Augraphy: A Data Augmentation Library for Document Images

Alexander Groleau\textsuperscript{1}, Kok Wei Chee\textsuperscript{1}, Stefan Larson\textsuperscript{2}, Samay Maini\textsuperscript{1}, and Jonathan Boarman\textsuperscript{1}

\textsuperscript{1} Sparkfish LLC (augraphy@sparkfish.com)  \textsuperscript{2} Vanderbilt University

Abstract. This paper introduces Augraphy\textsuperscript{,} a Python library for constructing data augmentation pipelines which produce 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 also for generating challenging test data to evaluate model robustness on document image modeling tasks.

1 Introduction and Motivation

Daily life in the modern world is increasingly data-driven, involving myriad tasks which require the generation and handling of unstructured data. Increasingly, this data occurs in the form of messages or images, often with complex contents. In the office space, most data is created and stored document-form, easy to print and distribute. The lifetime of these documents typically includes physical alterations by printing, scanning, or photocopying operations, so real documents may appear visually very different from their original born-digital counterparts. Real-world processes can introduce many types of distortions: 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; human-made annotations like highlighting or pencil marks can add noise to the page.

The outcomes of many machine learning operations involving documents are impacted by the presence of such noise; 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 [13] consists of scanned document images, many of which have substantial amounts of scanner-induced noise, as does the FUNSD form understanding benchmark [21]. Other intermediate-level tasks like optical character recognition (OCR) and page layout analysis may perform optimally if noise in a document image is minimized [7,35,39].

\textsuperscript{3} https://github.com/sparkfish/augraphy
lower-level task of document denoising tackles the document noise problem directly, by attempting to remove noise from a document image [5,11,27,33,34]. 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, reducing the need for human toil.

![Image of noisy and clean document images with Augraphy augmentation](image)

**Fig. 1.** Augraphy can be used to introduce noisy perturbations to document images like the original noise seen above, which is a real-life sample from RVL-CDIP. We re-created a clean reproduction and then created a noisy version by applying several Augraphy augmentations.

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 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 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.

Augraphy has been used in several research efforts: Larson et al. (2022) [28] used Augraphy to mimic scanner-like noise for evaluating document classifiers trained on RVL-CDIP; Jadhav et al 2022 [19] used Augraphy to generate noisy document images for training a document denoising GAN; and Kim et al. 2022 [25] used Augraphy as part of a document generation pipeline for document understanding tasks.

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 Pipeline | Centric Based | Python | License |
|------------------|-------------------------|-------------------|--------------|--------|---------|
| Augmentor [2]    | 27                      | ×                  | ✓            | ✓      | MIT     |
| Albumentations [3]| 216                     | ×                  | ✓            | ✓      | MIT     |
| imgaug [23]      | 168                     | ×                  | ✓            | ✓      | MIT     |
| Augly [36]       | 34                      | ×                  | ✓            | ✓      | MIT     |
| DocCreator [22]  | 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., [10,9,44]), audio and speech processing (e.g., [26,31,32]), and computer vision and image processing. In the latter realm, popular tools include Augly [36], Augmentor [2], Albumentations [3], DocCreator [22], and imgaug [23]. 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 [22], 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 [37].
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., [8,15,16,17,24,41,43]). 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 [40]. 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 features 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.

Augraphy’s suite of augmentations was designed to faithfully reproduce the level of complexity in the document lifecycle at seen in the real world. Such real-world conditions and features have direct implementations either within the library or on the development roadmap.

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.
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’s 3D mesh support here is excellent and has inspired Augraphy to add similar functionality in its roadmap. In the meantime, Augraphy currently offers support for paper folds.
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 [14], 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:

```python
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 [18] 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 modifications that might alter 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.

4.2 Augraphy Augmentations

Augraphy provides 26 out-of-the-box augmentations, listed in Table 2, with select examples shown in Figure 3. 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
Fig. 3. Examples of some Augraphy augmentations.

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. Here we find 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 also includes these types of augmentations for completeness and utility. Descriptions of all Augraphy augmentations are available online, along with the motivation for their development and usage examples. ⁴

⁴ https://augraphy.readthedocs.io/en/latest/
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               |
|                 | SubtileNoise              | JPEG                      | Geometric               |
|                 |                           | Markup                    | NoiseTexturize          |
|                 |                           | Faxify                    | PencilScribbles         |
|                 |                           | PageBorder                |                         |

4.3 The 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:

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 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 (behaves like a function). 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 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:

1. Ink is adhered to the paper material during the initial printing of a document.
2. The document is attached to a public bulletin board in a high-traffic area.
3. Over a period of several weeks, it is annotated, defaced, and eventually torn away from its securing staples, flying away in the wind.
4. Fifty years later, our protagonist page is discovered by cleaning staff, who turn it over to library archivists.
5. These conservationists use delicate tools to gently position and record an image of the document, storing this in a digital repository.

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

Realistically reproducing effects in document images requires careful attention to 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 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, born-digital document images.

The value added by the AugraphyPipeline class over a bare list of functions applied in sequence to 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 pipeline metadata 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.

There are also two classes that provide additional critical 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 [45]. 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 multi-pipeline operations.
Foreign. This class can be used to wrap augmentations from external projects like Albumentations and imgaug.

OverlayBuilder. This class uses various blending algorithms to fuse foreground and background images together, which is useful for “printing” or applying other artifacts like staples or stamps.

NoiseGenerator. This class uses make blobs algorithm to generate mask of noises in different shape and location.

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 in which Augraphy was used to augment a new collection of clean documents, producing a new corpus which fills a feature gap in existing public datasets. This collection was used to train a denoising convolutional neural network capable of high-accuracy predictions, demonstrating Augraphy’s utility for robust data augmentation.

All code used in these experiments is available in the augraphy-paper GitHub repository.

5.1 Model Architecture

To evaluate Augraphy, we used an off-the-shelf Nonlinear Activation-Free Network (NAFNet), [6], making only minor alterations to the model’s training hyperparameters, changing the batch size and learning epoch count to fit our training data.

5.2 Data Generation

Despite recent techniques ([4], [30]) for reducing the volume of input data required to train models, large datasets remain king; feeding a model more data during training can help ensure better latent representations of more features, improving robustness of the model to noise and increasing its capacity for generalization to new data.

To train our model, we developed the ShabbyPages dataset [38,12], a real-world document image corpus with Augraphy-generated synthetic noise. Gathering the 600 ground-truth documents was the most challenging part, requiring the efforts of several crowdworkers across multiple days. Adding the noise to these images was trivial however; Augraphy makes it easy to produce large training sets. For additional realistic variation, we also gathered 300 paper textures and used the PaperFactory augmentation to “print” our source documents onto these.

The initial 600 PDF documents were split into their component pages, totaling 6,202 clean document images exported at 150 pixels per inch. We iteratively developed an Augraphy pipeline which produced satisfactory output with few overly-noised images, then ran each of the clean images through this to produce the base training set. Patches were taken from each of these images and used for the final training collection.
Another new dataset — ShabbyReal — was produced as part of the larger ShabbyPages corpus [38,12]; this data was manually produced by applying sequences of physical operations to real paper. ShabbyReal is fully out-of-distribution for this experiment, and also has a very high degree of diversity, providing good conditions for evaluating Augraphy’s effect.

5.3 Training Regime

The NAFNet was given a large sample to learn from: for each image in the ShabbyPages set, we sampled ten patches of 400x400 pixels, using these to train the instance across just 16 epochs.

The model was trained with mean squared error as the loss function, using the Adam optimizer and cosine annealing learning rate scheduler, then evaluated with the mean average error metric.

5.4 Results

Sample predictions from the model on the validation task are presented in Figure 4. The SSIM score indicates that these models generalize very well, though they do overcompensate for shadowing features around text, by increasing the line thickness in the prediction, resulting in a bold font.

To compare the model’s performance on the validation task, we considered the root mean square error (RMSE), structural similarity index (SSIM), and peak signal-to-noise ratio (PSNR) metrics.

| Metric | Score  |
|--------|--------|
| SSIM   | 0.71   |
| PSNR   | 32.52  |
| RMSE   | 6.52   |

As can be seen in Figure 4, the model was able to remove a significant amount of blur, line noise, shadowing from tire tracks, and watercoloring.

6 Robustness Testing

In this section, we examine the use of Augraphy to add noise which challenges existing models.

We first use Augraphy to add noise to an image of text with known groundtruth. The Tesseract [42] pre-trained OCR model’s performance on the clean image is compared to the OCR result on the Augraphy-noised image using the Levenshtein distance.

Then, we add Augraphy-generated noise to document images which contain pictures of people, and perform face-detection on the output, comparing the detection accuracy to that of the clean images.
6.1 Character Recognition

We 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 that remains human readable suitable for evaluating OCR systems.

6.2 Face Detection Robustness Testing

In this section, we use Augraphy to investigate the robustness of object detection models on images that have been altered by Augraphy. Specifically, we explore using Augraphy to add noise to images containing faces in order to test the robustness of face detection models. In this way, we move beyond analyzing robustness of text-related tasks, but the images we analyze in this section can nonetheless appear in documents like newspapers [29] and identification documents [1], which are often scanned from the physical world by noisy scanners. Hence, it is important for
We begin by sampling 75 images from the FDDB face detection benchmark [20]. Then, we use Augraphy to generate 10 altered versions of each image and manually remove any augmented image where the face(s) is not reasonably visible. We then test four face detection models on the noisy and noise-free images. These models are: proprietary face detection models from Google,\(^6\) Amazon,\(^7\) and Microsoft,\(^8\) and lastly the freely-available UltraFace\(^9\) model.

---

\(^6\) https://cloud.google.com/vision/docs/detecting-faces
\(^7\) https://aws.amazon.com/rekognition/
\(^8\) https://azure.microsoft.com/en-us/services/cognitive-services/
\(^9\) https://github.com/onnx/models/tree/main/vision/body_analysis/ultraface
Table 4 shows face detection accuracy of both models on the noisy test set, and example images where no faces were detected by the Microsoft model are shown in Figure 7. We see that all four models struggle on the Augraphy-augmented data, with the proprietary models seeing detection performances drop to between roughly 50-60%, and UltraFace detecting only 4.5% of faces that it found in the noise-free data.

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 described the process for using Augraphy to create a new document image dataset containing synthetic real-world noise, then compared results obtained by training a convolutional NAFNet instance on this corpus.

Finally, we examined Augraphy’s efficacy in generating confounding data for testing the robustness of document vision models.
Augraphy: A Data Augmentation Library for Document Images

Fig. 7. Example augmented images that yielded false-positive predictions with the Azure face detector.

Future work on the Augraphy library will focus on adding new types of augmentations, increasing performance to enable faster creation of larger datasets on commodity 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. Arlazarov, V., Bulatov, K., Chernov, T., Arlazarov, V.: Midv-500: A dataset for identity document analysis and recognition on mobile devices in video stream. Computer Optics 43(5) (2019), https://arxiv.org/ftp/arxiv/papers/1807/1807.05786.pdf

2. 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

3. 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

4. Cao, Y., Yu, H., Wu, J.: Training vision transformers with only 2040 images. CoRR abs/2201.10728 (2022), https://arxiv.org/abs/2201.10728

5. Castro-Bleda, M.J., España-Boquera, S., Pastor-Pellicer, J., Zamora-Martinez, 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

6. Chen, L., Chu, X., Zhang, X., Sun, J.: Simple baselines for image restoration. arXiv preprint arXiv:2204.04676 (2022)

7. Cheriet, M., Khanna, N., Liu, C.L., Suen, C.Y.: Character Recognition Systems: A Guide for Students and Practitioners. Wiley (2007)
8. 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
9. 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
10. 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
11. 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
12. Groleau, A., Chee, K.W., Larson, S., Maini, S., Boarman, J.: Shabbypages: A reproducible document denoising and binarization dataset. arXiv preprint arXiv:2303.09339 (2023), https://arxiv.org/pdf/2303.09339.pdf
13. 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)
14. Harris, C.R., Millman, K.J., van der Walt, S.J., Gommers, R., Virtanen, P., Cournapeau, D., Wieser, E., Taylor, J., Berg, S., Smith, N.J., Kern, R., Picus, M., Hoyer, S., van Kerkwijk, M.H., Brett, M., Haldane, A., del Rio, J.F., Wiebe, M., Peterson, P., Gérard-Marchant, P., Sheppard, K., Reddy, T., Weckesser, W., Abbasi, H., Gohlke, C., Oliphant, T.E.: Array programming with NumPy. Nature 585(7825), 357–362 (Sep 2020). https://doi.org/10.1038/s41586-020-2649-2, https://doi.org/10.1038/s41586-020-2649-2
15. Hendrycks, D., Dietterich, T.: Benchmarking neural network robustness to common corruptions and perturbations. In: International Conference on Learning Representations (ICLR) (2019)
16. Homeyer, A., Geißler, C., Schwen, L.O., Zakrzewski, F., Evans, T., Strohmenger, K., Westphal, M., Bülow, D., Kargl, M., Karja, A., Munné-Bertran, I., Retzlaff, C.O., Romero-López, A., Soltysiński, T., Plass, M., Carvalho, R., Steinbach, P., Lan, Y.C., Boutilier, N., Haber, D., Rojas-Carrulla, 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
17. Hosseini, H., Xiao, B., Poovendran, R.: Google’s cloud vision API is not robust to noise. arXiv preprint arXiv:1704:05051 (2017)
18. Howard, J., Sylvain, G.: Fastai: A layered api for deep learning. Information 11(2) (2020), https://arxiv.org/pdf/2002.04688.pdf
19. Jadhav, P., Sawal, M., Zagade, A., Kamble, P., Deshpande, P.: Pix2pix generative adversarial network with resnet for document image denoising. In: 2022 4th International Conference on Inventive Research in Computing Applications (ICIRCA) (2022)
20. Jain, V., Learned-Miller, E.: FDDB: A benchmark for face detection in unconstrained settings. Tech. Rep. UM-CS-2010-009, University of Massachusetts, Amherst (2010)
21. Jaume, G., Ekenel, H.K., Thiran, J.P.: Funsdb: A dataset for form understanding in noisy scanned documents. In: Accepted to ICDAR-OST (2019)
22. 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

23. Jung, A.B., Wada, K., Crall, J., Tanaka, S., Graving, J., 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

24. Karahan, S., Kilinc Yildirim, 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

25. Kim, D., Hone, T., Yim, M., Kim, Y., Kim, G.: Technical report on web-based visual corpus construction for visual document understanding. arXiv preprint arXiv:2211.03256 (2022). https://arxiv.org/pdf/2211.03256.pdf

26. 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

27. Kulkarni, M., Kakad, S., Mehra, R., Mehta, B.: Denoising documents using image processing for digital restoration. In: Swain, D., Pattaik, P.K., Gupta, P.K. (eds.) Machine Learning and Information Processing. pp. 287–295. Springer Singapore, Singapore (2020)

28. Larsson, S., Lim, G., Ai, Y., Kuang, D., Leach, K.: Evaluating out-of-distribution performance on document image classifiers. In: Proceedings of the Thirty-sixth Conference on Neural Information Processing Systems Datasets and Benchmarks Track (2022), https://openreview.net/pdf?id=uDlkCI5N7Y

29. Lee, B.C.G., Mears, J., Jakeway, E., Ferriter, M., Adams, C., Yarasavage, N., Thomas, D., Zwaard, K., Weld, D.S.: The newspaper navigator dataset: Extracting headlines and visual content from 16 million historic newspaper pages in chronicling america. In: Proceedings of the 29th ACM International Conference on Information & Knowledge Management (CIKM) (2020). https://doi.org/10.1145/3340531.3412767

30. Lee, S.H., Lee, S., Song, B.C.: Vision transformer for small-size datasets (2021). https://doi.org/10.48550/ARXIV.2112.13492, https://arxiv.org/abs/2112.13492

31. Maguolo, G., Paci, M., Nanni, L., Bonan, L.: Audiogmenter: a matlab toolbox for audio data augmentation. Applied Computing and Informatics (2021)

32. 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)

33. 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)

34. 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

35. O’Gorman, L., Kasturi, R.: Document Image Analysis. IEEE Computer Society (1997)

36. Papakipos, Z., Bitton, J.: AugLy: Data augmentations for robustness. arXiv preprint arXiv:2201:06494 (2022)
37. 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

38. Project, T.A.: Shabbypages dataset (2022), https://github.com/sparkfish/shabby-pages

39. Rotman, D., Azulai, O., Shapira, I., Burshtein, Y., Barzelay, U.: Detection masking for improved OCR on noisy documents. arXiv preprint arXiv:2205.08257 (2022)

40. 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/2022/09956167/1IHoLM9J3qI

41. Schöning-Markiefka, B., Pryalukhin, A., Hull, 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

42. Smith, R.: An overview of the tesseract ocr engine. In: Ninth international conference on document analysis and recognition (ICDAR 2007). vol. 2, pp. 629–633. IEEE (2007)

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

44. 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

45. Zamora-Martínez, F., España Boquera, S., Castro-Bleda, M.: Behaviour-based clustering of neural networks applied to document enhancement. In: Computational and Ambient Intelligence (2007), https://link.springer.com/chapter/10.1007/978-3-540-73007-1_18