Streak2O: Data Augmentation for Handwritten Text Recognition in Neural Networks

Eduardo J. Beltran Feliciano
Master of Computer Engineering – Digital Signal Processing
Dr. Marvi Teixeira
School of Electrical and Computer Engineering
Polytechnic University of Puerto Rico

Abstract *— Streak2O is a machine learning data augmentation algorithm based on the combination of two other independent algorithms: Streak and Droplet. These three augmentations are implemented as non-trainable TensorFlow custom Keras layers to optimize execution time in a GPU based environment. They generate configurable random artifacts that imitate real life handwritten historical document or manuscript water damage and document mishandling. Testing this augmentation algorithm with small subsets of the NIST-SD19 dataset on a convolutional neural network architecture shows that they can help reduce neural network overfitting falling partially into the category of synthetic data generation.*

Key Terms *— Handwritten Text Recognition, Machine Learning, Synthetic Data Augmentation, TensorFlow.*

INTRODUCTION

One of the most widely studied problems in the field of pattern recognition and computer vision is optical character recognition (OCR) [1]. Handwriting Text Recognition (HTR) is a sub field of OCR that relates to detecting and classifying non-mechanized characters, those written with ink, graphite, or other substances over a physical media. HTR imposes its own challenges including segmentation, style variation by writer, irregular spacing and orientation, usage of non-standard symbols, and noise caused by degradation and mishandling [2][3].

An increase computational power and new tools for the usage of machine learning (ML) algorithms have influenced the way OCR and HTR are handled. Traditional approaches split the HTR problem into two main parts, segmentation, and classification [4][5]. It is important to distinguish online and offline recognition. "In online recognition a time series of coordinates, representing the movement of the pen-tip, is captured, while in the offline case only an image of the text is available" [6]. M. Liwicki, et. al. [7] captured life features from the users such as tracing speed that are not available when working with documents.

Semantic segmentation [8] and image classification [9][10] have shown promising results with the usage of convolutional neural networks (CNN) and deep CNN (DCNN). Each of these processes is equivalent to the two-step classic processing of HTR, segmentation and classification. This opens the possibility of designing an end-to-end neural network that can perform both tasks. Shi, et. al. [11] have a proposal for such an end-to-end DCNN applied to scene text recognition. Long Short-Term Memory (LSTM) units are commonly combined into the newer ML neural networks used for HTR as they can help classify by using the sequential appearance of features extracted by prior convolutional layers [1]. Namysl and Konya proposed using bidirectional LSTM units to achieve better results and indicate that Google's open-source Tesseract engine makes use of a similar neural architecture [1].

In image classification, augmentation algorithms are routinely utilized to enrich image data sets. Augmentation has two main purposes, generating synthetic data to enrich small data sets and to reduce overfitting over the training data. TensorFlow [12] and MATLAB [13] offer built-in tools for implementing common image augmentations such as rotation, horizontal or vertical reflection, scaling, translation, and shearing.

Literature offers additional augmentation methods useful for image classification [14][15][16].

CutOut is an augmentation technique that replaces a squared block in of the image with a constant colored or Gaussian pattern. CutOut showed varied improvement in validation accuracy for image classification depending on the size of the cutout region. This technique has the benefit of not being computationally intensive and therefore allowing it to be in-line with the neural network training [14].

“CopyPairing is a mixture of Copyout and SamplePairing"[16]. Copyout is an enhancement of CutOut, proposed by P. May, that replaces a square area from an image with a square area from a different image of the data set. SamplePairing, proposed by Inoue, uses an image of averaged colors from two images of the same class to train the network [15]. CopyPairing, proposed by May, mixes Copyout and SamplePairing augmentation techniques in alternating schedules during training. The result is a lower error rate against the test data. May theorizes that the imperfect sampling provided by these augmentations allows the neural network to focus on relevant features and distinguish them from misguiding details [16].

In general, the modified samples used for training are generally considered a type of synthetic data. For OCR and HTR, it is possible to generate synthetic data that is not based on a previous data set. Jaderber, et. al. [17] and Ahmad, et. al. [18] generated training samples by using different computer font typefaces that are then process by multiple random transformations. Ahmad, et. al. evaluated different type faces individually on its word recognition rate (WRR) against real world test samples and later trained with combined typefaces which reached a higher WRR.

DEVELOPMENT OF IMAGE AUGMENTATIONS

Development of the text image data augmentation algorithms follow a four-step process. A MATLAB proof of concept for both the Streak and Droplet augmentations was followed by three stages that focused on making the algorithms more efficient while combined with the TensorFlow Sequential [12].

The first development outside of MATLAB was performed under python using the CuPy [19] library as it offers an API for GPU accelerated operations based on the Nvidia CUDA library. The Streak and Droplet algorithms offered very good speed while using the CuPy library. However, when combined with TensorFlow and Keras this version of the code had two main problems.

Firstly, the CuPy library could not deliver its arrays located in the GPU directly to the TensorFlow model. CuPy arrays had to be send back to CPU/RAM as NumPy arrays that could then be accepted by the Keras ImageDataGenerator object and then back into the model as Tensor objects. Transmission of data from CPU memory to GPU memory greatly slowed the training during model fit.

Secondly, as the CuPy library takes over part of the GPU memory, this memory becomes inaccessible to the TensorFlow Library. In fact, the CuPy library had to be loaded first or TensorFlow would take possession of the GPU's RAM, as desired to allow training of larger models and batches, and the CuPy library would not load.

The similarities between CuPy and NumPy allowed the transition of the code from GPU execution to CPU execution quite easily. This new code working in the CPU made inaccessible the parallelism capabilities offered by the multiple GPU cores. However, by removing the conflicts between the libraries and reducing data transmission between CPU memory and GPU memory, the algorithm developed on top of the NumPy library running exclusively in CPU proofed faster for model training.

The last two approaches to the development of this algorithms considered inserting the augmentation as a prepossessing function within the keras ImageDataGenerator object. This object already offers common data augmentation procedures such as flip, rotation, shift, zoom, brightness, and shear.

A TensorFlow custom layer would insert the augmentation stage as the first layer of the model. These layers are not trained as they perform random transformations on the input data. They can be activated and deactivated randomly, selectively or by schedule by combining their usage with custom callback functions. They offer increase speed by utilizing the tensor object operation parallelism.

Previous image augmentation methods have shown to increase classification accuracy for object detection [14][15][16]. However, unlike the augmentations in this document, those augmentations do not translate directly to real examples of image degradation for handwritten documents or historical manuscripts. The augmentation effects proposed and developed in this project intend to imitate real world patterns found in water damaged manuscripts [2][3].

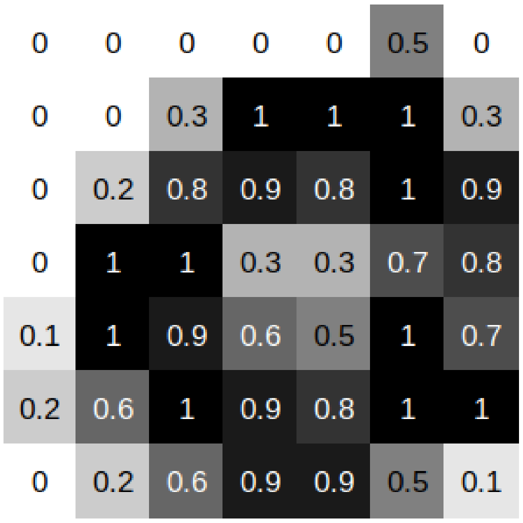
Description of Algorithms

The algorithms developed here are inspired in real world artifacts, that physical media may develop due to mishandling or environmental effects. The algorithms' objective is to generate realistic synthetic data or augment small datasets.

Ink and Floating-Point Operations

The algorithms developed consider the text over a background as pool of ink with different concentrations. Due to this any image received by the algorithm needs to be transformed into a floating-point value range from 0 to 1 where 0 represents background and 1 represents text as shown in Figure 1.

The original MATLAB and early Numpy algorithm could only handle grayscale images, but the current Numpy and the TensorFlow algorithms can work with single channel or three channel images. To achieve realistic artifacts in the augmented images, the algorithm may need to work with the image negative to appropriately map ink. Figure 2 displays the artifacts generated when failing to invert colors on the right compared to the default behavior on the left.



0	0	0	0	0	0.5	0
0	0	0.3	1	1	1	0.3
0	0.2	0.8	0.9	0.8	1	0.9
0	1	1	0.3	0.3	0.7	0.8
0.1	1	0.9	0.6	0.5	1	0.7
0.2	0.6	1	0.9	0.8	1	1
0	0.2	0.6	0.9	0.9	0.5	0.1

Figure 1
Single Channel Ink Mapping

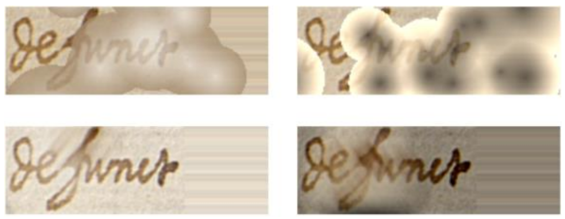


Figure 2
Examples of Droplet and Streak Augmentation

During the MATLAB proof of concept, the original images where initially cleaned simply by a low pass filter to affect only relevant text ink, but this proved to remove artifacts from the original source that gave the image variability and realism. Finally, the algorithm used a K-means algorithm to classify the image pixels and determine which image regions represented ink. This K-means approach allowed the image to retain its original artifacts. When the algorithm was moved to CuPy and then to NumPy the same steps were followed by using the K-means algorithms offered by OpenCV.

The text mask produced from the the K-means algorithm was a float valued mask. After initial erosion, the values masked where set to a full mask with value of 1.0. The mask is then dilated further than the original size and given a mask value of 0.5.

This produced a mask with a contour gradient which avoided hard lined effects on the execution of the augmentation, producing color boundaries on the limits of a hard-set mask. Preferably the contours would follow a multi-step gradient, but the 0.5 contour produced realistic effects. This float valued mask was multiplied against the original image to capture the inked areas.

Proof of Concept

The first iteration of the Streak augmentation was both highly iterative and recursive. The Streak augmentation proposed uses a K-means (n=2) algorithm to classify pixels between ink and background as described earlier. The mask generated was used to first take a random percent of the ink recursively at different steps of the predetermined path and leave a trace of the taken ink in the direction of the path, observe Figure 3. It intends to imitate the effect of an object displacing ink along the surface of the text as would happen with fresh ink or pencil graphite.

After obtaining the K-means mask, it was multiplied pixel by pixel against the input to extract the ink of the affected area. The matrix of the ink inside the mask is called the take. This take was constructed from the extraction achieved by combining the K-means mask with a circular region mask created from a parametric distance calculation. The take is then partially dropped along a path of pixels living a trace of ink. If the path was too long the iteration extracted ink from further steps along the path until the end of the path was reached.

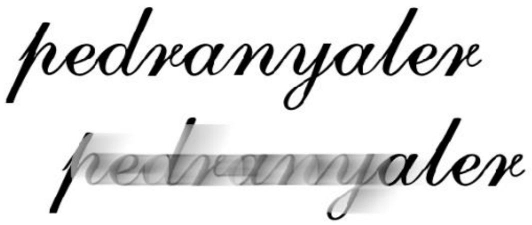


Figure 3
Example of Streak Augmentation

To avoid concentrating the ink at the end of the path, the midway extractions and tracing of the ink had to be performed before the original. This meant that the initial take, and path calculation had to stay in memory waiting to be used until all its children's iterations were performed, this version of the Streak augmentation was a recursive function. To maintain efficiency the next step of the recursion only happens after $\delta \times R$ pixels along the path, where R was the radius used to calculate the parametric distance, the radius of the regional mask, and δ is configurable fractional multiple, 0.5 by default. In this sense approximately $L \div (\delta \times R)$ steps for L the length of the path, and R and δ as previously defined.

Current TensorFlow implementation does not perform mid-path operations. By using the translate function from TensorFlow Addons [12] it can follow paths on any angle. The MATLAB and NumPy algorithms were limited to pixel size shifts to perform efficient translation. With the TensorFlow Addon translate function, batch translations of stacked gradients can be translated in different directions, and to different distances. Two loops in the previous iterations became a single loop with parallel computing that allowed movement in more flexible angles. Although this improves speed of execution, eliminating the double loop also limits the capacity of recursive execution, as each execution occupies $L_{max} \times B$ times the size of a single input, where Lmax is the configurable max length of a streak and B is the batch size during training.

TensorFlow Noise Handling

Custom TensorFlow Keras layers fall in two main categories: dynamic and non-dynamic. Dynamic layers allow for a broader set of control mechanics where the developer can intermingle python logic with TensorFlow logic. Dynamic layers however cannot be executed using eager execution [12] as eager execution prohibit python logic to access values from the CPU during run-time. Although disabling Eager Execution allows development of tools using python logical operations it also slows down execution due to the communication between the CPU RAM and video memory in the GPU.

Non-dynamic custom layers can run under eager execution but cannot make use of non-TensorFlow logic as memory exchanges between CPU and GPU are restricted. Non-dynamic layers variables are stored in tensor arrays or tensor constants which restricts the type of operations that can be performed against the data. Non-dynamic custom layers proofed to be much faster both during training and during evaluation making them the preferable development environment for any real-world implementation.

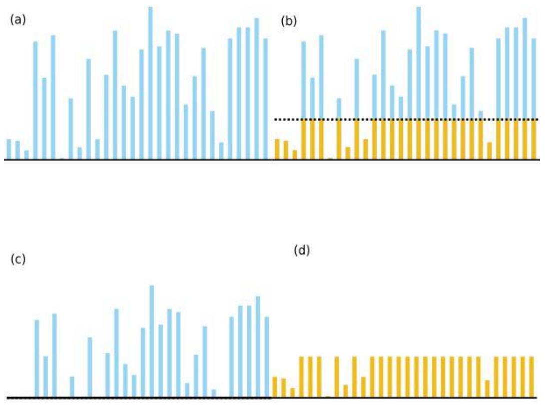


Figure 4
Example of Streak Augmentation

Due to execution constrains of the TensorFlow eager execution, predictable loop size, the K-means approach was abandoned. The noise handling algorithm for the non-dynamic layers also maps input values into a float range from 0 to 1. This algorithm clips the values to a random value between a fourth and a third of each channel average (the average color), as seen in Figure 4. The values below the threshold are kept in memory to be reintroduced into the image. The values clipped above the threshold are passed to the augmentation algorithm that generates a modified output. After

The noise is reintroduced to the output after the augmentation algorithm to preserve any artifacts present in the lower valued pixels of the original image. This clipping approach allowed parallel execution between batch images within the TensorFlow eager execution environment.

Previous image augmentation methods have shown to increase classification accuracy for object detection [14][15][16]. However, these methods do not translate directly to real examples of image degradation for handwritten documents. The augmentation effects proposed in this project intend to imitate real world patterns found in water damaged manuscripts. Although initial design into this effect considered parametric, iterative, and recursive operations, during design recursive execution was exchanged by iterative operations that could be translated into matrix operations and therefore TensorFlow layers.

TensorFlow Memory Requirements

The memory requirements of the TensorFlow implementations are predictable. The Streak augmentation, and therefore the Streak2O augmentation, memory increases at a cubic rate of the max length. In particular if we assume that the max length is linearly related to one of the inputs dimensions, for example $L_{max} = \sigma \cdot \min(s_X, s_Y)$ where s_X and s_Y are the dimensions of the input size and σ is a constant ratio, then assuming the two dimensional input has fixed aspect ratio, if $s_{X1} = k \cdot s_X$, the image becomes k times wider with the same aspect ratio, the augmentation will require k^3 more memory to execute.

Observe that, given the above, if we define memory requirements as $M_i = s_{Xi} \cdot s_{Yi} \cdot L_{max,i} \cdot M_P$ where M_P is the memory required by a single pixel then:

$$M = s_X \cdot s_Y \cdot L_{max} \cdot M_P \quad (1)$$

$$s_{X1} = k \cdot s_X \to s_{Y1} = k \cdot s_Y \wedge L_{max,i} = k \cdot L_{max}$$

$$\to M_1 = s_{X1} \cdot s_{Y1} \cdot L_{max,1} \cdot M_P$$

$$= k^3 \cdot s_X \cdot s_Y \cdot L_{max} \cdot M_P$$

$$\to \frac{M_1}{M} = \frac{k^3 \cdot s_X \cdot s_Y \cdot L_{max} \cdot M_P}{s_X \cdot s_Y \cdot L_{max} \cdot M_P} = k^3 \quad (2)$$

It should be possible to reduce memory requirements by restricting L_{max} to a multiple of average character height instead of the image size.

Droplet Augmentation

The Droplet augmentation proposed also uses a K-means (n=2) masking algorithm to detect ink areas. It generates a circular gradient effect around a center point with a color density equivalent to the display color density removed from the masked text. Figure 5 displays this effect on the right side.

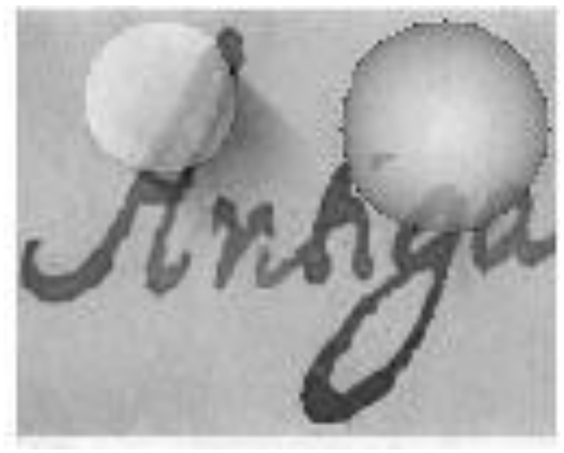


Figure 5
Example of Streak Augmentation

The Droplet Augmentation was designed to imitate the effects of water on ink text on paper. The idea is that ink diluted by water drops tends to form a darker ring at the end of the water flow through and on top the paper. The original MATLAB and NumPy algorithms only differ from the TensorFlow algorithm in terms of the noise removal procedure. However, with the parallelism used in the new TensorFlow Streak algorithm, the Droplet Augmentation was enhanced to combine multiple gradients into an irregular gradient that could be used to create shapes other than a single circular region. The custom layer accepts an additional argument during build called repetitions that sets the number of circular regions to combine.

Unlike the Streak augmentation, the repetition argument need not be configured in terms of the input size and should only cause lineal increments in memory requirements, instead of the exponential growth from the Streak length as shown in Equation 1.

Streak Augmentation

Figure 5 left side displays an earlier version of the Streak augmentation which had not yet implemented any form of masking mechanism. This earlier implementation affected the background color. By using masking, the alternate version shown in Figure 6 avoided generating the previous circular background artifact.

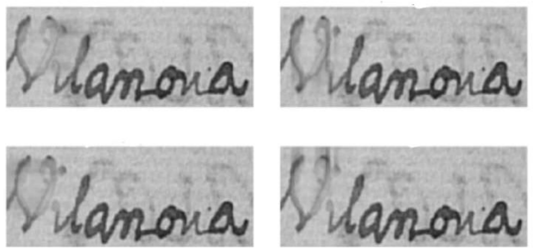


Figure 6
Streak Augmentation using a Mask

Data Sets

The NIST Special Database 19 (NIST-SD19) [20] is compose of samples of 62 categorized characters and symbols. The dataset offers a special organization for these categories under the ‘by_merge’ directory, where upper- and lower-case symbols that cannot be distinguished are integrated into a single category during training. The dataset offers non-grayscale, black and white handwritten characters over a noiseless white background.

The ICDAR 2017[21], from the International Conference on Document Analysis and Recognition, is composed of historical manuscripts. ICDAR 2017 was used to test the realism of the artifacts generated on actual documents and achieve results like the ones observed in examples such as seen in previous research [2][3].

Streak2O Augmentation

Streak2O is a combination of both the Streak and the Droplet augmentations performed in sequence, in that order. It therefore would share memory requirements and configurations with both augmentations.

DESCRIPTION OF TENSORFLOW LIBRARY

TensorFlow Layers

The final augmentation algorithms designed may run both in and out a TensorFlow eager environment. They are all based on a custom layer called *activateLayer* that allows interaction with callbacks for TensorFlow custom control. Current TensorFlow implementations do not trigger callback events between batches during eager execution. Due to this limitation, TensorFlow Variables used to control the flow of execution can only be updated at the start or end of an epoch by custom callbacks.

responsiveLayer

The *responsiveLayer* is the base layer that includes functionality to interact with the *activateLayer* callback to include flow control during training and evaluation.

Parameter usage for the responsiveLayer can be found in Table 1. This layer class also includes methods required to interact with the activateLayer callback: activate, deactivate, activate_out_of_-training, deactivate_out_of_training, and operation.

Table 1
Parameters of responsiveLayer constructor

Parameter	Description
dtype	Sets the data type for the Tensor objects inside a layer.
batch_size	Required as it is not necessarily given by a previous layer as part of the build method.]
dynamic	If false allows the layer to be used in TensorFlow eager execution.
trainable	Defaults to False. Augmentation layers are not expected to learn weights during training.
name	Defaults to “identity”. This layer is meant to be a parent class for augmentation layers.
kwargs	Accepts other named parameters.

The activate and deactivate methods are used to control the augmentation during training. The activate_out_of_training and deactivate_out_of_-training are used to control the augmentation during evaluation and testing.

The streakLayer inherits behavior from respon-siveLayer. This class creates a non-trainable layer that performs a random streak on each image in the batch. The streaks positions, sizes and directions are randomized and not shared between images of the same batch.

It can be customized using the parameters in Table 2. Parameters shared with the responsive-Layer behave as previously described.

Table 2
Parameters of streakLayer constructor

Parameter	Description
seed	Sets a randomization seed to allow repeatable behavior if desired.
negative	If True(default), do nothing, else inverts input before and after augmentation.
distance multiplier	Represents what percent of the smallest side's length should be considered the maximum length of a streak over the image.
radius multiplier	This multiplier allows customization of average radius or the initial take that defaults to half the length of the smallest side.
height limits	A tuple that defaults to (0.2, 0.8). Boundaries in percent for the vertical initial center of the effect in respect to the image.
width limits	A tuple that defaults to (0.1, 0.7). Boundaries in percent for the horizontal initial center of the effect in respect to the image.
ink percent	A tuple that defaults to (0.8, 0.95). Boundaries of the random percent amount of ink removed from the initial position.

The dropletLayer inherits behavior from respon-siveLayer. This class creates a non-trainable layer that performs one or more random droplets on each image in the batch. The droplets positions and sizes are randomized and not shared between images of the same batch.

The dropletLayer can be customized using the parameters in Table 3. Parameters shared with the previous layers behave as previously described.

The streak2OLayer inherits behavior from responsiveLayer and includes one instance of each previous augmentation, streakLayer and dropletLayer. This class creates a non-trainable layer that performs a streak and one or more random droplets on each image in the batch. The positions, sizes and directions of the augmentations are randomized and not shared between images of the same batch.

The parameters of the streak2OLayer have all been covered as part of the parameters for the responsiveLayer, the streakLayer and the dropletLayer. Parameters with the prefixes ‘streak_’ and ‘grad_’ refer to the streakLayer and the dropletLayer components of the streak2OLayer, respectively.

Table 3
Parameters of streakLayer constructor

Parameter	Description
repetitions	Sets the number of droplets to draw on each image. Defaults to 15.
radius multiplier	This multiplier allows customization of average radius or the initial take that defaults to a third of the length of the smallest side.

TensorFlow Custom Callbacks

The activateLayer callback used in junction with custom layers, derived from responsiveLayer, to conditionally activate or deactivate the augmentation at epoch begin. Allows configuring an initial number of epochs of training without the augmentation active. The callback can also follow a pattern of n epochs with the layer active followed by m epochs with the layer deactivated. The ‘sleep_init_epochs’ is an integer valued parameter that maintains an augmentation layer inactive for a given number of epochs at the start of training. The ‘sleep_wake_pattern’ is an iterable of two integer values parameter that describes a pattern of number of alternating active and inactive epoch lengths.

The captureObservations callback is derived from keras base Callback class. It was designed as a low computational cost alternative to the TensorBoard callback that captures a great set of information during model training. The callback has triggers in three training events: “on_-train_begin”, “on_train_end”, “on_epoch_end.” It references the best validation accuracy to keep a copy of the best model weights and can trigger a training early stop after a set number of consecutive epochs finishes without better accuracy or loss. The model also tracks loss and validation data across epochs to populate the database and keeps backup of best epoch and checkpoint epoch weights.

Neural Network Design

The code developed for this project includes a customize builder of Simoyan, K. and Zisserman, A. like models [10]. Their architecture proposes using multiple convolutional layers with small receptive field in sequence to simulate a larger receptive field. The model builder receives four tuples that describe the number of convolutional sets, if a convolutional set has an extra padding convolutional layer, if dropout will be used after a max-pooling, and the number of dense layers used before finishing with an argmax activation. These tuples also specify the number of filters for each convolutional set and the number of nodes in the dense layers.

Current training is using less convolutional layers and more dense layers to construct a system more prone to overfitting to test if the augmentation layers can delay this occurrence.

Training and Testing Augmentation

Evaluating Augmentation

To test the efficacy of the new augmentation methods we will use a control group where training will not include any type of augmentation using four predefined randomization seeds to create and feed 9 different train sample sizes from 20,000 to 180,000 samples increasing in steps of 20,000 more training samples from the NISTSD19 dataset.

The same process was followed for each augmentation layer developed using the same four randomization seeds to select the training data and generate the augmentation for the Streak, Droplet and Streak2O augmentations. CNN were trained for a maximum of 50 epochs with a patience of 10.

The Streak and Droplet augmentation layers had two different configuration formats 1 and 2. The Streak2O augmentation had three configurations $\{(1,1),(1,2),(2,2)\}$, where the first position indicates the configuration of its Streak component and the second position indicates the configuration for the Droplet, for a total of three configurations.

Results

The results include data from 4566 evaluation data points. Figures 7 to 10 displays a comparison between the loss and accuracy achieved by neural networks trained using different augmentation procedures.

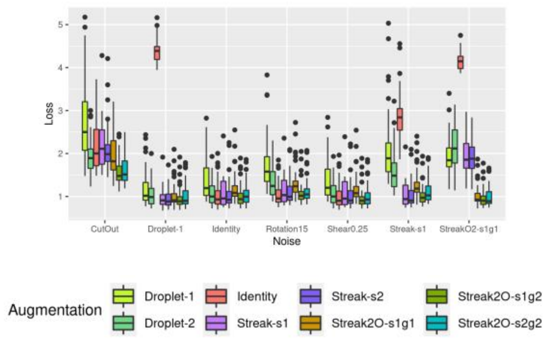


Figure 7
Loss at Best Epoch per Augmentation and Evaluation Noise

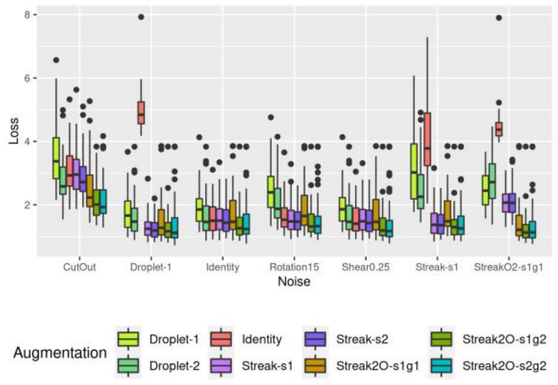


Figure 8
Loss at Last Epoch per Augmentation and Evaluation Noise

Conclusion

The Droplet algorithm showed only partial support in classification of samples under Streak noises. On other hand, Streak algorithm showed full support of Droplet noise. However, neither of them showed optimal support of the noise generated by the combine algorithm, Streak2O.

Although, the Streak algorithm showed similar performance to the Streak2O against Droplet and Streak noises, the Streak2O augmentation showed better performance against the Streak2O noise. We can conclude that the two artifacts generated are in fact different to the combination of both, Streak2O, during training. Streak2O offer better noise handling performance than just using the Streak or the Droplet algorithms independently.

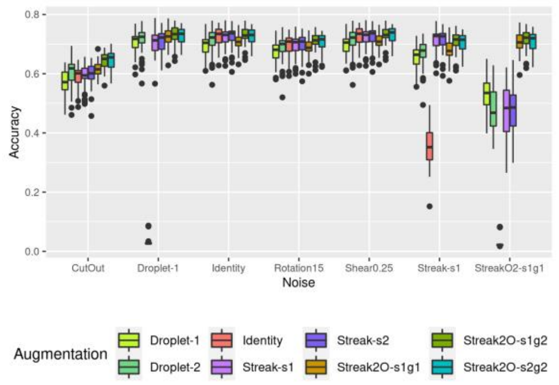


Figure 9
Accuracy at Best Epoch per Augmentation and Noise

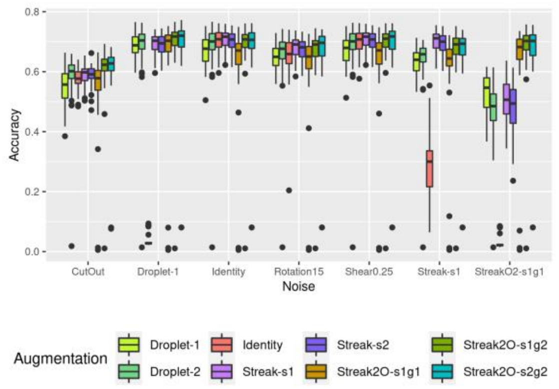


Figure 10
Accuracy at Last Epoch per Augmentation and Noise

All three augmentations show potential benefit in reducing overfitting in small datasets. These augmentations also show potential for the augmentation of synthetic data. They should help the neural network focus on relevant features and ignore degradation and mishandling artifacts in manuscripts and other physical handwritten media.

It is important to note that one of the configurations of the Droplet algorithm (Droplet-1) showed significant lower performance than the control group against the control (Identity), the transformation noises, Rotation and Shear, and the CutOut noise tests. This configuration also affected the Streak2O-s1g1 that used the same configuration for the Droplet augmentation. The Droplet effect configuration may affect the training efforts.

Future Work

The Streak2O algorithm should be tested for incremental training against manuscript and synthetic data. Memory and execution optimization for large images could be implemented using Tensor masks. The base gradient texture used for Droplet algorithm could be enhance for realism. It would be interesting to see if Streak2O and CutOut augmentations can replace one another in terms of increasing overall performance.

REFERENCES

[1] M. Namysl and I. Konya, "Efficient, Lexicon-Free OCR using Deep Learning," 2019. [Online]. Available: https://arxiv.org/abs/1906.01969. [Accessed: February 15, 2020].

[2] V. Rouchon, M. Desroches, V. Duplat, M. Letouzey, and J. Stordiau-Pallot, "Methods of aqueous treatments: the last resort for badly damaged iron gall ink manuscripts," *Journal of Paper Conservation: IADA reports = Mitteilungen der IADA*, vol. 13, no. 3, pp. 7–13, 2012.

[3] C. Carşote, P. Budrugeac, R. Decheva, N. S. Haralampiev, L. Miu, and E. Badea, "Characterization of a byzantine manuscript by infrared spectroscopy and thermal analysis," *Revue Roumaine de Chimie*, vol. 59, no. 6-7, pp. 429–436, 2014.

[4] M. Maheshwari, D. Namdev, and S. Maheshwari, "A Systematic Review of Automation in Handwritten Character Recognition," *International Journal of Applied Engineering Research*, vol. 13, no. 10, pp. 8090–8099, 2018.

[5] L. R. Schomaker, "Retrieval of handwritten lines in historical documents," Proceedings of the *International Conference on Document Analysis and Recognition, ICDAR*, vol. 2, no. June, pp. 594–598, 2007.

[6] A. Graves, M. Liwicki, S. Fernández, R. Bertolami, H. Bunke, and J. Schmidhuber, "A novel connectionist system for unconstrained handwriting recognition," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 31, no. 5, pp. 855–868, May 2009.]

[7] M. Liwicki, A. Graves, H. Bunke, and J. Schmidhuber, "A Novel Approach to On-Line Handwriting Recognition Based on Bidirectional Long Short-Term Memory Networks," in Proc. *9th Int. Conf. on Document Analysis and Recognition*, 2007.

[8] L. C. Chen, G. Papandreou, I. Kokkinos, K. Murphy, and A. L. Yuille, "DeepLab: Semantic Image Segmentation with Deep Convolutional Nets, Atrous Convolution, and Fully Connected CRFs," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 40, no. 4, pp. 834–848, 2018.

[9] D. Ciregan, U. Meier, and J. Schmidhuber, "Multi-column deep neural networks for image classification," Proceedings of the *IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, no. February, pp. 3642–3649, 2012.

[10] K. Simonyan and A. Zisserman, "Very deep convolutional networks for large-scale image recognition," *3rd International Conference on Learning Representations, ICLR 2015 - Conference Track Proceedings*, pp. 1–14, 2015.

[11] B. Shi, X. Bai, and C. Yao, "An End-to-End Trainable Neural Network for Image-Based Sequence Recognition and Its Application to Scene Text Recognition," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 39, no. 11, pp. 2298–2304, 2017.

[12] A. Martín et al. "{TensorFlow}: Large-Scale Machine Learning on Heterogeneous Systems," *Preliminary White Papers*, 2015.

[13] M. H. Beale, M. T. Hagan, and H. B. Demuth, *Deep Learning Toolbox™ Reference*, r2019b ed. The MathWorks, Inc, 2019.

[14] T. DeVries and G. W. Taylor, "Improved Regularization of Convolutional Neural Networks with Cutout," 2017. [Online]. Available: https://arxiv.org/abs/1708.04552. [Accessed: October, 2019].

[15] H. Inoue, "Data Augmentation by Pairing Samples for Images Classification," 2018. [Online]. Available: https://arxiv.org/pdf/1801.02929.pdf. [Accessed: March 13, 2020].

[16] P. May, "Improved Image Augmentation for Convolutional Neural Networks by Copyout and CopyPairing," *2019*. [Online]. Available: https://arxiv.org/pdf/1909.00390.pdf. [Accessed: October, 2019].

[17] M. Jaderberg, K. Simonyan, A. Vedaldi, and A. Zisserman, "Synthetic Data and Artificial Neural Networks for Natural Scene Text Recognition," *2014*. [Online] Available: https://arxiv.org/pdf/1406.2227.pdf. [Accessed: October, 2019].

[18] I. Ahmad and G. A. Fink, "Training an Arabic handwriting recognizer without a handwritten training data set," Proceedings of the *International Conference on Document Analysis and Recognition, ICDAR*, vol. 2015 Novem, pp. 476–480, 2015.

[19] C. R. Harris et al. "Array programming with NumPy," *Nature*, vol. 585, no. 7825, pp. 357–362, Sep. 2020. [Online]. Available: https://doi.org/10.1038/s41586-020-2649-2. [Accessed: November, 2019]

[20] P. J. Grother and K. K. Hanaoka, “NIST Special Database 19 - Handprinted Forms and Characters Database,” pp. 1–30, 2016. [Online]. Available: https://s3.amazonaws.com/nist-srd/SD19/1stEditionUserGuide.pdf%oAhttps://www.nist.gov/srd/nist-special-database-19. [Accessed: February, 2020].

[21] A. Fornes, et al. “Proceedings of the International Conference on Document Analysis and Recognition, ICDAR, vol. 1, pp. 1389–1394, 2017.