Unknown-box Approximation to Improve Optical Character Recognition Performance

by

Ayantha Randika Ponnamperuma Arachchige

A thesis submitted in partial fulfillment of the requirements for the degree of

Master of Science

Department of Computing Science

University of Alberta

Abstract

Optical character recognition (OCR) is a widely used pattern recognition application in numerous domains. Several feature-rich commercial OCR solutions and opensource OCR solutions are available for consumers, which can provide moderate to excellent accuracy levels. These solutions are general-purpose by design to serve a wider community. However, accuracy can diminish with difficult and uncommon document domains. Preprocessing of document images can be used to minimize the effect of domain shift. In this thesis, we investigate the possibility and the effect of using OCR engine feedback to train a preprocessor. The main obstacle in this approach is propagating the error signal through an opaque OCR engine. Circumventing this obstacle, we propose a novel preprocessor trained using gradient approximation. Unlike the previous OCR agnostic preprocessing techniques, the proposed training approach approximates a particular OCR engine's gradient and trains the preprocessor module eliminating the need for intermediate labels. We compare two different methods to our proposed approach to establish a better training pipeline. Experiments with two different datasets and two OCR engines show that the presented preprocessor is able to improve the accuracy of the OCR engine from the baseline accuracy by applying pixel-level manipulations to the document image.

Preface

Parts of this thesis have been submitted to The 16th International Conference on Document Analysis and Recognition (ICDAR 2021).

To my parents and friends
for the endless support and encouragement

The most exciting phrase to hear in science, the one that heralds new discoveries, is not "Eureka" but "That's funny..."

– Isaac Asimov (1920–1992).

Acknowledgements

I would like to express my gratitude to my supervisor Nilanjan Ray for his guidance and support throughout this study. This thesis would not have been possible without his help. Additionally, I would like to thank Xiao Xiao and Allegra Latimer from Intuit Inc. for their valuable input. I would also like to thank Intuit Inc. for the funding support provided for this project.

Contents

List of Tables

List of Figures

Chapter 1

Introduction

1.1 Motivation and Problem Statement

Optical Character Recognition(OCR) is the process of converting an image of a printed document or a handwritten text into a computer understandable form. In recent years OCR has thrived alongside machine learning technology with deep learning and new Recurrent Neural Network (RNN) architectures. Current OCR systems based on deep learning have achieved significant improvements in accuracy, and efficiency [9]. Thanks to these improvements, several commercial OCR solutions have been developed, and they are used in numerous fields to automate document handling tasks. Numerous big names in cloud technologies and OCR technologies have started to offer cloud-based OCR solutions. This new generation of OCR solutions comes with all the benefits of the Software as a Service (SaaS) delivery model where the consumer does not have to think about the hardware and maintenance.

One characteristic of commercial OCR solutions is that they are often general-purpose by design. This trait is desired both from the business perspective and consumer perspective. It allows a large number of consumers to use the same solution for a wider array of document types. This desired property can become an obstacle when it comes to specialized and uncommon document domains. Different domains may have different document types with unique aberrations and degradations, which can hinder the performance of a general-purpose OCR. While it is theoretically possible to either train an open-source OCR or to build a new OCR engine from scratch to accommo-

date the domain shift, these solutions are not realistic in practice for OCR users in the industry, given the amount of resources required for training and the potential degradation of efficiency. Therefore, the more viable approach is to enhance a commercial OCR solution with external components to improve accuracy.

OCR combines two different domains. Generally, input for the OCR is an image, and the output of the OCR process is a text string. The input for the OCR is a computer vision problem, while the output is frequently a natural language processing problem. Therefore, it is possible to add components to an OCR engine to preprocess the input and post-process the output based on these two technologies. The output of OCR is a text string and can be post-processed using natural language-based techniques [10], [54] or by leveraging outputs from multiple OCR engines [45]. This is beyond the scope of this work. Instead, we focus on preprocessing to enhance image quality prior to the OCR, which usually occurs in the image domain, though earlier hardware-based preprocessors have worked in the signal domain [19], [56]. Image preprocessing for OCR includes image binarization, background elimination, noise removal, illumination correction and correcting geometric deformations, all of which aim to produce a simpler and more uniform image that reduces the burden on the OCR engine [50].

Preprocessing is considered a separate problem when it is used in the context of OCR. Generally, a batch of images is preprocessed and then used to train the OCR model. The same preprocessing is applied to the new images at the inference time. We observe a few drawbacks in this approach. What if the later document images are deviated from the original image set used to train the model? Utilizing the general preprocessing technique used on training images will not mitigate the issue of domain shift of input images. On the other hand, even if the OCR does not use preprocessing, the same problem will occur with a different document domain. The preprocessing of these new images should be able to shift the appearance of them to the original images. At this point, the problem is that new images might not have attached ground truth images that are similar to the original image set. Therefore, a prepro-

cessor must be hand-crafted to manipulate the input images using the original images as a reference.

Instead of hand-crafting the preprocessor, one can utilize the OCR engine's feedback to train the preprocessor model by comparing the OCR output to the ground truth text and then using the classic backpropagation algorithm to propagate the OCR error to the preprocessor component. Instead of the OCR engine agnostic preprocessing approach, we hypothesize that a preprocessor trained based on the OCR feedback will perform better at producing an image closer to the optimal image expected by the OCR engine than the uninformed preprocessing techniques. This approach will mitigate the need for intermediate ground truth containing the 'clean' images by depending on the OCR engine feedback. Even though this approach sounds well-fitting, it has an unavoidable problem when it comes to commercial OCR solutions. The internal workings of commercial OCR solutions are not available to the ordinary consumer. Additionally, they can contain non-differentiable components, which will render the use of the backpropagation algorithm harder or impossible. Therefore another workaround will require to avoid this problem.

In this work, we present an approach to train a preprocessor while avoiding the aforementioned problem. Our training approach treats the preprocessing problem as an extension of the OCR engine by directly observing the OCR error produced by the images. According to our hypothesis, this type of more informed approach will allow our preprocessor to learn more subtle pixel-level manipulations producing an image closer to the optimal input image. In our training pipeline, we consider the OCR engine as an unknown-box and utilize a gradient approximation method to calculate and propagate the error to the preprocessing component to optimize it. During the training, the forward pass goes through the OCR engine to the evaluation function to calculate the output error. Our approximation method calculates the error gradient of the OCR engine in the backwards pass and propagates that to the preprocessor component. This technique allows us to use any OCR engine in our training pipeline to train an individualized preprocessor for that particular engine. Therefore one can use our approach to train a preprocessor for a specific doc-

ument domain and utilize a feature-rich commercial OCR engine combined with that preprocessor.

1.2 Contributions

We mark the following contributions in this thesis:

- We present a novel approach to train a preprocessor that performs pixel-level manipulations that can be tweaked to accommodate any OCR engine, including commercial unknown-box OCR solutions. We experiment with two different approaches to approximate the OCR engine's error gradient to establish a viable and pragmatic solution.

- To demonstrate the ability of our preprocessor, we use Point Of Sales (POS) receipts, which often have poor printing and ink quality and therefore renders a challenging task for OCR software [22]. Additionally, we include the VGG synthetic word dataset [25] to establish the generality of our solution. To establish the OCR engine agnostic property of our approach, we use two OCR engines to evaluate our method.

1.3 Thesis Outline

The rest of the thesis is organized as follows. The first part of Chapter 2 discusses the background of conventional and learning-based preprocessing techniques used in OCR. The second part of Chapter 2 explains the gradient approximation techniques relevant to our work. Chapter 3 of the thesis presents our approach and implementation details of our training pipeline. Chapter 4 contains the details of the experiment setup, results and the discussion of the results. Chapter 5 concludes the thesis with general observations and future directions for our work.

Chapter 2

Background

2.1 Document Image Preprocessing for OCR

Different preprocessing methods address different shortcomings in incoming images for OCR engines. Binarization and skeletonization focus on creating less complicated images, while techniques, such as super-resolution, attempt to add more details. Geometrical correction methods create uniform characters in shape and orientation. The shared goal of these different methods is to produce output images that the OCR engine is more comfortable or familiar with.

2.1.1 Conventional Preprocessing Methods

Image binarization is the process of converting a greyscale image to a completely black and white image. Perfect execution of binarization should convert all the foreground elements such as letters to black and reduce everything else to white or vice versa. Image binarization is a commonly used and studied preprocessing technique for OCR tasks since the 1980s [56]. The critical component of binarization is determining the thresholding value for a given image pixel. Two thresholding approaches are: global thresholding which applies the same threshold for the entire image, and local or adaptive thresholding, where the threshold is calculated for each pixel. Otsu method [40] is one of the earliest and popular binarization methods, which uses zeroth and the first order cumulative moments of the grayscale histogram to find a global threshold value. Niblack method [37] is a classic local thresholding method that uses

the properties of the neighbourhood to determine the threshold. It uses the mean and the standard deviation of the pixels in the given window around the pixel. 'Object Attribute Thresholding' [33] is an algorithm that generates a global threshold based on local attributes. It applies the Otsu method iteratively to find a set of global threshold values and uses run-length histogram based decision tree approach to choose an optimal threshold.'Object Attribute Thresholding' is developed to be used with unconstrained document images with complex backgrounds.

O'Gorman also proposed an iterative method to calculate a global thresholding value based on the local attribute: connectivity [39]. It utilizes a sliding window technique along with run-length histograms to maximize the connectivity preservation of thresholded areas. Chang et al. proposed a five-step document binarization method [7] focused on addressing the connectivity issues in characters. The primary operations in the five steps are: finding a threshold using the histogram and eliminating the background, applying histogram equalization to the thresholded image and improving the edges of the characters. Sauvola and Pietikäinen proposed an adaptive binarization method [50] which divides a given image into small regions and classifies them into two categories: text content and non-textual contents. An adapted version of the Niblack method [37] is used to binarize the text components in the image, while a fuzzy logic based algorithm is used for non-textual components. The double thresholding method proposed by Chen et al. uses the Canny edge detector to detect edges first and generates the binary image based on the cleaned-up edge image [8]. Rangoni et al. proposed a fully automatic method [44] to tune the parameters of Sauvola's method.

Skeletonization or thinning is another popular technique used to preprocess documents for OCR. The function of skeletonization is to reduce the dimensions of an object [48] and therefore, in the context of characters, it means reducing the stroke thickness to 1-D curves. [27] presents a comprehensive survey and a comparison of thinning from the perspective of OCR. In addition to the methods discussed in [27], [26] proposed a thinning algorithm that preserves connectivity and end-points. It mainly focuses on OCR applications.

Ahmed and Ward proposed a rule-based rotation invariant thinning algorithm intended for character recognition [1]. Skew correction is also a preprocessing technique used to improve OCR results. A skew detection algorithm finds the skewness angle, and image rotation is used to correct the issue. Srihari and Govindaraju used Hough transformations to analyze the document images and proposed a method to detect the skew angle [53]. Hough transformation is a computationally costly operation. Hinds et al. proposed a skew detection method that incorporates run-lengths to reduce the amount of data prior to Hough transformation [20]. Le et al. proposed an algorithm for further data reduction based on the same techniques[29]. [38], [59] and [16] proposed skew detection techniques based on the nearest neighbor clustering methods. Das and Chanda proposed a skew detection algorithm in document images taking a different approach; mathematical morphology [12].

2.1.2 Learning-based Preprocessing methods

Some learning-based preprocessing techniques have the same goals as conventional methods. These new techniques aim to improve upon conventional methods. The deep learning-based binarization approach by Vo et al. [55] is an example of that. This approach uses Convolutional Neural Networks (CNN) based hierarchical architecture. DeepOtsu [18] is a different approach to binarization. It uses CNNs iteratively to produce a more uniform image, on which the Otsu method [40] is used to generate the final binary image. Some learning-based preprocessing methods approach preprocessing with different angles than conventional methods. Deep learning-based Super-Resolution (SR) is employed in [28] and [43] to improve the OCR accuracy. These methods aim to present a more clear and crisp image to OCR with higher resolution. Independent component analysis based method, which focuses on images of inscriptions captured by handheld devices, was proposed by Garain et al. [13]. Bui et al. proposed a CNN-based method to select the most suitable set of preprocessing techniques from a set of preprocessing techniques which included binarization, noise reduction and sharpening [6]. Sporici et al. presented a preprocessing method specific to Tesseract 4.0 OCR, based on CNNs [52].

Figure 2.1: Overview of a sequentially arranged component pipeline where f_i represents the function of the component while L represents the loss function. Arrows indicate the flow of inputs and outputs.

Except for [44], [6] and [52], the other aforementioned methods do not take the OCR feedback into consideration. They each have an output target defined separately from the OCR engine. The success of the preprocessor is measured with respect to this target. Even the methods that specifically focus on OCR do not consider the OCR feedback. These methods might have been effective with early OCR attempts such as template matching, structure analysis or shallow Neural Networks (NN). However, the same behaviour cannot be expected from the current Deep Learning (DL) based OCR techniques. [6] shows that preprocessing can reduce the OCR accuracy depending on the document type. Following the same line of thought, we hypothesize that a preprocessor optimized for a given OCR engine would be able to produce a better approximation of the optimal images expected by this specific engine, leading to higher OCR accuracy compared to traditional generic preprocessing methods. Furthermore, we hypothesize that a DL-based image processing model would be more successful in applying pixel-level manipulations to the document images matching the expectations of the OCR than the conventional methods.

2.2 Backpropagation and Differentiability

Training a preprocessor based on the OCR feedback appears a sound and appealing approach. The classic backpropagation algorithm can be utilized to train a preprocessor component based on the error produced by the OCR engine. The backpropagation algorithm depends on the differentiability of each component in the pipeline. Let's consider a set of n components $\{f_1, f_2, \ldots f_{n-1}, f_n\}$

arranged sequentially where $\mathbf{x}$ is the input to f_1, and the input of each f_i is the output of f_{i-1} (also denoted by f_{i-1} for convenience) for $i = 2, ..., n$ due to the sequential nature of the arrangement. The final output, f_n goes into a differentiable loss function L, which compares it with some ground truth to produce a quantitative error value. This arrangement is depicted in Figure 2.1. Let ϕ_i be the parameters of each f_i and the gradient with respect to L can be calculated by the Jacobian-vector product:

$$\nabla_{\phi_i} L = (J_{\phi_i}^{f_i})(\nabla_{f_i} L), \tag{2.1}$$

where $J_{\phi_i}^{f_i}$ is the Jacobian of f_i with respect to ϕ_i and the gradient of L with respect to f_i, i.e., $\nabla_{f_i} L$ is computed by the following Jacobian-vector product:

$$\nabla_{f_i} L = (J_{f_i}^{f_{i+1}})(\nabla_{f_{i+1}} L), \tag{2.2}$$

where $J_{f_i}^{f_{i+1}}$ is the Jacobian of f_{i+1} with respect to f_i and the gradient of loss L with respect to f_{i+1}, i.e., $\nabla_{f_{i+1}} L$ would be computed at the $(i+1)^{\text{th}}$ module. The above two equations are the chain-rule of differentiation for multivariate calculus. If f_i is differentiable, the chain rule can be used in the backpropagation algorithm.

2.3 Backpropagation with Non-differentiable Components

Using a commercial OCR system in the training pipeline would render the chain-rule computation difficult because the proprietary system's internal mechanisms are not available to the average user. On the other hand, even if the code is available, OCR engines may contain non-differentiable components for which gradient or Jacobian computation may not be possible. Therefore, the OCR engine needs to be treated as an unknown-box in the training pipeline.

The straightforward way to avoid this problem is to use the optimal input distribution as ground truth when training the preprocessor, completely removing the OCR engine from the training pipeline. However, such intermediate training data are rarely available. Additionally, a preprocessor trained

in this setting would not be able to fine-tune the preprocessing operations according to the specific needs of the OCR engine. Therefore to get the benefit of the OCR engine feedback in the training process, the gradient of the OCR engine components needs to be approximated, enabling the use of back-propagation. There are approximation techniques developed in various fields to handle this type of non-differentiable components in the training pipeline. The next sections describe a few such techniques.

2.3.1 Straight Through Estimator

Straight Through Estimator (STE) is a simple estimation technique suggested in [4]. As the name implies the idea is to treat the non-smooth component or non-differentiable function as the identity function in the backward pass. This technique is suggested for when the hard threshold function is used as a layer in the NN. In the forward pass, the thresholding function is used in the backward pass STE is utilized. However, in our case, with the OCR engine in the pipeline, we are shifting from the image domain to the text-domain. STE cannot be used to bridge such a large gap.

2.3.2 Score Function Estimator

Another method of estimating the gradient is using the Score-Function Estimator (SFE) (2.3) [35]:

$$\nabla_\theta \mathbb{E}_{x\sim p_\theta}[f(x)] = \mathbb{E}_{x\sim p_\theta}[f(x)\nabla_\theta \log p_\theta(x)], \tag{2.3}$$

where $f(x)$ is the unknown-box function and p_θ is the input distribution parameterized by θ. (2.3) can be derived by starting from the calculation of the expectation of function f. This is a common task in Reinforcement Learning (RL). Starting from:

$$\nabla_\theta \mathbb{E}_{x\sim p_\theta}[f(x)] = \nabla_\theta \int p_\theta(x) f(x) dx, \tag{2.4}$$

where the derivative can be moved inside the integration with mild assumptions, and after applying identity trick ($\frac{p_\theta(x)}{p_\theta(x)} = 1$) and derivative rule of loga-

rithms, $\nabla_\theta \log p_\theta(x) = \frac{\nabla_\theta p_\theta(x)}{p_\theta(x)}$ we can get the following:

$$\begin{aligned}
\nabla_\theta \mathbb{E}_{x\sim p_\theta}[f(x)] &= \int \nabla_\theta p_\theta(x) f(x) dx \\
&= \int \frac{p_\theta(x)}{p_\theta(x)} \nabla_\theta p_\theta(x) f(x) dx \\
&= \int p_\theta(x) \nabla_\theta \log p_\theta(x) f(x) dx \\
&= \mathbb{E}_{x\sim p_\theta}[f(x) \nabla_\theta \log p_\theta(x)]
\end{aligned} \tag{2.5}$$

In RL, (2.3) was developed into the REINFORCE algorithm [58]. Even though this estimator is unbiased, it can have a high variance, especially in higher dimensions [35]. SFE can be easily converted to the following form where the gradient can be approximated by Monte Carlo sampling from p_θ:

$$\mathbb{E}_{x\sim p_\theta}[f(x) \nabla_\theta \log p_\theta(x)] \approx \frac{1}{n} \sum_{i=1}^{n} f(x^{(i)}) \nabla_\theta \log p_\theta(x^{(i)}). \tag{2.6}$$

There are numerous techniques suggested to lower the variance of SFE while maintaining unbiasedness. However, since we are working with images, the higher variance in higher dimensions can pose a problem.

2.3.3 Evolution Strategies

The Evolution Strategies (ES) method belongs to the family of evolutionary algorithms, which in turn comes under the umbrella of derivative-free optimization techniques. The evolutionary algorithms are inspired by natural selection, where the members of the population who has more favourable traits for survival in a given environment will procreate more due to the larger probability of survival. This higher reproduction will lead to more members with favourable traits in the next generation of offspring. After some generations, the most considerable portion of the population will carry these favourable traits making the population robust and more adapted to the environment. The same principle is utilized in evolutionary algorithms.

The fundamental procedure of an evolutionary algorithm is to generate a population of solutions for the function f according to the distribution p_θ, which is parameterized by θ. Then evaluate the performance of each solution

in the population, select n number of best-performing solutions and finally optimize θ so that p_θ would produce more such solutions. This optimization is an iterative process that goes until the convergence of solutions or another termination criterion.

Inspired by the natural evolution strategies [57], Salimans, et al. proposed to use ES to find optimal policy parameters in RL [49]. The following gradient approximation is used in their approach:

$$\nabla_\theta \mathbb{E}_{\epsilon\sim\mathcal{N}(0,\mathcal{I})}[f(\theta+\sigma\epsilon)] = \frac{1}{\sigma}\mathbb{E}_{\epsilon\sim\mathcal{N}(0,\mathcal{I})}[f(\theta+\sigma\epsilon)\epsilon]. \tag{2.7}$$

The equation (2.8) can be arrived by starting from (2.3). Let $x \sim \mathcal{N}(\theta, \sigma^2\mathcal{I})$ and therefore $p_\theta(x) = \frac{1}{\sigma\sqrt{2\pi}} \exp\left(-\frac{1}{2}\left(\frac{x-\theta}{\sigma}\right)^2\right)$. Now x can be expressed as $x = (\theta + \sigma\epsilon)$ where $\epsilon \sim \mathcal{N}(0, \mathcal{I})$.

$$\nabla_\theta \mathbb{E}_{x\sim p_\theta}[f(x)] = \nabla_\theta \mathbb{E}_{\epsilon\sim\mathcal{N}(0,\mathcal{I})}[f(\theta+\sigma\epsilon)] \tag{2.8}$$

By using the SFE result in (2.8):

$$\begin{aligned}
\nabla_\theta \mathbb{E}_{x\sim p_\theta}[f(x)] &= \nabla_\theta \mathbb{E}_{\epsilon\sim\mathcal{N}(0,\mathcal{I})}\left[f(\theta+\sigma\epsilon)\right] \\
&= \mathbb{E}_{\epsilon\sim\mathcal{N}(0,\mathcal{I})}\left[f(\theta+\sigma\epsilon)\nabla_\theta \log p_\theta(x)\right] \\
&= \mathbb{E}_{\epsilon\sim\mathcal{N}(0,\mathcal{I})}\left[f(\theta+\sigma\epsilon)\nabla_\theta\left(-\log\left(\sqrt{2\pi}\sigma\right) - \frac{(x-\theta)^2}{2\sigma^2}\right)\right] \\
&= \mathbb{E}_{\epsilon\sim\mathcal{N}(0,\mathcal{I})}\left[f(\theta+\sigma\epsilon)\left(\frac{x-\theta}{\sigma^2}\right)\right] \\
&= \mathbb{E}_{\epsilon\sim\mathcal{N}(0,\mathcal{I})}\left[f(\theta+\sigma\epsilon)\left(\frac{\epsilon}{\sigma}\right)\right] \\
&= \frac{1}{\sigma}\mathbb{E}_{\epsilon\sim\mathcal{N}(0,\mathcal{I})}\left[f(\theta+\sigma\epsilon)\epsilon\right]
\end{aligned} \tag{2.9}$$

A key part of this technique is adding the Gaussian noise to the input. Intuitively this can be seen as generating a population with random mutations. In the optimization process, the input distribution (θ) is pushed towards generating samples containing favourable mutations. Monte Carlo estimates of the reparameterization (2.8) presents much lower variance than SFE [47]. However, it can scale poorly with the increase of the dimensions of θ [49]. The result in (2.8) allows us to get the gradient approximation with respect to the input, unlike in SFE, where the gradient is approximated with respect

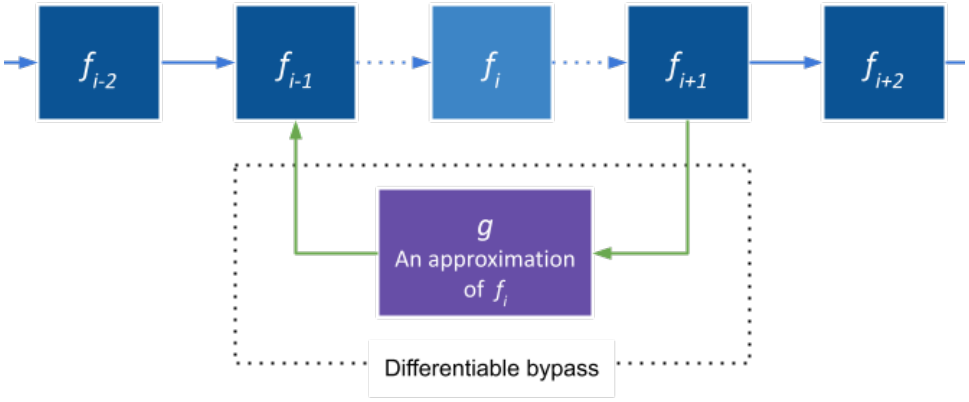

Figure 2.2: Overview of the differentiable bypass method where component f_i is non-differentiable in the training pipeline. The differentiable function g approximates f_i (optimally, given the same input, f_i and should output the same output). Solid blue arrows indicate both forward and backward passes. Broken blue arrows indicate only forward pass while solid green arrows depict only backward pass.

to the parameters of the input distribution. Additionally, ES can be easily parallelized with less communication overhead.

2.3.4 Differentiable Bypass

A fascinating property of NNs is that they are universal function approximators. Cybenko first proposed this property for NNs with sigmoid activation [11] and later Hornik et al. demonstrated that it is, in fact, a property of multilayer feed-forward NN, not the activation function [21]. They further establish that not only the function value, but also its gradient is simultaneously approximated by a neural network [21]. Based on this, Nguyen and Ray proposed EDPCNN [36] in the context of medical image analysis. The principle behind this approach is providing a differentiable bypass in place of the non-differentiable component so that the backpropagation algorithm can operate. Let f_i be a non-differentiable component in the training pipeline (Figure 2.2). Let g be a differentiable function with approximation capability, parameterized by θ. By optimizing the parameters of g for the following:

$$min_\theta \mathcal{L}(f_i(x), g(x)), \tag{2.10}$$

where $\mathcal{L}$ is an appropriate loss function, we can obtain an approximated surface of f_i. Therefore the gradient of g, $\nabla_x g$ can be used in the place of $\nabla_x f_i$. The assumption made here is that given g fits to the f sufficiently well, then:

$$\nabla_x f_i \approx \nabla_x g. \tag{2.11}$$

In [36], authors use a NN for approximator g to approximate a non-differentiable dynamic programming algorithm. This approximating NN is utilized during training and then discarded for testing. In the inference time, the original dynamic programming algorithm is used. Authors have applied EDPCNN in left ventricle segmentation of heart MRI images and demonstrated that it could achieve better results than a complete NN-based solution, especially when the number of training samples is small. Jacovi et al. also proposed the same method under the name 'Estinet' and demonstrated its performance with tasks such as answering 'greater than or less than' questions written in natural language [23]. Additionally, in [36], authors suggest adding Gaussian noise to enable exploration in the approximator while preventing it from overfitting. Intuitively this can be seen as allowing the approximator g to discover the function landscape of f_i more widely. With this modification, the optimization becomes the following:

$$min_\theta \mathcal{L}(f_i(x+\epsilon), g(x+\epsilon)), \tag{2.12}$$

where $\epsilon \sim \mathcal{N}(\mu, \sigma^2\mathcal{I})$ for some mean μ and standard deviation σ. In its primary components, this algorithm is similar to the Deep Deterministic Policy Gradient (DDPG) [32] algorithm in RL. Another approach similar to the differentiable bypass method is proposed by Jaderberg et al. [24] to enable asynchronous training of components in an NN by using synthetic gradients. In their approach, authors use NN models to directly predict the error gradient of a neural component (a layer in a deep NN) based on the layer output, state of the next layer and other external information. However, the gradient prediction models are trained using the actual error gradient later in the process. Therefore, it is hard to adapt to a scenario with an unknown-box.

Chapter 3

Training a Preprocessor with Unknown-box OCR

Out of the methods discussed in 2.3, two methods are selected to implement the training pipeline to train a document image preprocessor. The aim is to investigate and select a better performing method for the task. The differentiable bypass technique discussed in section 2.3 is a fitting choice since it has demonstrated its abilities with images. Therefore it is chosen as the primary avenue of investigation. Reparameterization of the SFE (2.8) in ES is selected as an additional avenue of investigation. Both approaches are implemented using the functions available in the PyTorch [42] library. The following sections discuss the implementation details of two methods with respect to our problem.

3.1 NN-based Approximation

Algorithm 1 is adapted from the algorithm in [36] to match our specific problem. Similar to the original algorithm, our algorithm consists of two loops: 'inner loop' and 'outer loop'. In the inner loop, Gaussian noise $\epsilon_s \sim \mathcal{N}(0, \sigma)$ is added to the image input to 'jitter' it and then the error $\mathcal{M}$ between OCR and the approximator is accumulated as $\sum_s \mathcal{M}_s$. The loss function $\mathcal{M}$ calculates the error between OCR engine output and the approximator output. The 'outer loop' optimizes approximator parameters (ϕ) by minimizing the accumulated error $\sum_s \mathcal{M}_s$. Note that at this point, this minimization only

Algorithm 1: Proposed training

input: $\sigma, S, \{Training\ Images, Ground\ Truths\}$
for $I, l_{gt} \in \{Training\ Images,\ Ground\ Truths\}$ **do**
  $g = Preprocessor(I)$;
  intialize s to 0;
  while $s < S$ **do**
    sample $\epsilon_s \sim \mathcal{N}(0, \sigma)$;
    $\mathcal{M}_s = \mathcal{M}(Approximator(g + \epsilon_s),\ OCR(g + \epsilon_s))$;
    $s = s + 1$;
  $min_\phi \sum_s \mathcal{M}_s$;
  $min_\psi \mathcal{Q}(Approximator(g), l_{gt})$;

applies to the approximator parameters (ϕ). Parameters of the preprocessor (ψ) are frozen and therefore not affected by the first minimization. For the second minimization in the outer loop, the error $\mathcal{Q}$ is calculated by comparing the approximator output with the ground truth. The second minimization only applies to the parameters of the preprocessor (ψ), and the parameters of the approximator (ϕ) are frozen. This algorithm is an alternating optimization process between the preprocessor and the approximating NN.

In [23] authors have categorized this type of alternating optimization as 'online training'. The counterpart approach is 'offline training', where the approximator is trained with the dataset separately from other components and added to the training pipeline with frozen weights. Offline training can be advantageous in situations where invoking the unknown-box function is expensive. However, a disadvantage of offline training is that the previous component's (in our case, the preprocessor) output, which is the input for the approximator, can be different from the original dataset [23]. In our preliminary studies, online training performed better than offline training. A drawback of online training is the 'cold start' problem where the approximator produces garbage outputs at the beginning of the optimization process; therefore, meaningful feedback from the OCR engine is not possible. We trained the approximator with the dataset for some epochs before adding it to the training pipeline to avoid this problem. Therefore, our approach can be considered a 'hybrid training' approach.

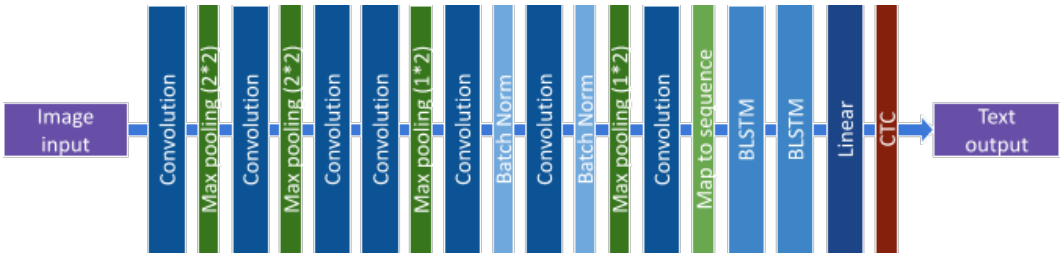

Figure 3.1: Architecture of the CRNN model.

3.1.1 Approximator Model

The requirement for the approximator model in the proposed training setup is to take an image as the input and output a text string. Based on this requirement, we selected the text recognition model, Convolution Recurrent Neural Network (CRNN) [51]. It is a simple yet powerful model for text recognition. However, it does not possess the capacity to detect text in an image. Therefore, the input image of this model needs to be a cropped line of text. CRNN is a combination of CNN layers and Recurrent Neural Networks (RNN). The convolutional layers act as the feature extractor, while the RNN part acts as the character recognizer. The 'map to sequence' layer, which sits between CNN layers and RNN layers, maps CNN feature maps to a sequence that RNN can consume. Two Bidirectional Long Short Memory (BLSTM) [14] layers are used as the RNN component. The architecture of the CRNN model is illustrated in Figure 3.1. CRNN model can handle arbitrary sequence lengths and can be used either lexicon-free or lexicon-based. Approximator, combined with the preprocessor, can yield a quite deep model and can be susceptible to vanishing gradient problem. In that perspective, the smaller size of the CRNN is an added benefit. Connectionist Temporal Classification (CTC) loss [15] is used as the loss function of the CRNN model.

3.1.2 Connectionist Temporal Classification Loss

CTC loss presented in [15] is alignment-free, and therefore it eliminates the need for labelling the positions of the individual sequence items. In our case, without CTC loss, one needs to label each pixel column of the text image with

the corresponding character. More formally, given training set $\mathcal{X} = \{I, \mathbf{l}\}$ where I is the set of images and $\mathbf{l}$ is the set of corresponding label sequences, CTC loss is:

$$\mathcal{L} = -\sum_{I_i, \mathbf{l}_i \in \mathcal{X}} \log p(\mathbf{l}_i \mid \mathbf{y}_i) \tag{3.1}$$

$\mathbf{y}_i$ is the sequence produced by the model. For a single label sequence, $p(\mathbf{l} \mid \mathbf{y})$ defined as:

$$p(\mathbf{l} \mid \mathbf{y}) = \sum_{\pi \in \mathcal{B}^{-1}(\mathbf{l})} p(\pi \mid \mathbf{y}) \tag{3.2}$$

In (3.2) $\mathcal{B}$ is a many-to-one mapping $\mathcal{B} : {L'}^T \mapsto L^{\leqslant T}$ where L is the alphabet of labels and T is the length of the sequence. Therefore $L^{\leqslant T}$ represents all sequences of length less than or equal to T (therefore, $\mathbf{l} \in L^{\leqslant T}$). $L' = L \cup \{blank\}$ where $\{blank\}$ refers to a special label used to indicate 'empty'. $\mathcal{B}$ function, merge all the repeated labels and then removes the blank labels (e.g. $\mathcal{B}(hel\text{-}lo) = \mathcal{B}(hhee\text{-}ll\text{-}loo\text{-}) = hello$). $p(\pi \mid \mathbf{y})$ in (3.2) is defined as follows:

$$p(\pi \mid \mathbf{y}) = \prod_{t=1}^{T} y^t_{\pi_i}, \; \forall \pi \in {L'}^T \tag{3.3}$$

y^t_k is the probability of observing label $k \in L'$ in the output at time t. Directly computing (3.2) can be computationally infeasible or extremely costly. Therefore it is solved by a dynamic programming algorithm named forward-backward algorithm. Given the input $\mathbf{x}$, the classifier should output the most probable labelling sequence $\mathbf{l}'$ for the input:

$$h(\mathbf{x}) = \underset{\mathbf{l}' \in L^{\leqslant T}}{arg\,max}\, p(\mathbf{l}' \mid \mathbf{x}) \tag{3.4}$$

Due to the lack of a tractable decoding algorithm to decode $\mathbf{l}'$ from $\mathbf{y}$, in [15], there are two approximations suggested. In our work, we use the trivial best path decoding approach with the following assumption:

$$h(\mathbf{x}) \approx \mathcal{B}(\pi^*)$$

where π^* is the concatenation of the most active output node at each time step.

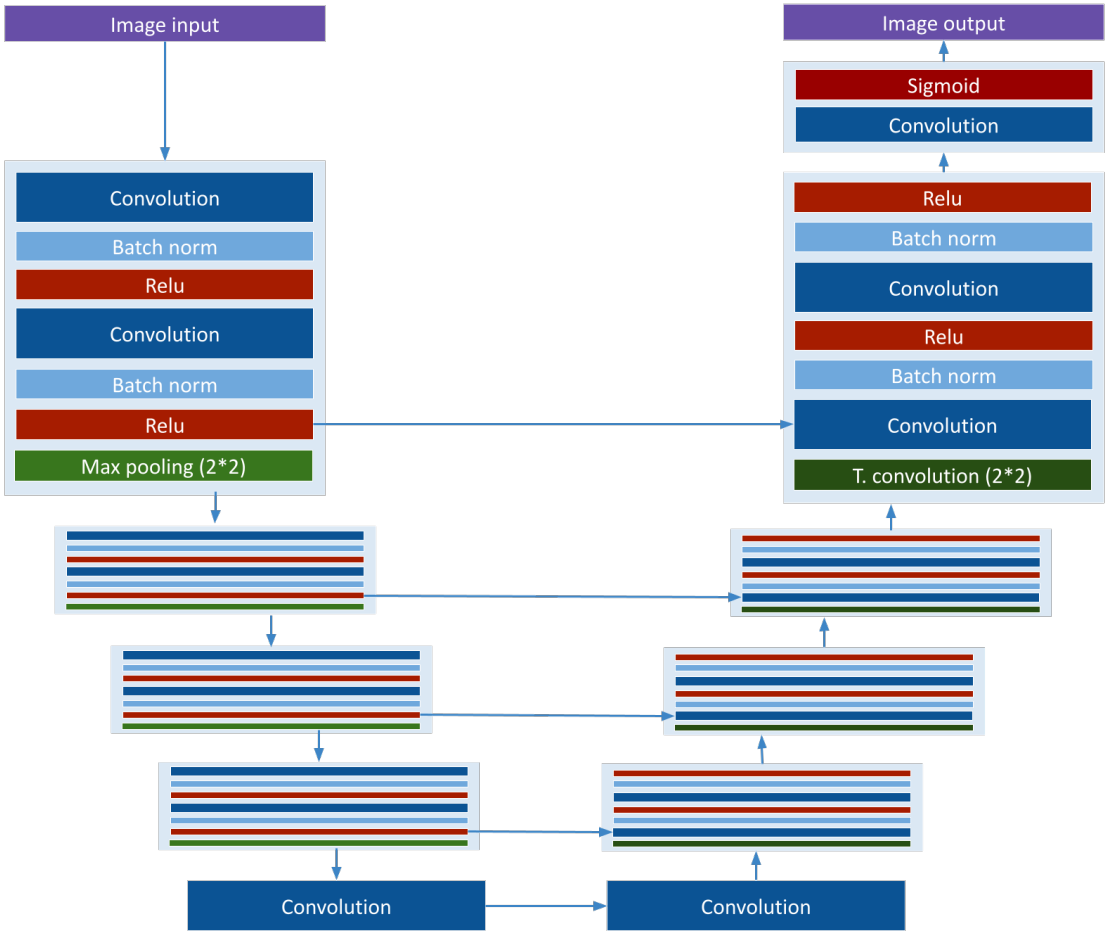

Figure 3.2: Architecture of the U-Net model. The downsizing blocks in the encoder (left) side reduce the input image size by half at each block using the 2×2 max-pooling layers, while the four upsizing blocks on the decoder (right) side increase the feature map size by a factor of two using Transpose Convolutional (T. Convolution) layers. Each encoding block is connected to a decoder block by a skip connection.

3.1.3 Preprocessor Model

The architecture of the preprocessor component is based on the U-Net [46] model. U-Net is a CNN-based encoder-decoder architecture widely used in medical image segmentation. We use the U-Net variation proposed in [5] with added batch normalization layers (Figure 3.2). This U-Net variation has the same input and output dimensions. The number of input and output channels is changed to 1 since we work with greyscale images. The sigmoid function is used as the final activation function to maintain output values

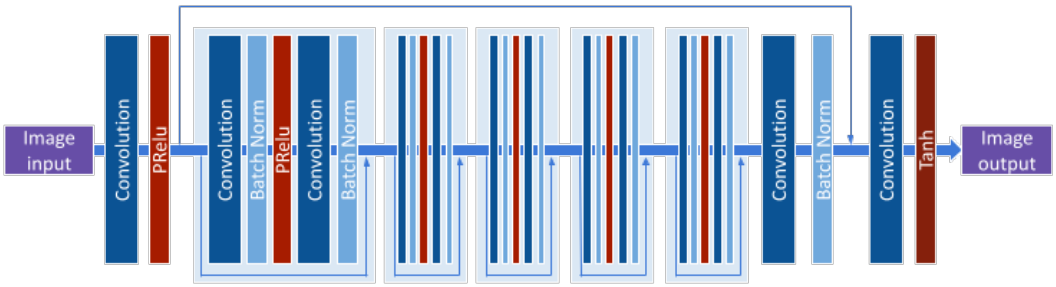

Figure 3.3: Architecture of the SRGAN [30] generator inspired preprocessor model. Pixel shuffle layers of the SRGAN are removed in this model.

in the range $[0, 1]$. In the preliminary studies, we experimented with RED-Net [34], another CNN-based encoder-decoder architecture designed for image restoration such as denoising and super-resolution. After the RED-Net, we experimented with the generator model of the SRGAN [30] after removing the pixel shuffle layers (Figure 3.3). SRGAN is a successful Generative Adversarial Networks (GAN) based approach for image super-resolution. However, the U-Net model achieved better performance at our experiments than RED-Net and SRGAN.

3.1.4 Training Pipeline

Figure 3.4 depicts the overview of the proposed training pipeline based on Algorithm 1. Our training pipeline utilizes two different loss functions for approximator and preprocessor. As described in section 3.1.1, CTC loss is used for the optimization of the approximator parameters (ϕ). This loss is referred to as $\mathcal{M}$ in Algorithm 1. In this error calculation, the comparison is between approximator output and the OCR engine output. A combination of CTC and Mean Squared Error (MSE) losses is used to optimize the preprocessor. CTC loss is calculated by comparing the approximator output to the ground truth. The MSE loss is calculated by comparing the preprocessor output with a 2-dimensional tensor of ones: $J_{m \times n}$ where n and m are the dimensions of the input image. In this context, it represents an entirely white image. Sum of the CTC loss and the MSE loss is used as the loss function (3.5) to optimize the preprocessor parameters (ψ) in Algorithm 1.

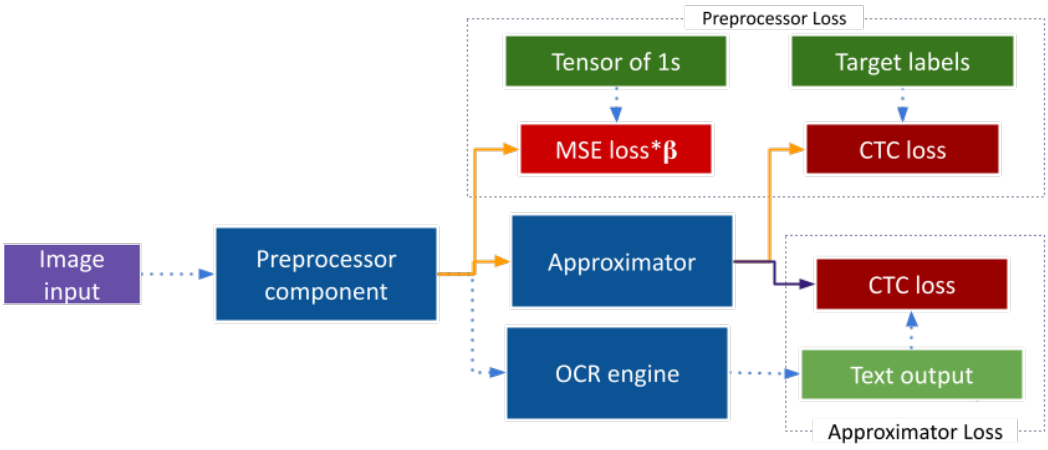

Figure 3.4: Overview of the proposed training pipeline with a NN approximator based on Algorithm 1. The yellow and purple arrows represent the computation paths equipped with backpropagation of preprocessor loss and approximator loss, respectively.

$$\mathcal{Q} = CTC(Approximator(g), l_{gt}) + \beta * MSE(g, J_{m \times n}). \tag{3.5}$$

In (3.5), $g = Preprocessor(Image)$ and l_{gt} is the associated ground truth text for the input $Image$. MSE loss component of the loss function nudges the preprocessor to produce a white image. An excessively bleached image implies no output or incorrect output from the approximator, which increases CTC error. On the other hand, it is beneficial to bleach out the noisy background. Therefore, we hypothesize that this composite loss function will reduce background clutter while preserving the characters. β acts as a hyperparameter to control the effect of MSE loss.

3.2 SFE-based Approximation

Algorithm 2 elaborates our implementation of reparameterization of the SFE in (2.8). The implementation is arranged as a function that returns the error gradient of the OCR engine. There are two inputs to the function in addition to the input image and the corresponding ground truth text. Two additional arguments are σ, the standard deviation of the added Gaussian Noise and

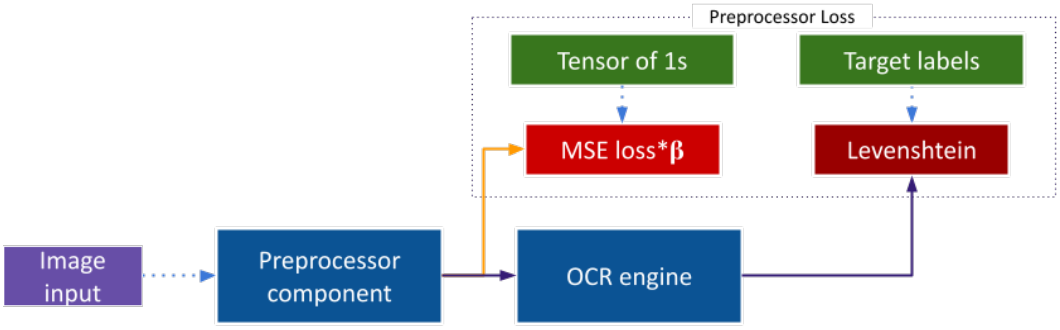

Figure 3.5: Overview of the proposed SFE based training pipeline utilizing Algorithm 2. Solid arrows represent the computation paths equipped with backpropagation. However, computation paths indicated by purple arrows can be considered a pseudo backpropagation since the OCR gradient is directly calculated by Algorithm 2.

Algorithm 2: Gradient approximation by SFE

Function `GetGradients(`$Image$, l_{gt}, σ, n`):`

$g = Preprocessor(Image)$;
sample $\epsilon_1, \dots \epsilon_n \sim \mathcal{N}(0, \mathcal{I})$;
$\epsilon_{n+i} = -\epsilon_i$ for $i = 1, \dots n$;
$\mathcal{L}_i = \mathcal{L}(OCR(g + \sigma\epsilon_i), l_{gt})$ for $i = 1, \dots 2n$;
$gradient = \frac{1}{2n\sigma} \sum_{i=1}^{2n} \mathcal{L}_i \epsilon_i$;
return $gradient$;

n, representing the number of perturbations of the image. In fact, in the gradient approximation process, $2n$ perturbations are generated by utilizing the 'mirrored sampling' technique to reduce the variance [49]. In mirrored sampling, the n number of ϵ originally sampled from the normal distribution are negated to create a total of $2n$ samples. Noise samples ϵ are added to the preprocessed image to obtain $2n$ perturbations of the image. The OCR is run on these images to get the output text, and the error compared to the ground truth text is accumulated and multiplied by corresponding ϵ to produce the error gradient.

Figure 3.5 depicts the training pipeline based on Algorithm 2. The main difference compared to the training pipeline in section 3.1 is the absence of the approximator component and two different loss functions. Therefore,

component-wise, this training pipeline is a more simple one. Unlike the probability distribution output of CRNN in the NN-based method, OCR outputs a text string. Therefore the loss function needs to compare two text strings. However, CTC loss is not intended for direct string comparison. Since there is no need for a differentiable loss function in this approach, we use Levenshtein distance [31], represented by $\mathcal{L}$ in Algorithm 2. It is similar to an evaluation metric discussed in section 4.1.3, except for the multiplication by a constant. Composite loss in equation 3.5 is modified as follows to optimize the preprocessor parameters.

$$\mathcal{R} = Levenshtein\,(s, l_{gt}) + \beta * MSE\,(g, J_{m\times n}) \tag{3.6}$$

where s is the OCR output for some input image I and l_{gt} is the associated ground truth text. $g = Preprocessor(I)$ and $J_{m\times n}$ is a 2-dimensional tensor of ones in the shape of the input image. Levenshtein component of this loss is appraximated by the gradient calculation in Algorithm 2. The same U-Net model (discussed in section 3.1.3) is used as the preprocessor model.

Chapter 4

Experiments and Results

4.1 Experiment Setup

4.1.1 Datasets

Document samples from two different domains are used to train and evaluate the preprocessor. The dataset referred to as 'POS dataset', consists of POS receipt images from three public datasets: ICDAR SOIR competition dataset [22], Findit fraud detection dataset [2] and CORD dataset[41]. Both the ICDAR SOIR dataset and CORD dataset are OCR datasets, and therefore they provide bounding box coordinates for each word on the image. A drawback of the ICDAR SOIR dataset is that its ground truth text is given in capital letters without considering the actual case in the document image. The Findit dataset is intended for fraud detection, and thus, words are not annotated with bounding boxes. Due to these reasons, we used the OCR service provided by Google Cloud Vision API[1] to generate ground truth text and bounding boxes for images in the Findit dataset and the ICDAR SOIR dataset. The output of Google Cloud Vision is manually inspected and adjusted later. Due to the extensive manual labour involved in this task, only a fraction of these two datasets' images is added to our dataset along with the entire CORD dataset. The final dataset (Figure 4.1) images are patches extracted from POS receipts. Patches are extracted without damaging the text areas on the images and resized to have a width of 500 pixels and a maximum of 400 pixels height. The

[1] `https://cloud.google.com/vision/`

Figure 4.1: A few randomly selected samples from the POS dataset.

complete POS dataset contains 3676 train, 424 validation and 417 test images with approximately 90k text areas.

Figure 4.2: A few randomly selected samples from the VGG dataset.

The dataset referred to as the 'VGG dataset' contains the image samples from the VGG synthetic word dataset [25]. The complete dataset in [25] contains approximately 2.5 million synthetic word samples. We randomly selected 60k samples from the original dataset. The final VGG dataset used in this work includes 50k train, 5k validation and 5k test images, each containing a single word.

4.1.2 OCR Engines

Two free and opensource OCR engines: Tesseract [2] and EasyOCR [3], are used to train the preprocessor. Tesseract is a well-known and well-established, open-source OCR engine. In our work, we use the Tesseract version 4.0.0 with LSTM based text recognizer. EasyOCR is a relatively new opensource OCR engine implemented using PyTorch [42] library. It uses CRAFT [3] algorithm for text detection and a CRNN based text recognizer with ResNet [17] as feature extractor. Even though EasyOCR is not particularly slow, it is slow and extremely time-consuming when running with our training scripts. Therefore a modification is added to its code to run on a different GPU than our training scripts. This modification resolved the issue slightly. Both OCR engines are treated as unknown-box components throughout the study except for the slight modification on EasyOCR mentioned earlier.

4.1.3 Evaluation Metrics

Two metrics are used to measure the OCR performance variation with preprocessing. One is a word-level accuracy metric, and the other is a character-level accuracy metric. Since word-level ground truth values are available for both datasets, word-level accuracy is defined as the percentage of words exactly matched with the ground truth. This metric is referred to as 'Accuracy' here onwards. As the character-level measurement, Levenshtein distance [31] based Character Error Rate (CER) is used. In this study, CER is defined as:

$$CER = 100 * (i + s + d)/m, \tag{4.1}$$

where i is the number of insertions, s is the number of substitutions, and d is the number of deletions done to the prediction to get the ground truth text. m is the number of characters in the ground truth. The OCR engine's performance is measured with original images to establish a baseline accuracy level. The preprocessing is then applied, and the OCR engine is run on preprocessed

[2] `https://github.com/tesseract-ocr/tesseract`
[3] `https://github.com/JaidedAI/EasyOCR`

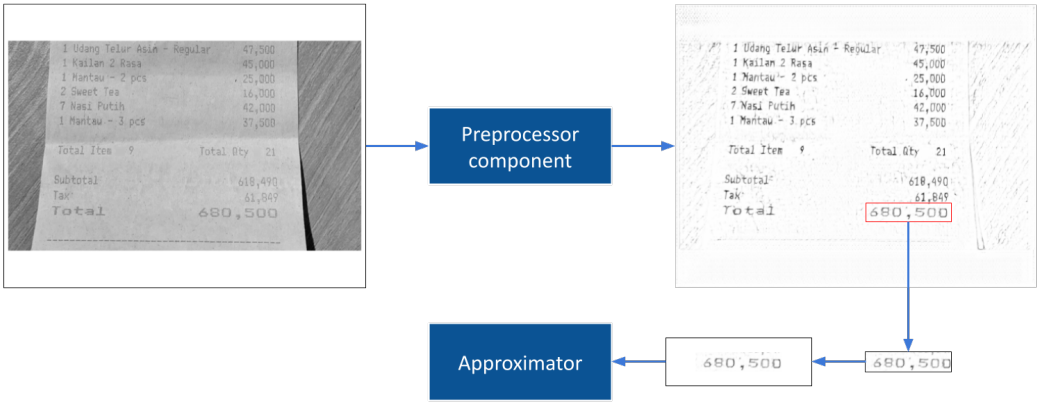

Figure 4.3: Overview of the image sizes (POS dataset) in the NN-approximation based training pipeline. Only a single cropping is depicted in the figure for clarity. All the text areas from the image patch are extracted and sent to the approximator in the training process.

images to measure the impact of preprocessing with respect to accuracy.

4.1.4 Training Details

NN-approximation based Training

A set of preprocessors are trained for fifty epochs each according to the Algorithm 1. A learning rate of 5×10^{-5} is used for the preprocessors with Adam optimizer. Similarly, for approximators, a learning rate of 10^{-4} is used with Adam optimizer. The approximator is pre-trained with the dataset for fifty epochs to avoid 'cold-start' before inserting it into the training pipeline. We maintained $\beta = 1$ in (3.5) and $S = 2$ in Algorithm 1. σ in Algorithm 1 is randomly selected from a uniform distribution containing 0, 0.01, 0.02, 0.03, 0.04 and 0.05. In this context, images are represented by tensors in the range $[0.0, 1.0]$. Therefore, a smaller σ used to introduce noise onto the images to avoid making the images noisy beyond recognition.

The POS dataset images are padded with white to the size of 500×400 pixels to feed into U-Net based preprocessor. Since POS dataset images equal to or less than the size 500×400 pixels, image scaling is unnecessary. CRNN based approximator is arranged only to accept images with a height of 32 pixels. Therefore, to feed into CRNN, words are cropped using bounding-box

values and padded to the size of 128×32 pixels. Figure 4.3 illustrates this process. If the size of the crop exceeded 128×32 pixels, then it is scaled down while maintaining the aspect ratio. Each sample from the VGG dataset only contains a single word. Therefore no cropping is necessary to feed into the approximator. Size of 128×32 pixels is used for both components in the case of VGG samples. If the sample size exceeds 128×32 pixels, it is scaled down while maintaining the aspect ratio. Similar to POS samples, white padding is used to pad the images. On average, it took 19 hours to train a preprocessor with an Nvidia RTX 2080 GPU, while EasyOCR is running on a different GPU.

SFE-based Training

A different set of preprocessors (referred to as SFE-preprocessor here onwards) are trained with Algorithm 2 using Adam optimizer and a learning rate of 5×10^{-5}. We discovered that this approach is also susceptible to a cold start problem. Often at the beginning of the training, preprocessor output does not contain recognizable characters. Therefore, the OCR engine outputs 'empty' for all the perturbations of the image. Due to the mirrored gradients, in the summation, errors cancel each other out and produce a gradient of 0. Additionally, OCR output does not vary much with the perturbations of ϵ if the σ is too small. If the σ is too large, the image becomes noisy beyond recognition. It leads the OCR engine to produce empty text for every perturbation. Both of these scenarios lead to 0 gradients.

At the beginning of the training, the preprocessor is trained to output the same image as input using MSE loss and Adam optimizer to avoid the cold start problem. By trial and error, a constant σ of 0.05 is used. The speed bottleneck of the pipeline is at the OCR engine. Especially in the case of EasyOCR. Therefore, the training time heavily depends on parameter n in Algorithm 2. $n = 5$ is used in our initial experiments to keep the training time acceptable when compared with the NN-approximation based training. Similar to the previous training scenario, we maintained $\beta = 1$ (refer to equation (3.6)), and the same image processing techniques are used for both datasets.

Table 4.1: Accuracy and CER before and after preprocessing with NN-approximation based preprocessor.

OCR engine (training/ testing)	Data	Without preprocessing		With preprocessing			
		Acc. ↑	CER ↓	Acc. ↑	CER ↓	Acc. gain	CER reduc.
Tesseract	POS	54.51%	26.33	83.36%	8.68	28.86%	17.66
Tesseract	VGG	18.52%	64.40	64.94%	14.70	46.42%	49.70
EasyOCR	POS	29.69%	44.27	67.97%	16.46	38.27%	27.81
EasyOCR	VGG	44.80%	26.90	57.48%	17.15	12.68%	9.75

On average, it took 79 hours to train an SFE-preprocessor with an Nvidia RTX 2080 GPU, while EasyOCR is running on a different GPU.

4.2 Results and Discussion

4.2.1 Preprocessor Performance

Table 4.2: Accuracy and CER before and after preprocessing with SFE-preprocessor.

OCR engine (training/ testing)	Data	Without preprocessing		With preprocessing			
		Acc. ↑	CER ↓	Acc. ↑	CER ↓	Acc. gain	CER reduc.
Tesseract	POS	54.51%	26.33	69.17%	16.62	14.67%	9.71
Tesseract	VGG	18.52%	64.40	24.36%	51.97	5.84%	12.43
EasyOCR	POS	29.69%	44.27	46.63%	28.13	16.94%	16.14
EasyOCR	VGG	44.80%	26.90	47.02%	24.69	2.22%	2.21

Results in Table 4.1 and Table 4.2 show that the performances of both OCR engines are improved by preprocessing. With both types of preprocessors, it can be seen that the more the OCR struggles with the dataset more the improvement. Tesseract performed poorly with the VGG dataset and gained a significant accuracy improvement after the preprocessing. The same goes for

Table 4.3: Proposed NN-approximation based vs. SFE-based preprocessors.

OCR engine (training/ testing)	Dataset	NN-based preprocessing		SFE-based preprocessing	
		Accuracy ↑	CER ↓	Accuracy ↑	CER ↓
Tesseract	POS	83.36%	8.68	69.17%	16.62
Tesseract	VGG	64.94%	14.70	27.76%	52.98
EasyOCR	POS	67.97%	16.46	46.63%	28.13
EasyOCR	VGG	57.48%	17.15	47.02%	24.69

the EasyOCR and the POS dataset. Table 4.3 summarizes the accuracy and CER levels of the two approaches. NN-approximation based preprocessors outperform SFE-preprocessors in both OCR engines for both datasets. To some extent, the poor performance of the SFE-preprocessor might be attributed to the small number of perturbations ($n = 5$). However, when considering computational time, using large n appears unpragmatic. In both cases, gradient approximation has proved to work, and it appears that NN-approximation based bypass technique handles the image domain better than SFE.

Figure 4.4 depicts some sample output images produced by the NN-approximation based preprocessor running on the VGG test set. The most notable difference between input images and the processed images is the increased contrast between text and the background. Preprocessing has eliminated complex background components. In the case of Tesseract, the text has become darker, and the background has become lighter. This effect can be seen even when the input image has lighter text and a darker background. Additionally, the images preprocessed for Tesseract appear to have slight skew corrections. On the other hand, in the case of EasyOCR, several images were converted to have light colour text and darker backgrounds. In both cases, preprocessed images appear to be bleached, and it can be the effect of the MSE loss component in the composite loss function (equation 3.5). Additionally, preprocessing has warped and aberrated the text.

Figure 4.5 depicts sample output images produced by the NN-approximation

Figure 4.4: Sample inputs and outputs from the NN-approximation based preprocessor on the VGG test dataset. Column 1: input images, Column 2: image output produced by the model trained with Tesseract, Column 3: image output produced by the model trained with EasyOCR.

based preprocessors running on the POS dataset. It can be observed that shadows, complex backgrounds and noise are suppressed to improve the contrast of the text. Clean, uncluttered backgrounds and high contrast between text and the background provide a clear advantage to the OCR engine. Similar to the VGG samples, the bleaching effect can be seen on the preprocessed images. Furthermore, characters have gained more fluid and continuous strokes, especially low-resolution characters printed with visible 'dots'. Additionally, preprocessing has introduced new artifacts to the characters. This effect is clearly visible in the images preprocessed for EasyOCR. Based on the accuracy improvements and reduction of CER, it can be concluded that these mutations and added artifacts provide extra guidance in character recognition.

Figure 4.6 and Figure 4.7 depict sample output images produced by the

Figure 4.5: Sample inputs and outputs from the NN-approximation based preprocessor on the POS test dataset. Column one contains the sample input images. Columns two and three contain outputs produced by the preprocessor trained with Tesseract and EasyOCR, respectively. Images are slightly cropped.

Figure 4.6: Sample inputs and outputs from the SFE-preprocessor on the VGG test dataset. Column 1: input images, Column 2: image output produced by the model trained with Tesseract, Column 3: image output produced by the model trained with EasyOCR.

SFE-preprocessors running on the VGG test dataset and POS test dataset, respectively. Compared to NN-approximation based preprocessor outputs, it can be observed that the document images are relatively intact, and the text is not warped. The bleaching effect is not prominent except for some cases, and preprocessing has not added new artifacts. Even though preprocessing has improved the text areas in few cases, it has not removed complex backgrounds effectively.

4.2.2 Comparison with Other Methods

Our approach does not require clean document images as targets to train the preprocessor model. It is trained directly with the ground truth text omitting the need for intermediate clean images. There are existing datasets for OCR

Figure 4.7: Sample inputs and outputs from the SFE-preprocessor on the POS test dataset. Column one contains the sample input images. Columns two and three contain outputs produced by the preprocessor trained with Tesseract and EasyOCR, respectively. Images are slightly cropped.

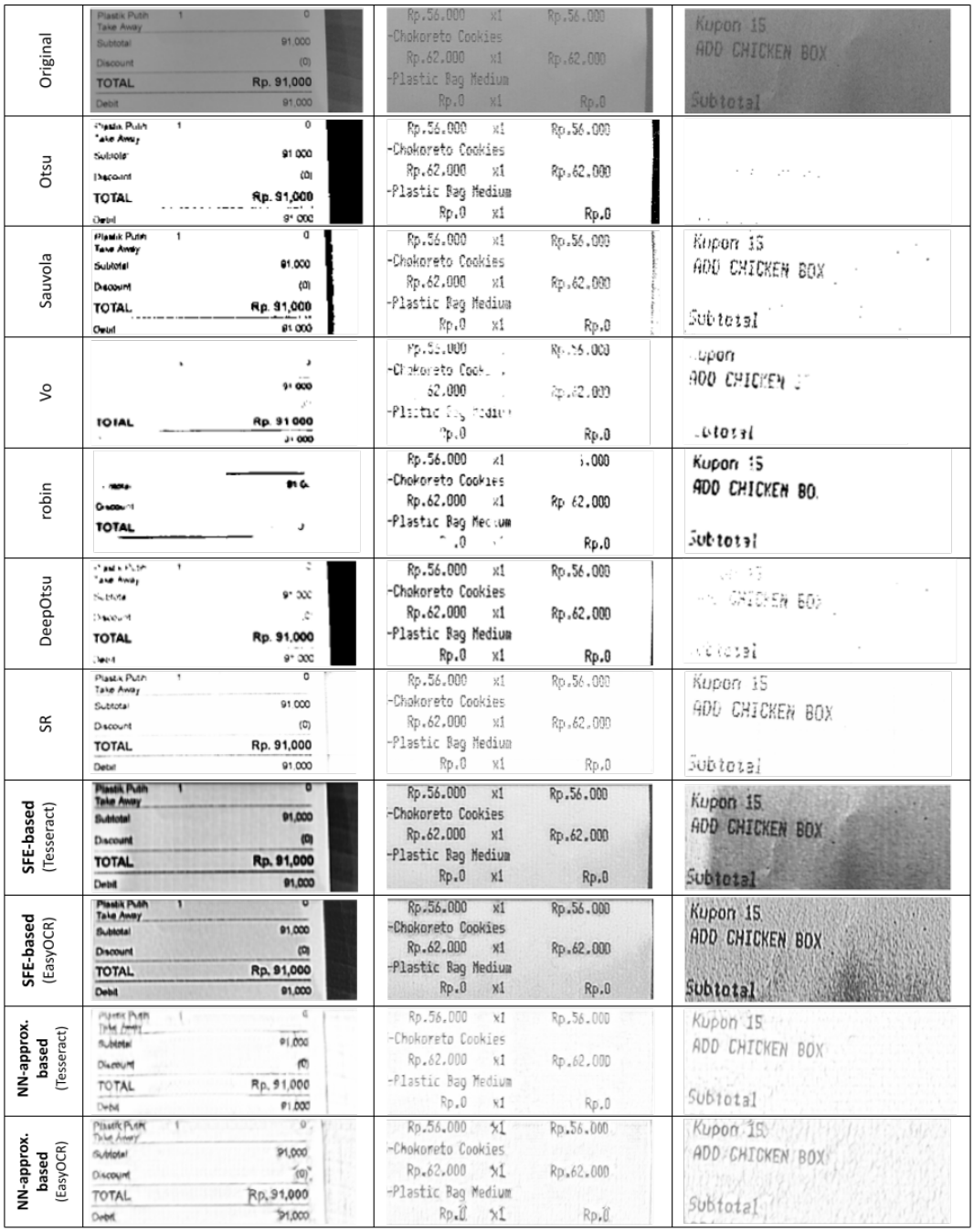

Figure 4.8: Cropped input (POS test dataset) and output images from methods considered in Table 4.4 and our models. The images in three columns have average CER reduction of 6.5, 61 and 73.5 respectively from left to right. CER is based on our NN-based models.

Table 4.4: OCR engine accuracy on POS dataset: comparison with other preprocessing methods.

Preprocessing Method	Tesseract		EasyOCR	
	Accuracy ↑	CER ↓	Accuracy ↑	CER ↓
OCR *(no preprocessing)*	54.51%	26.33	29.69%	44.27
Otsu [40]	50.98%	29.84	16.96%	52.30
Sauvola [50]	55.39%	25.20	20.19%	48.49
Vo [55]	51.72%	31.81	21.95%	50.07
robin	57.18%	28.45	27.59%	43.55
DeepOtsu [18]	62.47%	21.88	26.33%	42.63
SR [43]	67.13%	15.90	37.51%	31.11
SFE-based	**69.17%**	**16.62**	**46.63%**	**28.13**
NN-approx. based	**83.36%**	**8.68**	**67.97%**	**16.46**

tasks, and there are datasets for image preprocessing tasks such as binarization. However, to the best of our knowledge, there are no publicly available OCR datasets that contain optimal intermediate images in addition to the ground truth texts. This lack of intersection between two tasks in the context of data renders it challenging to compare our model with other preprocessing techniques.

Due to the lack of clean ground truth images for our dataset, we used pre-trained weights to compare with other learning-based preprocessing methods. Five binarization methods and one SR method are compared against our pre-processors on the POS dataset. The binarization methods Vo [55], DeepOtsu [18] and robin[4] are originally trained with high resolutions images. However, the images in the POS dataset have considerably low resolution compared to the images used for training these models. Therefore, to mitigate the effect of low-resolution, POS dataset images are enlarged by factors of 2 and 3 before binarization. After the binarization, binarized images were reduced back to the original size before presenting them to the OCR engine. In the evaluation, images enlarged by a factor of 2 performed better than the images enlarged by

[4] `https://github.com/masyagin1998/robin`

a factor of 3. Therefore we recorded the accuracy measurements of the images enlarged by a factor of 2. Similarly, with SR method [43], the images are enlarged by a factor of 2 and presented the same enlarged images to the OCR since the objective of this experiment is to test the impact of high resolution on the OCR engine's performance.

According to the results listed in Table 4.4, two methods based on gradient approximation have outperformed all six methods compared. Sauvola, robin, DeepOtsu and SR methods increased the Tesseract accuracy, and the SR method shows the largest improvement. With robin, CER has increased despite the slight accuracy gain. Only the SR method improved EasyOCR accuracy. We have to note that learning-based methods such as Vo, robin, DeepOtsu and SR methods might perform better if trained with our dataset. However, clean target images are rarely available for different document domains. Often document domains are defined by their unique aberrations and degradations. Therefore, to train these methods, clean target images must be manually created based on the original document images. Figure 4.8 depicts the inputs and outputs from the considered method. Severe loss of details can be observed in the outputs produced by the preprocessing methods except for our two methods.

4.2.3 Other Experiments

Table 4.5: OCR accuracy when trained and tested with different engines.

Dataset	OCR used for training	OCR used for testing	Test accuracy ↑	Test CER ↓
POS	Tesseract	EasyOCR	40.44%	31.28
VGG	Tesseract	EasyOCR	47.14%	21.77
POS	EasyOCR	Tesseract	60.94%	21.81
VGG	EasyOCR	Tesseract	21.64%	51.96

Our approach trains an individualized preprocessor specific to a particular OCR engine. To test the effect of having an individualized preprocessor,

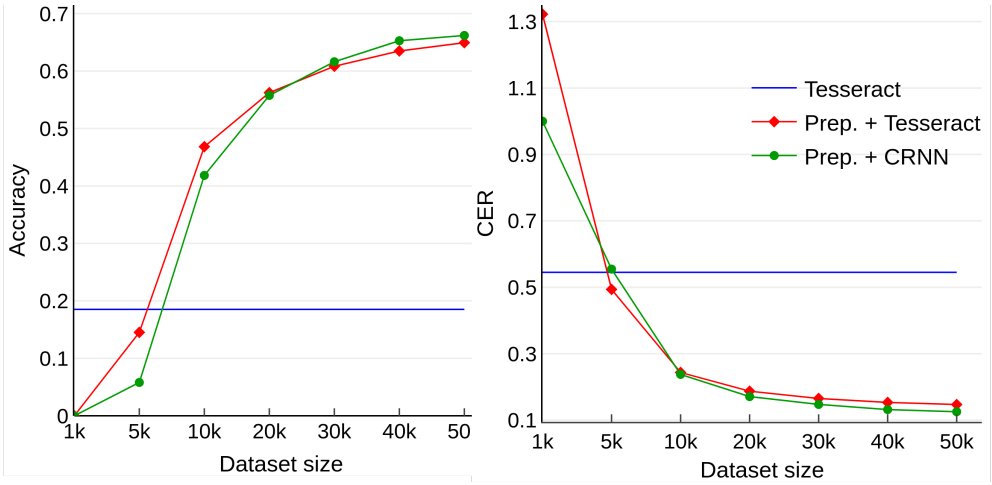

Figure 4.9: Test set accuracy (graph on the left) and CER (graph on the right) of OCR engine and CRNN model with different training sizes of VGG dataset. Tesseract accuracy and CER without preprocessing are added for reference.

we have cross-tested the preprocessors with OCR engines. The preprocessors (NN-approximation based) trained for Tesseract are tested with EasyOCR and vice versa. Table 4.5 depicts the results obtained. After the preprocessing, the OCR engine yields better accuracy than the baseline. owever, the accuracy gain is lower than the accuracy obtained by the same OCR engine the preprocessor has trained with. Therefore its is reasonable to assume that preprocessing has added OCR engine specific artifacts and mutations to the image to improve recognition. This behaviour confirms that different OCR engines expect inputs to be optimized differently, thus individualized preprocessing serves better than generic preprocessing.

Figure 4.9 shows that the CRNN model has well approximated the OCR engine's recognition capability with different dataset sizes. However, note that the CRNN model can only recognize text but does not have text detection capabilities; it cannot fully replace the OCR engine. Additionally, in Figure 4.9, it can be seen that with only 10k samples of the VGG dataset, our preprocessor was able to improve Tesseract significantly. Figure 4.10 depicts the change of accuracy and training time with the increase of the number of image perturbations (n in Algorithm 2) in the SFE-based training pipeline. With the increase of image perturbations, CER has gone down in a nearly linear

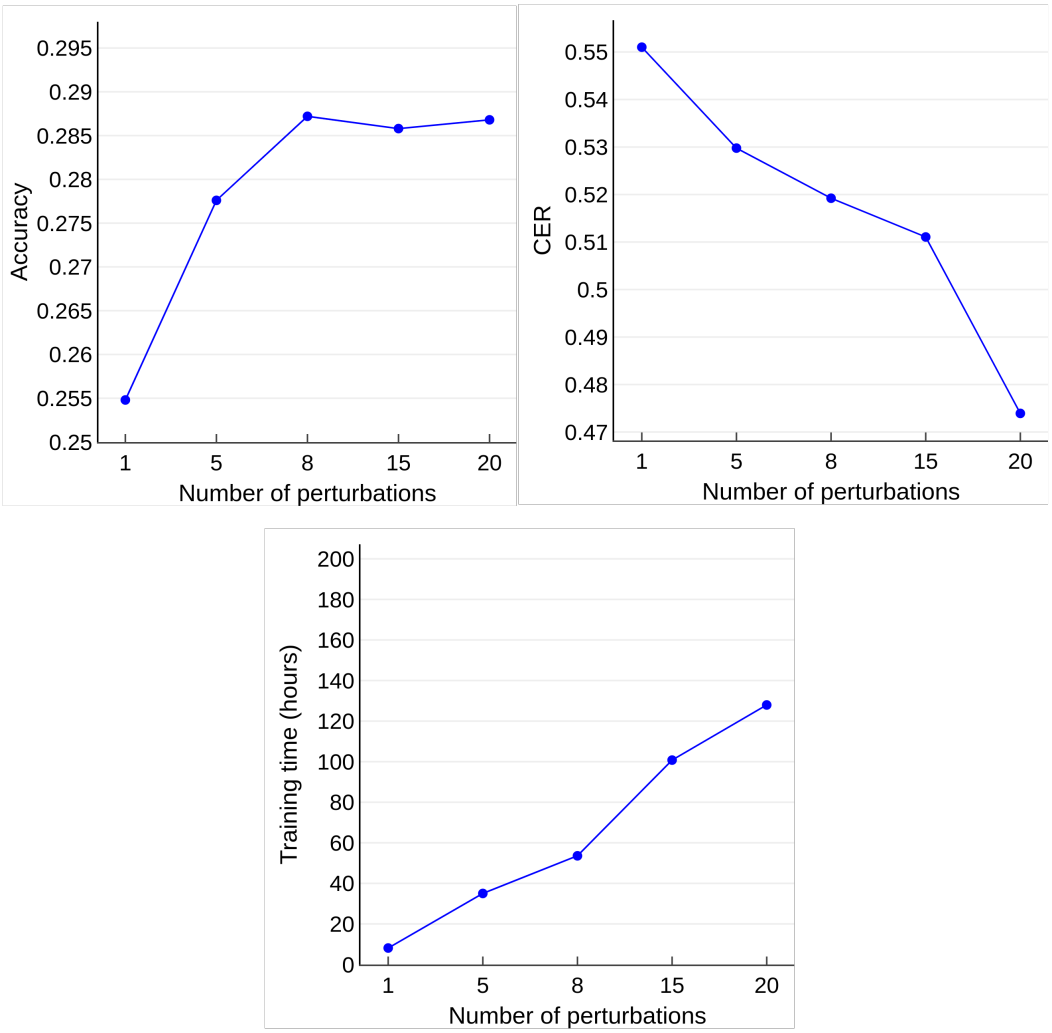

Figure 4.10: VGG test set accuracy (graph on the left) CER (graph on the right) and the training time (graph on the bottom) variation of SFE-preprocessor with the different number of perturbations (n in Algorithm 2).

manner even though the accuracy has not changed significantly after eight perturbations. Therefore it can be assumed that the increase of perturbations might lead to better results. Training time has also increased linearly with the number of image perturbations. Therefore, increasing n might not be a pragmatic solution.

4.2.4 Assumptions and Shortcomings

Both avenues we investigated along our hypothesis were able to improve the accuracy of the considered OCR engines. The recognized downside of our

approach is the added complexity in the training regimen and the increased number of hyperparameters, and the need for ground truth text with bounding boxes. Furthermore, we have observed that if the OCR engine does not perform at all for a given dataset, the NN-approximation-based method does not converge or improve the input image. In the implementation of NN-based gradient approximation, our approximator only approximates the text recognition capability of the OCR engine. Therefore, an assumption made in this approach is that the preprocessing does not negatively affect other OCR engine components such as text detection. Additionally, in the NN-based gradient approximation method, training can be unstable due to the concatenation of different models. However, in our experiments, it is clear that the NN-based gradient approximation handles high dimensional input better than the SFE reparameterization we implemented.

Chapter 5

Conclusion and Future Work

In this thesis, we have proposed a novel training pipeline to create an individualized preprocessor to improve the accuracy of existing OCR solutions, including the commercial unknow-box OCR engines. Two different gradient approximation approaches: an NN-based approach and an SFE based approach, were implemented and tested. Both training pipelines were able to produce document image preprocessors that improve the performance of two OCR engines on two different datasets, demonstrating the power and versatility of the proposed preprocessing method. The preprocessors trained on the NN-approximation based training pipeline performed better than those trained on the SFE-based training pipeline. Our preprocessing method was able to improve the OCR accuracy up to 46% from the baseline accuracy.

Both implementations of our approach of training a preprocessor based on OCR feedback were able to outperform (within assumed conditions) the established preprocessing techniques as we have hypothesized. Even though our approach has produced promising results, it has a few undesirable properties that might hinder its usage in a production environment. The relatively large number of hyperparameters and the added complexity of connecting different models is an inherent drawback of this approach. Further, our current implementations need to sample from the OCR engine a considerably large number of times depending on the dataset size. This sampling can be undesirable in a cloud SaaS setting where it is billed per request since a single training session can be significantly costly. When comparing our two approaches,

NN-approximation based training has a higher sample efficiency than the SFE-based training. However, it still can be costly, and therefore we think that an extension of this project that focuses on sample efficiency as well as accuracy would be an exciting area to investigate. In this work, we mainly focused on 'online training' where the training pipeline samples from the unknow-box constantly; however, a system focusing on sample efficiency can investigate to improve 'offline training' where constant access to the unknown box is not required.

In our current implementation of NN-approximation based method, we only approximate the character recognition ability of the OCR engine. Expanding the capacity of the approximation model may lead to better outcomes, and if the approximator is able to detect text, it will eliminate the need for bounding boxes. Therefore, as future work, different models can be considered as the approximator to expand approximation capabilities. Our preprocessor can apply numerous unconventional pixel-level modifications to the document images to improve the OCR accuracy. Even though slight skew corrections are visible in some cases, our training does not focus on geometric corrections. Therefore, parameterized geometric corrections can be suggested as another future extension of this work. Additionally, different preprocessor architectures can be investigated in the same setting to see whether they yield better results than the models considered here.

When it comes to the SFE-based training paradigm, there is a bottleneck at the OCR engine due to the large number of sampling from OCR required for the image perturbations. Even though the reparameterization used in SFE-based training can be efficiently parallelized, accessing an OCR engine with multiple threads may be problematic. Even in a fully parallel environment, the inherent problem of low sample efficiency in the SFE-based method will not be resolved.

Based on the success of the training approach discussed in this thesis, the same approach can be extended to other image processing tasks, which attempt to improve the output of a secondary function that is non-differentiable. Especially this approach can be used to train preprocessors to preprocess images

for classic image processing algorithms, which are non-differentiable. Even though NN-approximation based training pipeline performed well, many aspects of this training pipeline were established by a trial and error approach. However, a mathematical investigation of this approach may reveal a more systematic approach to implementing this type of training pipelines. I.e. a lower bound of model complexity necessary to approximate the gradient of a larger complex model effectively. Therefore we think that a mathematical investigation of this approach would be an exciting avenue of future work.

References

[1] M. Ahmed and R. Ward, "A rotation invariant rule-based thinning algorithm for character recognition," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 24, no. 12, pp. 1672–1678, 2002. DOI: 10.1109/TPAMI.2002.1114862.

[2] C. Artaud, N. Sidère, A. Doucet, J. Ogier, and V. P. D. Yooz, "Find it! fraud detection contest report," in *2018 24th International Conference on Pattern Recognition (ICPR)*, 2018, pp. 13–18. DOI: 10.1109/ICPR.2018.8545428.

[3] Y. Baek, B. Lee, D. Han, S. Yun, and H. Lee, "Character region awareness for text detection," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, 2019.

[4] Y. Bengio, N. Léonard, and A. C. Courville, "Estimating or propagating gradients through stochastic neurons for conditional computation," *ArXiv*, vol. abs/1308.3432, 2013.

[5] M. Buda, A. Saha, and M. A. Mazurowski, "Association of genomic subtypes of lower-grade gliomas with shape features automatically extracted by a deep learning algorithm," *Computers in Biology and Medicine*, vol. 109, pp. 218–225, 2019, ISSN: 0010-4825. DOI: https://doi.org/10.1016/j.compbiomed.2019.05.002. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0010482519301520.

[6] Q. A. Bui, D. Mollard, and S. Tabbone, "Selecting automatically preprocessing methods to improve ocr performances," in *2017 14th IAPR International Conference on Document Analysis and Recognition (ICDAR)*, vol. 01, 2017, pp. 169–174. DOI: 10.1109/ICDAR.2017.36.

[7] M. S. Chang, S. M. Kang, W. S. Rho, H. G. Kim, and D. J. Kim, "Improved binarization algorithm for document image by histogram and edge detection," in *Proceedings of the International Conference on Document Analysis and Recognition, ICDAR*, vol. 2, IEEE Computer Society, 1995, pp. 636–639, ISBN: 0818671289. DOI: 10.1109/ICDAR.1995.601976.

[8] Q. Chen, Q. sen Sun, P. A. Heng, and D. shen Xia, "A double-threshold image binarization method based on edge detector," *Pattern Recognition*, vol. 41, pp. 1254–1267, 4 Apr. 2008, ISSN: 00313203. DOI: 10.1016/j.patcog.2007.09.007.

[9] Y. Chen and Y. Shao, "Scene text recognition based on deep learning: A brief survey," in *2019 IEEE 11th International Conference on Communication Software and Networks (ICCSN)*, 2019.

[10] G. Chiron, A. Doucet, M. Coustaty, and J. Moreux, "Icdar2017 competition on post-ocr text correction," in *2017 14th IAPR International Conference on Document Analysis and Recognition (ICDAR)*, vol. 01, 2017, pp. 1423–1428. DOI: 10.1109/ICDAR.2017.232.

[11] G. Cybenko, "Approximation by superpositions of a sigmoidal function," *Mathematics of control, signals and systems*, vol. 2, no. 4, pp. 303–314, 1989.

[12] A. K. Das and B. Chanda, "A fast algorithm for skew detection of document images using morphology," *International Journal on Document Analysis and Recognition*, vol. 4, pp. 109–114, 2 2001, ISSN: 14332833. DOI: 10.1007/PL00010902. [Online]. Available: https://link.springer.com/article/10.1007/PL00010902.

[13] U. Garain, A. Jain, A. Maity, and B. Chanda, "Machine reading of camera-held low quality text images: An ica-based image enhancement approach for improving ocr accuracy," in *2008 19th International Conference on Pattern Recognition*, 2008, pp. 1–4. DOI: 10.1109/ICPR.2008.4761840.

[14] A. Graves and J. Schmidhuber, "Framewise phoneme classification with bidirectional lstm networks," in *Proceedings. 2005 IEEE International Joint Conference on Neural Networks, 2005.*, vol. 4, 2005, 2047–2052 vol. 4. DOI: 10.1109/IJCNN.2005.1556215.

[15] A. Graves, S. Fernández, F. Gomez, and J. Schmidhuber, "Connectionist temporal classification: Labelling unsegmented sequence data with recurrent neural networks," in *Proceedings of the 23rd International Conference on Machine Learning*, ser. ICML '06, Association for Computing Machinery, 2006, ISBN: 1595933832. DOI: 10.1145/1143844.1143891.

[16] A. Hashizume, P.-S. Yeh, and A. Rosenfeld, "A method of detecting the orientation of aligned components," *Pattern Recognition Letters*, vol. 4, no. 2, pp. 125–132, 1986, ISSN: 0167-8655. DOI: https://doi.org/10.1016/0167-8655(86)90034-6. [Online]. Available: https://www.sciencedirect.com/science/article/pii/0167865586900346.

[17] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2016.

[18] S. He and L. Schomaker, "Deepotsu: Document enhancement and binarization using iterative deep learning," *Pattern Recognition*, vol. 91, pp. 379–390, 2019, ISSN: 0031-3203. DOI: https://doi.org/10.1016/j.patcog.2019.01.025. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0031320319300330.

[19] J. R. Hicks and J. J. C. Eby, "Signal processing techniques in commercially available high-speed optical character reading equipment," in *Real-Time Signal Processing II*, T. F. Tao, Ed., vol. 0180, SPIE, Sep. 1979, pp. 205–216. DOI: 10.1117/12.957332.

[20] S. C. Hinds, J. L. Fisher, and D. P. D'Amato, "A document skew detection method using run-length encoding and the hough transform," in *[1990] Proceedings. 10th International Conference on Pattern Recognition*, vol. i, 1990, 464–468 vol.1. DOI: 10.1109/ICPR.1990.118147.

[21] K. Hornik, M. B. Stinchcombe, and H. White, "Universal approximation of an unknown mapping and its derivatives using multilayer feedforward networks," *Neural Networks*, vol. 3, pp. 551–560, 1990.

[22] Z. Huang, K. Chen, J. He, X. Bai, D. Karatzas, S. Lu, and C. V. Jawahar, "Icdar2019 competition on scanned receipt ocr and information extraction," in *2019 International Conference on Document Analysis and Recognition (ICDAR)*, 2019, pp. 1516–1520. DOI: 10.1109/ICDAR.2019.00244.

[23] A. Jacovi, G. Hadash, E. Kermany, B. Carmeli, O. Lavi, G. Kour, and J. Berant, "Neural network gradient-based learning of black-box function interfaces," in *International Conference on Learning Representations*, 2019. [Online]. Available: https://openreview.net/forum?id=r1e13s05YX.

[24] M. Jaderberg, W. M. Czarnecki, S. Osindero, O. Vinyals, A. Graves, D. Silver, and K. Kavukcuoglu, "Decoupled neural interfaces using synthetic gradients," in *Proceedings of the 34th International Conference on Machine Learning*, D. Precup and Y. W. Teh, Eds., ser. Proceedings of Machine Learning Research, vol. 70, International Convention Centre, Sydney, Australia: PMLR, 2017, pp. 1627–1635. [Online]. Available: http://proceedings.mlr.press/v70/jaderberg17a.html.

[25] M. Jaderberg, K. Simonyan, A. Vedaldi, and A. Zisserman, "Synthetic data and artificial neural networks for natural scene text recognition," in *Workshop on Deep Learning, NIPS*, 2014.

[26] Y. S. Kim, W. S. Choi, and S. W. Kim, "High-speed thinning processor for character recognition system," *IEEE Transactions on Consumer Electronics*, vol. 38, no. 4, pp. 762–766, 1992. DOI: 10.1109/30.179963.

[27] L. Lam and C. Y. Suen, "An evaluation of parallel thinning algorithms for character recognition," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 17, pp. 914–919, 8 1995, ISSN: 01628828. DOI: 10.1109/34.406659.

[28] A. Lat and C. V. Jawahar, "Enhancing ocr accuracy with super resolution," in *2018 24th International Conference on Pattern Recognition (ICPR)*, 2018, pp. 3162–3167. DOI: 10.1109/ICPR.2018.8545609.

[29] D. S. Le, G. R. Thoma, and H. Wechsler, "Automated page orientation and skew angle detection for binary document images," *Pattern Recognition*, vol. 27, no. 10, pp. 1325–1344, 1994, ISSN: 0031-3203. DOI: https://doi.org/10.1016/0031-3203(94)90068-X. [Online]. Available: https://www.sciencedirect.com/science/article/pii/003132039490068X.

[30] C. Ledig, L. Theis, F. Huszar, J. Caballero, A. Cunningham, A. Acosta, A. Aitken, A. Tejani, J. Totz, Z. Wang, and W. Shi, "Photo-realistic single image super-resolution using a generative adversarial network," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2017.

[31] V. I. Levenshtein, "Binary codes capable of correcting deletions, insertions, and reversals," *Soviet physics. Doklady*, vol. 10, pp. 707–710, 1965.

[32] T. P. Lillicrap, J. J. Hunt, A. Pritzel, N. Heess, T. Erez, Y. Tassa, D. Silver, and D. Wierstra, "Continuous control with deep reinforcement learning," *arXiv preprint arXiv:1509.02971*, 2015.

[33] Y. Liu, R. Feinrich, and S. N. Srihari, "An object attribute thresholding algorithm for document image binarization," in *Proceedings of 2nd International Conference on Document Analysis and Recognition (ICDAR '93)*, 1993, pp. 278–281. DOI: 10.1109/ICDAR.1993.395732.

[34] X. Mao, C. Shen, and Y.-B. Yang, "Image restoration using very deep convolutional encoder-decoder networks with symmetric skip connections," in *Advances in Neural Information Processing Systems*, D. Lee, M. Sugiyama, U. Luxburg, I. Guyon, and R. Garnett, Eds., vol. 29, Curran Associates, Inc., 2016. [Online]. Available: https://proceedings.neurips.cc/paper/2016/file/0ed9422357395a0d4879191c66f4faa2-Paper.pdf.

[35] S. Mohamed, M. Rosca, M. Figurnov, and A. Mnih, "Monte carlo gradient estimation in machine learning," *Journal of Machine Learning Research*, vol. 21, pp. 1–63, Jun. 2019. [Online]. Available: http://arxiv.org/abs/1906.10652.

[36] N. M. Nguyen and N. Ray, "End-to-end learning of convolutional neural net and dynamic programming for left ventricle segmentation," in *Proceedings of the Third Conference on Medical Imaging with Deep Learning*, ser. Proceedings of Machine Learning Research, vol. 121, PMLR, 2020, pp. 555–569.

[37] W. Niblack, *An Introduction to Digital Image Processing*, ser. Delaware Symposia on Language Studies5. Prentice-Hall International, 1986, ISBN: 9780134806747.

[38] L. O'Gorman, "The document spectrum for page layout analysis," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 15, no. 11, pp. 1162–1173, 1993. DOI: 10.1109/34.244677.

[39] L. Ogorman, "Binarization and multithresholding of document images using connectivity," *CVGIP: Graphical Models and Image Processing*, vol. 56, pp. 494–506, 6 Nov. 1994, ISSN: 10499652. DOI: 10.1006/cgip.1994.1044.

[40] N. Otsu, "A threshold selection method from gray-level histograms," *IEEE transactions on systems, man, and cybernetics*, vol. 9, pp. 62–66, 1 1979.

[41] S. Park, S. Shin, B. Lee, J. Lee, J. Surh, M. Seo, and H. Lee, "Cord: A consolidated receipt dataset for post-ocr parsing," in *Workshop on Document Intelligence at NeurIPS 2019*, 2019. [Online]. Available: https://openreview.net/forum?id=SJl3z659UH.

[42] A. Paszke, S. Gross, F. Massa, A. Lerer, J. Bradbury, G. Chanan, T. Killeen, Z. Lin, N. Gimelshein, L. Antiga, A. Desmaison, A. Kopf, E. Yang, Z. DeVito, M. Raison, A. Tejani, S. Chilamkurthy, B. Steiner, L. Fang, J. Bai, and S. Chintala, "Pytorch: An imperative style, high-performance deep learning library," in *Advances in Neural Information Processing Systems 32*, H. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché-Buc, E. Fox, and R. Garnett, Eds., Curran Associates, Inc., 2019, pp. 8024–8035. [Online]. Available: http://papers.neurips.cc/paper/9015-pytorch-an-imperative-style-high-performance-deep-learning-library.pdf.

[43] X. Peng and C. Wang, "Building super-resolution image generator for ocr accuracy improvement," in *Document Analysis Systems*, X. Bai, D. Karatzas, and D. Lopresti, Eds., Cham: Springer International Publishing, 2020, pp. 145–160, ISBN: 978-3-030-57058-3.

[44] Y. Rangoni, F. Shafait, and T. M. Breuel, "Ocr based thresholding.," in *MVA*, 2009, pp. 98–101.

[45] C. Reul, U. Springmann, C. Wick, and F. Puppe, "Improving ocr accuracy on early printed books by utilizing cross fold training and voting," in *2018 13th IAPR International Workshop on Document Analysis Systems (DAS)*, 2018, pp. 423–428. DOI: 10.1109/DAS.2018.30.

[46] O. Ronneberger, P. Fischer, and T. Brox, "U-net: Convolutional networks for biomedical image segmentation," in *Medical Image Computing and Computer-Assisted Intervention – MICCAI 2015*, Springer International Publishing, 2015, pp. 234–241, ISBN: 978-3-319-24574-4.

[47] F. J. R. Ruiz, M. K. Titsias, and D. M. Blei, *The generalized reparameterization gradient*, 2016. arXiv: 1610.02287 [stat.ML].

[48] P. K. Saha, G. Borgefors, and G. Sanniti di Baja, "Chapter 1 - skeletonization and its applications – a review," in *Skeletonization*, P. K. Saha, G. Borgefors, and G. Sanniti di Baja, Eds., Academic Press, 2017, pp. 3–42, ISBN: 978-0-08-101291-8. DOI: 10.1016/B978-0-08-101291-8.00002-X.

[49] T. Salimans, J. Ho, X. Chen, S. Sidor, and I. Sutskever, *Evolution strategies as a scalable alternative to reinforcement learning*, 2017. arXiv: 1703.03864 [stat.ML].

[50] J. Sauvola and M. Pietikäinen, "Adaptive document image binarization," *Pattern Recognition*, vol. 33, pp. 225–236, 2 Feb. 2000, ISSN: 00313203. DOI: 10.1016/S0031-3203(99)00055-2.

[51] 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. DOI: 10.1109/TPAMI.2016.2646371.

[52] D. Sporici, E. Cuşnir, and C. A. Boiangiu, "Improving the accuracy of tesseract 4.0 ocr engine using convolution-based preprocessing," *Symmetry*, vol. 12, p. 715, 5 May 2020, ISSN: 20738994. DOI: 10.3390/SYM12050715.

[53] S. N. Srihari and V. Govindaraju, "Analysis of textual images using the hough transform," *Machine vision and Applications*, vol. 2, no. 3, pp. 141–153, 1989.

[54] P. Thompson, J. McNaught, and S. Ananiadou, "Customised ocr correction for historical medical text," in *2015 Digital Heritage*, vol. 1, 2015, pp. 35–42. DOI: 10.1109/DigitalHeritage.2015.7413829.

[55] Q. N. Vo, S. H. Kim, H. J. Yang, and G. Lee, "Binarization of degraded document images based on hierarchical deep supervised network," *Pattern Recognition*, vol. 74, pp. 568–586, Feb. 2018, ISSN: 00313203. DOI: 10.1016/j.patcog.2017.08.025.

[56] J. M. White and G. D. Rohrer, "Image thresholding for optical character recognition and other applications requiring character image extraction," *IBM Journal of Research and Development*, vol. 27, no. 4, pp. 400–411, 1983. DOI: 10.1147/rd.274.0400.

[57] D. Wierstra, T. Schaul, T. Glasmachers, Y. Sun, and J. Schmidhuber, *Natural evolution strategies*, 2011. arXiv: 1106.4487 [stat.ML].

[58] R. J. Williams, "Simple statistical gradient-following algorithms for connectionist reinforcement learning," *Machine Learning*, vol. 8, pp. 229–256, 3-4 May 1992, ISSN: 0885-6125. DOI: 10.1007/bf00992696.

[59] Xiaoyi Jiang, H. Bunke, and D. Widmer-Kljajo, "Skew detection of document images by focused nearest-neighbor clustering," in *Proceedings of the Fifth International Conference on Document Analysis and Recognition. ICDAR '99 (Cat. No.PR00318)*, 1999, pp. 629–632. DOI: 10.1109/ICDAR.1999.791866.

Appendix A

Hyperparameter Tuning

This section discusses the steps and the process of hyperparameter tuning. For the final training details, refer to Section 4.1.4.

A.1 Hyperparameters of the NN-approximation Training Pipeline

Table A.1 lists the hyperparameters involved in the main training pipeline of the NN-approximation based preprocessor. CRNN model (approximator) is pre-trained before adding it to the main pipeline to avoid the cold-start problem. Therefore, in addition to the parameters listed in Table A.1, the CRNN model can be considered another hyperparameter defined by the pre-training process. The pre-training process involves general hyperparameters such as learning rate, number of epochs and batch size. Additionally, similar to the main training pipeline, the standard deviation is considered a hyperparameter in noise generation of the pre-training of the CRNN model. Due to this relatively large number of hyperparameters involved in the training process and the unclear nature of the relationship between parameters, an ad hoc approach is followed in the hyperparameter tuning process.

Our initial experiments were conducted with SRGAN inspired preprocessor model, Tesseract and the POS dataset. A smaller subset of the POS training dataset is used with a smaller number of epochs to reduce the training time. As the first step, a set of CRNN models were trained with different hyperparameter settings. Table A.2 lists the hyperparameter values of the CRNN

Table A.1: Hyperparameters used in the training process of NN-approximation based method.

Hyperparameter	Explanation
Standard deviation (σ in Algorithm 1)	Either a fixed standard deviation or random standard deviation is used in the experiments to generate noise. In the case of random standard deviation, the standard deviation is selected from a discrete uniform distribution containing a few values. In the experiments, in the fixed case, 0.02 is used and in the random case following values are used: $[0, 0.01, 0.02, 0.03, 0.04, 0.05]$.
Epochs	The number of training epochs.
CRNN learning rate	The learning rate used for the CRNN model (Approximator) when training the preprocessor. In our experiments, initial training of the CRNN is also conducted with the same learning rate.
Prep. learning rate	The learning rate used for the preprocessor model.
Inner limit (S in Algorithm 1)	The number iterations in the inner loop in Algorithm 1.
Scalar (β in equation 3.5)	The scalar value used in the loss function (3.5) to control the effect of MSE loss.
Batch size	The number of images in a single batch.

models selected to be included in the preprocessor training pipeline. Our preliminary studies with models led to the learning rates 0.0001 and 0.00001 for the approximator and the preprocessor, respectively. Therefore our first set of experiments were conducted to determine a batch size. Experiments were conducted with 18 epochs and later 50 epochs. Table A.3 lists the hyperparameters and the accuracy levels of these experiments. According to Table A.3, smaller batch sizes performed better.

The second set of experiments were conducted to determine an inner limit (S in Algorithm 1) value. Table A.4 lists the hyperparameters, and the accuracy levels and inner limits 1 (batch size 2) and 5 (batch size 1) performed

Table A.2: Hyperparameters and accuracy levels of the CRNN models selected to train the preprocessor. $[0.01 - 0.05]$ indicates a uniform distribution containing following values: $[0, 0.01, 0.02, 0.03, 0.04, 0.05]$

CRNN model No.	Standard deviation	Epochs	Learning rate	Acc. compared to ground truth	Acc. compared to OCR output
1	0.02	15	0.0001	54%	56%
2	$[0 - 0.05]$	15	0.0001	57%	57%
3	$[0 - 0.05]$	50	0.0001	57%	60%

Table A.3: Hyperparameters and accuracies of the first set of experiments conducted. The column 'CRNN model' corresponds to the column 'CRNN model No.' in Table A.2.

CRNN model	Std.	Epochs	CRNN l.r.	Prep. l.r.	Inner limit	Scalar	Batch size	Acc.
1	0.02	18	0.0001	0.00001	3	0.5	5	71.18%
1	0.02	18	0.0001	0.00001	3	0.5	2	71.55%
1	0.02	18	0.0001	0.00001	3	0.5	1	72.52%
1	0.02	50	0.0001	0.00001	3	0.5	5	74.15%
1	0.02	50	0.0001	0.00001	3	0.5	2	76.74%
1	0.02	50	0.0001	0.00001	3	0.5	1	77.66%

better than 3. Since a higher inner limit increases the training time significantly and the accuracy difference is minimal between 1 and 5, the inner limit value is set to 1 and batch size is set to 2 for the next set of experiments. The third set of experiments listed in Table A.5 aimed to determine a value for the scalar (β in equation 3.5). The highest accuracies were produced by the values 0.1, 0, 0.15 and 1. A reasonable range for the scalar value does not appear in these results.

In the fourth set of experiments, the standard deviation of the noise is changed from fixed to stochastic by randomly choosing the σ value from a list of values. The CRNN model is pre-trained using a similar standard deviation for the noise. Staying within the previous experiments' observations,

Table A.4: Hyperparameters and accuracies of the second set of experiments conducted. The column 'CRNN model' corresponds to the column 'CRNN model No.' in Table A.2.

CRNN model	Std.	Epochs	CRNN l.r.	Prep. l.r.	Inner limit	Scalar	Batch size	Acc.
1	0.02	25	0.0001	0.00001	1	0.5	2	73.86%
1	0.02	25	0.0001	0.00001	1	0.5	1	72.15%
1	0.02	25	0.0001	0.00001	3	0.5	2	73.30%
1	0.02	25	0.0001	0.00001	3	0.5	1	73.51%
1	0.02	25	0.0001	0.00001	5	0.5	2	70.56%
1	0.02	25	0.0001	0.00001	5	0.5	1	73.78%

Table A.5: Hyperparameters and accuracies of the third set of experiments conducted. The column 'CRNN model' corresponds to the column 'CRNN model No.' in Table A.2.

CRNN model	Std.	Epochs	CRNN l.r.	Prep. l.r.	Inner limit	Scalar	Batch size	Acc.
1	0.02	25	0.0001	0.00001	1	0	2	74.05%
1	0.02	25	0.0001	0.00001	1	0.1	2	74.85%
1	0.02	25	0.0001	0.00001	1	0.15	2	73.82%
1	0.02	25	0.0001	0.00001	1	0.2	2	72.52%
1	0.02	25	0.0001	0.00001	1	0.4	2	72.85%
1	0.02	25	0.0001	0.00001	1	0.6	2	72.07%
1	0.02	25	0.0001	0.00001	1	1	2	73.73%

slight deviations were added to the learning rates, scalar value, and batch size in the experiments. Table A.6 lists the hyperparameter values and the accuracies of this set of experiments. The best performing model produced an accuracy of 78.04%. As the next step of experimentation, a U-Net model was trained using the same parameters as the best performing SRGAN model in the fourth experiment set. It produced an accuracy of 79.66%. Therefore a new set of experiments were conducted with the U-Net model by adding slight modifications to the hyperparameter values. Table A.7 lists the results of both SRGAN and U-Net models trained with final hyperparameter values derived for the U-Net model. A small subset of experiments was conducted with the

Table A.6: Hyperparameters and accuracies of the fourth set of experiments conducted. The column 'CRNN model' corresponds to the column 'CRNN model No.' in Table A.2. $[0.01 - 0.05]$ indicates a uniform distribution containing following values: $[0, 0.01, 0.02, 0.03, 0.04, 0.05]$

CRNN model	Std.	Epochs	CRNN l.r.	Prep. l.r.	Inner limit	Scalar	Batch size	Acc.
2	$[0 - 0.05]$	50	0.0001	0.00001	1	0.1	2	75.42%
2	$[0 - 0.05]$	50	0.0001	0.00001	3	0.5	1	77.41%
2	$[0 - 0.05]$	50	0.0001	0.00001	3	0.5	2	73.88%
2	$[0 - 0.05]$	50	0.00001	0.00001	5	0.5	1	74.53%
2	$[0 - 0.05]$	50	0.0001	0.00005	1	0.5	1	77.71%
2	$[0 - 0.05]$	50	0.0001	0.00005	2	0.5	1	77.68%
2	$[0 - 0.05]$	50	0.0001	0.0001	1	0.6	1	76.09%
2	$[0 - 0.05]$	50	0.0001	0.00005	2	0.7	1	78.04%

Table A.7: Final Hyperparameter values determined for the U-Net model. Both U-net and SRGAN models were tested with the same set of hyperparameters. The column 'CRNN model' corresponds to the column 'CRNN model No.' in Table A.2. $[0.01 - 0.05]$ indicates a uniform distribution containing following values: $[0, 0.01, 0.02, 0.03, 0.04, 0.05]$

Prep. model	CRNN model	Std.	Epochs	CRNN l.r.	Prep. l.r.	Inner limit	Scalar	Batch size	Acc.
U-Net	3	$[0 - 0.05]$	50	0.0001	0.0001	2	1	1	80.47%
SRGAN	3	$[0 - 0.05]$	50	0.0001	0.00005	2	1	1	78.27%

OCR engine, EasyOCR as well. It was observed that the same hyperparameter values worked well with EasyOCR. Additionally, the same hyperparameters were used to train the preprocessor with the VGG dataset.

A.2 Hyperparameters of the SFE-based Training Pipeline

The hyperparameters preprocessor learning rate, batch size, scalar and epochs of the SFE-based method are similar to the NN-approximation method in their application. The number of perturbations (n) mentioned in Algorithm 2 is somewhat analogous to the inner limit (S) in Algorithm 1. Therefore, experiments of the SFE method started with using the final hyperparameter

values listed in Table A.7 where applicable. A set of experiments similar to the NN-approximation based method was conducted to determine the SFE-based training pipeline's hyperparameters. However, due to the significant training time required, experiments were limited to a smaller number of tests compared to the NN-approximation method. The final training details and the hyperparameters are discussed in section 4.1.4.