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 common perturbations seen in real-world document image datasets. Augraphy stands apart from other data augmentation tools by providing many different augmentation strategies to produce augmented versions of clean document images that appear as if they have been distorted from 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 (1) producing diverse training data for tasks such as document de-noising, and (2) generating challenging test data for evaluating model robustness on document image modeling tasks.

1 Introduction and Motivation

The modern world provides a plethora of tasks that require the need for automated and intelligent solutions for handling unstructured data. Often, this data is in the form of documents, and these documents may appear noisy, especially if they have been captured from the physical world via 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 annotations like highlighting or pencil marks can add noise to the page.

Many tasks involving machine learning are impacted by document noise. High-level tasks like document classification and information extraction must often be able to perform 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 [4,24,27]. Further, the lower-level task of document de-noising tackles the document noise problem more directly by attempting to remove noise from a document image [3,8,19,22,23]. Such 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.

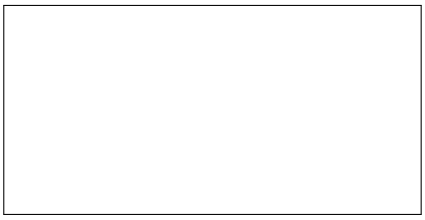

Fig. 1. Placeholder for compelling figure to help the reader understand what Augraphy is about.

For this reason we introduce *Augraphy*, an open-source Python-based data augmentation library for generating versions of document images that contain realistic noise artefacts commonly introduced via scanning, photocopying, and other office room procedures. *Augraphy* differs from most image data augmentation tools by specifically targeting the types of alterations and degradations seen in document images. *Augraphy* offers n individual augmentation methods out-of-the-box across three "phases" of augmentations, and these individual phase augmentations can be composed together along with a 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 X and in Figure Y. 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.

Library Name	Num. Augmentations	Document-Centric	Pipeline-Based	Python	License
Augmentor [1]		✗		✓	MIT
Albumentations [2]		✗	✓	✓	MIT
imgaug [16]		✗	✓	✓	MIT
Augly [25]		✗		✓	MIT
Pytorch [26]		✗	✓	✓	BSD-style
DocCreator [15]		✓	✗	✗	LGPL-3.0
Augraphy (ours)	n	✓	✓	✓	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 ranging from natural language processing (e.g., [7,6,31]), audio and speech processing (e.g., [18,20,21]), and computer vision and image processing. In the image realm of image processing and computer vision, data augmentation tools and pipelines include *Augly* [25], *Augmentor* [1], *Albumentations* [2], *DocCreator* [15], Pytorch [26], and *imgaug* [16]. Augmentation strategies from these image-centric libraries are typically general purpose, and include image transformations like rotations, warps, and color modifications. Table 1 compares *Augraphy* with other image-based data 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 ways, however. 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 meant to replicate noise caused by noisy office room procedures. *DocCreator* is also written in the C++ programming language and is a monolithic what-you-see-is-what-you-get (WYSIWYG) 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, and can easily be used alongside other Python packages.

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., [5,10,11,12,17,29,30]). 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 [28]. 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.

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 has either a direct implementation already within the library or on the development roadmap, with 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

We were unable to find any tools for producing training images of modern documents, in a way suitable for automated use. The Augraphy library is our attempt at filling this gap.

The library contains 18 unique transformations — augmentations, in Augraphy's parlance — which may be sequenced into a pipeline object which carries 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 is concerned with transformations of the underlying paper on which the ink gets printed; here we find the PaperFactory generator for creating 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 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 1 shows the individual phases of an example pipeline combining to produce a noised document image.

5 The Augraphy API

The library was developed in Python, to allow maximal accessibility for practitioners, and was designed with an object-oriented structure, with concerns divided across a class hierarchy. When composed, different parts of the library interact to produce complex sequences of document image transformations, generating new synthetically-augmented datasets.

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. We wanted the library 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 makes use of all of the augmentations available in Augraphy, with starting parameters selected after manual visual inspection of several thousand images. Adding a diverse array of realistic features to documents can be done with the following five-line incantation:

```
from augraphy import *
pipeline = default_augraphy_pipeline()
img = cv2.imread("image.png")
```

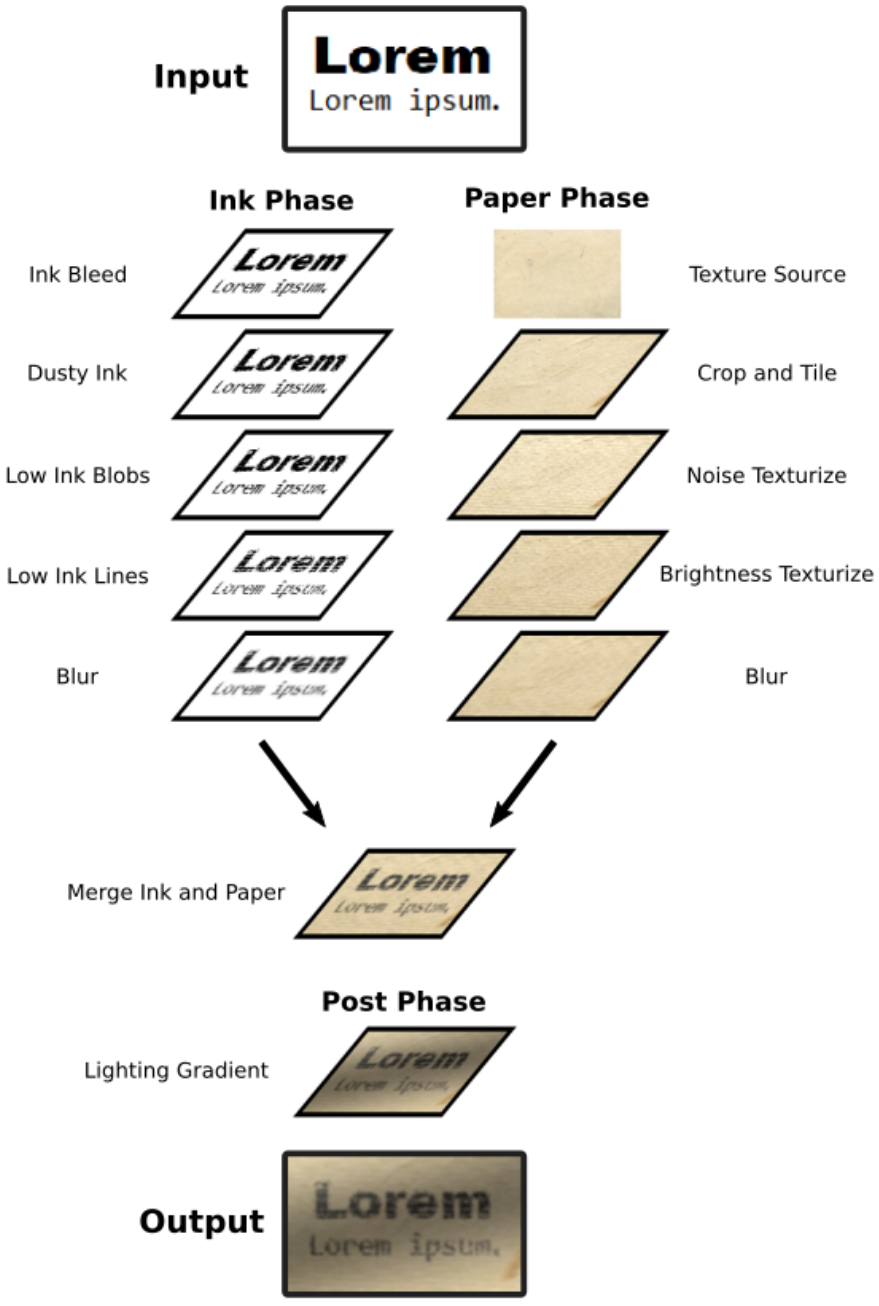

Fig. 2. Visualization of an Augraphy pipeline.

```
data = pipeline.augment(img)
augmented_img = data["output"]
```

5.1 Base

`augraphy.base` is the core of the library, containing the central class for augmentations, another class for sequences of augmentations, and the pipeline class itself. The base classes exist to coordinate the compilation of discrete transformations into a larger pipeline, and to facilitate the operation and maintenance of these.

There are four "main sequence" classes in `augraphy.base`, which together provide the bulk of the library's functionality. Two additional classes, `OneOf` and `PaperFactory`, provide additional variation to pipelines in ways explored soon.

Augmentation Augmentation 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 object is constructed with a probability of that augmentation being applied during the execution of a pipeline containing it.

This class also contains a method which interacts with the probability data, dynamically generating a second floating point probability value and using this to decide whether the augmentation should be applied at runtime.

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 an exact copy of the augmentation object that was instantiated and applied, 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 `AugmentationPipeline` phases, and in each case yield the image, transformed by some of the Augmentations in the sequence.

AugmentationPipeline The bulk of the innovation in Augraphy resides in the Augmentation pipeline, which is an abstraction over one or more events in a physical document's life. Events in this case could be 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 meant 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. Augraphy pipelines, then, are constructed from three sequences of augmentations, to be applied one after the other in each of those phases. After transformations occur to the ink and to the paper separately, the pipeline fuses them together, simulating the printing of that ink on that paper. Changes to the document after that point happen within the post layer, where we might find effects like crumpling up the page and smoothing it out again, or using a pen to underline a section of text.

Use of the Augraphy library to produce a dataset for model training — will be covered in some detail later — boils down to the definition and application of one or more pipelines to one or more images.

The value added by the `AugrapyPipeline` class over a bare list of functions mapped over an image is principally in the collection of 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.

OneOf Real-world processes generally allow for objects to exist in multiple states over time. 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 a 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 effectively possible to re-print a document onto other surfaces, like hemp paper, cardboard, or wood.

5.2 Augmentations

The `base` modules provide scaffolding for working with general transformations on document images, called `augmentations`. We have already included many such transforms in the standard library, and have also provided utilities for easily building new augmentations and integrating them into the Augraphy workflow.

Augraphy was designed to replicate effects seen in real-world documents in training data for image models, so each augmentation represents a feature that will appear in the final dataset. Augmentations come in many flavors and types, varying not only in their visual effect but also in their software design. Other libraries already exist for adding basic effects like blur, scaling, and rotation; Augraphy includes augmentations for these as well. The more interesting effects come from a combination of primitive effects, and mimic document types out in the world.

Figures 2 and 3 contain representative images of each augmentation applied to this very document. Descriptions of and motivation for each augmentation are available on GitHub [1].

5.3 Utilities

Interoperability and flexibility are core requirements of any data augmentation library, so Augraphy includes several utility classes designed to improve developer experience. Chief among these are `ComposePipelines`, `Foreign`, and `ImageOverlay`, which respectively

1. provide a means of composing two pipelines into one, allowing for the construction of complex pipeline algebras,
2. wrap augmentations from projects like `Albumentations` [2] and `imgaug` [16], and
3. uses various blending algorithms to fuse foreground and background images together, useful for simulating "printing".

[1] `https://github.com/sparkfish/augraphy`

Fig. 3. Examples of each augmentation.

6 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 continue to return to this database when testing our model training pipelines and new architectures, and felt it an appropriate jumping-off point for Augraphy analysis.

Fig. 4. Examples of each augmentation.

6.1 Model Architecture

To evaluate Augraphy, we trained a U-net convolutional neural network, built with the Keras library. This network achieved a high score on the NoisyOffice Kaggle competition [2]; we selected this one for its simplicity and the clarity of its exposition, and use it with few changes.

This low-layer model contains two layers of a convolution, each followed by a rectified linear unit activation function, then a batch normalization layer as the encoding step. After the encoding step, max pooling is applied, followed by dropout, to improve translation invariance of feature encoding and to avoid overfitting respectively. The decoding step closely mirrors the encoding step,

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

and contains two layers of ReLU-activated convolutions followed by a batch normalization layer, but with the convolution dimensions reversed; in this case the model is "unpacking" higher-dimensional features from its low-dimensional latent representation. After the decoding step, we perform 2-dimensional upsampling, compensating for the 2D max pooling applied earlier. Finally, the output of previous layers is convolved with a 3x3 kernel, while retaining the same image dimensions. This final convolution uses the sigmoid function as its activation function. All convolution steps use a 3x3 kernel and pad edges with zeroes, the max pooling and upsampling steps both use 2x2 kernels, and a 50% unit dropout rate was used.

6.2 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)
```

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 on Bing.

6.3 Training Regime

We fit the model architecture 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 proceeded for 600 epochs or until the model began to overfit, with an overfit patience of 30 epochs. The NoisyOffice model finished training after 416 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.

6.4 Results

Sample predictions from each model on the validation task are presented in Figure 1. As expected, the NoisyOffice model performs admirably, but does struggle to fully remove the coffee stain feature, leaving some residue. The Augraphy model clearly outperforms the NoisyOffice model at stain removal, but does not generalize well to the folding and wrinkling noise; this was expected, since the Augraphy training data did not include fold or wrinkle features. Further, the Augraphy model overcompensates 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)

[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`

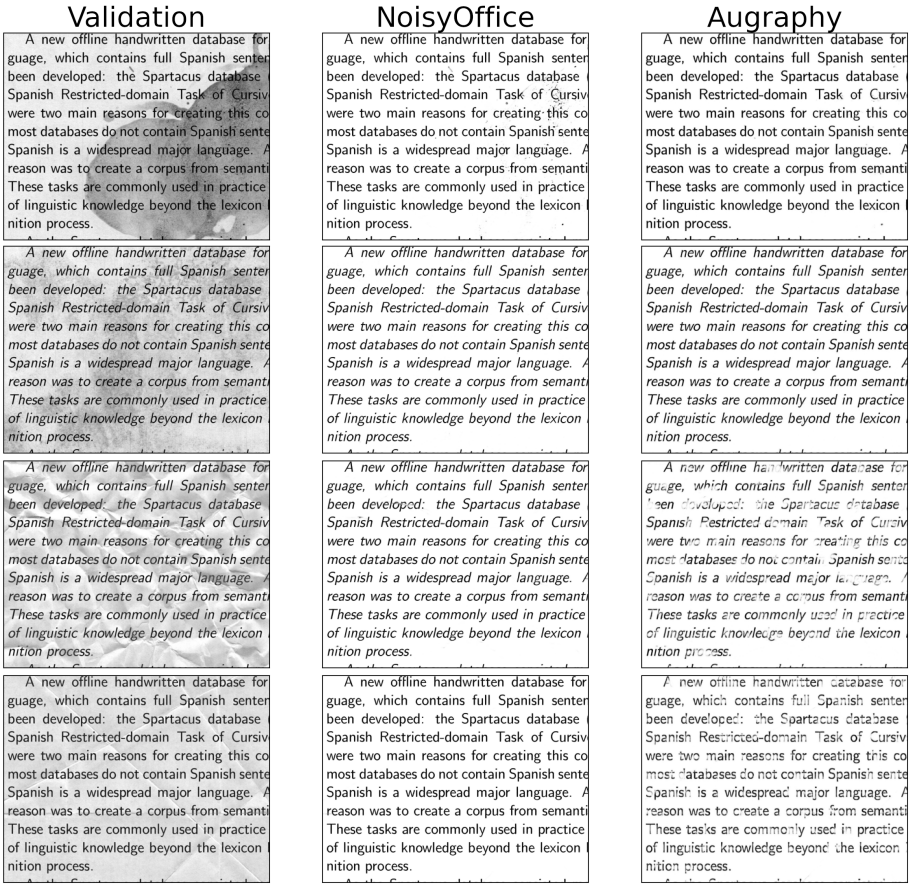

Fig. 5. Validation images (left), with the images predicted by the NoisyOffice (center) and Augraphy (right) models.

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 Augraphy model predicts fewer pixels in the text (RMSE lower by 2.5%), but removes more noise and with a higher degree of

Table 2. Model training statistics and performance on NoisyOffice validation task

Metric	NoisyOffice	Augraphy
PSNR	63.09152246942623	63.3491691640664
SSIM	0.8726093605602885	0.8662918306486613
RMSE	0.1878745854176291	0.18310202817380003
Test Loss	0.0011	0.0012
Test MAE	0.0150	0.0129
Validation Loss	7.5551e-04	8.22202e-04
Validation MAE	0.01236	0.01076
Training Time	190ms/step	218ms/step

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.

7 Future Work

This section describes new directions for research and development.

7.1 Tuning the Pipeline

The pipeline used to generate training data for the Augraphy model was extremely naive, and only contained four augmentations, most with default parameters, demonstrating Augraphy's high degree of both utility and ease-of-use. However, much of the data used in training contained an excessive amount of added noise, making the image unreadable to the human eye, and too noisy to recover text features from. Producing the most accurate model with Augraphy requires careful fine-tuning of the augmentation input parameters to generate training images closer to the validation set.

7.2 Additional Techniques

For brevity, this article only includes experimental results for one type of model. We plan to evaluate denoisers built with other architectures, particularly transformers, diffusion and generative adversarial networks, and ensembles of these. During the validation task, the naive Augraphy model correctly removed the page fold and wrinkle noise, but visually degraded regions of the foreground text. By comparison, the NoisyOffice model left more of the text intact, but permitted more of the stain features to remain in the output. Better results may be achieved by a sequential model built from an Augraphy-trained denoiser followed by an inpainting model to repair the text.

7.3 An Augraphy Dataset

Privacy and security concerns typically preclude the assembly of large sets of modern document images: most documents are not intended for general viewing. The authors have searched frequently, for over a year, and have been unable to find decently-sized (¿10 images) sets of modern document images, besides the NoisyOffice set. We intend to release an Augraphy-generated dataset in the coming months.

7.4 Augraphy Enhancements

Several upgrades to the Augraphy library itself are also planned.

Scaling While much work has gone into tuning Augraphy's defaults, and we feel that the effects produced are quite realistic, none of the augmentations were designed to be scale-invariant, and so we plan to introduce pre-trained networks into the library to generate effects in the future.

Performance The authors typically run substantial Augraphy jobs on enthusiast or datacenter hardware. Performance enhancements to the library are already underway, which will decrease pipeline execution time dramatically, enabling faster creation of larger datasets on more common hardware.

8 Conclusion

We presented Augraphy, an augmentation framework for generating realistic datasets of modern document images. Two other players in the same space were examined and found lacking for our purposes, motivating the creation of this library. We described creating an Augraphy-noised version of the NoisyOffice dataset, then compared some preliminary results obtained by training a convolutional U-Net on these datasets. Finally, we discussed some future directions for research, and the continued evolution of this tool. Augraphy is licensed under the MIT open source license, and readers are invited to 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 NoisyOffice 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. Paszke, A., Gross, S., Massa, F., Lerer, A., Bradbury, J., Chanan, G., Killeen, T., Lin, Z., Gimelshein, N., Antiga, L., Desmaison, A., Kopf, A., Yang, E., DeVito, Z., Raison, M., Tejani, A., Chilamkurthy, S., Steiner, B., Fang, L., Bai, J., Chintala, S.: Pytorch: An imperative style, high-performance deep learning library. In: Advances in Neural Information Processing Systems 32, pp. 8024–8035. Curran Associates, Inc. (2019), http://papers.neurips.cc/paper/9015-pytorch-an-imperative-style-high-performance-deep-learning-library.pdf
27. Rotman, D., Azulai, O., Shapira, I., Burshtein, Y., Barzelay, U.: Detection masking for improved OCR on noisy documents. arXiv preprint arXiv:2205.08257 (2022)
28. 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

29. 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
30. Vasiljevic, I., Chakrabarti, A., Shakhnarovich, G.: Examining the impact of blur on recognition by convolutional networks. arXiv preprint arXiv:1611.05760 (2016)
31. 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