Machine Learning, Image Processing, and Transfer Learning for Handwritten Spreadsheet Digitization

A Major Qualifying Project (MQP) Report
Submitted to the Faculty of
WORCESTER POLYTECHNIC INSTITUTE
in partial fulfillment of the requirements
for the Degree of Bachelor of Science in

Computer Science,
Data Science,
and Mathematical Sciences

By:
Matthew Haley
Liam Hall
Christopher Langevin
Cameron Norton
Harsh Patel
Elliot Trilling

Project Advisors:
Professor Oren Mangoubi
Professor Randy Paffenroth

Sponsored By:
Professor Gregory Noetscher, U.S. Army Natick Soldier Systems Center

Date: April 2024

Abstract

This report details the development of a software program aimed at converting handwritten parachute data records from the Natick Army Lab into an analyzable digital format. Due to the failure of traditional OCR models to recognize handwritten fractions, a crucial part of the data, we generated a synthetic dataset to specifically fine-tune these models. The software utilizes image processing and OCR technologies to translate text and replicate the physical document's layout in a digital format. This innovation streamlines data analysis, enhancing the Army's ability to monitor and understand parachute integrity and lifecycle.

Acknowledgements

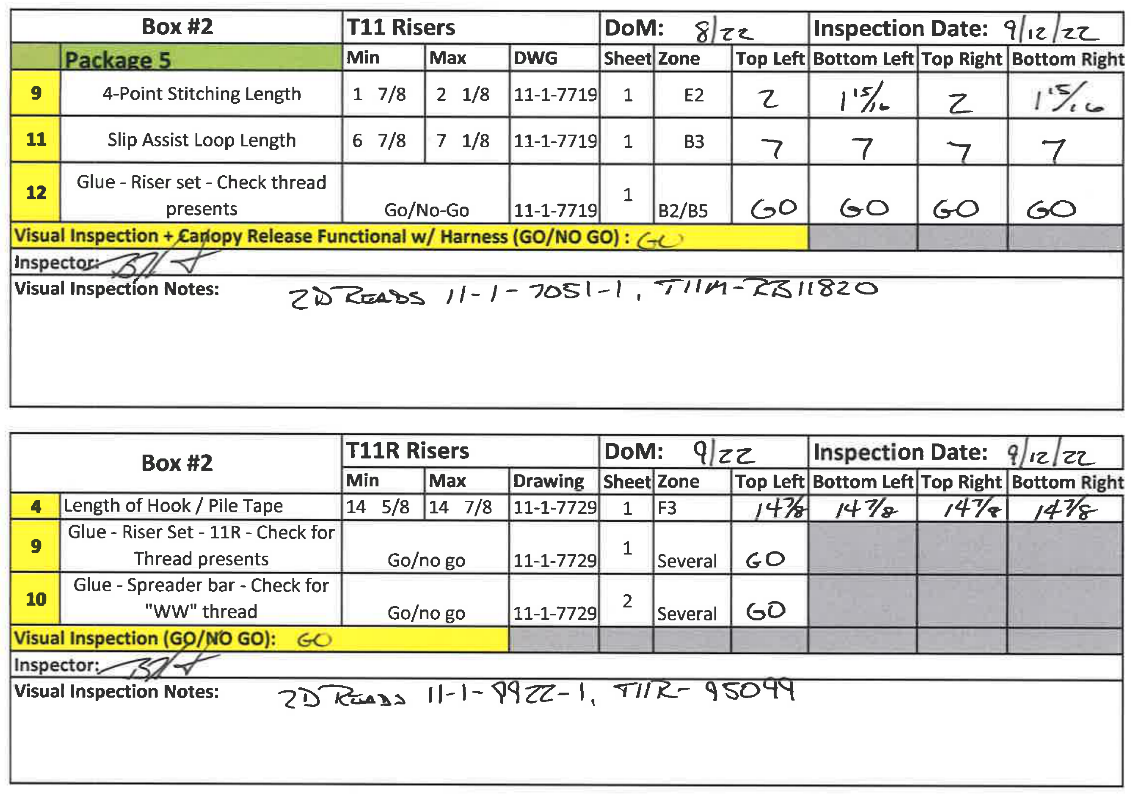
Our project would not have been possible without the help of our sponsors and advisors. We would like to thank Dr. Greg Noetscher for his guidance in helping us understanding the requirements of this project. We are grateful to Natick Army Research Labs for allowing us to work on an impactful project. We would also like to thank our advisors, Professor Oren Mangoubi and Professor Randy Paffenroth. Both advisors shared their expertise in Math, Data Science, and Computer Science. Lastly, we appreciate the departments of Mathematical Sciences and Computer Science at Worcester Polytechnic Institute for providing the educational foundation that prepared us for this project through a diverse range of classes and coursework during our undergraduate studies.

Contents

List of Figures

1 Introduction

The US Army Natick Soldier Systems Center (NSSC or DEVCOM), which is also known as "Natick Labs", is located in Natick, Massachusetts. NSSC focuses on a multitude of technologies, including the T-11 parachute [NSC, 2024]. In the US Army, T-11 parachutes are critical for a multitude of tasks. These parachutes can transport both personnel and supplies from the sky to the ground. Every time a T-11 parachute is put into use, it is then examined by an officer in the army, using a sheet like the one shown below. Knowing how much wear and tear has been sustained by particular parachutes is paramount to their success in performing safely. With the number of parachutes in the inventory of DEVCOM, or the U.S. Army Combat Capabilities Development Command, there are many of these physical data sheets [U.S. Army Combat Capabilities Development Command, 2023].



Box #2		T11 Risers			DoM: 8/22		Inspection Date: 9/12/22			
	Package 5	Min	Max	DWG	Sheet	Zone	Top Left	Bottom Left	Top Right	Bottom Right
9	4-Point Stitching Length	1 7/8	2 1/8	11-1-7719	1	E2	2	1 15/16	2	1 15/16
11	Slip Assist Loop Length	6 7/8	7 1/8	11-1-7719	1	B3	7	7	7	7
12	Glue - Riser set - Check thread presents	Go/No-Go		11-1-7719	1	B2/B5	GO	GO	GO	GO
Visual Inspection + Canopy Release Functional w/ Harness (GO/NO GO) : GO										
Inspector: [illegible]										
Visual Inspection Notes: 2D READS 11-1-7051-1, T11M-RB11820										

Box #2		T11R Risers			DoM: 9/22		Inspection Date: 9/12/22			
		Min	Max	Drawing	Sheet	Zone	Top Left	Bottom Left	Top Right	Bottom Right
4	Length of Hook / Pile Tape	14 5/8	14 7/8	11-1-7729	1	F3	14 7/8	14 7/8	14 7/8	14 7/8
9	Glue - Riser Set - 11R - Check for Thread presents	Go/no go		11-1-7729	1	Several	GO			
10	Glue - Spreader bar - Check for "WW" thread	Go/no go		11-1-7729	2	Several	GO			
Visual Inspection (GO/NO GO): GO										
Inspector: [illegible]										
Visual Inspection Notes: 2D READS 11-1-8922-1, T11R-95099										

Figure 1: An example parachute datasheet.

While these record sheets contain valuable information, they do not have much value in their

APPROVED FOR PUBLIC RELEASE

physical form as they do not allow for easy analysis. Digitizing them would allow the military to easily track the status and measurements of the parachutes they are working with.

To improve the statistical processing of data gathered by the U.S. Military on their parachutes, our team has endeavored to make a machine learning pipeline to transform scans of documents of parachute data into a digital process that can be more easily processed. We used state-of-the-art machine learning techniques to perform optical character recognition on each sheet and produce a resulting Excel file. We hope this pipeline will enable large speedups in the digitization efforts of these records.

Through this project, we will explore many core principles of machine learning. Our main tool will be neural networks, with a particular focus on the transformer architecture as introduced by the paper "Attention Is All You Need" [Vaswani et al., 2017]. Our team used pre-trained models effectively and fine-tuned them for our specific use case. To prepare the dataset for processing by these models, standard image preprocessing techniques such as resizing, deskewing, grayscaling, blurring, binarization, and morphological transformations were used. Built onto this are the OpenCV contour detection functions, which aided the processing of this parachute dataset which contains many contours (in the many gridlines that are present).

2 Background

2.1 Image Processing

Preprocessing images into a consistent and usable format is a critical first step in training and testing many models. Some relevant image processing techniques for this project include resizing, deskewing, grayscaling, blurring, binarization, and morphological transformations. These techniques represent the groundwork for preparing images for various applications including contour detection and object recognition.

2.1.1 About OpenCV

The Open Source Computer Vision Library, or OpenCV for short, is a library full of machine learning and computer vision software. The library has over 2500 algorithms, including programs that can recognize faces, classify human actions by watching certain videos, track movements of objects, produce 3D points, stitch images together, etc [OpenCV Developers, 2024d]. As the library is Apache 2 licensed, governing bodies are allowed to use this library [Apa, 2004]. As we are creating this model for the military, it was very important that this be the case. In addition, OpenCV has a python interface and is supported for

Windows, Linux, Mac OS, and Android. Since the group did not know what operating system the military uses at the beginning of the project, this made OpenCV a safe bet.

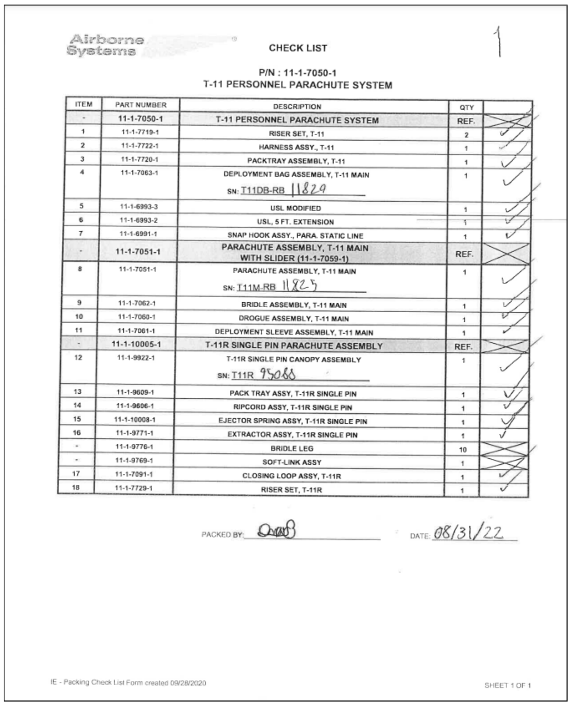
2.1.2 Resizing

The process of resizing an image involves altering its dimensions, either increasing or decreasing its size while maintaining the original aspect ratio. In the realm of image processing, resizing holds critical importance for standardizing image dimensions. In research applications, ensuring uniformity in input data through resizing facilitates consistent and accurate analysis across varied images, providing a stable foundation for subsequent image processing algorithms and methodologies.

2.1.3 Deskewing

Deskewing emerges as a vital technique aimed at rectifying the skewness or non-alignment within images. We use the term "deskewing" to refer to the act of rotating an image so that a maximum number of lines are horizontal or vertical. Skewed images can impede precise text recognition or line detection, affecting the performance of subsequent processing algorithms. Rectifying skewness significantly enhances the accuracy and reliability of analyzing textual or line-based content within images, ensuring more precise and effective interpretation.

We utilized a Python library known as "deskew" to perform deskewing of our images. See 2 for an example.



Airborne Systems

CHECK LIST

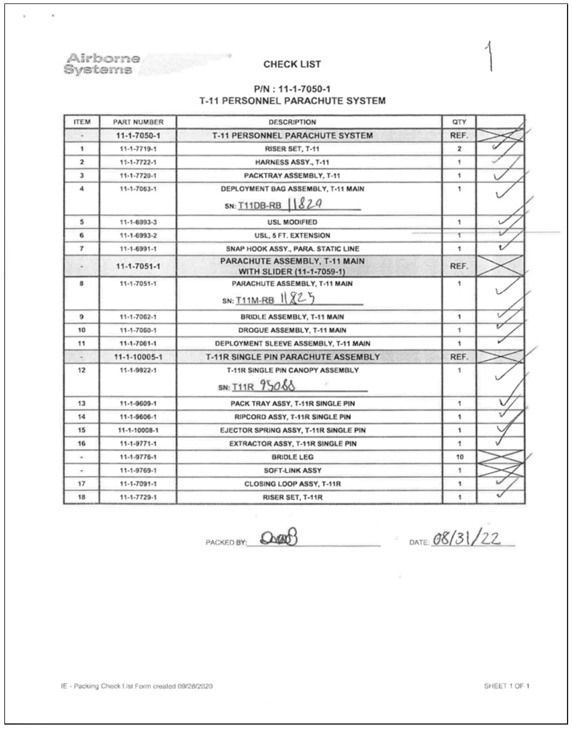
1

P/N : 11-1-7050-1
T-11 PERSONNEL PARACHUTE SYSTEM

ITEM	PART NUMBER	DESCRIPTION	QTY	
-	11-1-7050-1	T-11 PERSONNEL PARACHUTE SYSTEM	REF.	
1	11-1-7719-1	RISER SET, T-11	2	✓
2	11-1-7722-1	HARNESS ASSY., T-11	1	✓
3	11-1-7720-1	PACKTRAY ASSEMBLY, T-11	1	✓
4	11-1-7063-1	DEPLOYMENT BAG ASSEMBLY, T-11 MAIN SN: T11DB-RB 11829	1	✓
5	11-1-6993-3	USL MODIFIED	1	✓
6	11-1-6993-2	USL, 5 FT. EXTENSION	1	✓
7	11-1-6991-1	SNAP HOOK ASSY., PARA. STATIC LINE	1	✓
-	11-1-7051-1	PARACHUTE ASSEMBLY, T-11 MAIN WITH SLIDER (11-1-7059-1)	REF.	
8	11-1-7051-1	PARACHUTE ASSEMBLY, T-11 MAIN SN: T11M-RB 11825	1	✓
9	11-1-7062-1	BRIDLE ASSEMBLY, T-11 MAIN	1	✓
10	11-1-7060-1	DROGUE ASSEMBLY, T-11 MAIN	1	✓
11	11-1-7061-1	DEPLOYMENT SLEEVE ASSEMBLY, T-11 MAIN	1	✓
-	11-1-10005-1	T-11R SINGLE PIN PARACHUTE ASSEMBLY	REF.	
12	11-1-9922-1	T-11R SINGLE PIN CANOPY ASSEMBLY SN: T11R 95088	1	✓
13	11-1-9609-1	PACK TRAY ASSY, T-11R SINGLE PIN	1	✓
14	11-1-9606-1	RIPCORD ASSY, T-11R SINGLE PIN	1	✓
15	11-1-10008-1	EJECTOR SPRING ASSY, T-11R SINGLE PIN	1	✓
16	11-1-9771-1	EXTRACTOR ASSY, T-11R SINGLE PIN	1	✓
-	11-1-9776-1	BRIDLE LEG	10	
-	11-1-9769-1	SOFT-LINK ASSY	1	
17	11-1-7091-1	CLOSING LOOP ASSY, T-11R	1	✓
18	11-1-7729-1	RISER SET, T-11R	1	✓

PACKED BY: [illegible] DATE: 08/31/22

IE - Packing Check List Form created 09/28/2020 SHEET 1 OF 1



Airborne Systems

CHECK LIST

1

P/N : 11-1-7050-1
T-11 PERSONNEL PARACHUTE SYSTEM

ITEM	PART NUMBER	DESCRIPTION	QTY	
-	11-1-7050-1	T-11 PERSONNEL PARACHUTE SYSTEM	REF.	
1	11-1-7719-1	RISER SET, T-11	2	✓
2	11-1-7722-1	HARNESS ASSY., T-11	1	✓
3	11-1-7720-1	PACKTRAY ASSEMBLY, T-11	1	✓
4	11-1-7063-1	DEPLOYMENT BAG ASSEMBLY, T-11 MAIN SN: T11DB-RB 11829	1	✓
5	11-1-6993-3	USL MODIFIED	1	✓
6	11-1-6993-2	USL, 5 FT. EXTENSION	1	✓
7	11-1-6991-1	SNAP HOOK ASSY., PARA. STATIC LINE	1	✓
-	11-1-7051-1	PARACHUTE ASSEMBLY, T-11 MAIN WITH SLIDER (11-1-7059-1)	REF.	
8	11-1-7051-1	PARACHUTE ASSEMBLY, T-11 MAIN SN: T11M-RB 11825	1	✓
9	11-1-7062-1	BRIDLE ASSEMBLY, T-11 MAIN	1	✓
10	11-1-7060-1	DROGUE ASSEMBLY, T-11 MAIN	1	✓
11	11-1-7061-1	DEPLOYMENT SLEEVE ASSEMBLY, T-11 MAIN	1	✓
-	11-1-10005-1	T-11R SINGLE PIN PARACHUTE ASSEMBLY	REF.	
12	11-1-9922-1	T-11R SINGLE PIN CANOPY ASSEMBLY SN: T11R 95088	1	✓
13	11-1-9609-1	PACK TRAY ASSY, T-11R SINGLE PIN	1	✓
14	11-1-9606-1	RIPCORD ASSY, T-11R SINGLE PIN	1	✓
15	11-1-10008-1	EJECTOR SPRING ASSY, T-11R SINGLE PIN	1	✓
16	11-1-9771-1	EXTRACTOR ASSY, T-11R SINGLE PIN	1	✓
-	11-1-9776-1	BRIDLE LEG	10	
-	11-1-9769-1	SOFT-LINK ASSY	1	
17	11-1-7091-1	CLOSING LOOP ASSY, T-11R	1	✓
18	11-1-7729-1	RISER SET, T-11R	1	✓

PACKED BY: [illegible] DATE: 08/31/22

IE - Packing Check List Form created 09/28/2020 SHEET 1 OF 1

Figure 2: An example datasheet before and after being deskewed. The original image is on the left. The deskewed image is on the right.

2.1.4 Grayscaling

Grayscaling, transforming colored images into gray representations, is a foundational technique in image processing. By eliminating color information and encoding pixel intensity through shades of gray (ranging from 0 for black to 255 for white), this reduces the amount of information needed for every image. Grayscaling provides a way to standardize input images for streamlined analysis. This simplification reduces computational complexity by focusing solely on luminance values and mitigating color-related variations. By rendering images in grayscale, it becomes easier to perform additional processing or analysis across diverse image datasets [Lehn et al., 2023].

There are two main methods to grayscale images. The first method takes an average of the red, green, and blue values for the grayscaling value. If we specify r to be red intensity value, b to be the blue intensity value, and g to be the green intensity value, the grayscaling, or intensity, value will be:

$$I = \frac{r+b+g}{3} \tag{1}$$

This method seems flawless at first glance. However, in practice it is not optimal. The human eye does not react the same to seeing red, green, and blue. While the eye is very sensitive to green light, it is much less sensitive to blue light. Because of this, a second, more accurate, way of grayscaling was created. Weighted grayscaling gives each color channel a specific weight which is represented in the following formula:

$$I = 0.299 \cdot r + 0.587 \cdot g + 0.114 \cdot b \tag{2}$$

where r, g, b are the color intensities in the range [0, 255] [Gra, 2024]. To see an example of the method used on a picture, see 3.

2.1.4.1 Grayscaling in OpenCV

In OpenCV, grayscaling is done using the function cvtColor, which takes in an image and a color type and converts the image to that color type. For the purposes of this project, this function was used to convert an image using RGB coloring into a grayscale version of the image. In order to do this, the RGB image and an OpenCV variable called cv2.COLOR_BGR2GRAY were passed into cvtColor, and a grayscale version of the image was returned [OpenCV Developers, 2024a].

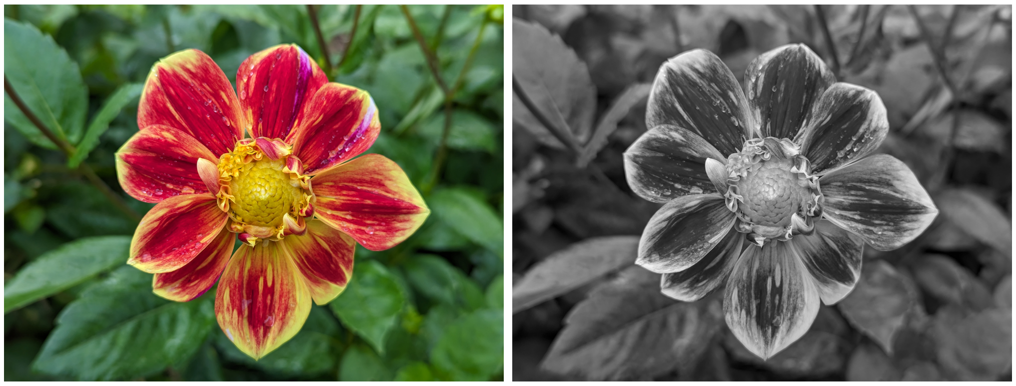


Figure 3: An image of a flower before and after grayscaling is applied. The image on the left is the original image. The image on the right has been grayscaled.

2.1.5 Gaussian Blurring

Image blurring is a procedure that diminishes the sharpness and intricacy of an image. Image blurring smooths the fine details of the image, resulting in reduced clarity. The primary purpose of applying image blurring is to eliminate noise [Domke and Aloimonos, 2009].

All images have some amount of intrinsic noise. For example, a photo of a white sheet of paper will have hundreds of individual pixel values recording how light reflects off the paper and is processed by a camera sensor in slightly different ways. To reduce some of this noise and improve future image processing, we utilize Gaussian blurring to smooth local values together [Hummel et al., 1987].

To see an example of Gaussian blurring at work, see 4.

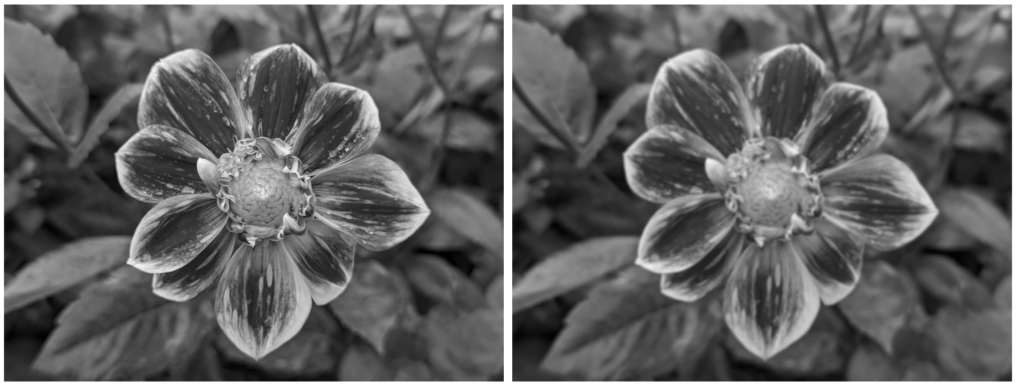


Figure 4: An image of a flower before and after Gaussian blur is applied. The image on the left is the original image. The image on the right has been blurred.

The Gaussian blurring process works by convoluting an input image with a Gaussian kernel. A Gaussian kernel is a matrix of values generated from a Gaussian distribution. The values generated are set by the distance from the center of the kernel and the standard deviation of the distribution, both which are chosen by the user. To generate the pixel values for the blurred image, the kernel is moved around the image, with the new pixel values being the sum-product of all pixels in the kernel centered on the pixel being calculated times their corresponding weight given by the Gaussian kernel. Once this image of new pixel values is calculated, it is returned as the new "smoothed" image [Singhal et al., 2017].

Below is the equation used to determine the values of a Gaussian kernel:

$$G(x,y) = \frac{1}{2\pi\sigma^2} e^{-\frac{x^2+y^2}{2\sigma^2}} \tag{3}$$

Within the equation, σ represents the standard deviation, set by the user, while x and y represent the horizontal and vertical distance from the origin of the kernel respectively [Singhal et al., 2017]. These values allow for the center pixel to be weighted the highest, with the weights decreasing in magnitude the further you travel from the center. This results in pixels retaining most of their value during the blurring process, leading to a still recognizable image.

2.1.5.1 Blurring in OpenCV

In OpenCV, blurring is done by using the function GaussianBlur. This function takes in an input image, a kernel size, and the standard deviation of the kernel in the X and Y directions and returns a blurred image. [OpenCV Developers, 2024b]

2.1.6 Binarization (Thresholding)

Binarization/image thresholding is an important technique in image processing. It is used for converting grayscale images into binary images by highlighting regions of interest based on pixel intensity values. Traditional thresholding methods apply a fixed threshold value to all pixels in an image. One of the limitations of traditional thresholding is its inability to adapt to local variations in illumination across an image. Adaptive thresholding algorithms calculate the threshold for each pixel based on a local neighborhood around it [Ope, 2024].

2.1.6.1 Adaptive Thresholding in OpenCV

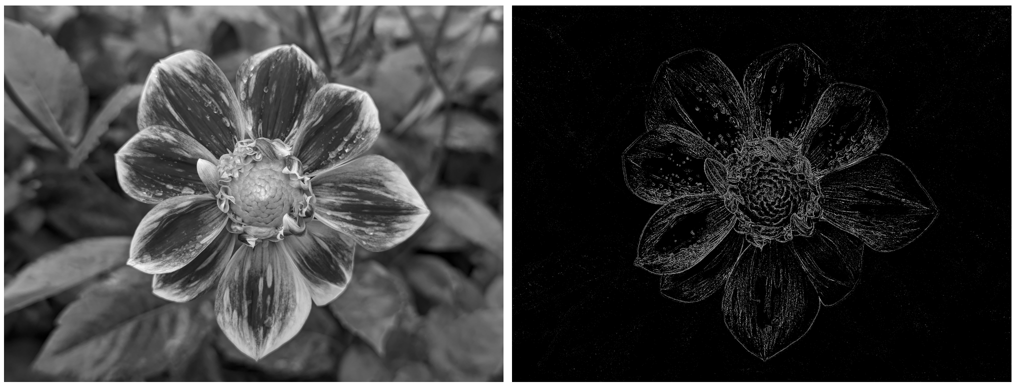


Figure 5: An image of a flower before and after adaptiveThreshold is applied. The image on the left is the original image. The image on the right is the binary result.

In OpenCV, adaptive thresholding is done using the function adaptiveThreshold. This function takes in an image, and alters all pixel values to either be zero or the maximum pixel intensity, which in our case is 255, based on if the pixel value is greater than the weighted sum of the pixels in a small neighborhood containing the pixel in question [Ope, 2024].

The adaptiveThreshold function takes two parameters. The first, adaptiveMethod, is the function that is used to determine the pixel value threshold that differentiates a pixel from being either black or white in the binary image. For this project, we used cv2.ADAPTIVE_THRESH_GAUSSIAN_C. This setting tells adaptiveThreshold to use the weighted sum of the kernel, with the weights being generated from a Gaussian distribution, minus a constant set by the user. In our project, a kernel size of 5x5 was used and the constant that was subtracted off each weighted sum was two, which are represented by the blockSize and C parameters respectively. The second parameter, thresholdType, determines what value the pixel should be set to. We used cv2.THRESH_BINARY_INV which setting the pixel value to 0 if it has a higher value then the calculated threshold for that pixel. Otherwise, the pixel is set to the max value of 255. [Ope, 2024]

Below is the equation used to determine the new value of a given pixel, based on the local threshold value with src(x, y) representing the value of the pixel being evaluated.

$$dst(x,y) = \begin{cases} 0 & \text{if src}(x,y) > T(x,y) \\ \text{maxValue} & \text{otherwise} \end{cases} \tag{4}$$

As seen above, if a given pixel is darker than the local threshold, then the pixel is turned to white, otherwise the pixel is turned black. The local threshold value is determined by the weighted average of the Gaussian filter being applied to a given pixels and its neighbors, determined by using the following equation:[OpenCV Developers, 2024c]

$$\text{T(x,y)} = \frac{\sum_{(x',y')\in\text{block}} \text{src}(x',y') \cdot \text{weight}(x',y')}{\sum_{(x',y')\in\text{block}} \text{weight}(x',y')} - C \tag{5}$$

The constant C is predetermined by the user, while the weights themselves are determined by the same function used for Gaussian Blur, equation 3 with the sigma in this case being determined by the following equation: [OpenCV Developers, 2024b][Ope, 2024]

$$\sigma = 0.3 * ((\text{blockSize} - 1) * 0.5 - 1) + 0.8 \tag{6}$$

To see an example of the method used on an image, see 5.

2.1.7 Morphological Transformations

Morphological transformations are a type of simple operation on binary images that require two inputs: the original image and a structuring element (or kernel) that dictates the operation's nature. Two fundamental morphological operators are Erosion and Dilation, with additional variants such as opening and closing being combinations of the fundamental ones [Mor, 2024].

Erosion, like soil erosion, diminishes the boundaries of the foreground object in binary images. As the kernel slides through the image in a 2D convolution manner, a pixel in the original image becomes 1 only if all of the pixels under the kernel are also 1; otherwise, it is eroded (set to zero). This process discards pixels near the boundary, reducing the thickness of the foreground object. Erosion is effective for eliminating noises and detaching connected objects [Mor, 2024]. To see an example of erosion on an image, see 6.

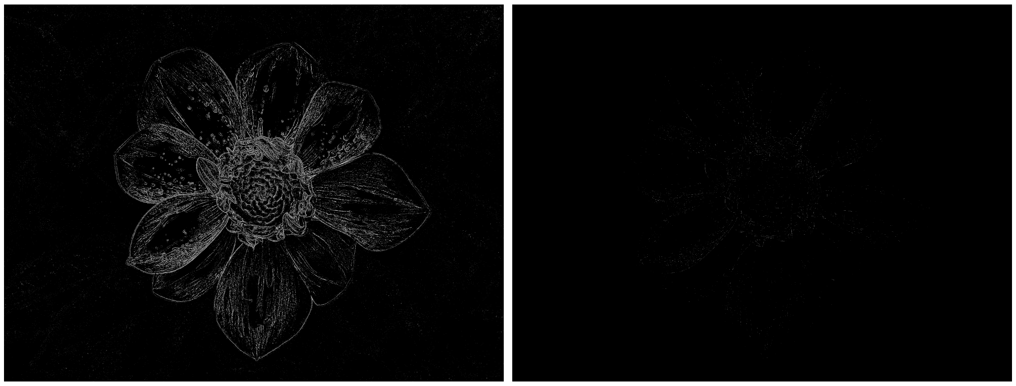


Figure 6: An image of a flower before and after applying erosion with a 3x3 square kernel. The image on the left is the original image. The image on the right has been eroded.

Dilation is the opposite of erosion. A pixel element becomes 1 if at least one pixel under the kernel is 1, leading to an increase in the white region. Typically, dilation follows erosion, especially in noise removal scenarios, as erosion shrinks the object. Dilation helps restore the object's size without reintroducing the eliminated noise and is also useful for connecting broken parts of an object [Mor, 2024]. To see an example of dilation on an image, see 7.

Opening, a combination of erosion and then dilation, is employed to remove noise, as erosion takes care of noise elimination [Mor, 2024]. This combination of the two returns an overall smoother image, as the erosion removes the sharper edges of an image, while the dilation returns the image back to about the same size, however it cannot fully reverse an erosion [Sreedhar and Panlal, 2012]. To see an example of opening on an image, see 8.

Closing, the reverse of opening, involves dilation and then erosion. This operation is valuable for

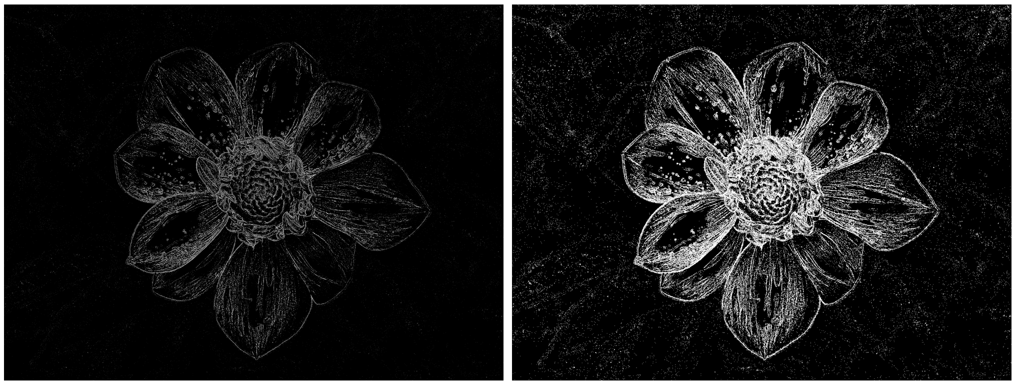


Figure 7: An image of a flower before and after applying dilation with a 3x3 square kernel. The image on the left is the original image. The image on the right has been dilated.

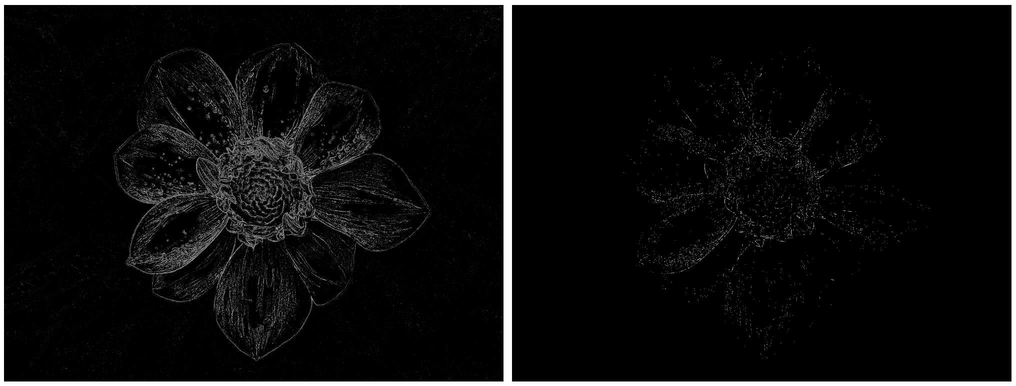


Figure 8: An image of a flower before and after applying an “open” operation with a 3x3 square kernel. The image on the left is the original image. The image on the right has been “opened”.

closing small holes within foreground objects [Mor, 2024]. Especially since the dilation going first means that gaps can be connected in the image, so when erosion occurs, there is a higher likelihood that along the connection point all of the pixels within the kernel are present [Sreedhar and Panlal, 2012]. To see an example of closing on an image, see 9.

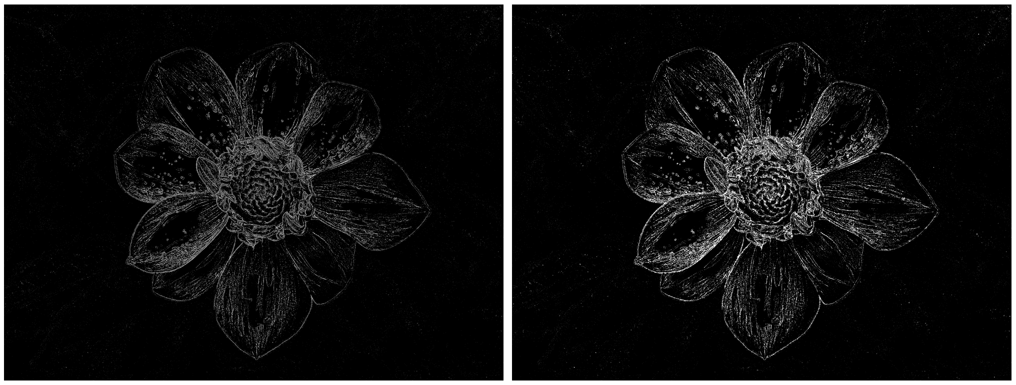


Figure 9: An image of a flower before and after applying a "close" operation with a 3x3 square kernel. The image on the left is the original image. The image on the right has been "closed".

2.1.7.1 Morphological Transformations in OpenCV

Within OpenCV, *morphologyEx* is used to perform two morphological transformations. This function takes as input an image and a "structuring element" (a small binary matrix). Structuring elements can be easily generated by making calls to *getStructuringElement* to generate commonly used kernels.

2.1.8 Perspective Transformation

To introduce the "Perspective Transformation", we present three perspectives of a post-it note 10.

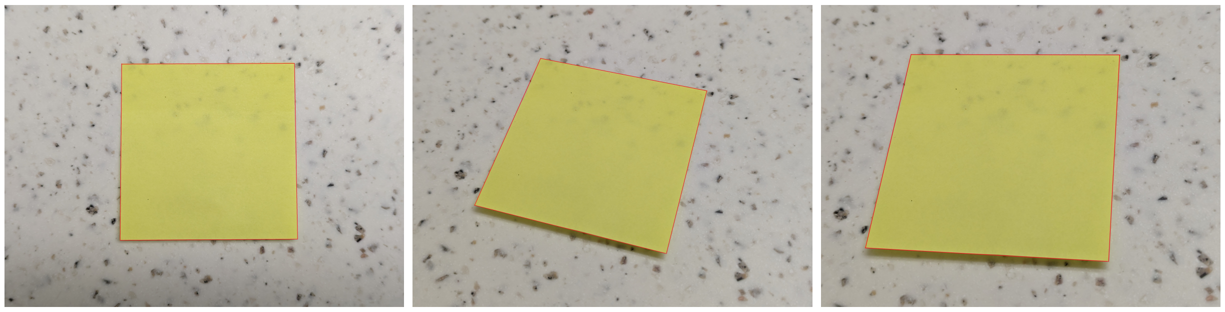


Figure 10: Three photos of a post-it taken to demonstrate perspective warping. In the first image, the photo is taken "straight-on" so the post-it note is accurately depicted as a square. In the second image, the photo is taken from an angle so the post-it note becomes a non-square quadrilateral. The third image takes the second image and aligns it so the top edge is level (deskewing it) emphasizing that it is truly non-square from this perspective.

In the first image, the post-it note appears as a square, and this might be considered to be its intrinsic "shape". In the second image, the viewpoint is no longer orthogonal to the table on which the post-it sits. The third image emphasizes this warping effect by aligning the top edge of the post-it note (from the second image) to be parallel with the top of the photo. In the third image, we can see that the left and right edges of the post-it are no longer at right angles to the top edge. Worse, the edges are not parallel to each-other. Each of the images of the post-it note contain important information. It would be useful to make the second and third images to be identical to the first, as it would be easiest for a computer or a person to read that information. This process can be done through "homography", or the perspective transformation.

The perspective transformation is a projective transformation between two images. Simply, the transform takes a matrix composed of points in an image and multiplies it by the homography matrix to get a new matrix, which represents the points of a new, deskewed, matrix [Pesce et al., 2023]. In mathematical notation, the transform is:

$$\hat{P} = HP \tag{7}$$

Where H represents the following:

$$H = \begin{bmatrix} h_{11} & h_{12} & h_{13} \\ h_{21} & h_{22} & h_{23} \\ h_{31} & h_{32} & 1 \end{bmatrix} \tag{8}$$

Notice that only one value of the matrix is not a variable. This means that many parts of the matrix must change to account for the different image matrix P [Pesce et al., 2023]. So then, how do we calculate these variables? In order to do that, a calibration process is done by picking specific points that relate to each other though these equations.

$$\begin{bmatrix} x_a \\ y_a \\ z_a \end{bmatrix} = H * \begin{bmatrix} x \\ y \\ 1 \end{bmatrix} \tag{9}$$

$$\frac{1}{z_a}\begin{bmatrix} x_a \\ y_a \\ z_a \end{bmatrix} = \begin{bmatrix} \hat{x} \\ \hat{y} \\ 1 \end{bmatrix} \tag{10}$$

Above, x, y are the original picked points, $\hat{x}, \hat{y}$ are the second picked points, and x_a, y_a, z_a are the chosen points after they are put through the transform. When we combine both of these equations, we get:

$$H\begin{bmatrix} x \\ y \\ 1 \end{bmatrix} = z_a\begin{bmatrix} \hat{x} \\ \hat{y} \\ 1 \end{bmatrix} \tag{11}$$

And after distributing them, we get

$$\begin{aligned} z_a\hat{x} &= h_{11}x + h_{12}y + h_{13} \\ z_a\hat{y} &= h_{21}x + h_{22}y + h_{23} \\ z_a &= h_{31}x + h_{32}y + 1 \end{aligned}$$

the relationship between one pair of matching points. As the homograph matrix, H, has exactly eight degrees of freedom, at minimum four corresponding points are needed to solve this matrix [Lee, 2022]. We combine this relationship between the four points to get the following:

$$\begin{bmatrix} x^{(1)} & y^{(1)} & 1 & 0 & 0 & 0 & -\hat{x}^{(1)}x^{(1)} & -\hat{x}^{(1)}y^{(1)} \\ 0 & 0 & 0 & x^{(1)} & y^{(1)} & 1 & -\hat{y}^{(1)}x^{(1)} & -\hat{y}^{(1)}y^{(1)} \\ \dots \\ x^{(i)} & y^{(i)} & 1 & 0 & 0 & 0 & -\hat{x}^{(i)}x^{(i)} & -\hat{x}^{(i)}y^{(i)} \\ 0 & 0 & 0 & x^{(i)} & y^{(i)} & 1 & -\hat{y}^{(i)}x^{(i)} & -\hat{y}^{(i)}y^{(i)} \\ \dots \\ x^{(n)} & y^{(n)} & 1 & 0 & 0 & 0 & -\hat{x}^{(n)}x^{(n)} & -\hat{x}^{(n)}y^{(n)} \\ 0 & 0 & 0 & x^{(n)} & y^{(n)} & 1 & -\hat{y}^{(n)}x^{(n)} & -\hat{y}^{(n)}y^{(n)} \end{bmatrix} \begin{bmatrix} h_{11} \\ h_{12} \\ h_{13} \\ h_{21} \\ h_{22} \\ h_{23} \\ h_{31} \\ h_{32} \end{bmatrix} = \begin{bmatrix} \hat{x}^{(1)} \\ \hat{y}^{(1)} \\ \dots \\ \hat{x}^{(i)} \\ \hat{y}^{(i)} \\ \dots \\ \hat{x}^{(n)} \\ \hat{y}^{(n)} \end{bmatrix} \tag{12}$$

In this new matrix form, we are now able to solve for all values of the homography matrix [Lee, 2022].

2.2 Neural Networks

In order to accurately create a program that can detect text, we first need it to understand several different types of handwriting. As the US military is so large, many different people are involved in parachute testing, meaning many individuals write up needed information about data that will be processed through this program. Thus, we have to show the program many different types of handwriting, and have it learn what specific characters look like. One of the best ways to do this is with neural networks.

A neural network is a type of machine learning algorithm. The specific type of neural network in machine learning is comprised of many nodes that communicate information to each other. Traditional (fully connected/feedforward) neural networks come in layers, with a "weight" connecting each neuron in the previous layer to a neuron in the next layer, and a "bias" upon each neuron itself, where these are all just real numbers. These weights and biases can be stored in the structures of linear algebra (matrices and vectors), as they will be linearly related to each other. Specifically, we can store all the weights between two layers of neurons in a "weight matrix", and all the biases for a layer (the layer of neurons after the weight matrix) in a "bias vector". If an output of a node is above the specific node's bias, that node is activated and a signal is passed on to future nodes. If this bias value is not met, no signal will be passed on.

This notation is as follows: Let $w \in \mathbb{R}^{m \times n}$ be the weight one of the connections in the neural network. We write w^l_{jk} to represent the weight for the connection from the k^{th} neuron in the $(l-1)^{\text{th}}$ layer to the j^{th} neuron in the l^{th} layer [Nielsen, 2019]. In this scenario, j is the output neuron and k is the input neuron. As an example, w^3_{21} represents the weight for the connection of the 4^{th} neuron in the 2^{nd} to the 1^{st} neuron in the 3^{rd} layer of a network. In addition, b^l_j will represent the bias of the j^{th} neuron in the l^{th} layer, while a^l_j will represent the activation of the j^{th} neuron in the l^{th} layer [Nielsen, 2019].

With all these notations, we can say that the activation a^l_j is related to all activations in the $(l-1)^{\text{th}}$ layer with the following equation:

$$a^l_j = \sigma(\sum_k w^l_{jk} a^{l-1}_k + b^l_j) \tag{13}$$

For clarification, the sum is over all neurons k located in the $(l-1)^{\text{th}}$ layer. We can write this expression in matrix form by defining w^l as the weight matrix for the layer l. Each entry in this matrix w^l are all the weights connected to the l^{th} layer of neurons [Nielsen, 2019]. We say that the entry in the k^{th} column and the j^{th} row will be w^l_{jk}. For every layer l, a bias vector can be defined, b^l, where all of the components are the values of b^l_j. Similarly, we can define an activation vector a^l, where the components are the activations components a^l_j for each neuron in the l^{th} layer [Nielsen, 2019]. Lastly, we can vectorize σ, in equation 13, to get the following equation:

$$a^l = \sigma(w^l a^{l-1} + b^l) \tag{14}$$

This formulation allows us to understand how activations from one layer relate to the activations on the most recent previous layer, letting our view become more "global" than before (focusing not on each neuron separately). This new expression is also useful in practical situations, as matrix libraries allow easy and quick ways to implement matrix multiplication, vector addition and subtraction, and vectorization which makes programs run faster.

Before moving on, let us discuss briefly weighted input. We consider z^l to be the weighted input of all the neurons in layer l. The equation for the weighted input is the following:

$$z^l \equiv w^l a^{l-1} + b^l \tag{15}$$

2.2.1 Model Based Transfer Learning

One of the primary bottlenecks our project faced was a lack of data as we were only provided with a few example scanned documents. This poses an issue when training a new model. For example, one type of model we wished to train is one that could perform OCR on handwritten fractions and mixed numbers. Since we were not able to discover an existing dataset on the topic, part of our group's approach was to create a synthetic dataset of handwritten fractions to help fine-tune a pre-trained model.

Model-based transfer learning allows knowledge to be transferred and stored in a model's settings. It works on the idea that both the original and new tasks have some similarities. Instead of storing detailed information about features, this method focuses on storing broader knowledge about how the model works. This makes the model more efficient and better at understanding the original data without needing to do complex operations such as re-sampling or inference [Pan, 2014].

If we have a well-trained source model, we can use its knowledge to help train a new model for a similar task, even if we do not have notable amounts of labeled data for the new task.

For our project, we decided to use utilize transfer knowledge through shared model components. This type of transfer learning creates a target model by using components or hyperparameters in the source model. We took parts of the pre-trained model that were good at recognizing general patterns and used them to help our model recognize fractions [Aggarwal, 2014].

The prior, or the prior probability distribution is what one may assume may happen before anything does. For instance, a coin flip. If one feels it is more likely to land on heads, they will guess heads. It is a prior belief. Using this prior knowledge can help make better estimates before starting. In real-world tasks, applying this prior info can help build useful models even if there is not an immense amount of data to work with [Pan, 2014].

In our model this concept can be applied by taking a pre-existing model and fine-tuning it to whatever dataset we choose. For example, if we have a model that is trained on basic image recognition, we can tune the final layers to interpret mixed numbers better. To do this we would supply the model with a dataset of mixed fractions and use this dataset to do our tuning.

2.2.2 Synthetic Dataset Generation

The need for data access, particularly from publicly funded sources, continues to expand. However, worries regarding the exposure of respondents' identities and sensitive information are causing data collectors to restrict access. Synthetic data sets, designed to mimic crucial aspects of real data while enabling valid statistical analysis, offer a solution to grant broad access to data while addressing privacy and confidentiality concerns [Raghunathan, 2021].

Synthetic data itself refers to data generated using a purpose-built mathematical model or algorithm. This is in contrast to real data, which originates from real-world systems such as satellite images or medical tests [Jordon et al., 2022]. An example of synthetic data is shown below, with a synthetic handwritten calculus dataset.

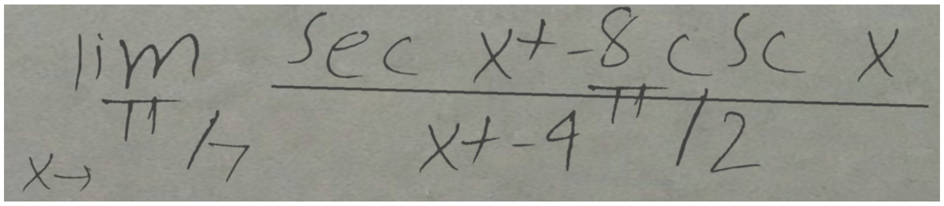


Figure 11: Synthetic Handwritten Calculus Data.

2.2.3 Transformers

Transformers are a neural network architecture proposed by a team of researchers at Google in 2017 [Toews, 2023], in their landmark paper "Attention is All You Need" [Vaswani et al., 2017]. The motivation for this model was to improve on previous language modeling and machine translation approaches, which relied on recurrence [Vaswani et al., 2017]. Such recurrent networks are subject to the "vanishing gradient" problem, in which the signal is lost after passing through a network many times [Zhang, 2023]. The major introduction by this paper is the "Attention" mechanism, which allows for the better handling of long input or output sequences. In the diagram of the full architecture, these attention mechanisms drive the "Multi-Head Attention" and "Masked Multi-Head Attention". The full architecture can be seen below 12.

The proposed solution, and the central element of the paper is the "Attention" mechanism. In its most basic form, it is as follows:

$$\text{Attention}(Q, K, V) = \text{softmax}\left(\frac{QK^T}{\sqrt{d_k}}\right)V \tag{16}$$

where Q, K, and V are vectors called "Queries", "Keys", and "Values". Specifically, they represent the output from whichever previous layer(s) that they come from, processed through a linear layer (as found in a standard feedforward network) corresponding to each of these sets of "queries", "keys", and "values". These linear transformations upon the previous input are what differentiate and ultimately determine the queries, keys and values that will be input to the attention mechanism as described in 16. Queries "query" the keys by means of a dot-product in QK^T, as the dot-product is the mechanism for determining the similarity between vectors. These "similarites" are then scaled down by $\sqrt{d_k}$, where d_k is the dimension of both the queries and the keys (these are equal), then normalized with a softmax function, to produce a probability distribution which, conceptually, represents the keys which were queried for. These are then multiplied by the "values" (the information store), at which point, the important information of the input sequence has been returned from the attention mechanism, without using recurrence or convolution. In this way, the keys and values make a sort of continuous lookup table that is trained within the model, and the queries extract information from this lookup table. This "attention" mechanism represents the model "attending" to different parts of the input sequence, which is the crucial development of this paper, as now information can flow between tokens in a sequence without using a recurrent model. To achieve the "Multi-head Attention" and "Masked Multi-Head Attention" of the full model architecture, several of these attention mechanism are

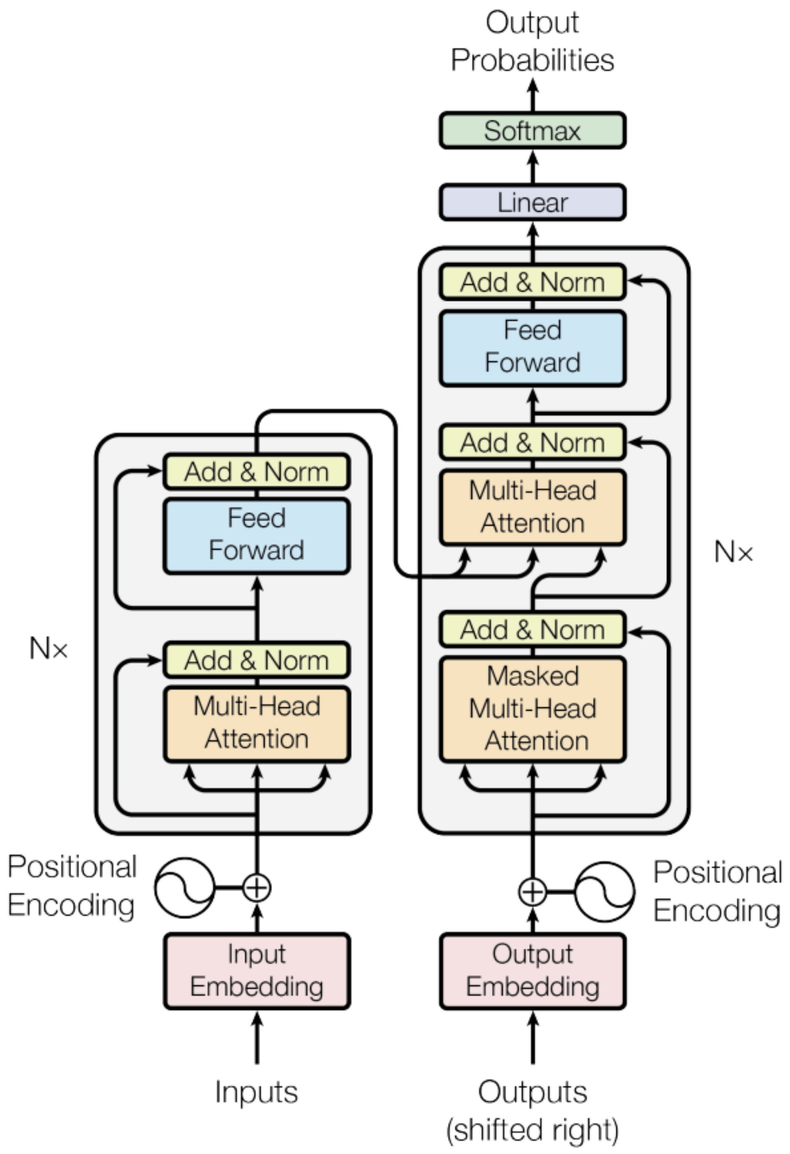


Figure 12: The transformer architecture as presented by "Attention is All You Need" [Vaswani et al., 2017].

concatenated according to the following:

$$\text{MultiHead}(Q, K, V) = \text{Concat}(\text{head}_1, \ldots, \text{head}_h)W^O \tag{17}$$

$$\text{where head}_i = \text{Attention}(QW_i^Q, KW_i^K, VW_i^V) \tag{18}$$

The motivation for this concatenation is to allow the model to concurrently understand different parts of the input. The only difference between the "Multi-Head Attention" and "Masked Multi-Head Attention" that are seen in the full architecture is a "mask", which is simply setting "illegal" connections in the relevant sublayer to be equal to negative infinity. In the right-hand side block (the decoder), this has the effect of preventing tokens that have not been produced yet from communicating with the existing tokens (those that have been generated with previous inferences of the model and passed in as the output embedding), meaning that the model must learn how to predict the next token only using information from the previous tokens, and not subsequent tokens.

The "Feed Forward" layer is a standard "fully-connected" sub-layer here. Also depicted between sub-layers are "residual connections"/"skip connections" which combine input to the sub-layer with output from the sub-layer. This is seen in the diagram as an arrow into the side of the block called "Add & Norm". The activations are then normalized (the "Norm" here), according to:

$$h_i = f\left(\frac{g_i}{\sigma_i}(a_i - \mu_i) + b_i\right) \tag{19}$$

Where μ and σ are the mean and standard deviation of the particular layer's activations, respectively, calculated as follows:

$$\mu^l = \frac{1}{H}\sum_{i=1}^{H} a_i^l \tag{20}$$

$$\sigma^l = \sqrt{\frac{1}{H}\sum_{i=1}^{H}(a_i^l - \mu^l)^2} \tag{21}$$

The effect of equation 19 is that the the activations a_i have been standardized, which in the general case means that their mean is zero and that their standard deviation is one. The only deviation from that more common definition here is that, after the normalization, a learnable gain coefficient is multiplied to the activations, and a learnable bias is added [Ba et al., 2016].

Globally, the left block is called the "Encoder", and the right block is called the "Decoder". The basic usage of this architecture is to feed the source sentence into the encoder, and the current state of the target sentence into the decoder (this being however much of the output sequence has been generated thus far), to sample the next word. It is important to remember that this is not a recurrent model, which is the advantage over previous methods, because this reduces the complexity of the path that the information has to flow down when the model is inferenced from $O(n)$ in the recurrent case, to $O(1)$ in this case [Vaswani et al., 2017].

2.2.4 Pre-Trained Models from HuggingFace

Although models previously may have needed to be created and trained from scratch for specific use cases, many pre-trained models have become freely available for use through platforms such as Hugging Face, an online repository for users to share and download existing models. The final application uses two of these pre-trained Transformer models that were later fine-tuned for the specific tasks.

2.2.4.1 Microsoft TrOCR Architecture

Microsoft's TrOCR is a Transformer model trained to perform Optical Character Recognition tasks. The model consists of an image Transformer as an encoder and a text Transformer decoder, allowing the model to receive an input image and output a string of characters. The Hugging Face Python library provides two classes needed to run the model, one for the model itself and one for the image processing step needed beforehand. The image that needs to be passed is first converted into multiple Tensor objects by the Processor object, which can then be fed into the Model object to return an output of the string of text that was in the original image [Li et al., 2021, Microsoft, 2022, Face, 2024].

Figure 13 shows the flow of information being passed in through the different stages of the model. The bottom two images in the diagram show the original image on the bottom right as well as the processed images on the left. The Processor object converts the original image into smaller 16x16 patches, which are then flattened into a 1D array. Each of the patches is also labelled with positional information, and each of this combination of image and position information is then encoded by the first Transformer model. The second Transformer model then decodes this encoded information and converts this into a series of character chunks [Li et al., 2021].

On Hugging Face, Microsoft has provided multiple instances of the model trained for different text, such as handwritten text, printed text, larger sentence text, etc. The ones used for this application were their

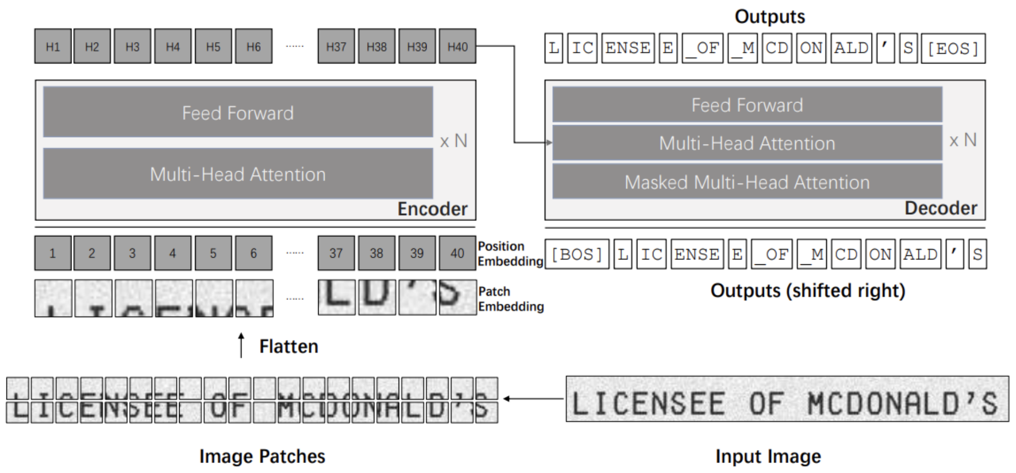


Figure 13: An architectural diagram of the steps the TrOCR model takes to convert an image to text. Diagram from the original paper by Li et al. [Li et al., 2021].

base printed model to convert regular text, as well as their base handwritten model which was fine-tuned for fractional data in order to convert the handwritten fractions found throughout the documents.

2.2.4.2 Google ViT Architecture

Google's ViT model is a Vision Transformer model that has been trained to work well for image classification tasks. It is an encoder model that takes an image and gives a numerical value for the category the image falls into. The model is useful for determining categories for similar looking images, and has numerous user fine-tuned versions on Hugging Face for tasks such as determining the specific species of cat from an image or determining how healthy a specific species of plant is based on the leaves. Just like the TrOCR model, Hugging Face's library provides a Model class and Processor Class, with the Processor class converting the image into an input for the Model class. The numerical value for the category goes from 0 to n-1, with n being the total number of categories as defined beforehand by the user. The user also specifies which category each number corresponds to (ex. 0 being "healthy leaf", 1 being "slightly infected", 2 being "very unhealthy", etc.) [Google, 2021, Dosovitskiy et al., 2021].

Figure 14 shows the flow of information being passed in through the model. The steps are very similar to the first half of the TrOCR model. First the processor turns the original image into 16x16 pixel patches. These patches are flattened into a 1D array and given positional information. This combination is fed into the encoder and provides the numerical category value [Dosovitskiy et al., 2021].

On Hugging Face, Google provides a base version of their model as well as instructions on how to

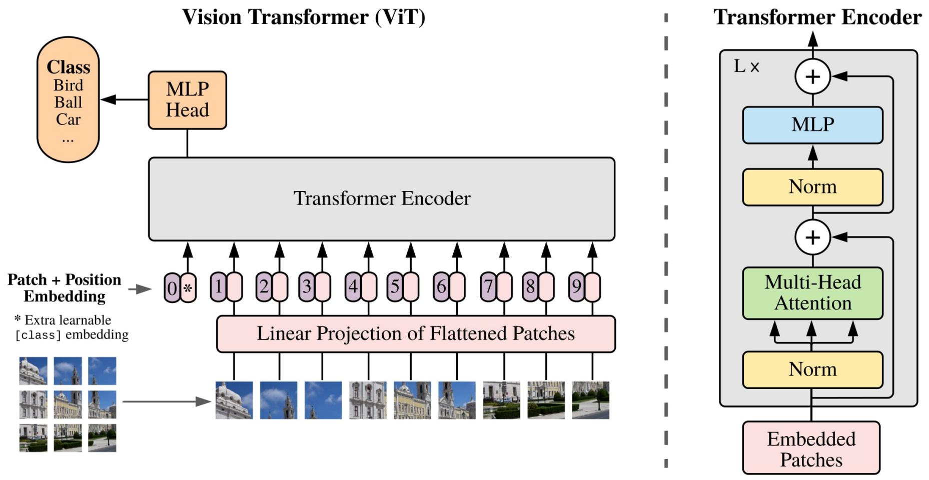


Figure 14: An architectural diagram of the steps the ViT model takes to convert an image to a numerical classification. Diagram from the original paper by Dosovitskiy et al. [Dosovitskiy et al., 2021].

fine-tune it for a specific use case. For the application, this model was used to classify each image into 4 categories: Handwritten, Printed, Fraction, and Blank. This was because the various models worked better for certain text types, so classifying the image beforehand allows our pipeline to use the model that would work best with the image.

2.2.4.3 Seq2Seq Trainer

The Hugging Face model guide provides quickstart links and tutorial notebooks that contain code and parameters that were found to work well with fine-tuning these models. The main class used in these quickstart guides is Seq2SeqTrainer, a class that automates the process of backpropagation and gradient descent. This class is based on Hugging Face's generic Trainer class but has been modified to work well for translation and classification tasks. These Trainer classes do various tasks, such as automatic distribution over multiple GPUs, saving intermediate models after a certain number of steps, saving the model with the lowest loss, etc. These are tasks that can also be done with base PyTorch but this class makes it easier to do so. In combination with the TrainerArguments class to pass in the necessary arguments, the Seq2Seq Trainer class allows us to immediately start finetuning once we have a dataset we want to use for training. [Face, 2023]

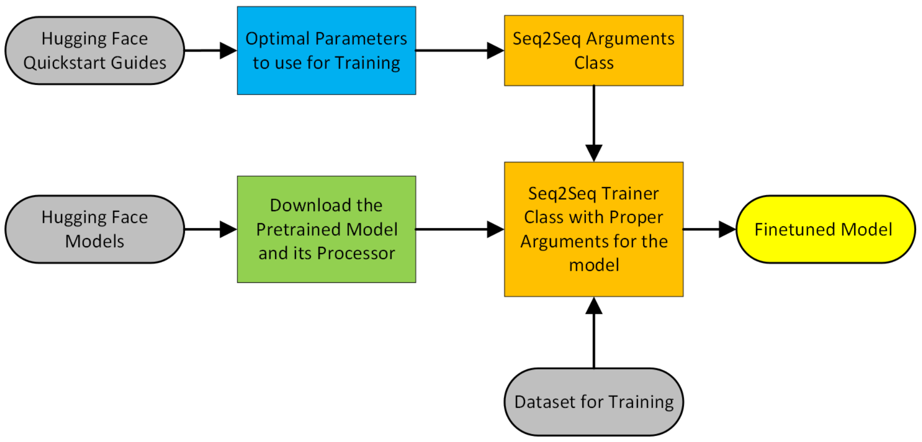

Hugging Face Quickstart Guides
Optimal Parameters to use for Training
Seq2Seq Arguments Class
Hugging Face Models
Download the Pretrained Model and its Processor
Seq2Seq Trainer Class with Proper Arguments for the model
Finetuned Model
Dataset for Training

Figure 15: The flow of information using Seq2Seq. Using the parameters from the quickstart guides for each specific model we can pass these into the arguments class. Once the arguments have been created, we can pass to Seq2SeqTrainer these arguments, the pretrained model, and the dataset we want to finetune the model on. From there, Seq2SeqTrainer will run and output the training results as well as the final finetuned model.

3 Methods

3.1 Overview of Methods

In the following sections we will describe the various steps we undertook to complete this project. This included fine-tuning multiple pre-trained models and building a pipeline to transform a PDF of scanned datasheets into a folder of Excel files. What follows is a brief description of the steps that occur in this pipeline.

Our program a PDF file of scanned datasheets and converts each page into a separate PNG file. In order to improve future steps, we start by pre-processing each page. This involves a deskew to de-rotate the page followed by a perspective transformation to fix any perspective deformation. After the document has been straightened, we locate the edges of each cell in the image to be processed individually. Within each cell, we try to circle each chunk of text or writing and pass it through a classifier to determine if the group is blank, handwritten text, typed text, or handwritten numbers. After the group has been classified, it is then passed to the respective OCR model to be translated. Lastly, the text representation of the group is placed into the correct cell on a generated Excel sheet. Figure 16 depicts this process visually as a flowchart.

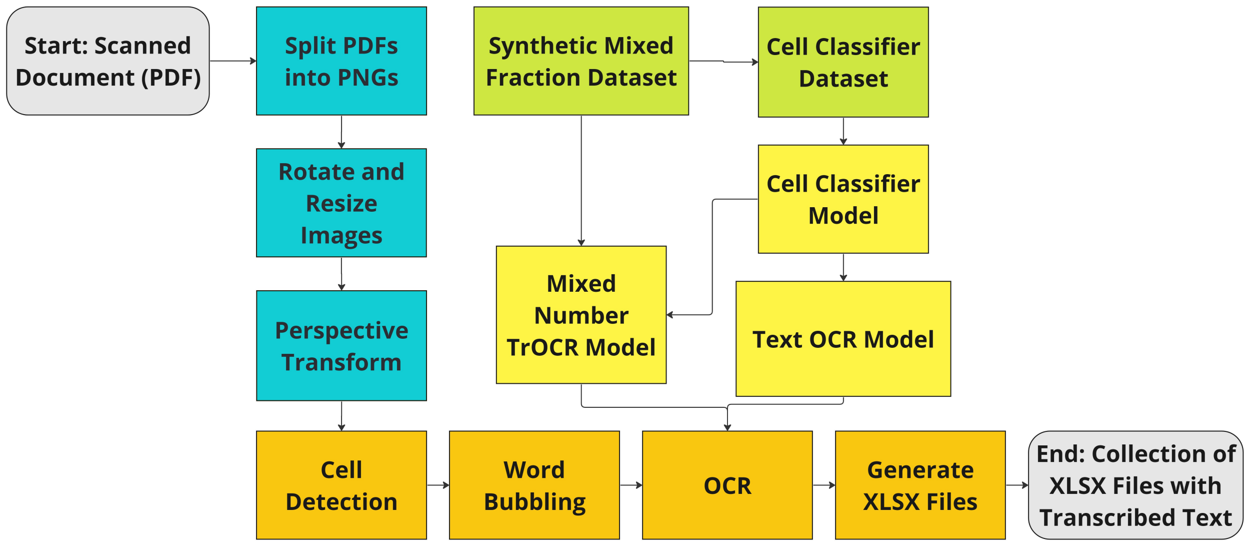


Figure 16: A high-level flowchart of the steps a file takes passing through our code pipeline. Gray is the endpoints of the pipeline, blue is image preprocessing, orange is for excel sheet generation, green is synthetic data generation, and yellow is OCR models.

3.2 Splitting PDF Into Images

Implemented in SplitPDFsIntoImages.py

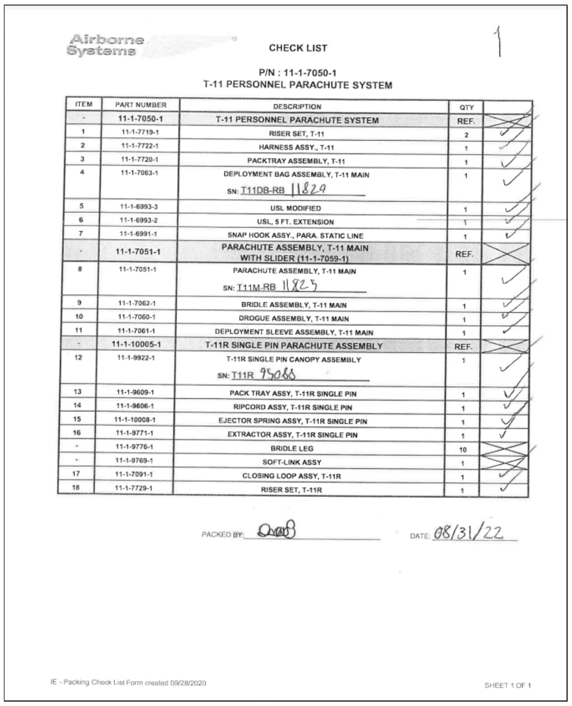
Since images (2D arrays of RGB values) are much easier to manipulate then PDFs, our first step is to transform each page of an input PDF into a PNG. We do this using a python package called pdf2image, which makes the process very straightforward.

3.3 Image Preprocessing

Implemented in PreprocessImages.py

Because many datasheets were scanned in at an angle, our first image pre-processing step is to try to rotate each image until a maximum number of straight lines are horizontal. That is, the image is as level as it can be. We do this by using a Python package called “deskew” which library calculates the angle by which the image should be rotated to achieve this effect. An example image 17 before and after being deskewed is show below.

In addition to deskewing images, we also resize the image in order to have the largest dimension be four thousand pixels. This makes the images more uniform reducing problems that might occur if each image was set at a drastically different pixel scale. This process of resizing is the same as described in 2.1.2.



Airborne Systems

CHECK LIST

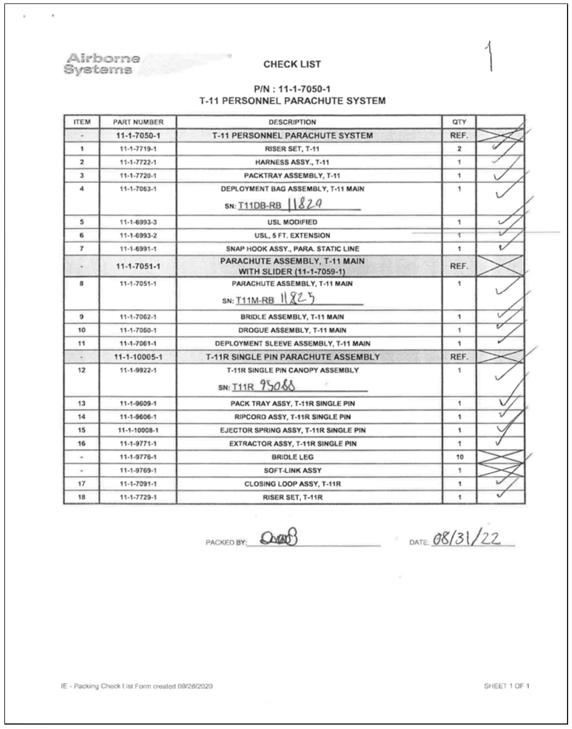
1

P/N : 11-1-7050-1
T-11 PERSONNEL PARACHUTE SYSTEM

ITEM	PART NUMBER	DESCRIPTION	QTY	
-	11-1-7050-1	T-11 PERSONNEL PARACHUTE SYSTEM	REF.	✗
1	11-1-7719-1	RISER SET, T-11	2	✓
2	11-1-7722-1	HARNESS ASSY., T-11	1	✓
3	11-1-7720-1	PACKTRAY ASSEMBLY, T-11	1	✓
4	11-1-7063-1	DEPLOYMENT BAG ASSEMBLY, T-11 MAIN SN: T11DB-RB 11829	1	✓
5	11-1-6993-3	USL MODIFIED	1	✓
6	11-1-6993-2	USL, 5 FT. EXTENSION	1	✓
7	11-1-6991-1	SNAP HOOK ASSY., PARA. STATIC LINE	1	✓
-	11-1-7051-1	PARACHUTE ASSEMBLY, T-11 MAIN WITH SLIDER (11-1-7059-1)	REF.	✗
8	11-1-7051-1	PARACHUTE ASSEMBLY, T-11 MAIN SN: T11M-RB 11825	1	✓
9	11-1-7062-1	BRIDLE ASSEMBLY, T-11 MAIN	1	✓
10	11-1-7060-1	DROGUE ASSEMBLY, T-11 MAIN	1	✓
11	11-1-7061-1	DEPLOYMENT SLEEVE ASSEMBLY, T-11 MAIN	1	✓
-	11-1-10005-1	T-11R SINGLE PIN PARACHUTE ASSEMBLY	REF.	✗
12	11-1-9922-1	T-11R SINGLE PIN CANOPY ASSEMBLY SN: T11R 95088	1	✓
13	11-1-9609-1	PACK TRAY ASSY, T-11R SINGLE PIN	1	✓
14	11-1-9606-1	RIPCORD ASSY, T-11R SINGLE PIN	1	✓
15	11-1-10008-1	EJECTOR SPRING ASSY, T-11R SINGLE PIN	1	✓
16	11-1-9771-1	EXTRACTOR ASSY, T-11R SINGLE PIN	1	✓
-	11-1-9776-1	BRIDLE LEG	10	✗
-	11-1-9769-1	SOFT-LINK ASSY	1	✗
17	11-1-7091-1	CLOSING LOOP ASSY, T-11R	1	✓
18	11-1-7729-1	RISER SET, T-11R	1	✓

PACKED BY: [illegible] DATE: 08/31/22

IE - Packing Check List Form created 09/28/2020 SHEET 1 OF 1



Airborne Systems

CHECK LIST

1

P/N : 11-1-7050-1
T-11 PERSONNEL PARACHUTE SYSTEM

ITEM	PART NUMBER	DESCRIPTION	QTY	
-	11-1-7050-1	T-11 PERSONNEL PARACHUTE SYSTEM	REF.	✗
1	11-1-7719-1	RISER SET, T-11	2	✓
2	11-1-7722-1	HARNESS ASSY., T-11	1	✓
3	11-1-7720-1	PACKTRAY ASSEMBLY, T-11	1	✓
4	11-1-7063-1	DEPLOYMENT BAG ASSEMBLY, T-11 MAIN SN: T11DB-RB 11829	1	✓
5	11-1-6993-3	USL MODIFIED	1	✓
6	11-1-6993-2	USL, 5 FT. EXTENSION	1	✓
7	11-1-6991-1	SNAP HOOK ASSY., PARA. STATIC LINE	1	✓
-	11-1-7051-1	PARACHUTE ASSEMBLY, T-11 MAIN WITH SLIDER (11-1-7059-1)	REF.	✗
8	11-1-7051-1	PARACHUTE ASSEMBLY, T-11 MAIN SN: T11M-RB 11825	1	✓
9	11-1-7062-1	BRIDLE ASSEMBLY, T-11 MAIN	1	✓
10	11-1-7060-1	DROGUE ASSEMBLY, T-11 MAIN	1	✓
11	11-1-7061-1	DEPLOYMENT SLEEVE ASSEMBLY, T-11 MAIN	1	✓
-	11-1-10005-1	T-11R SINGLE PIN PARACHUTE ASSEMBLY	REF.	✗
12	11-1-9922-1	T-11R SINGLE PIN CANOPY ASSEMBLY SN: T11R 95088	1	✓
13	11-1-9609-1	PACK TRAY ASSY, T-11R SINGLE PIN	1	✓
14	11-1-9606-1	RIPCORD ASSY, T-11R SINGLE PIN	1	✓
15	11-1-10008-1	EJECTOR SPRING ASSY, T-11R SINGLE PIN	1	✓
16	11-1-9771-1	EXTRACTOR ASSY, T-11R SINGLE PIN	1	✓
-	11-1-9776-1	BRIDLE LEG	10	✗
-	11-1-9769-1	SOFT-LINK ASSY	1	✗
17	11-1-7091-1	CLOSING LOOP ASSY, T-11R	1	✓
18	11-1-7729-1	RISER SET, T-11R	1	✓

PACKED BY: [illegible] DATE: 08/31/22

IE - Packing Check List Form created 09/28/2020 SHEET 1 OF 1

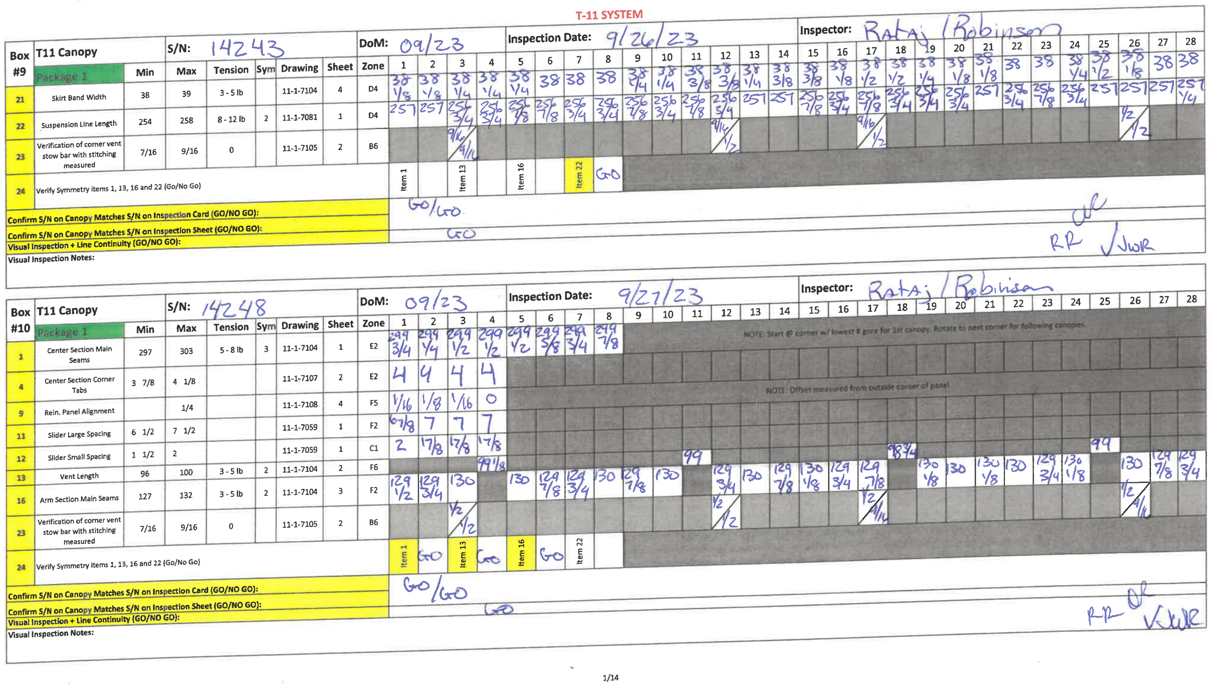
Figure 17: An example of deskewing done, the image on the left is skewed while the image on the right has been straightened.

3.4 Perspective Transformation

Implemented in DewarpPerspective.py

Not only were the documents rotated, we also found that they had been warped. This is likely from them being scanned using a phone instead of a flatbed scanner. Instead of the datasheets being perfectly rectangular, they were non-parallel quadrilaterals. This was discussed in 2.1.8.

The datasheets needed to be unwarped in order for our text box detector to be able to accurately and consistently detect every text box within a given datasheet. The process of dewarping an image has several steps. We first convert the page to a binary image that contains only the cell edges. Next we try to find the largest quadrilateral on the page (which we assume to be the border of a table). Lastly we do transformation that maps the corners of this quadrilateral into a rectangle thus dewarping the image. See 18 for an example datasheet.



T-11 SYSTEM

Box #9 T11 Canopy	S/N: 14243	DoM: 09/23	Inspection Date: 9/26/23	Inspector: Rataj / Robinson

Box #9	Package 1	Min	Max	Tension	Sym	Drawing	Sheet	Zone
21	Skirt Band Width	38	39	3 - 5 lb		11-1-7104	4	D4
22	Suspension Line Length	254	258	8 - 12 lb	2	11-1-7081	1	D4
23	Verification of corner vent stow bar with stitching measured	7/16	9/16	0		11-1-7105	2	B6
24	Verify Symmetry items 1, 13, 16 and 22 (Go/No Go)							

Box #9	1	2	3	4	5	6	7	8	9	10	11	12	13	14
21	38 1/8	38 1/8	38 1/4	38 1/4	38 1/4	38	38	38	38 1/4	38 1/4	38 3/8	38 3/8	38 1/4	38 3/8
22	257	257	256 3/4	256 3/4	256 7/8	256 7/8	256 3/4	256 3/4	256 7/8	256 3/4	256 7/8	256 3/4	257	257
23			9/16 / 9/16									9/16 / 1/2		
24	Item 1		Item 13		Item 16		Item 22	GO						

Box #9	15	16	17	18	19	20	21	22	23	24	25	26	27	28
21	38 3/8	38 1/8	38 1/2	38 1/2	38 1/4	38 1/8	38 1/8	38	38	38 1/4	38 1/2	38 1/8	38	38
22	256 7/8	256 3/4	256 7/8	256 3/4	256 3/4	256 3/4	257	256 3/4	256 7/8	256 3/4	257	257	257	257 1/4
23			9/16 / 1/2									1/2 / 1/2		
24														

Confirm S/N on Canopy Matches S/N on Inspection Card (GO/NO GO): GO/GO

Confirm S/N on Canopy Matches S/N on Inspection Sheet (GO/NO GO): GO

Visual Inspection + Line Continuity (GO/NO GO):

Visual Inspection Notes:

RR JWR

Box #10 T11 Canopy	S/N: 14248	DoM: 09/23	Inspection Date: 9/27/23	Inspector: Rataj / Robinson

Box #10	Package 1	Min	Max	Tension	Sym	Drawing	Sheet	Zone
1	Center Section Main Seams	297	303	5 - 8 lb	3	11-1-7104	1	E2
4	Center Section Corner Tabs	3 7/8	4 1/8			11-1-7107	2	E2
9	Rein. Panel Alignment		1/4			11-1-7108	4	F5
11	Slider Large Spacing	6 1/2	7 1/2			11-1-7059	1	F2
12	Slider Small Spacing	1 1/2	2			11-1-7059	1	C1
13	Vent Length	96	100	3 - 5 lb	2	11-1-7104	2	F6
16	Arm Section Main Seams	127	132	3 - 5 lb	2	11-1-7104	3	F2
23	Verification of corner vent stow bar with stitching measured	7/16	9/16	0		11-1-7105	2	B6
24	Verify Symmetry items 1, 13, 16 and 22 (Go/No Go)							

NOTE: Start @ corner w/ lowest # gore for 1st canopy. Rotate to next corner for following canopies.

NOTE: Offset measured from outside corner of panel

Box #10	1	2	3	4	5	6	7	8	9	10	11	12	13	14
1	299 3/4	299 1/4	299 1/2	299 1/2	299 1/2	299 5/8	299 3/4	299 7/8						
4	4	4	4	4										
9	1/16	1/8	1/16	0										
11	6 7/8	7	7	7										
12	2	1 7/8	1 7/8	1 7/8										
13				99 1/8							99			
16	129 1/2	129 3/4	130		130	129 7/8	129 3/4	130	129 7/8	130		129 3/4	130	129 7/8
23			1/2 / 1/2									1/2 / 1/2		
24	Item 1	GO	Item 13	GO	Item 16	GO	Item 22							

Box #10	15	16	17	18	19	20	21	22	23	24	25	26	27	28
1														
4														
9														
11														
12														
13				98 3/4							99			
16	130 1/8	129 3/4	129 7/8		130 1/8	130	130 1/8	130	129 3/4	130 1/8		130	129 7/8	129 3/4
23			1/2 / 9/16									1/2 / 9/16		
24														

Confirm S/N on Canopy Matches S/N on Inspection Card (GO/NO GO): GO/GO

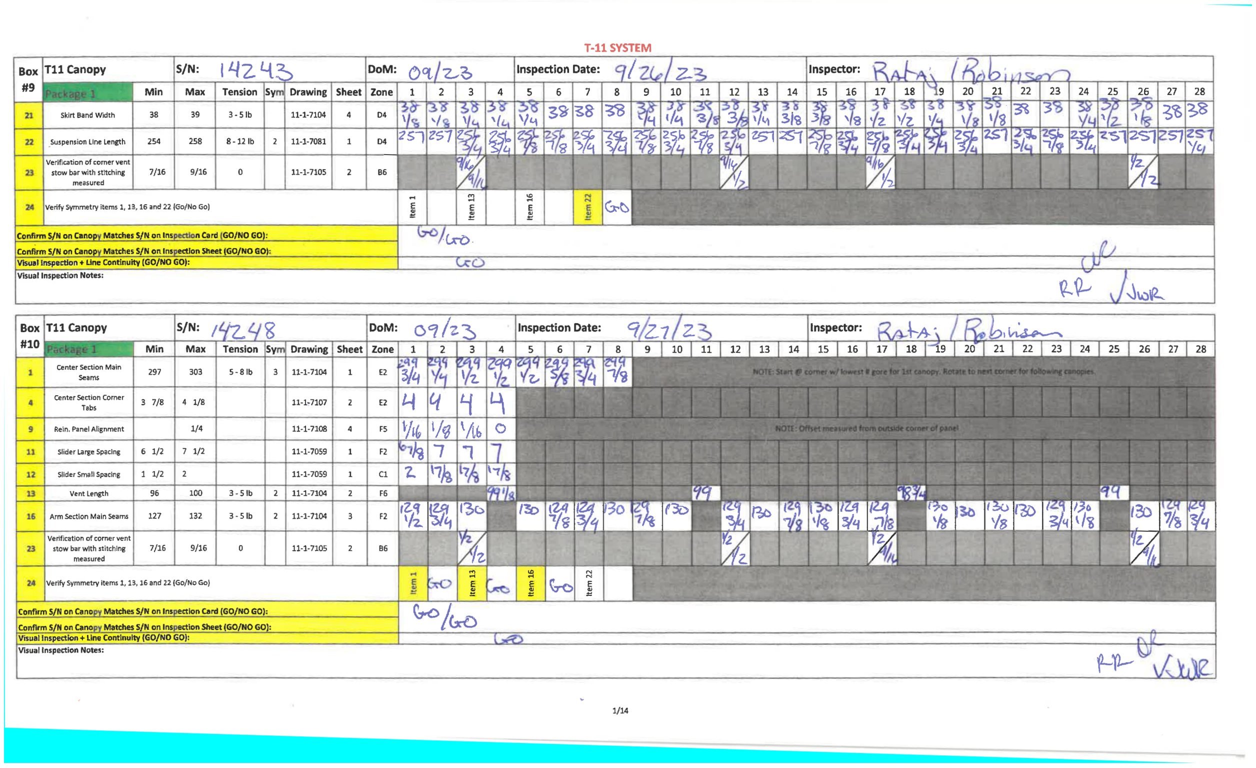
Confirm S/N on Canopy Matches S/N on Inspection Sheet (GO/NO GO): GO

Visual Inspection + Line Continuity (GO/NO GO):

Visual Inspection Notes:

RR JWR

1/14



T-11 SYSTEM

Box #9 T11 Canopy S/N: 14243 DoM: 09/23 Inspection Date: 9/26/23 Inspector: Rataj / Robinson

Box #10 T11 Canopy S/N: 14248 DoM: 09/23 Inspection Date: 9/27/23 Inspector: Rataj / Robinson

1/14

Figure 18: An image of a datasheet before and after applying a perspective transform. The image on the top is the original image. The image on the bottom right has been transformed. The light blue on the bottom of the transformed image is an arbitrary color to fill in newly generated pixels.

3.4.1 Binary Edge Image Creation

To generate the binary image with only vertical and horizontal cell edges highlighted, we start by converting the image to binary. This is done in a three step process: first the image is converted from color to grayscale as described in 2.1.4.1, secondly that image has Gaussian blur applied to it as described in 2.1.5.1, and finally the smooth gray image is converted into a binary image as described in 2.1.6.1. Lastly, we utilize several morphological transformations described in 2.1.7.1 with carefully selected kernels to strategically extract the horizontal and vertical lines. See 19 for an example.

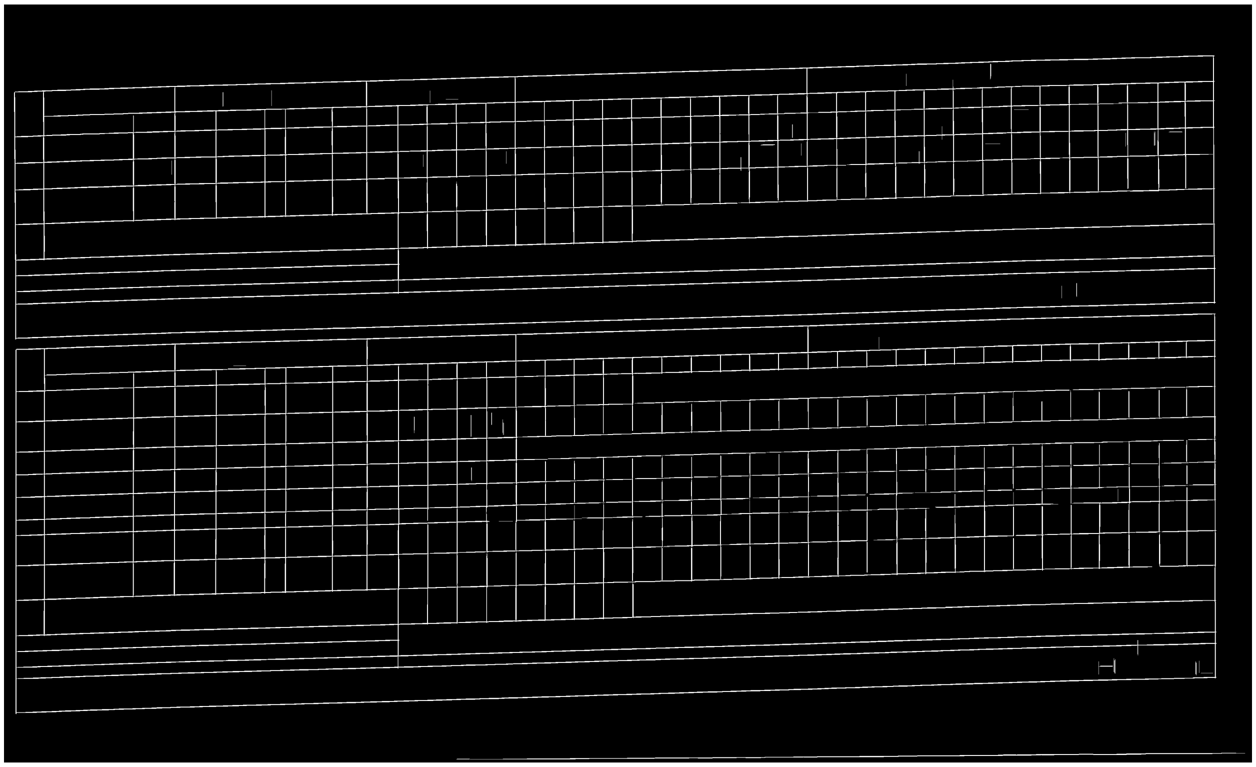
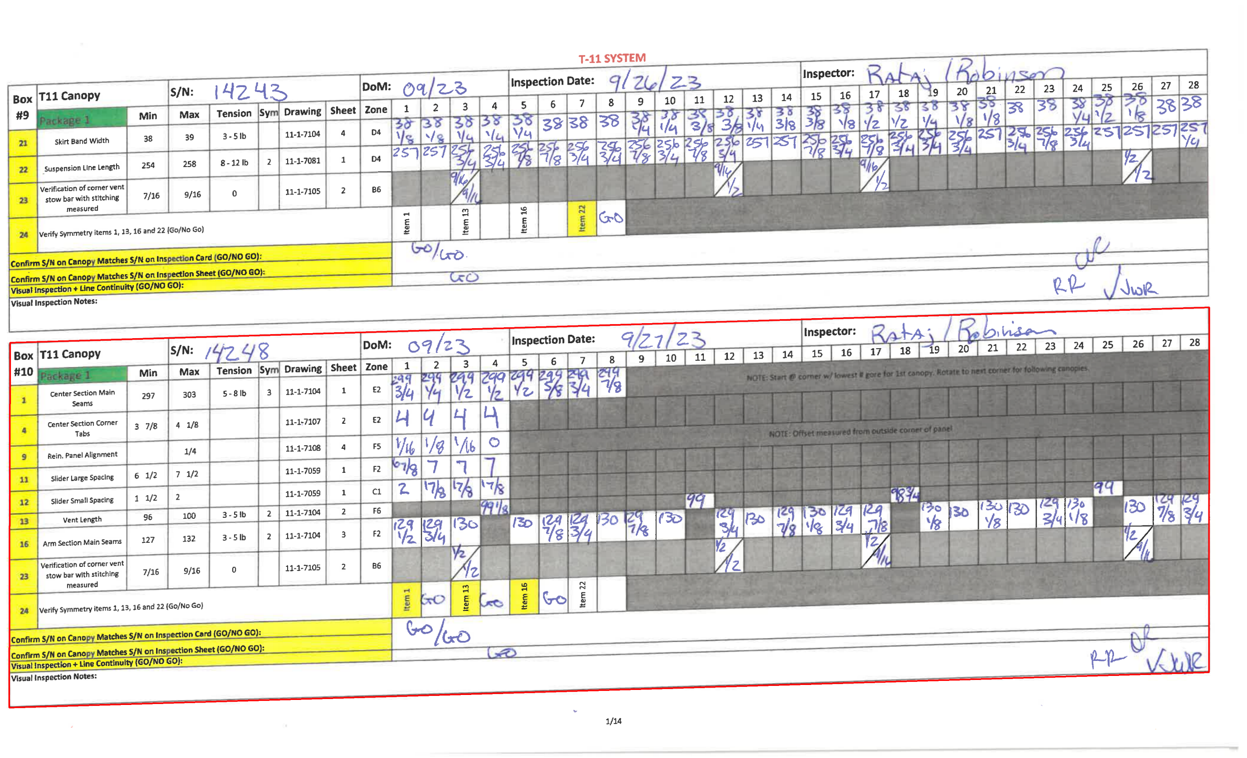


Figure 19: An image of a datasheet that has been converted to binary and and had the cell edges extracted.

3.4.2 Largest Quadrilateral Detection

In order to reorient a datasheet, we start by finding the largest quadrilateral which hope is the border of a table of cells. Once we have the largest quadrilateral, we can then perform a perspective transformation to convert the quadrilateral into a rectangle. To find the largest quadrilateral, we take the previously generated binary image of edges and iterate through all of the possible closed contours in the image from largest to smallest by area. We try to approximate each contour using a subset of its initial points. If we are able to approximate the contour well using a subset set of four points, we have located a quadrilateral. Next, we check to make sure the top and bottom sides of the quadrilateral are roughly parallel. In all of the warped images we have seen, the top and bottom are roughly parallel. If the top and bottom sides are not roughly parallel, we have likely encountered an error and should keep looking for the next largest quadrilateral. Lastly, we have included a minimum area parameter in order to ensure confidence that the image can be properly dewarped. If there is no quadrilateral found that is large enough, then the image is not dewarped and the original image is returned. See 20 showing the largest quadrilateral overlaid on the original image.


T-11 SYSTEM

Box #9 T11 Canopy S/N: 14243 DoM: 09/23 Inspection Date: 9/26/23 Inspector: Rataj / Robinson

Box #9	Package 1	Min	Max	Tension	Sym	Drawing	Sheet	Zone	1	2	3	4	5	6	7	8	9	10	11	12	13	14
21	Skirt Band Width	38	39	3 - 5 lb		11-1-7104	4	D4	38 1/8	38 1/8	38 1/4	38 1/4	38 1/4	38	38	38	38 1/4	38 1/4	38 3/8	38 3/8	38 1/4	38 3/8
22	Suspension Line Length	254	258	8 - 12 lb	2	11-1-7081	1	D4	257	257	256 3/4	256 3/4	256 7/8	256 7/8	256 3/4	256 3/4	256 7/8	256 3/4	256 7/8	256 3/4	257	257
23	Verification of corner vent stow bar with stitching measured	7/16	9/16	0		11-1-7105	2	B6			9/16 / 9/16									9/16 / 1/2		
24	Verify Symmetry items 1, 13, 16 and 22 (Go/No Go)								Item 1		Item 13		Item 16		Item 22	GO						

Box #9	15	16	17	18	19	20	21	22	23	24	25	26	27	28
21	38 3/8	38 1/8	38 1/2	38 1/2	38 1/4	38 1/8	38 1/8	38	38	38 1/4	38 1/2	38 1/8	38	38
22	256 7/8	256 3/4	256 7/8	256 3/4	256 3/4	256 3/4	257	256 3/4	256 7/8	256 3/4	257	257	257	257 1/4
23			9/16 / 1/2									1/2 / 1/2		
24														

Confirm S/N on Canopy Matches S/N on Inspection Card (GO/NO GO): GO/GO.
Confirm S/N on Canopy Matches S/N on Inspection Sheet (GO/NO GO): GO
Visual Inspection + Line Continuity (GO/NO GO): RR ✓JWR
Visual Inspection Notes:

Box #10 T11 Canopy S/N: 14248 DoM: 09/23 Inspection Date: 9/27/23 Inspector: Rataj / Robinson

Box #10	Package 1	Min	Max	Tension	Sym	Drawing	Sheet	Zone	1	2	3	4	5	6	7	8	9	10	11	12	13	14
1	Center Section Main Seams	297	303	5 - 8 lb	3	11-1-7104	1	E2	299 3/4	299 1/4	299 1/2	299 1/2	299 1/2	299 5/8	299 3/4	299 7/8						
4	Center Section Corner Tabs	3 7/8	4 1/8			11-1-7107	2	E2	4	4	4	4										
9	Rein. Panel Alignment		1/4			11-1-7108	4	F5	1/16	1/8	1/16	0										
11	Slider Large Spacing	6 1/2	7 1/2			11-1-7059	1	F2	6 7/8	7	7	7										
12	Slider Small Spacing	1 1/2	2			11-1-7059	1	C1	2	1 7/8	1 7/8	1 7/8										
13	Vent Length	96	100	3 - 5 lb	2	11-1-7104	2	F6				99 1/8							99			
16	Arm Section Main Seams	127	132	3 - 5 lb	2	11-1-7104	3	F2	129 1/2	129 3/4	130		130	129 7/8	129 3/4	130	129 7/8	130		129 3/4	130	129 7/8
23	Verification of corner vent stow bar with stitching measured	7/16	9/16	0		11-1-7105	2	B6			1/2 / 1/2									1/2 / 1/2		
24	Verify Symmetry items 1, 13, 16 and 22 (Go/No Go)								Item 1	GO	Item 13	GO	Item 16	GO	Item 22							

Box #10	15	16	17	18	19	20	21	22	23	24	25	26	27	28
1	NOTE: Start @ corner w/ lowest # gore for 1st canopy. Rotate to next corner for following canopies.													
4														
9	NOTE: Offset measured from outside corner of panel													
11														
12														
13				98 3/4							99			
16	130 1/8	129 3/4	129 7/8		130 1/8	130	130 1/8	130	129 3/4	130 1/8		130	129 7/8	129 3/4
23			1/2 / 9/16									1/2 / 9/16		
24														

Confirm S/N on Canopy Matches S/N on Inspection Card (GO/NO GO): GO/GO
Confirm S/N on Canopy Matches S/N on Inspection Sheet (GO/NO GO): GO
Visual Inspection + Line Continuity (GO/NO GO): RR ✓JWR
Visual Inspection Notes:

1/14

Figure 20: An image of a datasheet with an overlay of the largest detected quadrilateral on image (in red).

3.4.3 Four Point Transformation

To perform the perspective transformation, we take in the points denoting the corners of the largest quadrilateral we found in the last step and use them to calculate a set of target points. We can then use OpenCV's *getPerspectiveTransform* and *warpPerspective* functions in order to get the transformation matrix required to transform the original quadrilateral into the target quadrilateral. Assuming the whole image is warped in the same manner as the largest quadrilateral (which it has been in all our observed data), the rest of the image will also be dewarped correctly.

3.5 Making a Mixed Fraction OCR Model

The biggest problem the Natick Army Lab found with already existing software was the lack of support for fractional datasets. Already existing and pretrained OCR models that we could easily evaluate did not function on any fractions we gave them. Finetuning these models using readily available math datasets also did not work well on the document fractions as they often overfitted to a specific format of text. To properly create a fraction model, we needed to create a synthetic dataset of fractions that would look similar to examples found in the parachute documents. This synthetic dataset allowed us to finetune an existing OCR model in order to work well on the documents.

3.5.1 Initial Fine-tuning using Existing Datasets

The first initial attempts to train a model involved using existing datasets that were publicly available for use. This dataset needed to be a) Handwritten, b) Labelled, and c) Involved Fractions, which limited the amount of viable ones that could be used. The most promising one that fit these criteria was a synthetic handwritten calculus dataset specifically made for OCR tasks. This dataset contained 100,000 synthetically generated images spread evenly across 10 batches, with each of these batches containing a JSON file with the correct labeling for each image in LaTeX form.[Pearson, 2023]

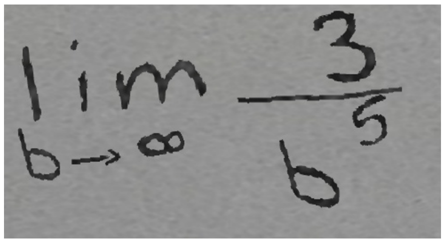


Figure 21: An example of one of the images from the synthetic calculus dataset. [Pearson, 2023].

For initial testing one batch of 10,000 images was used to fine-tune the TrOCR model (details regarding how fine-tuning works can be found later on in section 3.6.2). However, validating this fine-tuned model with the parachute data resulted in incorrect answers as the model was trained in a calculus dataset with limits, resulting in every output from the model also being a limit.

Even though these calculus datasets did contain fractions, they would overfit in training leading to the model always outputting a limit. From this attempt, we concluded we would not be able to finetune a model on any dataset that contained fractions, but instead on a dataset containing only fractions, which limited the amount of datasets even further. No public dataset was found that fit this criteria, meaning that

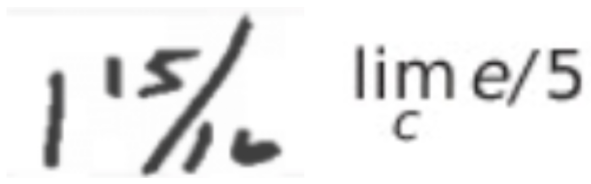


Figure 22: The image on the left is one of the handwritten fractions found in the parachute data. The image on the right is the output produced by the model that was fine-tuned on the calculus dataset when acting on the left image.

fine-tuning the model would first require generating a dataset of our own.

3.5.2 Synthetic Mixed Number Dataset Generation

Implemented in GenerateSyntheticMixedNumberData.py

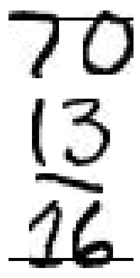


Figure 23: An example image of a generated synthetic fraction.

In order to fine-tune our model, we created a synthetic dataset of handwritten mixed numbers and fractions because we were unable to locate such a dataset on the web. This dataset was created by concatenating together different digits from the MNIST dataset in order to create any whole number, proper fraction, or mixed fraction. The images we created are modeled after the numbers we found in the cells on the datasheets. Because there are multiple layouts we found in cells, we have multiple layouts in our synthetic numbers. They are:

1. A whole number above a horizontally arranged fraction
2. A whole number above a vertically arranged fraction
3. A whole number next to a horizontally arranged fraction
4. A whole number next to a vertically arranged fraction
5. Only a whole number
6. Only a horizontally arranged fraction
7. Only a vertically arranged fraction

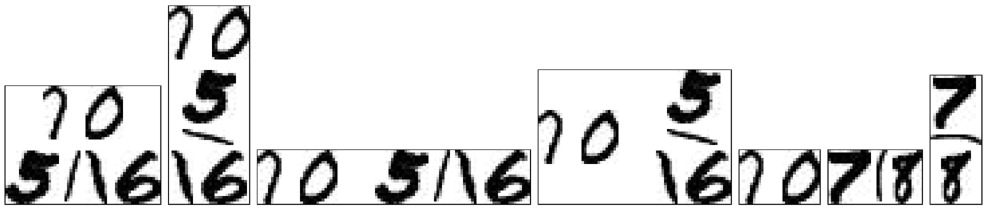


Figure 24: Example images of synthetic fractions. From left to right: a whole number above a horizontally arranged fraction, a whole number above a vertically arranged fraction, a whole number next to a horizontally arranged fraction, a whole number next to a vertically arranged fraction, only a whole number, only a horizontally arranged fraction, only a vertically arranged fraction.

To create the fraction slashes, we would randomly select a “1” from the MNIST dataset and would use it as is when creating a fraction in the horizontal orientation. When the fraction is of vertical orientation, we would first rotate the “1” ninety degrees to make a horizontal fraction slash. Once an image is created, we add anywhere from zero to four straight lines near the borders of the image to mimic if there is some overlap between a written number and the cell it goes into. We initially randomly selected a “1” from any of the ones provided by MNIST, however since someones had flags we instead made a small group of ones to select from that were relatively straight lines rotated by different amounts.

After looking through the handwritten mixed numbers, fractions, and whole numbers provided from the datasheets received from the Natick Labs, we were able to create a list of assumptions about what future handwritten numbers would look like. These assumptions were: the leading digit of the whole number will not be zero, all denominators are powers of two with the largest denominator value being sixteen, and all fractions are proper so the numerator must be odd and less than the denominator. These assumptions were then reflected in the dataset we generated. This process of tailoring the training dataset to match expected testing data is known as data snooping.

After images could successfully be generated, the Python script was then altered to create a user-specified amount of random images into one folder as well as a master JSON file that contained info regarding each image. This information was the file’s name, the correct labeling of the image, and the type of fraction/number, although the latter was only used for debugging and not needed for training. The file’s name and label were what was needed to create a dataset that could be used to fine-tune a model.

3.5.3 Fine-tuning Mixed Fraction Model

Implemented in TrainMixedNumberTrOCR.py

After the dataset was created, we could now begin fine-tuning our model to work well with reading fractional data. The process of fine-tuning involved three main steps to successfully create a model to be used for our application. The first was converting our dataset into a format that could be used for the fine-tuning process. The second was the process of fine-tuning itself using built-in Hugging Face functions as well as parameters recommended for the TrOCR model. At the very end we would evaluate the model against examples from the parachute dataset to verify the model would work properly.

3.5.3.1 Creating a Torch Dataset

The first step is turning our dataset of images in labels into a format that could be used by Python and other trainer libraries for fine-tuning. The format, in this case, is a Torch Dataset, which is an abstract class with generic functions such as "__getitem__" that can be called by existing trainer functions. The information from the JSON file was brought into our Dataset class by first converting the file into a Pandas Dataframe. This Dataframe is what our Dataset class iterated through to create Tensors for the image and the label [PyTorch, 2023].

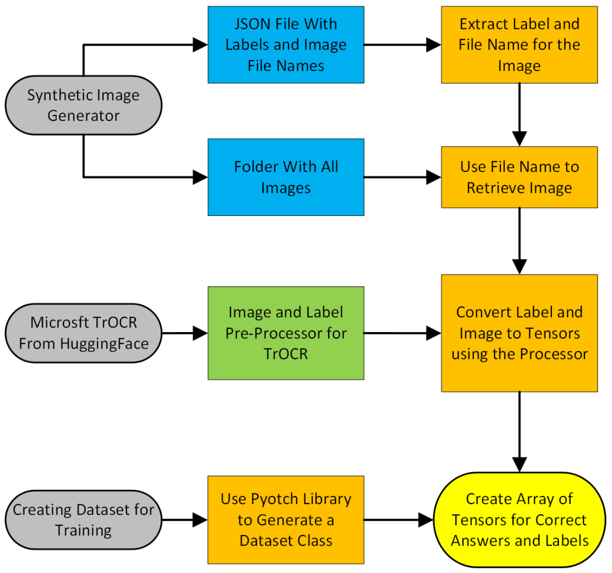
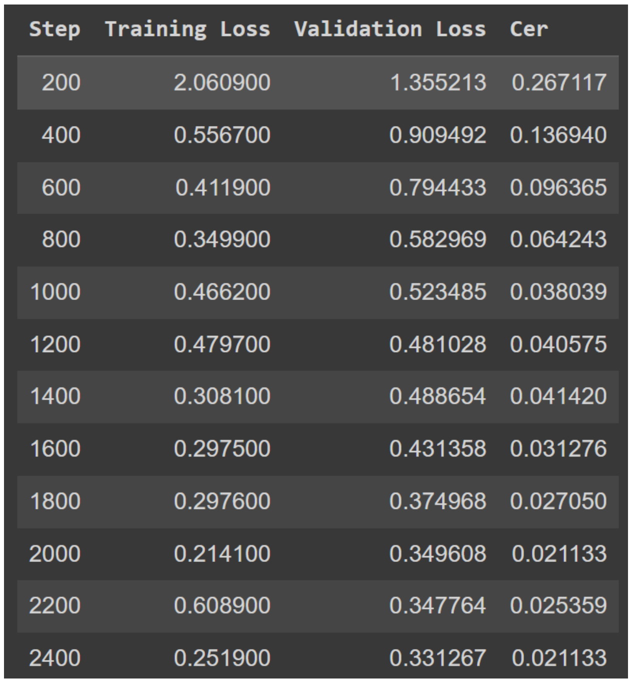

JSON File With Labels and Image File Names
Extract Label and File Name for the Image
Synthetic Image Generator
Folder With All Images
Use File Name to Retrieve Image
Microsft TrOCR From HuggingFace
Image and Label Pre-Processor for TrOCR
Convert Label and Image to Tensors using the Processor
Creating Dataset for Training
Use Pyotch Library to Generate a Dataset Class
Create Array of Tensors for Correct Answers and Labels

Figure 25: After the synthetic dataset is created and parsed into Python, it then needs to be converted into a format that can be used for training. The TrOCR model on Hugging Face provides a processor that allows these labels and images to be turned into tensors, which can then be used for our training dataset.

This Dataset class is called twice, once to create a training dataset and evaluation dataset. The training dataset is the larger dataset and is used for giving the model an image and performing backpropagation and gradient descent by comparing the model's output to the correct label. The evaluation is a smaller dataset of values where the model is given an image and the output is compared to the correct label but no gradient descent is performed. This is to test for any potential over-fitting from the training dataset and to verify the model can give correct output for images it was not trained on.

3.5.3.2 Fine-tuning using Seq2Seq Trainer

Once the two datasets have been created, they can now be used to start fine-tuning our model. As explained in section 2.2.4.3, this process would involve using Seq2SeqTrainer and passing in our datasets as well as other training parameters that have already been found to be best for this model. In total, 40,000 images (evenly split over the 7 image categories), were used for the dataset, with 32,000 for training and 8,000 reserved only for validation.



Step	Training Loss	Validation Loss	Cer
200	2.060900	1.355213	0.267117
400	0.556700	0.909492	0.136940
600	0.411900	0.794433	0.096365
800	0.349900	0.582969	0.064243
1000	0.466200	0.523485	0.038039
1200	0.479700	0.481028	0.040575
1400	0.308100	0.488654	0.041420
1600	0.297500	0.431358	0.031276
1800	0.297600	0.374968	0.027050
2000	0.214100	0.349608	0.021133
2200	0.608900	0.347764	0.025359
2400	0.251900	0.331267	0.021133

Figure 26: The following table was generated by the Seq2SeqTrainer. An evaluation was done every 200 steps against a set of data the model was not trained on to verify the model was not overfitting on the test data. Training loss is the loss from the train dataset, validation loss is from the evaluation dataset, and Cer is the character error rate from validation.

3.5.3.3 Evaluation Against Parachute Data

Once the model was trained on the synthetic dataset, we needed to verify that it still worked on the actual parachute dataset. Although our synthetic dataset was made to mirror the types of fractions we would see in the parachute documents as much as possible, saw previously how overfitting on the data led to incorrect results so still needed to evaluate the model on the real-world examples to truly know if the model was ready. We tested a wide variety of different types of fractions, including some typed fractions alongside the handwritten types, to verify the model was outputting correct fractions.

The results of the validation were satisfactory as we were getting correct results from passing in

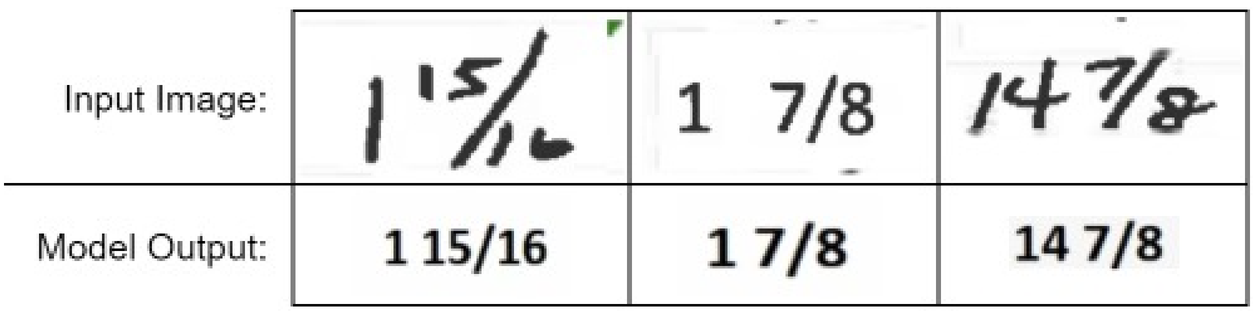

Input Image:
Model Output:
1 15/16
1 7/8
14 7/8

Figure 27: The pictures here are examples of fractions from the parachute data that were then passed into the model. The top row is all images that come from the parachute document, and the bottom row is all the outputs that the model predicted the text was based on the input image.

these images. This allowed us to conclude that this fine-tuned model was ready to be used in our program.

3.6 Making a Cell Classifier Model

As we saw from the fraction model training, the OCR models are very susceptible to overfitting on their trained datasets, leading to our fraction models to convert every text into a fraction, our regular OCR models to miss fractions completely, etc. This was a problem as the documents often contained a mixture of different types of text, including written and handwritten text. Because of this mixing of text types, we needed to instead create models that were good for individual tasks and then delegate them to translate the texts they were trained for. This meant we also needed to create a way to classify the types of text beforehand to know which model it should be passed onto. We solved this by using Google's ViT model to classify every text type beforehand so our program could then delegate images to the appropriate model.

3.6.1 Cell Classifier Dataset Generation

With the goal of using specifically tuned OCR models for the different types of cells, we needed a way to train a classifier on the four types of data that can be within a given text box: typed text, handwritten text, handwritten numbers, and blank cells. We generated a synthetic dataset to help train a classifier that would detect the different types of text. The aforementioned synthetic mixed number generator was used to generate the handwritten numbers for this dataset. To generate the handwritten text, we selected images from the IAM Handwriting Database, which is an online database of images of handwritten words that were written by 657 different people [Marti and Bunke., 2002]. To generate the typed text, we used a Python library called TextRecognitionDataGenerator to generate words picked from a list of the 1000 most common words in English. Finally, the blank spaces were created by using OpenCV to generate a blank image. The labels for the dataset are: typed, printed, number, and blank.

3.6.2 Fine-tuning Cell Classifier Model

Similar to the TrOCR model, the training used Seq2SeqTrainer with specific parameters from the guides that were best suited for the specific model. The dataset was made in the same way as the one for TrOCR using the four different image types, with 40,000 images (evenly split between the 4 categories), being split in an 8:2 ratio for training and evaluating. Once the model was done, it was then integrated into the program to evaluate the results using the actual parachute data. We then made sure that each of the cells was being passed into the correct model and translated properly by all of our models.

3.7 Converting Processed Images Into XLSX Files

Implemented in ConvertImagesToXLSX.py

Generating XLSX files from our newly cleaned up images (resized, deskewed, dewarped) is a non-trivial multi-step process. To better determine all of the content on a given datasheet, we determined that it would be best to process each cell separately. This allows us to focus on processing many smaller/simpler images with higher accuracy, rather than attempting to perform OCR on the full sheet. This also allows us to more easily represent the datasheet as an Excel file, since we already have all of the content broken up by cell.

The first step in the process is to locate the boundary of each cell in the image using various image processing techniques. Next, we verify the image is in the correct orientation and rotate it 90 degrees if needed. We then process each cell in the image one by one. Within each cell, we find groups of text and process each group separately. Each text group is classified as one of "blank", "printed", "handwritten", or "fraction" and the text group is passed to the appropriate OCR model. Once each text group within a cell has been processed, the resulting strings are stitched back together to produce the full contents of the cell. Once we have performed OCR on each cell within an image, we generate an Excel file containing the extracted data.

3.7.1 Finding All Cells on a Page

To begin the process of locating cells on a page, we first simplify the image by converting it into binary. This step requires three sub-steps: generating a grayscale image, blurring the image to reduce noise, and lastly using an adaptive threshold function to generate the binary image. With the binary image in hand, our next goal is to produce a reduced binary image containing only the edges of cells. We utilize a few simple morphological transformations to extract both horizontal and vertical lines then combine the results to produce the target image. See an example image here 28.

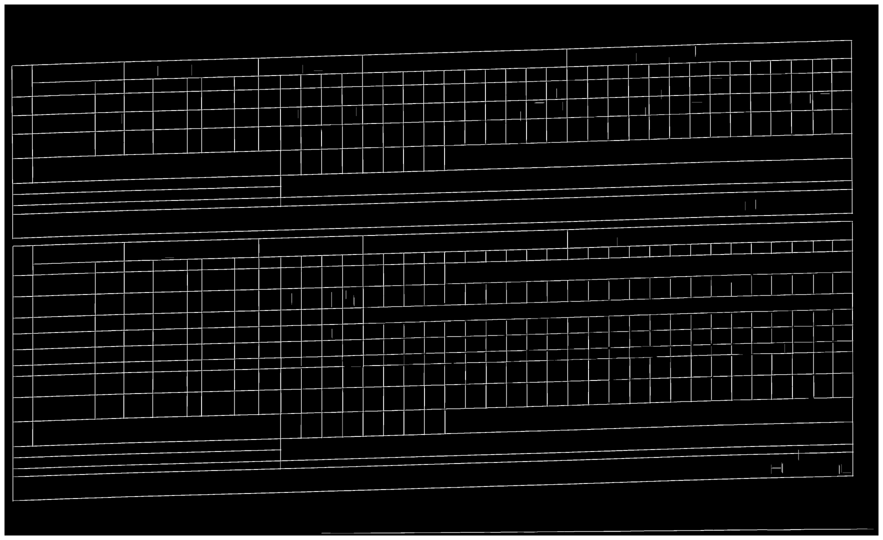


Figure 28: An image of a datasheet that has been converted to binary and and had the cell edges extracted.

With our new binary image of cell edges, the next step is to find the boundary of all individual cells. This is done using OpenCV's ”findContours” function, which returns a list of all contours in the image as well as a hierarchy containing topological information about the contours. The list from ”findContours” includes all contours, not just the ones we want. So, we need to filter this list to keep only the contours around cells. We can do this by only keeping those that have a parent contour but no grandparent contour. That is, contours that were one level down from the outermost level and none that were two levels down or lower. This eliminated all contours but except the contours around cells.

3.7.2 Final Preprocessing and Datasheet Check

After finding all of our cell contours, we can now do a few final checks to make sure the page is ready to be processed. Even after our image pre-processing, a few of the datasheets were still rotated by exactly 90 degrees out of the correct orientation. We can determine whether or not an image is rotated on its side by evaluating the average aspect ratio of all contour bounding boxes. By manual inspection of our training data we were able to determine that on average a text box is going to be about five times wider than it is tall. If the average aspect ratio falls far enough below this value we conclude the image must be rotated. If we conclude the image is rotated, we undo this by rotating it by 90 degrees.

One last step is to ensure that the document we are processing is actually capable of being converted into an Excel sheet. Because there are multiple types of documents that may be scanned but our model can only process spreadsheets, some documents need to be excluded. For example, we saw some documents with a series of bar-codes related to the parachute that is being tested. We check if a minimum of twenty cell contours, a somewhat arbitrary number, are detected on a page. If there isn't at least twenty contours then we determine that the page was not suitable for our model. If this step is passed, we continue onto Excel sheet generation.

3.7.3 Breaking up Cell Contents for OCR

There are cases where there is two or more different types of text within a cell. One example can be seen in figure 29. This makes it difficult to do OCR on the whole cell, so instead we opted to break each cell into clusters of words to process separately. This was done by converting the cell into a binary image, removing the cell borders, and then drawing contours around all clusters of words in the remaining image.

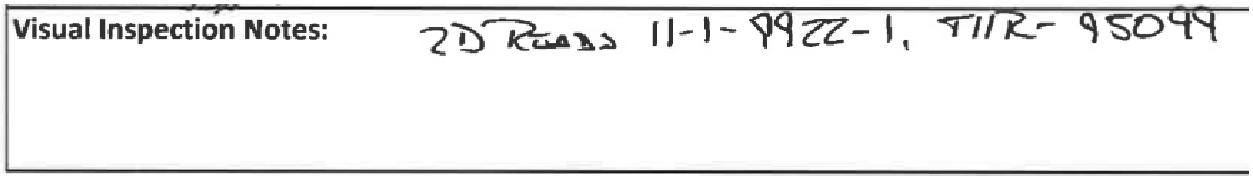

Visual Inspection Notes: 2D [illegible] 11-1-99ZZ-1, T/R- 95099

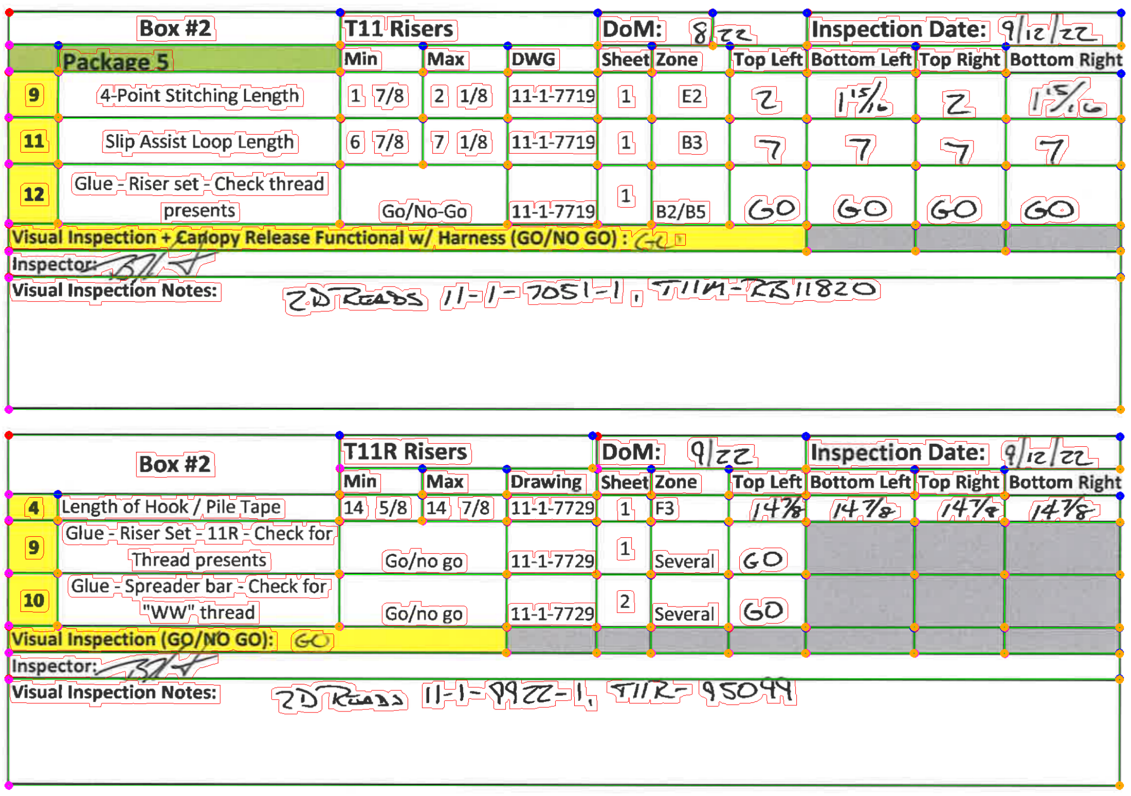
Figure 29: An image of a cell containing both printed text and handwritten numbers.

3.7.4 Creating Images for Debugging Purposes

In order to visually verify that our contour detection methods are working well, we generate a debug image with all of the cell and word contours overlaid on the original image. We can then manually inspect the result to verify that all text boxes have been identified and also that the cell contents have been broken up. See 30 for an example.

3.7.5 Performing OCR

We separately classify each word contour found in a cell to determine what type of text it is. The possible options are "printed", "handwritten", "fraction", or "empty". Once we have determined what type of text is within the word contour, we select the appropriate fine-tuned model to use for OCR. For each cell, we iterate through each word contour found and append their OCR outputs into a single string. To ensure that the different word contours appear in the same order in the excel sheet as they do in the original



Box #2		T11 Risers			DoM: 8/22			Inspection Date: 9/12/22		
	Package 5	Min	Max	DWG	Sheet	Zone	Top Left	Bottom Left	Top Right	Bottom Right
9	4-Point Stitching Length	1 7/8	2 1/8	11-1-7719	1	E2	2	1 15/16	2	1 15/16
11	Slip Assist Loop Length	6 7/8	7 1/8	11-1-7719	1	B3	7	7	7	7
12	Glue - Riser set - Check thread presents	Go/No-Go		11-1-7719	1	B2/B5	GO	GO	GO	GO
Visual Inspection + Canopy Release Functional w/ Harness (GO/NO GO) : GO								▨	▨	▨
Inspector: [illegible]										
Visual Inspection Notes: 2D Reads 11-1-7051-1, T11M-RB11820										

Box #2		T11R Risers			DoM: 9/22			Inspection Date: 9/12/22		
		Min	Max	Drawing	Sheet	Zone	Top Left	Bottom Left	Top Right	Bottom Right
4	Length of Hook / Pile Tape	14 5/8	14 7/8	11-1-7729	1	F3	14 7/8	14 7/8	14 7/8	14 7/8
9	Glue - Riser Set - 11R - Check for Thread presents	Go/no go		11-1-7729	1	Several	GO	▨	▨	▨
10	Glue - Spreader bar - Check for "WW" thread	Go/no go		11-1-7729	2	Several	GO	▨	▨	▨
Visual Inspection (GO/NO GO): GO				▨	▨	▨	▨	▨	▨	▨
Inspector: [illegible]										
Visual Inspection Notes: 2D Reads 11-1-9922-1, T11R-95044										

Figure 30: An image of a datasheet with overlays of cell edge contours (green) and word contours (red).

document, all of the word contours are sorted based on their location.

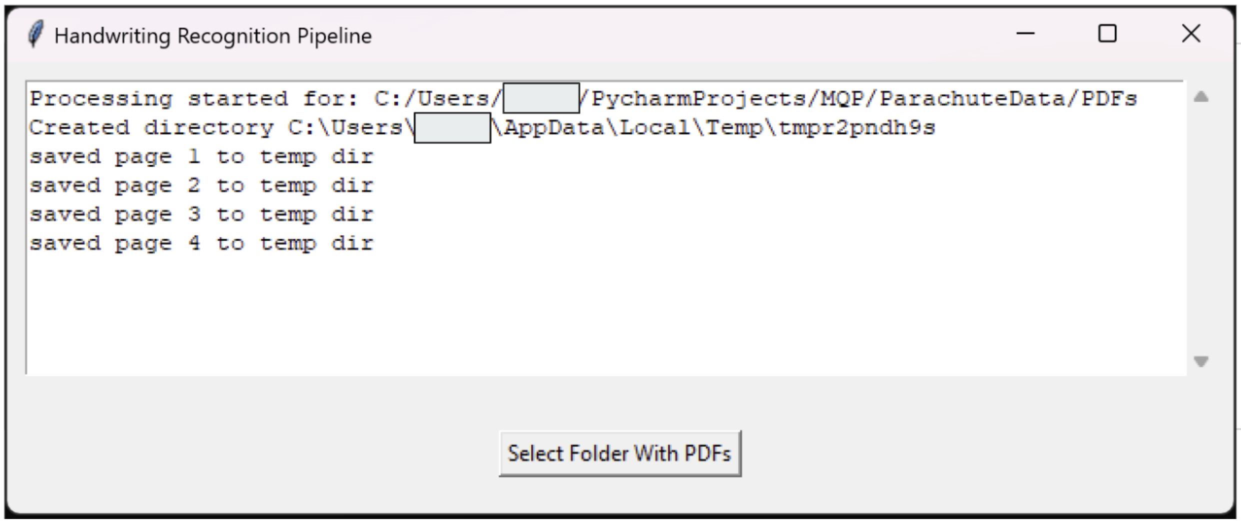
3.7.6 Excel Sheet Generation

Once we have performed OCR on each word group within the document, the last step is to generate an Excel sheet using the gathered data. By keeping track of the pixel coordinates of each cell contour we can figure out the correct row and column to place each cell. We can also determine if cells are merged or not. Merged cells will take up more then one column and/or row. In addition, we also determine the correct background color of each cell. Finally, we iterate through all of our previously gathered data to generate the Excel sheet.

3.8 Making a GUI and Generating an Executable

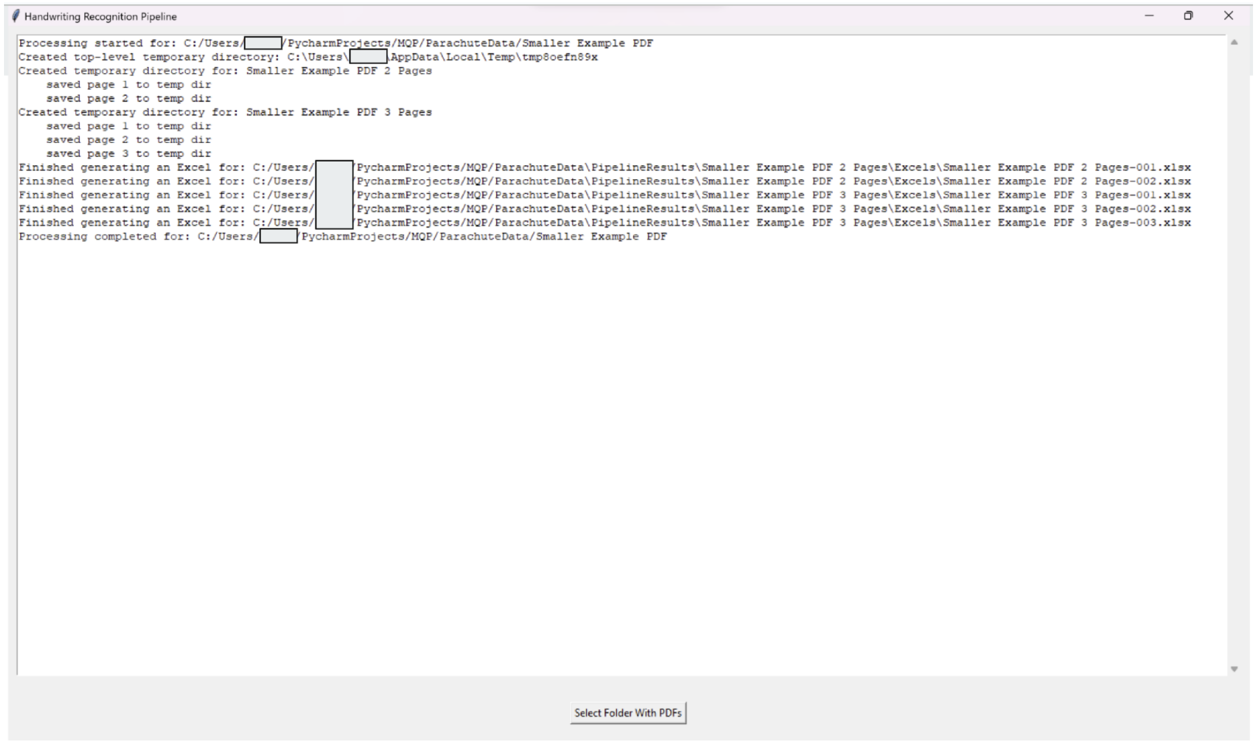
3.8.1 Making a GUI

In order to facilitate ease and enjoyment of use, we have designed a graphical user interface (GUI) for the project using tkinter, a part of the Python standard library. This is intended to facilitate the easy uploading of files as well as provide a way of monitoring the progress of the running program. The following are screenshots showing standard usage of the GUI 31 32.



```
Processing started for: C:/Users/[redacted]/PycharmProjects/MQP/ParachuteData/PDFs
Created directory C:\Users\[redacted]\AppData\Local\Temp\tmpr2pndh9s
saved page 1 to temp dir
saved page 2 to temp dir
saved page 3 to temp dir
saved page 4 to temp dir
```

Figure 31: An image of the graphical user interface beginning the processing of a large folder.

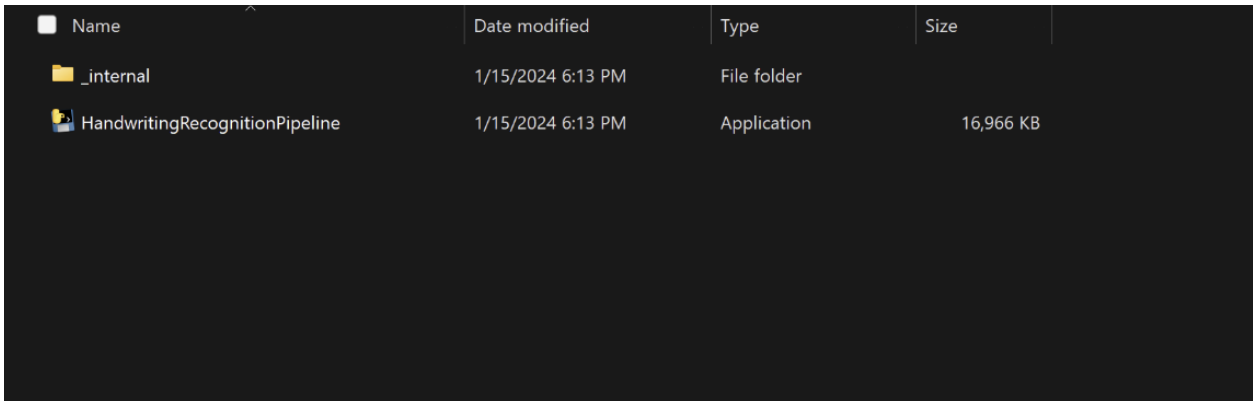


```
Processing started for: C:/Users/[redacted]/PycharmProjects/MQP/ParachuteData/Smaller Example PDF
Created top-level temporary directory: C:\Users\[redacted]\AppData\Local\Temp\tmp8oefn89x
Created temporary directory for: Smaller Example PDF 2 Pages
    saved page 1 to temp dir
    saved page 2 to temp dir
Created temporary directory for: Smaller Example PDF 3 Pages
    saved page 1 to temp dir
    saved page 2 to temp dir
    saved page 3 to temp dir
Finished generating an Excel for: C:/Users/[redacted]/PycharmProjects/MQP/ParachuteData\PipelineResults\Smaller Example PDF 2 Pages\Excels\Smaller Example PDF 2 Pages-001.xlsx
Finished generating an Excel for: C:/Users/[redacted]/PycharmProjects/MQP/ParachuteData\PipelineResults\Smaller Example PDF 2 Pages\Excels\Smaller Example PDF 2 Pages-002.xlsx
Finished generating an Excel for: C:/Users/[redacted]/PycharmProjects/MQP/ParachuteData\PipelineResults\Smaller Example PDF 3 Pages\Excels\Smaller Example PDF 3 Pages-001.xlsx
Finished generating an Excel for: C:/Users/[redacted]/PycharmProjects/MQP/ParachuteData\PipelineResults\Smaller Example PDF 3 Pages\Excels\Smaller Example PDF 3 Pages-002.xlsx
Finished generating an Excel for: C:/Users/[redacted]/PycharmProjects/MQP/ParachuteData\PipelineResults\Smaller Example PDF 3 Pages\Excels\Smaller Example PDF 3 Pages-003.xlsx
Processing completed for: C:/Users/[redacted]/PycharmProjects/MQP/ParachuteData/Smaller Example PDF
```

Figure 32: An example usage of the GUI on a small sample, to show a complete output.

3.8.2 Generating an Executable

To create an executable, we used a python library called "PyInstaller". This application makes a Python program into an executable file that can be run on whichever operating system PyInstaller is being run on [PyI, 2024]. By gathering all of the dependencies of the target application into one folder, the application can more easily be run on different machines.



Name	Date modified	Type	Size
_internal	1/15/2024 6:13 PM	File folder	
HandwritingRecognitionPipeline	1/15/2024 6:13 PM	Application	16,966 KB

Figure 33: The result of running PyInstaller. The folder "_internal" holds the dependencies of the executable file, and must be in the same folder as the executable file when it is run.

4 Results

4.1 Pipeline Summary

First, each page of the input document image was split into its own image file. The images were resized and deskewed. Each image was dewarped as described in 3.4. A binary image representing the cell edges was generated using a process very similar to the one described in 3.4.1. We next used OpenCV to identify contours in the binarized image. These contours represent the boundaries of cells within the image. Contours were filtered based on their size and topological location to keep only those that were likely cells in a table.

Because some pages of our input PDFs were rotated by 90 degrees, an aspect ratio analysis was conducted on the filtered contours to determine whether the image required rotation or not. A mean aspect ratio was calculated, and if it fell below a certain threshold, the image was rotated 90 degrees clockwise as described in 3.7.2.

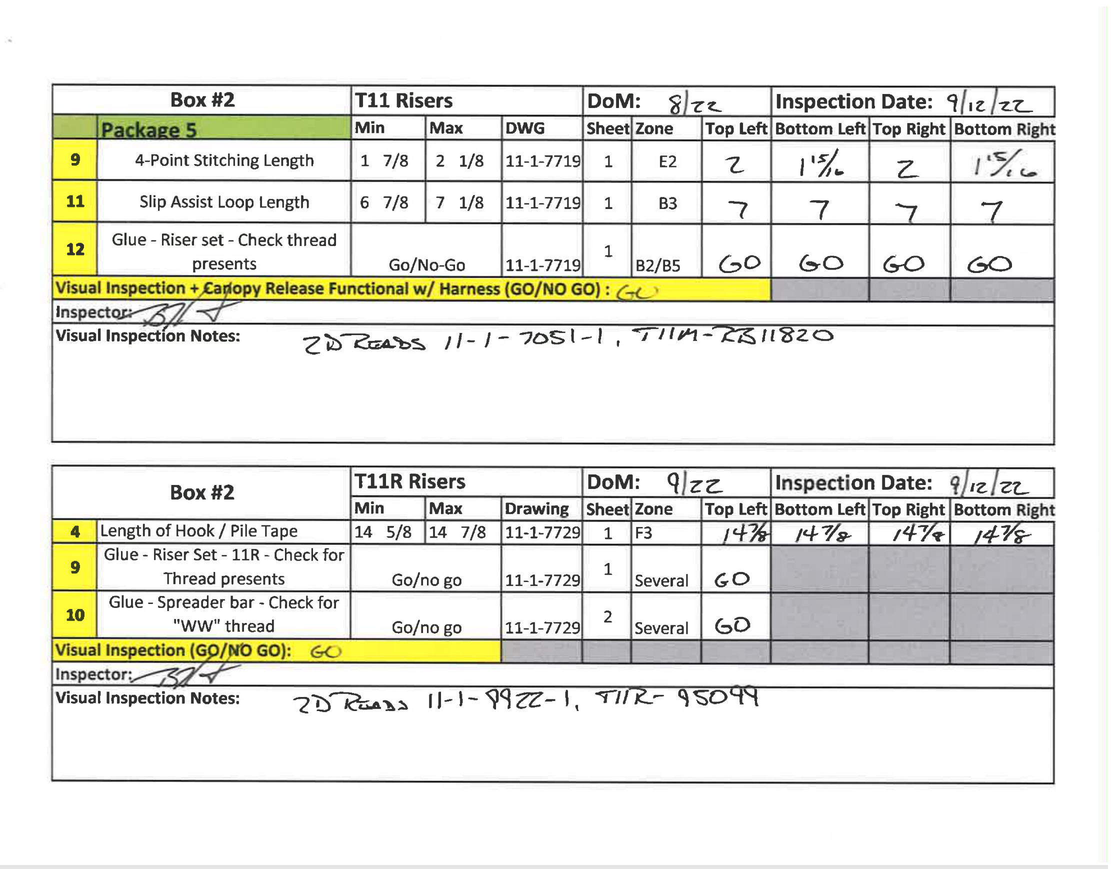
Each cell was split into groups of words which were converted to text separately. Each word group was fed into our classifier model to be categorized as one of handwritten numbers, typed text, handwritten text, or a blank group. An appropriately fine-tuned ORC model would then be applied to the word group.

Once all cells have had OCR performed on them, a final output Excel sheet is generated with the same format as the scanned datasheet.

4.2 Quality of Produced XLSX Files

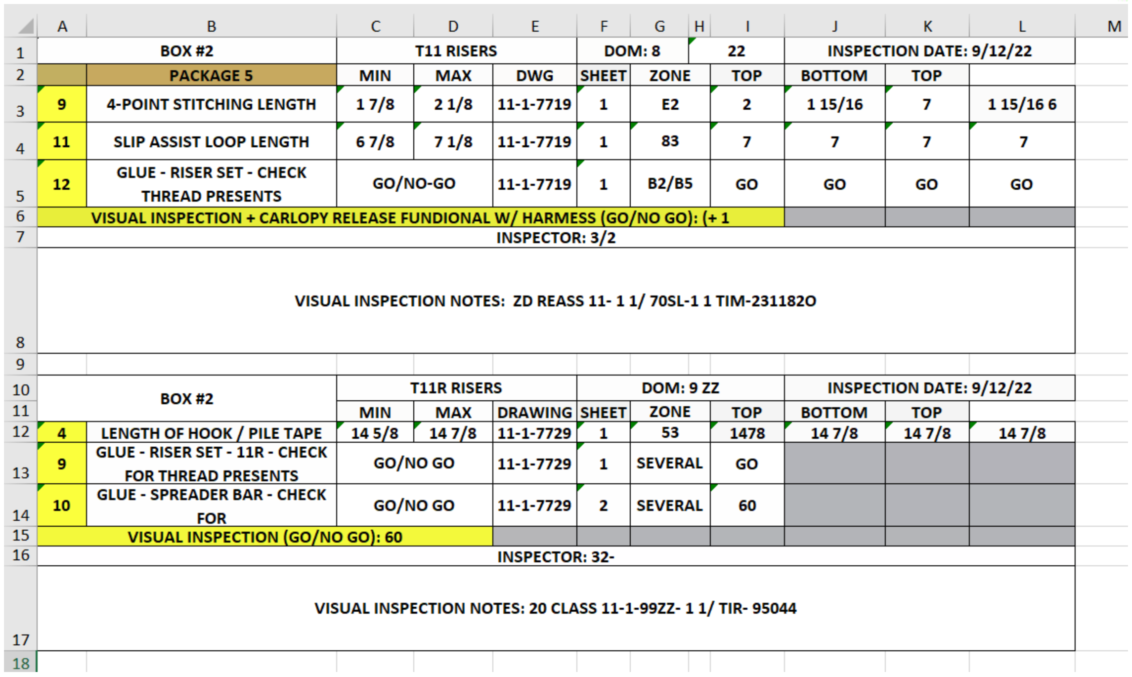
While our initial goal for this project was to completely automate the process of digitizing parachute datasheets, as we experimented and learned more, the many challenges associated with this goal became evident. See 4.4 for specific details.

On most documents, we can accurately capture most of the table layout (how cells are are laid out and merged), most of the printed text, and much of the handwritten text. We also color cells to (roughly) match their colors in the datasheets. Below 34 is a standard example showing an input datasheet and the resulting Excel output. A brief visual inspection shows that the digitized document captures most (but not all) of the information in the original datasheet.



Box #2		T11 Risers			DoM: 8/22		Inspection Date: 9/12/22			
	Package 5	Min	Max	DWG	Sheet	Zone	Top Left	Bottom Left	Top Right	Bottom Right
9	4-Point Stitching Length	1 7/8	2 1/8	11-1-7719	1	E2	2	1 15/16	2	1 15/16
11	Slip Assist Loop Length	6 7/8	7 1/8	11-1-7719	1	B3	7	7	7	7
12	Glue - Riser set - Check thread presents	Go/No-Go		11-1-7719	1	B2/B5	GO	GO	GO	GO
Visual Inspection + Canopy Release Functional w/ Harness (GO/NO GO) : GO										
Inspector: [illegible]										
Visual Inspection Notes: 2D READS 11-1-7051-1, T11M-2311820										

Box #2		T11R Risers			DoM: 9/22		Inspection Date: 9/12/22			
		Min	Max	Drawing	Sheet	Zone	Top Left	Bottom Left	Top Right	Bottom Right
4	Length of Hook / Pile Tape	14 5/8	14 7/8	11-1-7729	1	F3	14 7/8	14 7/8	14 7/8	14 7/8
9	Glue - Riser Set - 11R - Check for Thread presents	Go/no go		11-1-7729	1	Several	GO			
10	Glue - Spreader bar - Check for "WW" thread	Go/no go		11-1-7729	2	Several	GO			
Visual Inspection (GO/NO GO): GO										
Inspector: [illegible]										
Visual Inspection Notes: 2D READS 11-1-9922-1, T11R- 95044										



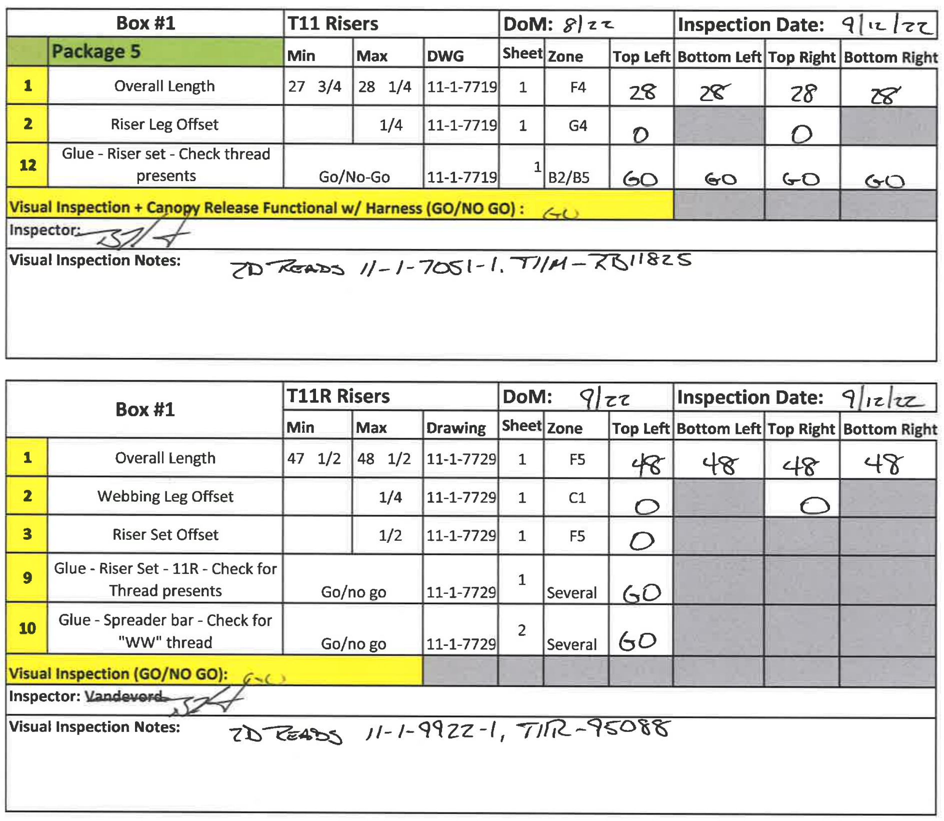
	A	B	C	D	E	F	G	H	I	J	K	L	M
1	BOX #2		T11 RISERS			DOM: 8			22	INSPECTION DATE: 9/12/22			
2		PACKAGE 5	MIN	MAX	DWG	SHEET	ZONE		TOP	BOTTOM	TOP		
3	9	4-POINT STITCHING LENGTH	1 7/8	2 1/8	11-1-7719	1	E2		2	1 15/16	7	1 15/16 6	
4	11	SLIP ASSIST LOOP LENGTH	6 7/8	7 1/8	11-1-7719	1	83		7	7	7	7	
5	12	GLUE - RISER SET - CHECK THREAD PRESENTS	GO/NO-GO		11-1-7719	1	B2/B5		GO	GO	GO	GO	
6	VISUAL INSPECTION + CARLOPY RELEASE FUNDIONAL W/ HARMESS (GO/NO GO): (+ 1												
7	INSPECTOR: 3/2												
8	VISUAL INSPECTION NOTES: ZD REASS 11- 1 1/ 70SL-1 1 TIM-231182O												
9													
10	BOX #2		T11R RISERS			DOM: 9 ZZ				INSPECTION DATE: 9/12/22			
11			MIN	MAX	DRAWING	SHEET	ZONE		TOP	BOTTOM	TOP		
12	4	LENGTH OF HOOK / PILE TAPE	14 5/8	14 7/8	11-1-7729	1	53		1478	14 7/8	14 7/8	14 7/8	
13	9	GLUE - RISER SET - 11R - CHECK FOR THREAD PRESENTS	GO/NO GO		11-1-7729	1	SEVERAL		GO				
14	10	GLUE - SPREADER BAR - CHECK FOR	GO/NO GO		11-1-7729	2	SEVERAL		60				
15	VISUAL INSPECTION (GO/NO GO): 60												
16	INSPECTOR: 32-												
17	VISUAL INSPECTION NOTES: 20 CLASS 11-1-99ZZ- 1 1/ TIR- 95044												
18													

Figure 34: An example datasheet (top) and the resultant XLSX output file (bottom).

As it seems that we cannot achieve perfectly faithful digitizations of these parachute datasheets, our new goal for our system is to provide a good starting point for a human to touch up minor mistakes by hand. It is our opinion that the formatting and the coloring of each of the cells alone is a major time saving step in such a digitization workflow.

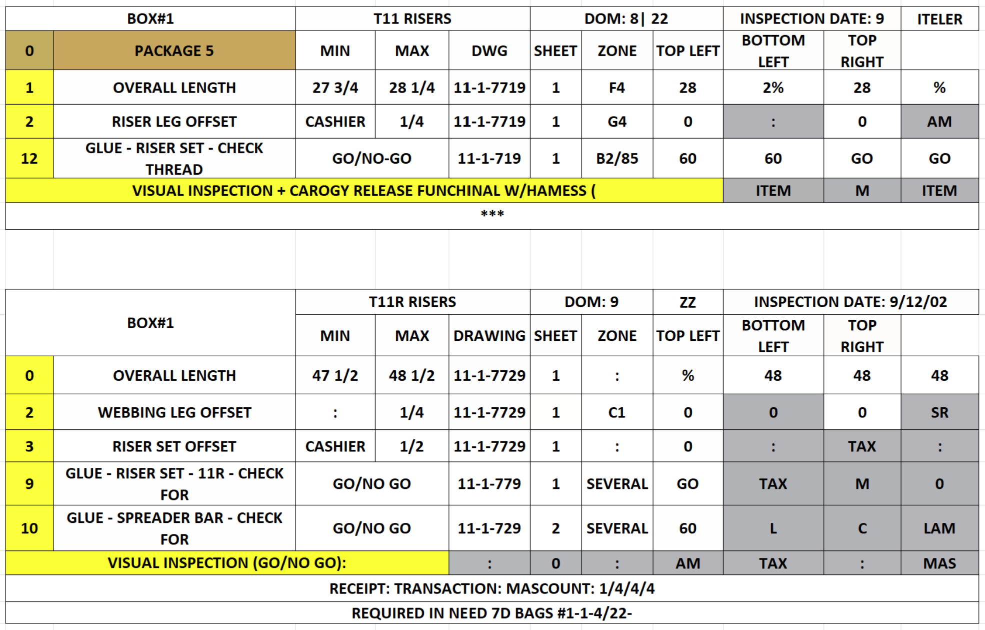
4.3 Comparison of Different Models Used

At the start of this project we wanted to first see if we could use one OCR model in order to translate all of our different types of text (typed text, handwritten text, handwritten numbers, and blank cells). So we tried using a pre-trained TrOCR model from Hugging Face. A sample output can be seen in figure 35. It can be seen that while a majority of the printed text is translated correctly, the model struggled with a lot of the handwritten text. The model would also "hallucinate" random text when presented with blank cells.



Box #1		T11 Risers			DoM: 8/22		Inspection Date: 9/12/22			
	Package 5	Min	Max	DWG	Sheet	Zone	Top Left	Bottom Left	Top Right	Bottom Right
1	Overall Length	27 3/4	28 1/4	11-1-7719	1	F4	28	28	28	28
2	Riser Leg Offset		1/4	11-1-7719	1	G4	0		0	
12	Glue - Riser set - Check thread presents	Go/No-Go		11-1-7719	1	B2/B5	GO	GO	GO	GO
Visual Inspection + Canopy Release Functional w/ Harness (GO/NO GO) : GO										
Inspector: [illegible]										
Visual Inspection Notes: ZD READS 11-1-7051-1, T11M-ZB11825										

Box #1		T11R Risers			DoM: 9/22		Inspection Date: 9/12/22			
		Min	Max	Drawing	Sheet	Zone	Top Left	Bottom Left	Top Right	Bottom Right
1	Overall Length	47 1/2	48 1/2	11-1-7729	1	F5	48	48	48	48
2	Webbing Leg Offset		1/4	11-1-7729	1	C1	0		0	
3	Riser Set Offset		1/2	11-1-7729	1	F5	0			
9	Glue - Riser Set - 11R - Check for Thread presents	Go/no go		11-1-7729	1	Several	GO			
10	Glue - Spreader bar - Check for "WW" thread	Go/no go		11-1-7729	2	Several	GO			
Visual Inspection (GO/NO GO): GO										
Inspector: ~~Vandevord~~ [illegible]										
Visual Inspection Notes: ZD READS 11-1-9922-1, T11R-95088										

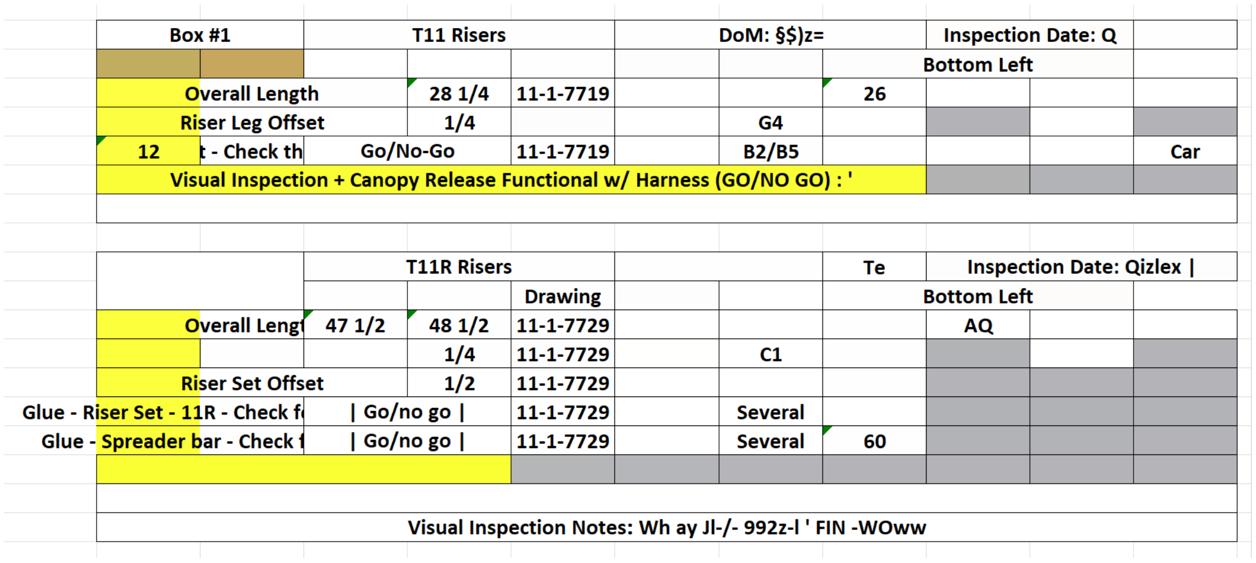


BOX#1		T11 RISERS			DOM: 8| 22			INSPECTION DATE: 9		ITELER
0	PACKAGE 5	MIN	MAX	DWG	SHEET	ZONE	TOP LEFT	BOTTOM LEFT	TOP RIGHT	
1	OVERALL LENGTH	27 3/4	28 1/4	11-1-7719	1	F4	28	2%	28	%
2	RISER LEG OFFSET	CASHIER	1/4	11-1-7719	1	G4	0	:	0	AM
12	GLUE - RISER SET - CHECK THREAD	GO/NO-GO		11-1-719	1	B2/85	60	60	GO	GO
VISUAL INSPECTION + CAROGY RELEASE FUNCHINAL W/HAMESS (								ITEM	M	ITEM
***										

BOX#1		T11R RISERS			DOM: 9		ZZ	INSPECTION DATE: 9/12/02		
		MIN	MAX	DRAWING	SHEET	ZONE	TOP LEFT	BOTTOM LEFT	TOP RIGHT	
0	OVERALL LENGTH	47 1/2	48 1/2	11-1-7729	1	:	%	48	48	48
2	WEBBING LEG OFFSET	:	1/4	11-1-7729	1	C1	0	0	0	SR
3	RISER SET OFFSET	CASHIER	1/2	11-1-7729	1	:	0	:	TAX	:
9	GLUE - RISER SET - 11R - CHECK FOR	GO/NO GO		11-1-779	1	SEVERAL	GO	TAX	M	0
10	GLUE - SPREADER BAR - CHECK FOR	GO/NO GO		11-1-729	2	SEVERAL	60	L	C	LAM
VISUAL INSPECTION (GO/NO GO):				:	0	:	AM	TAX	:	MAS
RECEIPT: TRANSACTION: MASCOUNT: 1/4/4/4										
REQUIRED IN NEED 7D BAGS #1-1-4/22-										

Figure 35: An example datasheet (top) along with the excel sheet output (bottom) using a pre-trained TrOCR model.

Another model we initial tried using was an off the shelf model called Tesseract (via PyTesseract) which is often touted for its ability to perform OCR on handwritten text. As seen in figure 36, the PyTesseract model was capable of performing OCR on cells where there was just printed text, however once handwriting was introduced the model failed. One example was the "DoM" cell, which contained a mixture of printed text and handwritten numbers. PyTesseract was able to interpret DoM, however it interpreted the date as a series of random symbols.



	Box #1		T11 Risers			DoM: §$)z=			Inspection Date: Q		
									Bottom Left		
	Overall Length			28 1/4	11-1-7719			26			
	Riser Leg Offset			1/4			G4				
	12	t - Check th	Go/No-Go		11-1-7719		B2/B5				Car
	Visual Inspection + Canopy Release Functional w/ Harness (GO/NO GO) : '										
			T11R Risers					Te	Inspection Date: Qizlex |		
					Drawing				Bottom Left		
	Overall Lengt		47 1/2	48 1/2	11-1-7729				AQ		
				1/4	11-1-7729		C1				
	Riser Set Offset			1/2	11-1-7729						
Glue - Riser Set - 11R - Check f			| Go/no go |		11-1-7729		Several				
Glue - Spreader bar - Check f			| Go/no go |		11-1-7729		Several	60			
	Visual Inspection Notes: Wh ay Jl-/- 992z-l ' FIN -WOww										

Figure 36: Output of our model using PyTesseract for OCR on a random datasheet.

Our final model used a combination of three models, a Google ViT model to classify the cell contents, an off the shelf TrOCR model to read most printed text, and a TrOCR model fine-tuned for handwritten numbers and fractions. The output of this combination of models was previously shown in 4.2. The final model does a much better job at performing OCR on all different text types, not sacrificing its the ability to interpret printed text in order to perform better on interpreting handwritten numbers. With the inclusion of the cell type classifier model, blank cells are left blank so no longer is there random strings of text or symbols where there should be blank space.

4.4 Failure Modes of Program and Problems in Spreadsheets

In this section we will try to enumerate most of the areas in which our program fails to produce a completely correct results. We believe it is possible to correct these errors in multiple ways. The most obvious is that more work could be done on this pipeline to improve it's performance. The second, is that the initial spreadsheets could be improved so they are easier to process.

4.4.1 Largest Problem: Accurate Handwriting Translation

The single greatest problem we encountered is that despite our best efforts, we are still unable to recognize some handwritten text. While we are often able to recognize clear writing that stays within cell boundaries, not all of the text on these datasheets conforms to these descriptions. Below are some examples of text we struggle to translate correctly 37. This is representative of the more general fact that handwriting recognition is not yet a completely solved problem. Some of the text is also so garbled that most humans would struggle to decipher it. This leaves little hope for most OCR models.

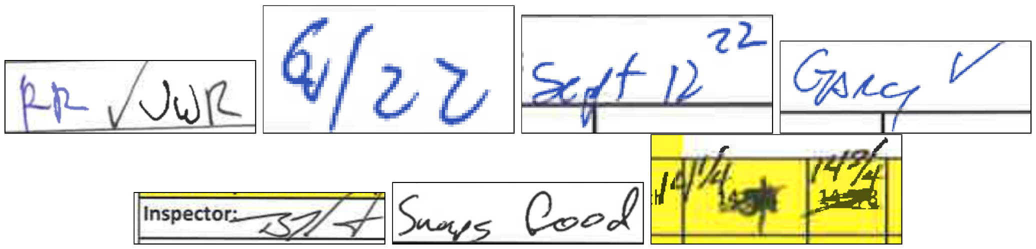


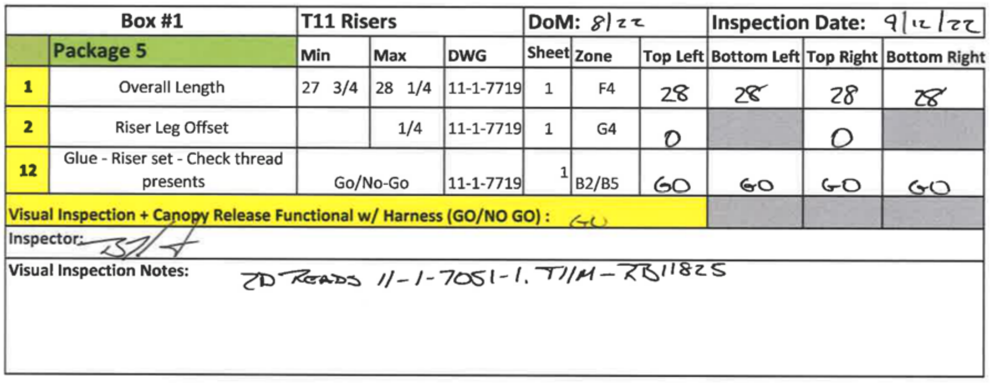
Figure 37: Example images of handwriting that is difficult to process.

4.4.2 Other Common Problems

Some other problems we noted when testing our model on the provided datasheets are described in the below sections.

4.4.2.1 Missing Lines in Tables Where Printed Text Overflows a Cell

On some tables the text contained within a cell is too large for the cell to accommodate it. This causes the cell line on the right side of the cell to be removed. When the cell line is removed, our program no longer detects the cell properly.



	Box #1	T11 Risers		DoM: 8/22		Inspection Date: 9/12/22			
	Package 5	Min	Max	DWG	Sheet	Zone	Top Left	Bottom Left	Top Right	Bottom Right
1	Overall Length	27 3/4	28 1/4	11-1-7719	1	F4	28	28	28	28
2	Riser Leg Offset		1/4	11-1-7719	1	G4	0		0	
12	Glue - Riser set - Check thread presents	Go/No-Go		11-1-7719	1	B2/B5	GO	GO	GO	GO

Visual Inspection + Canopy Release Functional w/ Harness (GO/NO GO) : GO

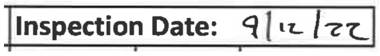
Inspector: [illegible]

Visual Inspection Notes: ZD READS 11-1-7051-1, T7/M-ZB11825

Figure 38: Example datasheet with a missing cell line in the upper right hand corner. The offending cell contains the text "Bottom Right".

4.4.2.2 Large Vertical Handwritten "1" Treated as Cell Line

Occasionally, vertical handwritten text that spans nearly the full height of the cell will get misinterpreted as as a cell line. Often this occurs when someone writes a date and includes a large nearly vertical "/" between parts of the date. Below is an example 38. Below is an example of such an occurrence 39.

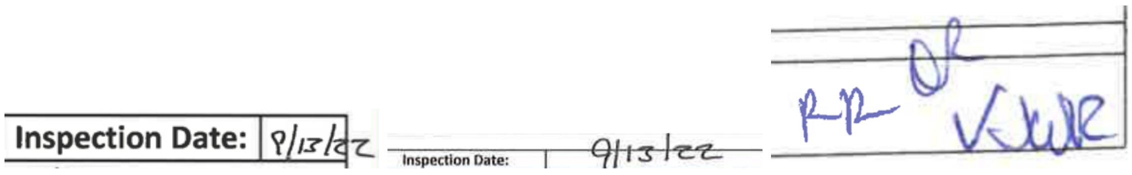


Inspection Date: 9/12/22

Figure 39: Example cell containing handwritten text that gets detected as a cell line. The first slash after the "9" is the offending character.

4.4.2.3 Handwritten Text Overflowing Cells

Occasionally, handwritten text will overflow the cell it is indented to be contained within. This leads to problems as our program only processes text within the boundary of cells. This happens to a minor extent in many places, but is often not significant enough to cause problems. Below are some more problematic examples 40.



Inspection Date: 9/13/22

Inspection Date: 9/13/22

[illegible]

Figure 40: Examples of cells with handwriting that extends past the boundary of the cell.

4.4.3 Rare Problems

While most of the common problems have been enumerated in the previous section, we have noticed a handful of rare (sometimes one-off) problems that our program is unable to properly deal with. Some are depicted in the figure below 41.

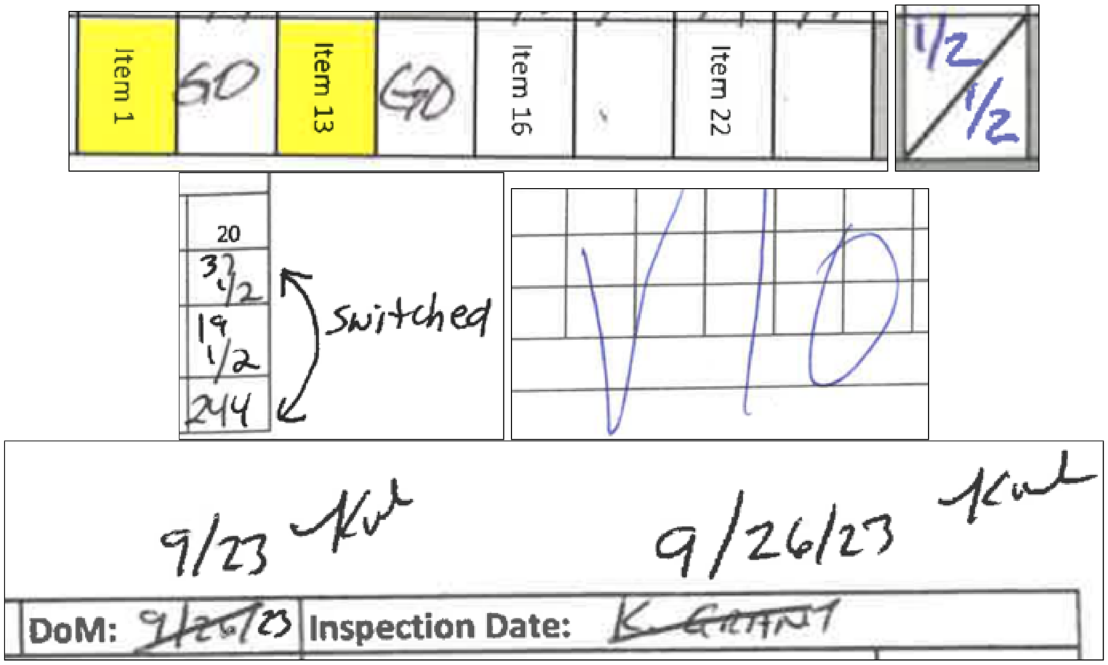


Figure 41: Examples of rare problems that can occur. In order they are: cells containing rotated printed text; cells with a diagonal slash to accommodate two fractions; arrows indicating entries should be swapped; big text written over the table; cells with crossed out handwriting.

5 Conclusion

This research endeavor has successfully addressed the extensive challenge of digitizing parachute data recorded in these physical fact sheets. By generating multiple synthetic datasets encompassing diverse fraction formats, we were able to fine-tune pre-existing OCR models to work for this specific problem set. Furthermore, using sophisticated image processing techniques, we could identify text cells within any imperfectly scanned parachute document. This allowed us not only to feed information into our different models that were specialized for specific text tasks, but also to use these identified cells to reproduce precise digital copies of the original pages in the form of Excel spreadsheets. The culmination of this effort is an automated executable program capable of transforming scanned PDFs into structured, analyzable spreadsheet data. By digitizing this large amount of information, analysts at the Natick Army Lab will be able to gain a more refined understanding of parachute life cycles. This will facilitate enhanced decision-making for important questions, such as "How long can a parachute be used safely before having to be discarded?" This approach ensures that critical insights can be extracted efficiently, ultimately bolstering the integrity and effectiveness of military parachute operations.

Despite all the work done here, there are still some tasks left to be completed to further help the Army Lab with their work. One would be to tackle the edge cases and other common problems, such as misplaced text, text that overflows from the cell, etc. Another would be to try to automatically organize the Excel documents based on parachutes ahead of time, making an analyst's job even easier as they would not need to do the extra step of organizing all of the information based on parachute number beforehand. Including confidence values and a log file for how accurate the model believes it was could also be helpful as it could allow a human to then check the least confident values so they could manually verify and edit any incorrect values. Finally, turning these physical sheets into digital forms that field operatives could use on tablets/laptops could help with future analysis as it would not require the extra steps of converting scanned pages into digital files.

Appendices

A How to Run Our Executable

The project code is hosted at: https://github.com/Handwriting-MQP/Parachute-Converter

Step 1: Select PDF Document to Scan

The user selects the document(s) that they want to be converted from the parachute data.

Step 2: Image Preprocessing : Deskew and PDF to PNG

The selected document(s) pass through the image preprocessing steps.

First deskewing the PDF and then converting it to an image (PNG)

Step 3: Text Box Detection

The newly converted PNG's text boxes are detected.

Step 4: Text Detection

Within these newly detected text boxes, physical text is identified.

Step 5: Generate XLSX

Utilizing the detected text and text boxes, the program is able to read this information and output a file in XLSX format.

B Procrustes Problems

Procrustes is a robber in Greek legend. In his story, Procrustes would kidnap people and shackle them to an iron bed. If a person was too short for the bed, meaning their feet did not reach the end of the bed, he would stretch their body with hammers, until they fit. If a person was too large for the bed, he would cut parts of their legs off until they fit. In both cases, the victim would die. When we want to skew an image, we do something similar, we shrink or stretch it until the output image is desired. Let's try to mathematical define what Procrustes is doing to implement it into images.

There are three different parts to the Procrustes story, the traveler that is being tortured, the iron bed, and the "treatment" being given by Procrustes. Let's label the travel as a matrix X_1, the thing we want to stretch or shrink, the bed as a matrix X_2, the marker for which we will stretch X_1, and T, the matrix that fits X_1 to X_2. For each matrix, X_1 has dimensions $N \times P_1$,X_2 has dimensions $N \times P_2$, and T has dimensions $P_1 \times P_2$ [Gower and Dijksterhuis, 2004]. In the simplest form, all forms of Procrustes problems attempt to find a matrix T that fulfills the following statement:

$$min||X_1T - X_2||_F \tag{22}$$

For the rest of our discussion in this section, $\|\cdot\|$ is the Frobenius norm. As an example, let's discuss some forms of Procrustes problems that are simpler than the ones done for image transformations. Let's begin with Two-set Procrustes problems, which are the most basic terms of these problems. Here are some assumptions and criteria that are common in all two-set Procrustes problems.

The most common criterion is called the least squared criterion. This defines the best matrix T such that the following is minimized:

$$S = ||X_1T - X_2||_F \tag{23}$$

In this instance and unless otherwise stated, T has no constraint. As in equation 23, X_1 is matched to X_2, it is common to treat both matrices symmetrically. There are two different ways to do this. The first way is called the two-sided variant, which creates two different transform matrices T_1 and T_2 with R columns such that the following is minimized:

$$S = ||X_1T_1 - X_2T_2||_F \tag{24}$$

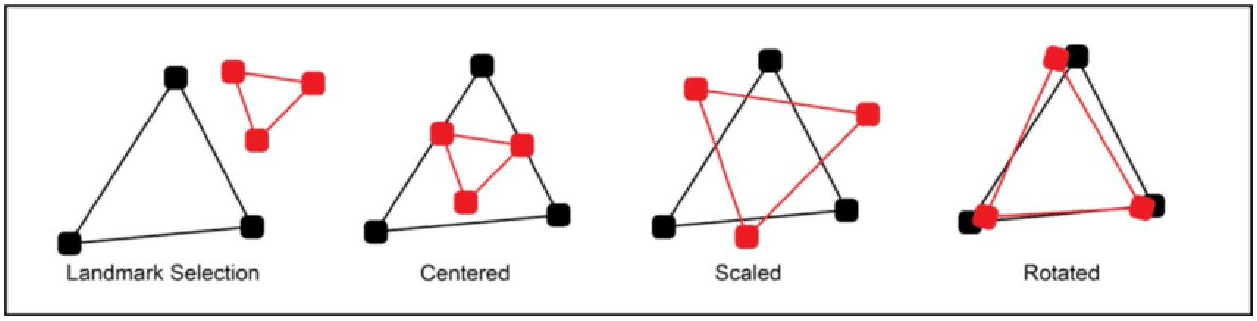


Figure 42: A simplified example of generalized Procrustes analysis modified from Zelditch et al., 2012. After landmarks are selected for each specimen, landmark configurations are then centered, scaled, and rotated such that the Procrustes distance between the configurations is minimized.

As a quick note, if no constraints are put on either T_1 or T_2, then equation 24 results in the trivial null solutions. [Gower and Dijksterhuis, 2004]. To avoid this, we can express a symmetric relation between X_1 and X_2 is with the following equation:

$$\frac{1}{4}S = ||X_1T - G||_F = ||X_2T - G||_F \tag{25}$$

In this case $G = \frac{1}{2}(X_1T + X_2T)$. This method avoids the issues above, that the only solution given was the trivial null solution, where the solution is the zero vector [Gower and Dijksterhuis, 2004].

The least squared method is the main criterion in all Procrustes problems. Here are a few other criteria that may be picked that are more common. The first one of these is called the RV coefficient. This coefficient has two main forms, but the one we will focus on takes the following form:

$$r_V^2 = \frac{(trace(X_2'X_1T))^2}{\text{trace}((X_1T_1)'(X_1T_1))trace((X_2T_2)'(X_2T_2))} \tag{26}$$

In this example, each matrix is assumed to be "strung out" by each column to form a single vector, which is called an uncentered correlation. [Gower and Dijksterhuis, 2004]. In equation 26, if $||X_1|| = ||X_1T||$, the denominator becomes independent from T. This leads to another criteria, called the inner-product criteria, which for the equation above will be:

$$trace(X_2'X_1T) \tag{27}$$

This criterion is normally used in the two-sided variant, specifically when T_1 and T_2 are orthogonal [Gower and Dijksterhuis, 2004]. This is because this is the only case where the correlation interpretation is available. Criteria 26 is different from other product-moment correlations in that no correlation for the mean is specified. This will be resolved later.

In many circumstances, translating either X_1 or X_2 can help make the morphing from X_1 to X_2 better. This is doable, as the shapes of either matrix or image will not be affected if their centers are at the origin or not [Gower and Dijksterhuis, 2004]. In the section, we will say that a translation of X_1 can be called $X_1 - 1a_1'$, and a translation of X_2 is $X_2 - 1a_2'$. The details for this translation usually depend on the fit criterion used and on the Procrustean model. If we want to translate X_1 and X_2, we get the following:

$$min_T ||X_1 T - a' - X_2||_F \tag{28}$$

where $a' = a_1' T - a_2'$. Translation can be represented with a single variable a as the only operational significance occurs when an image is move into a relative position from the origin. Due to this, we can just minimize Equation 28 over T and a. For Equation 28, a is considered a P_2 vector [Gower and Dijksterhuis, 2004]. This means that the term $1a'$ is a translation of the rows of $X_1 T - X_2$. The term $||A - 1a'||$ is minimized when a' is the mean of all the columns of A, which is $1'A/N$ [Gower and Dijksterhuis, 2004]. A great way to put this translation on $X_1 T - X_2$ is to remove the column means from both X_1 and X_2 apart. This process is known as centering, which involves evaluating:

$$A - 1a' = A - 11'A/G = A(1 - 11'/G) = GA \tag{29}$$

In the expression above, G is a centering matrix. This means that the results of this matrix multiplication is the same whenever we center X or XT, or $(GX)T = G(XT)$ [Gower and Dijksterhuis, 2004]. In most cases, as it is usually easier to do, X is centered.

For equation 27, it may be necessary to for X_1 and X_2 to have two different translations. In this case, we can edit Equation 27 to be:

$$trace(X_2 - 1a_2)'(X_1 T - a_1') \tag{30}$$

We will now try and show that similar to above, we will want to show translations to either variable can be taken out through centering. A quick way of doing this would be selecting specific values for X_1 and

X_2 that are both very large and negative [Gower and Dijksterhuis, 2004]. For a more formal method, we start by expanding Equation 26 with the equations above to get:

$$r_V^2 = \frac{(trace(X_2 - 1a_2')'(X_1T - a_1'))^2}{trace((X_1T_1 - 1a')'(X_1T_1 - 1a'))trace((X_2 - 1a_2')'(X_2 - 1a_2'))} \tag{31}$$

for easier calculations, we will be setting up three variables, A, B, and C, such that:

$$r_V^2 = \frac{C^2}{AB} \tag{32}$$

If we differentiate concerning a_1, we get the following equation:

$$2ABC\frac{\partial C}{\partial a_1} = C^2B\frac{\partial A}{\partial a_1} \tag{33}$$

After simplification, we get:

$$A(1'X_2 - Na_2') = -C(1'X_1T - Na_2') \tag{34}$$

Now, we differentiate concerning a_2. When we do this we get:

$$C(1'X_2 - Na_2') = -B(1'X_1T - Na_2') \tag{35}$$

Now, we shall combine both of the equations to get:

$$C^2 = AB \tag{36}$$

This implies that $r_V^2 = 1$, or $1'X_2 - Na_2' = 1'X_1T - Na_2' = 0$ [Gower and Dijksterhuis, 2004]. Through this process, we are left with two solutions. The first result is that $X_1T - 1a_1'$ is proportional to $X_2 - 1a_2'$. However, this is not true all of the time and is very rare, so we will ignore this. The second solution sets $X_1T - 1a_1' = NX_1T$ and $X_2 - 1a_2' = NX_2$ [Gower and Dijksterhuis, 2004]. This leads to both a_1 and a_2, the translation terms, are eliminated, which means that translations do not affect this process.

B.1 Orthogonal Procrustes Problems

For the upcoming discussion, we will be constraining T to be a square orthogonal matrix. As a reminder, a matrix A is considered to be orthogonal if the following condition is fulfilled:

$$A^T = A^{-1} \tag{37}$$

We are focusing on Procrustes problems where T is orthogonal, as it represents a rotation of an image, an act that is important for prepossessing images in our data, as many images were flipped in different directions. Orthogonal Procrustes analysis finds the rotation that gives the biggest average configuration among all rotations. As the norm in Procrustes problems, we aim to minimize equation 22 [Gower and Dijksterhuis, 2004]. As T is now orthogonal, we get the following equation:

$$||X_1T - X_2||_F = trace(X_1'X_1 + X_2'X_2) - 2 * trace(X_2'X_1T) \tag{38}$$

Notice above that the first term on the right does not depend on what T is. This means that as we change what T is, that term will not change, meaning we will treat that as a constant. Thus, minimizing the equation will involve maximizing $trace(X_2'X_1T)$ [Gower and Dijksterhuis, 2004]. This way, we can make sure that Equation 26, the inner product criteria, is the same as 22, the least squared criteria [Gower and Dijksterhuis, 2004].

Recount that any orthogonal transformation does not change the size of the matrix. Due to this two things are true. First, the interpretation of the correlation of the inner-product criterion is valid. In addition, the following statement is true:

$$||X_1T||_F = ||X_1||_F \tag{39}$$

Now let's consider the following expression $||X_1T_1 - X_2T_2||$, where T_1 and T_2 are different orthogonal matrices. Minimizing $||X_1T_1 - X_2T_2||$ is identical to minimizing Equation 22, if $T = T_1T_2'$ [Gower and Dijksterhuis, 2004]. This implies that rotating X_1 by T_1 and X_2 by T_2 is the same as rotating X_1 by T_1T_2', which implies two-sided Procrustes problems can be written as one-sided orthogonal Procrustes problems [Gower and Dijksterhuis, 2004].

Now let's find the solutions to these types of problems, which means finding the values T such that

Equation 38 is minimized. First, let's make the following substitution :

$$X_2' X_1 = U \Sigma V'. \tag{40}$$

This is called the single value composition of $X_2' X_1$ and will be very useful in finding the solution. Using this expression, we can simplify the trace to be:

$$trace(X_2' X_1 T) = trace(U \Sigma V' T) \tag{41}$$

Now, we will rearrange the matrices to be:

$$trace(\Sigma V' T) = trace(\Sigma V' T U) \tag{42}$$

Now we shall make another substitution , $H = V' T U$ to get the following:

$$trace(\Sigma V' T U) = trace(\Sigma H) \tag{43}$$

V', T, and U are all considered orthogonal matrices. Thus, H by definition is a product of orthogonal matrices, which then means:

$$\text{trace}(\Sigma H) = \sum_{i=1}^{P} h_{ii} \sigma_i \tag{44}$$

As σ_j must all be greater than 0, the sum is at its max when $h_i i = 1$. Due to this, the trace is at its max when $H = I$ [Gower and Dijksterhuis, 2004]. When putting this into the equation above, we get:

$$V' T U = I \tag{45}$$

Finally, after some rearranging, we get:

$$T = V' U \tag{46}$$

Thus, we say that $T = V' U$ is the solution of Orthogonal Procrustes problems.

Before we move on, we will discuss a special case, where $X_1 = I$. In this case, the orthogonal matrix T will be accurate if it is X_2 [Gower and Dijksterhuis, 2004]. In addition, V' and U are both singular vectors of X_2'.

C Back Propagation

Each neural network has a corresponding cost function. The derivative of the function due to the weight of a node is a crucial part of the machine-learning process. This specific derivative shows us the speed at which the specific cost of our network changes as the weight changes. However calculating this derivative is very computationally heavy if we use the multi-variable chain rule, as the number of terms that would need to be calculated in a chain rule is exponential to the number of layers in the neural networks. So, calculating this derivative with a neural network with over 1000 layers becomes impossible. Thus, we need an algorithm to calculate this for us, which propagation gives us.

Backpropagation is an algorithm to obtain the partial derivatives in terms of weight w and bias b of the cost function C in the network, both $\frac{\partial C}{\partial b}$ and $\frac{\partial C}{\partial w}$ [Nielsen, 2019]. There are many different cost functions, but for our purposes, we will be focusing on the quadratic cost function. The quadratic cost function is the following:

$$C = \frac{1}{2n} \sum_x ||y(x) - a^L(x)||^2 \tag{47}$$

In this example, n is the number of training examples, the sum is over each training example, denoted as x; $y(x)$ is the desired output for the specific training example x, L is the number of layers in the network, and a^L is the activations output vector from the network [Li, 2023]. For backpropagation to work, two assumptions need to be made about the cost function. The first assumption is that the cost function is the average $C = \frac{1}{n} \sum_x C_x$ over cost functions C_x for x, or individual training examples. We need this assumption as backpropagation lets us compute both partial derivatives, $\frac{\partial C}{\partial b}$ and $\frac{\partial C}{\partial w}$, for a single training example [Nielsen, 2019]. Only after do we recover $\frac{\partial C}{\partial b}$ and $\frac{\partial C}{\partial w}$ by averaging over the training examples. We can suppose the training examples are fixed, and write C_x as C. The second assumption, which is more of a fact, is that the cost function can be written as a function of the neural network's output [Nielsen, 2019].

Backpropagation is about understanding how changing certain weights of connections and biases of neurons affects the cost function [Nielsen, 2019]. This leads to computing the partial derivatives $\frac{\partial C}{\partial b^l_j}$ and $\frac{\partial C}{\partial w^l_{jk}}$. To do that, we must first introduce δ^l_j, the error in the j^{th} neuron in the l^{th} layer [Li, 2023]. Backpropagation gives a process to compute this error and then relates it to both partial derivatives. The equation for the error is defined as:

$$\delta_j^l \equiv \frac{\partial C}{\partial z_j^l} \tag{48}$$

Backpropagation revolves around four equations, which allow us to compute both the error and the gradient of the cost function. The first equation is one for the error in the output layer, δ^L. The component of this error is given by the following:

$$\delta^L = \frac{\partial C}{\partial a_j^L}\sigma'(z_j^L) \tag{49}$$

In this equation, $\frac{\partial C}{\partial a_j^L}$ measures how fast the cost is changing as a function of the j^{th} output activation [Li, 2023]. For example, if C only lightly depends on an output neuron, j, then we know δ_j^L will be small. $\sigma'(z_j^L)$ measures the speed at which the activation function σ is changing at z_j^L [Nielsen, 2019]. Equation 6 is considered easily computable, as z_j^L can be computated while computing the behavior of the network [Li, 2023]. In addition, computing $\sigma'(z_j^L)$ requires little overhead. This is due to $\frac{\partial C}{\partial a_j^L}$ being used to compute many cost functions. This equation is a componentwise expression for δ^L. To get it into a matrix-based form, we rewrite it as the following:

$$\delta^L = \Delta_a C \odot \sigma'(z^L) \tag{50}$$

In the equation above, $\Delta_a C$ is a vector whose components are the partial derivatives $\frac{\partial C}{\partial a_j^L}$ [Nielsen, 2019]. It could be thought of as the rate of change of C with respect to the output activations [Nielsen, 2019]. $\odot$ is the symbol for the Hadamard product. In particular, if we have two vectors with the same dimensions, s and t. Then $(s \odot t)_j = s_j t_j$. Here's a quick example of a Hadamard product.

$$\begin{pmatrix} 1 \\ 2 \end{pmatrix} \odot \begin{pmatrix} 4 \\ 3 \end{pmatrix} = \begin{pmatrix} 4 \\ 6 \end{pmatrix}$$

Derivation: To begin, remember the definition of δ_j^l which is present is Equation 59. By using the chain rule, we can re-express the partial derivative in terms of partial derivatives with respect to the output activations:

$$\delta_j^L = \sum_k \frac{\partial C}{\partial a_k^L}\frac{\partial a_k^L}{\partial z_j^L} \tag{51}$$

Above, the sum is over all neurons k in the output layer. We know that the output activation a_k^L depends on the weighted input z_j^L for the j^{th} neuron, when k = j [Li, 2023]. In addition, when $k \neq j$, $\frac{\partial a_k^L}{\partial z_j^L}$ will become 0. This means we can simplify the following:

$$\delta_j^L = \frac{\partial C}{\partial a_j^L} \frac{\partial a_j^L}{\partial z_j^L} \tag{52}$$

Lastly, as $a_j^L = \sigma(z_j^L)$, $\frac{\partial a_k^L}{\partial z_j^L} = \sigma'(z_j^L)$ [Li, 2023]. The final equation then becomes:

$$\delta_j^L = \frac{\partial C}{\partial a_j^L} \sigma'(z_j^L) \tag{53}$$

This is equation 49 in component form, so the derivation is finished.

The second equation of backpropagation is the equation for δ^l, the error, in terms of δ^{l+1}, the error on the next layer [Nielsen, 2019]. The equation is the following:

$$\delta^l = ((w^{l+1})^T \delta^{l+1}) \odot \sigma'(z^l) \tag{54}$$

For reference, $(w^{l+1})^T$ is the transpose of the weight matrix for the $(l+1)^{\text{th}}$ layer [Nielsen, 2019].

Derivation: To do this derivation, we need to get an equation for δ^l in terms of the error in the next layer, δ^{l+1}. To start, we will rewrite $\delta_j^l = \frac{\partial C}{\partial z_j^l}$ in terms of $\delta_k^{l+1} = \frac{\partial C}{\partial z_k^{l+1}}$ [Li, 2023]. Using the chain rule, we get the following:

$$\delta_j^l = \frac{\partial C}{\partial z_j^l} \tag{55}$$

$$= \sum_k \frac{\partial C}{\partial z_k^{l+1}} \frac{\partial z_k^{l+1}}{\partial z_j^l} \tag{56}$$

Now substituting the definition of δ_k^{l+1}, we get:

$$= \sum_k \frac{\partial z_k^{l+1}}{\partial z_j^1} \delta_k^{l+1} \tag{57}$$

For the next step, we must know the following:

$$z_k^{l+1} = \sum_j w_{kj}^{l+1} a_j^l + b_k^{l+1} = \sum_j w_{kj}^{l+1} \sigma(z_j^l) + b_k^{l+1} \tag{58}$$

If we differentiate the following, we get:

$$\frac{\partial z_k^{l+1}}{\partial z_j^1} = w_{jk}^{l+1} \sigma'(z_j^l) \tag{59}$$

Substituting this back into our original equation, we get:

$$\delta_j^l = \sum_k w_{jk}^{l+1} \sigma'(z_j^l) \delta_k^{l+1} \tag{60}$$

This is just the component form of the original equations, equation 54, thus the derivation is complete [Li, 2023].

If we combine both the first two equations of Backpropagation, we can then compute δ^l for every layer in a network [Nielsen, 2019]. To do this, we begin by using Equation 6 in order to calculate δ^L for a specific layer of the network. Then, we use Equation 12 to compute δ^{L-1}, δ^{L-2}, and so on until all the errors for the layers all computed [Nielsen, 2019].

The third equation of backpropagation is for the rate of change of the cost with respect to any bias in the network, which is the following:

$$\frac{\partial C}{\partial b_j^l} = \delta_j^l \tag{61}$$

This equation, in simple terms, shows that the error is equal to the rate of change of the cost function [Li, 2023]. As the first two equations of backpropagation can find the error on any layer, this third equation gives us the derivative of the cost function on any layer.

Derivation:

To begin the derivation, first remember the following:

$$z_j^l = \sum_k w_{jk}^l a_k^{l-1} + b_j^l \tag{62}$$

If we take the derivative of z_j^l in terms of the bias, or b_J^l, we get:

$$\frac{\partial z_j^l}{\partial b_j^l} = 1 \tag{63}$$

Next,we will multiply each side by $\frac{\partial z_j^l}{\partial b_j^l}$, we will get the following equation:

$$\delta_j^l = \frac{\partial C}{\partial z_j^l} \frac{\partial z_j^l}{\partial b_j^l} = \frac{\partial C}{\partial b_j^l} \tag{64}$$

This is the original equation thus the derivation is complete [Li, 2023].

The final equation is the rate of change of the cost with respect to any weight in the network, which is the following derivative: [Li, 2023].

$$\frac{\partial C}{\partial w_{jk}^l} = a_k^{l-1} \delta_j^l \tag{65}$$

The equation is very useful as we can calculate both a^{l-1} and δ_j^l with the other three equations we have proved.

Derivation:

This is very similar to the derivation for Equation 3. We begin with the definition of z_j^l:

$$z_j^l = \sum_k w_{jk}^l a_k^{l-1} + b_j^l \tag{66}$$

Now, we will take the derivative in terms of w_{jk}^l

$$\frac{\partial z_j^l}{\partial w_{jk}^l} = a_k^{l-1} \tag{67}$$

Now we multiply each side by the error formula and get the following:

$$\delta_j^l a_k^{l-1} = \frac{\partial C}{\partial z_j^l} \frac{\partial z_j^l}{\partial w_{jk}^l} = \frac{\partial C}{\partial w_{jk}^l} \tag{68}$$

This is the original equation, so the derivation is complete [Li, 2023].

Here are some final comments about these equations. First, let's look at the function $\sigma(z_j^l)$. A popular function $\sigma(z_j^l)$ can be is called the sigmoid functions. The sigmoid function has a very important property, that is z is approximately 0 or 1 [Nielsen, 2019]. This implies that the derivative of the sigmoid function, $\sigma(z_j^l)$, is approximately 0 [Nielsen, 2019]. Thus, the weight of the final layer will slowly learn as long as the output neuron is either low activation, meaning approximately 0, or high activation, meaning approximately 1. In this case, we can say the output neuron has become saturated, causing the weight to stop learning, or the learn very slowly. In addition, this also holds for the biases for the output neuron [Nielsen, 2019].

We gain something similar in Equation 2. The $\sigma'(z^l)$ implies that the error or δ_j^l will become small if the neuron is saturated, which means that any input to that neuron will learn slowly [Nielsen, 2019].

One last comment before moving on. All of the equations for backpropagation hold for any activation function, not just the sigmoid function above. This means that these equations can be used to design activation functions that have particular desired learning properties. For example, let's make the activation function non-sigmoid such that its derivative is always positive and never approaches 0 [Nielsen, 2019]. This would prevent learning to not slow down, which would usually occur using a sigmoid function.

Now let us go into the backpropagation algorithm. The backpropagation equations will all be used in order to compute the gradient of the cost function. Here is the full algorithm: [Nielsen, 2019].

Input: Set a^1, the activation for the input layer.

Feedforward: For every layer $l = 1, 2, 3, ..., L$, compute $z^l = w^l a^{l-1} + b^l$ and $a^l = \sigma(z^l)$.

Output error: Compute the vector $\delta^L = \Delta_a C \odot \sigma'(z^L)$

Backpropagate the error: For every layer $l = L-1, L-2, ..., 2$, compute $\delta^l = ((w^{l+1})^T \delta^{l+1} + \odot \sigma'(z^l)$

Output: The gradient of the cost function.

Now let's analyze this algorithm. This system computes all the error vectors δ^l backward, where we end at the final layer. This movement backward is due to the fact that the cost is a function of outputs from the network. This algorithm is considered fast. However, to understand this, we must go back [Nielsen, 2019]. Initially, in the 1950s and 1960s, when neural network research began, researchers came up with another way to compute the gradient of the cost function [Nielsen, 2019]. It involved regarding the cost as a function of the weights $C = C(w)$ alone, without the biases. The weights were numbers $w_1, w_2, ...$, and to compute $\frac{\partial C}{\partial w_j}$ for any w_j, the following approximation was made:

$$\frac{\partial C}{\partial w_j} \approx \frac{C(w + \epsilon e_j)}{\epsilon} \tag{69}$$

In this approximation, $\epsilon > 0$ is small and e_j is the unit vector in the j^{th} dimension. Using this method, we can estimate $\frac{\partial C}{\partial w_j}$ by just computing C, the cost function, for two w_j, and then apply the equation above [Nielsen, 2019]. This similar idea will let you compute $\frac{\partial C}{\partial b}$ This approach had a few positives. First off, it is easier to understand conceptually, as it avoids much of the complicated math listed above. In addition, it is very easy to code, as it only takes a few lines to implement [Nielsen, 2019]. However, this approach is very slow, as for each distinct w_j we need to compute $C(w+\epsilon e_j)$ to get the specific $\frac{\partial C}{\partial w_j}$ This means if we had a network with ten million weights, we would need to compute the cost function ten million times to get the gradient, requiring ten million forward passes through the network [Nielsen, 2019]. Backpropagation allows us to compute all partial derivatives $\frac{\partial C}{\partial w_j}$ at once, with only two passes through the network, one forward and one backward. As the computational cost of a backward pass is approximately the same as a forward pass, the total cost of backpropagation is two forward passes, much less than the ten million needed above [Nielsen, 2019].

D CNN Encoder/Decoder to Dewarp Images

Written by Elliot Trilling

D.1 Introduction

One of the early problems we faced in this project was figuring out how to dewarp images that were not scanned straight. That is, figuring out how to transform images so that general quadrilaterals would be converted back into rectangles. See 2.1.8 for more details. We came up with an effective procedure using image processing techniques using OpenCV as detailed in 3.4. However, I wanted to explore if this procedure could be done with a neural network.

One common use for autoencoder neural networks is to remove noise from images. This can be done by training a network on a dataset where the target output is a clean image and the input is that clean image with some noise added. This way the network learns how to transform images with noise into images without noise. Below is a simple example of an autoencoder removing noise from MNIST images 43. It seemed possible that a similar encoder/decoder model might be able to dewarp images.

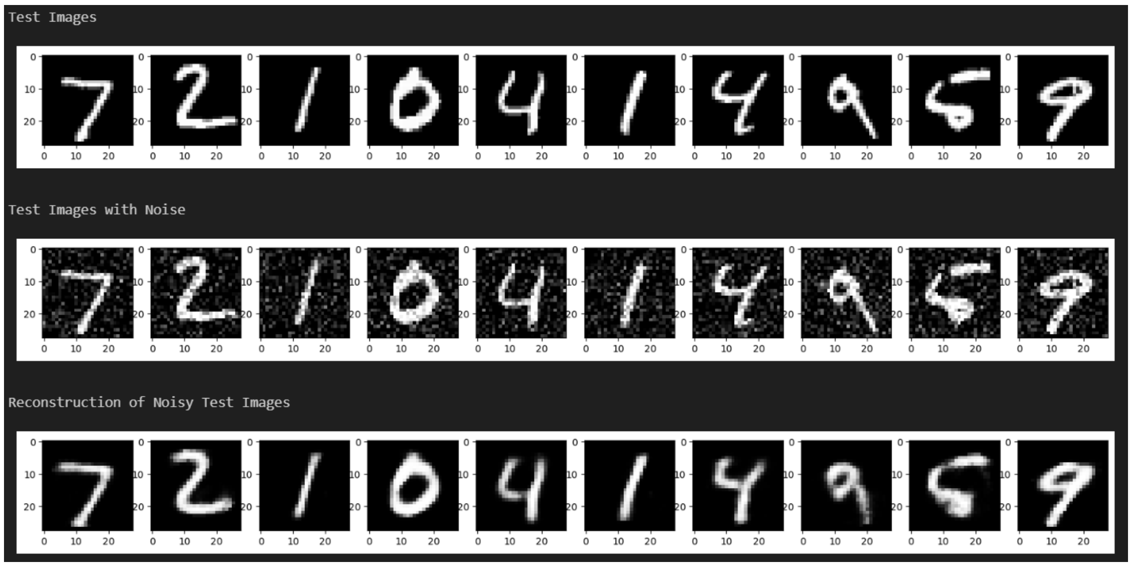


Figure 43: Denoising Autoencoder Example on MNIST Dataset. The first row contains input images, the second row contains the input images with added noise, and the third row contains the model's attempt to reconstruct the original input image from the noisy image.

D.2 Everything at Once

I began by generating some synthetic data that roughly resembled the spreadsheets we worked with through this project. I then applied random warps using OpenCV's "getPerspectiveTransform" and "warpPerspective" to produce a dataset of warped input images and straight output images 44.

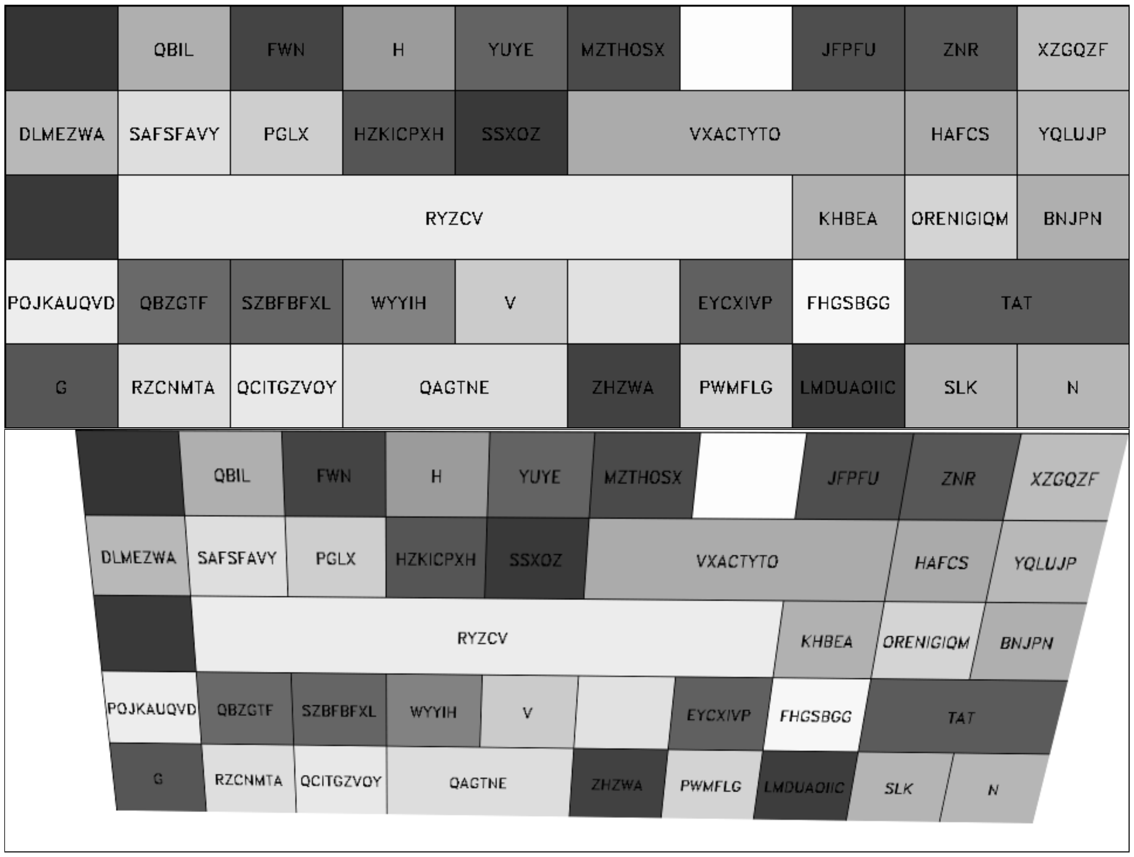


Figure 44: An image of a randomly generated synthetic table before and after being randomly warped. The top image is the original table. The bottom is the warped table.

I trained a simple CNN (Convolutional Neural Network) based encoder/decoder model on this synthetic dataset. The results of this first test were not very promising. Figure 45 depicts an example result.

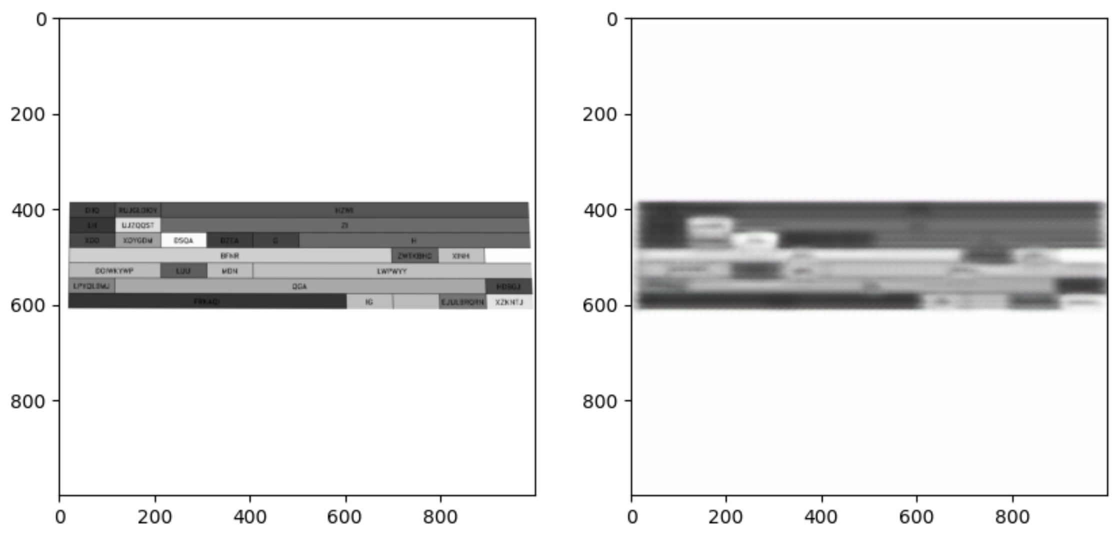


Figure 45: A warped input image (left) and the model output image (right).

Additionally, the model I came up with required a huge amount of memory (36GB) to process even medium sized images (1000x1000). It became clear that a more systematic approach with smaller steps would likely yield better results. In particular, I decided to try using smaller images, simpler transformations, and spending more time figuring out the details of my selected network architecture.

D.3 Simple Toy Problem

Given the difficulties associated with my first attempt, I decided to attempt to solve a simpler toy problem. In particular, building an encoder/decoder model that could de-rotate small toy images. I generated a dataset of binary images with randomly spaced horizontal and vertical lines. I thought these images would provide enough detail to evaluate if my model was working but be simple enough to evaluate at a glance. I generated four separate datasets of different sized images (28x28, 56x56, 84x84, and 112x112) 46.

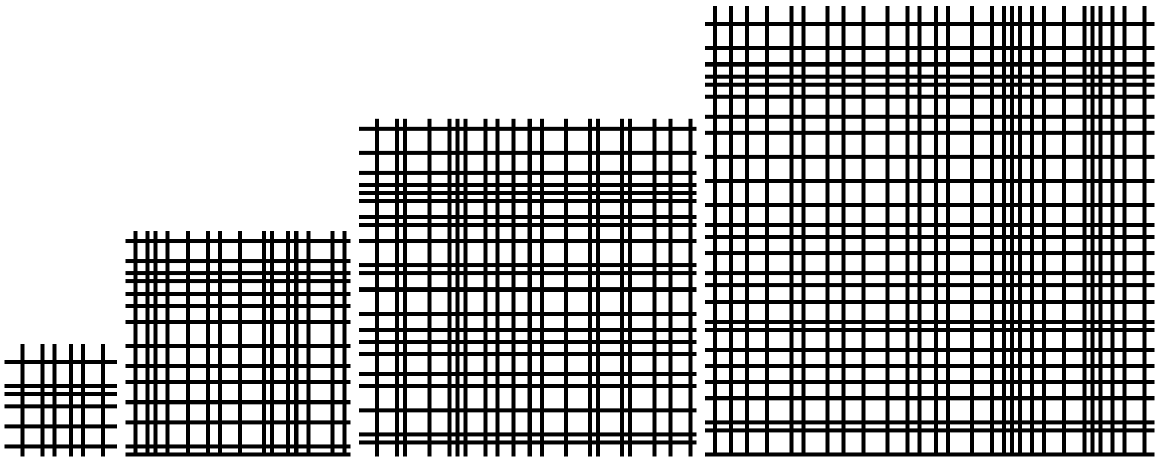
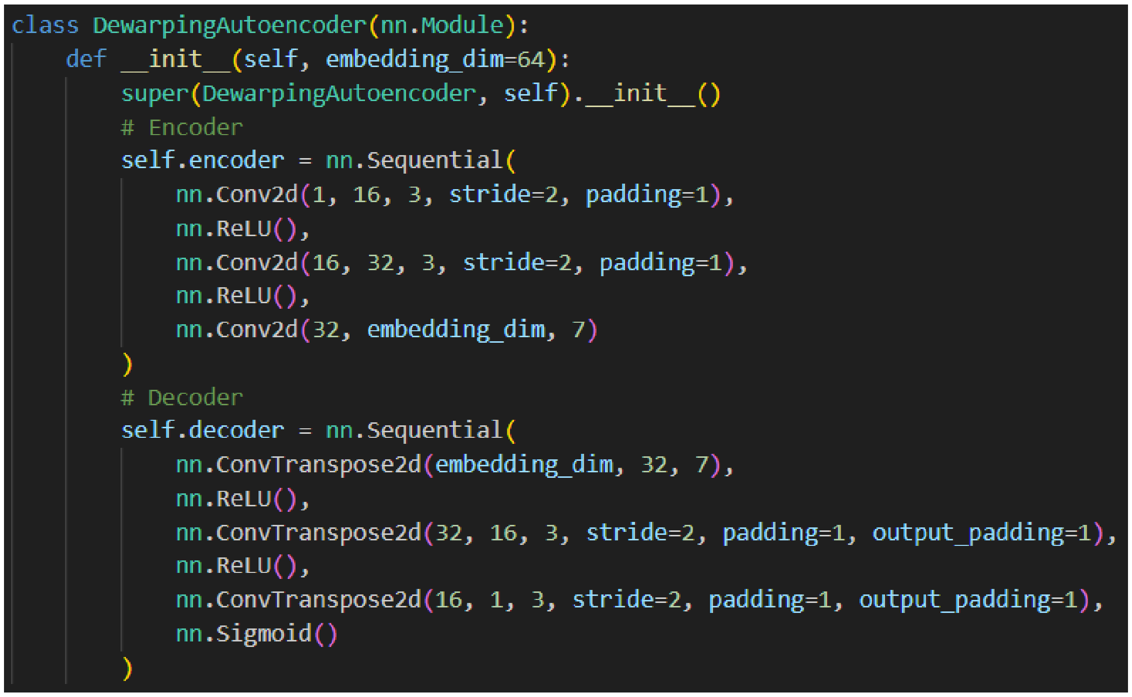


Figure 46: Images of random grids of different sizes (28x28, 56x56, 84x84, 112x112).

I wrote a simple CNN encoder/decoder to start experimenting with my datasets. The initial network architecture can be seen described using PyTroch below 47.



```
class DewarpingAutoencoder(nn.Module):
    def __init__(self, embedding_dim=64):
        super(DewarpingAutoencoder, self).__init__()
        # Encoder
        self.encoder = nn.Sequential(
            nn.Conv2d(1, 16, 3, stride=2, padding=1),
            nn.ReLU(),
            nn.Conv2d(16, 32, 3, stride=2, padding=1),
            nn.ReLU(),
            nn.Conv2d(32, embedding_dim, 7)
        )
        # Decoder
        self.decoder = nn.Sequential(
            nn.ConvTranspose2d(embedding_dim, 32, 7),
            nn.ReLU(),
            nn.ConvTranspose2d(32, 16, 3, stride=2, padding=1, output_padding=1),
            nn.ReLU(),
            nn.ConvTranspose2d(16, 1, 3, stride=2, padding=1, output_padding=1),
            nn.Sigmoid()
        )
```

Figure 47: Initial encoder/decoder architecture written in PyTorch.

While the exact details of this architecture are not too important, one salient detail is the shape of the image embeddings when the model is trained on datasets of images of different sizes. When trained on images with size 28x28, the image embedding has a shape of [64, 1, 1]. That is, the images get embedded as 1x1 images with 64 channels of data in each pixel. When trained on images with size 56x56, the image embedding has a shape of [64, 8, 8]. That is, the images get embedded as 8x8 images with 64 channels of data in each pixel. 84x84 results in a shape of [64, 15, 15], and 112x112 in a shape of [64, 22, 22]. The relevance of these different sizes will be referenced below.

The results of this initial architecture were mixed. It worked very well on small and medium sized images (28x28 and 56x56) but not on larger images (84x84 or 112x112) 48. In particular, while it managed to successfully de-rotate the edges of larger images it struggled to de-rotate the center.

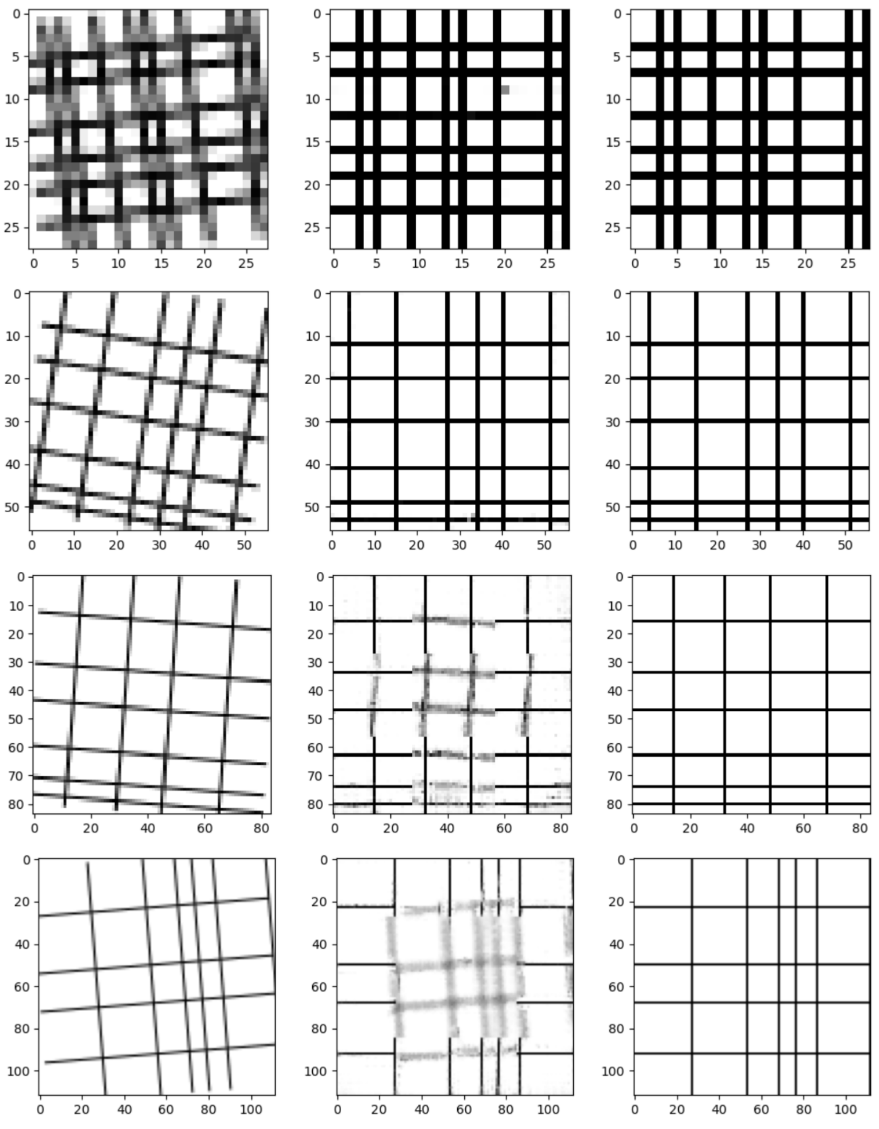


Figure 48: Examples produced by the original model when trained and tested on datasets of different sized images. From top to bottom, datasets of images of size 28x28, 56x56, 84x84, 112x112. From left to right, the rotated input image, the model prediction, and the true target image.

After much testing I was able to pinpoint the problem occurring in larger images. Because of the network architecture I had selected, pixels in the output image didn't always have access to information about all pixels in the input image. This meant the network sometimes didn't have enough information to correctly de-rotate the larger images.

The CNN encoder/decoder architecture I used had two types of layers (not counting layer activations). It uses nn.Conv2d layers in the encoder to reduce the size of the image and nn.ConvTranspose2d layers in the decoder to produce an output image from the encoded image. One key feature of both of these layers is that they only process information locally. That is, they only act on local groups of pixels. This is depicted in 49. Let us consider the model that was trained and tested on images of shape 112x112. After passing through the encoder, the original input image, which had a shape of 1x112x112, would have an output shape of 64x22x22. *Note that the "1x" in the input images just means it just had one data channel (thus a grayscale image).*

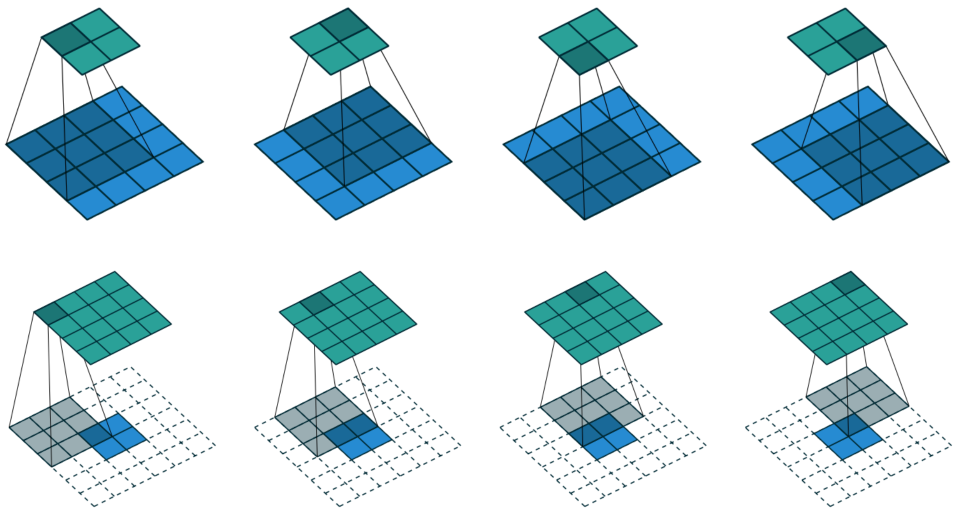


Figure 49: A convolution (left) and transposed convolution (right) [Dumoulin and Visin, 2018].

A feature/problem of the encoded image is that it still contains some amount of pixel position data. For example the top left pixel in the encoded image (which in this case has 64 channels of data) would only contain information from a group of pixels in the top left of the input image. More generally each pixel in the embedded image is a function of a group of adjacent pixels in the input image. A diagram of this idea in 1D can be see below 50. Mathematically each embedded pixel, (x_{em}, y_{em}), could be described by $(x_{em}, y_{em}) = g(\{(x_{input}, y_{input})| \|(x_0, y_0) - (x_{input}, y_{input})\| < c\})$ for some function g, some central input pixel (x_0, y_0), and some distance c.

After passing through the encoder, the decoder uses transposed convolutions on the encoded image

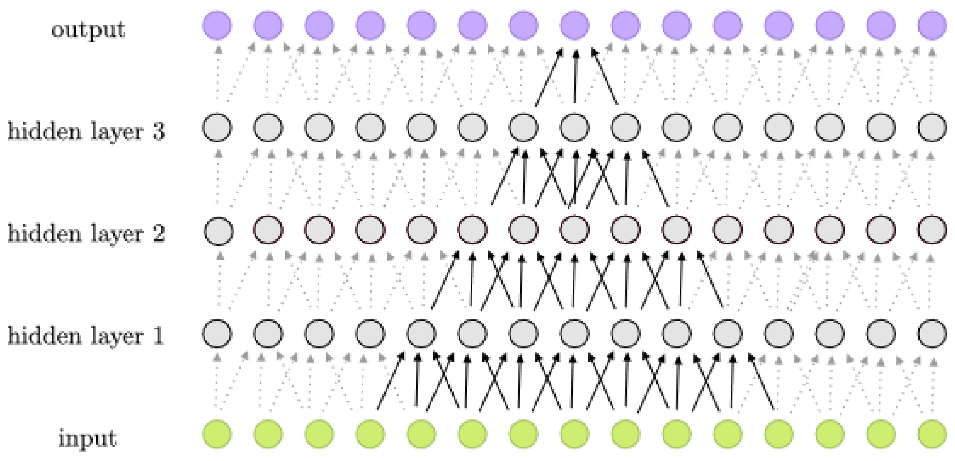


Figure 50: An 1D CNN feed forward example showing how embedding pixels are a function of a local group of pixels [M, 2020].

to produce an output image of shape 1x112x112. In a similar manner to the encoder, the top left pixel in the output image is constructed from a cluster of upper left pixels in the encoded image. As before, a general pixel in the output image is some function of a group of local pixels in the embedded image. Mathematically each output pixel, (x_{out}, y_{out}), could be described by $(x_{out}, y_{out}) = h(\{(x_{em}, y_{em}) | \|(x_0, y_0) - (x_{em}, y_{em})\| < d\})$ for some function h, some central embedded pixel (x_0, y_0), and some distance d.

The combined effect of the encoder and decoder preserving this sort of structure is that a pixel in the output image is a function of a group of local pixels in the input image. This can be seen visually in figure 51. Mathematically each output pixel, (x_{out}, y_{out}), could be described by $(x_{out}, y_{out}) = f(\{(x_{input}, y_{input}) | \|(x_{out}, y_{out}) - (x_{input}, y_{input})\| < b\})$ for some function f and some distance b.

The problem is that the middle pixels in the output image don't know anything about what is happening at the edges of the input image. It seems like this does not provide the model with a sufficient amount of data to de-rotate the image.

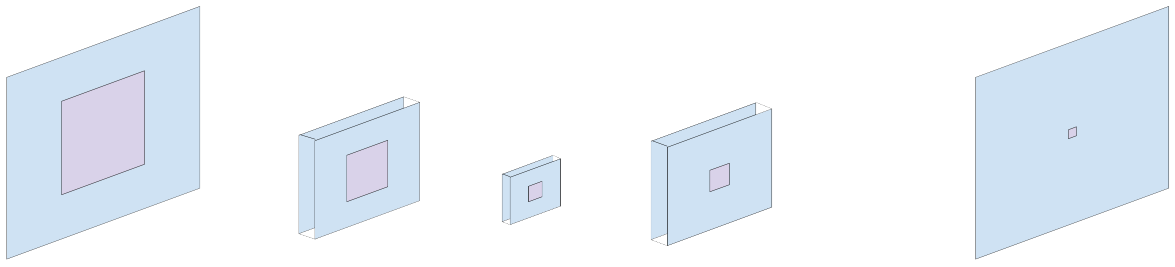


Figure 51: A diagram showing how one pixel in the output image (far right) is a function of a group of local pixels on the input image (far left).

The reason this didn't happen with the 1x28x28 images is because they get encoded into 64x1x1 images. That is, the complete image gets encoded in a single pixel (with 64 channels worth of detail). Each pixel of the output image is a function of this one encoded pixel which in tern is a function of all of the pixels in the input image. Thus, each pixel in the output image is a function of each pixel in the input image. This can be seen in figure 52.

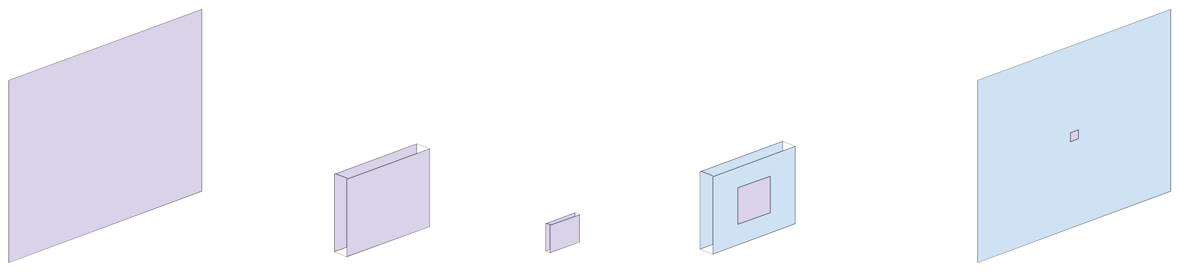


Figure 52: A diagram showing how one pixel in the output image (far right) is a function of a group of local pixels on the input image (far left). Because the image is small, each output pixel is a function of each input pixel.

In a similar manner, 1x56x56 images get encoded into 64x8x8 images. Because the first layer of the decoder uses large enough kernels (7x7), we still get an effect where each pixel in the output image is a function of each pixel in the input image. Results only begin to break down with 1x84x84 images which get encoded as 64x15x15 images.

There are multiple possible ways to solve this problem. One way is to add a fully connected layer after the last layer of the encoder. One fully connected layer allows information from each part of the encoded image to be shared with any other part of the encoded image. This works well, but in my tests it seems to be somewhat computationally expensive (adding many millions of parameters to the network). Another way is to simply reduce the height and width of the embedded image so that the kernel in the first layer of the decoder can capture information from from each part of the embedded image. By using dilated convolutions 53 with large strides I reduced the embedding size of 1x112x112 images down to 500x4x4. Here, 500 channels are used instead of 64 so as to not introduce a data bottleneck that lowers output quality. Because this image only has a width and height of 4 (with 500 channels per pixel), the first layer of the decoder is able to use information from each pixel of the input image.

The results of the updated autoencoder with dilations can be seen here 54.

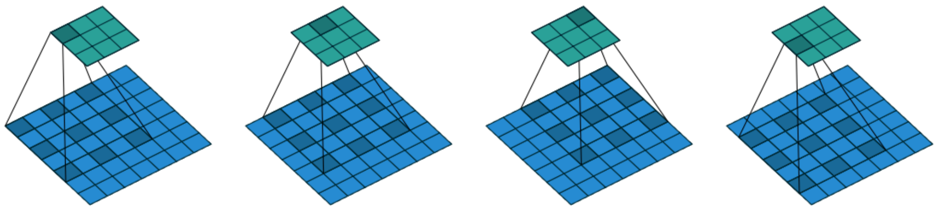


Figure 53: A dilated convolution [Dumoulin and Visin, 2018].

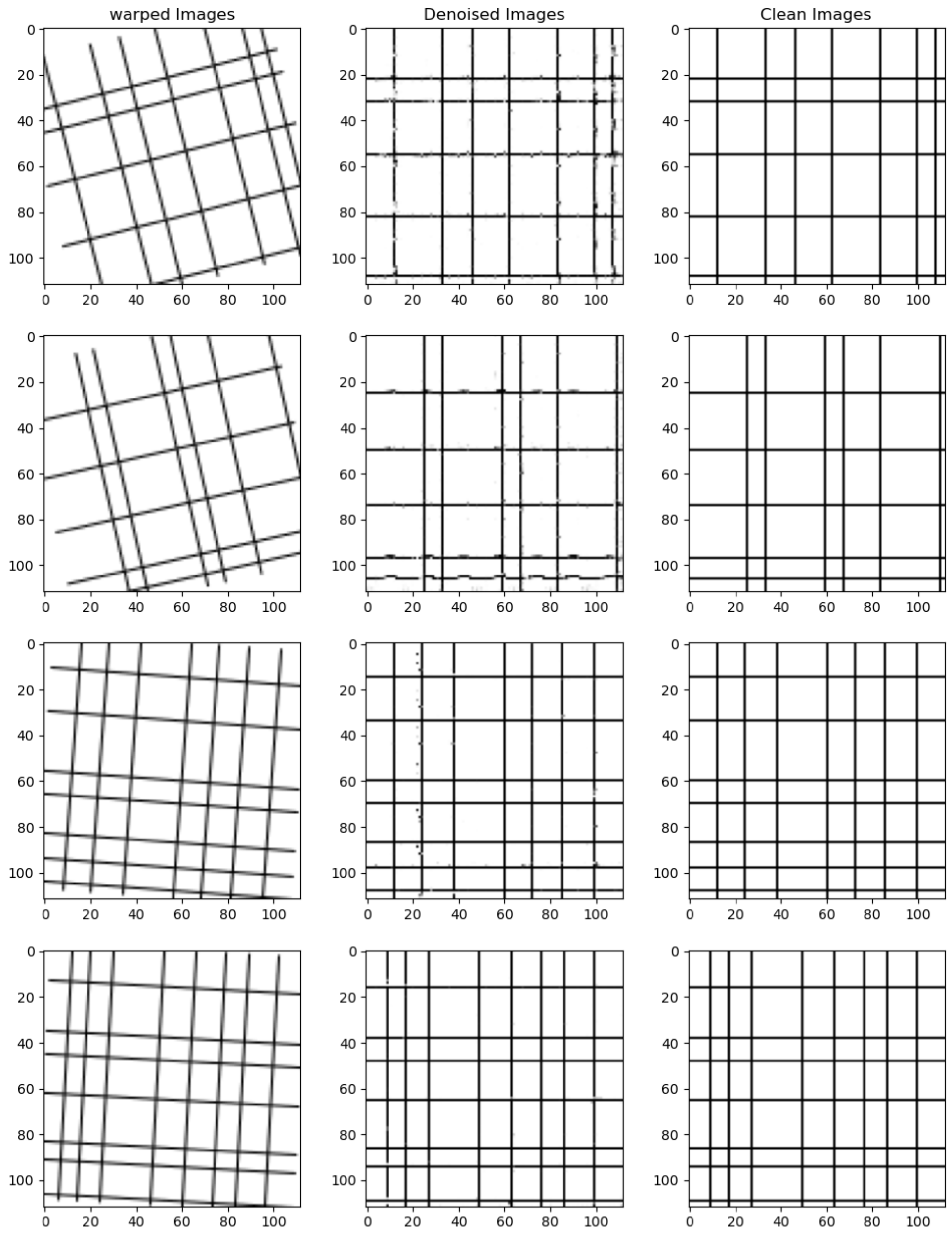


Figure 54: Results of the new encoder/decoder using dilations. Each row is a separate example. The first column is the input image, the second column is the model output, the third column is the target image.

While even these simple toy results are not perfect, they are good enough that the idea of using encoder/decoder models to dewarp images may be an interesting topic to pursue further.

D.4 Ideas for Future Testing

In completing the testing described in the previous section D.3, I located some possible areas for future experimentation. A first step would be to replace the toy images I produced with more realistic ones to see if these models could still accurately de-rotate them. If they could, a next task would be to replace random image rotations with more general random transformations under continuous $\mathbb{R}^2 \to \mathbb{R}^2$ functions. While this represents a huge class of transformations, a much more manageable subset to deal with would be functions of the form $f(x, y) = \langle g(x, y), h(x, y)\rangle$ where $g(x, y) = a_{22}x^2y^2 + a_{21}x^2y + a_{12}xy^2 + a_{11}xy + a_{10}x + a_{01}y + a_{00}$ and $h(x, y) = b_{22}x^2y^2 + b_{21}x^2y + b_{12}xy^2 + b_{11}xy + b_{10}x + b_{01}y + b_{00}$. It seems likely that many cases of real world image warping could be well approximated by such a polynomial. Thus, a next reasonable step would be to see if a network could reconstruct images that were warped under a randomly selected polynomial of the type described. If image reconstruction proved too difficult, another interesting avenue to explore would be training a parameter estimation network instead of a full encoder/decoder. The goal of this network would simply be to estimate the amount an image was rotated or, for the polynomial just discussed, predict coefficients. If these parameters were known, it could be possible to reconstruct the output image using other algorithmic techniques.

References

[Apa, 2004] (2004). Apache license, version 2.0. https://www.apache.org/licenses/LICENSE-2.0. Accessed: 2024-02-13.

[Gra, 2024] (2024). Gray scaling with the algorithms. https://medium.com/@mjbharmal2002/gray-scaling-with-the-algorithms-b83f87975885. Accessed: 2024-02-14.

[NSC, 2024] (2024). Natick soldier systems center (nssc). https://installations.militaryonesource.mil/in-depth-overview/natick-soldier-systems-center-nssc. Accessed: 2024-02-14.

[Ope, 2024] (2024). Opencv : Image thresholding. https://docs.opencv.org/4.x/d7/d4d/tutorial_py_thresholding.html. Accessed: 2024-02-14.

[Mor, 2024] (2024). Opencv: Morphological transformations. https://www.educative.io/answers/what-is-image-blurring. Accessed: 2024-02-14.

[PyI, 2024] (2024). Pyinstaller: Freeze (package) python programs into stand-alone executables. https://github.com/pyinstaller/pyinstaller. Accessed: 2024-02-13.

[Aggarwal, 2014] Aggarwal, C. C., editor (2014). *Data Classification: Algorithms and Applications*. Chapman and Hall/CRC, 1st edition.

[Ba et al., 2016] Ba, J. L., Kiros, J. R., and Hinton, G. E. (2016). Layer normalization.

[Domke and Aloimonos, 2009] Domke, J. and Aloimonos, Y. (2009). Image transformations and blurring. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 31(5):1000–9999.

[Dosovitskiy et al., 2021] Dosovitskiy, A., Beyer, L., Kolesnikov, A., Weissenborn, D., Zhai, X., Unterthiner, T., Dehghani, M., Minderer, M., Heigold, G., Gelly, S., Uszkoreit, J., and Houlsby, N. (2021). An image is worth 16x16 words: Transformers for image recognition at scale.

[Dumoulin and Visin, 2018] Dumoulin, V. and Visin, F. (2018). A guide to convolution arithmetic for deep learning.

[Face, 2023] Face, H. (2023). Trainer class documentation. https://huggingface.co/docs/transformers/main_classes/trainer. Accessed: 2023-12.

[Face, 2024] Face, H. (2024). Trocr model documentation. https://huggingface.co/docs/transformers/en/model_doc/trocr. Accessed: 2023-12.

[Google, 2021] Google (2021). Vit-base-patch16-224. https://huggingface.co/google/vit-base-patch16-224. Accessed: 2023-12.

[Gower and Dijksterhuis, 2004] Gower, J. C. and Dijksterhuis, G. B. (2004). 1Introduction. In *Procrustes Problems*. Oxford University Press.

[Hummel et al., 1987] Hummel, R. A., Kimia, B., and Zucker, S. W. (1987). Deblurring gaussian blur. *Computer Vision, Graphics, and Image Processing*, 38(1):66–80.

[Jordon et al., 2022] Jordon, J., Szpruch, L., Houssiau, F., Bottarelli, M., Cherubin, G., Maple, C., Cohen, S. N., and Weller, A. (2022). Synthetic data – what, why and how?

[Lee, 2022] Lee, S. (2022). Understanding homography (a.k.a perspective transformation).

[Lehn et al., 2023] Lehn, K., Gotzes, M., and Klawonn, F. (2023). *Greyscale and Colour Representation*, pages 193–210. Springer International Publishing, Cham.

[Li, 2023] Li, H. (2023). Proofs for the four fundamental equations of the backpropagation and algorithms in feedforward neural networks. *Researchgate preprint*.

[Li et al., 2021] Li, M., Lv, T., Cui, L., Lu, Y., Florencio, D., Zhang, C., Li, Z., and Wei, F. (2021). Trocr: Transformer-based optical character recognition with pre-trained models.

[M, 2020] M, V. O. (2020). *English: A 3 layer 1D CNN feed-forward diagram with kernel size of 3 and stride of 1.*

[Marti and Bunke., 2002] Marti, U. and Bunke., H. (2002). The iam-database: An english sentence database for off-line handwriting recognition. *Int. Journal on Document Analysis and Recognition, Volume 5.*

[Microsoft, 2022] Microsoft (2022). Trocr-base-printed. https://huggingface.co/microsoft/trocr-base-printed. Accessed: 2023-12.

[Nielsen, 2019] Nielsen, M. (2019). *Neural Networks and Deep Learning.* Determination Press.

[OpenCV Developers, 2024a] OpenCV Developers (2024a). Color space conversions. https://docs.opencv.org/4.x/d8/d01/group__imgproc__color__conversions.html. Accessed: 2024-02-14.

[OpenCV Developers, 2024b] OpenCV Developers (2024b). Image filtering. https://docs.opencv.org/4.x/d4/d86/group__imgproc__filter.html. Accessed: 2024-02-14.

[OpenCV Developers, 2024c] OpenCV Developers (2024c). Miscellaneous image transformations. https://docs.opencv.org/4.x/d7/d1b/group__imgproc__misc.html. Accessed: 2024-02-14.

[OpenCV Developers, 2024d] OpenCV Developers (2024d). Opencv: Introduction. https://docs.opencv.org/4.x/d1/dfb/intro.html. Accessed: 2024-02-13.

[Pan, 2014] Pan, S. J. (2014). Transfer learning. *Data Classification: Algorithms and Applications*, 21:537–570.

[Pearson, 2023] Pearson, A. (2023). Ocr data. https://www.kaggle.com/datasets/aidapearson/ocr-data. Accessed: 2023-12.

[Pesce et al., 2023] Pesce, V., Hermosin, P., Rivolta, A., Bhaskaran, S., Silvestrini, S., and Colagrossi, A. (2023). Chapter nine - navigation. In Pesce, V., Colagrossi, A., and Silvestrini, S., editors, *Modern Spacecraft Guidance, Navigation, and Control*, pages 441–542. Elsevier.

[PyTorch, 2023] PyTorch (2023). Pytorch tutorials - learning pytorch with examples. https://pytorch.org/tutorials/beginner/basics/data_tutorial.html. Accessed: 2023-12.

[Raghunathan, 2021] Raghunathan, T. E. (2021). Synthetic data. *Annual Review of Statistics and Its Application*, 8(Volume 8, 2021):129–140.

[Singhal et al., 2017] Singhal, P., Verma, A., and Garg, A. (2017). A study in finding effectiveness of gaussian blur filter over bilateral filter in natural scenes for graph based image segmentation. In *2017 4th international conference on advanced computing and communication systems (ICACCS)*, pages 1–6. IEEE.

[Sreedhar and Panlal, 2012] Sreedhar, K. and Panlal, B. (2012). Enhancement of images using morphological transformation. *arXiv preprint arXiv:1203.2514.*

[Toews, 2023] Toews, R. (2023). Transformers revolutionized ai. what will replace them? *Forbes.*

[U.S. Army Combat Capabilities Development Command, 2023] U.S. Army Combat Capabilities Development Command (2023). Devcom home. https://www.army.mil/devcom. Accessed: 2023-09.

[Vaswani et al., 2017] Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., Kaiser, Ł., and Polosukhin, I. (2017). Attention is all you need. *Advances in neural information processing systems*, 30.

[Zhang, 2023] Zhang, M. (2023). Neural attention: Enhancing qkv calculation in self-attention mechanism with neural networks. https://arxiv.org/pdf/2310.11398.pdf. [Accessed 04-04-2024].