AFEKA אפקה המכללה האקדמית להנדסה בתל-אביב TEL-AVIV ACADEMIC COLLEGE OF ENGINEERING

The department of Software Engineering

Final Report

Project Name: Deep learning-based handwriting text editor, generator, and analyzer

Student Names: Daniel Ivkovich

Eylon Mizrahi

Supervisor Name: Dr. Eyal Katz.

Date: July 10, 2021

Supervisor's Approval

<Eyal Katz <eyalk@afeka.ac.il
ג 04:56 13/07/2021
אל: Eylon Mizrahi
עותק: Daniel Ivkovic

אילון ודניאל שלום,
הדוח הסופי מאושר.
בברכה, אייל

Table of Contents

List of Figures:

List of Equations:

List of Tables:

Acknowledgement

Special thanks to Dr. Eyal Katz, for his contributions in key aspects of our work – mainly for proposing a system that could identify forgery in handwritten documents, and for his guidance to fulfill the project's requirements correctly, starting from the advices for straight and organized work for hedging the risks to professional advises that could be critical in some phases of the project.

1. Executive Summary (Hebrew)

כיום, לאור הגידול המשמעותי ביכולות הטכנולוגיה, הצורך בנוחות הוא מאפיין חשוב עבור לקוחות. הצורך הזה גם מתאפיין בעולם עריכת מסמכים הכתובים בכתב יד. היות ואין פתרונות נוחים כיום בעולם עריכת כתב היד, אנשים מסוימים נוטים להתעכב ולבזבז את זמנם בעבודה על מסמכים כאלו, מה שגורם לעבודה פחות יעילה וחסרת סדר. הצורך לכלי שיעזור לייעל את העריכה של מסמכים אלו עולה.

מעבר לנושא היעילות, בשנים האחרונות נוצרו כלים אשר יודעים לזייף תמונות של אובייקטים מגוונים ולגרום להם להיראות אוטנטיים לחלוטין. ניתן להשתמש בכלים זדוניים אלו כדי לזייף במחשב מסמכים בכתב יד של בני אדם. לכן, אנו מוצאים את עצמינו בפני אתגר אבטחה חדש בעולם – זיוף מסמכים בכתב יד. לא נוכל לדעת האם מסמך מסוים הוא אמיתי או שנוצר על ידי מכונה. לשם כך, טכנולוגיה אשר תוכל לאמת אותנטיות של מסמך הכתוב בכתב יד ואף לאמת את זהותו של הכותב היא דבר נחוץ בשוק.

המטרות הבאות אפשרו לנו להתמודד עם האתגרים שלנו:

1. עריכה של כתב יד בעזרת חילול טקסט.
2. ניתוח אותנטיות של מסמכים הכתובים בכתב יד.

עבור הגשמת המטרות לעיל, היעדים הבאים נקבעו:

1. חילול טקסט בכתב יד על ידי מכונה.
2. אימות זהותו של כותב המסמך הכתוב בכתב יד.
3. אימות אותנטיות של מסמך – האם נכתב על ידי אדם או שחולל על ידי מכונה.

הפתרון שלנו ממומש על ידי מתודולוגיות של למידה עמוקה עם אינטגרציה בתוך ארכיטקטורת Client-Server המונגשת בתור אפליקציה לאנדרואיד, הפרוייקט רחב מבחינה טכנולוגית ואלגוריתמית ועושה שימוש בידע חדשני בעולם הלמידה העמוקה ממאמרים מהשנתיים האחרונות.
כידוע, כלים של למידה עמוקה דורשים מאיתנו להמציא דוגמאות אימון מייצגות. אחת מקבוצות האימון הייתה מאגר גדול של תמונות של מילים בכתב יד, עם התרגום המתאים שלהן וזהות הכותב שלהן.

במהלך תכנון הארכיטקטורה של הפרויקט שלנו, היו כמה אפשרויות בנוגע לאופן המימוש והכלים שבחרנו להשתמש בהם. כדי לייעל את תהליך המימוש, חילקנו את הפתרון שלנו לשלוש תתי מערכות, כל אחת מהן אחראית על יעד אחד מתוך שלושת היעדים שהוגדרו למעלה. עבור כל פאן של המערכת שלנו, חיפשנו את הפתרונות הכי טובים הקיימים בעולם האקדמי ובחרנו את אותם הפתרונות שקיבלו את התוצאות הטובות ביותר וכאלה שיתאימו לצרכי הפרויקט שלנו.

חילול של מילים סינתטיות נעשה על ידי ארכיטקטורת למידה עמוקה הנקראת Generative Adversarial Network (GAN). הארכיטקטורה החדישה שבחרנו משתמשת בעקרונות בסיסיים שלארכיטקטורת GAN, עם חידושים כמו מתן דגש על חילול מילה בעלת משמעות מילולית ומראה שזהה לסגנון כתיבה של אדם. באופן כללי המודל לומד לחולל תמונות של מילים אשר יענו לאילוצים שנכפו עליו. כל אילוץ גורם למחולל לייצר תמונות שיספקו אותו ויגרמו לפלט להיראות אמיתי ביחס לאותו אילוץ. ביחד, נקבל תמונות עם מראה של סגנון כתיבה מסוים בשילוב של תרגום ברור של משמעות המילה וחדות האותיות.

נדרוש שאימות זהות של כותב תהיה מדויקת מספיק כדי להבדיל בין כותבים בעלי סגנון דומה. לכן אנחנו מציעים ארכיטקטורה שמנתחת את המאפיינים של כתב היד ברמת המיקרו וגם ברמת המאקרו. דבר זה נעשה על ידי מיצוי של חלקים קטנים של התמונה ושילוב שלהם עם הידע המקדים הנוצר מניתוח של התמונה המקורית. באמצעות השיטה הזו ניתן להגיע למודל אשר לומד כמות גבוהה של אינפורמציה איכותית על זהות הכותב על ידי מספר נמוך של דוגמאות אשר מאפשר הגעה לרמת דיוק גבוהה באופן יחסי.

נוסף על כך, השתמשנו גם בארכיטקטורה עם גישה שונה בניסיון לשפר את התוצאות: רשת מסוג סיאמי (Siamese Network), מדובר במודל עם שני שבילי חישוב זהים עבור זוג תמונות, המודל לומד לחשב שוני בין זוג התמונות כמרחק מתמטי. כך, ניתן לאמן את המודל למזער את המרחק עבור תמונות של כתב יד של כותב זהה ולמקסם את המרחק עבור כותבים שונים לפי סגנון הכתיבה בתמונות.

באשר לאימות אוטנטיות (בדיקת זיוף) של מסמך הכתוב בכתב יד, אנו משתמשים בעקרון חדש. מקורו של הרעיון טוען שהתמרת הפלט למרחב התדר באמצעות פעולה מתמטית הנקראת DCT (התמרת קוסינוס בדידה) עשוי לייעל את דיוק המודל. הפעולה הזו יעילה עבור תרגום של מידע המצוי בפיקסלים של תמונה

למידע במרחב התדר שלה. הייצוג של התמונה במרחב התדר שלה מאפשר מציאה של אובייקטים המצויים במרחב הזה באופן בלעדי שנוצרים כאשר מכונה מנסה לזייף תמונה. המודל יעביר את דוגמאות האימון למרחב התדר וילמד לזהות את האובייקטים הללו. לאחר מכן, קיבלנו תוצאות יותר טובות במקצת עבור מודל שנלמד במרחב התמונה עם תמונות שעברו Binary Threshold.

בשלבי הפיתוח של הפרויקט, עבור חילול טקסט בכתב יד, התוצאות נמדדו על 15 אלף זוגות של תמונות אמיתיות ומזויפות, על ידי מדד הנקרא FID (Fréchet Inception Distance) והתקבל ערך של 38. ערך זה הוא יחסית נמוך ומשום כך, המודל מצליח לייצר תמונות אשר נראות אמיתיות לעין האנושית. לכן, נשאף בעתיד למזער את המדד הזה עוד על ידי שיפור יכולות הקומפוננטות הפנימיות של המודל עם דגש על תכונות סגנון כתיבה.

לבסוף, מודל אימות הכותב בגרסת האב השיגה אחוז דיוק של 97.7% על מבחן האימון שמכיל 1600 תמונות של כותב ספציפי אל מול שאר הכותבים בIAM Dataset אשר נכתב על ידי קבוצת אנשים ממוצא אחיד, דבר שעלול להקשות על דיוק המודל כאשר יידרש להבחין בין קבוצות אתניות שונות בעולם בהקשר של סגנון כתיבה. באופן דומה, בחנו את מודל אימות הזיוף שלנו והשגנו מדד דיוק של 95% על סט מבחן שמורכב מתמונות מזויפות ותמונות אותנטיות. תוכנית לעבודה עתידית עשויה לכלול איסוף של עוד תמונות כתב יד שמוצאן ממספר מודלים מחוללים שונים כדי לייצר מודל יותר עמיד. עבור אימות זהות כותב, אנו ממליצים לאסוף קבוצת אימון שמייצגת יותר קבוצות אתניות כדי שהמודל המאומן לא יאותגר כתוצאה משינויים גאוגרפיים של זהות הכותבים.

2. Executive Summary

Nowadays, with the high growth in technology, the requirement for convenience is considered important by consumers.

This trend is also reflected in the scope of handwriting editors. Due to a lack of technology offered in the scope of handwritten document management, people waste time while working on handwritten documents, work inefficiently and disorderly, and struggle to decipher the information within the document. The need for a new tool to maximize efficiency regarding these kinds of documents is growing.

Besides these issues, these times with the rapidly growing free access to deep-faking tools, document forging is becoming easier and faster, and working with scanned handwritten documents creates new security challenges, and now, we could never really know whether the document presented to us is authentic or not. Therefore, a technology that could validate the authenticity of handwritten documents or validate the author's identity is surely needed.

The following goals allowed us to deal with our problem:

1. Editing of handwritten documents using generated text.
2. Analyzing the text authenticity of handwritten documents.

In order to satisfy the goals above, it is necessary to meet such objectives:

1. Generation of general human handwritten text.
2. Validation of whether an author has written a given handwritten document.
3. Validation of whether a document is authentic or edited/generated by automated means.

Our solution is implemented using deep learning methodologies integrated inside a Client-Server architecture that is available as an Android application, The project is vast in terms of technology used and algorithms designed and uses state-of-the-art methods from papers written recently.

As known, deep learning tools require us to collect representing training data. One of the datasets was a large reserve of handwritten text images and their corresponding transcription labels, and the other one was generated by one of our artificial neural networks in order to teach how to detect fake images.

During the planning of the product's architecture, there were a few options regarding the implementation method and the tools that we chose to use. In order to optimize the implementation process, we partitioned our solution into three main sub-systems, each of which is responsible for one objective described before. For any aspect of our system, we searched for the existing state-of-the-art solutions and chose the ones that presented the best results and would best fit our needs.

The generation of synthetic handwritten text was done by a deep learning architecture known as a generative adversarial network (GAN). The chosen state-of-the-art architecture that we use is utilizing the basic concepts of adversarial training with modifications such as word content and handwriting style conditioning. In general, the model is learning to generate images that would satisfy the critique of 3 other models – each of these navigate the generator to output a handwritten word image that will contain a specific transcription on the one hand and realistic features to the external observer on the other hand.

The author validation of documents is required to be precise enough to differentiate between similar authors in real-time. Therefore, we suggest an architecture that analyzes both the high-level and the low-level features of a given author. This is done by extracting different fragments of an image and combining them with the knowledge gained from the rest of the fragments, all of those are combined with the knowledge gained from the

original image. This results in a model that learns a high amount of information from a single image, allowing us to train it on a small amount of data and achieving high accuracy in author validation.

In addition, an architecture with a different approach was used to try and optimize results: the Siamese neural network is a model with two identical pathways that learns to calculate distances between a pair of images, each of which go through the same feature extraction process. This way, the model could learn to minimize distances for handwritten images with the same style and maximize the distance for different styled images.

As for the validation of the authenticity of a given document, we use a new concept. The idea lies in transforming an input document/word image into the frequency domain, using the mathematical operation, known as DCT (Discrete Cosine Transform). As a kind of Fourier transform, this operation is useful to translate pixel data into its representation in the frequency domain. This representation utilizes the process of learning features of synthetic images, by exposing artifacts that are easy to be detected by simple machine learning models. We have later achieved slightly better results using an image domain model with binary threshold pre-processing.

While developing the project, regarding the word generation, the results in terms of measuring the distance between the distribution of 15,000 synthetic images and the distribution of 15,000 real ones, using the FID (Fréchet Inception Distance) measure are standing at 38, which is a relatively low value. Therefore, the generation model generates words that look similar to the human eye. Hence, we will seek to minimize the distance even further, by fine-tuning our model's inner components, prioritizing the handwriting style features.

Finally, the author style-validation system prototype achieved 97.7% accuracy on a set of images that contain 1600 words both of a specific writer and other writers from the IAM dataset which is ethnically uniform, thus making the classification challenging for writers from around the world. Similarly, in order to validate the authenticity of handwriting, we tested our model by measuring accuracy of 95%, tested on synthetic images, generated by our word generator, and the same amount of authentic word images. A future plan might be to try to get more artificial data from various generative models, in order to create a more versatile authenticator. For writer identity, we recommend collect a more representative dataset containing more styles.

3. Introduction

The analysis of handwriting is usually studied as initial steps to become a deep learning engineer in the field of computer vision and usually starts with the implementation of simple pattern recognition neural networks such as single-digit classifiers. Most of the time, this is where the attention to this subject ends and deep learning engineers typically start to pursue other types of projects. We believe that contributing to this field more profoundly can open the possibility for the creation of new usable and interesting products and some of them will be implemented in our work.

Our project is intended to be both research and an attempt to implement real-world tools. Its purpose is to examine and strive to expand the field of handwritten documents in the scope of computer vision, and to analyse the features of handwriting styles that different people have.

We used our newly acquired knowledge in order to allow a machine to create an artificially generated handwriting style that may resemble a specific individual. Next, we used the generative model in order to create a real-fake dataset that will allow us to train a deep-fake validator model that would learn to distinguish between authentic images and artificially generated ones, that would else be indistinguishable to the naked eye. In Addition, we aimed to create a validator model that could learn to calculate distance between pairs of images in terms of writing style that would allow us to create a tool capable of stating if two given documents were written by the same author.

As for the division of our work on the project, we divided each subtask evenly between us while still having both taken part in major key areas of the project.

We have both participated in the gathering and organization of our initial datasets as well as the creation of new unique datasets required explicitly for our objectives. In addition, we were both involved in the procedure of training our models and have profoundly discussed ways to improve the training procedure and innovate in the field of computer vision. Once we were contempt with our results, we integrated our models' capabilities into an application.

In addition, Eylon Mizrahi has fully implemented the front-end side of our app via the React Native framework, including the processing of input and output through the client-side REST API. Daniel Ivkovich was in charge of implementing the bridge between the trained

text generator and its usage in the app on the back-end side including developing image processing algorithms to fit the needs of our unique text editor, such as addition and deletion of words from and to an existing file and fitting these functionalities for the REST API on the client-side.

4. Goals, Objectives, and Measures

In order to examine and develop these notions, it is necessary to define goals, objectives regarding these goals, and measures for estimating the achievement of these. All of these are described as follows:

4.1. Goals

The goals of our system are:

4.1.1. Editing of handwritten documents using generated text.

4.1.2. Analyzing the text authenticity of handwritten documents.

4.2. Objectives

The objectives are divided into **Data Objectives** and **Functional Objectives**:

4.2.1. Data Objectives

4.2.1.1 Finding a large, tagged Dataset of handwritten text, including a sufficient number of examples with a balanced distribution of letters in the dataset.

4.2.1.2 Creating a new Dataset containing real author examples and generated examples that shall be used to train the fake detection sub-system.

4.2.2. Functional Objectives

4.2.2.1 Generation of general human handwritten text.

4.2.2.2 Validation of whether an author has written a given handwritten document.

4.2.2.3 Validation of whether a document is authentic or edited/generated by automated means.

4.3. Measures

During our project, it is essential to set relevant measures in order to fulfill the pre-defined objectives:

4.3.1.1. Achieving at most 90 Fréchet Inception Distance (FID) score for the generation of handwriting.

4.3.1.2. Achieving at least 85% accuracy and precision (minimizing the false positive rate) scores for author identity validation.

4.3.1.3. Achieving at least 75% accuracy and precision (minimizing the false positive rate) scores for the authenticity validation system.

4.4. Compliance with measures / objectives

Similarly, the partition of the objectives above, it is necessary to consider two subcategories. All measures were met or exceeded, as detailed below:

4.4.1. Compliance with the Data Objectives

4.3.1.1 We used the IAM handwriting dataset which contains 657 writers, and 115,320 isolated and labeled word images. Also, we collected another small amount of handwritten word images (above 800 samples) for the use of the writer identification system. By examining the writers'/words' histograms, we gained relatively balanced data.

4.3.1.2 We created a real-fake dataset, containing roughly 60,000 samples for each class (120,000 total).

4.4.2. Compliance with the Functional Objectives / Measures

4.3.2.1 **Two major models were evaluated in the scope of Handwriting generation:**

4.3.2.1.1 Using the GANWriting architecture [13]: We managed to achieve a 38.06 FID score after training the model. (38.06<90, which complies with measure no'4.3.1.1).

4.3.2.1.2 Using the handwriting line-generation model [14]: We did not calculate this metric for this model as we used its pre-trained weights which were estimated in the paper by 20.65 FID score (20.65<90, which also complies with the same measure.

4.3.2.2 **Two major models were evaluated in the scope of Writer Identification:**

4.3.2.2.1 The metrics for the classifier which was trained to identify a unique writer from the IAM dataset are Accuracy: 98% (98%>85% which complies with measure no'4.3.1.2), precision: 97% (97%>85% which also complies with the same measure). In addition, we assessed the metrics: recall: 98.4%, F1: 97.6%.

4.3.2.2.2 The metrics for the model which was trained to determine whether a couple of handwritten word samples are written by the same writer, regardless of the handwriting ethnicity style of the writer: 81.4% Accuracy (81.4%<85% which exceeds from measure no'4.3.1.2), precision: 79% (81.4%<85% which exceeds from the same measure), In addition, we assessed the metrics: recall: 83%, F1: 81%.

4.3.2.3 **Our deepfake recognition model was assessed as:**

14.1.1. Our deepfake recognition algorithm reached 97.8% (97.8%>75% which complies with measure no'4.3.1.3) accuracy value and 97.6% precision value (97.6%>75% which complies with measure no'4.3.1.3 in order to minimize the false positive rate) on 1,500 Real images and 1,500 generated images by our handwriting generative system.

5. Literature Review

The main focus of this section is to explore each functionality within the scope of our system among the existing literature. As mentioned earlier, there is a large number of goals in the scope of our project. To simplify our research, we divided our review into three parts of responsibilities, including Deep-Fake Generation, Deep-Fake Detection/Recognition, and Writer Identification:

5.1. Handwriting (Deep-Fake) Generation Review:

At the user level, after detecting the words from a given document using the known Microsoft OCR tool, we would like to build a handwriting generation model, based on the writer's handwriting style. As the current image generation methods have reached impressive quality levels, they are still unable to produce plausible yet diverse images of handwritten words. Some papers on this topic claim to deal with this problem by the conditioned GAN (Generative Adversarial Network) technology.

In the GANWriting paper [13], the authors propose a generator model, which is guided by three complementary learning objectives: To produce realistic images, to imitate a certain handwriting style, and to convey a specific textual content. These objectives are achieved by three different sub-models, each of which is responsible for a specific objective as described: Discriminator model, Writer Classifier, and a Text Recognizer. The proposed generation model is unconstrained to any predefined vocabulary, being able to render whatever input word. Given a sample writer, it is also able to mimic its calligraphic features in a few-shot setup. This research used the IAM-words handwriting dataset, which contains thousands of words and different styles in order to train its models. The results shown in the paper were measured by the well-known FID (Fréchet Inception Distance) score and are 90.43.

Another state-of-the-art project on this topic is the Handwriting Line Generation paper [14]. Here, the authors generate images of lines of handwriting, conditioned on the desired text and a latent style vector. Handwriting is an expressive and unique form of communication that is often considered more intimate than typed text. Generating images that mimic an author’s style would allow people to

generate their own handwriting from the typed text. The paper's results achieve human plausibility and begin to approximate the style of example handwriting images. Handwritten image generation can also provide additional data to train more accurate general handwriting recognition models. Generating a large number of images in the style of each user of an application could allow us to train personalized single author recognition models. Personalized models tend to be more accurate for their target author's writing than a general recognition model. Their method uses widely available offline handwriting data, i.e., images of the physical media. They frame handwriting generation as conditional image generation, directly learning from and predicting pixels. The paper's approach achieves realism by combining Generative Adversarial Networks (GAN) and autoencoder methods with an auxiliary loss to achieve legibility. The encoder can map an example image into a latent style space, and then the model can produce images in a similar style, either reconstructing the original or using arbitrary text. The width of an image of real handwriting depends on the text and writing style. Instead of fixing the model's output size or heuristically determining the output width from the input text, a deep network is used to predict the horizontal layout of the characters. The model achieves state-of-the-art visual quality for offline handwriting generation and, unlike prior methods generate entire lines of handwriting conditioned on arbitrarily long text. It is challenging to train a network using many (sometimes competing) loss functions that yield gradients that differ in size by orders of magnitude. To overcome this, the authors propose an improvement upon the gradient balancing technique. There are potential ethical concerns due to nefarious uses of this technology, e.g., low-skill convincing forgery. However, we believe this concern is minor as the method is not targeted at imitating signatures and can only produce digital images, not physical documents.

The papers primary contributions are:

1. A method combining GANs and autoencoders to train a handwriting generator on offline handwriting images to produce realistic handwritten images that mimic example image styles.
2. A model that generates variable-length handwritten line images from arbitrary length text and style.
3. Improved multi-loss training through gradient balancing, allowing disparate losses to be used together more easily.

Lately, a newer version of a handwriting generation method called TextStyleBrush [17] was published. This generative model uses the same notions that were presented before, in addition to some new ones as focusing on the background style too, and not only on the calligraphic style. This architecture is built from varied convolutional subnetworks:

First, a style encoder that contains the style of the whole image, including both the background style and the text style. Its input is the whole original text image (could be handwriting), without any background noise-cleaning. This encoder uses a Mask RCNN architecture in order to separate the text from the background and to feed it into the Generator.

Second, a content encoder gets two input binary-segmented word images: The first one is the exact text and font/style of the text within the style encoder's input and the second one has the same font/style of the original text, but including different content to generate.

Third, the Generator subnetwork produces text conditionally by the style produced by the stye encoder and the content produced by the content encoder. By the former ideas of the CNN units such as Recognizer, Discriminator, and style identifier (called typeface) the TextStyleBrush architecture is able to generate text\handwriting at any background with a result of 44.68 FID score.

5.2. Handwriting Writer (Style) Identification Review:

After detecting each word written by a user, another scope of our project is on the topic of handwriting style identification. In a more accurate explanation, this scope deals with author validation with emphasis on having a small amount of text to compare with. Because there is only a small amount of text to decide upon there is a need to extract powerful features on these word images. In order to do so, a deep neural network, named FragNet [9], is proposed as our first method. The FragNet has two pathways: the feature pyramid which is used to extract feature maps and the fragment pathway which is trained to predict the writer's identity based on fragments extracted from the input image and the feature maps on the feature pyramid. As classic CNNs can only learn deep abstract and high-level features on a small amount of text, such as word images or text blocks which contain several characters, the proposed model was also trained on the low-level features, given by the different fragments cropped from the input word images.

The proposed method is evaluated on four datasets: IAM, CVL, Firemaker, and CERUG-EN. IAM is a widely used dataset for writer identification. There are 1,452 pages, which are written by 657 different writers and each writer contributes a variable number of handwritten pages. CVL contains 310 writers, and each writer contributes at least 5 pages (27 writers wrote 7 pages). In this paper, the first three pages are used for training and the rest pages are used for testing. Similar to the IAM dataset, word images are also available on this dataset. The Firemaker dataset contains handwritten documents from 250 Dutch subjects and each writer contributes four pages. The CERUGEN dataset contains handwritten documents from 105 Chinese subjects with two paragraphs in English. In the paper, they used the first paragraph for training and the second paragraph for testing.

One important parameter for fragment segmentation is the size of the fragment corresponding to the input image. In this paper, they used a square window with the size of q × q to cut the fragment in the input image. The resulting architecture is denoted as FragNet-q with the fragment size of q × q in the input image. For each fragment, the loss is defined as the cross-entropy loss between the prediction and the ground-truth.

Note that the aim of the proposed method is not to achieve state-of-the-art performance, but to show that the fragment-based network can improve the performance and can be visualized for end users. In addition, the proposed FragNet could be used for on-line writer identification with a sliding window strategy. One of the limitations of the proposed FragNet is that it needs word image or region segmentation, which is challenging on the highly cursive writing documents. This limitation leads to the direction of future works, focusing on extending FragNet for writer identification on any handwritten document without segmentation.

In the past few months, another relater paper was published by the same authors [18]. This paper mostly refers to FragNet [9], while improving its results using the time factor as a relevant parameter, along with a GRU architecture. This paper has yielded results of 86.1% accuracy on the IAM dataset.

Another related work is on the subject of signature verification, which may also be useful to identify writers by their handwriting font style. This work called SigNet [15] and proposes an architecture of a convolutional Siamese network. The datasets used in this paper are CEDAR, GPDS300, GPDS Synthetic, and BHSig260. All of these datasets are composed of signature image samples.

Based on the notion of One-Shot-Learning, Siamese networks usually contain two identical subnetworks (in our case - CNNs) with shared weights, which can be trained to learn a feature space where similar observations are placed in proximity. This is achieved by exposing the network to a pair of similar and dissimilar observations and minimizing the Euclidean distance between similar pairs while simultaneously maximizing it between dissimilar pairs. Hence, a regression loss function is needed such as Contrastive Loss, which is an acceptable loss function in the scope of this kind of architecture.

The idea of One-Shot-Learning deals with a few problems that are involved when using a regular classification technique e.g. classic CNNs. First, a lot of data is necessary for each class in order to classify objects by their features, as neural networks initialized with no prior knowledge about these objects. Second, classic

classification cases are not scalable: In general, adding a new class to the classification problem or removing one from it requires model retraining. When prior knowledge exists, the model is able to use it without the need to learn this knowledge by itself, in other words, a smaller number of features to learn may lower the need of a large number of samples. In Siamese networks, this prior knowledge is reflected by learning the generic distances between pairs of images, having the model familiar with the first image (from the training procedure), thus less sensitive to new data. In case that a new object is being validated, there is no need to retrain the model, because the model already knows how to calculate distances from any object to the training references.

These projects are very relevant to us because, in our system, we believe that the player will ask for author validation using a small amount of validation text and this is exactly the solution brought by this paper. We would like to understand how to implement author identity validation for the security scope of our system and we hope that we could learn how to use the methods shown in the papers.

5.3. Handwriting Deep-Fake Detection / Recognition Review:

Following the process of generating a new synthetic dataset, we could use this data to train a model for deep-fake detection/recognition. A paper on this topic [11] copes with the security problem of forgery by automated means, as the GAN model mentioned before is capable of. The datasets used in the paper were the CelebA faces images and LSUN bedrooms, which do not include handwriting images. Also, in order to learn the features of the generated images, the authors created fake datasets, generated by varied GAN models such as StyleGAN [16].

To achieve this goal, the authors focused on the preprocessing of images, rather than creating complex architectures. In fact, one of their goals was to achieve a high accuracy result using non-complex models, trained on the preprocessed images. The main idea was to transform the images into the frequency domain, by the DCT (Discrete Cosine Transform) operation. The DCT is commonly used in image processing due to its excellent energy compaction properties and its separability, which allows for efficient implementations. Together with a circular

convolution-multiplication relationship, it enables fast filtering. The paper reveals that in frequency space, GAN-generated images exhibit severe artifacts. The artifacts are caused by upsampling operations found in all current GAN architectures, indicating a structural and fundamental problem in the way images are generated via GANs. It is shown that these artifacts can be easily identified by non-complex models, for example, a simple CNN.

In general, three main steps were made on both the real and fake datasets, as the preprocessing stage: DCT operation, log-scale the coefficients and, finally, normalize them by removing the mean and scaling to unit variance. In the paper, they had varied experiments, considering different classifiers, different noises and augmentations, and the best CNN-DCT result measured at 99.64% accuracy on the LSUN dataset. In order to use this work in our system, we will have to investigate the scope of handwriting generation in both the image and the frequency space. The goal here is to reduce the problem of deep-fakes, learned in the paper to the scope of handwriting.

6. Competition

According to our scope of the system which includes different functions to manipulate handwritten text, the following table presents our features against other apps from a similar or a wider scope:

Table 6. 1 - Competition

Apps / Goals	**6.1. GR-RNN** [18]	**6.2. TextStyleBrush (Facebook)** [17]	**6.3. Google Lens**	**6.4. Pen To Print**	**6.5. Handwriting OCR**	**6.6. Our Proposed System**
Handwritten Text Detection	✓	✓	✓	✓	✓	✓
Writer Identification	✓	✗	✗	✗	✗	✓
Real/Fake Handwriting Detection	✗	✗	✗	✗	✗	✓
Handwriting Faking and Editing	✗	✓	✗	✗	✗	✓

Details about the above competition are given in the following paragraphs.

6.1. **GR-RNN [18]:**

https://github.com/shengfly/writer-identification

A new state-of-the-art paper in the scope of writer identification. Given a set of handwritten word images written by the same writer, it can learn the writer's specific style and predict if another set of test words are written by the same writer or not. This paper contains a code implementation by the notion of FragNet [9] and another improved one, by the same notion.

6.2. **TextStyleBrush (Facebook) [17]:**

https://ai.facebook.com/blog/ai-can-now-emulate-text-style-in-images-in-one-shot-using-just-a-single-word

New state-of-the-art research in the scope of handwriting generation and editing. This paper suggests a new and accurate method to edit handwritten text which lies at any kind of background. This research did not publish its application yet.

6.3. **Google Lens:**

https://lens.google.com/

Translate text in real-time, including handwritten text. Also, look up words, add events to your calendar, call a number, and more.

6.4. **Pen to Print:**

https://www.pen-to-print.com/

Pen to Print is the first handwriting-to-text app converting scanned handwritten notes into digital text available for edits, search, and storage in any digital platform. Although the digital text is easier to edit, search and store, handwriting on paper is still commonly used, since it's fast, easy, and accessible.

Pen to Print's handwriting recognition is a great solution for those who still like the feel of pen on paper but want to enjoy the benefits of digital.

It is the best way to convert scanned handwriting to text, it is easy to use, fast and affordable.

6.5. **Handwriting OCR:**

https://play.google.com/store/apps/details?id=com.globalbusiness.ocr&hl=iw

Recognize handwritten text from notes, letters, essays, whiteboards, forms, and other sources. Be more productive by taking photos of handwritten notes instead of transcribing them. Only available for English text now.

6.6. **Our proposed system:**

After analyzing all of the above solutions against the scope of our system, we found that many of the existing solutions overlap with only one of our suggested features, detecting handwritten words within a document. Although the new project (1) could improve our goal to identify a specific writer by its calligraphic style its integration

in our system does not lack problems. In contrast to (1), we suggested a way to distinguish between writers by their handwriting style, without focusing on the actual style features but on the difference of the two given samples. Thus it is more robust to the ethnicity of the writer can be used by writers from any ethnic group. But, our method is still limited to a subset of only 5 text labels.

The great solution of (2) for editing handwritten documents could improve our scope of handwriting generation and editing, but there is no published implementation yet.

Also, our project supplies a real/fake handwriting detection system, to deal with the problem which arises from handwriting editing.

7. System Alternatives

The current section describes in general lines the different system alternatives for our project. These alternatives present different ways of realizing or developing the project in manner of complexity (runtime and storage), convenient cost, and functionality:

7.1. Computation Alternatives

7.1.1. Editing of handwritten documents using generated text.
7.1.2. Cloud computations: All deep learning model computations will take place on the cloud.
7.1.3. Runtime computations of a specific device: All computations will take place on the device RAM.
7.1.4. Parallel computations: Heavy functionality shall be computed using parallel methodologies either on the cloud or on the device RAM

There are some services that deal with these alternatives. Each row in the table describes whether the current service has a high cost or not and whether the service is capable to deal with our proposed models, in terms of storage and performance. The scores are given on a scale between 0 to 3:

Services	**Cost Level (0-3)**	**Storage (0-3)**	**Performance (CPU/GPU/RAM) (0-3)**	**Result**
AWS EC2 instances	0	3	3	6
Google Colab Pro	3	3	2	8
Afeka's Local GPU	3	3	2	7

Table 7. 1 – System alternatives

At the beginning of the project, we used an AWS EC2 instance which is known for its high performance when it comes to Deep Learning algorithms, as some of the architectures we dealt with require a high GPU computing power e.g. GAN. But, after this expensive experience, we found that the Google Colab Pro service satisfies both our needs – cost and performance. In addition, our local device was discovered as satisfactory regarding the inference server of the application.

In addition, the major functions of our system were examined against a variety of deep learning architectures, hence, we defined the following alternatives:

7.2. Functional Alternatives

7.2.1. **Handwriting Generation Alternatives:** Two alternatives given by two papers on this scope – **GANWriting** [14], **Handwriting Line Generation** [14]. We chose to use the second architecture as it is a newer system that is based on the GANWriting model's idea to generate handwriting, conditioned with a given calligraphic style.

7.2.2. **Writer Identification Alternatives:** A few versions based on the notion of the **FragNet** [9] architecture, a **Resnet50** classifier, a **Siamese** NN architecture, based on Resnet50 backbones, or the **SigNet** [15] model. After analyzing the results, we chose to use the Resnet50 CNN as it achieved the best measures, in a situation when the target writer belongs to the IAM ethnic group. Another alternative is whether to refer to this problem as a **recognition** problem or a **detection** one. With the Resnet50 classification solution, we can learn the handwritten style of such writers and **detect** handwritten words that are compatible with their style within a given document. But, when dealing with different types of ethnicity, different from the one from the IAM dataset, the Siamese NN architecture is necessary. The problem of optimally calculating the distances between two different calligraphic styles was harder than expected, thus we defined our solution as a document style **recognition** solution, by using all of the distance calculations between words within a document. Additional details on this subject can be found in the Results section (number 14).

7.2.3. **Handwriting Authenticity Validation Alternatives:** Using a Resnet50 architecture, we evaluated our model against both domains – **the image and the frequency**. After the evaluation process, we decided to rely on the image domain, in contrast to the paper [11] that did not cope with the topic of handwriting analysis. Indeed, when training the model on a dataset from both of the domains, there was no distinction between both of the results in the manner of detection of upsampling artifacts. By using an external OCR system and classifying every word within a handwritten document, we defined our solution as a handwritten word **detection** solution.

8. System Requirements

This section summarizes the system requirements, relying on the SRD document. Also, the details below present the preserved requirements against the changes that were made with the development progress.

8.1. Functional Requirements

8.1.1 Insertion of a general-styled handwritten text to a chosen location within the document. We met this requirement using the model from [14].

8.1.2 Insertion of the writer's authentic handwriting to a chosen location within the document. This requirement has been partially met as there is a major challenge to mimic the author's low-level style features (the exact complex view of letters). On the one hand, the generative models can easily impersonate the high-level features of a specific handwriting style, e.g. thickness and tilt. On the other hand, both of the models – [13] which we trained and [14] can imitate a given style on a scale of quality levels, depends on the complexity level of a unique handwriting font. After discussing with the author of [14] about this challenge, we have been told that such a challenge indeed exists.

8.1.3 Validation of the authenticity of a given a handwritten document: This requirement has been met by implementing a handwritten generated word detection algorithm.

8.1.4 Validation of the author's identity given a handwritten document. This requirement has been met when the inference image can be interpreted by the IAM words dataset features, as the ethnicity feature.

8.1.5 Deletion of handwritten text from a chosen location within the document. This requirement has been met by some of our image processing implementations.

8.2. Non-Functional Requirements:

8.2.1. **Performance** – The response time of the system will stand at a maximum of 2 seconds for buttons and UI functionality, and a maximum of 15 seconds for heavy pre-processing computations on uploaded files, and 15 minutes for real-time training. This requirement was measured manually using timers.

8.2.2. **Supportability** – The system will supply time measuring for all of its functions. Due to the fact that this requirement is more of a bonus feature and less important for a beta version, we have decided to remove this requirement at this point in time.

9. Software Specifications

This chapter will include the software specifications and the systems` design following the logic in the attached SRD document.

9.1. Player description

The system provides solutions to different kinds of problems and therefore will appeal to a variety of players. Every entity with an interest in editing handwritten text and every entity that requires help with detecting fake documents or asserting author identity for a document might be a potential player of the system.

9.1.1. **Main Player:** students, teachers, forensic departments.

9.1.2. **Secondary Players:** security companies, police, army, smartphone-users, etc.

9.1.3. **Stakeholders:** banks, security companies, police, army, smartphone-users, students, etc.

9.2. Block Diagram

The concept of our system is demonstrated by a block diagram, as can be seen in Figure 9. 1. Our concept contains three main models: A word generator, a deep fake recognizer, and a writer identity validator. Each of them is presented as a unique subsystem of our project. The general communication between the player (inputting a document) and the system is described as follows and by the diagram's pipeline:

9.2.1. The player uploads a handwritten document image into the system.

9.2.2. The document is passed through preprocessing functions (noise cleaning, etc.) and a word detection computation.

9.2.3. The player chooses an option: edit the document, analyze the authenticity of the words within the document, or analyze the author's identity by the document.

9.2.4. Each of the subsystems passes the document through its unique algorithms, of the structure: preprocessing, main functionality, and postprocessing.

9.2.5. The desired results are rendered on the player's display.

The above processes are summarized in

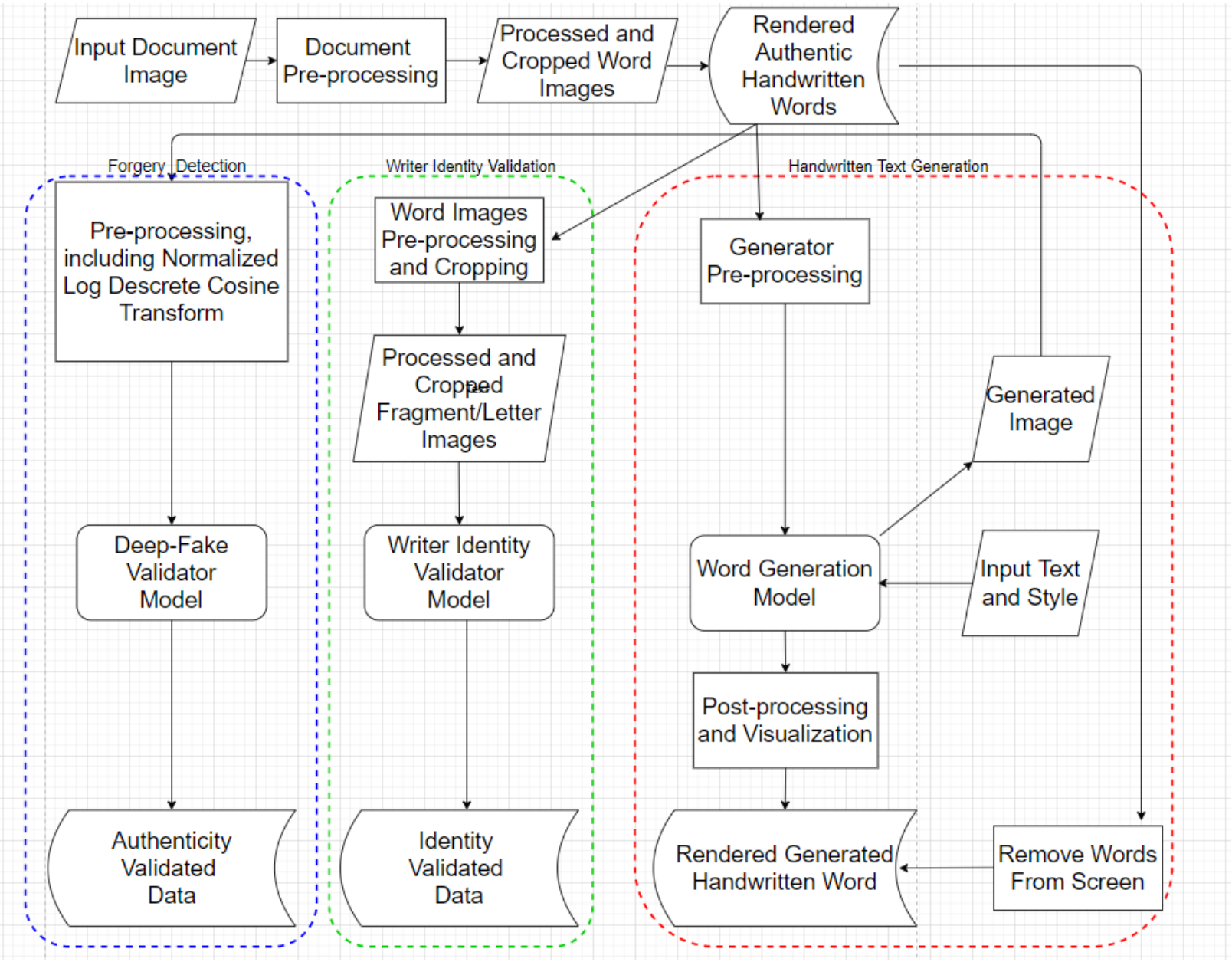


Figure 9. 1 – Block Diagram

The system is further detailed in chapters 11 and 12.

9.3. Usecase Diagram

The functionality of our system could be simply depicted by a Usecase diagram, which includes the main blocks in our block diagram (9.2) and visualizes the communication of the player which is defined in 9.1, with these system blocks:

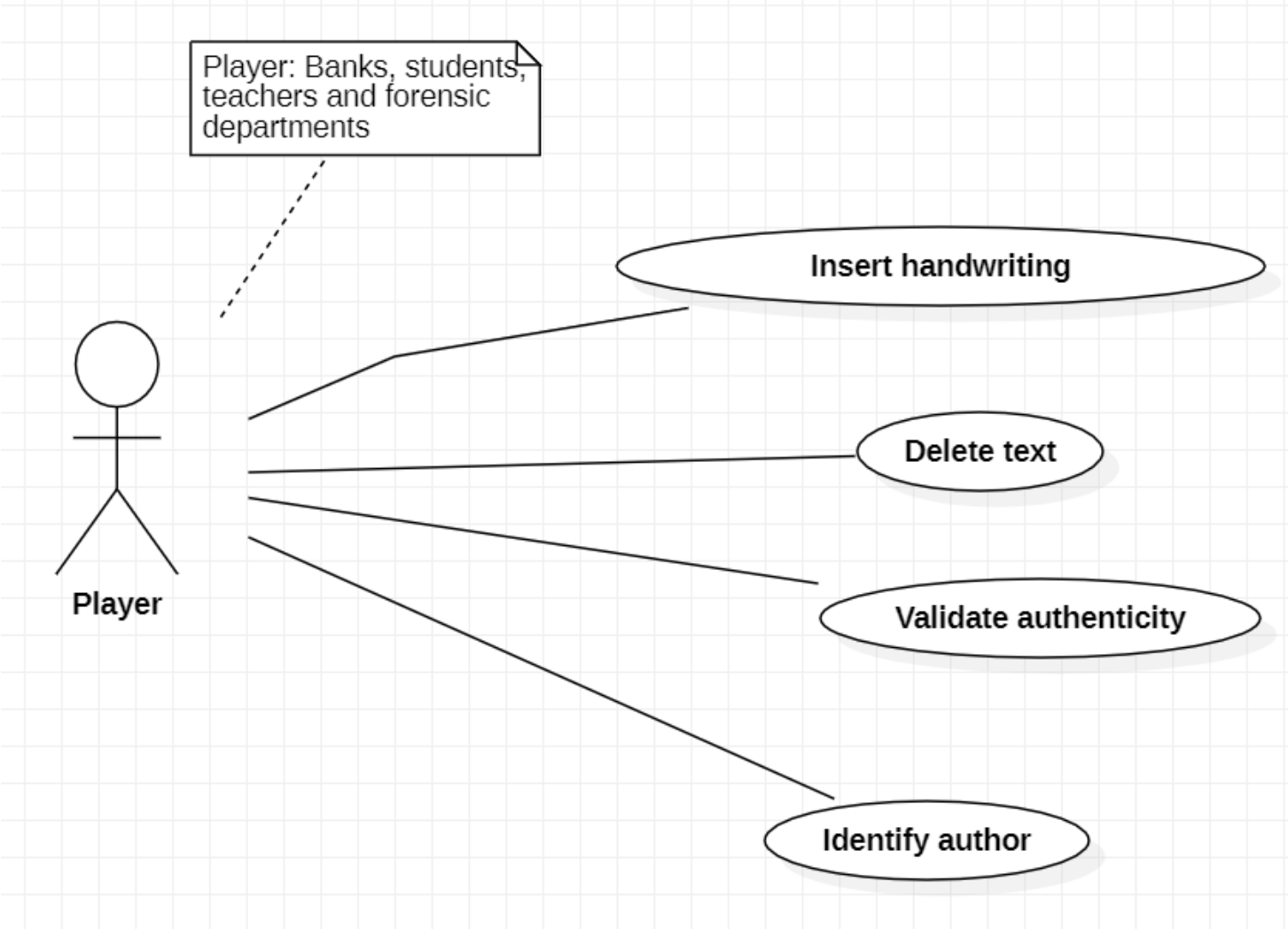


Figure 9. 2 - Usecase Diagram

10. Technological Alternatives

There are varied technological alternatives (Programming Languages, Frameworks, and libraries) for implementing the different tasks of our project. An important step is to consider the advantages and disadvantages of each of them in order to utilize the project's progress and to avoid system risks. These alternatives are presented as follows:

10.1. Programming Languages

10.1.1. C/C++ programming languages.

10.1.2. Python programming language.

10.1.3. Java programming language.

10.2. Deep Learning Libraries

10.2.1. Pytorch library (C++ or Python).

10.2.2. TensorFlow library (C++ or Python).

10.3. Front-End Frameworks

10.3.1. React framework for web UI.

10.3.2. React Native framework for web mobile applications.

10.3.3. Flutter framework for application development.

10.3.4. Angular framework for web mobile applications.

We scored the relevant measures for each alternative in the table below and chose the highlighted ones(in blue).

Services	Convinience/Familiarity	Ease of use	Traceabillity with literature	Total Score
Programming language				
C/C++	4	2	0	6
Python	5	5	5	15
Java	5	3	0	8
Machine learning library				
Pytorch	5	4	4	13
Tensorflow	2	3	3	8
Front-end library				
React	4	4	N/A	4
Flutter/Angular	0	4	N/A	0

Table 10. 2 – Technological alternatives scores

As the project is complex enough, we chose to use the technologies that we are most experienced with, over the other ones. Also, our solution is based on some of the implementations from the literature review, which use these technologies either, and our project's needs are satisfied by them. We chose Python as the programming language and Pytorch as the Deep-Learning API for the image processing and the computer vision parts. We also chose Python as the server-side programming language. For the client-side, we chose to use React Native and Javascript as the framework/programming language.

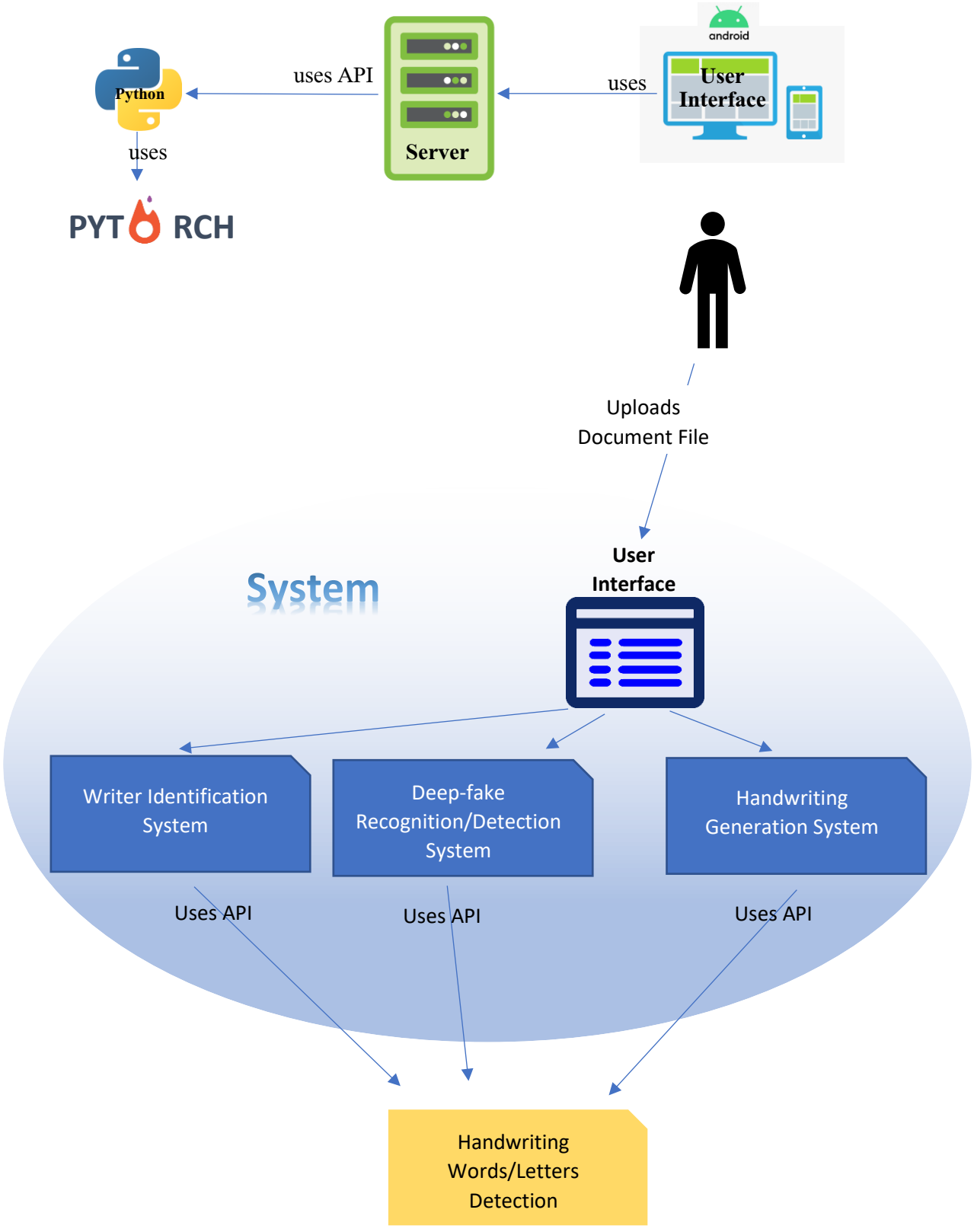
11. Software Design

As mentioned before, our system is a composition of three main sub-systems, each contains modules that deal with a specific problem. In order to manage player requests, we will wrap our system with a server that deals with them. The Basic flow of communication between the player and the system modules are as follows:

1. The player uploads a handwritten document file.
2. The player chooses an option as preferred by him.
3. The controller navigates the players request through the user interface, to one of the sub-systems that contain the relevant modules and functionality:
    - Add words to the screen, conditioned by the existing word images within the screen and a given input text.
    - Learn and identify the document writer's style, using both word and word fragment images features.
    - Validate the authenticity of the document by the given detected words or the document itself.
4. The chosen sub-system requests the cropped/processed data from the Handwriting Detection/Recognition system (that can be found outside the system).
5. The chosen sub-system deals with the player request and render the deserved results on the GUI.

The Front-end side of our project consists of controllers that match the players requests, the player may choose to upload a handwritten document, and analyze an uploaded document. The corresponding requests are handled at the back-end side and are sent to varying possible computation pathways respectively. Every pathway has its unique deep learning architecture that would satisfy a specific goal on a given document via image processing and computer vision algorithms.

The compatible architecture diagrams are presented on the next page.


Python
uses API
Server
uses
android
User Interface
uses
PYTORCH
Uploads
Document File
System
User Interface
Writer Identification System
Deep-fake Recognition/Detection System
Handwriting Generation System
Uses API
Uses API
Uses API
Handwriting Words/Letters Detection

The diagrams above describe the high-level concept of communication between the different technologies and components within our system. In simple words, the system uses a UI that communicates with the system's modules, which are localized on the system server. The server uses the relevant APIs in order to calculate the player requests.

We preferred this type of design over others, because there are multiple computation pathways available to the players in the system, and each of them affects a global document instance. Thus, separating each functionality from the others would result in a more orderly way of executing calculations and would prevent unnecessary bugs in different areas of the code while providing high coherence inside each module and low coupling between them.

12. Methods: Architectures and Algorithms

This section contains elaborated information regarding the methods we used to meet the requirements of the project and to achieve the goals. The major concepts of our system are from the scope of deep learning, each solution is supplied by a different method from this scope.

The following figure is a modified version of Figure 9. 1 – Block Diagram, showing the section we are located in.

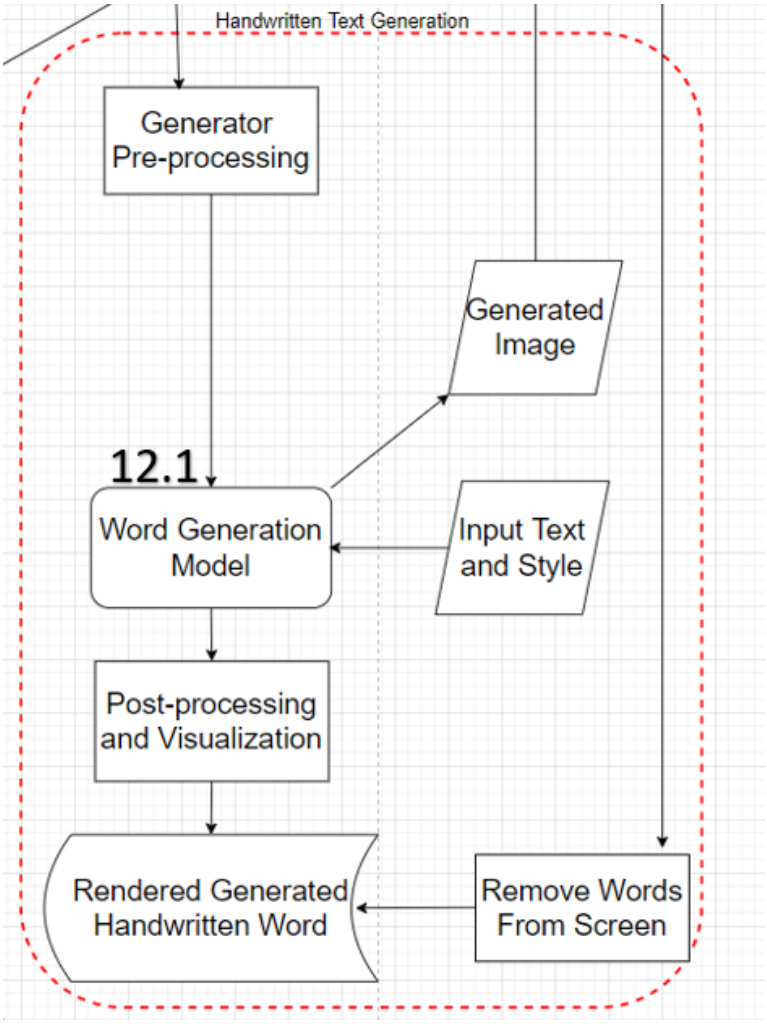


Figure 12. 1 – Word Generation model block flow

The first subsystem we elaborate on is the handwriting generation/editing system.

12.1. Generative Adversarial Networks

In order to generate handwriting conditioned with text and calligraphic style, we used two main architectures from two main papers we deeply examined [13] [14]. The first architecture we dealt with was the GANWriting [13] model.

12.1.1. **The GANWriting model:**

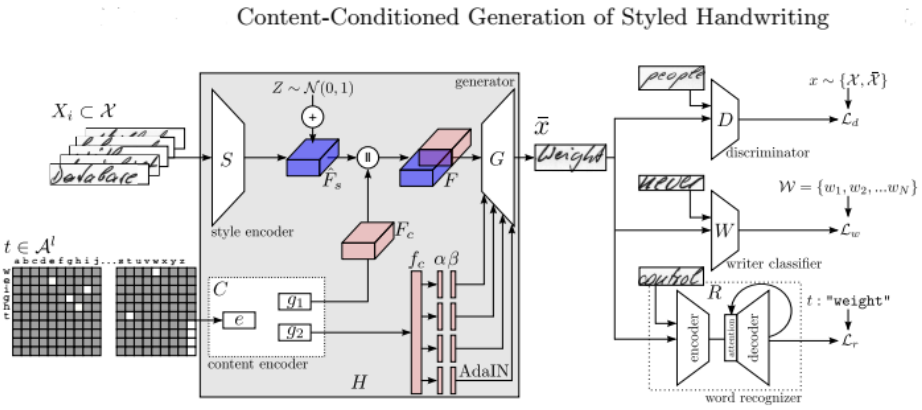


Figure 12. 2 - Style and content conditioned GAN architecture

Let {X, Y, W} be a multi-writer handwritten word dataset, containing grayscale word images X, their corresponding transcription string Y, and their writer identifiers W. Let A be the alphabet containing the allowed characters, A^l being all the possible text strings with length l. Given a set of images X_i as a few-shot example of the calligraphic style attributes for writer w_i on the one hand, and given a textual content provided by any text string $t \in A^l$ on the other hand; the proposed generative model has the ability to combine both sources of information. It has the objective to yield a handwritten word image having textual content equal to t and sharing calligraphic style attributes with writer w_i. Following this formulation, the generative model H is defined as:

$$\bar{x} = H(t, X_i) = H(t, \{x_1, \ldots, x_K\})$$

Equation 12. 1 – Generative model H

The proposed generative architecture H consists of:

S - calligraphic style encoder.

C - textual content encoder.

G - conditioned image generator.

D – classic image discriminator.

W – writer classifier.

R – word recognizer.

S - Calligraphic style encoding: Given the set $Xi \subset X$ of $K = 15$ word images from the same writer w_i, the style encoder aims at extracting the calligraphic style attributes, i.e. slant, glyph shapes, stroke width, character roundness, ligatures etc. from the provided input samples. Specifically, the proposed network S learns a style latent space mapping, in which the obtained style representations $F_s = S(X_i)$ are disentangled from the actual textual contents of the images Xi. The VGG-19-BN architecture is used as the backbone of S. In order to process the input image set X_i, all the images are resized to have the same height h, padded to meet a maximum width w and concatenated channel-wise to end up with a single tensor $h \times w \times K$. If a human would be asked to write the the same word several times, slight involuntary variations appear. In order to imitate this phenomenon, randomly choosing permutations of the subset X_i will already produce such characteristic fluctuations. In addition, an additive noise $Z \sim N(0, 1)$ is applied to the output latent space to obtain a subtly distorted feature representation $F\hat{}_s = F_s + Z$.

C - Textual content encoding: The textual content network C is devoted to produce an encoding of the given text string t – the content to artificially write. The proposed architecture outputs content features at two different levels. Low-level features encode the different characters that form a word and their spatial position within the string. A subsequent broader representation aims at guiding the whole word consistency. Formally, let $t \in A^l$ be the input text string, character sequences shorter than l are padded with the empty symbol ε. Let us define a character-wise embedding function $e: A \rightarrow R^n$. The first step of the content encoding stage embeds with a linear layer each character $c \in t$, represented by a one-hot vector, into a character-wise latent space. Then, the architecture is divided into two branches.

Character-wise encoding: Let $g_1 : R^n \rightarrow R^m$ be a Multi-Layer Perceptron (MLP). Each embedded character e(c) is processed individually by g_1 and their results are later stacked together.

Global string encoding: Let $g_2 : R^{l \cdot n} \rightarrow R^{2p \cdot q}$ be another MLP aimed at obtaining a much broader and global string representation.

The character embeddings e(c) are concatenated into a large one-dimensional vector of size l·n that is then processed by g_2. Such global representation vector f_c will be then injected into the generator splitted into p pairs of parameters. Both functions g_1 and g_2 make use of three fully-connected layers with ReLU activation functions and batch normalization.

G - Generator: Let F be the combination of the calligraphic style attributes and the textual content information character-wise; and f_c the global textual encoding. The generator G is composed of two residual blocks using the AdaIN as the normalization layer. Then, four convolutional modules with nearest neighbor upsampling and a final tanh activation layer generates the output image $\bar{x}$. AdaIN is formally defined as:

$$\mathrm{AdaIN}(z, \alpha, \beta) = \alpha \left(\frac{z - \mu(z)}{\sigma(z)} \right) + \beta,$$

Equation 12. 2 – AdaIN formula

where z ∈ F, μ and σ are the channel-wise mean and standard deviations. The global content information is injected four times (p = 4) during the generative process by the AdaIN layers. Their parameters α and β are obtained by splitting f_c in four pairs. Hence, the generative network is defined as:

$$\bar{x} = H(t, X_i) = G(C(t), S(X_i)) = G\left(g_1\left(\hat{t}\right), g_2\left(\hat{t}\right), S(X_i)\right),$$

Equation 12. 3 – Broader representation of the generative model H

where tˆ= [e(c); ∀c ∈ t] is the encoding of the string t character by character.

Discriminator Loss: The discriminative loss only controls that the general visual appearance of the generated image looks realistic. However, it does not take into consideration neither the calligraphic styles nor the textual contents. This loss is formally defined as:

$$\mathcal{L}_d(H, D) = \mathbb{E}_{x\sim\mathcal{X}}\left[\log\left(D\left(x\right)\right)\right] + \mathbb{E}_{\bar{x}\sim\bar{\mathcal{X}}}\left[\log\left(1 - D\left(\bar{x}\right)\right)\right].$$

Equation 12. 4 – Discriminator loss

Given that:

D(x) – is the probability that the sample is real given a real example.

D(x~) – is the probility that the sample is real given a generated example.

Style Loss (Writer Classifier Loss): In that sense, the proposed style loss guides the generative network H to generate samples conditioned to a particular writing style by means of a writer classifier W. Given a handwritten word image, W tries to identify the writer wi ∈ W who produced it. The writer classifier W follows the same architecture of the discriminator D with a final classification MLP with the amount of writers in our training dataset. The classifier W is only optimized with real samples drawn from X , but it is used to guide the generation of the synthetic ones. Using the cross entropy loss, formally defined as:

$$\mathcal{L}_w(H, W) = -\mathbb{E}_{x\sim\{\mathcal{X},\bar{\mathcal{X}}\}}\left[\sum_{i=1}^{|\mathcal{W}|} w_i \log\left(\hat{w}_i\right)\right],$$

Equation 12. 5 – Style loss

Classic Cross entropy loss including all of the writers in the training data.

The probability distribution vector for an image x is: $\hat{w} = W(x)$

where w_i is the true identity of the writer in image x.

Content Loss (Word Recognizer): A handwritten word recognizer network R is used to guide the generator towards producing synthetic word images with specific textual content. This is a sequence-to-sequence model for handwritten word recognition to examine whether the produced images ¯x are actually decoded as the string t. The recognizer, depicted in the image above, consists of an encoder and a decoder coupled with an attention mechanism. Handwritten word images are processed by the encoder and high-level feature representations are obtained. A VGG-19-BN architecture followed by a two-layered Bi-directional Gated Recurrent Unit (B-GRU) is used as the encoder network. The decoder is a one-directional RNN that outputs character by character predictions at each time step.

The attention mechanism dynamically aligns context features from each time step of the decoder with high-level features from the encoder, hopefully corresponding to the next character to decode. The Kullback Leibler divergence loss is used as the recognition loss at each timestep. This is formally defined as:

$$\mathcal{L}_r\left(H,R\right)=-\mathbb{E}_{x\sim\{\mathcal{X},\bar{\mathcal{X}}\}}\left[\sum_{i=0}^{l}\sum_{j=0}^{|\mathcal{A}|}t_{i,j}\log\left(\frac{t_{i,j}}{\hat{t}_{i,j}}\right)\right],$$

Equation 12. 6 – Kullback Liebler divergence loss

where $\hat{t} = R(x)$; $\hat{t}_i$ being the i-th decoded character probability distribution by the word recognizer, $\hat{t}_{i,j}$ being the probability of j-th symbol in A for $\hat{t}_i$, and $t_{i,j}$ being the real probability corresponding to $\hat{t}_{i,j}$. The empty symbol ε is ignored in the loss computation; t_i denotes the i-th character on the input text t.

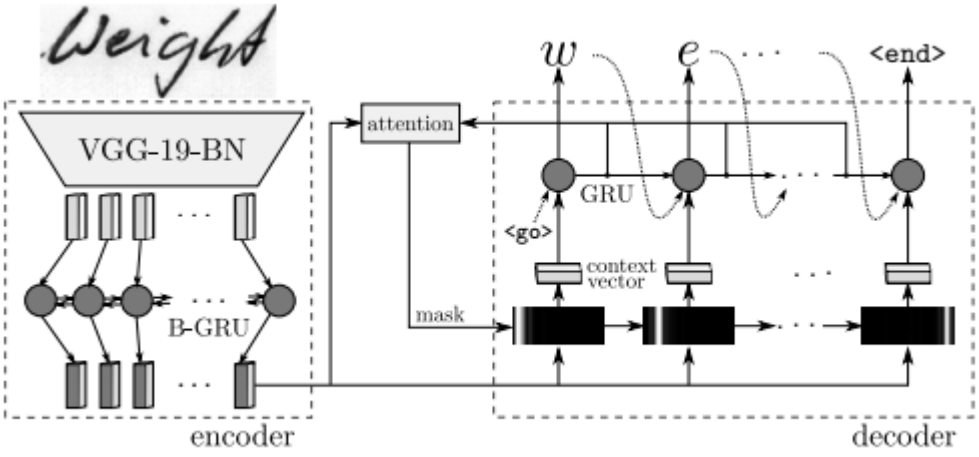


Figure 12. 3 – A content loss for word recognizer

General Loss for the Generator: Overall, the whole architecture is trained end to end with the combination of the three proposed loss functions:

$$\mathcal{L}(H,D,W,R)=\mathcal{L}_d(H,D)+\mathcal{L}_w(H,W)+\mathcal{L}_r(H,R),$$

$$\min_{H,W,R}\max_{D}\mathcal{L}(H,D,W,R).$$

Equation 12. 7 – General loss

The Algorithm below presents the training strategy that has been followed in this work. Γ(·) denotes the optimizer function. Note that the parameter optimization is performed in two steps. First, the discriminative loss is computed using both real and generated samples (line 3). The style and content losses are computed by just providing real data (line 4). Even though W and D are optimized using only real data and, therefore, they could be pre-trained independently from the generative network H, the authors of this paper [] obtained better results by initializing all the networks from scratch and jointly training them altogether. The network parameters Θ_D are optimized by gradient ascent following the GAN paradigm whereas the parameters Θ_W and Θ_R are optimized by gradient descent. Finally, the overall generator loss is computed following the equation above where only the generator parameters Θ_H are optimized (line 8).

Input: Input data $\{\mathcal{X}, \mathcal{Y}, \mathcal{W}\}$; alphabet $\mathcal{A}$; max training iterations T
Output: Networks parameters $\{\Theta_H, \Theta_D, \Theta_W, \Theta_R\}$.

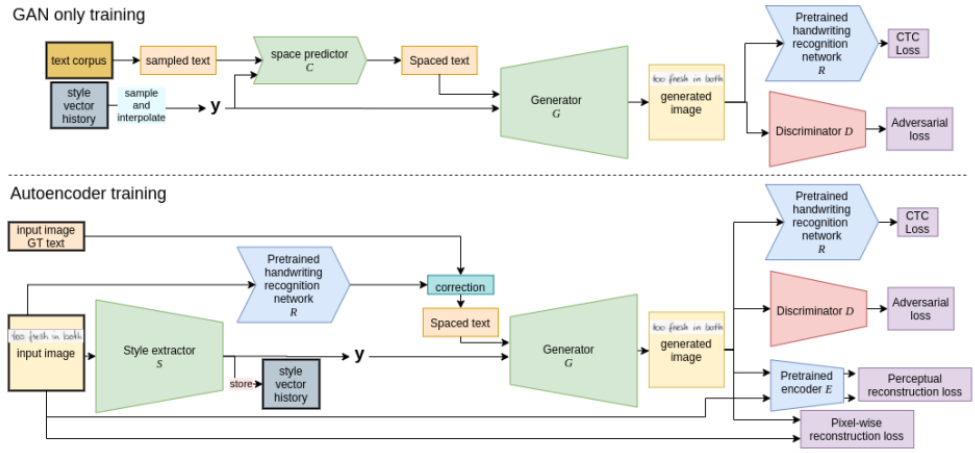
1: **repeat**
2: Get style and content mini-batches $\{X_i, w_i\}_{i=1}^{N_B}$ and $\{t^i\}_{i=1}^{N_B}$
3: $\mathcal{L}_d \leftarrow$ Eq. 4 ▷ Real and generated samples $x \sim \{\mathcal{X}, \bar{\mathcal{X}}\}$
4: $\mathcal{L}_{w,r} \leftarrow$ Eq. 5 + Eq. 6 ▷ Real samples $x \sim \mathcal{X}$
5: $\Theta_D \leftarrow \Theta_D + \Gamma(\nabla_{\Theta_D} \mathcal{L}_d)$
6: $\Theta_{W,R} \leftarrow \Theta_{W,R} - \Gamma(\nabla_{\Theta_{W,R}} \mathcal{L}_{w,d})$
7: $\mathcal{L} \leftarrow$ Eq. 7 ▷ Generated samples $x \sim \bar{\mathcal{X}}$
8: $\Theta_H \leftarrow \Theta_H - \Gamma(\nabla_{\Theta_H} \mathcal{L})$
9: **until** Max training iterations T

Figure 12. 4 – GAN training algorithm (Handwriting word generation)

12.1.2. **The Handwriting Line Generation model:**

After using the GANWriting model we encountered a newer paper on this topic with some similar ideas [14] and this is the model used in the final product for word and line generation. The handwriting line generation process has three inputs: content, style, and noise. Content is the desired text. Style is the unique way a writer forms characters using a particular physical medium and writing instrument. Noise is the natural variation of individual handwriting, even when writing the same content in the same style. The model was trained with GAN, reconstruction, perceptual, and text recognition losses. The image shows an overview of the training process, including six networks:

1. A generator network G to produce images from spaced text, a style vector, and noise.
2. A style extractor network S, that produces a style vector from an image and the recognition predictions.
3. A spacing network C, which predicts the horizontal text spacing based on the style vector.
4. A patch-based convolutional discriminator D.
5. A pre-trained handwriting recognition network R to encourage image legibility and correct content.
6. A pre-trained encoder E, to compute a perceptual loss.



Overview of our method. To generate an image, we take input text and style vector **y**. Text is spaced by spacer C using **y**. This spaced text and **y** are passed to the generator G. We use both fully GAN training steps (top) and autoencoder based training steps (bottom).

Figure 12. 5 – Overview of GAN (Handwriting line generation)

Generator G and Discriminator D: G is based on StyleGAN but differs in architecture and receives the 1D spaced text as input with the style vector concatenated at each spatial position. Spaced text is a one-hot encoding of the target text with additional blank characters and repeated characters, which encode the spacing information. This informs horizontal character placement and was key to training the model successfully. The network blocks consist of a convolutional layer, additive noise, ReLU activation, and AdaIN, which uses the style vector to determine feature map statistics. To increase resolution, nearest-neighbor upsampling was used followed by a convolution and blurring operation. Most upsampling is in only the vertical dimension because the spaced text input is already wide. Our discriminator D must be able to handle variable-sized inputs, so it is a fully convolutional, multi-resolution patch-based discriminator that the authors of this paper trained with a hinge loss.

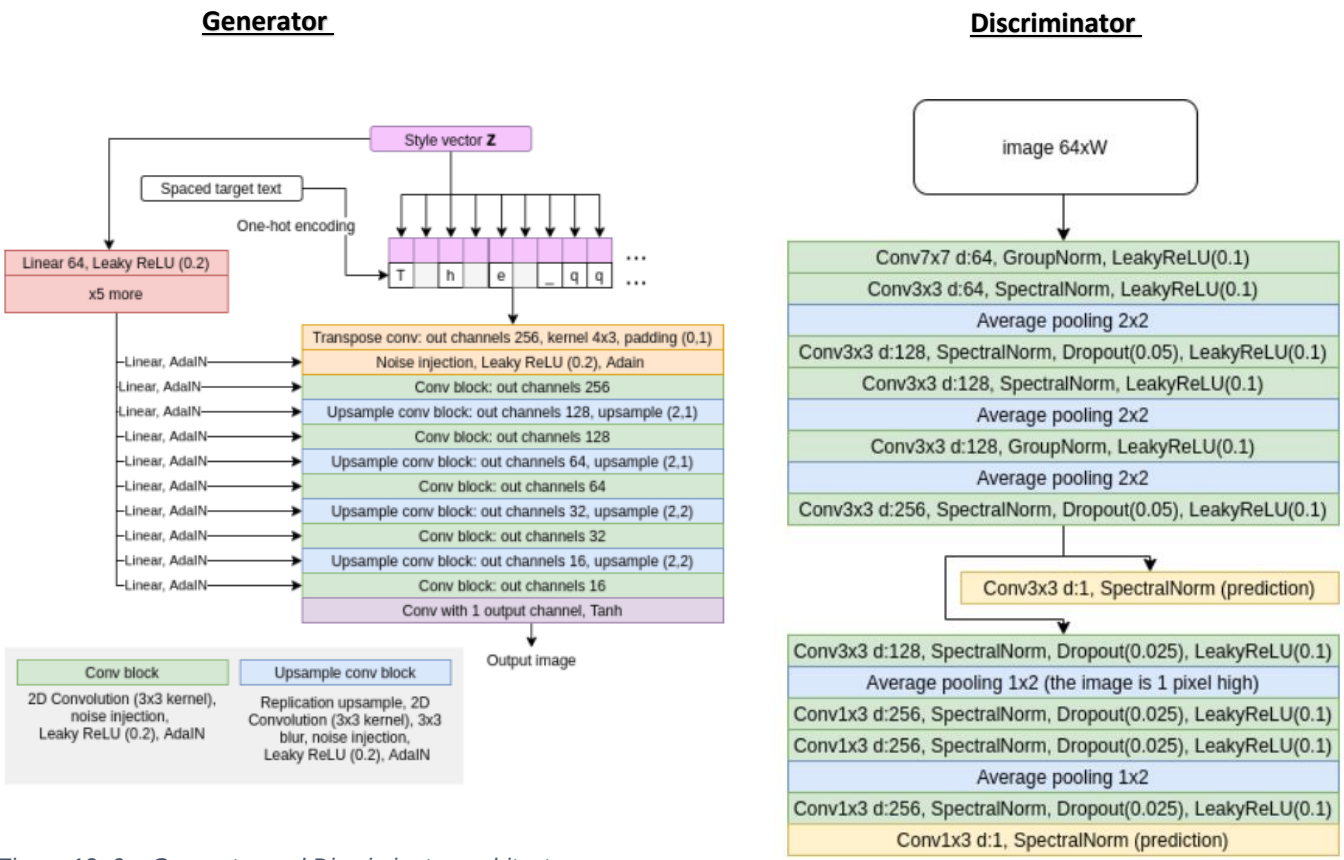
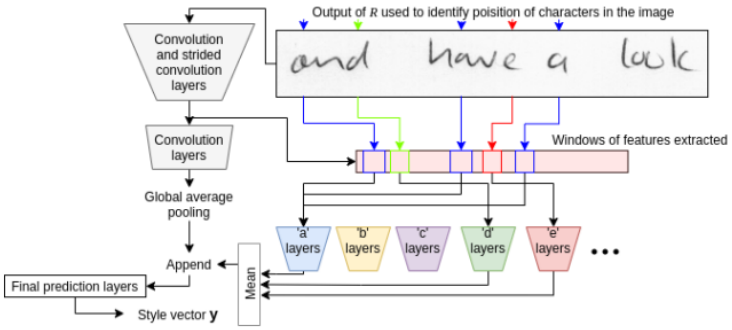
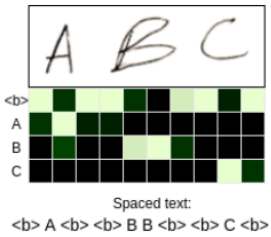


Figure 12. 6 – Generator and Discriminator architecture

Style Extractor S: S inputs the image and the output of R on the image to produce a style vector. First, it uses a convolutional network to extract a 1D (horizontal) sequence of features. Then the recognition result is used to roughly localize each recognized character in the feature sequence. For each predicted character the next steps were done: Cropping the feature sequence with a window size of 5 (roughly 40 pixels, an area slightly larger than most characters in the IAM dataset) centered on the character and then pass each window through character specific layers to extract character features. Features from all instances of all characters are averaged, weighted by R's predicted confidence for each instance, giving a final character feature vector. To obtain global style features, passing the entire feature sequence through 1D convolutional layers and performing global average pooling is necessary. This is appended to the character feature vector, from which fully connected layers predict the final style vector of dimension 128.



Style Extractor S architecture with character-specific heads to process feature windows from detected character locations.



Example output of R (lighter is higher prob.) and corresponding spaced text.

Figure 12. 7 – Style extractor architecture

It leverages the output of R both as additional input and to (roughly) locate characters. The locations are used to crop features to pass to character specific layers (the learn to extract features for one character).

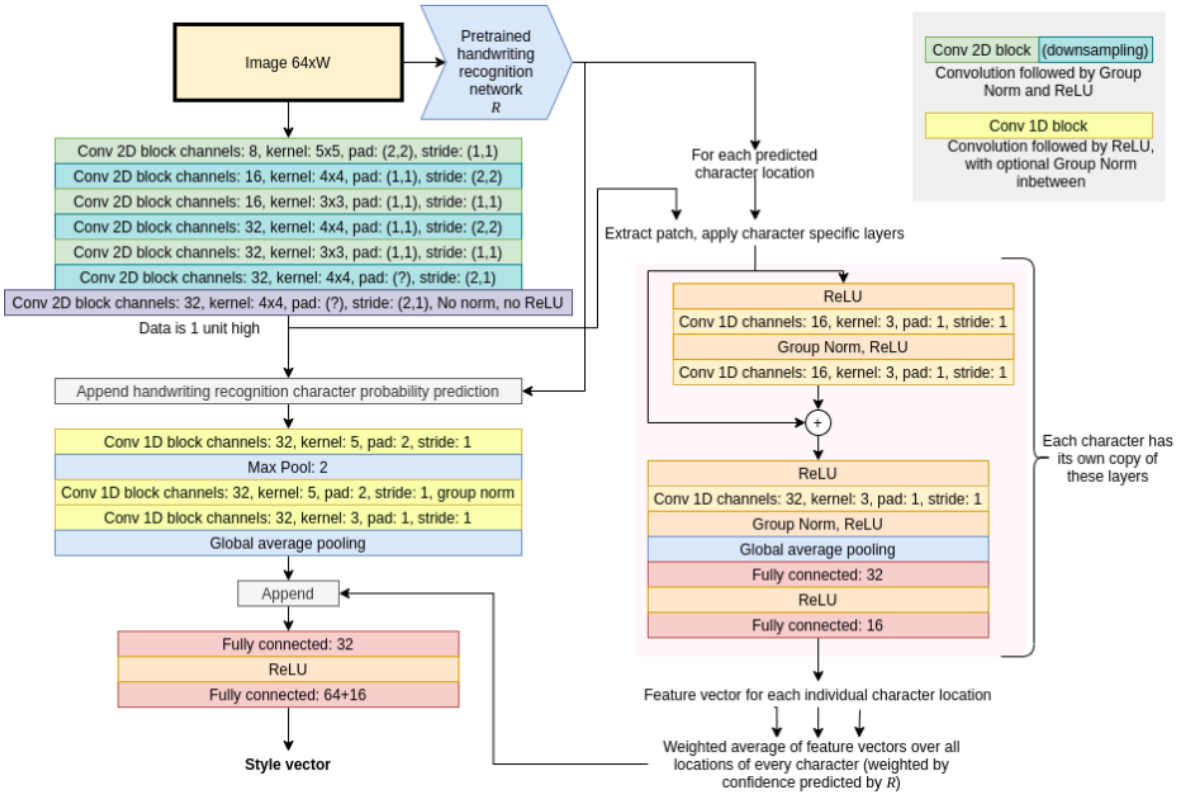


Figure 12. 8 – Handwriting line generation flow architecture

Spaced Text and Spacing Network C: In the paper, they found spaced text essential for training with a reconstruction loss. Without it, G has difficulty achieving horizontal alignment with the input image and fails to train. Spaced text can be derived for a particular image from the output of R or predicted directly by C for a novel style. Width and spacing are encoded using repeated characters and blank symbols <b>. Dataset spaced text is obtained by taking the predicted character at each horizontal position from the output of R on a dataset image, keeping blanks and repeated characters (artifacts typically removed when decoding the output of a Connectionist Temporal Classification – CTC trained model). The authors of this paper corrected the recognition errors in the dataset spaced text using the ground truth text. C is a 1D convolutional network that consumes one-hot encoded target text with the style vector concatenated to each position. For each character c_i, C predicts how many blanks precedes c_i and how many times c_i is repeated in the spaced text. Multiple blanks are then appended to the output. C is trained to imitate the dataset spaced text using an MSE loss.

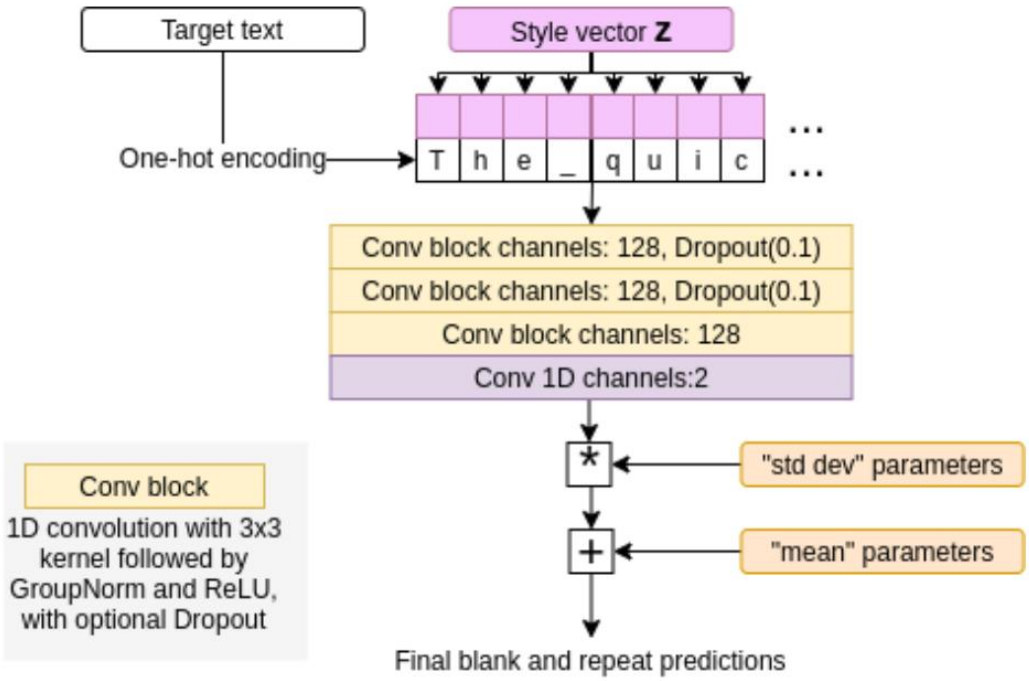


Figure 12. 9 – Spacing architecture

Handwriting Recognition Network R: R is a pre-trained handwriting recognition network that encourages generated images to contain the specified text by applying the Connectionist Temporal Classification (CTC) loss. R's weights are frozen so the gradient merely flows through R to supervise G. While state-of-the-art handwriting recognition methods use CNN-RNNs, better results were shown with R as a fully convolutional network. RNNs have arbitrarily large context windows and may predict characters based on linguistic context instead of visual character shapes. In contrast, R only uses local visual features for character predictions and therefore provides better feedback for generating characters.

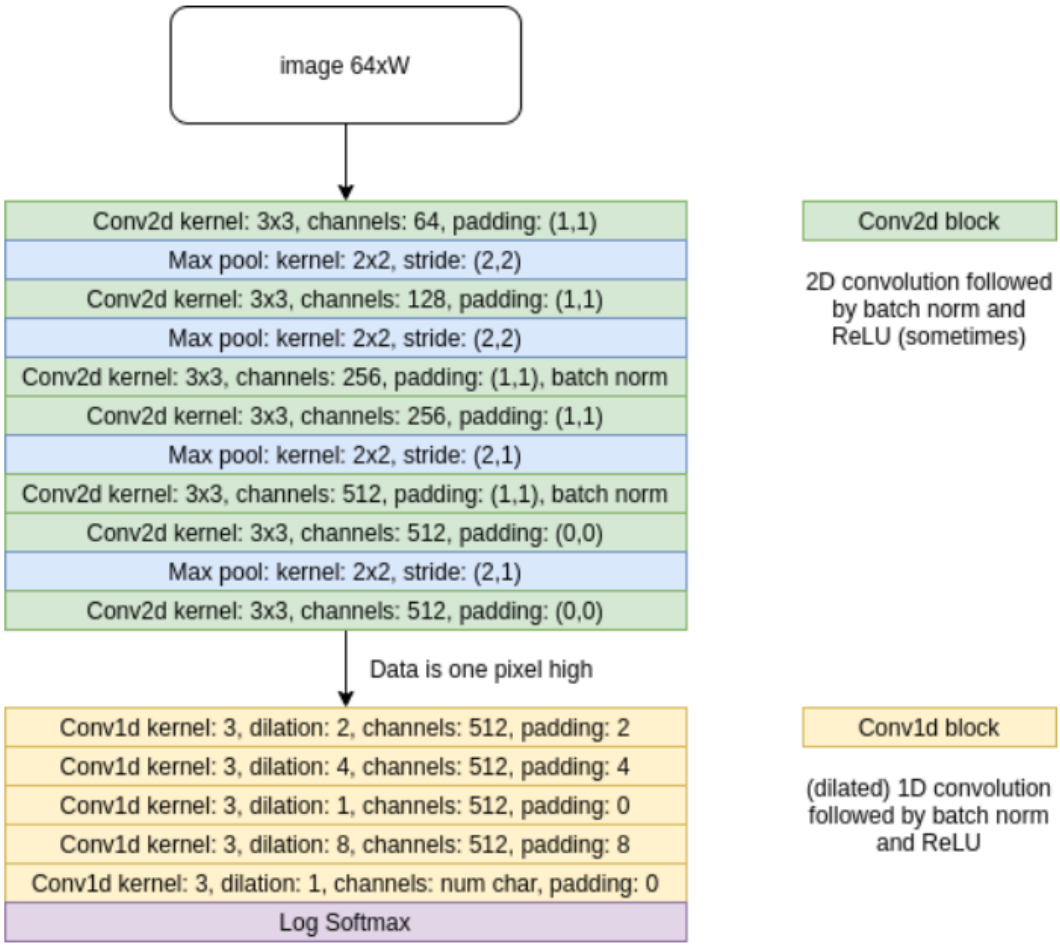


Figure 12. 10 – Handwriting line recognition architecture

Encoder Network E: E provides features for the perceptual loss. Traditional methods for computing perceptual loss use features from pre-trained image classification networks. However, images of handwriting are different from natural images, so a different approach was needed. E is a fully convolutional network that collapses the image to a one-dimensional feature series capturing visual and semantic features. E is trained both as an autoencoder with a decoder and L_1 reconstruction loss, and as handwriting recognizing network with CTC loss.

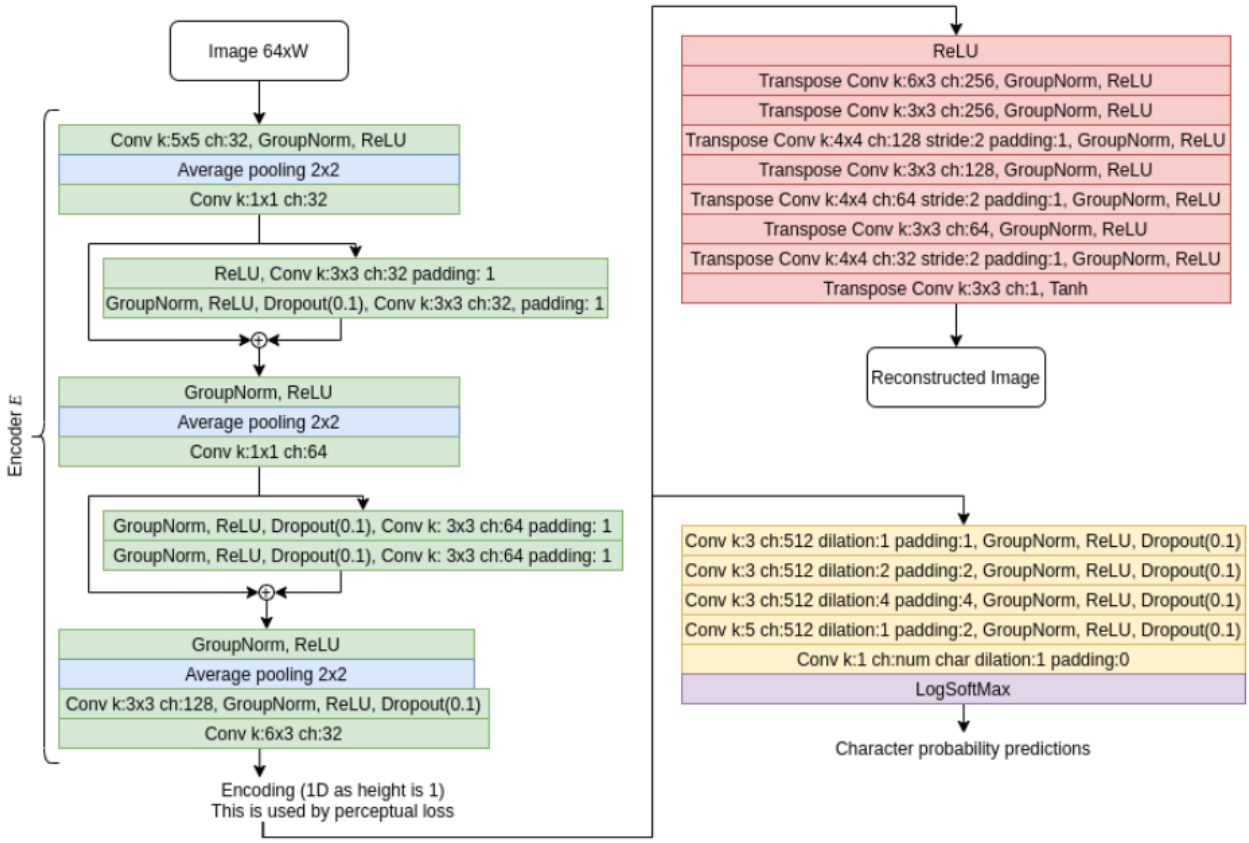


Figure 12. 11 – Encoder network architecture

Training/Losses: The generation has three objectives: legible handwriting matching the target text, realistic handwriting that appears to be a human's, and handwriting style that mimics example images. Each of these is achieved primarily through the respective losses: CTC loss backpropagating through R, adversarial loss, and reconstruction losses (pixel and perceptual).

Additionally, MSE is used to train the spacing network C, and hinge loss is used to train D. When using multiple loss terms, balancing them is crucial. This was done by improving the gradient-balancing method. Without balancing, training failed due to exploding gradients or failed to converge. Stable hyper-parameters possibly exist, but gradient balancing easily solved the problem. The authors of this paper balanced the gradients from the Connectionist Temporal Classification (CTC), adversarial, and the reconstruction losses. The two reconstruction losses have equal weight and are summed.

The CTC loss gradient is normalized to have the same mean and standard deviation as the adversarial loss gradient. However, this does not preserve the sign of the CTC gradient, so normalizing the gradients was done to have the same mean magnitude (per layer). This additionally allows balancing multiple gradients. Totally equal contributions may not be desirable and can be adjusted by multiplicative weights on each gradient after normalization. For gradient normalization, the authors always used the gradient magnitude of the reconstruction loss.

The proposed architecture uses the following subsystems:

1. Spacing submodule: A style is extracted from two dataset images by the same author using the style extractor S, and C predicts the spacing. The MSE loss between the prediction and dataset spaced text updates both C and S.
2. Discriminator: To update D the authors sampled styles by interpolating/extrapolating styles sampled from a running window history of the 100 most recently extracted styles (during Spacing or Autoencoder steps). Extrapolation is kept within 0.5 of the distance between the two styles and is sampled uniformly from that range.
3. GAN-only: This follows standard GAN training while including the handwriting recognition supervision. It does not update the model but saves the gradient information. It samples styles like the Discriminator step.
4. Autoencoder: Pairs of images by the same author are concatenated width-wise, and a single style vector is extracted for both of them. Then each image is individually reconstructed using that style. The authors compute the reconstruction, adversarial, and handwriting recognition losses with the reconstructed images. The gradients from this step and the GAN-only step are balanced. Both S and G are updated.

We now define the loss functions used in training our model and formalize the gradient balancing. Let I be a dataset image, t_I its corresponding text, and c_I its corresponding dataset spaced text. Let I_0 be the concatenation of I and another image by the same author. Let y_s be a sampled style, obtained by sampling two stored style vectors from the running window history and interpolating/extrapolating a point on the line between them. Let t_s be text sampled from a text corpus. CTC is a Connectionist Temporal Classification loss, and D is the discriminative network.

$$\text{Spacing network loss } l_c = MSE(C(t_I, S(I')), c_I) \quad (1)$$

$$\text{Discriminator loss } l_d = max(1 - D(I), 0) + max(1 + D(G(C(t_s, y_s), y_s)), 0) \quad (2)$$

$$\text{Generated image adversarial loss } l_{adv,g} = -D(G(C(t_s, y_s), y_s)) \quad (3)$$

$$\text{Generated image recognition loss } l_{rec,g} = \text{CTC}(R(G(C(t_s, y_s), y_s), t_s) \quad (4)$$

$$\text{Reconstructed image adversarial loss } l_{adv,r} = -D(G(c_I, S(I'))) \quad (5)$$

$$\text{Reconstructed image recognition loss } l_{rec,r} = \text{CTC}(R(G(c_I, S(I')), t_I)) \quad (6)$$

$$\text{Combined reconstruction loss } l_{auto,r} = L1(G(c_I, S(I')), I) + L1(E(G(c_I, S(I'))), E(I)) \quad (7)$$

12.2. Frequency / Image Domain Architectures

The following figure is a modified version of Figure 9. 1 – Block Diagram, showing the section we are located in.

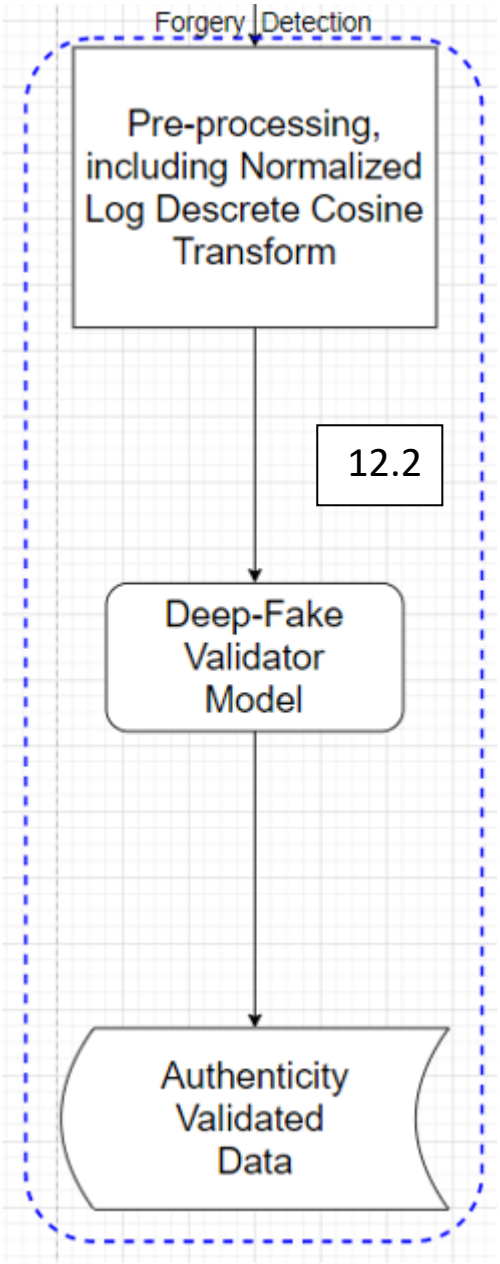


Figure 12. 12 - Deep-Fake validator model block flow

In order to classify handwritten word images within a document as generated/fake or real ones, most of the concept was already dealt with prior to the insertion of the actual images into the neural network. To do so, we compared two models with the same architecture that were trained on different domains: One model was trained on the frequency images dataset and the other on the image space.

To transform the dataset into the frequency domain, we used the DCT (Discrete Cosine Transform) operation. The coefficients drop very quickly in magnitude when moving to high frequencies, then it is necessary to pass the DCT results through log transformation and normalization (scaling the data by the mean and standard deviation) operations. The technique is based on finding unique artifacts in the frequency domain due to the GANs upscaling operation to the generated images. Finding the artifacts is a generally easy problem (given that they exist) and this is why we decided to work with the Resnet50 architecture in order to efficiently classify these images.

In contrast to [11], we did not notice unique artifacts within the generated images from the IAM dataset distribution. To test [11] on handwriting images, we trained the same Resnet50 architecture on both of the domains. After the training process, we did not recognize better results using the frequency domain dataset than the image space one. Hence, we chose to reduce the process of the DCT transformation in order to spare some functionalities.

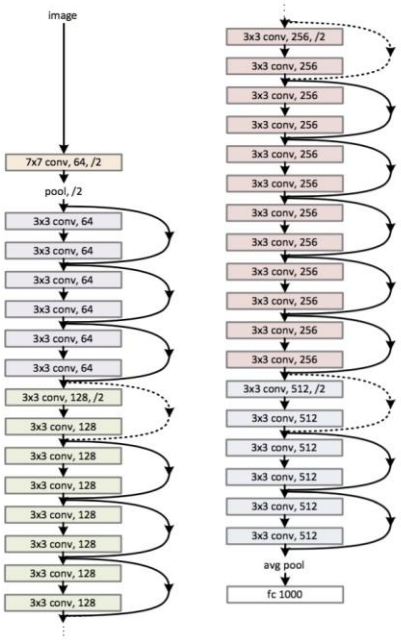


Figure 12. 13 – Fake Frequency/Image domain Deep validation architecture

12.3. Writer Classifier Architectures

We purpose several different models in order to cope with the problem of identifying writer given his handwriting font style. The first architectures rely on the notion of the FragNet [9].

The following figure is a modified version of Figure 9. 1 – Block Diagram, showing the section we are located in.

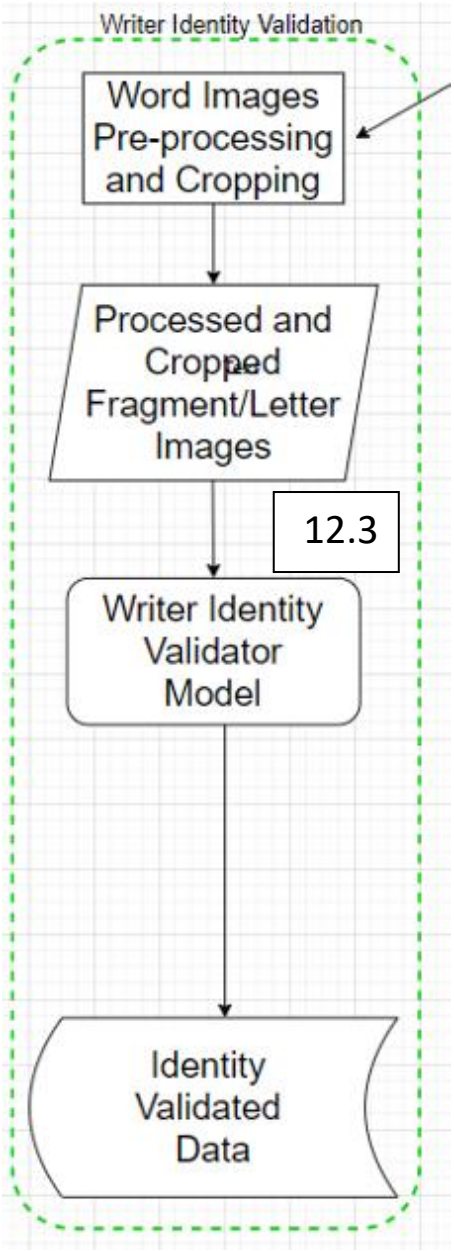


Figure 12. 14 – Writer Identity validation block flow

Using the FragNet model

The generic architecture of FragNet has a feature pyramid and a fragment pathway, which are fused by lateral connections. The Image above illustrates the concept of the FragNet network.

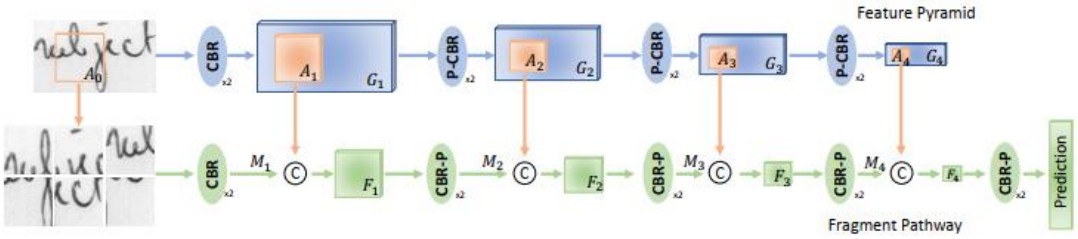


Figure 12. 15 – FragNet architecture

1) **Feature pyramid:** The feature pyramid (the blue pathway in the image) is the traditional convolutional neural network which computes the feature maps hierarchy in different scales. It contains four (P)-CBR blocks, in which P is the max-pooling layer, C is the convolutional layer, B is the batch normalization layer and R is the Rectified Linear Units (ReLU) layer. The kernel size of each convolutional layer is 3 × 3 and the stride step is 1. The max-pooling with a kernel size of 2 × 2 and stride step of 2 is applied after every two convolutional layers in order to reduce the spatial size of the feature maps and obtain translation invariance.

The input of the feature pyramid is the whole input image, such as the word image in the paper. Four feature maps, G_i, i = 1, 2, 3, 4 as shown in the image, are obtained in the feature pyramid after every two convolutional layers.

$$G_i^n = \circledast(G_{i-1}^m, W_i^{m\times 3\times 3\times n}, W_i^b)$$

Equation 12. 8 – Fragnet feature map

where $\circledast$ denotes of the P-CBR blocks, $G^1{}_0$ is the input image with the channel size of 1 and $W^{m\times 3\times 3\times n}$ is the kernel with the size of (m × 3 × 3 × n) in convolutional layers (m is the channel size of the input feature map and n is the channel size of the output feature map and W_i^b is the set of parameters in the batch normalization

layer. The channel numbers of four feature maps are set to [64; 128; 256; 512], respectively.

2) **Fragment pathway:** Fragments are segmented from the input image and feature maps on the feature pyramid. The fragment defined as Ai which is segmented on the i-th feature map Gi in the feature pyramid by:

$$A_i = \mathrm{Crop}\big(G_i, \theta = (x, y, h, w)\big)$$

Equation 12. 9 – Fragment cropping

Specially, A_0 is the fragment segmented from the input image G_0.
The feature map of the fragment is denoted as F_i, which is defined as:

$$M_i^n = \circledast\big(F_{i-1}^m, W_i^{m\times 3\times 3\times n}, W_i^b\big)$$
$$F_i = [M_i, A_i]$$

Equation 12. 10 – Feature map of the fragments

where M_i is computed by $\circledast$ representing two convolutional layers C, followed by the Batch Normalization layer B and the ReLU layer R. The feature map of the fragment is denoted as F_i, which is the concatenation of M_i and A_i. Note that $F_0 = A_0$.

Specifically, the fragment pathway fuses the information from two directions: one is from the previous convolutional layer in the fragment pathway and one is from the fragment segmented from the corresponding convolutional layer in the feature pyramid. Similar to DenseNet, the concatenate operation is used for merging these two different information. The fragments are segmented in different feature maps (G_1; G_2; G_3; G_4), including the input image G_0.

In order to keep the segmented fragment consistency in the spatial space, the fragment is cropped in the same space with respect to the input image. For example, assuming that the segment position is p_i(x; y) with the size of (h; w) in the feature map G_i of the feature pyramid, the segment position in the next feature scale G_i+1 is p_i+1(x=2; y=2) with the size of (h=2; w=2) when the stride of the max-pooling is 2.

The predictions from all fragments in one input image are averaged to infer the word-level writer evidence, which is inspired by the bagging ensemble learning.

One important parameter for the fragment segmentation is the size of fragment corresponding to the input image. A square window with the size of q × q was used to cut the fragment in the input image. The resulting architecture is denoted as FragNet-q with the fragment size of q × q in the input image. For each fragment, the loss is defined as the cross entropy loss between the prediction and the ground-truth, which is defined as:

$$L_i = -\sum_{i}^{M} g_i \cdot \log(p_i)$$

Equation 12. 11 – Cross Entropy for each fragment

Until month April, 2021 the FragNet [9] paper did not include any code implementation, hence, we decided to do so ourselves, before its original implementation was published. In addition to the architecture described in [9], we implemented two more extensions of the classic FragNet model. The idea behind these extensions was the overlapping of the handwriting analysis scope with the scope of NLP, which considers the time-series between parts of the text. Our proposed implementations:

1. Attention with a general concept of the FragNet model (SimpleFragNet): This implementation uses the basic concepts shown in the paper including the extraction of sub-images of a pre-determined moving window with four timestamps taken from the input word images (with shapes of 64 x 256). The partition of the words into timestamps is equivalent to a fragment window of dimention 64 x 64 (q=64). Followed by forwarding these fragments into a feature extractor network, the input also goes through a LSTM with an attention mechanism. This architecture strives to be as simplistic as possible while combining the important parts of the paper with the capabilities of the attention mechanism. One difference from the classic FragNet implementation is that here we chose to extract one feature map from the input as opposed to the feature pathway in the paper including two feature extraction paths.

2. Classic FragNet Implementation: This architecture was made to try and replicate the papers instructions and results, The generic architecture of FragNet has a feature pyramid and fragment pathways, which are fused by lateral connections. The paper review above explains the concept of the FragNet network in detail.

3. Attention LSTM FragNet Implementation: This implementation includes all of the architecture presented in the paper combined with a full attention mechanism. The goal here is to use everything we have learned in the course to try to innovate and strengthen the classic FragNet's results using both the positive contributions of each methodology in one model.

As for the 1st and the 3rd models, we chose to include an attention mechanism because we concluded that in the field of handwriting recognition, focusing on specific details in the scripts is important for distinguishing between different styles and their low-level features. The reason we included LSTM layers in the first place is that the positions of the letters in the image domain and the way that they interact with each other influence the way that the whole word is being written/classified and therefore, there is a great value to observe the data with different timestamps.

12.3.1. **Resnet50**

Another architecture that was tested in our project is the known Resnet50 backbone with a different output fully-connected layer. For example, when tested against two classes, there will be two output neurons (the player's handwriting style and everyone else's style).

12.3.2. **SigNet / Resnet50 Siamese**

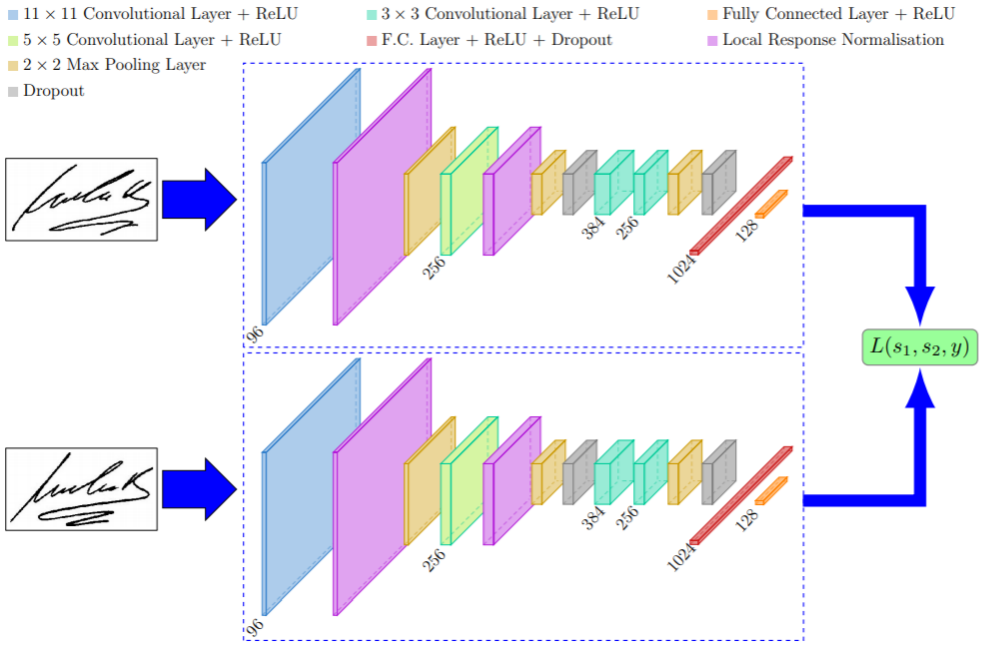


Figure 12. 16 – Signet example figure

The SigNet architecture is as follows: The input layer, such as the 11×11 convolution layer with ReLU, is shown in blue, whereas all the 3×3 and 5×5 convolution layers are depicted in cyan and green respectively. All the local response normalization layers are shown in magenta, all the max pooling layers are depicted in brick color and the dropout layers are exhibited in gray. The last orange block represents the high-level feature output from the constituting CNNs, which are joined by the loss function. Despite the backbone architecture above, we decided to mostly use a resnet architecture (as shown in the section of deepfake detection above) with 50 layers as the Siamese twin backbones, because we were more satisfied by its results, the architecture is as follows:

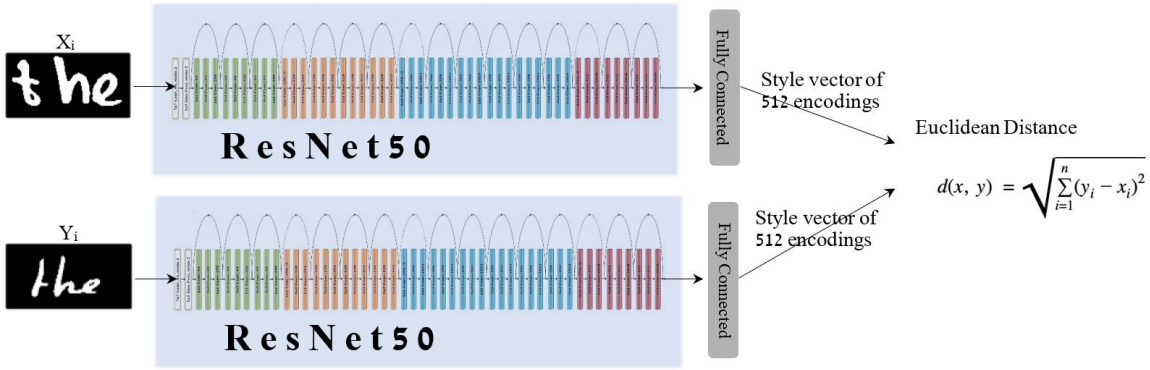


Figure 12. 17 – Siamese network distance diagram

In order to train the network, the Contrastive Loss function, is defined as:

$$L(s_1, s_2, y) = \alpha(1 - y)D_w^2 + \beta y \max(0, m - D_w)^2$$

Equation 12. 12 – Contrastive loss function

where s_1 is one of the target handwritten word images that are compatible with the calligraphic style to learn, and s_2 is a general handwritten word image, could be compatible either with the target calligraphic style or with a different one. y is a binary indicator function denoting whether the two samples belong to the same class or not, α and β are two constants and m is the margin equal to 1 in our case. $D_w = ||f(s_1,w_1) - f(s_2,w_2)||_2$ is the Euclidean distance computed in the embedded feature space, f is an embedding function that maps an image to a real vector space through CNN, and w_1, w_2 are the learned weights for a particular layer of the underlying network. Unlike conventional approaches that assign binary similarity labels to pairs, Siamese network aims to bring the output feature vectors closer for input pairs that are labelled as similar and push the feature vectors away if the input pairs are dissimilar. Each of the branches of the Siamese network can be seen as a function that embeds the input image into a space. Due to the loss function selected, this space will have the property that images of the same class (the target handwritten style) will be closer to each other than images of different classes (different calligraphic styles). Both branches are joined together by a layer that computes the Euclidean distance between the two points in the embedded space. Then, in order to decide if two images belong to the similar or a dissimilar class one needs to determine a threshold value on the distance.

12.4. Additional Algorithms for the product

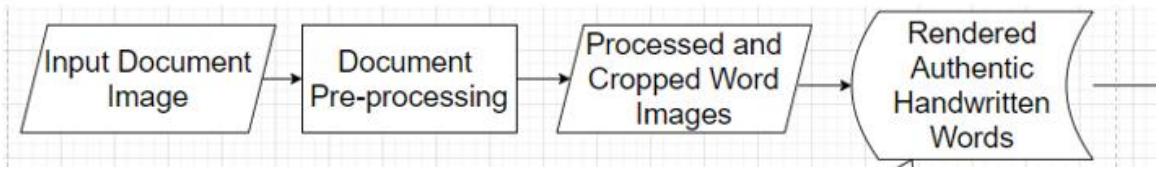


Figure 12. 18 – Document pre-processing block flow

These are some key algorithms that were designed to help fulfil our requirements:

12.4.1. **Scanning a handwritten document:**

This algorithms goal is to segment every word from a handwritten document and set each one with back-end attributes such as transcription, position, dimentions, spacing, and the word image itself, and place it in the word array of a custom Document object in python in order to allow for future actions to be executed on the document with ease.

12.4.1.1. Feed the document through an out-sourced microsoft OCR API that segments each word and line with a bounding box.

12.4.1.2. For each bounding box: crop the word inside, resize it to a fixed size, and add the sub-image to the array, add the label into a corresponding array. After this algorithm – we will have arrays with cropped images and their corresponding labels.

12.4.1.3. For each bounding box that represents a word: calculate the original spacing between adjacent images and store every forward space inside of the image behind it.

12.4.1.4. For each bounding box that represents a whole line: calculate the length of new lines in the original document and add them to the Document object in order to preserve the original lines spacing in future changes.

12.4.2. **Learn the writer's features via deep learning:**

12.4.2.1. Find the optimal threshold which separates a specific calligraphic handwriting style features of a unique writer from other styles, using a brute-force search, given the writer's handwritten words, as a prior-knowledge.

12.4.2.2. Classify each handwritten word within the document, using the optimal threshold, as a binary classification separator between the given writer's style to others.

12.4.2.3. Ensemble each of the predictions (Average) to a single prediction and render the result on the user's display.

12.4.3. **Feed the newly scanned images to a Deep Fake Detection network:**

For each image object in the custom Document class:

12.4.3.1 Feed its image to the Deep fake detector (using the image domain model)

12.4.3.2 Mark the result on the each image with a red rectangle for fake or a green rectangle for authentic.

12.4.3.3 Add that result and compute the mean probability of the document being fake, and the mean probability of the document being authentic.

12.4.4. **Generate handwritten text in the desired location:**

This algorithms goal is to use the handwriting word generator to generate a desired word or line and to insert it in the chosen word index in the document.

12.4.4.1 Generate a word or line image according to the transcription and a given writer style using the GAN[14], record its total length including spacing.

12.4.4.2 Create a custom word python object for each new generated word that includes attributes such as dimentions, image, transcription, and spacing.

12.4.4.3 Insert each generated word in the word array at the chosen word index and perform algorithm 12.4.5 (draw the content of the document) that will re-order the document in an online fashion.

12.4.5. **Draw the content of the word array into a document image:**

This algorithms goal is to use the content of the Document objects word array in order to create an image containing these words, organized as written in their attributes.

Note: each word object in the word array is already sorted into its correct position in terms of the order in which the words should appear on the screen.

12.4.5.1 While keeping track of the current cursors moving position, for each word in the word array:

- 12.4.5.1.1 Draw the word image into the cursors current position
- 12.4.5.1.2 Update the cursors current x position including spacing
- 12.4.5.1.3 If the cursors position enters the documents margin: reset the x position and add a pre-determined new line length to the y position of the cursor.
- 12.4.5.1.4 If the cursors y position is bigger than the document height: break the loop and signal that the document is full.

13. Product

Our product in this project is a beta version of a mobile application called "Penalyzer", which allows people to scan handwritten documents, edit them, and analyze them in terms of authenticity or writer identity validation.

The main flow of the application requires the player to upload an image of a handwritten document and choose between three main options:

(1) Edit document

(2) Validate document authenticity

(3) Validate author identity.

After uploading a document to the application, it is immediately sent to the server for processing, enhancement, and segregation of word images, this includes tracking the positions of each word and other types of features, such as spacing and font size.

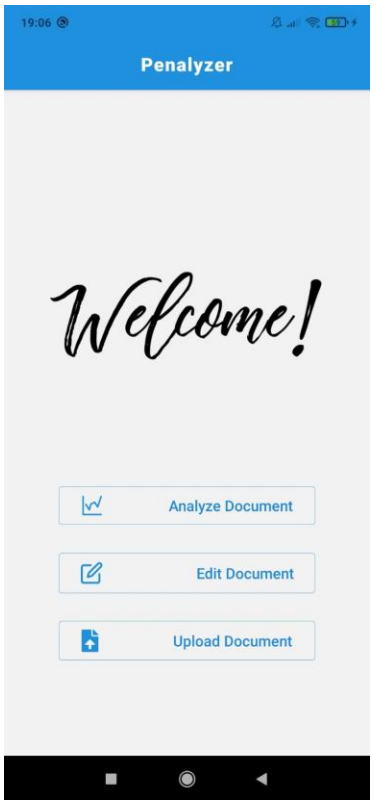
Depending on the functionality chosen by the player, the corresponding command is sent to the back-end, where the chosen function will occur, each of the functionalities requires using one of our deep-learning models in order to fulfil each request:

The addition of generated text will incur a process of extracting the existing style from the current document and feeding that extraction to our generative adversarial network with the required text to generate. After the generation is complete, the sentence or words generated will be placed at the chosen position (by the player) and if there are any following words they will be pushed forward to make room for the newly added text.

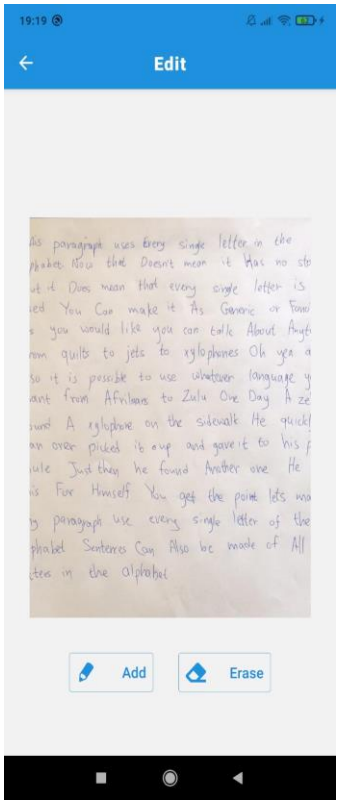
The validation process of a document's authenticity happens for each word by itself, each word is sent to our Deep-Fake validator model to determine whether that word was written by a human or generated in automated means, and the results are shown to the player on a per-word basis in the returned output. This means that the player would not only know if the document is fraudulent or not in general but would additionally be aware of specific suspicious areas from within the document.

In order to test the writer validator on our handwritten words, and not on the IAM's, we chose to use the Siamese architecture which is trained to distinguish between handwriting styles regardless (as much as possible) of any features that refer to the ethnicity of the writers. The validation procedure of a writer's identity regarding a document is also calculated on a per-word basis. Given reference samples of the content 'the' and 'and' – positive word images and negative ones (written by other writers), we search the distance thresholds that optimally separate two samples with different calligraphic styles, for each of the text labels. Then, a smart algorithm shall extract words in an inference document and will feed pairs of words – positive word images against inference ones to the writer identity validator, using the calculated thresholds. The model will then extract writer-style features from the word images, and determine the distance between them. If the distance would be greater than the compatible threshold, the model will be prone to suggest that the styles are too different, and otherwise, the same. An average of the model decisions on each pair of words will determine if the given document is a match with the reference document, or not, and the player will receive the corresponding answer as a similarity percentage.

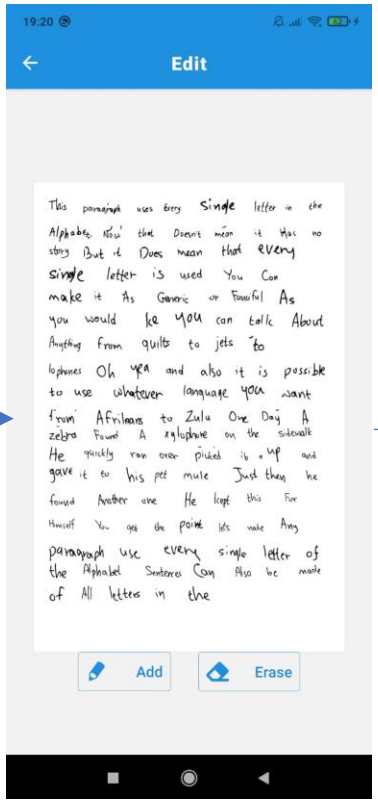
Home Page



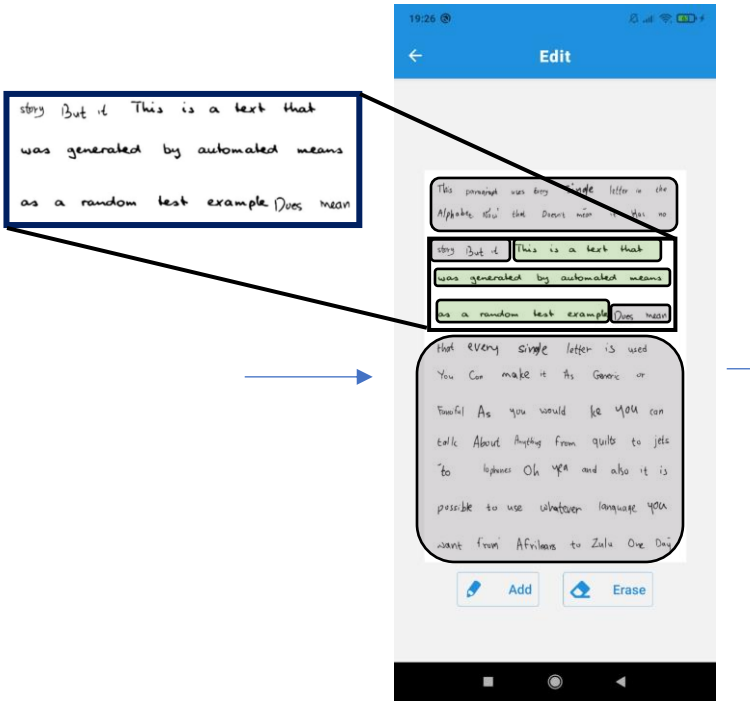
Upload Document



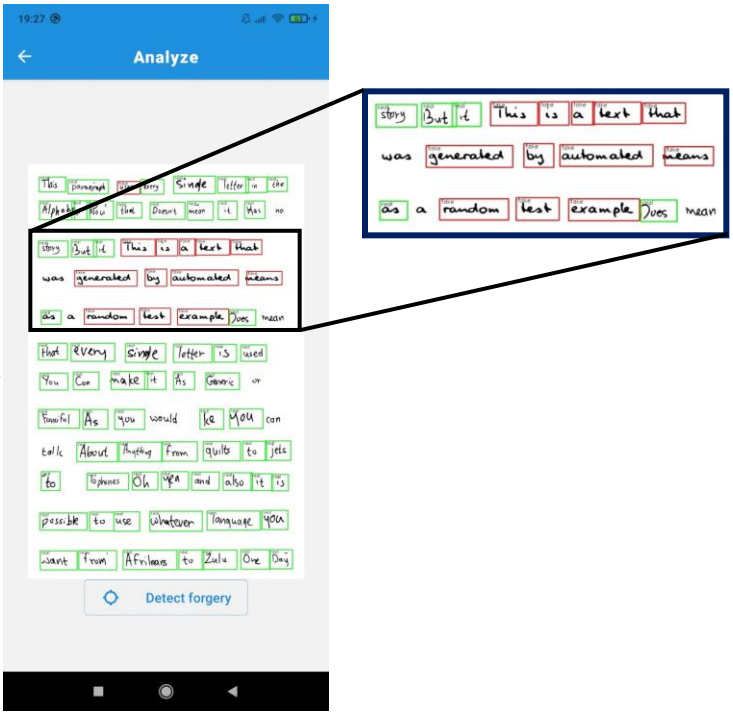
Pre-processing



Edit (Insert words)



Detect Forgery



14. Results and Discussion

After training the varied architectures, we evaluated them on a separate test of handwritten word images. The summary of our results is presented as follows. We uniquely discuss each of the functionalities of our system.

14.2. Deep-fake Detection

When we first profoundly tested this subsystem, using different hyper-parameters and different threshold values, we realized that our model was exposed to some adversarial attacks.

Given the delicate nature expressed in the field of deep-fake detection, both our Image Domain model and our Frequency Domain model have their weak points when it comes to Adversarial Attacks.

As explained in detail before, the accuracy and recall scores of both variations of our models highly depend upon the existence of upsampling artifacts in the images, which are generated exclusively by a GAN architecture trying to mimic the handwriting of a genuine individual (or any image in general).

One weakness that was found in these models can be exploited quite easily by performing minor changes to the images which the human eye can barely spot. If we attempt to clean the images from the previously mentioned artifacts, usually positioned in a small circumference around the characters in the word image, without ruining the appearance of the words, our models would more likely infer that these images may be authentic – due to the fact that to the models, a lack of these upsampling artifacts suggest that an image is real.

Knowing this weakness has motivated us to train the models again, this time using various intensities of an artifact destroying augmentation that changes brightness and removes gray noise above a random threshold. This has in fact helped the models be more resilient to these types of attacks. Although when an augmentation that abolishes 100% of the artifacts is applied even after that type of training, it will still contribute to fooling these models. Finally, we have applied the strongest possible augmentation against these types of attacks – a binary threshold which will decide whether to blacken or whiten each pixel depending on a certain threshold. We hypothesized that an augmentation like that will completely ruin the training procedure, but the results were surprisingly better than the last experiments, we can explain this by saying that the model has learned to focus on different and less sensitive up-sampling artifacts that are not hindered by the images brightness, therefore, when we apply these intense augmentations the model can still detect them and learn to distinguish a real image from a fake one.

Training graphs for the Deep Fake Recognizer (Image Space): The following metrics are the evaluations of our final Deep-Fake Validation model, trained on samples from the image domain. The next graphs are the training set results (evaluating on each 10th batch while training).

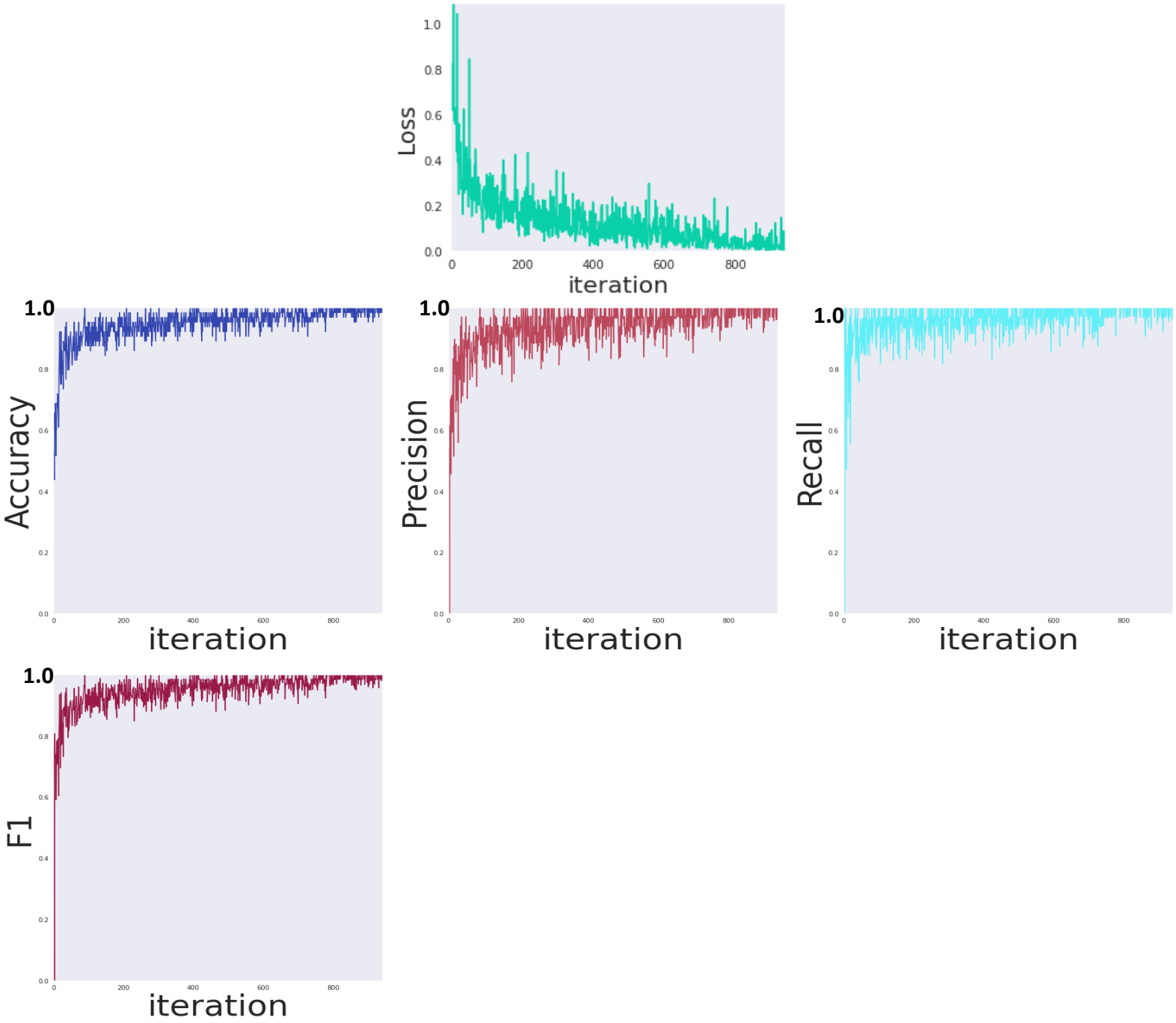


Figure 14. 1 - Training metrics of the Deep-Fake recognizer (Image Space)

The next metrics were measured on a test set of images, containing 1,500 real handwritten word samples against 1,500 fake ones. The following results are the metrics on the whole set of images against the number of epoch:

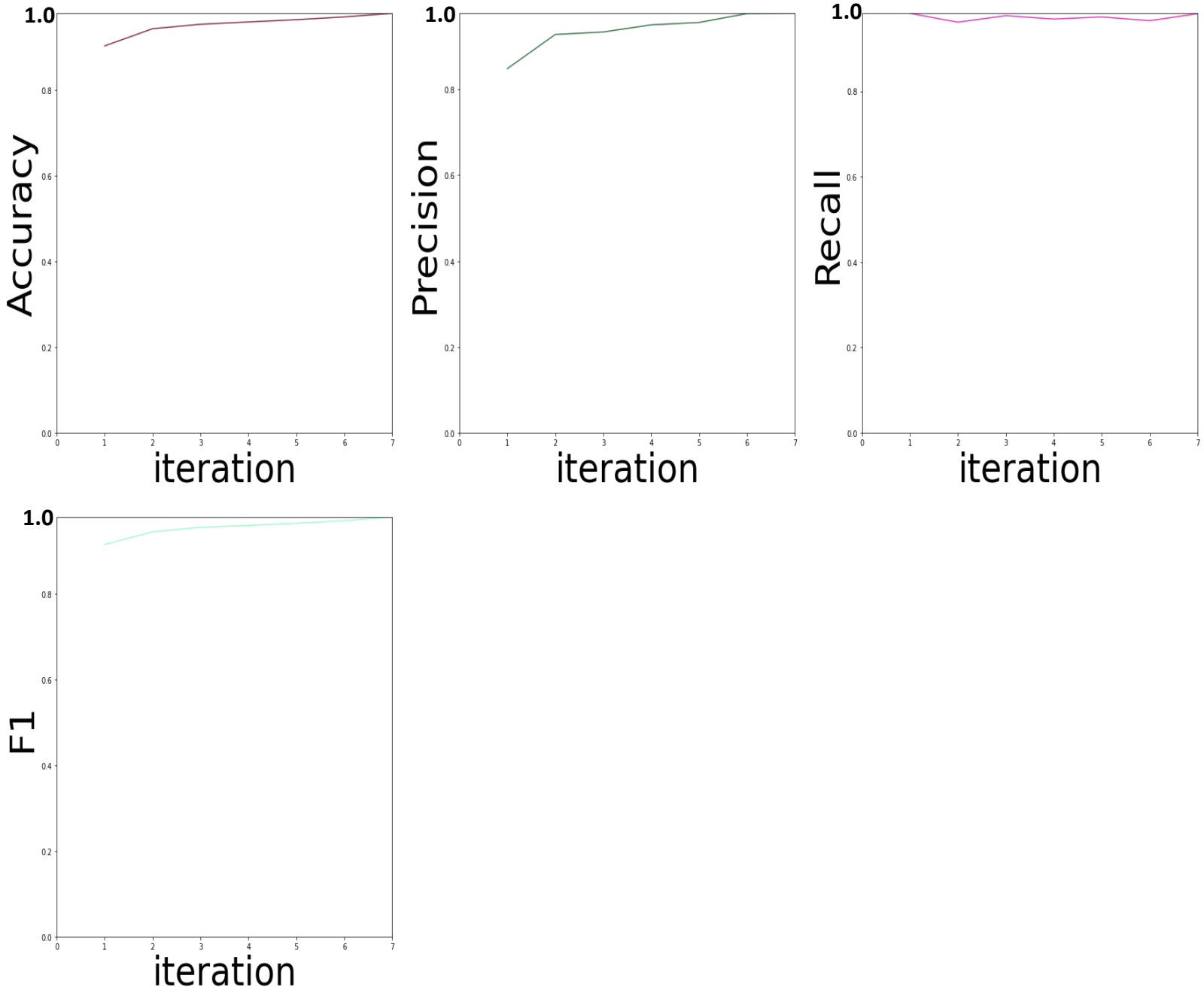


Figure 14. 2 - Test/Validation of the Deepfake Recognizer (Image Domain)

The final values are Accuracy: 0.978, Precision: 0.976, Recall: 0.981, F1: 0.979.

Training graphs for the Deep Fake Recognizer (Frequency Domain): Aside from the image space training procedure, the following metrics are the evaluations of our final Deep-Fake Validation model, trained on samples from the frequency domain. The next graphs are the training set results (evaluating on each 10th batch while training).

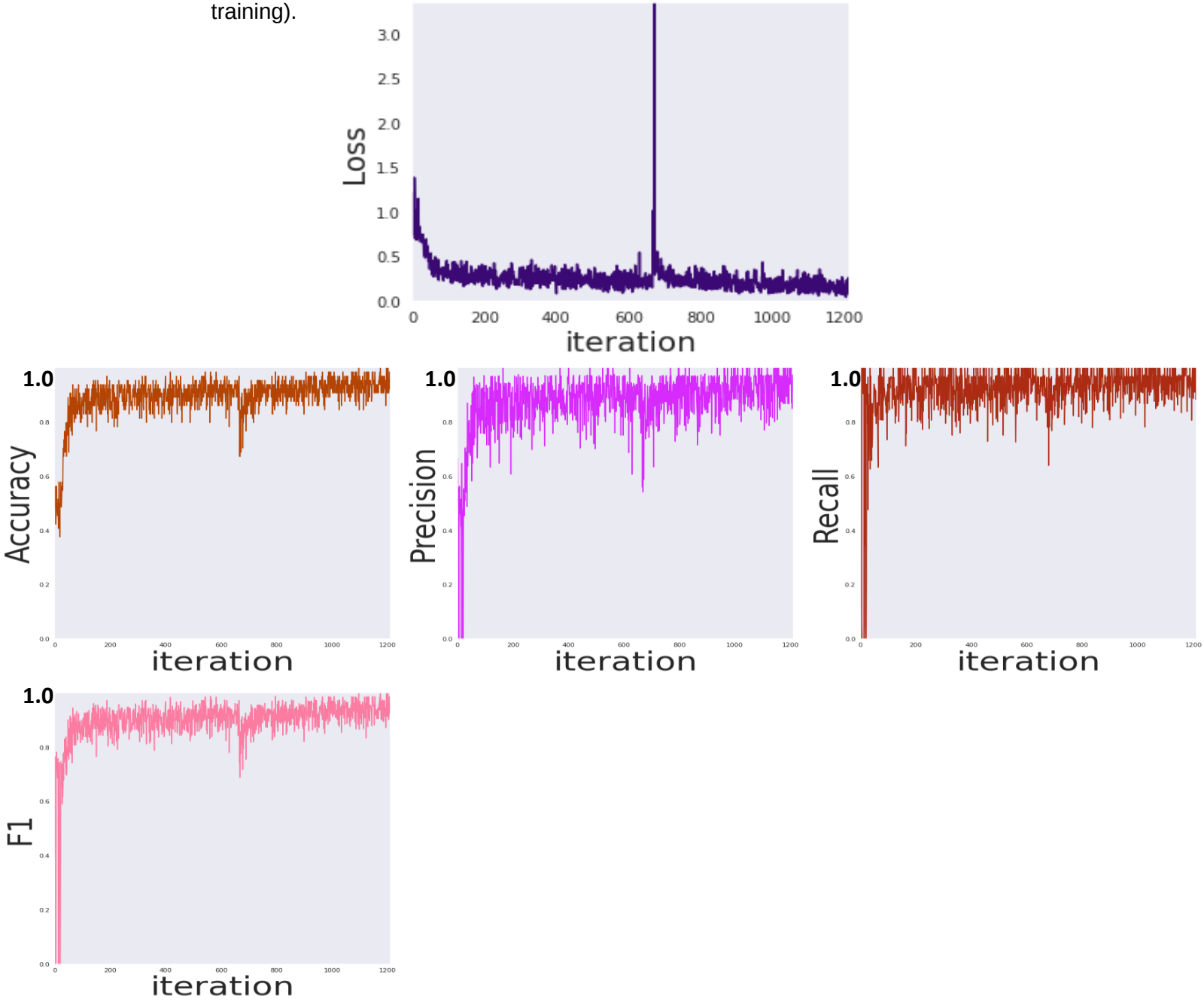


Figure 14. 3 - Training metrics of the Deep-Fake recognizer (Frequency Domain)

The same metrics were measured on a test set of images, containing 1,500 real handwritten word samples against 1,500 fake ones. The following results are the metrics on the whole set of images against the number of training epoch:

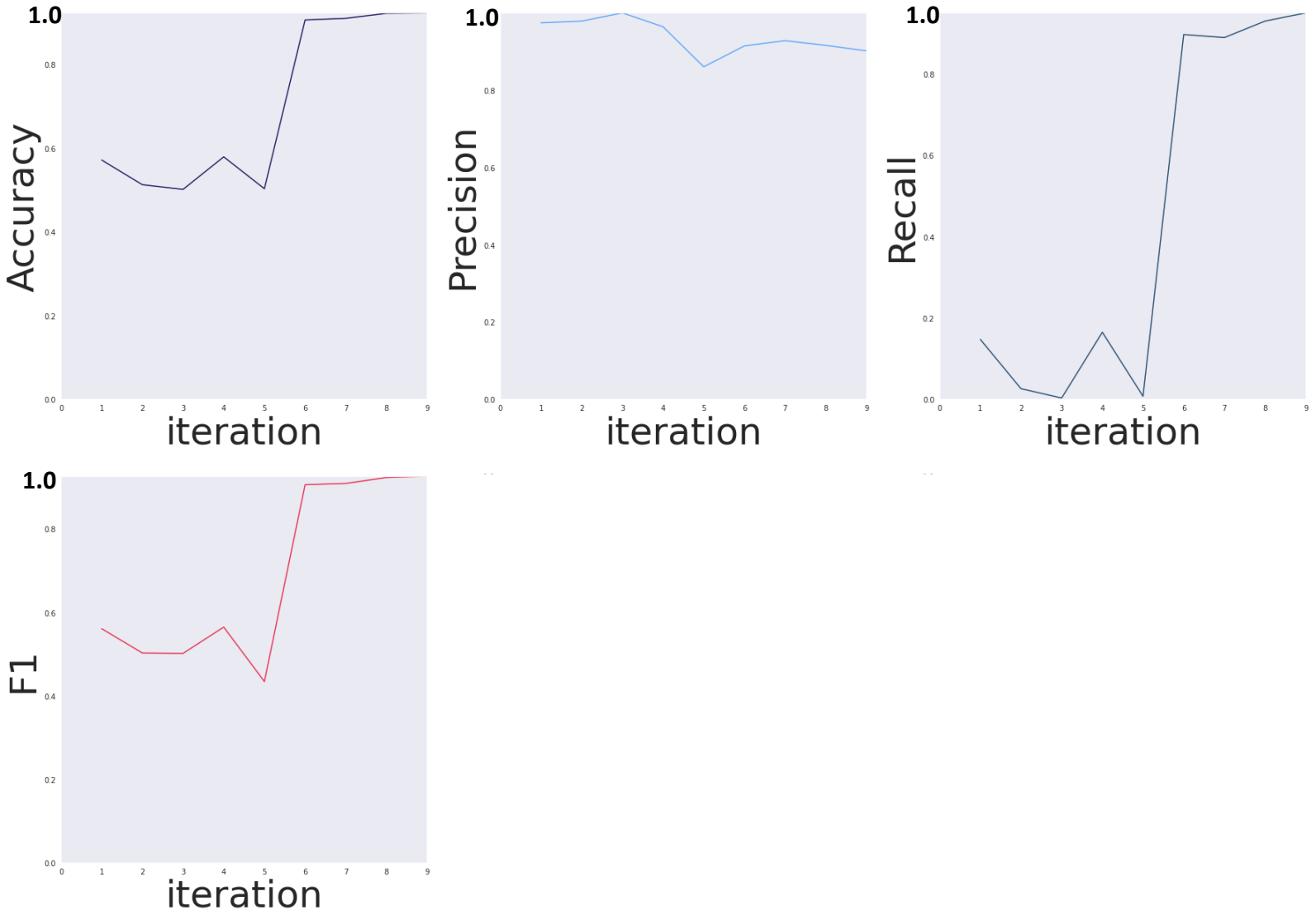


Figure 14. 4 - Validation / Test metrics of the Deep-Fake recognizer (Frequency Domain)

The final values are Accuracy: 0.923, Precision: 0.901, Recall: 0.95, F1: 0.926.

Deepfake detection system results summary: Evaluating our deepfake detection models, we managed to gain higher metrics than planned by directing the models to learn from other features than the handwriting deepfake artifacts which could be seen even from a human eye. This was done by converting each sample to a binary mask of black and white pixels only. Without doing so, the models could easily be deceived by erasing those artifacts.

In addition, we decided to use other measures as Recall, Precision, F1 score to gain more relevant information regarding the models' performance.

14.3. Writer Identity Validation

As an examination of the scope of writer font/style identification, we committed several experiments. Each of them involved a different architecture and a unique subset of handwritten words dataset, mostly from the IAM words dataset. The image samples were passed through preprocessing and augmentations:

1. **Random zero-padding (without resizing):** This function is necessary as the models can handle samples with the same resolution. Also, random padding may strengthen the models against different OCR cropping results in the testing process.

2. **Random minimizing/squeezing:** As handwriting font style is highly sensitive to features, like the writer's mood/feelings, the usage of different pens, or writing surfaces, squeezing the samples randomly among both dimensions could mimic this style sensitivity and make the models familiar with this changes.

3. **OTSU thresholding for inverted binary mask images:** In addition, in order to make our models robust in terms of varying brightness levels, we turned the images into an inversed binary mask with a uniform brightness level.

Our research is partitioned into two different phases:

14.3.1. **Phase 1:**

In this phase, we implemented three different versions of the FragNet model as the paper has no public code implementation, each with a unique approach towards solving the problem at hand. To do so, we followed step-by-step after the paper methodologies, then, we mixed these methods with our ideas, described as follows:

1. Using LSTM recurrent layers as part of the FragNet architecture: The paper's model does not use the recurrent neural network concept. As the dataset consists of handwritten word images, there is a major influence of the order in which the letters were written on the style features. This order can be interpreted as the different time-series in recurrent architectures, thus, an input word image can be represented as a sequence of letters or arranged fragments in the fragment pathway which the FragNet owns.

2. Using an Attention mechanism: Training a model on a small dataset is challenging. Hence, the attention concept may direct the model to pay attention to the relevant style features of a given writer. In this way, the performance may raise, and the complexity could be lower, due to focusing on specific calligraphic features and ignoring the noises.

First, to validate our architectures, we approached an easier classification problem of only five writers from the IAM dataset. We suggested a several architectures by the Attention and the LSTM notions. These classifiers are described as follows:

1. Attention with a general concept of the FragNet model

This implementation uses the basic concepts shown in the paper including the extraction of sub-images of a pre-determined moving window with four timestamps taken from the input word images (with shapes of 64 x 256). The partition of the words into timestamps is equivalent to a fragment window of dimension 64 x 64 (q=64). Followed by forwarding these fragments into a feature extractor network, the input also goes through an LSTM with an attention mechanism. This architecture strives to be as simplistic as possible while combining the important parts of the paper with the capabilities of the attention mechanism. One difference from the classic FragNet implementation is that here we chose to extract one feature map from the input as opposed to the feature pathway in the paper including two feature extraction paths.

2. Classic FragNet Implementation

This architecture was made to try and replicate the papers instructions and results, The generic architecture of FragNet has a feature pyramid and fragment pathways, which are fused by lateral connections. The paper review above explains the concept of the FragNet network in detail.

3. Attention LSTM FragNet Implementation

This implementation includes all of the architecture presented in the paper combined with a LSTM architecture and a full attention mechanism.

A for the 1st and the 3rd models, we chose to include an attention mechanism because we concluded that in the field of handwriting recognition, focusing on specific details in the scripts is important for distinguishing between different styles and their low-level features. The reason we included LSTM layers in the first place is that the positions of the letters in the image domain and the way that they interact with each other influence the way that the whole word is being written/classified and therefore, there is a great value to observe the data with different timestamps.

Phase 1 – results: As an initial experiment, in order to test our new implementations, we trained each architecture on an easier subset of the IAM words dataset, containing word images that were written only by 5 different writers. The training processes lasted 60 epochs. In each presented plot in this phase, each upper graph contains a metric value and the mean value of the same metric. The two lower graphs refer to the train loss/accuracy against the validation loss/accuracy values.

1. **Attention with a general concept of the FragNet model:**

   This (1st) implementation produced the best accuracy results over 5 different writers. This model has achieved an accuracy of 97.5% in the test. A few reasons related to the success of this particular model could be:

   1.1 The usage of well-known and successful backbones as Resnet50 (feature extraction pathway), VGG19 (specific fragment feature extraction).

   1.2 The usage of LSTM with an attention mechanism in the fragments pathway as described before.

   1.3 The current implementation is a deeper model than the original FragNet (which includes several convolution blocks composed of only two convolutional layers). This means that the network is capable of learning more complicated feature templates for writer styles.

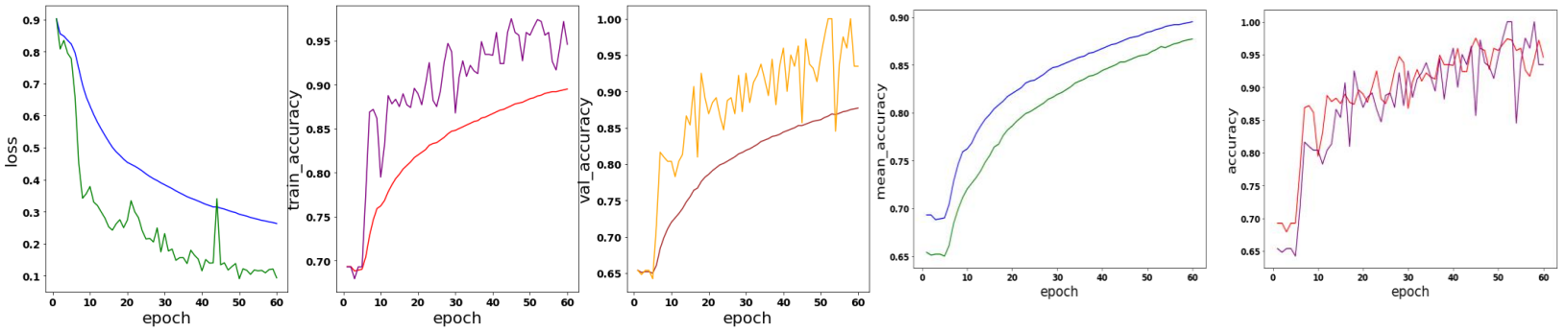


Figure 14. 5 – Training metrics for an Attention model with a general FragNet concept

2. **Classic FragNet Implementation:**

This implementation achieved lower metrics than the expected paper results, including accuracy of 83%. The reasons for these results could be as following:

2.1. The dataset contains words with varying lengths, making it difficult to choose the correct window size for the fragments given a batch of images, resulting in bad input to the network in some cases. For example, black blank fragments are cropped from short word images because the whole dataset was zero-padded.

2.2. The paper may have not included exact details regarding the precise implementation of the fragment pathway.

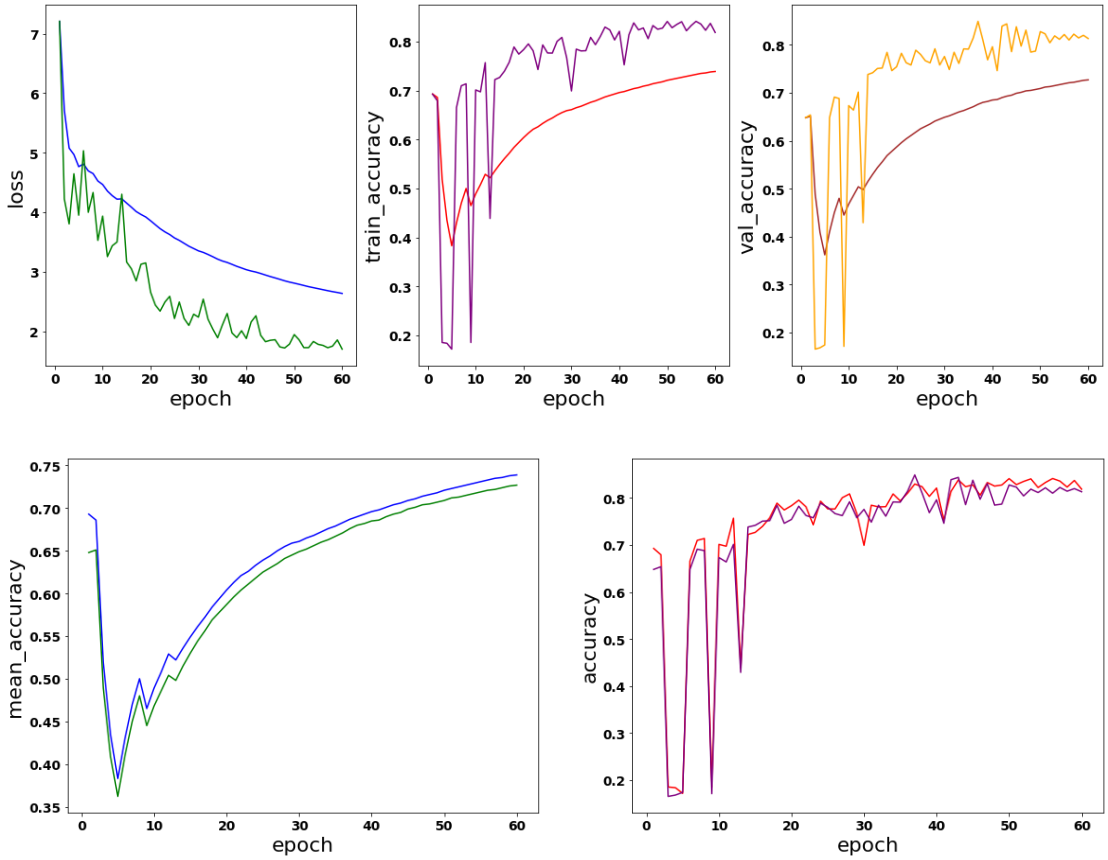


Figure 14. 6 – Training metrics for the classic FragNet implementation

3. **Attention LSTM FragNet Implementation**

This implementation achieved an accuracy value of 89.7% which is an improvement compared to the classic implementation. Some logical reasons for these results would be:

3.1 This architecture relies on the basic FragNet notion, hence the same difficulties are involved here too.

3.2 The usage of LSTM with an attention mechanism in the fragment pathway may have led this architecture to achieve a better result than the basic one.

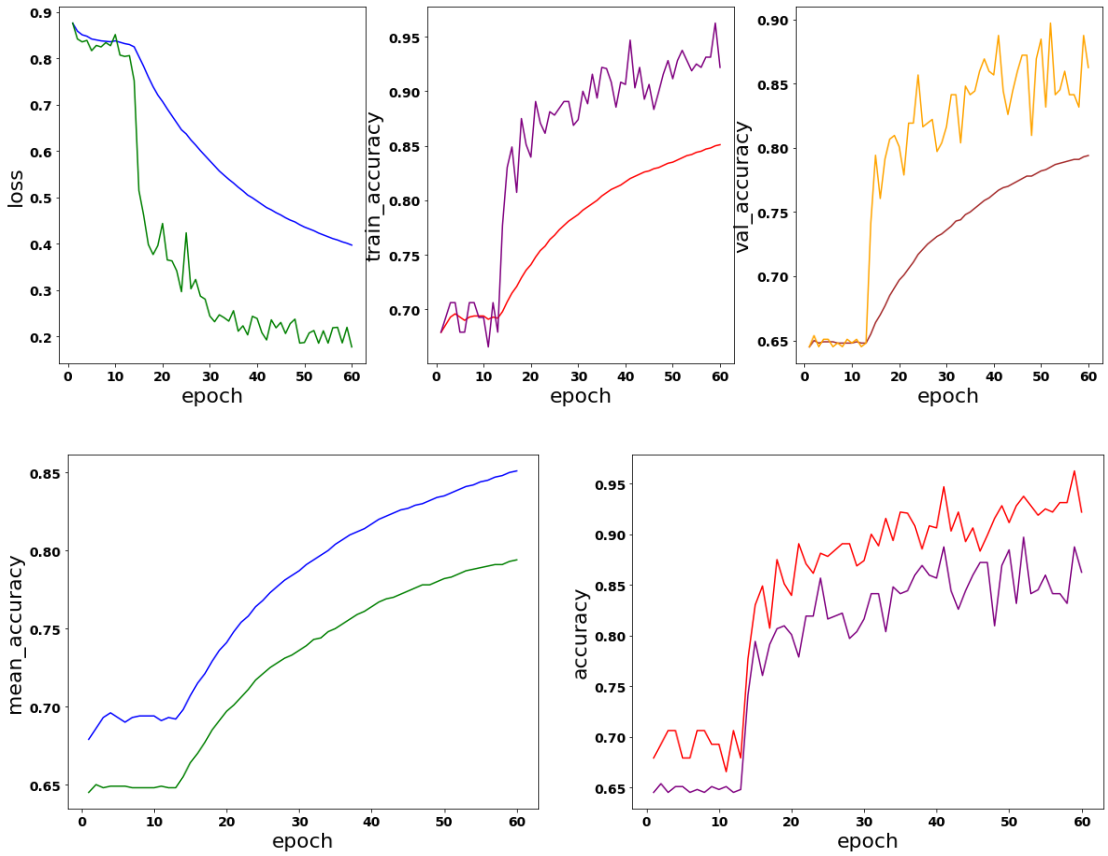


Figure 14. 7 – Training metrics for an Attention LSTM FragNet implementation

14.3.2. **Phase 2:**

After the well-known backbones (Resnet/VGG) supplied the best results on the initial validation set, we chose to move forward with them to the next experiments.

Each of the experiments treats this problem as a two classes question. The first class corresponds with the target style to validate, given a handwritten word image. To do so, we collected 443 word images, from handwritten documents, written by one of us. The document contains every letter in the English alphabet and the words within have a relatively balanced letters distribution. This small set of words was created to imitate a real-time behavior, in which the samples are written by the player, given a few paragraphs of text. The second class refers to every other calligraphic style, different from the target one. In order to make this class representative enough, we considered a subset of word images from the IAM words varied dataset. This subset of words represents 296 styles histogram that refers to writers with 100 to 800 amount of samples. While training, we chose only to use 1-2 samples for each writer of the 296 ones, to ensure the data is balanced by the two classes. The graph on the next page shows the mentioned histogram:

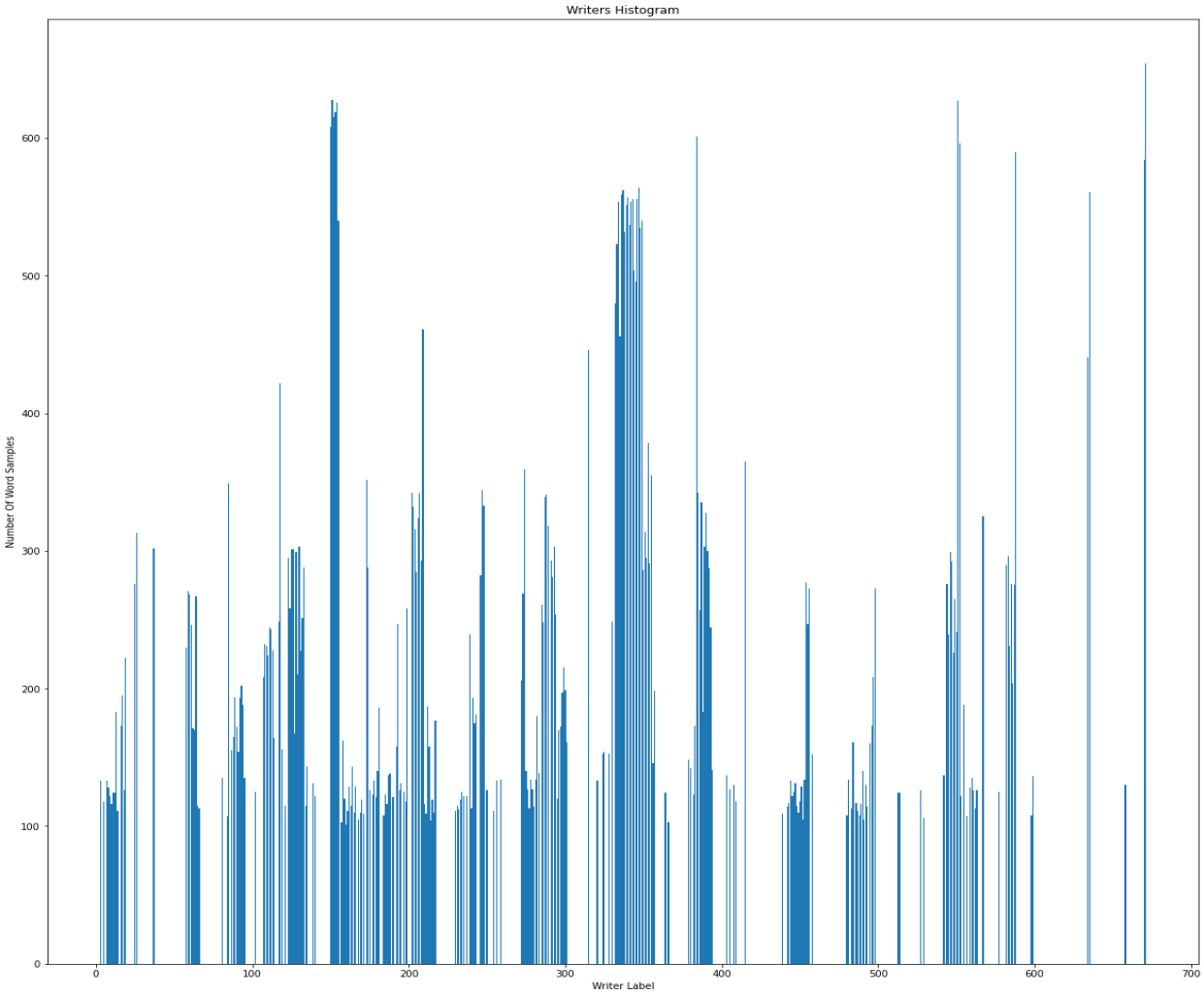


Figure 14. 8 – Writer word count histogram from the IAM dataset

14.3.2.1. **Experiment 1:**

After a trial and error process, the training in this phase was defined by the following hyper-parameters: 60 epochs, an ADAM optimizer with 0.001 learning rate, and a cosine scheduler.

Training a Resnet50 model with a fully connected classification layer, containing two output neurons, using a cross-entropy loss. The training results are shown on the next page.

The horizontal axis of each graph corresponds with the epoch numbers (0-60) and the vertical axis is the type of metric.

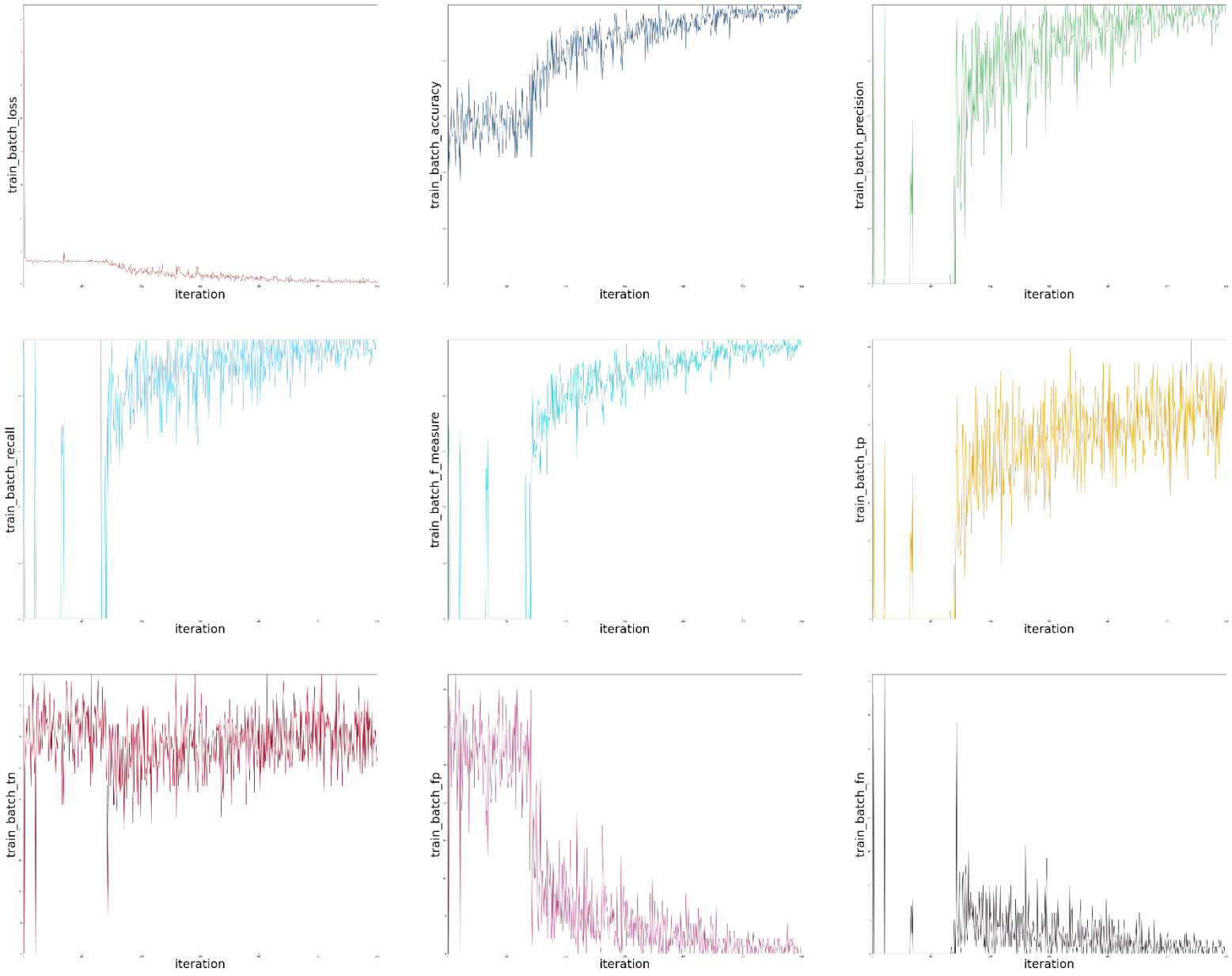


Figure 14. 9 – Training metrics for experiment 1.1

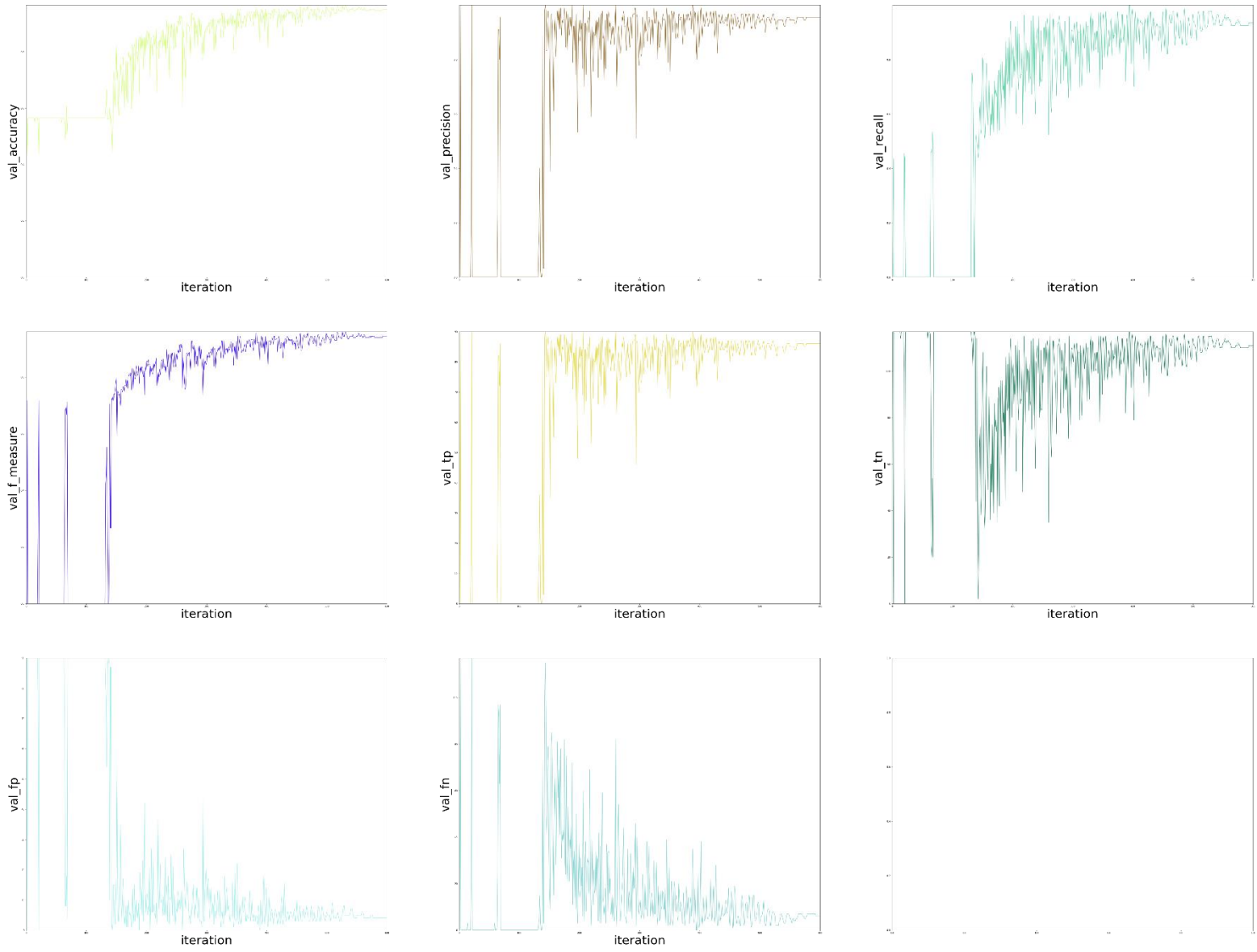


Figure 14. 10 – Training metrics for experiment 1.2

Last training set scores on a full mini-batch: Batch loss: 0.021, batch accuracy: 0.999, batch precision: 0.999, batch recall: 0.999, batch F1: 0.999, true-positives: 28, true-negatives: 36, false-positives: 0, false-negatives: 0.

Last validation set scores: Accuracy: 0.976, precision: 0.977, recall: 0.966, F1: 0.972, true-positives: 87, true-negatives: 115, false-positives: 2, false-negatives: 3.

For the test process, we collected another set of 800 words, written by 8 different writers of Israeli ethnicity (without negative samples – different calligraphic style from the target one). Each word sample subset that was written by a specific writer has the same text, including all of the letters in the English alphabet. The results as follows:

Israeli writers test set scores: Accuracy: 0.7, true-negatives: 548, false-negatives: 285.

On the one hand, the validation process on a dataset that balances between the target writer's style and the IAM writers was successful in terms of the evaluated metrics. On the other hand, the first testing process on the Israeli writers' dataset yielded a high false-negative rate. This discover may attest that the model did not focus on learning a unique handwriting calligraphic style but also learned to distinguish between an American writer from the IAM dataset to an Israeli as the target one.

14.3.2.2. **Experiment 2:**

This experiment's purpose is to confirm the hypothesis above. At this time, the same Resnet50 model with 2 classification output neurons was trained using the same training parameters. The major difference between this experiment to the previous one is the target writer to identify, which is one of the native IAM dataset writers. The training results:

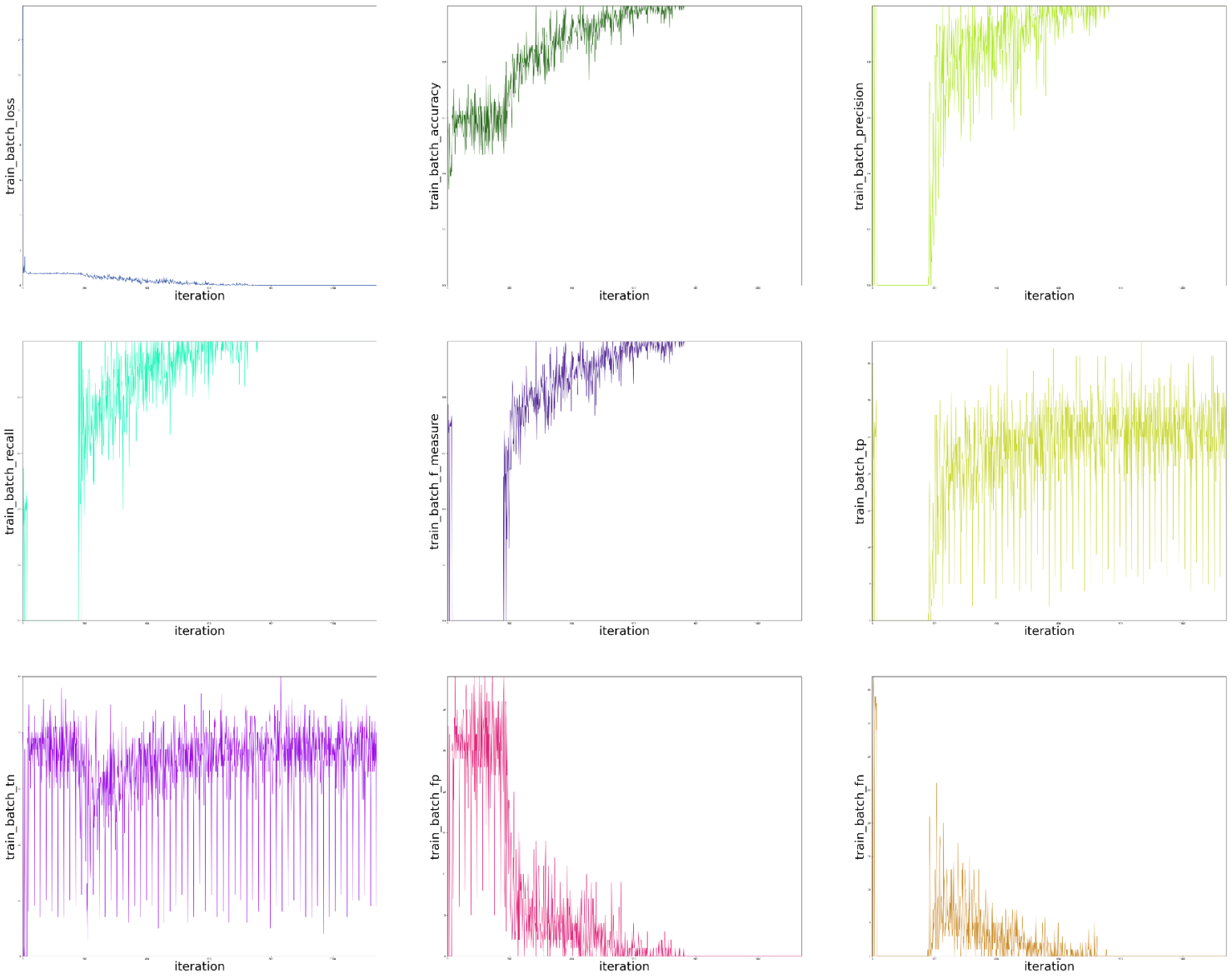


Figure 14. 11 – Training metrics for experiment 2.1

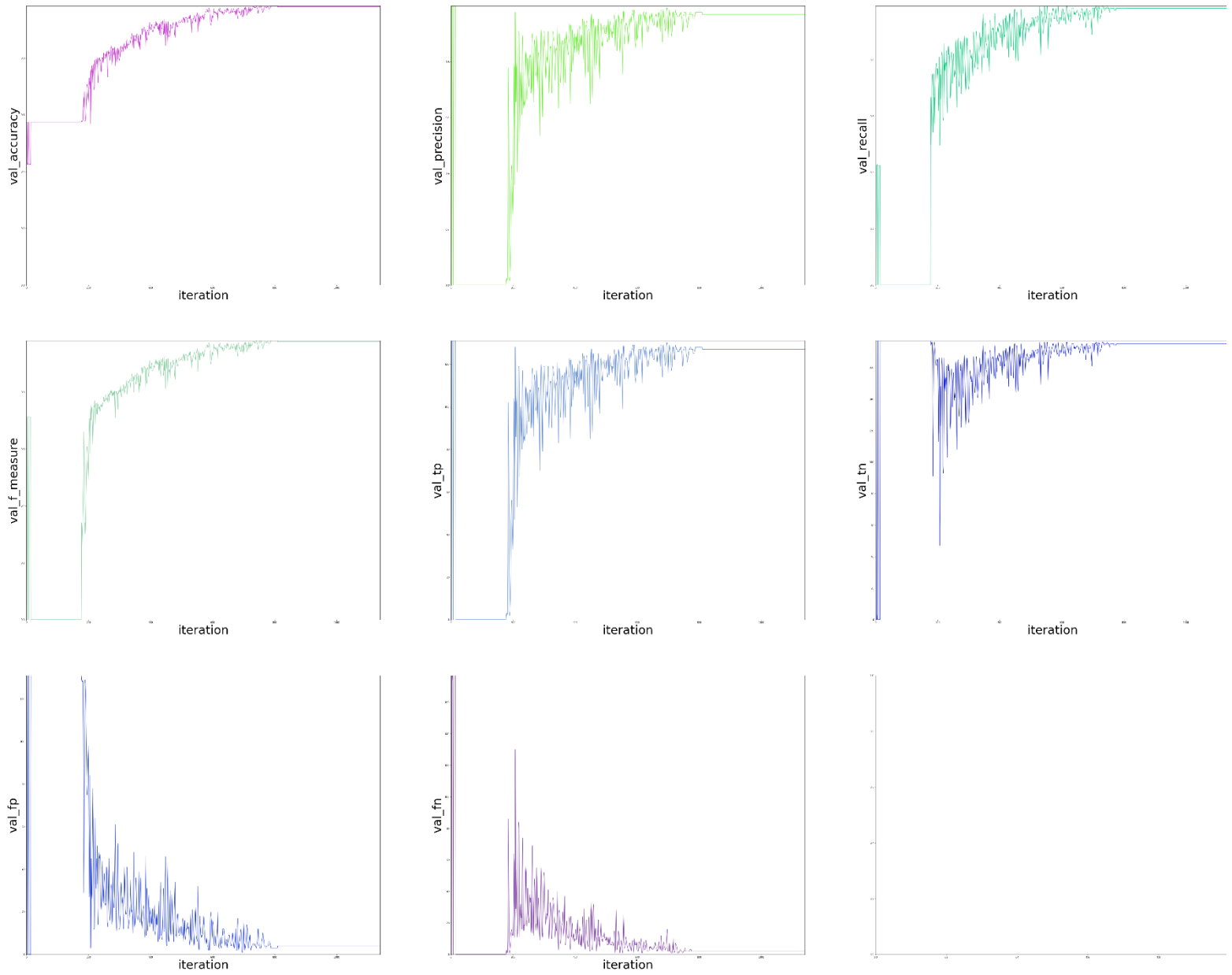


Figure 14. 12 - Training metrics for experiment 2.2

Last training set scores on a full mini-batch: Batch loss: ~0, batch accuracy: 0.999, batch precision: 0.999, batch recall: 0.999, batch F1: 0.999, true-positives: 22, true-negatives: 38, false-positives: 0, false-negatives: 0.

Last validation set scores: accuracy: 0.98, precision: 0.97, recall: 0.984, F1: 0.976, true-positives: 127, true-negatives: 175, false-positives: 4, false-negatives: 2.

We tested this model on our Israeli dataset too (without negative samples – different calligraphic style from the target one), despite it was not familiar with any Israeli writer while training. At this time, the results are:

Israeli writers test set scores: Accuracy: 0.926, true-negatives: 725, false-negatives: 58.

Although this model is not familiar with the Israeli writers' styles, the results showed that the model successfully interprets them as different styles than the target one from the IAM dataset. Hence, our conclusion is that the model is strong enough to identify American writers.

In order to examine deeply the difference between the distribution of words from the IAM dataset to the distribution of words written by writers of Israeli ethnicity, we committed another couple of experiments. In the next experiments, we built a Siamese Neural Network architecture, composed of twins Resnet50 backbones. The idea was to measure the Euclidean distance between the distribution of word images written by the target writer to those that were written by others. This way we may find the optimal threshold to isolate those two classes when prior knowledge is available – a comparative word image that refers to the target write's calligraphic style. After a trial and error process, we chose to use a threshold of 0.15.

14.3.2.3. **Experiment 3:**

By the same process above, we trained our Siamese model on the dataset which contains words from the IAM dataset and words written by the same Israeli writer, as the target class. With a Contrastive Loss function, the training results are:

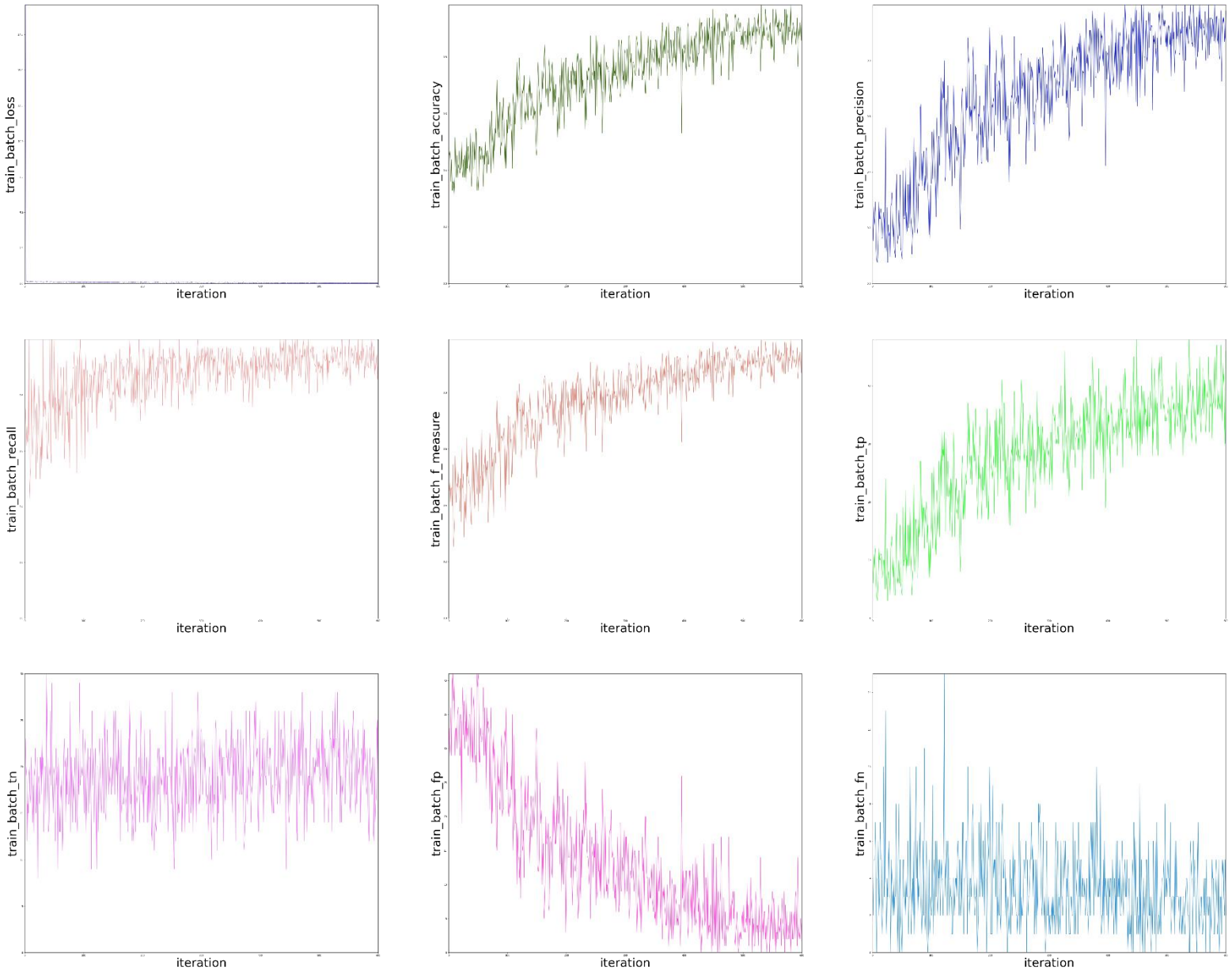


Figure 14. 13 - Training metrics for experiment 3.1

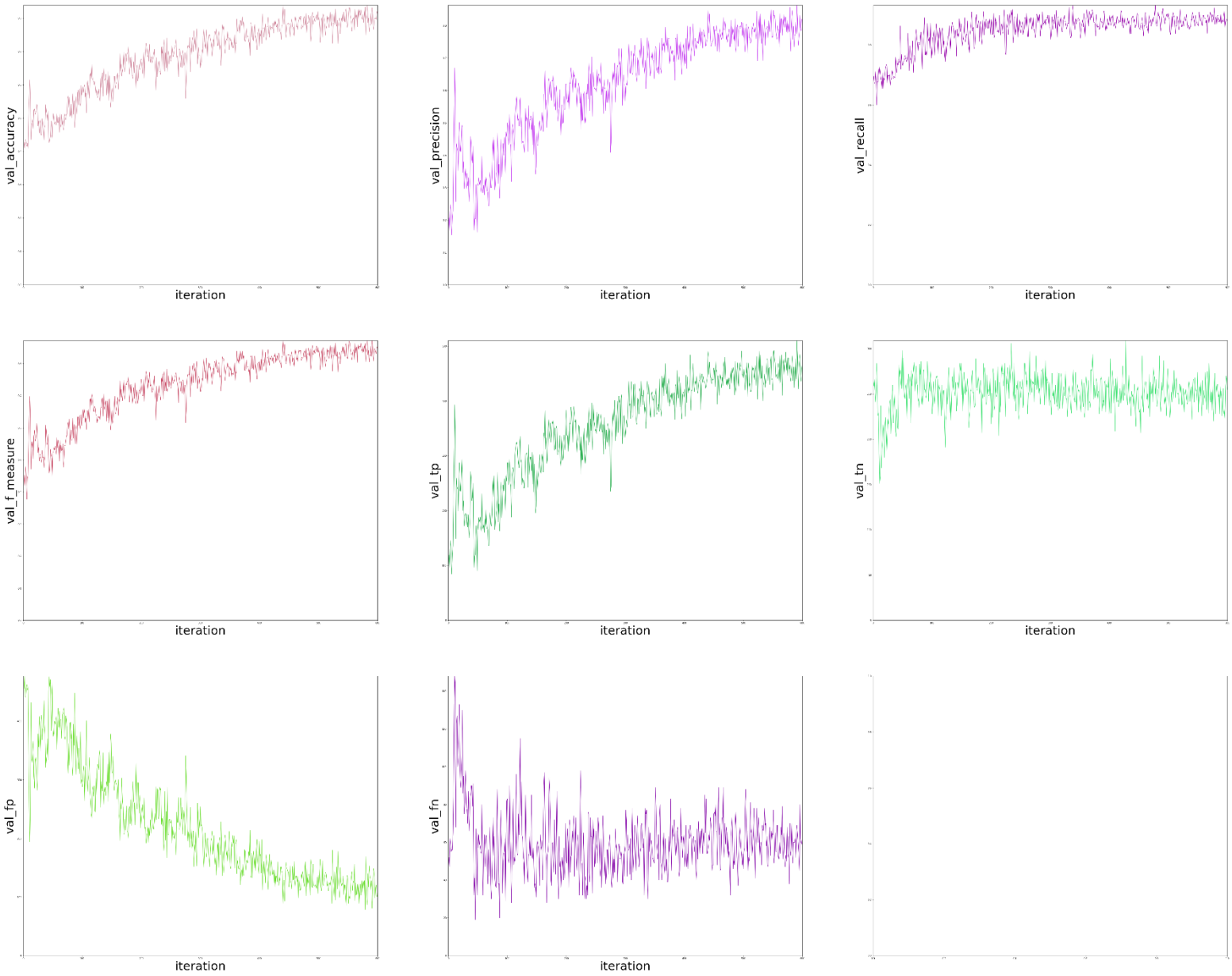


Figure 14. 14 - Training metrics for experiment 3.2

Last training set scores on a full mini-batch: Batch loss: 0.442, batch accuracy: 0.859, batch precision: 0.882, batch recall: 0.857, batch F1: 0.87, true-positives: 30, true-negatives: 25, false-positives: 4, false-negatives: 5.

Last validation set scores: accuracy: 0.82, precision: 0.82, recall: 0. 886, F1: 0.853, true-positives: 459, true-negatives: 267, false-positives: 101, false-negatives: 59.

The results of the test process on our Israeli-style handwritten word images (without any negative samples) are as follows:

Israeli writers test set scores: Accuracy: 0.747, true-negatives: 585, false-negatives: 198.

In comparison to the first experiment, the ability to distinguish between the handwriting style of a writer with an Israeli ethnicity to the handwriting of a writer from the IAM dataset was a bit improved, but still, the model from the second experiment is much better in terms of the metric values.

14.3.2.4. **Experiment 4:**

As known, Siamese Neural Networks are strongly used for accurate and deep template matching, e.g., signatures verification (SigNet [15]), face verification for security, and more. The usage of this architecture to identify the style of specific handwriting could be challenging due to the comparison of style vectors yielded from two different words in

terms of text content. To utilize this problem, we created another data histogram of the IAM dataset:

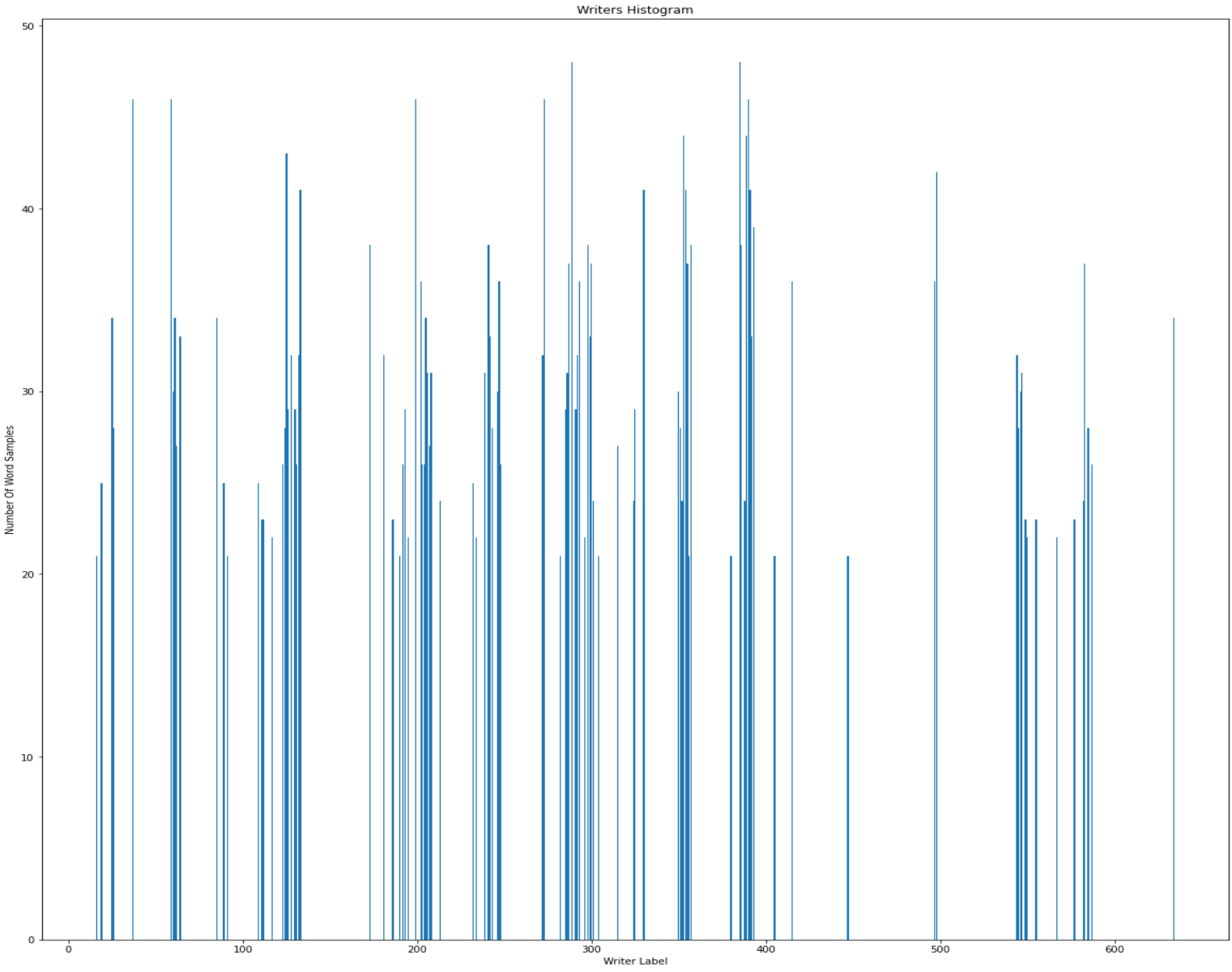


Figure 14. 15 – Word count by writer id histogram

This histogram depicts this experiment's dataset. We created clusters that contain samples with the same text content and their writer IDs. Considering a relatively balanced distribution (20-50 samples for each text label), we trained our Siamese architecture in the following way:

Each training iteration, we fed the model twice with words of the same content. At the first forward pass, the model measured the distance between two words that share the same writer ID. After, the model measured the distance between words that belong to different writer IDs. Two Contrastive Losses are handled by an average value, in order to balance between the positive and the negative samples. This way, we tried to train the model to classify better a test sample, regardless of the ethnicity of the word image that is given as prior knowledge to the model.

Siamese Resnet50 results: The results of the Siamese NN, using a Resnet50 twin backbones are visualized as follows:

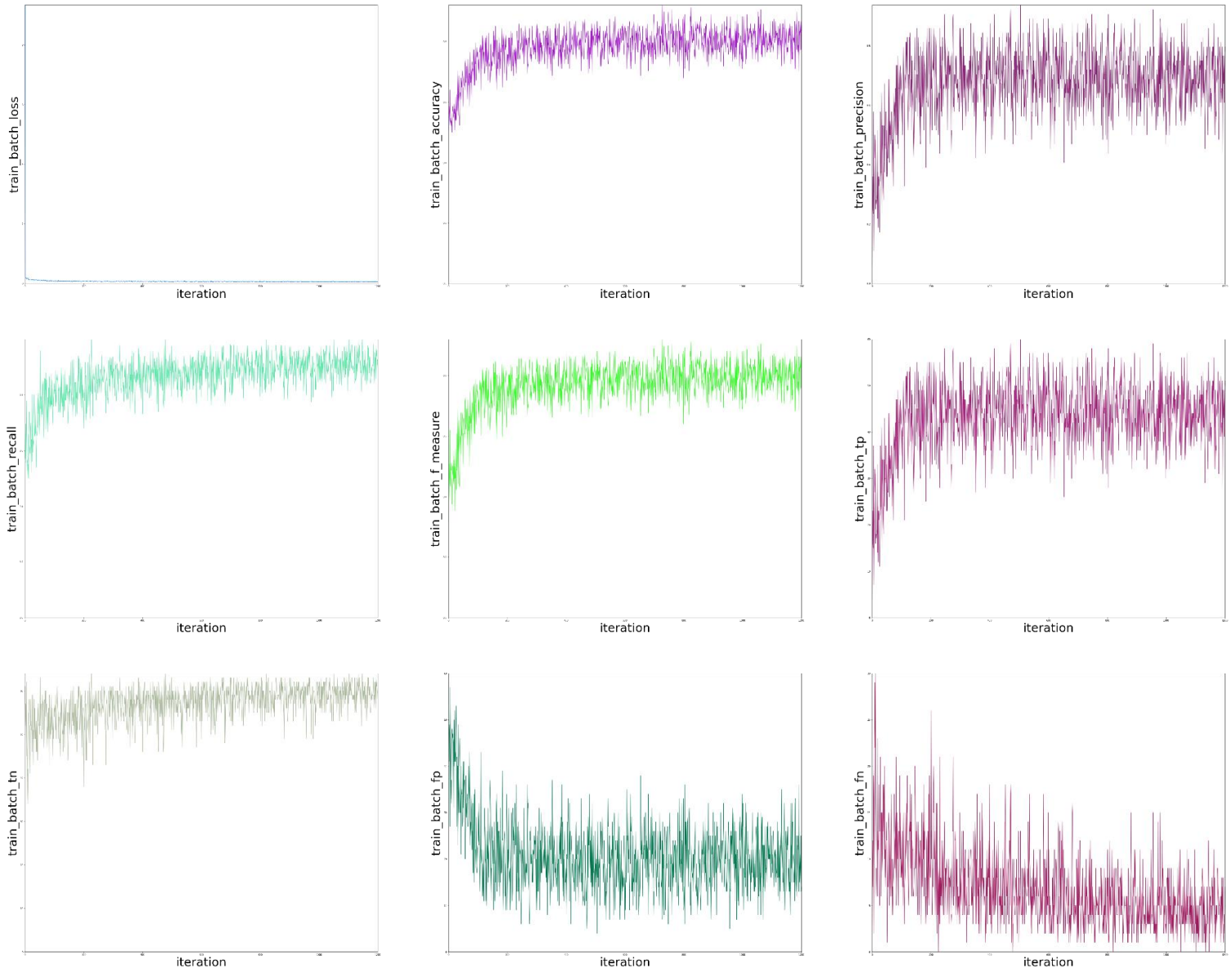


Figure 14. 16 - Training metrics for experiment 4.1

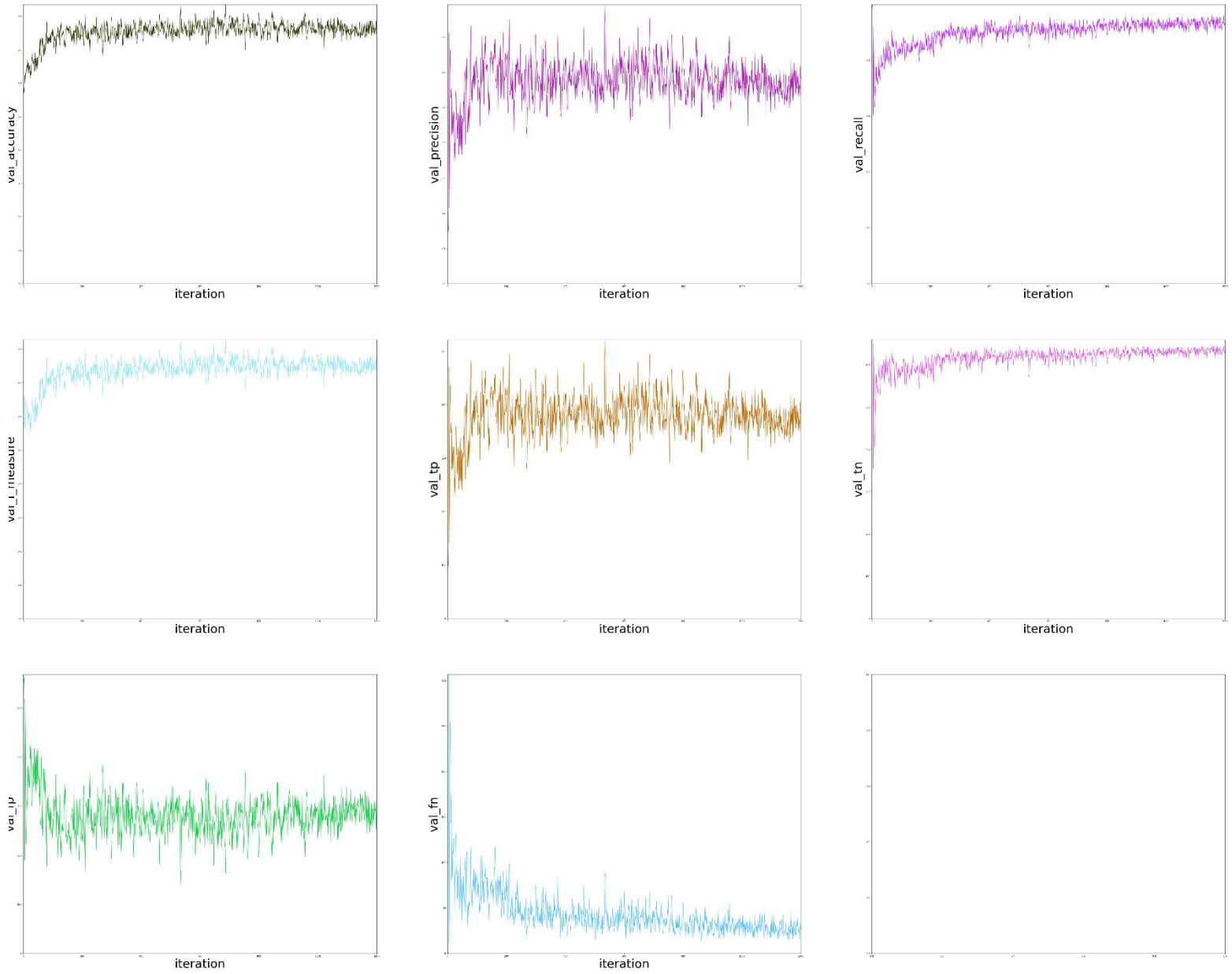


Figure 14. 17 - Training metrics for experiment 4.3

Last training set scores on a full mini-batch: Batch loss: 0.055, batch accuracy: 0.867, batch precision: 0.797, batch recall: 0.927, batch F1: 0.862, true-positives: 51, true-negatives: 60, false-positives: 13, false-negatives: 4.

Last validation set scores: accuracy: 0.769, precision: 0.588, recall: 0.921, F1: 0.755, true-positives: 388, true-negatives: 627, false-positives: 272, false-negatives: 33.

SigNet results: Unlike the Resnet50 backbones that we used above, we also tested the architecture from the SigNet [15] paper. Training this model on the handwriting styles from the IAM dataset, the results are as follows:

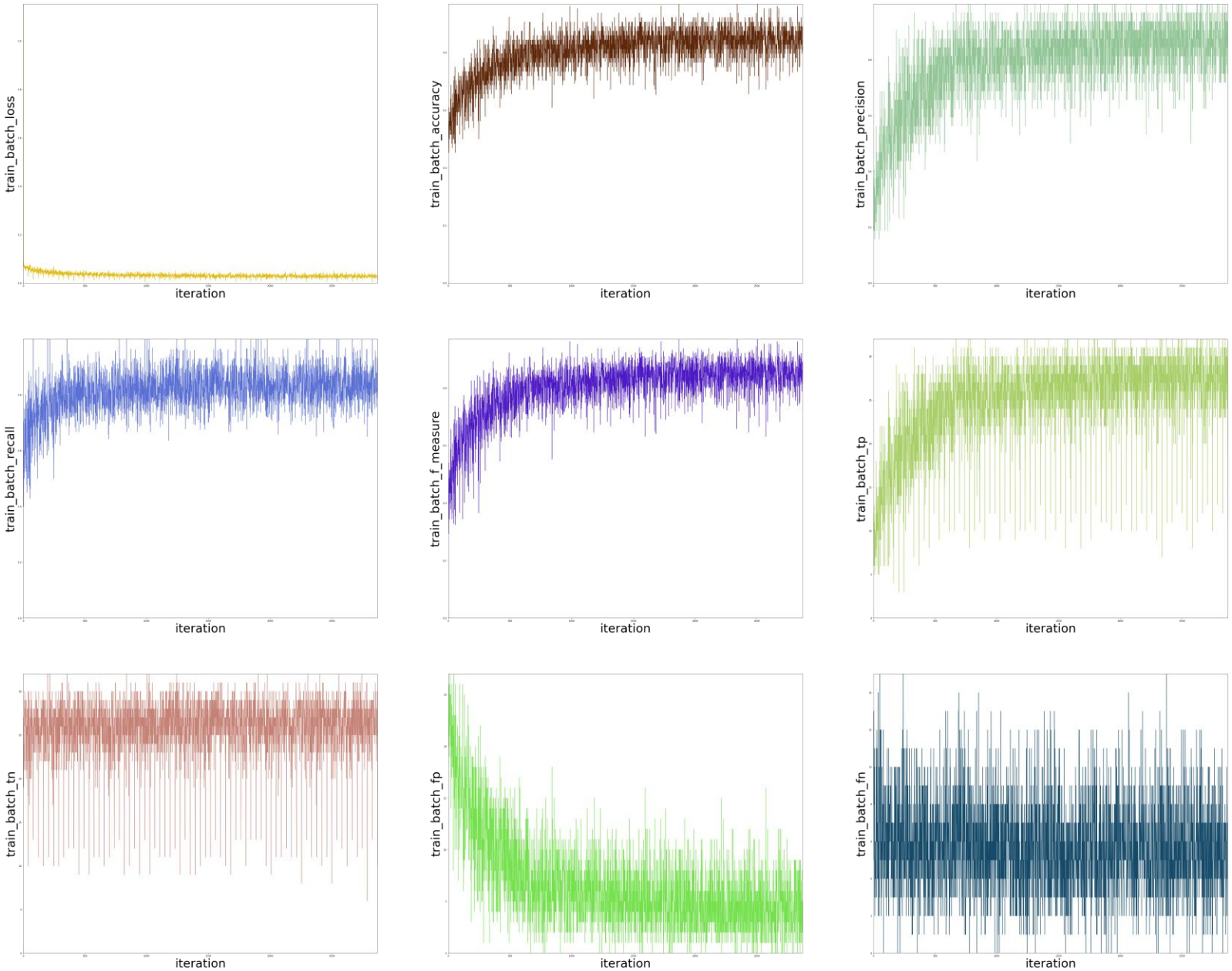


Figure 14. 18 - Training metrics for experiment 4.2

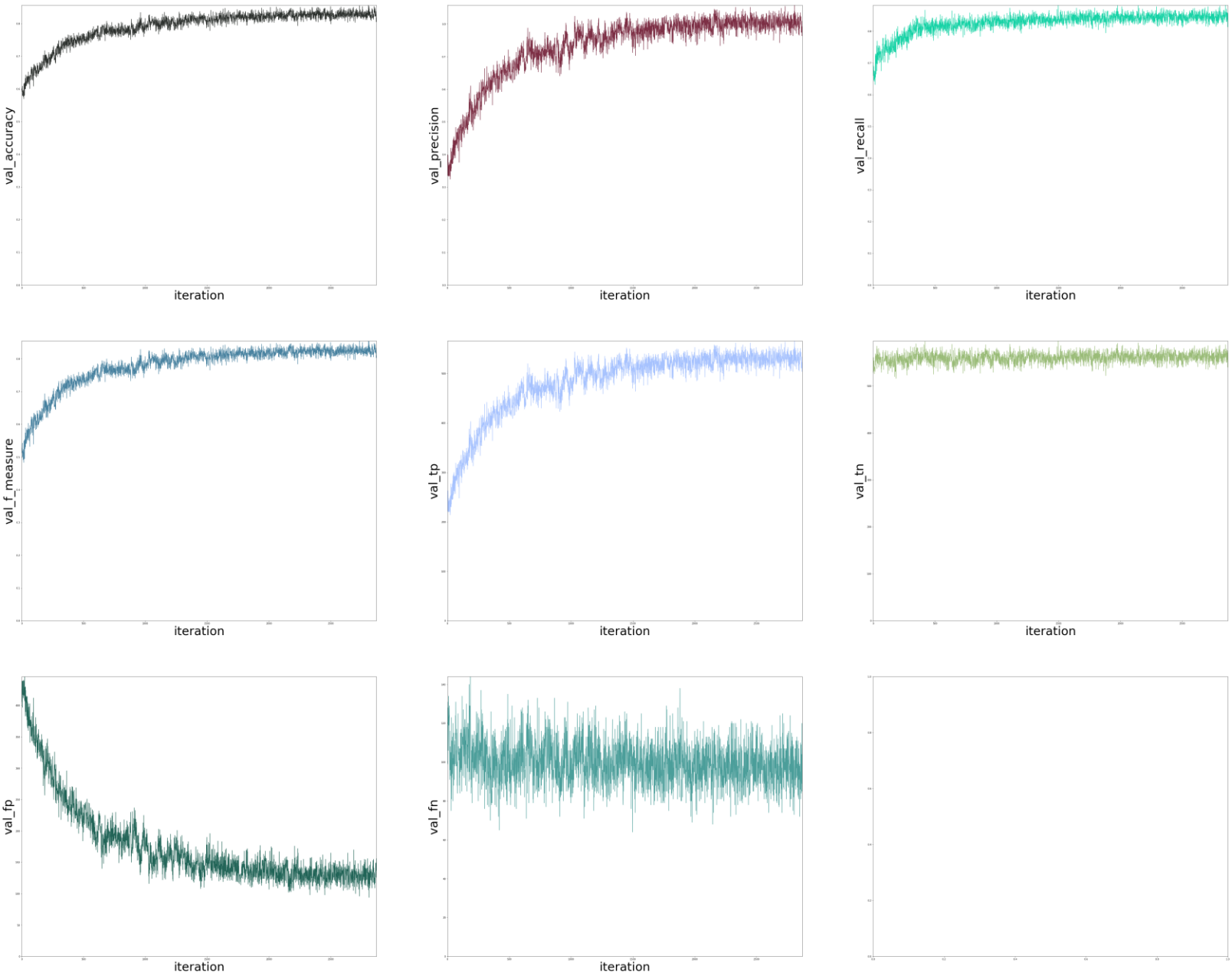


Figure 14. 19 - Training metrics for experiment 4.4

Last training set scores on a full mini-batch: Batch loss: 0.014, batch accuracy: 0.807, batch precision: 0.846, batch recall: 0.786, batch F1: 0.816, true-positives: 11, true-negatives: 10, false-positives: 2, false-negatives: 3.

Last validation set scores: accuracy: 0.814, precision: 0.79, recall: 0.83, F1: 0.81, true-positives: 521, true-negatives: 553, false-positives: 139, false-negatives: 107.

Writer Identification results summary: The FPR and the accuracy rates were raised above the expected when testing the proposed models against the IAM writers. But the models that were trained to distinguish between a writer from a different ethnic group than the IAM writers were only successful enough when it came to authors from the IAM dataset, and poorly distinguished between the target author and the writers that belong to their ethnic group. We discovered that the ethnicity of the target author regarding the scope of writer identification is a high-impact feature. In contrast, the model that was trained to identify one of the IAM writers' calligraphic style successfully surpassed these measures on both test sets, derived from the IAM dataset or from a one that is compatible with another ethnic group than the IAM ethnicity. We decided to not search/collect other datasets with authors from a different ethnicity than the IAM because such a process could highly lead to a time risk. In order to emulate IAM's ability to generalize the handwriting of British writers, it would be necessary to achieve tens of thousands of samples.

14.4. The GANWriting [13] Generator

Our generative model presents a score of 38.06 Fréchet Inception Distance (FID). Measured with 15,000 real samples vs 15,000 generated samples with the same transcription and style.

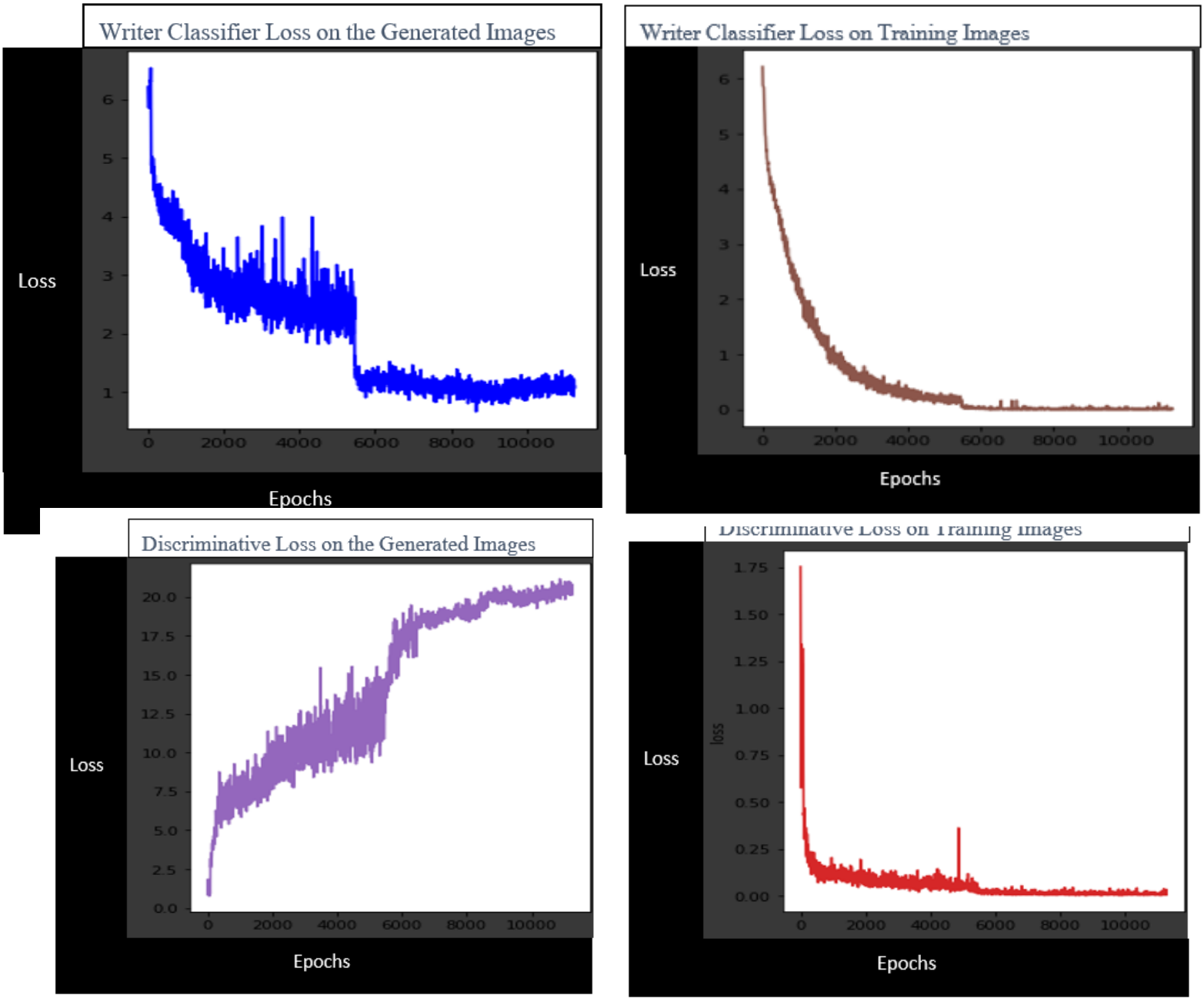
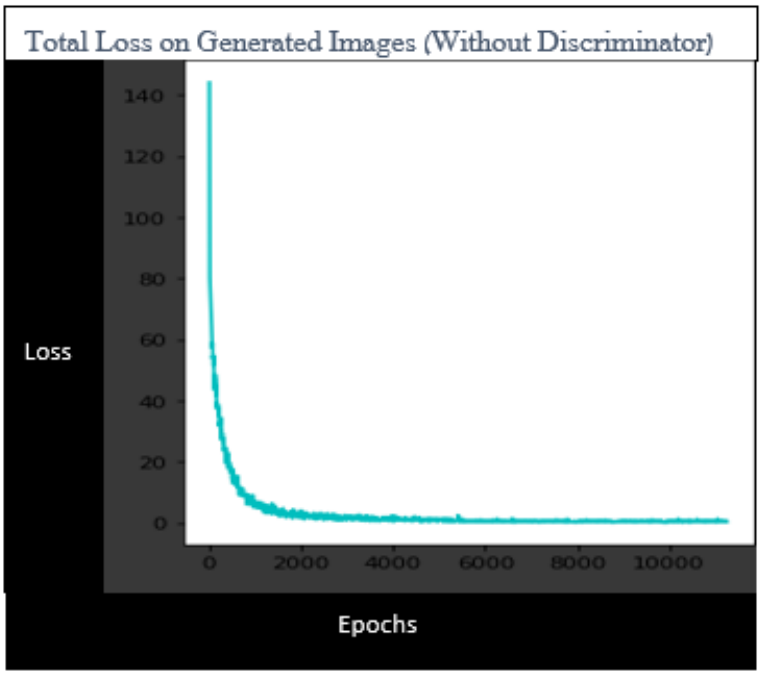
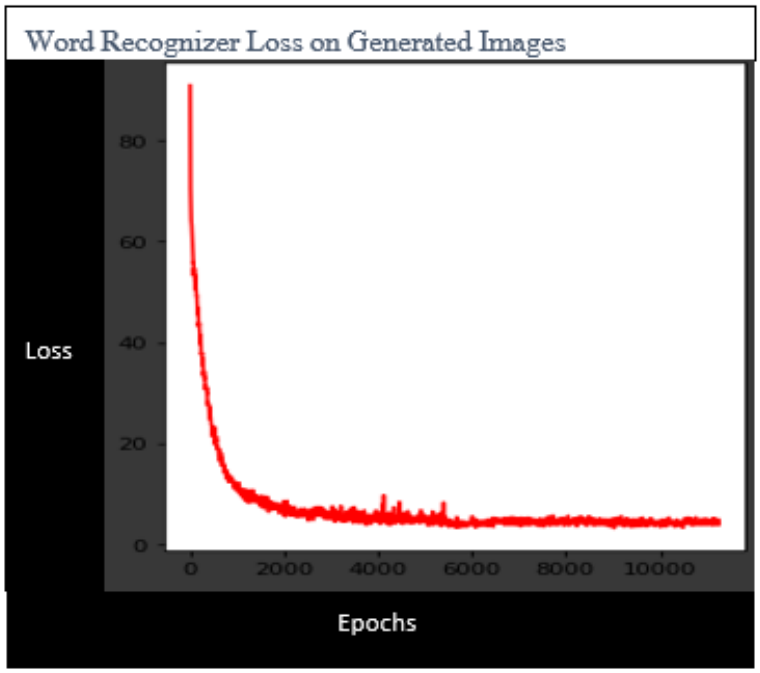
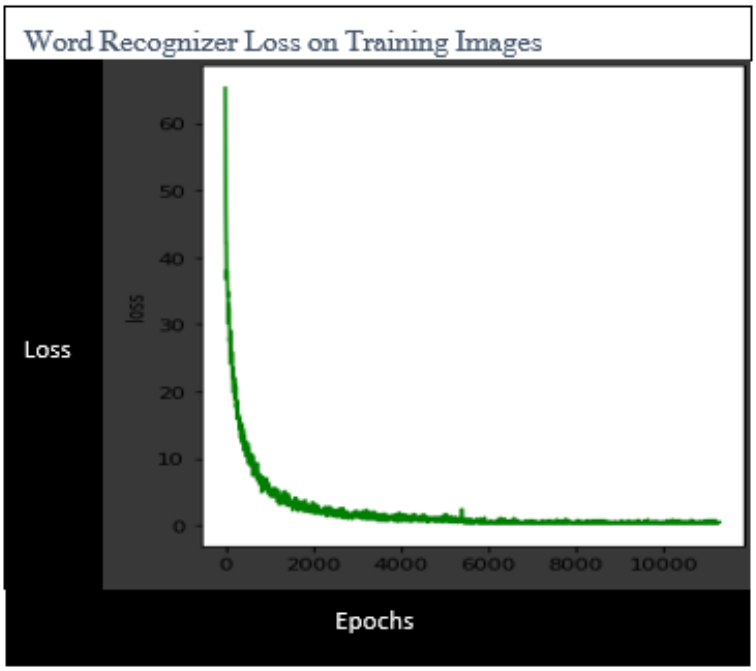


Figure 14. 20 - Training metrics for handwriting word generation (1)







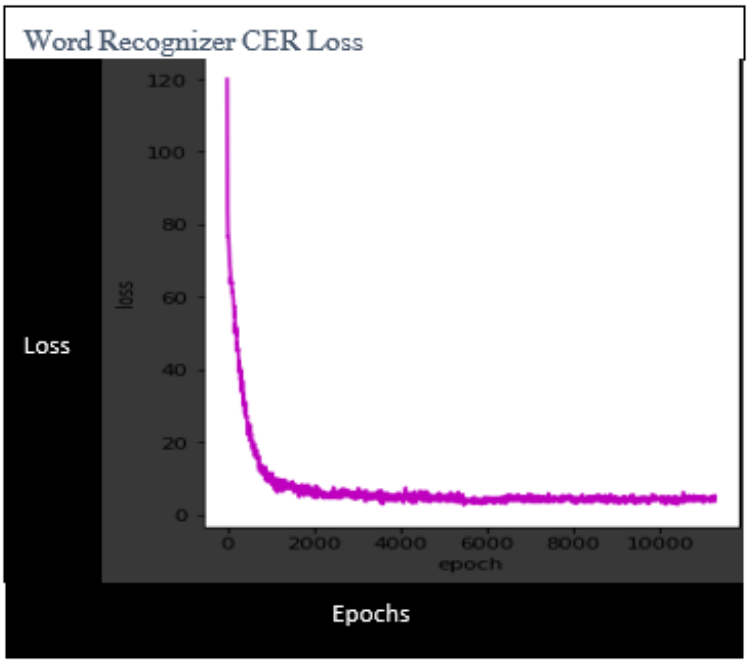
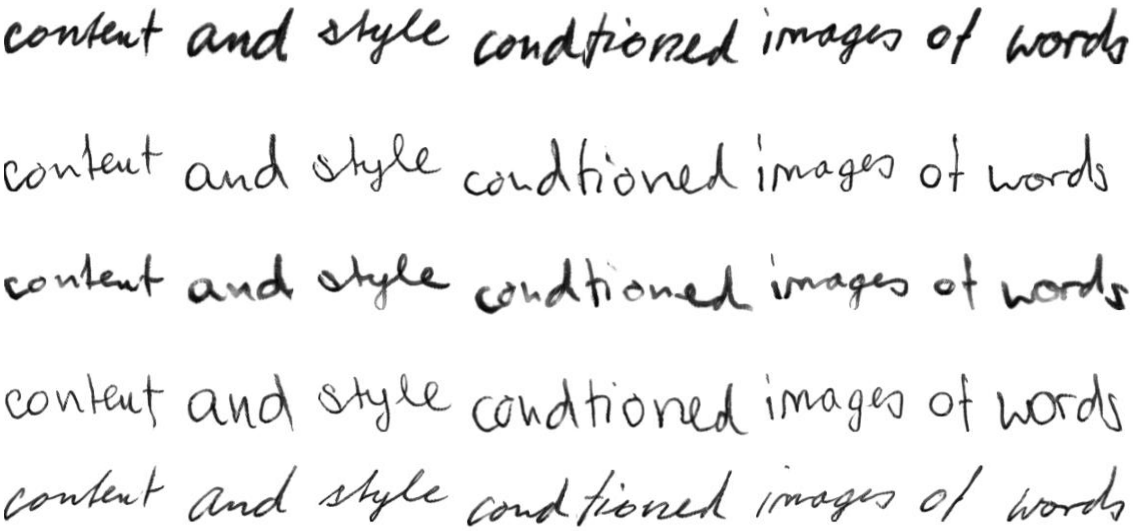


Figure 14. 21 - Training metrics for handwriting word generation (2)

Synthetic handwriting sentences, generated by the our first generative model:



From our point of view, the analysis of the generated sentences above has two aspects. On the one hand, most of the handwritten words pass the Turing Test as it is hard to distinguish whether the handwriting is authentic or not. On the other hand, some of the sentences have a similar handwriting style. It appears that the Discriminative model did a great job while the writer classifier was focused more on the high-level features like a tilt or a dense handwriting style, instead of focusing on the actual shapes of the letters and other low-level features. Next, one solution might be to use the FragNet [9] writer identification model in order to identify the writers by their handwriting style.

15. Project's Plan

In this section, the plan for the project is demonstrated below in the form of a Gantt chart. Every part of the project's process belongs to a specific timeline in the Gantt chart:

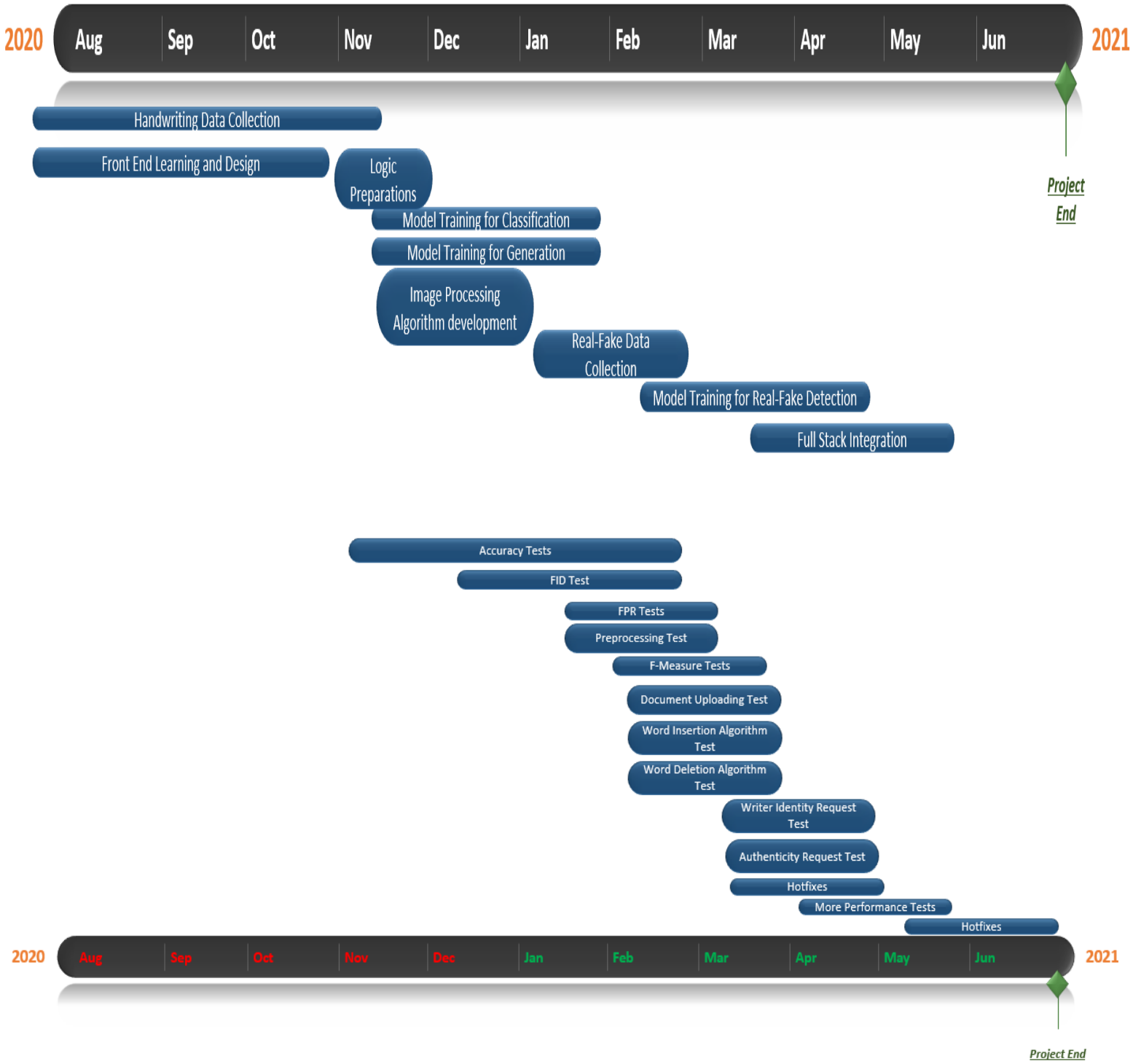


Figure 15. 1 - Gantt Diagram

16. Changes and Updates

The following table describes the changes and updates that were recently taken care of, regarding all of our project subsystems.

Change	Info	Cause	Effect
Replacing the re-training of the Writer Identification subsystem with a procedure for finding the optimal distance thresholds.	We prefer to replace the retraining process of the writer validator on a target player and replace it with a better method given our constraints of time and data.	Mainly due to the fact that the training procedure was executed on native English writers and our use case is for Israeli writers, thus much challenging the Resnet50 classifier that was trained on our handwriting against the IAM's. This difference in ethnic groups is crucial and was expanded upon earlier.	One of the main goals is to expand the knowledge in the field of computer vision, and we do contribute with information. The change does not affect that goal. In addition, we replaced the Resnet50 classifier with a Siamese validator to ignore the ethnicity features as much as we can. Hence it is not necessary to retrain it by its definition. But, finding the optimal distances between style images is a relevant process to take into account.
Replacement of the F-Measure score with the FID (Fréchet Inception Distance) score to measure our GAN models.	Changing the generative architectures' measure definition from the achievement of at least 85% F-Measure score to at most 90 FID score, using two datasets of real and fake images from the same	After deeply examining this topic, this metric was replaced with the FID metric, as it's a more common way for GAN evaluations. We achieved a high FID score (Results section – page 93) after training the architecture from the paper of GANWriting	1. It is easier for us to rely on a more common estimate to measure our generative models. 2. We spare a lot of time by using the handwriting-line generation [14] pre-trained weights.

	writers and text distributions.	[13]. As for the second architecture, we did not measure this metric and relied on its paper [14] achievements, for a few reasons: (1) The source of the paper is reliable. (2) We did not train the model ourselves, but used the published pre-trained weights. (3) The paper was released while our project was already in a critical phase of advanced development (time risk).	

Table 16. 1 – Changes and updates

This period was planned to be dedicated to the finalization and analysis of our final data and for Front-End learning, designing and integration with the rest of the project. We managed to start learning Front-End development including the designing part as anticipated by the former plan. In addition, we collected the desired state-of-the-art English datasets and not Hebrew ones as preferred before. Moreover, we finished the process of logic preparations for the initial prototype of the desired functionalities, based on the chosen papers.

17. Risk Management

The system shown in this document has a few risks involved with it. Below is a table including the possible risks and the management regarding each one of them:

NO'	Risk	Management	Risk Severity Level (1-5)	Risk Probability
1	Response time will be poor	• Efficient implementations. • Usage of external GPUs.	3	60%
2	Unethical use of the application	• The application suggests a solution: An authenticity test.	5	10%
3	Delay	• Following the project plan and minimizing schedule changes. In addition, as the project functions are dependent on each other, it is necessary to develop the project as these dependencies pipe architecture suggests OCR → Writer Identification / Handwriting Generation → Deepfake recognition.	2	80%
4	Time risk for the collection of a new dataset	• As we expanded upon, improvements can be made using a stronger dataset that includes writers of various ethnic groups and that may take time that this project has ran out of as this dataset does not currently exist and needs to be assembled using real people from all around the world.	1	100%

Table 17. 1 – Risk management

18. Software Testing and Evaluation

The proposed system relies on Deep Learning Computations. Therefore, except the controller and GUI functionalities, the tests of the system should be described by the test methodologies of the Deep Learning technique. Hence, the partition should be:

18.1. Deep Learning Tests, Measures as Accuracy, F-Measure, FID (Fréchet Inception Distance):

18.1.1. Achieving at most 90 FID score for the style-conditioned handwriting generation.

18.1.2. Achieving at least 85% accuracy and precision (minimizing the false positive rate) scores for author identity validation.

18.1.3. Achieving at least 75% accuracy and precision (minimizing the false positive rate) scores authenticity validation system.

Measurements of machine learning metrics are presented throughout the report, mainly in the Product section and therefore, are not presented here.

18.2. "Sanity Tests":

18.2.1. **Validate document uploading to the screen** – this was checked manually via simulations.

18.2.2. **Validate preprocessing algorithm, by checking whether the cropped images and the labels are correct –** we have built the real-fake dataset using crop algorithms both on our side and with Microsoft API for word detection. The results are shown in the collected dataset with thousands of correctly cropped images. This in our minds counts as a good automation.

18.2.3. **Validate handwritten words insertion to the document (on the screen)** - this was automated on google colab using randomizers and loops for different indexes on various documents and attempt to add random words, catching any run-time exceptions along the way. We also tested this on the mobile application manually on different documents and target texts.

18.2.4. **Validate handling of http requests** – this was done manually via "Postman" software for testing http requests and was previously tested on the back-end side prior to integration with the REST-API on the front-end.

18.2.5. **Validate removal of text from the screen** – this was automated on "Google Colab" using randomizers and loops for different indexes on various documents in an attempt to remove them, catching any run-time exceptions along the way. We also tested this on the mobile application manually on different documents.

Automation code for testing insertion of text:

```
def test_add_text_at_index():
    paths = glob('/content/sentences/*')
    doc = initialize_document(document_path=None, document_height=2000, document_width=1500, font_size=64)
    num_of_words = len(paths)
    for i in range(num_of_words):
        line = cv2.imread(paths[i])
        try:
            random_index = random.randint(0, i)
            imgs, labels, boxes, spaces = ocr_generated_line(line)
            doc.add_generated_sentence(random_index, imgs, labels, boxes, spaces)
        except Exception e:
            print(e)
            print("Add Word Test Failed")
            return
```

Automation code for testing deletion of text:

```
def test_delete_word_at_index():
    doc = initialize_document(document_path='/content/hand.jpeg', document_height=2000, document_width=1500, font_size=64)
    length = len(doc.words)
    while length > 0:
        try:
            random_index = random.randint(0, length)
            doc.delete_word(random_index)
            length = len(doc.words)
        except Exception e:
            print(e)
            print("Delete Word Test Failed")
            return
```

Machine Learning metrics:

```
def batch_stats(preds, wids):
    tp = 0
    tn = 0
    fp = 0
    fn = 0
    softmax = nn.Softmax(dim=0)
    for pred, wid in zip(preds, wids):
        pred = softmax(pred)
        pred = torch.argmax(pred)
        if pred == wid.item() and pred == 1:
            tp +=1
        elif pred == wid.item() and pred == 0:
            tn += 1
        elif pred != wid.item() and pred == 1:
            fp +=1
        elif pred != wid.item() and pred == 0:
            fn += 1
    return tp, tn, fp, fn
```

```
def get_all_metrics(model, dataloader, device, alpha=0.5):
    model.eval()
    TP, TN, FP, FN = 0, 0, 0, 0
    exp = 0.0000001
    for x, y in tqdm(dataloader):
        x = x.to(device)
        y = y.to(device)
        tp, tn, fp, fn = batch_stats(model(x), y)
        TP += tp
        TN += tn
        FP += fp
        FN += fn
    acc = (TP + TN) / (TP + TN + FP + FN + exp)
    precision = TP / (TP + FP + exp)
    recall = TP / (TP + FN + exp)
    f_measure = alpha * precision + (1-alpha) * recall
    print(f"\nTP={TP}\nTN={TN}\nFP={FP}\nFN={FN}")
    print(f"\nAccuracy={acc}\nPrecision={precision}\nRecall={recall}\nF_Measure={f_measure}")
    return acc, precision, recall, f_measure
```

Figure 18. 1 – Automation and machine learning algorithms

19. Summary and Conclusions

Due to the fact that our project is divided into different subjects, we will present the conclusions regarding each of those subjects.

19.1. Handwriting Generation

In terms of a general handwriting style generation, we have managed to use two of the main papers. First, we developed an adequate environment for the training procedure of the model of the first paper (GANWriting) and found that generating a general handwriting style is possible with this model. Afterward, we decided to test a newer state-of-the-art architecture with the same purpose. Although the results for style conditioned generation were approximating the target writer in a decent level, we saw that the results were better for a general handwriting style and also for a style conditioned handwriting. Thus, we chose to integrate the 2nd architecture into our system.

19.2. Deep-fake detection

Using both GAN architectures described above, we created a real-fake dataset of handwritten words and forms with varying writers (and fake writers for generated images) and transcriptions. The dataset constitutes some 120,000 images, half of which are real images from the IAM dataset, and the other half is generated by both are models. The chosen transcriptions match the transcriptions on the real side of the dataset and the writer's style was extracted from the real images in order to be as close as possible and add more challenge to the training procedure. Based on this dataset, we trained a deep-learning model capable of detecting GAN-generated images and distinguish them from authentic ones. The model learns to detect features that are created during up-sampling procedures in the generation architecture of modern generative models.

19.3. **Writer Identity Validation**

After deeply examining this scope, we realized the greatest solution is using a suitable dataset, containing all of the relevant handwriting features as the common calligraphic features among different cultures. Thus, we achieved the goal under the assumption that the validation process is always done on a handwritten text that was written by the same ethnic group as the authors from the IAM dataset. Although it wasn't perfect, a lot of experiments were made in order to gain better results on word images that correspond with our ethnic group. These experiments contained different architectures and unique data partitioning.

20. Future Work

We believe that further work can be done on these topics, here are the main points for each topic that we think are important to follow up on:

20.1. Writer Identity Validation

After a large amount of experiments on this topic, an assumption was raised: The ethnicity of the datasets writers is significant and may confuse models that are familiar only with the ethnicity of the IAM writers. In order to validate this assumption, it is necessary to collect a new dataset that is compatible with the writer's ethnic group. High metrics scores will prove that the ethnicity of a writer is important to identify his calligraphic-ethnic style, even when the words are written in a foreign language, as in the current research. Another important step is to collect an ethnic test set to test the model described in the fourth experiment against another ethnic group. In addition, it may be important to collect a new dataset that contains the same samples as the dataset from the last experiment, but also contains enough word samples that were written by the target writer. This way, machine learning models could directly focus on word images of a specific target writer, in contrast to the last experiments.

20.2. Deep-Fake Detection

In terms of deep fake detection and with the rapid growth in generative architectures, it would be a good idea to strengthen the versatility of the model proposed in our work by training it on samples from a greater pool of deep fake generators. This way the model will learn to look for varying types of artifacts that lie in images created by a great set of up-sampling and image processing algorithms that occur in these types of models and architectures.

20.3. Handwriting Generation

One of the key issues with the state of the art architectures in terms of style-conditioned handwriting output in cases where the target style is not in the training set, is that it doesn't resemble the original handwriting enough in order to be invisible to the naked eye and fit in a seamlessly into the paragraph as we expect from it. This is probably largely due to the writer identity classifier attempting to classify images through detection of weak visual features such as color, thickness, tilt or spacing, that may be enough for a regular classifier to achieve a high accuracy rate due to the probabilities being borderline correct, but does not extract an adequate feature map that will force the generative model to output a very similar style. This is why we believe that the road to a more seamless handwriting generation model is highly dependent on a writer style extractor that would learn to detect very subtle visual features in one's calligraphic scripts. Resulting in more meaningful losses in the training process, and a stronger feature map that could truly draw a multi-dimensional plane of key features that could differentiate between authors.

21. References

[1] Alonso, Eloi, Bastien Moysset, and Ronaldo Messina, "Adversarial generation of handwritten text images conditioned on sequences", 2019 International Conference on Document Analysis and Recognition (ICDAR), IEEE, 2019.

[2] Shao, Lei, et al. "Attention GAN-based method for designing intelligent making system", IEEE Access 7: 163097-163104, 2019.

[3] Nguyen, Huy H., Junichi Yamagishi, and Isao Echizen. "Capsule-forensics: Using capsule networks to detect forged images and videos", ICASSP 2019-2019 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP). IEEE, 2019.

[4] Poznanski, Arik, and Lior Wolf. "Cnn-n-gram for handwriting word recognition", Proceedings of the IEEE conference on computer vision and pattern recognition, 2016.

[5] Bao, Jianmin, et al. "CVAE-GAN: fine-grained image generation through asymmetric training", Proceedings of the IEEE international conference on computer vision, 2017.

[6] Hsu, Chih-Chung, Yi-Xiu Zhuang, and Chia-Yen Lee. "Deep fake image detection based on pairwise learning", Applied Sciences 10.1: 370, 2020.

[7] Xing, Linjie, and Yu Qiao. "Deepwriter: A multi-stream deep CNN for text-independent writer identification", 2016 15th international conference on frontiers in handwriting recognition (ICFHR). IEEE, 2016.

[8] Zhang, Xu-Yao, et al. "End-to-end online writer identification with recurrent neural network", IEEE transactions on human-machine systems 47.2: 285-292, 2016.

[9] He, Sheng, and Lambert Schomaker. "Fragnet: Writer identification using deep fragment networks", IEEE Transactions on Information Forensics and Security 15: 3013-3022, 2020.

[10] Bappy, Jawadul H., et al. "Hybrid LSTM and encoder–decoder architecture for detection of image forgeries", IEEE Transactions on Image Processing 28.7: 3286-3300, 2019.

[11] Frank, Joel, et al. "Leveraging frequency analysis for deep fake image recognition", International Conference on Machine Learning. PMLR, 2020.

[12] Hitaj, Briland, Giuseppe Ateniese, and Fernando Perez-Cruz. "Deep models under the GAN: information leakage from collaborative deep learning", Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security, 2017.

[13] Kang, Lei, et al. "GANwriting: Content-conditioned generation of styled handwritten word images", European Conference on Computer Vision. Springer, Cham, 2020.

[14] Davis, Brian, et al. "Text and Style Conditioned GAN for Generation of Offline Handwriting Lines", arXiv preprint arXiv:2009.00678, 2020.

[15] Dey, Sounak, et al. "Signet: Convolutional siamese network for writer independent offline signature verification", arXiv preprint arXiv:1707.02131, 2017.

[16] Karras, Tero, Samuli Laine, and Timo Aila. "A style-based generator architecture for generative adversarial networks", Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition, 2019.

[17] Krishnan, Praveen, Rama Kovvuri, Guan Pang, Boris Vassilev, and Tal Hassner. "TextStyleBrush: Transfer of Text Aesthetics from a Single Example." *arXiv preprint arXiv:2106.08385* (2021).

[18] He, Sheng, and Lambert Schomaker. "GR-RNN: Global-context residual recurrent neural networks for writer identification." *Pattern Recognition* 117 (2021): 107975.

22. Appendices

23. SRD (Software Requirements Document).
24. SDD (Software Design Description).
25. STD (Software Test Documentation).