth images, 144 training images, and 72 validation images. For the Augraphy model, we produced a dataset 10x larger, by duplicating each of the ground truth images, then running 10 Augraphy pipelines against each copy. Doing this was trivial; Augraphy's value lies in its ease of use in producing large training sets.

The NoisyOffice dataset contains folded sheets, wrinkled sheets, coffee stains, and footprint noise. The features given by the wrinkle and fold distortions can be mimicked by overlaying the foreground text on wrinkled and folded paper textures, as the NoisyOffice team did, and the features created by the stains and footprints can be mimicked by introducing dark regions and thin lines. With Augraphy, we expected that we could use the BadPhotoCopy augmentation to produce the dark regions and a combination of the strikethrough behavior from the Markup augmentation and the smooth curve shading behavior of the PencilScribbles augmentation to add the last feature to the ground-truth data. The PaperFactory augmentation makes the paper texture overlay trivial and repeatable. In the end, we executed the following pipeline:

```
ink_phase = []
paper_phase = [PaperFactory(p=0.5)]
post_phase = [
  BadPhotoCopy(p=0.5),
  PencilScribbles(p=0.5),
  Markup(markup_color=(0,0,0), p=0.5)
]
AugraphyPipeline(ink_phase, paper_phase, post_phase)
```

Listing 1.2. Pipeline used for test data generation.

In each of the augmentations created above, the probability of applying to the image passing through the pipeline was set to 50%, and the strikethrough behavior (the default) for the Markup augmentation was set to strike out words with only black lines.

The PaperFactory augmentation reads and randomly crops image textures from a local directory; to this we added two [3], [4] public domain images of wrinkled paper found with Bing.

[3] https://p2.piqsels.com/preview/642/889/110/paper-crease-creased-texture.jpg

[4] https://blog.miklavcic.si/wp-content/uploads/2011/11/white_paper_1.png

5.3 Training Regime

We fit the models described in the previous section to both the NoisyOffice corpus and a derivative work generated with Augraphy applied to the NoisyOffice ground truth images.

Training of the UNets proceeded for 600 epochs or until the model began to overfit, with an overfit patience of 30 epochs. The NoisyOffice model finished training after *426*epochs, while the Augraphy model trained for the full 600 epochs.

Both models were trained with mean squared error as the loss function, using the Adam optimizer, and evaluated with the mean average error metric.

The NAFNet instances trained for 100 epochs, using the default settings provided from the GitHub version. All code used in these experiments is available in the Augraphy-Paper GitHub [5].

5.4 Results

Sample predictions from each model on the validation task are presented in Table 1. A new dataset, ShabbyPagesReal, was produced as part of the larger ShabbyPages corpus [**?**]; this data was manually produced by applying sequences of physical operations to real paper. ShabbyPagesReal is fully out-of-distribution for the models in our experiments, and has a much higher degree of diversity than the NoisyOffice set, providing good conditions for evaluating Augraphy's effect. As expected, the NoisyOffice models perform admirably, fitting extremely well to the low-diversity data of the NoisyOffice set, but they do struggle to fully remove the coffee stain feature, leaving some residue. The SSIM score indicates that the Augraphy models generalize much better, though they do overcompensate for the `BadPhotoCopy` behavior on text, by increasing the line thickness in the predicted text, resulting in a bold font.

To compare the models' performance on the validation task, we considered the following metrics:

1. Root mean square error (RMSE)
2. Structural similarity index (SSIM)
3. Peak signal-to-noise ratio (PSNR)

Over the last 5 epochs of training, the performance of the models on average loss, average mean-average-error (MAE), average validation loss, and average validation MAE were recorded. The models predicted cleaned versions of the validation images (Figure 4), which were then compared to the groundtruth versions according to each metric. The average over all such results obtained during validation was taken. These metrics are displayed in Table 1.

The Augraphy model outperforms the PSNR score of the NoisyOffice model on the validation task by half a percent, and has a lower mean average error both in test and validation during training, but underperforms by 0.8% on structural similarity and 2.5% on RMSE in validation, with a higher average test and validation loss. These numbers are consistent with the visual prediction results displayed in Figure 1: the

[5] `https://github.com/sparkfish/augraphy-paper`

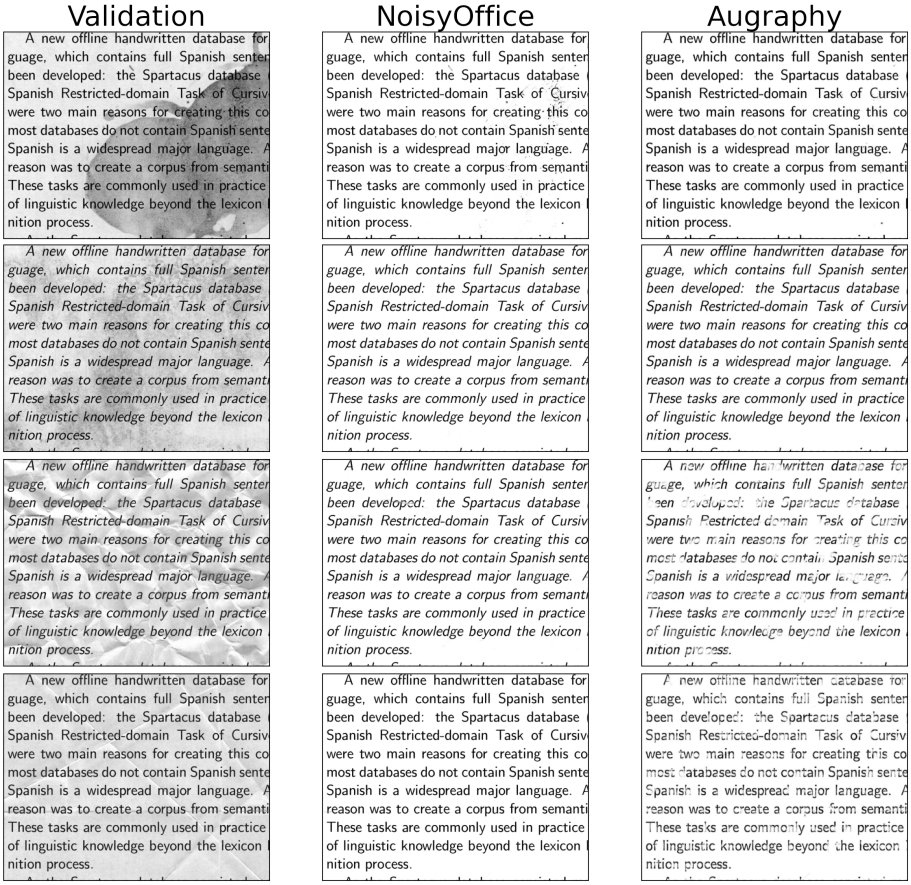

Fig. 4. Validation images (left), with the images predicted by the NoisyOffice (center) and Augraphy (right) models.

Table 3. Model training statistics and performance on NoisyOffice validation task

Metric	PSNR	SSIM	RMSE
NoisyOffice_UNet	63.09	0.87	0.19
Augraphy_UNet	63.35	0.87	0.18
NoisyOffice_NAFNet			Augraphy_NAFNet
		height	

Augraphy model predicts fewer pixels in the text (RMSE lower by 2.5%), but removes more noise and with a higher degree of fidelity than the NoisyOffice model (PSNR higher by 0.2576466946). The training metrics collected indicate that the Augraphy model has lower variance so is more precise than the NoisyOffice model, but exhibits higher loss and thus less accuracy in its predictions.

6 Robustness Testing

We use Augraphy to add noise to an image of text with known groundtruth. The Tesseract [**?**] pre-trained OCR model's performance on the clean image is compared to the OCR result on the Augraphy-noised image, then to the true groundtruth string, using the Levenshtein distance. These measures are averaged over 1000 trials, with different Augraphy effects, to give some indication of Augraphy's effect on producing human-readable document images for challenging OCR tasks.

We use the widely-used open-source Tesseract 2 engine and first compiled 15 ground-truth, noise-free document images from a new corpus of born-digital documents, whose ground-truth strings are known. We then used Tesseract to generate OCR predictions on these noise-free documents, as a baseline for comparison. We considered these OCR predictions as the ground-truth labels for each document. Next, we generated noisy versions of the 15 documents by running them through an Augraphy pipeline, and again used Tesseract to generate OCR predictions on these noisy documents. We compared the word accuracy rate on the noisy OCR results versus the ground-truth noise-free OCR results, and found that the noisy OCR results were on average 52% less accurate, with a range of up to 84%. This example use-case demonstrates the effectiveness of using Augraphy to create challenging test data for evaluating OCR systems.

7 Conclusion and Future Work

We presented Augraphy, a framework for generating realistic synthetically-augmented datasets of document images. Other available image augmentation tools were examined and found to lack features needed for our purposes, motivating the creation of this library specifically targeting the types of alterations and degradations seen in document images.

We also described the process for creating new document image datasets that contain synthetic real-world noise using Augraphy, then compared results obtained by training a convolutional U-Net and a NAFNet on these datasets.

Future work on the Augraphy library will focus on adding new types of augmentations, increasing performance to enable faster creation of larger datasets on more common hardware, providing more scale-invariant support so that augmentations perform well at all document image resolutions and responding to community-initiated feature requests.

Augraphy is licensed under the MIT open source license, and readers are invited to share feedback and participate in its development on GitHub.

References

1. Bloice, M.D., Roth, P.M., Holzinger, A.: Biomedical image augmentation using Augmentor. Bioinformatics **35**(21), 4522–4524 (04 2019). https://doi.org/10.1093/bioinformatics/btz259, https://doi.org/10.1093/bioinformatics/btz259
2. Buslaev, A., Iglovikov, V.I., Khvedchenya, E., Parinov, A., Druzhinin, M., Kalinin, A.A.: Albumentations: Fast and flexible image augmentations. Information **11**(2) (2020). https://doi.org/10.3390/info11020125, https://www.mdpi.com/2078-2489/11/2/125
3. Castro-Bleda, M.J., España-Boquera, S., Pastor-Pellicer, J., Zamora-Martínez, F.: The Noisy-Office Database: A Corpus to Train Supervised Machine Learning Filters for Image Processing. The Computer Journal **63**(11), 1658–1667 (11 2019). https://doi.org/10.1093/comjnl/bxz098, https://doi.org/10.1093/comjnl/bxz098
4. Cheriet, M., Kharma, N., Liu, C.L., Suen, C.Y.: Character Recognition Systems: A Guide for Students and Practitioners. Wiley (2007)
5. Dodge, S., Karam, L.: Understanding how image quality affects deep neural networks. In: 2016 Eighth International Conference on Quality of Multimedia Experience (QoMEX). pp. 1–6 (2016). https://doi.org/10.1109/QoMEX.2016.7498955
6. Fadaee, M., Bisazza, A., Monz, C.: Data augmentation for low-resource neural machine translation. In: Proceedings of the 55th Annual Meeting of the Association for Computational Linguistics (Volume 2: Short Papers). pp. 567–573. Association for Computational Linguistics, Vancouver, Canada (Jul 2017). https://doi.org/10.18653/v1/P17-2090, https://aclanthology.org/P17-2090
7. Feng, S.Y., Gangal, V., Wei, J., Chandar, S., Vosoughi, S., Mitamura, T., Hovy, E.: A survey of data augmentation approaches for NLP. In: Findings of the Association for Computational Linguistics: ACL-IJCNLP 2021. pp. 968–988. Association for Computational Linguistics, Online (Aug 2021). https://doi.org/10.18653/v1/2021.findings-acl.84, https://aclanthology.org/2021.findings-acl.84
8. Gangeh, M.J., Plata, M., Motahari Nezhad, H.R., Duffy, N.P.: End-to-end unsupervised document image blind denoising. In: 2021 IEEE/CVF International Conference on Computer Vision (ICCV). pp. 7868–7877 (2021). https://doi.org/10.1109/ICCV48922.2021.00779
9. Harley, A.W., Ufkes, A., Derpanis, K.G.: Evaluation of deep convolutional nets for document image classification and retrieval. In: International Conference on Document Analysis and Recognition (ICDAR) (2015)
10. Hendrycks, D., Dietterich, T.: Benchmarking neural network robustness to common corruptions and perturbations. In: International Conference on Learning Representations (ICLR) (2019)
11. Homeyer, A., Geißler, C., Schwen, L.O., Zakrzewski, F., Evans, T., Strohmenger, K., Westphal, M., Bülow, D., Kargl, M., Karjauv, A., Munné-Bertran, I., Retzlaff, C.O., Romero-López, A., Sołtysiński, T., Plass, M., Carvalho, R., Steinbach, P., Lan, Y.C., Bouteldja, N., Haber, D., Rojas-Carulla, M., Sadr, A.V., Kraft, M., Krüger, D., Tick, R., Lang, T., Boor, P., Müller, H., Hufnagl, P., Zerbe, N.: Recommendations on test datasets for evaluating ai solutions in pathology. arXiv preprint arXiv:2204.14226 (2022), https://arxiv.org/pdf/2204.14226.pdf
12. Hosseini, H., Xiao, B., Poovendran, R.: Google's cloud vision API is not robust to noise. arXiv preprint arXiv:1704:05051 (2017)
13. Howard, J., Sylvain, G.: Fastai: A layered api for deep learning. Information **11**(2) (2020), https://arxiv.org/pdf/2002.04688.pdf

14. Jaume, G., Ekenel, H.K., Thiran, J.P.: Funsd: A dataset for form understanding in noisy scanned documents. In: Accepted to ICDAR-OST (2019)
15. Journet, N., Visani, M., Mansencal, B., Van-Cuong, K., Billy, A.: Doccreator: A new software for creating synthetic ground-truthed document images. Journal of Imaging **3**(4) (2017). https://doi.org/10.3390/jimaging3040062, https://www.mdpi.com/2313-433X/3/4/62
16. Jung, A.B., Wada, K., Crall, J., Tanaka, S., Graving, J., Reinders, C., Yadav, S., Banerjee, J., Vecsei, G., Kraft, A., Rui, Z., Borovec, J., Vallentin, C., Zhydenko, S., Pfeiffer, K., Cook, B., Fernández, I., De Rainville, F.M., Weng, C.H., Ayala-Acevedo, A., Meudec, R., Laporte, M., et al.: imgaug. https://github.com/aleju/imgaug (2020), online; accessed 01-Feb-2020
17. Karahan, S., Kilinc Yildirum, M., Kirtac, K., Rende, F.S., Butun, G., Ekenel, H.K.: How image degradations affect deep cnn-based face recognition? In: 2016 International Conference of the Biometrics Special Interest Group (BIOSIG). pp. 1–5 (2016). https://doi.org/10.1109/BIOSIG.2016.7736924
18. Ko, T., Peddinti, V., Povey, D., Khudanpur, S.: Audio augmentation for speech recognition. In: Proc. Interspeech 2015. pp. 3586–3589 (2015). https://doi.org/10.21437/Interspeech.2015-711
19. Kulkarni, M., Kakad, S., Mehra, R., Mehta, B.: Denoising documents using image processing for digital restoration. In: Swain, D., Pattnaik, P.K., Gupta, P.K. (eds.) Machine Learning and Information Processing. pp. 287–295. Springer Singapore, Singapore (2020)
20. Maguolo, G., Paci, M., Nanni, L., Bonan, L.: Audiogmenter: a matlab toolbox for audio data augmentation. Applied Computing and Informatics (2021)
21. McFee, B., Humphrey, E., Bello, J.: A software framework for musical data augmentation. In: Muller, M., Wiering, F. (eds.) Proceedings of the 16th International Society for Music Information Retrieval Conference, ISMIR 2015. pp. 248–254. Proceedings of the 16th International Society for Music Information Retrieval Conference, ISMIR 2015, International Society for Music Information Retrieval (2015)
22. Mohamed, S.S.A., Rashwan, M.A.A., Abdou, S.M., Al-Barhamtoshy, H.M.: Patch-based document denoising. In: 2018 International Japan-Africa Conference on Electronics, Communications and Computations (JAC-ECC) (2018)
23. Mustafa, W.A., Kader, M.M.M.A.: Binarization of document image using optimum threshold modification. Journal of Physics: Conference Series **1019**, 012022 (jun 2018). https://doi.org/10.1088/1742-6596/1019/1/012022, https://doi.org/10.1088/1742-6596/1019/1/012022
24. O'Gorman, L., Kasturi, R.: Document Image Analysis. IEEE Computer Society (1997)
25. Papakipos, Z., Bitton, J.: AugLy: Data augmentations for robustness. arXiv preprint arXiv:2201:06494 (2022)
26. Rotman, D., Azulai, O., Shapira, I., Burshtein, Y., Barzelay, U.: Detection masking for improved OCR on noisy documents. arXiv preprint arXiv:2205.08257 (2022)
27. Saifullah, Siddiqui, S.A., Agne, S., Dengel, A., Ahmed, S.: Are deep models robust against real distortions? a case study on document image classification. In: Proceedings of the 26th International Conference on Pattern Recognition (ICPR) (2022), https://www.computer.org/csdl/proceedings-article/icpr/2022/09956167/1IHoLM9J3gI
28. Schömig-Markiefka, B., Pryalukhin, A., Hulla, W., Bychkov, A., Fukuoka, J., Madabhushi, A., Achter, V., Nieroda, L., Büttner, R., Quaas, A., Tolkach, Y.: Quality control stress test for deep learning-based diagnostic model in digital pathology. Modern pathology : an official journal of the United States and Canadian Academy of Pathology, Inc **34**(12), 2098L = https://europepmc.org/articles/PMC8592835 (December 2021). https://doi.org/10.1038/s41379-021-00859-x

29. Vasiljevic, I., Chakrabarti, A., Shakhnarovich, G.: Examining the impact of blur on recognition by convolutional networks. arXiv preprint arXiv:1611.05760 (2016)
30. Wei, J., Zou, K.: EDA: Easy data augmentation techniques for boosting performance on text classification tasks. In: Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing (EMNLP-IJCNLP). pp. 6382–6388. Association for Computational Linguistics, Hong Kong, China (Nov 2019). https://doi.org/10.18653/v1/D19-1670, https://aclanthology.org/D19-1670