POLITECNICO DI TORINO

MASTER DEGREE COURSE IN COMPUTER ENGINEERING

Master´s Degree Thesis

Automated forms cleaning by considering form with coloured backgrounds

Supervisors
Prof. Alessandro SAVINO
Dr. Kilian FOTH

Candidate
Mahsa FARJOO

April 2023

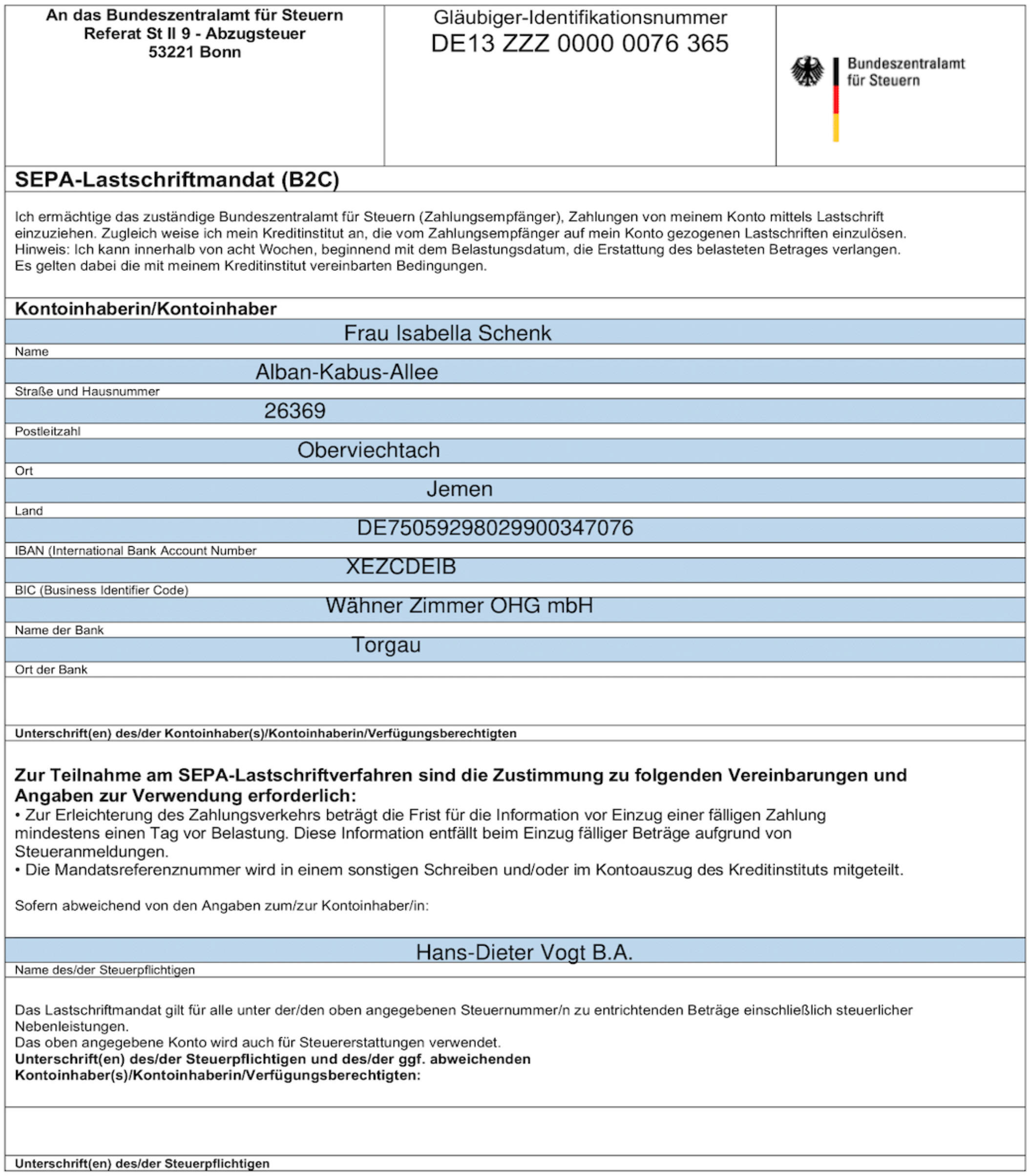
Summary

The primary objective of this project is to extract or erase information from completed forms. As you may already know, forms comprise both static and dynamic contents. While the static contents remain the same for every client/customer, the dynamic contents vary.
Considering the scarcity of real datasets for these forms, a simulation method was employed to generate a fictitious dataset. These unfilled forms are in PDF format. Thus, the first step involves generating the dynamic content of the forms and inserting them in different locations on the form. Then, these filled forms are converted from PDF to PNG.
To generate the dynamic content of the form, I first analyze the required dynamic content since there are various categories and subcategories of dynamic data. Subsequently, a fake Python library is used to generate random dynamic data and fill the form. For example, to fill the empty form shown in Figure 1, the Python library generates information such as:

- German full name (First name and Last name)
- Address (Street, House Number, Zip Code, City, Country)
- Bank account information (City, Name of Bank, IBAN, BIC)

After generating dynamic data and filling out the forms, our dataset is automatically created. Consequently, we require automatic information extraction since human labor is insufficient for extracting and erasing dynamic information, which is a time-consuming process.
To train a Deep Learning model that extracts static forms from user-filled dynamic forms, we can use such simulated datasets. In this project, I utilized Generative Adversarial Networks (GANs), which are one of the most popular topics in Deep Learning and are a type of Neural Network utilized for unsupervised learning.

A CycleGAN includes four models, two generators, and two discriminators. CycleGAN generators take the form of either U-Net or Res-Net. I employed both U-Net and Res-Net architectures. The Deep Learning model will be trained to differentiate between static and dynamic content and erase only the dynamic content. As these forms have colored backgrounds, the background must be reconstructed after removing the dynamic contents. Consequently, static templates can be extracted automatically from simulated complete documents.



An das Bundeszentralamt für Steuern **Referat St II 9 - Abzugsteuer** **53221 Bonn**	Gläubiger-Identifikationsnummer DE13 ZZZ 0000 0076 365	Bundeszentralamt für Steuern

SEPA-Lastschriftmandat (B2C)

Ich ermächtige das zuständige Bundeszentralamt für Steuern (Zahlungsempfänger), Zahlungen von meinem Konto mittels Lastschrift einzuziehen. Zugleich weise ich mein Kreditinstitut an, die vom Zahlungsempfänger auf mein Konto gezogenen Lastschriften einzulösen.
Hinweis: Ich kann innerhalb von acht Wochen, beginnend mit dem Belastungsdatum, die Erstattung des belasteten Betrages verlangen.
Es gelten dabei die mit meinem Kreditinstitut vereinbarten Bedingungen.

Kontoinhaberin/Kontoinhaber

Name	Frau Isabella Schenk
Straße und Hausnummer	Alban-Kabus-Allee
Postleitzahl	26369
Ort	Oberviechtach
Land	Jemen
IBAN (International Bank Account Number	DE75059298029900347076
BIC (Business Identifier Code)	XEZCDEIB
Name der Bank	Wähner Zimmer OHG mbH
Ort der Bank	Torgau

Unterschrift(en) des/der Kontoinhaber(s)/Kontoinhaberin/Verfügungsberechtigten

Zur Teilnahme am SEPA-Lastschriftverfahren sind die Zustimmung zu folgenden Vereinbarungen und Angaben zur Verwendung erforderlich:
• Zur Erleichterung des Zahlungsverkehrs beträgt die Frist für die Information vor Einzug einer fälligen Zahlung mindestens einen Tag vor Belastung. Diese Information entfällt beim Einzug fälliger Beträge aufgrund von Steueranmeldungen.
• Die Mandatsreferenznummer wird in einem sonstigen Schreiben und/oder im Kontoauszug des Kreditinstituts mitgeteilt.

Sofern abweichend von den Angaben zum/zur Kontoinhaber/in:

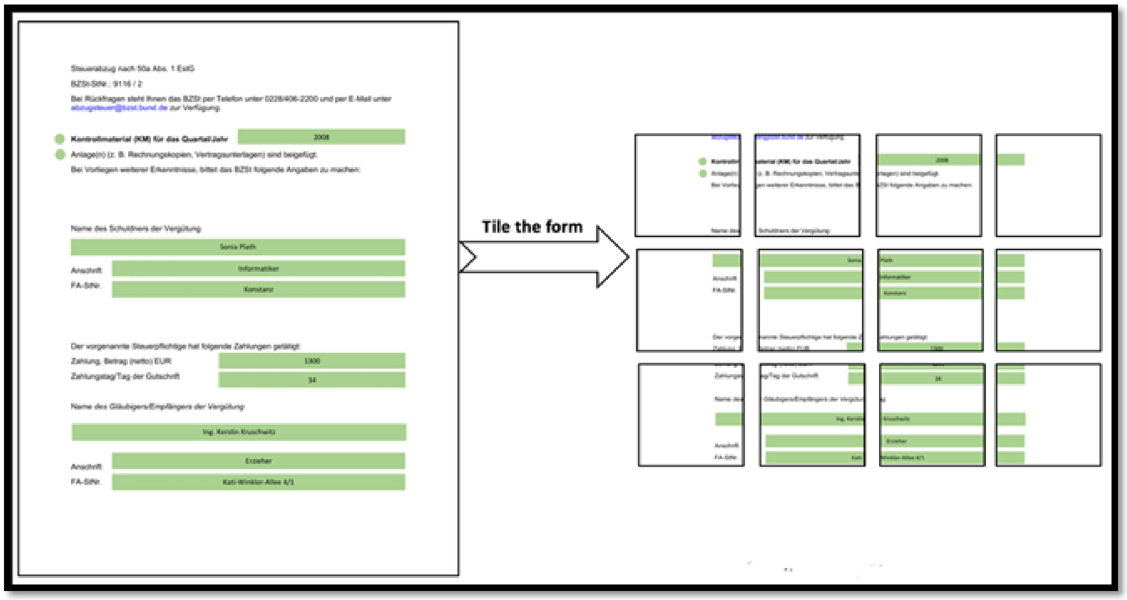
Name des/der Steuerpflichtigen	Hans-Dieter Vogt B.A.

Das Lastschriftmandat gilt für alle unter der/den oben angegebenen Steuernummer/n zu entrichtenden Beträge einschließlich steuerlicher Nebenleistungen.
Das oben angegebene Konto wird auch für Steuererstattungen verwendet.
Unterschrift(en) des/der Steuerpflichtigen und des/der ggf. abweichenden Kontoinhaber(s)/Kontoinhaberin/Verfügungsberechtigten:

Unterschrift(en) des/der Steuerpflichtigen

Figure 1: Sample of Filled form

Once the forms are filled out and converted from PDF to PNG, the dataset is prepared for feeding into a deep neural network model. However, due to their large size of 1700*2200 pixels, the forms/images are processed in tiles of size 128*128. Since the size of the forms/images does not align with the tile size, it is necessary to first center the images on the canvas and then pad them with white pixels all around the image. Once this step is complete, the images can be converted into tiles of size 128*128 (as depicted in Figure 2).


Steuerabzug nach 50a Abs. 1 EstG

BZSt-StNr.: 9116 / 2

Bei Rückfragen steht Ihnen das BZSt per Telefon unter 0228/406-2200 und per E-Mail unter abzugsteuer@bzst.bund.de zur Verfügung.

Kontrollmaterial (KM) für das Quartal/Jahr 2008

Anlage(n) (z. B. Rechnungskopien, Vertragsunterlagen) sind beigefügt.

Bei Vorliegen weiterer Erkenntnisse, bittet das BZSt folgende Angaben zu machen:

Name des Schuldners der Vergütung: Sonia Plath

Anschrift: Informatiker

FA-StNr.: Konstanz

Der vorgenannte Steuerpflichtige hat folgende Zahlungen getätigt:

Zahlung, Betrag (netto) EUR: 1300

Zahlungstag/Tag der Gutschrift: 34

Name des Gläubigers/Empfängers der Vergütung: Ing. Kerstin Kruschwitz

Anschrift: Erzieher

FA-StNr.: Kati-Winkler-Allee 4/1

Figure 2: Converting Images to tiles with 128*128 size

Dynamic content in forms has two main characteristics: a coloured background and variable location within the form. Therefore, it is necessary for a deep learning model to differentiate between dynamic and static content. We introduce a model that can learn to distinguish between the two types of content in an image, erase dynamic content in filled forms, and restore the original background color.

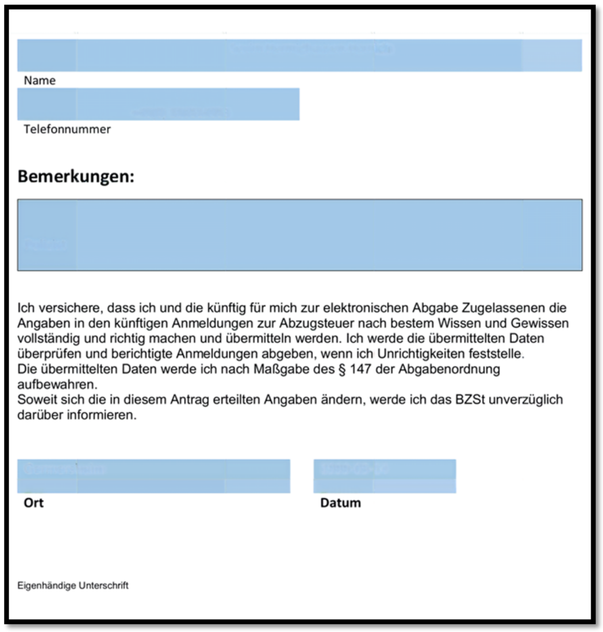
The experiments were conducted using the Python programming language, and both U-Net and Res-Net generators were used to train the dataset. Results of the experiments with varying hyper-parameters can be found in the table below.

Table 1: Experiments result

Generator-type	Tile Size	Epochs	Learning-Rate	D-Filters	G-Filters	SSIM score
U-Net	(128*128*3)	200	0.0002	32	32	0.02251853
U-Net	(128*128*3)	400	0.0002	32	32	0.00943532
Res-Net	(128*128*3)	400	0.0002	64	32	0.00027418
Res-Net	(128*128*3)	500	0.0002	64	32	0.00015080
Res-Net	(128*128*3)	500	Learning-Rate-scheduler	64	32	0.00007836
Res-Net	(512*512*3)	500	Learning-Rate-scheduler	64	32	0.00003809

The average SSIM score between the original empty form image and the generated form image was 0.00003809, indicating successful erasure of dynamic content.

The original and generated empty forms can be observed in Figures 3a and 3b, respectively.



Name

Telefonnummer

Bemerkungen:

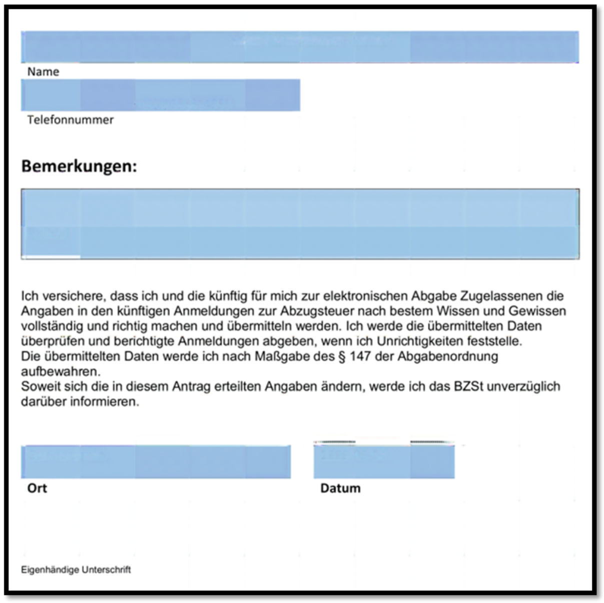
Ich versichere, dass ich und die künftig für mich zur elektronischen Abgabe Zugelassenen die Angaben in den künftigen Anmeldungen zur Abzugsteuer nach bestem Wissen und Gewissen vollständig und richtig machen und übermitteln werden. Ich werde die übermittelten Daten überprüfen und berichtigte Anmeldungen abgeben, wenn ich Unrichtigkeiten feststelle.
Die übermittelten Daten werde ich nach Maßgabe des § 147 der Abgabenordnung aufbewahren.
Soweit sich die in diesem Antrag erteilten Angaben ändern, werde ich das BZSt unverzüglich darüber informieren.

Ort

Datum

Eigenhändige Unterschrift

(a) Original Empty Form



Name

Telefonnummer

Bemerkungen:

Ich versichere, dass ich und die künftig für mich zur elektronischen Abgabe Zugelassenen die Angaben in den künftigen Anmeldungen zur Abzugsteuer nach bestem Wissen und Gewissen vollständig und richtig machen und übermitteln werden. Ich werde die übermittelten Daten überprüfen und berichtigte Anmeldungen abgeben, wenn ich Unrichtigkeiten feststelle.
Die übermittelten Daten werde ich nach Maßgabe des § 147 der Abgabenordnung aufbewahren.
Soweit sich die in diesem Antrag erteilten Angaben ändern, werde ich das BZSt unverzüglich darüber informieren.

Ort

Datum

Eigenhändige Unterschrift

(b) Predicted Empty Form

Figure 3: original empty form image and generated form image

ACKNOWLEDGMENTS

Working on my Master's thesis at Lang.Tec Company in Germany was an outstanding opportunity for me.

First and foremost, I am deeply grateful to my supervisor, Dr. Kilian Foth, Dr. Patrick McCrae, and Prof. Alessandro Savino, for their continuous support throughout my thesis study and research. Their patience, motivation, dynamism, and vast knowledge were invaluable to me, and it was a great privilege to work and study under their guidance.

I would like to express my appreciation to my parents, two sisters, and friends for their constant encouragement and support.

Additionally, I would like to extend my gratitude to my colleagues during my internship and thesis at Lang.Tec Company for their collaboration, support, and valuable guidance.

(a) Mahsa Farjoo

(b) Dr. Kilian Foth

(c) Dr. Patrick McCrae

(d) Prof. Alessandro Savino

Contents

List of Figures

List of Tables

List of Abbreviations

- **ML** Machine Learning
- **DL** Deep Learning
- **DNN** Deep Neural Networks
- **ANN** Artificial Neural Networks
- **RNN** Recurrent Neural Networks
- **CNN** Convolution Neural Networks
- **BN** Batch Normalization
- **GAN** Generative Adversarial Networks
- **SSIM** Structural SIMilarity

Chapter 1

Introduction

1.1 Motivation

Today, many organizations across various fields have to process, analyze and extract useful information from thousands of documents. These documents may contain empty forms, filled-in customer forms, contracts, and various other data. Due to the large volume of data, it is not feasible to rely solely on human efforts to extract relevant information.
The dataset used in this thesis consists of German tax forms. The creation of cleared forms was a largely manual and slow process, so there is a need for automated information extraction. This approach has been successfully employed by Lang.Tec Company.
The objective of this project is to automate the cleaning of forms with colored backgrounds. The forms comprise of Tax forms-2019 written in the German language. Since there are a large number of images, they are divided into tiles with a size of 512*512. These tiling images are used to train a deep learning model to distinguish dynamic and static content. The model then erases dynamic content from the forms, and the result should be empty tiles. These empty tiles can be placed next to each other to recreate empty forms.
These empty forms will serve as templates to create synthetic document variants with full annotations to be used in training machine learning models for information extraction. The goal of this project is to obtain training material for machine learning that can automatically extract information.

1.2 Thesis Objectives

The current research was conducted at LangTec in Hamburg, Germany. LangTec is a research-driven technology provider of natural language processing (NLP) solutions and automated text generation (NLG). They develop innovative language technology solutions for the efficient processing of large amounts of text and data.
One of the project opportunities at LangTec is in the field of image processing. The goal of this project is to automate the cleaning of forms with colored backgrounds for dynamic contents and white backgrounds for static contents, by erasing dynamic content. However, this work has been done in erasing dynamic content from documents, which will be explained in Chapter 3.

The main goals for this project include the following items:

1. To train document scanning and information extraction systems using machine learning.
2. To create training sets consisting of pairs of filled forms with labels, which are required for machine learning.
3. To automatically construct training data using empty forms.
4. To develop a method for obtaining empty forms from used forms, as it is a valuable component for achieving the above goals.

1.3 Thesis Structure

The rest of the thesis is organized as follows:

- Chapter 2 provides a review of machine learning and deep learning, including the learning process in deep learning and neural networks.
- Chapter 3 presents a literature review on partial text removal and the architectures used in previous research.
- Chapter 4 describes the data and tools used in this research, including data preparation and preprocessing. The performance metrics used in this research are also introduced.
- Chapter 5 presents the model architectures used in this research. Hyper-parameter tuning is used to find the best set of hyper-parameters, and the results obtained from the architectures are presented.
- Chapter 6 proposes a solution for future work.
- Chapter 7 includes the implemented code in Python.

Chapter 2

Deep Learning Review

2.1 What is Machine Learning?

Machine Learning can be defined in various ways, such as:

- According to James Murdoch et al., Machine Learning models are capable of learning complex patterns and making predictions, which is a remarkable achievement [13].
- Peter Flach defines Machine Learning as a set of methods for identifying data patterns and characteristics, allowing for the prediction of new data behaviour. Data analysis methods are used to process and extract useful information from the data [17].
- Machine learning is the scientific field that uses the most effective techniques to extract information and patterns from raw data.

2.2 What is Deep Learning?

Deep Learning can be defined in various ways, such as:

- Deep Learning is a machine learning algorithm that employs multiple layers of processing units to learn unstructured data features and extract patterns from input data [07].
- It is a type of machine learning that utilizes artificial neural networks with multiple layers of analyzing and processing to extract features from data.
- Deep Learning is a machine learning and artificial intelligence (AI) technique that emulates how humans acquire certain types of knowledge. It is a vital component of data science, which encompasses statistics and predictive modelling [14].

To gain a better understanding of Deep Learning and its significance in our lives, we must first become acquainted with different types of data: unstructured data and structured data [07].

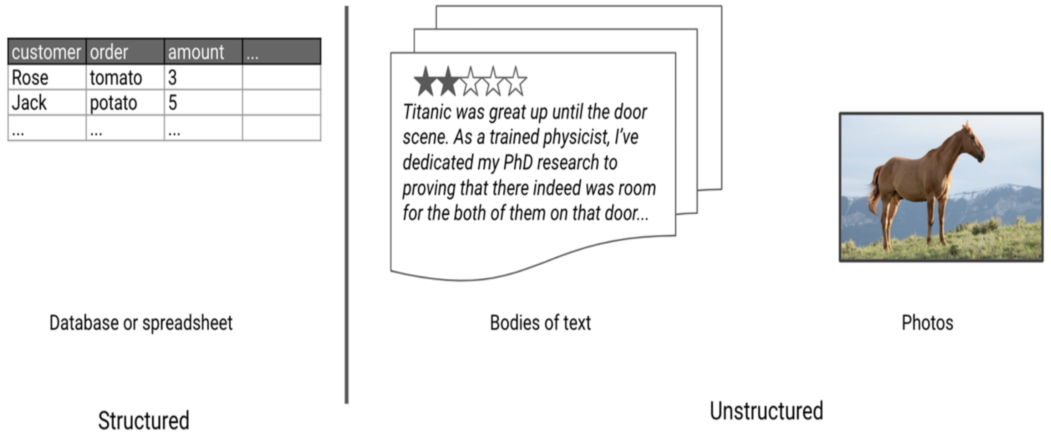

customer
order
amount
...
Rose
tomato
3
Jack
potato
5
Titanic was great up until the door scene. As a trained physicist, I've dedicated my PhD research to proving that there indeed was room for the both of them on that door...
Database or spreadsheet
Bodies of text
Photos
Structured
Unstructured

Figure 2.1: The sample of structured data and unstructured data [07]

2.3 Structured and Unstructured Data

Machine learning algorithms use two different types of data: "unstructured data" and "structured data".

- **structured data:** These are tabular data that contain rows and columns. Each column represents features for each observation. For example, source of income, job, interests, and the number of websites visited in a recent month are features that help to predict if the customer will follow a particular on-line service in the next month. We can use structured data of features to train data with a random forest or with logistic regression to predict the binary response variable: did the person follow (0) or not (1)? Each feature includes information about the row data, and the model can learn how to convert and extract these responses from features.

- **Unstructured data:** Unstructured data are like images, audio, and text. Since the data does not arrive in columns of features, it is considered unstructured data, as shown in Figure 2.1 [07].

  For generative modelling, we can use deep learning with unstructured data. Most often, we want to generate unstructured data such as new images or original strings of text, which is why deep learning has had such a profound impact on the field of generative modelling [07].

2.4 Deep Neural Networks

Lots of deep learning methods are artificial neural networks (ANNs) with multiple hidden layers. A Neural Network (NN) is a network of artificial neurons that simulates how the human brain operates. A real neural network in the brain is shaped by many neurons in different layers. Similarly, the artificial network is made up of several nodes distributed in various layers. Each node implements an algorithm that guides the machine to identify patterns in the dataset. Systems learn instead of receiving specific instructions given by programmers and perform tasks by evaluating samples. Today, NNs are used to solve many problems in different areas and industries. For example, they are essential in computer training for face recognition, driving, demand forecasting, and text erasure [18].

Let's examine how a deep neural network can generate an output from a given input. A deep neural network comprises multiple layers, each of which contains several neurons or units that are connected to the neurons or units in the previous layer through a set of weights [07].

There are different types of layers, but one of the most important is the dense layer that directly connects all neurons or units in the layer to every neuron or unit in the previous layer. The neurons or units in each layer can capture different aspects of the original input. For instance, the first layer comprises neurons or units that detect basic features of the input data, such as an image. The output of the first layer is then transmitted to the neurons or units in the second layer, which use this information to identify more complex features [07].

2.5 Different types of Neural Networks in Deep Learning

There are three important types of neural networks that are represented as follows:

2.5.1 Artificial Neural Networks (ANN)

Artificial Neural Network, or ANN, is a group of units/neurons at each layer. ANNs are also known as Feed-Forward Neural Networks because inputs are processed only in one direction/path:

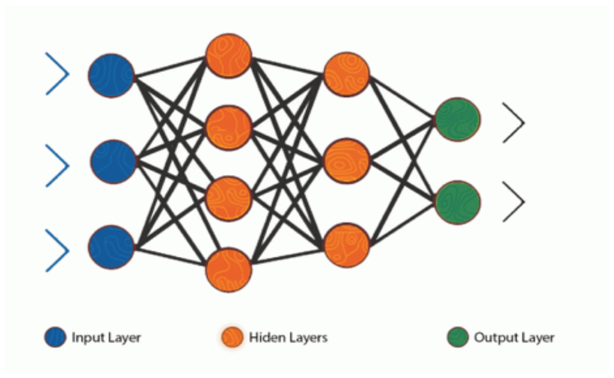


Figure 2.2: Representing the neurons at each layer Artificial Neural Network (ANN)[08]

As you can see here, Artificial Neural Networks consist of 3 layers – Input, Hidden, and Output. The input layer accepts the inputs, the hidden layer processes the inputs, and the output layer produces the result. Essentially, each layer tries to learn certain weights.

2.5.2 Recurrent Neural Networks (RNN)

As you can see here, Recurrent Neural Networks have a repetitive connection.

By using loops and repetitive connections, the network becomes like a state machine and learns new things from the memory of the past. Recurrent NN works very well in time-varying systems, but the training process becomes much more complicated [19].

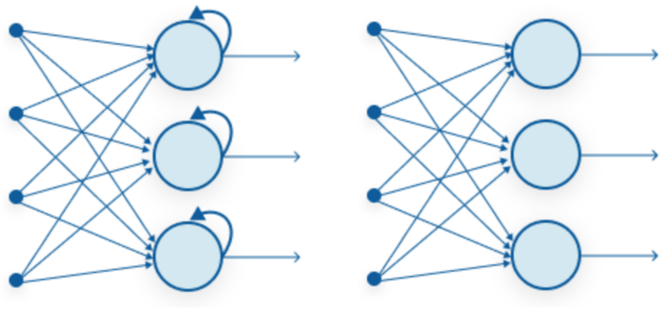


Figure 2.3: Representing the neurons at each layer Recurrent Neural Networks (RNN) [08]

2.5.3 Convolution Neural Networks (CNN)

Convolutional neural networks are now widely used in various applications and domains for image and video processing projects.
The building blocks of convolutional neural networks include filter kernels, which are used to detect and extract important features from input images through the convolution operation. These filters play a crucial role in detecting and extracting information from the input image. Figure 2.4 represents the original image before and after the convolutional neural network (CNN) process.

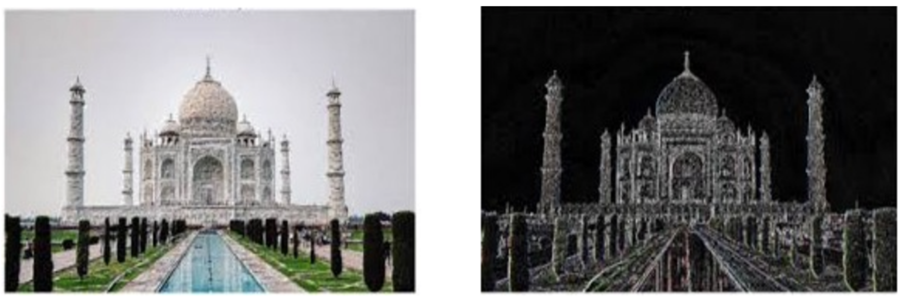


Figure 2.4: Representing the original image before and after Convolution Neural Networks (CNN)[08]

2.5.4 Learning Process of Neural Networks

The learning process in an artificial neural network depends on four factors [20, 21]:

1. **Number of layers in the network:** This factor can be a single layer or multiple layers.
2. **Feed forward or Recurrent:** The second factor should be either a feedforward neural network or a recurrent neural network.
3. **The number of nodes in layers:** The third factor is determined by the number of nodes in the input layer, which is equal to the number of features in the input dataset. The number of nodes that appear in the output layer will depend on the possible results.

4. **Weight of interconnected nodes:** The last important factor is the weight of interconnected nodes. Deciding on the amount of weight attached to each connection among each node so that a particular learning problem can be properly solved has been one of the difficult tasks.

In summary, this process involves feeding labeled data to the network and changing the network's parameters, such as learning rate, epoch, and cost function, to bring the results close enough to the desired output [20, 21].

2.6 Keras and TensorFlow

2.6.1 Keras

Keras is a high-level Python library for building neural networks. It is highly flexible and has a very user-friendly API. Keras provides numerous useful building blocks that can be combined to create highly complex deep learning architectures through its functional API.

2.6.2 TensorFlow

TensorFlow is an open-source Python library for machine learning developed by Google. It is now one of the most widely used frameworks for building machine learning solutions, with particular emphasis on deep learning.

2.7 Building the model

There are two different ways to build the structure of neural networks in Keras, as defined by David Foster [07]: the Sequential model and Functional API.

- Sequential model : " A Sequential model is useful for defining a linear stack of layers, where one layer follows directly from the previous layer without any branching."
- Functional API : "To build networks with branches, we need to use the Functional API, which is a lot more flexible. As you know, there are different types of layers: Input Layer, Flatten Layer, Dense Layer, Activation Function Layer, BatchNormalization/InstanceNormalization Layer, Convolutional Neural Networks Layer, and more [07]."

2.7.1 Input Layer

In a neural network, the input layer is responsible for taking in the input elements, such as data or images, and passing them through to the next layer for processing [09].

2.7.2 Flatten Layer

To flatten the input into a vector, we can use a Flatten layer because the input value of a Dense layer should be flat rather than a multidimensional array. However, other layer types do require a multidimensional array [07, 10].

2.7.3 Dense Layer

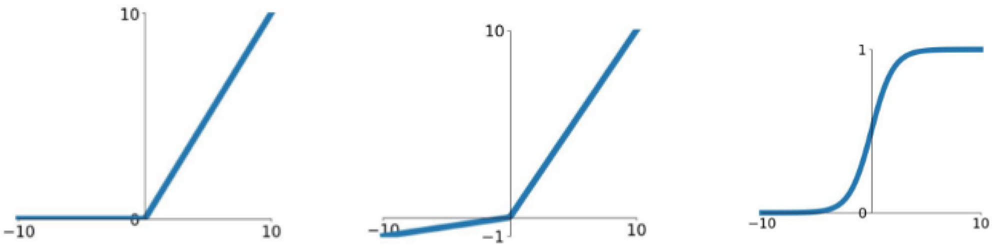
The Dense layer is one of the most important layer types in any neural network. It consists of multiple units, and each unit can connect to every unit in the previous layer using a single connection with a weight, which may be positive or negative.
The output of the Dense layer is the weighted sum of the inputs from the previous layer, and before passing to the next layer, a non-linear activation function can be applied [07, 11].

2.7.4 Activation

The activation function is not only used to generate a linear combination of input, but also to learn complex behavior. In his book "Generative Deep Learning," David Foster presents an interesting definition of the activation function and the different types of it: "The activation function is critically used to ensure that the Neural Network can learn complex functions [07]."
There are different kinds of activation functions in a neural network model, but the most important ones are [07]:

1. **ReLU:** The ReLU activation function is defined to be zero when the input is negative and equal to the input otherwise.
2. **LeakyReLU:** The LeakyReLU activation function is very similar to ReLU, with one key difference: while the ReLU activation function returns zero for input values less than zero, the LeakyReLU function returns a small negative number proportional to the input.
3. **Sigmoid:** The sigmoid activation function is useful if you want the output of the layer to be scaled between 0 and 1. The softmax activation is useful if you want the total sum of the output from the layer to equal 1.
4. **Softmax:** Softmax is a mathematical function that converts a vector of numbers into a vector of probabilities, where the probabilities of each value are proportional to the relative scale of each value in the vector.

You can see a comparison of the ReLU, LeakyReLU, and sigmoid activation functions in Figure 2.4.



(a) ReLU **(b)** LeakyRELU **(c)** Sigmoid

Figure 2.5: The ReLU, LeakyReLU, and sigmoid activation functions [07]

2.7.5 Convolutional Layers

You can see a sample of a filter in Figure 2.6. We can apply it to a convolution layer by multiplying the filter with different portions of the image and summing the result. If we use this filter for the whole image, from left to right and top to bottom, we can receive a new array that represents a set of features of the input. There are different types of filters, and sometimes it's possible to use multiple filters rather than just one [07, 12]. Figure 2.6 shows a 331 portion of an image being convoluted with a 331 filter (or kernel).

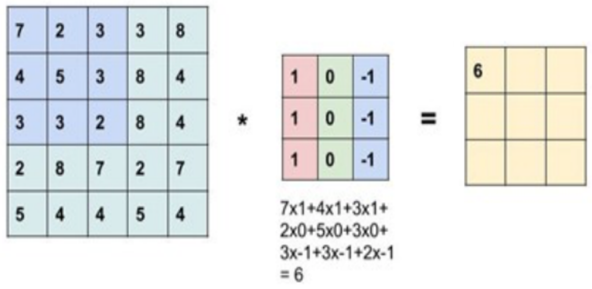
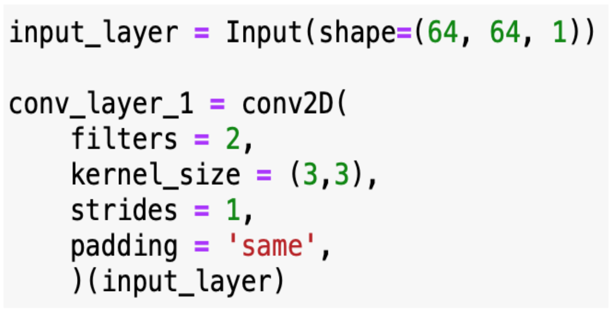


Figure 2.6: The convolution operation with a 3*3*1 filter(or kernel)[12]

In conv2D layer, there are some parameters. Figure 2.7 represents different parameters in conv2D layer.



```
input_layer = Input(shape=(64, 64, 1))

conv_layer_1 = conv2D(
    filters = 2,
    kernel_size = (3,3),
    strides = 1,
    padding = 'same',
    )(input_layer)
```

Figure 2.7: The Python code in Keras represents a sample of conv2D layer

- **Strides:** The Strides parameter is the step size used by the layer to move the filters across the input. Increasing the stride reduces the size of the output tensor.
- **Padding:** The 'same' input parameter pads the input data with zeros so that the output size from the layer is exactly the same as the input size when the stride is 1.

2.7.6 Batch Normalization

David Foster, in his book "Generative Deep Learning" [07], explains why we have to use the Batch Normalization Layer: "When training a deep neural network, a common problem that should be addressed to ensure that the weights remain within a reasonable range of values. If they start to become too large, this is a sign that your network is suffering from what is known as the exploding problem. BatchNormalization is a solution that solves this problem."

A batch normalization layer is implemented by calculating the mean and standard deviation of each of its input channels across the batch and normalizing by subtracting the mean and dividing by the standard deviation [07]. In Keras, the BatchNormalization layer implements the batch normalization functionality: BatchNormalization (momentum=0.9) [07].
In keras, the BatchNormalization layer implements the batch normalization functionality: BatchNormalization (momentum = 0.9) [07].

- **The momentum:** Momentum parameter is the weight given to the previous value when calculating the moving average and moving standard deviation [07].

2.7.7 Instance Normalization

In the book "Generative Deep Learning" [07], the reason why some models use Instance Normalization Layer instead of Batch Normalization Layer is explained. Certain models utilize Instance Normalization layers as they normalize each observation individually, rather than as a batch. Unlike the Batch Normalization layer, it does not require calculating mu and sigma parameters as a running average during training. This is because at test time, the layer can normalize per instance in the same way as it does during training [11].

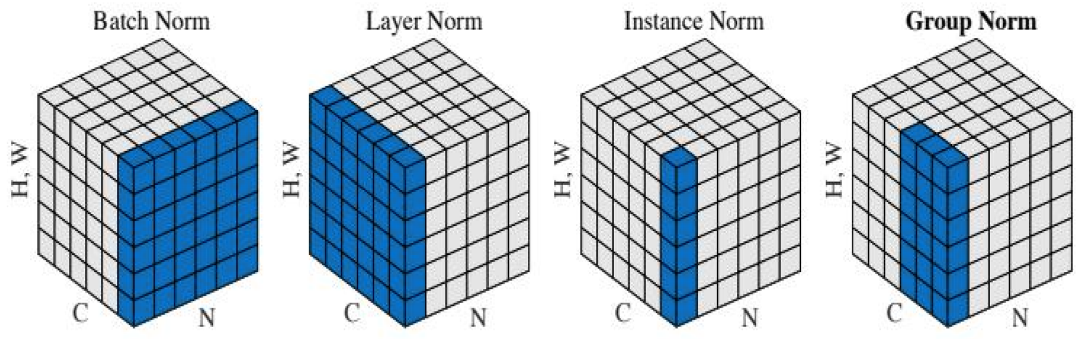


Figure 2.8: Four different normalization methods [07]

2.7.8 Droupout Layers

To implement this layer during training, a number of neurons/units take a value of zero and are randomly selected. You can see the process of a dropout layer in Figure 2.8 [10].

- Dropout layers are most commonly used after Dense layers since these are most prone to overfitting due to the higher number of weights, although you can also use them after convolutional layers.

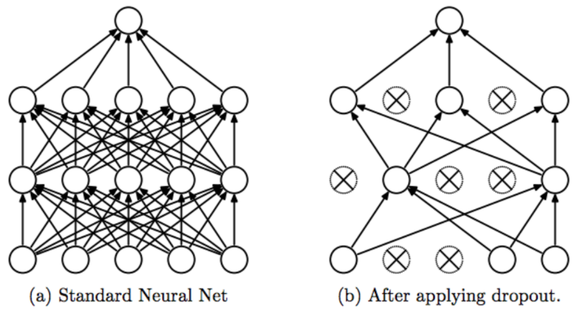


Figure 2.9: A dropout layer [07]

2.8 Introduction GAN and CycleGAN

As previously described, the goal of this project is to generate blank forms from completed forms. To achieve this objective, I intend to use the GAN method.

2.8.1 Generative Adversarial Networks (GANs)

Generative Adversarial Networks (GANs) are one of the most popular topics in Deep Learning and a type of Neural Networks used for Unsupervised learning. There are many definitions of Generative Adversarial Networks. It is represented as follows:

- Z. Pan et al. provided a definition of generative models in their work [16]. "Artificial intelligence aims to enable machines to comprehend the complex world in the same way as humans. Researchers in machine intelligence have proposed generative models to describe the world in terms of probability and statistics.
  Generative models can be divided into three categories: Generative Adversarial Networks (GANs), Variational Autoencoder (VAE), and AutoRegressive Networks. VAE is a probabilistic graphical model that attempts to model the probability distribution of data. However, its final probabilistic simulation is biased, resulting in generating more blurred samples than GANs. PixelRNN [11] is an autoregressive network that translates image generation into the problem of pixel prediction and generation, processing each pixel one by one. In contrast, GANs process the sample in one shot, allowing them to produce samples faster than PixelRNN.
  As probabilistic generative models, traditional generative models relying on the natural interpretation of data cannot be trained and applied when the probability density is not provided. However, GANs can still be used in this situation since they introduce a very clever internal adversarial training mechanism."

- The first word, "Generative," means that there is a network that constantly generates new data. The second word, "Adversarial," means that it involves two networks opposing each other. "Network" simply means an ordered set of data that is used to generate new data. As you can see in Fig. 2.10, GANs consist of two networks: a Generator G(x) and a Discriminator D(x). They both play an adversarial game, where the generator tries to fool the discriminator by generating data similar to that in the training set, while the discriminator tries not to be fooled by distinguishing between fake and real data. They work simultaneously to learn and train complex data, such as audio, video, or image files.

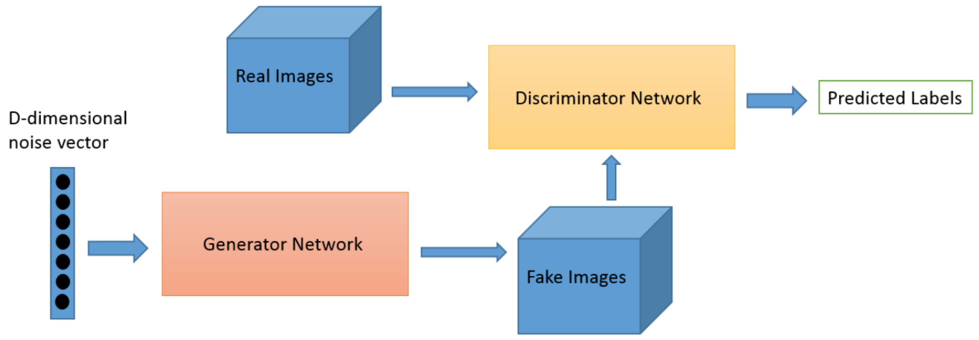


Figure 2.10: Inputs and outputs of the two networks in a GAN

Image-to-image translation refers to a set of vision and graphics problems that involve learning how to map an input image to an output image using a training set of image pairs. However, in many cases, it's not feasible to obtain paired examples. In such situations, we propose an approach for learning how to translate an image from a source domain X to a target domain Y without relying on paired data. Our goal is to learn a mapping $G : X \rightarrow Y$ such that the distribution of images from G(X) is indistinguishable from the distribution Y using an adversarial loss. As you can see in Fig. 2.11, paired training data consists of training examples xi,yi, where the correspondence between xi and yi exists.

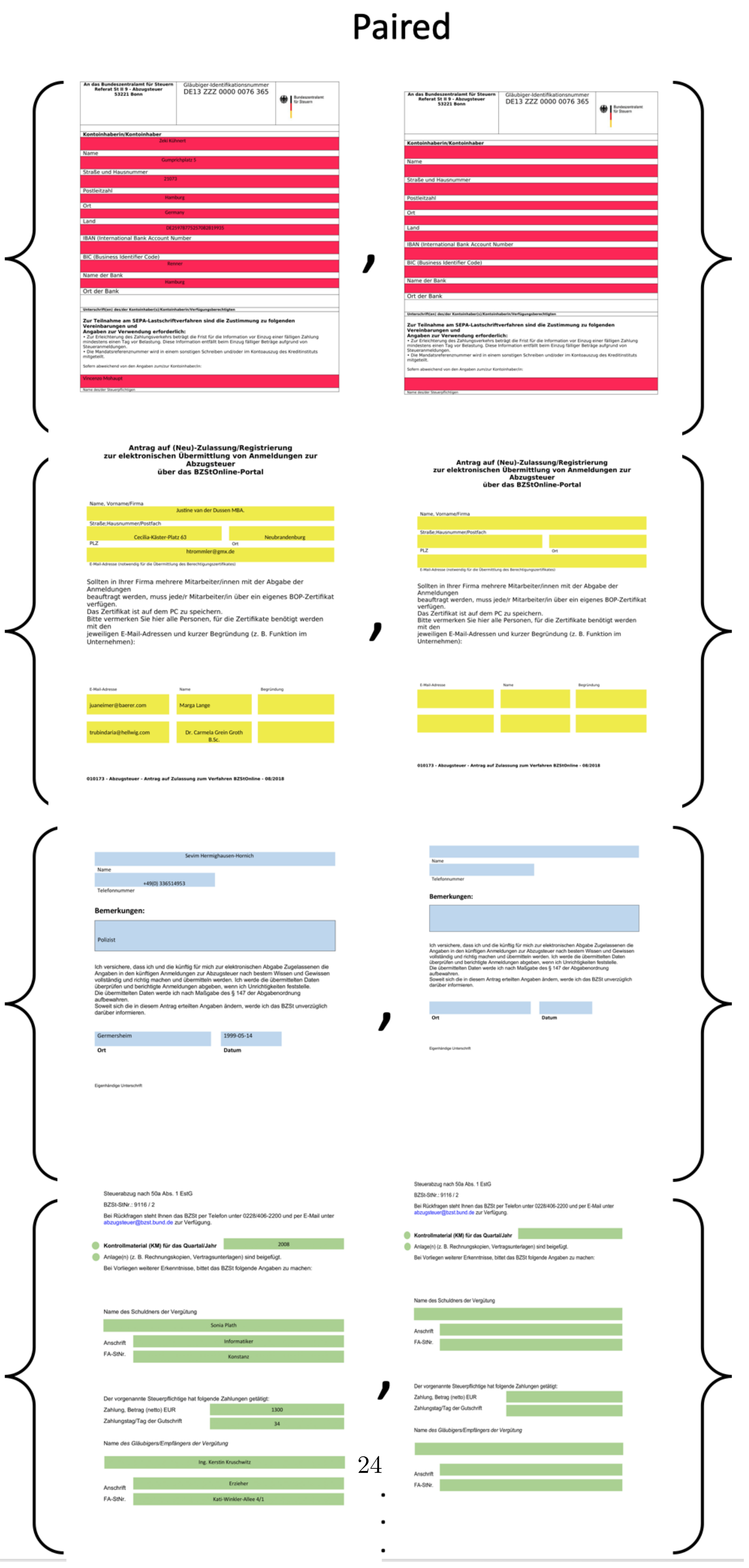


Figure 2.11: Paired training data, where the correspondence between xi and yi

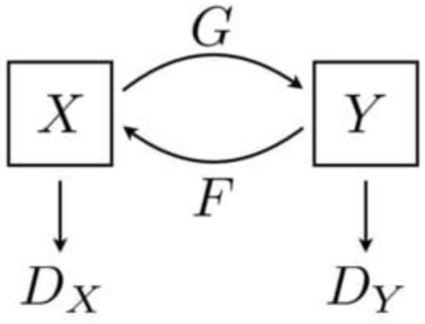
The architecture intended for this work is CycleGAN, which is chosen for its ability to perform two key steps:

- Distinguishing and erasing dynamic contents
- Reconstructing the background color and structures

David Foster presented a definition in his book "Generative Deep Learning" [07], stating that CycleGAN is composed of four models: two generators and two discriminators.

- The first generator, G-AB, converts images from domain A into domain B, generating empty forms from filled forms.
- The second generator converts images from domain B into domain A, reconstructing the background color and structures. To accomplish this task, two discriminators must be trained to determine the authenticity of the generated images.
- The first discriminator, d-A, is trained to differentiate between real images from domain A and fake images produced by generator G-AB.
- The second discriminator, d-B, is trained to differentiate between real images from domain B and fake images produced by generator G-AB.

Figure 2.12 illustrates the relationship between the four models.



$$\begin{tikzcd}
\boxed{X} \arrow[r, bend left, "G"] \arrow[d] & \boxed{Y} \arrow[l, bend left, "F"] \arrow[d] \\
D_X & D_Y
\end{tikzcd}$$

Figure 2.12: The diagram of the four CycleGAN models

2.8.2 Loss function

Both the generator and the discriminator have loss function. In this case, we call it Structural SIMilarity (SSIM).

- The Structural SIMilarity (SSIM) index is a perceptually motivated metric which decomposes the similarity measurement task into three comparison functions: luminance (l), contrast (c) and structure (s). Given two signal x and y, the three comparison functions are defined as equations 2.1, 2.2, 2.3:

$$L(x,y) = \frac{2\mu_x\mu_y + C_1}{\mu_x^2 + \mu_y^2 + C_1} \tag{2.1}$$

$$c(x,y) = \frac{2\sigma_x\sigma_y + C_2}{\sigma_x^2 + \sigma_y^2 + C_2} \tag{2.2}$$

$$s(x,y) = \frac{\sigma_{xy} + C_3}{\sigma_x\sigma_y + C_3} \tag{2.3}$$

Here μ_x, σ_x^2 and σ_{xy} are the mean of x, the variance of x, and the covariance of x and y, respectively. C_1, C_2 and C_3 are constants and stabilize the divisions. Then the general form of the SSIM index between x and y is defined in equation 2.4:

$$SSIM(x,y) = [L(x,y)]^\alpha . [c(x,y)]^\beta . [s(x,y)]^\gamma \tag{2.4}$$

where α, β and γ are parameters to control the relative omportance of the three comparison functions. By setting $C_3 = C_2$ / 2 and $\alpha = \beta = \gamma = 1$, the formula of the SSIM index can be reduced to the form shown in equation 2.5:

$$SSIM(x,y) = \frac{(2\mu_x\mu_y + C_1)(2\sigma_{xy} + C_2)}{(\mu_x^2 + \mu_y^2 + C_1)(\sigma_x^2 + \sigma_y^2 + C_2)} \tag{2.5}$$

For image quality evaluation, the SSIM index is typically calculated using a sliding Gaussian window. The window can be displaced pixel-by-pixel on the image to create an SSIM quality map of the image, whose mean defines the SSIM index of two images X and Y:

$$MSSIM(x,y) = \frac{1}{M}\sum_{i=0}^{M} SSIM(X_i, Y_i) \tag{2.6}$$

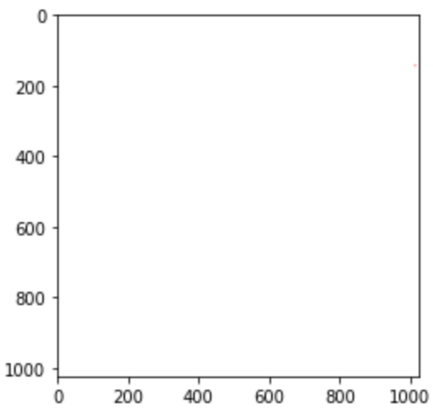
where X_i and Y_i are the image contents of the $i^t h$ local window; and M is the number of local windows in the image.

2.8.3 The worth value and best value for SSIM loss

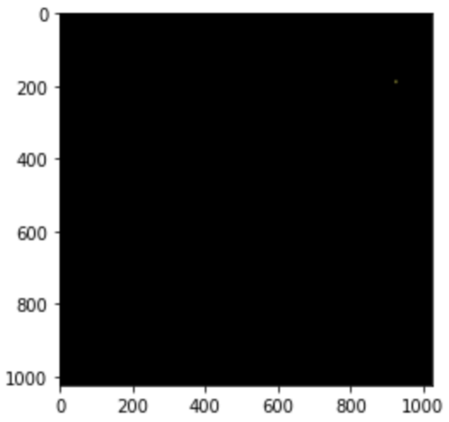
For better understanding, in the following step, I want to show you the worst value and best value for SSIM loss.

- The worth value for SSIM is between the full black image and full white one and it is equal to 0.133283.
- The best value of SSIM loss is between two same images and it is equal to 0.00.

It means that if two images are the same, the SSIM loss between them should be zero. The SSIM loss between two different images will have a value between one and zero.



(a) Full white image



(b) Full black image

Figure 2.13: The worth value for SSIM between full black and full white images

Chapter 3

Literature Review

Recognizing and extracting text from an image are typical computer vision applications, such as image translation or background reconstruction.

3.1 Previous work on clearing text

- In 2017, Nakamura et al. proposed a method, as shown in Figure 3.1, where a single-scale sliding window was applied to the original input images. To implement this method, they resized all of the images to 64x64. The dataset was then fed into a pre-trained DNN, and the size of each output result was also 64x64. However, there was a problem with the overlap regions. To solve this problem, they considered only the center part of the output to be valid and put it back into the original location. The size of the center part was 32x32 pixels, and after this process, a single text-hidden image was generated. The details of this method were described in their paper [03].

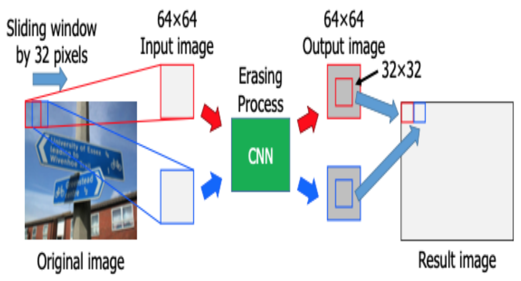


Figure 3.1: The proposed method for scene text erasing [03]

A feedforward DNN includs two parts:

  - **With half convolution part**
  - **With half deconvolution part**

The architecture of the DNN is as shown in Figure 3.2 and follows these specifications. The convolution part includes four convolutional layers, with the following details and parameter values:

1. **Filter Size:** Filter Size: The filter size of each convolutional layer is 4×4.
2. **Stride and Padding Size:** Stride and Padding Size: The stride step and padding size are set to 2 and 1, respectively.
3. **Each layer:** Therefore, in each layer, the size of the feature maps is halved compared to the previous ones.

The deconvolution part has the same structure, but replaces the convolutional layers with deconvolution layers, with the following details and parameter values:

1. **Filter Size:** The filter size of each convolutional layer is 4×4.
2. **Stride and Padding Size:** The stride step and padding size is set to 2 and 1, respectively.
3. **The size of feature maps:** Thus, as the layers go deeper, the size of the feature maps is doubled. Due to the image reduction caused by convolution and the image enlargement caused by deconvolution, the output image has the same size as the original image.

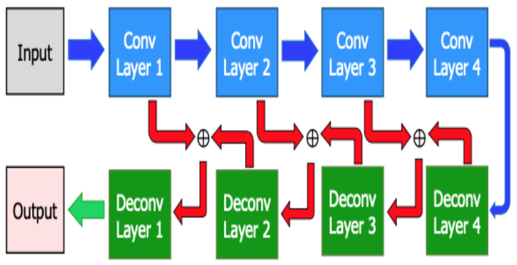


Figure 3.2: The architecture of DNN in their proposed method [03]

- In 2020, Tursun et al. presented a one-stage text removal method structure, called MTRNet++, in their paper [01]. This method is composed of three branches: mask-refine (GMR), coarse-inpainting (GCI), and fine-inpainting (GFI). The input to all branches is a concatenation of an image and a mask, which can be provided to the network, but is not mandatory and can be an empty mask. In the latter case, the network will automatically detect and remove the text regions. Figure 3.3 shows the structure of the MTRNet++ one-stage text removal method, where the generator G is depicted.

  Figure 3.3 represents the structure of one-stage text removal method, MTRNet++. It includes generator, G, is composed of three branches:

  1. **Mask-refine (GMR) branch:**
  2. **coarse-inpainting (GCI) branch:**
  3. **Fine-inpainting (GFI) branch:**

  All branches have a similar architecture. Each branch has a front-end, a mid-section, and a back-end.

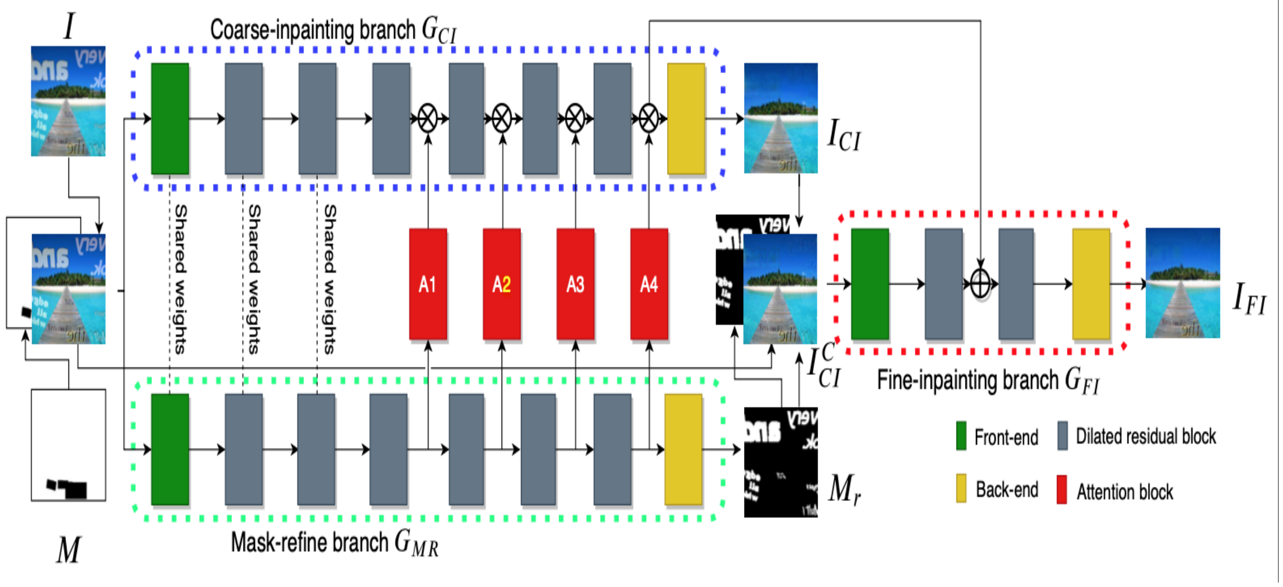


Figure 3.3: The structure of proposed one-stage text removal method, MTRNet++[01]

The front-end: is composed of down-sampling convolutional layers that reduce the image dimensions by a factor of two.

The back-end: while the back-end is a transposed version of the front-end.

The mid-section: is combined of a set of residual blocks.

GCI and GMR each have six residual blocks. Their first two residual blocks share weights, while there are attention blocks connecting the last-four residual blocks. MTRNet++, therefore, has a larger receptive field compared to MTRNet, which is important when inpainting text with a large font size [01].

3.2 Watermark Removal with using cycleGANs

- In 2019, Li et al, in their paper [02], presented their watermark removal framework which was implemented by using cycleGANs method. For reconstructing the back-door information in original image, commonly the cycleGANs are used to reconstruct. In that work, they presented the idea of the cGANs and suggested a framework for watermark removal based on cycleGANs. The architectures of their watermark removal cycleGANs is illustrated in Fig. 3.3.

  That network is implemented by using a generator and a discriminator.

  - In the generator, for converting a watermarked image to a watermark-free one, we force a U-net architecture.
  - In the discriminator, they used a patch-based classifier to distinguish those recovered images generated by the generator from the ground-truth watermark-free images in a patch level

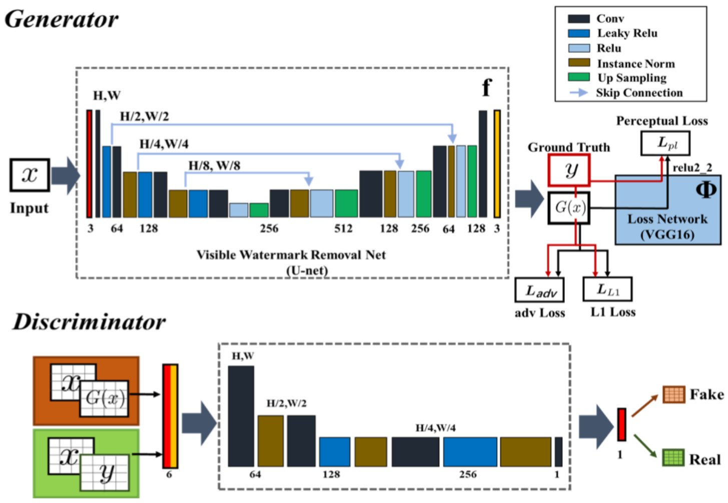


Figure 3.4: The architecture of visible watermark removal framework [02]

[02].

That network takes as input a watermarked image and used from the u-net generator G to generate a image without watermark.

- Fake images are images that are generated by a generator and cannot be distinguished as original images without a watermark.
- Ground-truth images, on the other hand, are original images without a watermark.

To make the generated images by the generator as similar as possible to the original watermark-free images, we introduce a new objective that combines the L1 loss, perceptual loss, and patch-based adversarial loss to restrict the training of the generator. Meanwhile, an adversarially trained discriminator (D) is employed to detect the "fake" images from the ground-truth images.

3.3 Previous work: Architecture For Artifact Segmentation

- In 2020, Ronny Huang et al. presented the architecture for artifact segmentation in their paper [04]. They utilized a fully convolutional network to create a binary segmentation mask that maps the raw input image. The mask indicates whether each pixel in the image is an artifact or not. After obtaining the mask, all the pixels on the mask indicating the presence of an artifact are set to 255 (white) on the input image. This effectively cleanses the image from artifacts. For training data, the researchers automatically compiled a corpus of dirty images paired with their segmentation masks for both printed and handwritten text [04].

  The network shown in Figure 3.5 is a simple U-net architecture that predicts a segmentation mask for each pixel, indicating whether it is an artifact or not. Convolutions are performed in blocks of two layers:

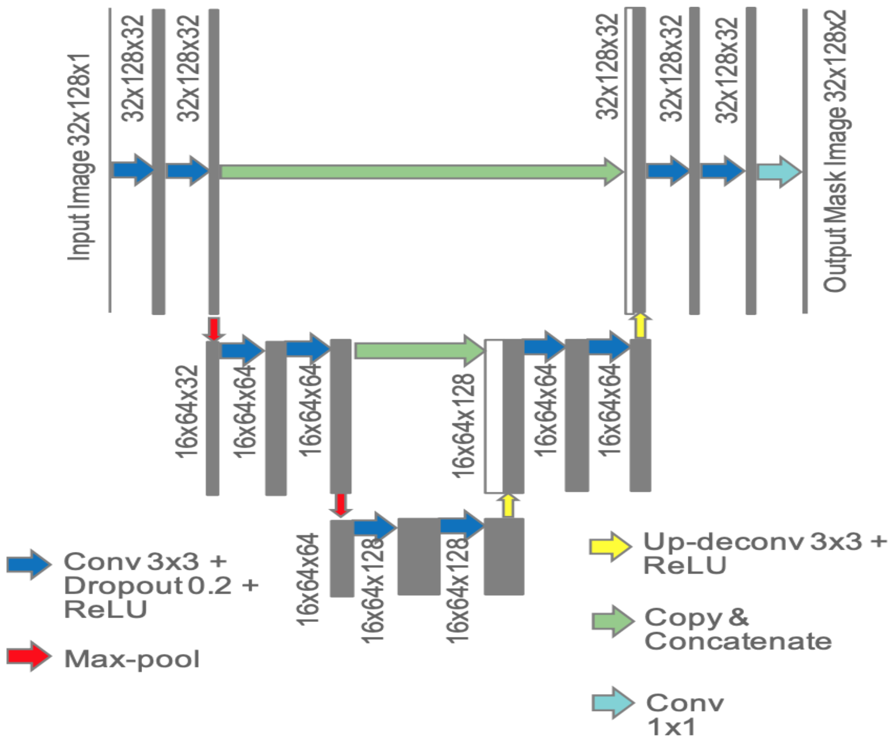


Figure 3.5: Architecture for artifact segmentation[04]

– where at the end of each block, maxpooling is used to downsample the feature map, and the number of channels is doubled.

– After two blocks, the feature maps are upsampled using deconvolution (or transposed convolution) for two blocks until the feature map resolution matches that of the original image.

– In each upsampling block, the first feature map is concatenated with the last feature map from the corresponding downsampling block, following the U-net method.

The training objective is to minimize the cross-entropy loss between the true segmentation mask and the predicted segmentation mask on a per-pixel basis, with averaging at the end. No regularizers are used in the training objective, and the RMSProp optimizer is utilized to minimize the objective [04].

To encourage translation and size invariance, data augmentation is applied in the form of resizing, followed by horizontal and vertical shifts of the image within a fixed 32x128 canvas [04].

3.4 Previous work: The proposed framework with (1) LSTM network (2) Encoder-Decoder and (3) Skip connections

- In 2019, Ghazal Mazaheri et al. [06] presented a method for detecting image manipulation that can differentiate between manipulated and non-manipulated regions in an image. Their framework consists of three main parts: an LSTM network, an Encoder-Decoder, and Skip connections. The resampled features of each patch are extracted and used as input for the LSTM network.

  1. **LSTM network:** The primary role of the LSTM cells is to learn the transition between manipulated and non-manipulated blocks in the frequency domain.
  2. **Encoder-Decoder:** The Encoder-Decoder architecture includes convolutions to learn spatial information.
  3. **Skip connections:** The Skip connections are used to take advantage of the early layers in the CNN that are rich in spatial details [06].

  Figure 3.6 demonstrates their proposed architecture.

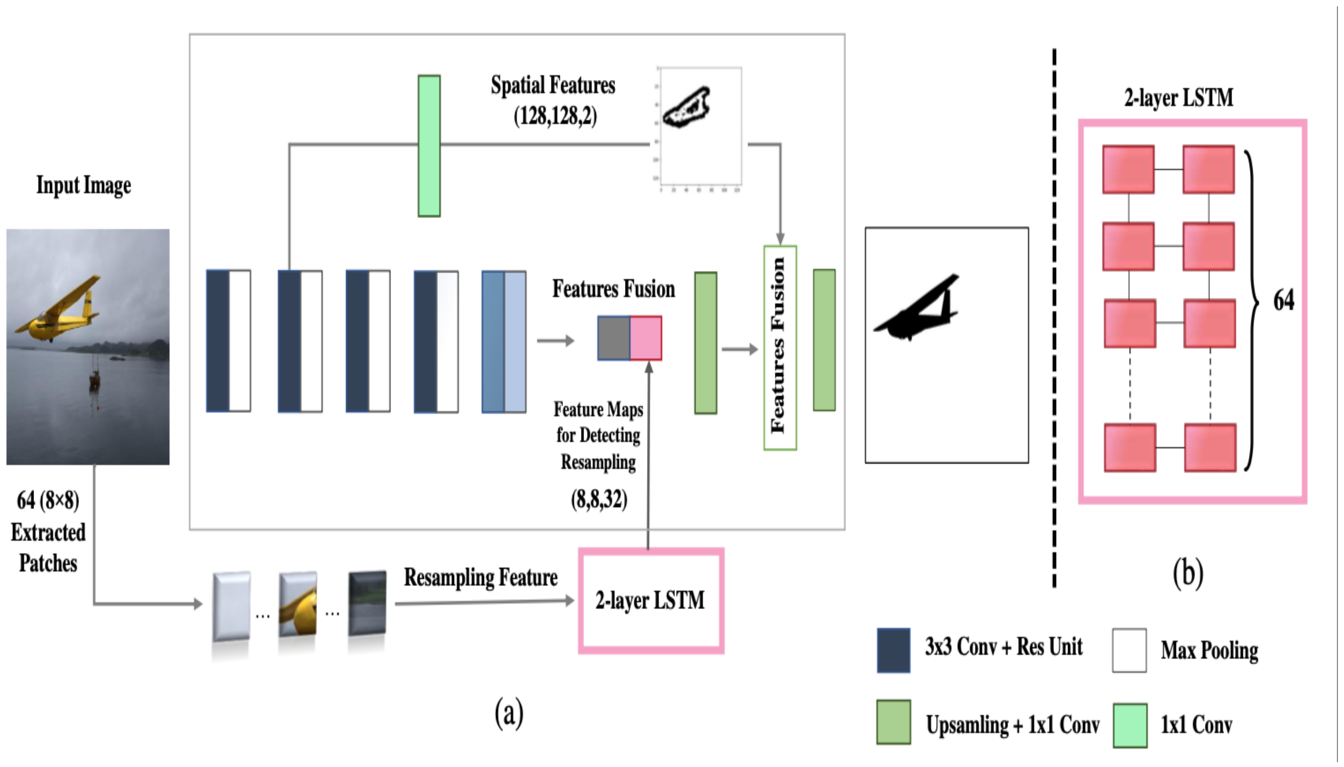


Figure 3.6: The proposed framework (LSTM-EnDec-Skip) [06]

3.5 Previous work: 3D U-Net Architecture

- In 2019, Nabil Ibtehaz and M. Sohel Rahman presented the 3D U-Net architecture in their paper [05]. The model comprises an encoder and a decoder pathway with skip connections between the corresponding layers, as shown in Figure 3.7. The U-Net architecture was extended through a few changes to 3D U-Net for segmentation.

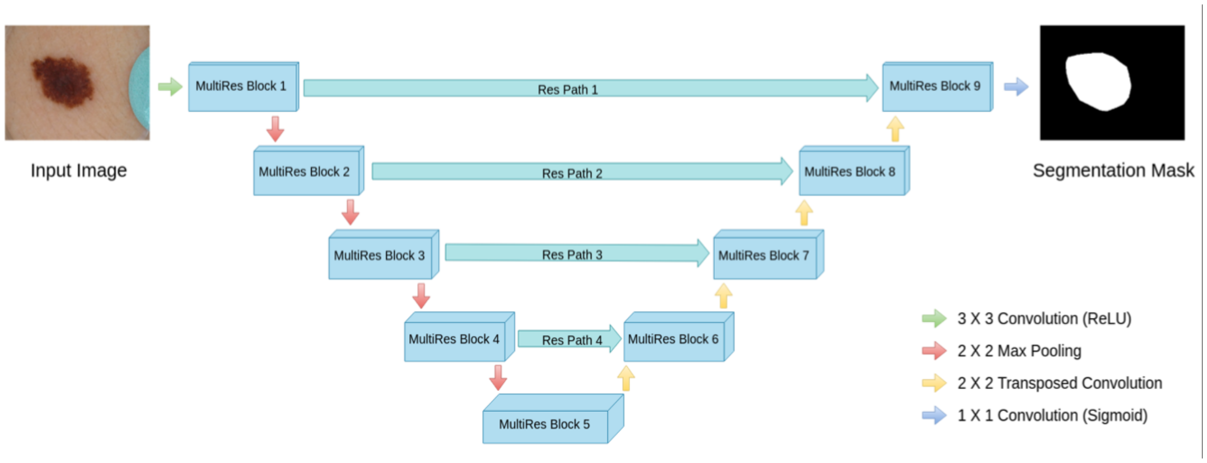


Figure 3.7: The 3D U-Net Architecture [05]

Figure 3.7, the U-Net architecture was extended through a few changes to 3D U-Net for segmentation.

In this new architecture, they used three-dimensional convolution, max pooling, and transposed convolution operations instead of using two-dimensional counterparts. Some new features are represented as follows:

1. The depth of new network was decreased by one due to the reduce the number of parameters.
2. The number of filters was increased to avoid bottlenecks.
3. The original U-Net did not use batch normalization, but the new 3D U-Net architecture used it to avoid hurting performance [05].

Chapter 4

Implementation and Preparing the Dataset

4.1 System Description

Figure 4.1 demonstrates an overview of our system during training time. The dataset includes various forms, each of which includes static and dynamic content in PDF format that must be converted to an image. Since the dimensions of these converted images are large, we have to divide them into acceptable sizes of 512 * 512. Now, the dataset is ready to be fed to the deep learning model. We use SSIM loss between the generated empty tile and original empty tile, which is validated on a set of 30 test forms.

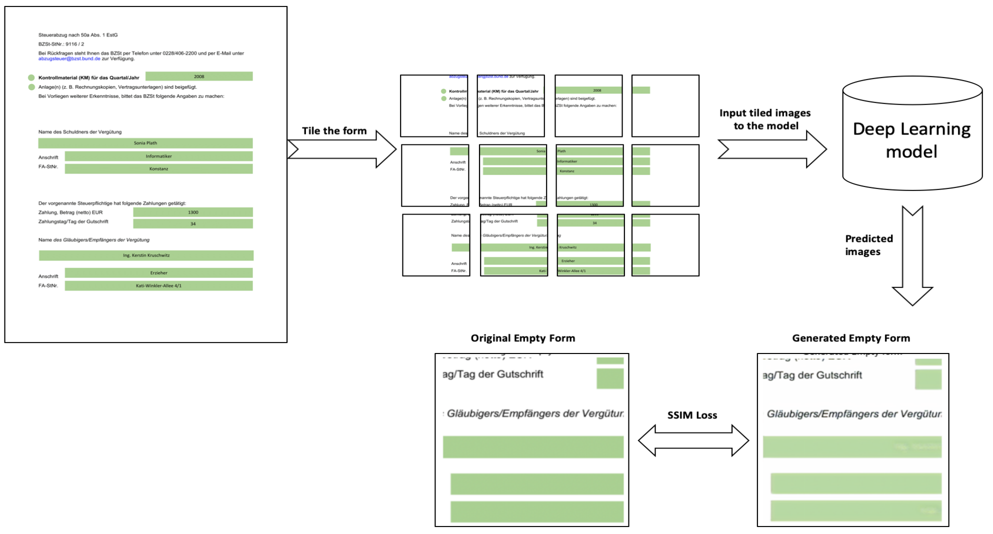


Figure 4.1: Block diagram of model during training time

To evaluate the model during testing, we need to divide the input images into tiles of an acceptable size of 512 * 512. These tiled images are then sent to the pre-trained model to generate corresponding predicted tiles for the complete form, which are then merged back together. Finally, the original empty forms and predicted empty forms are evaluated by applying the SSIM loss function to measure the similarity between the two images. Figure 4.2 provides an overview of our system during and after the inference mode.

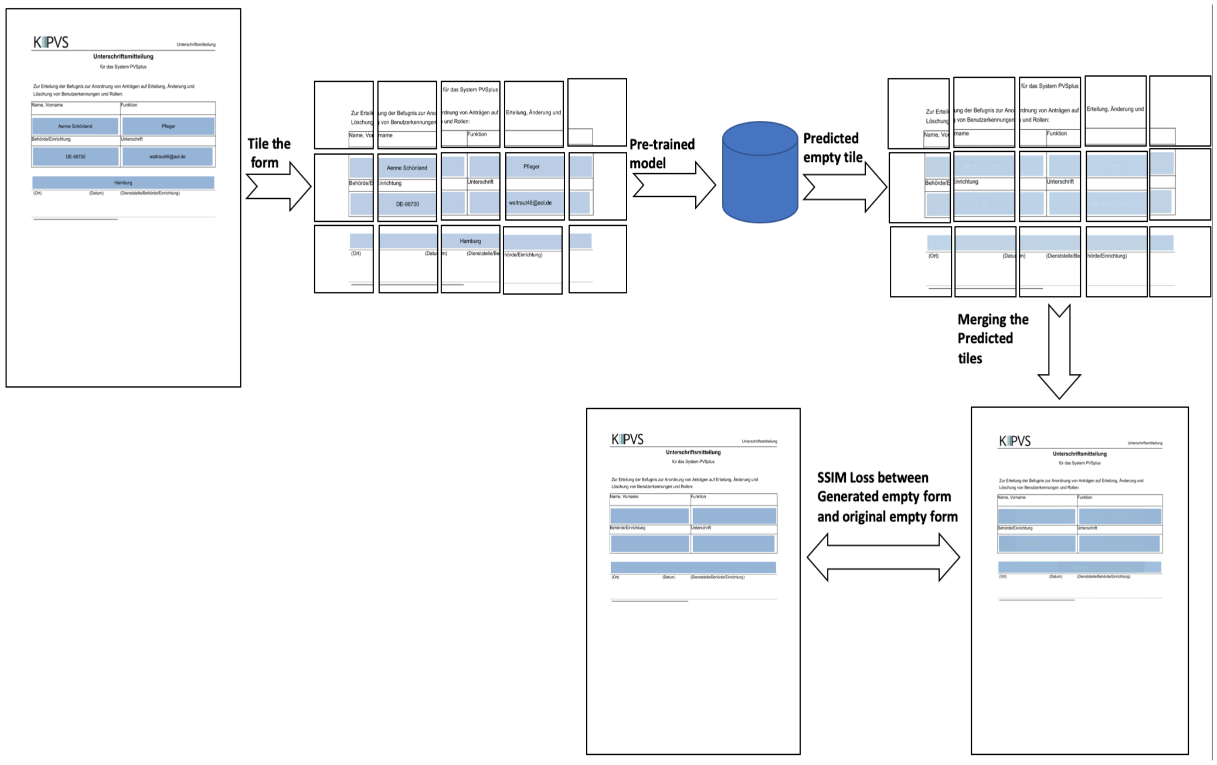

Tile the form
Pre-trained model
Predicted empty tile
Merging the Predicted tiles
SSIM Loss between Generated empty form and original empty form

Figure 4.2: Block diagram of model during test time

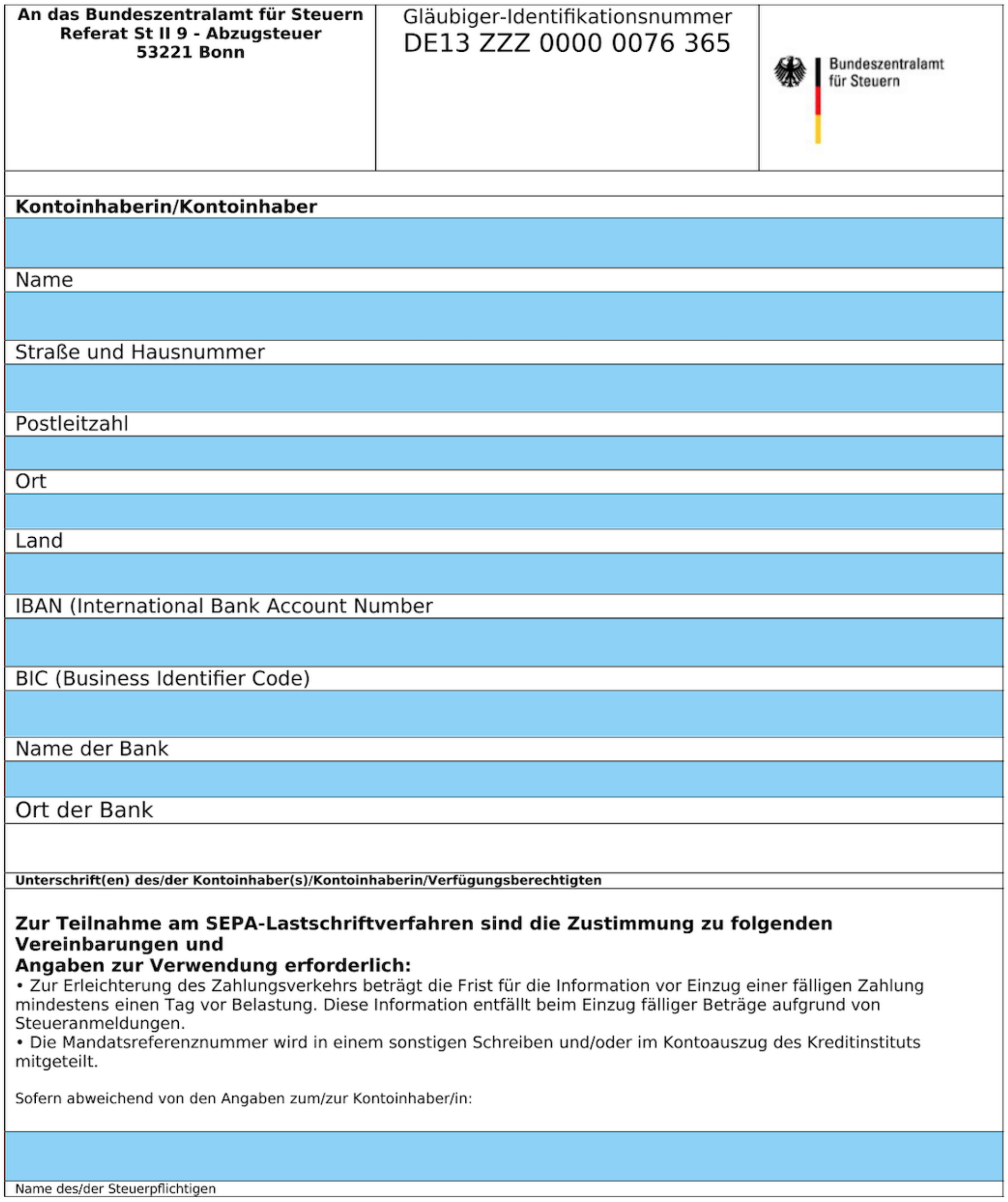
4.2 Dataset

The dataset considered for this project consists of 2019 Tax forms in the German language that are manually prepared and automatically filled.
The dataset includes coloured PDFs of the Tax forms, which need to be converted into images. These images have a size of 1700*2200. Since the images are very large, we need to divide them into smaller sizes, which are called tiles. Therefore, we process images in tiles of size 128*128 and 512*512, and further experiments are performed with tile sizes of 128*128 and 512*512. An example of the form is shown in Figure 4.3.

Each form contains static contents and dynamic contents.

- Static content refers to information that is consistent and the same for all customers, such as name, address, bank information, date, email, and telephone number. This information is provided to the customer in the form of a pre-printed form.

- Dynamic content refers to information related to individual customers. Each customer's information is completely unique, such as name, job, address, and email. This information is filled by the customer.



An das Bundeszentralamt für Steuern **Referat St II 9 - Abzugsteuer** **53221 Bonn**	Gläubiger-Identifikationsnummer DE13 ZZZ 0000 0076 365	Bundeszentralamt für Steuern

Kontoinhaberin/Kontoinhaber

Name

Straße und Hausnummer

Postleitzahl

Ort

Land

IBAN (International Bank Account Number

BIC (Business Identifier Code)

Name der Bank

Ort der Bank

Unterschrift(en) des/der Kontoinhaber(s)/Kontoinhaberin/Verfügungsberechtigten

Zur Teilnahme am SEPA-Lastschriftverfahren sind die Zustimmung zu folgenden Vereinbarungen und
Angaben zur Verwendung erforderlich:
• Zur Erleichterung des Zahlungsverkehrs beträgt die Frist für die Information vor Einzug einer fälligen Zahlung mindestens einen Tag vor Belastung. Diese Information entfällt beim Einzug fälliger Beträge aufgrund von Steueranmeldungen.
• Die Mandatsreferenznummer wird in einem sonstigen Schreiben und/oder im Kontoauszug des Kreditinstituts mitgeteilt.

Sofern abweichend von den Angaben zum/zur Kontoinhaber/in:

Name des/der Steuerpflichtigen

Figure 4.3: A sample of empty form (form-07) with light blue background colored

The forms in this dataset include only static content. Therefore, in the first step, the forms need to be filled, and in the next step, they need to be converted from PDF format to PNG format to feed the deep learning model.

4.2.1 Format of dynamic content

- The forms are filled with random strings and digits, with some text including a mixture of upper-case and lower-case letters.
- The size and font-family used to fill the forms for dynamic content are exactly the same as those used for static content.
- The color of the font used for dynamic content is black.
- Other types of dynamic content, such as checkboxes and signatures, are excluded.

4.2.2 categories of dynamic content

For filling those form, I need different categories of dynamic data and you can see sub-categories in table 4.1:

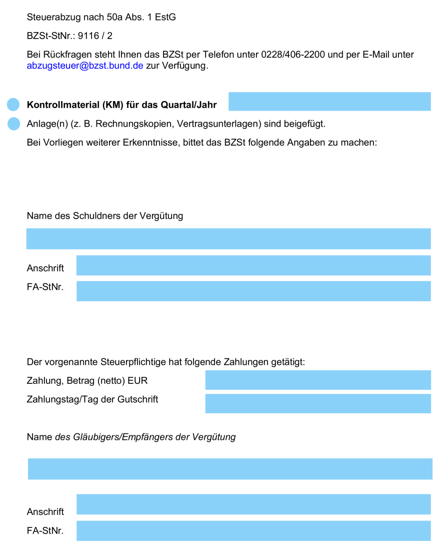
Table 4.1: Synthetic dynamic content

category	sub-category
Name	Full-name, Name-female, Name-male, Last-name
Address	Full-address, Street-address, Street-name, Postcode, City-name, Country
Bank	Bank-country, IBAN, Swift-code
Company	Company-name, Company-email, Company-suffix
Date	Date, Date-between-dates, Date-of-birth, Date-this-month, Date-this-year, Date-time
Phone number	phone-number, Country-code
Job Title	job
Email	email
Text	random text with maximum number of chars equal 200

Dynamic contents vary across the different forms, and for each form, I need to generate some of these contents and insert them into specific locations on the form.
The dataset includes 10 different forms with five different colors: gray, blue, green, yellow, and red. Each color has various shades, such as dark blue and light blue. Fig. 4.4 shows various shades of blue color.

4.2.3 Filling the form

This step involves filling out the form. The locations of the dynamic contents differ in different forms. For each form, the appropriate location of dynamic contents and the categories of dynamic contents related to that form have to be extracted.
Figure 4.5 shows the code for filling out Form-07, which is displayed in Fig. 4.3. I used the random.randint(0,100) method to generate a random number between 0 and 100 and added it to a constant



Steuerabzug nach 50a Abs. 1 EstG

BZSt-StNr.: 9116 / 2

Bei Rückfragen steht Ihnen das BZSt per Telefon unter 0228/406-2200 und per E-Mail unter abzugsteuer@bzst.bund.de zur Verfügung.

● **Kontrollmaterial (KM) für das Quartal/Jahr**

● Anlage(n) (z. B. Rechnungskopien, Vertragsunterlagen) sind beigefügt.

Bei Vorliegen weiterer Erkenntnisse, bittet das BZSt folgende Angaben zu machen:

Name des Schuldners der Vergütung

Anschrift

FA-StNr.

Der vorgenannte Steuerpflichtige hat folgende Zahlungen getätigt:

Zahlung, Betrag (netto) EUR

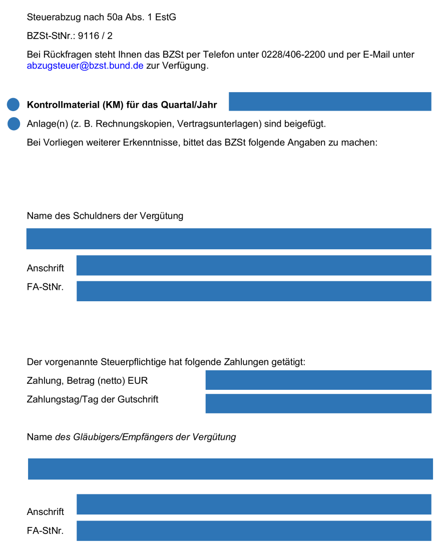
Zahlungstag/Tag der Gutschrift

Name *des Gläubigers/Empfängers der Vergütung*

Anschrift

FA-StNr.

(a) form-01-r1



Steuerabzug nach 50a Abs. 1 EstG

BZSt-StNr.: 9116 / 2

Bei Rückfragen steht Ihnen das BZSt per Telefon unter 0228/406-2200 und per E-Mail unter abzugsteuer@bzst.bund.de zur Verfügung.

● **Kontrollmaterial (KM) für das Quartal/Jahr**

● Anlage(n) (z. B. Rechnungskopien, Vertragsunterlagen) sind beigefügt.

Bei Vorliegen weiterer Erkenntnisse, bittet das BZSt folgende Angaben zu machen:

Name des Schuldners der Vergütung

Anschrift

FA-StNr.

Der vorgenannte Steuerpflichtige hat folgende Zahlungen getätigt:

Zahlung, Betrag (netto) EUR

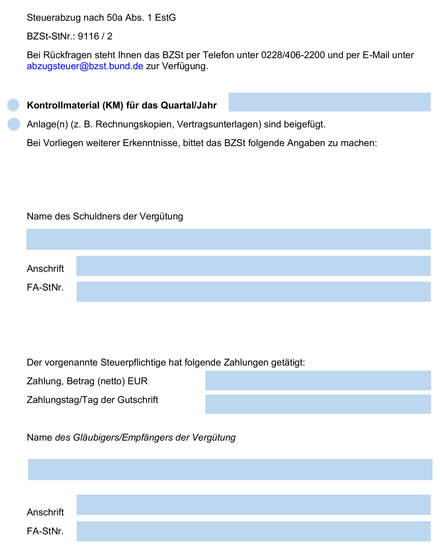
Zahlungstag/Tag der Gutschrift

Name *des Gläubigers/Empfängers der Vergütung*

Anschrift

FA-StNr.

(b) form-01-r2



Steuerabzug nach 50a Abs. 1 EstG

BZSt-StNr.: 9116 / 2

Bei Rückfragen steht Ihnen das BZSt per Telefon unter 0228/406-2200 und per E-Mail unter abzugsteuer@bzst.bund.de zur Verfügung.

● **Kontrollmaterial (KM) für das Quartal/Jahr**

● Anlage(n) (z. B. Rechnungskopien, Vertragsunterlagen) sind beigefügt.

Bei Vorliegen weiterer Erkenntnisse, bittet das BZSt folgende Angaben zu machen:

Name des Schuldners der Vergütung

Anschrift

FA-StNr.

Der vorgenannte Steuerpflichtige hat folgende Zahlungen getätigt:

Zahlung, Betrag (netto) EUR

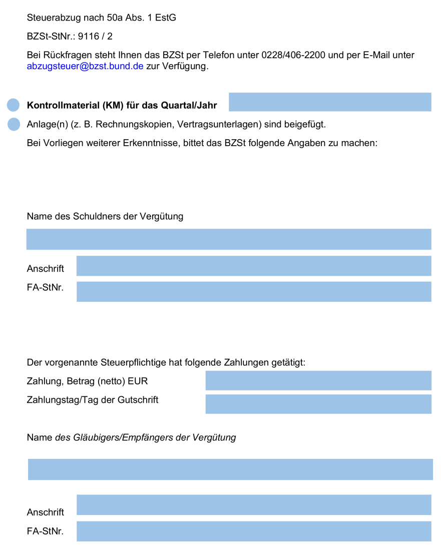
Zahlungstag/Tag der Gutschrift

Name *des Gläubigers/Empfängers der Vergütung*

Anschrift

FA-StNr.

(c) form-01-r3



Steuerabzug nach 50a Abs. 1 EstG

BZSt-StNr.: 9116 / 2

Bei Rückfragen steht Ihnen das BZSt per Telefon unter 0228/406-2200 und per E-Mail unter abzugsteuer@bzst.bund.de zur Verfügung.

● **Kontrollmaterial (KM) für das Quartal/Jahr**

● Anlage(n) (z. B. Rechnungskopien, Vertragsunterlagen) sind beigefügt.

Bei Vorliegen weiterer Erkenntnisse, bittet das BZSt folgende Angaben zu machen:

Name des Schuldners der Vergütung

Anschrift

FA-StNr.

Der vorgenannte Steuerpflichtige hat folgende Zahlungen getätigt:

Zahlung, Betrag (netto) EUR

Zahlungstag/Tag der Gutschrift

Name *des Gläubigers/Empfängers der Vergütung*

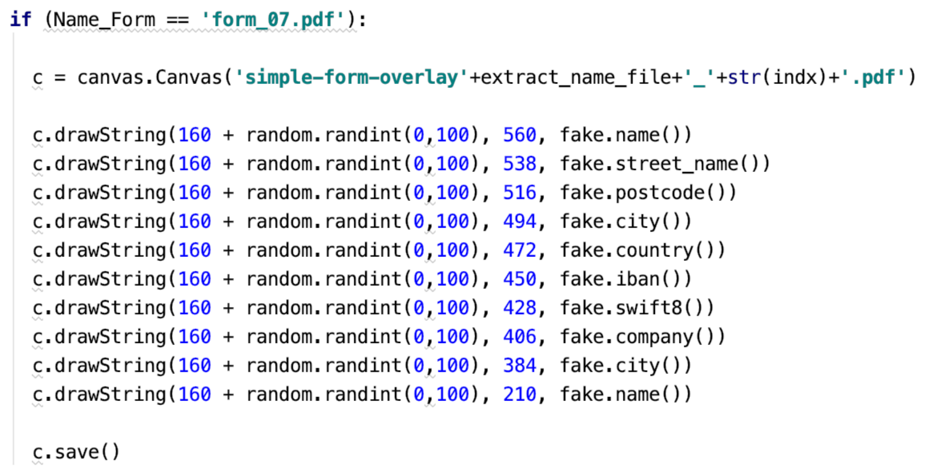
Anschrift

FA-StNr.

(d) form-01-r4

Figure 4.4: Form-01 with blue background color in different shades

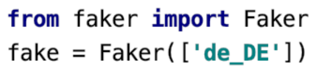
value to create a random location for the horizontal axis. The vertical axis has a constant value. The horizontal and vertical numbers are obtained as the location to insert the desired dynamic content in that location.



```
if (Name_Form == 'form_07.pdf'):

    c = canvas.Canvas('simple-form-overlay'+extract_name_file+'_'+str(indx)+'.pdf')

    c.drawString(160 + random.randint(0,100), 560, fake.name())
    c.drawString(160 + random.randint(0,100), 538, fake.street_name())
    c.drawString(160 + random.randint(0,100), 516, fake.postcode())
    c.drawString(160 + random.randint(0,100), 494, fake.city())
    c.drawString(160 + random.randint(0,100), 472, fake.country())
    c.drawString(160 + random.randint(0,100), 450, fake.iban())
    c.drawString(160 + random.randint(0,100), 428, fake.swift8())
    c.drawString(160 + random.randint(0,100), 406, fake.company())
    c.drawString(160 + random.randint(0,100), 384, fake.city())
    c.drawString(160 + random.randint(0,100), 210, fake.name())

    c.save()
```

Figure 4.5: A sample of code for filling the form-07

To generate the fake dynamic contents in German, I imported the Faker library and used the 'de-DE' language code. A sample of the code can be seen in Figure 4.6.



```
from faker import Faker
fake = Faker(['de_DE'])
```

Figure 4.6: Fake library for generating fake dynamic contents

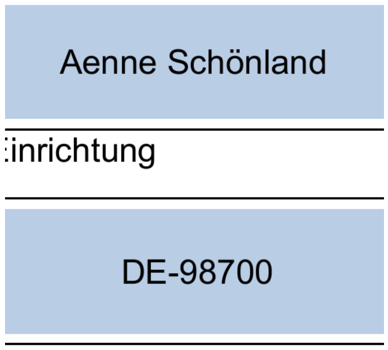
4.2.4 Conver PDF to PNG

The above datasets introduced comprise of A-4 size PDFs, which need to be further transformed into images to be processed by a deep learning model.
There are some image-processing libraries such as Poppler and ImageMagick for converting documents from PDF to PNG, but a standard quality image is required for input to the model. An ideal conversion would be an image that is neither too big nor too small and has adequate image quality so that every letter in the PDF can be clearly separated.
Initially, the conversion of a PDF to an image using 200 dpi resulted in a low-dimensional (2339 x 1654) image but with poor image quality, while 400-600 dpi resulted in a high-resolution image but with high-dimensional (4678 x 3308) [22].

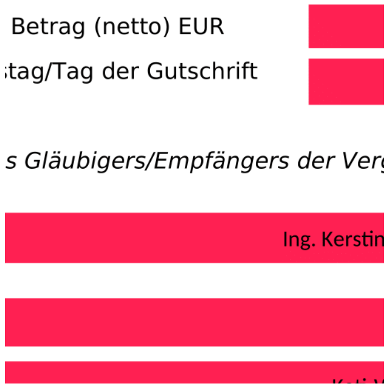
There are different type of DPI for images:

- **200 dpi conversion:** The image generated with 200 dpi has some letters that are quite close to each other. For example, the letter "r" here is merged with "v", "f", "s", "t", and "ä" at separate instances. These merges were quite frequent in the overall document image and could mislead the model to learn such irrelevant patterns [22].

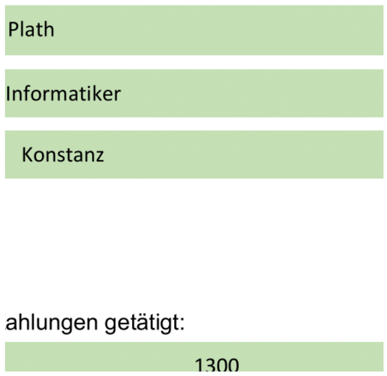
- **400 dpi conversion:** Converting images with a large dpi can lead to inefficient tiling. Since the images are processed in tiles, they need to capture the context efficiently in a single tile. Having a large dimensional image would require bigger tiles. Since the image has large dimensions of (4678 x 3308), the tiled images would be of larger size in order to include enough context in a single tile [22].
- **300 dpi conversion:** The images generated using 300 dpi not only have adequate spacing between each letter but can also be divided into suitable tiles that can be processed by a neural network in a single batch. Hence, all PDFs are converted to images using 300 dpi in all further experiments. Fig. 4.7 shows some samples of tiles in different forms with 300 dpi [22].


Aenne Schönland
:inrichtung
DE-98700

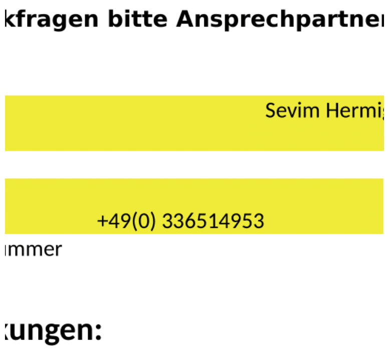
(a) tile-01


Betrag (netto) EUR
;tag/Tag der Gutschrift
s Gläubigers/Empfängers der Ver
Ing. Kerstin

(b) tile-02


Plath
Informatiker
Konstanz
ahlungen getätigt:
1300

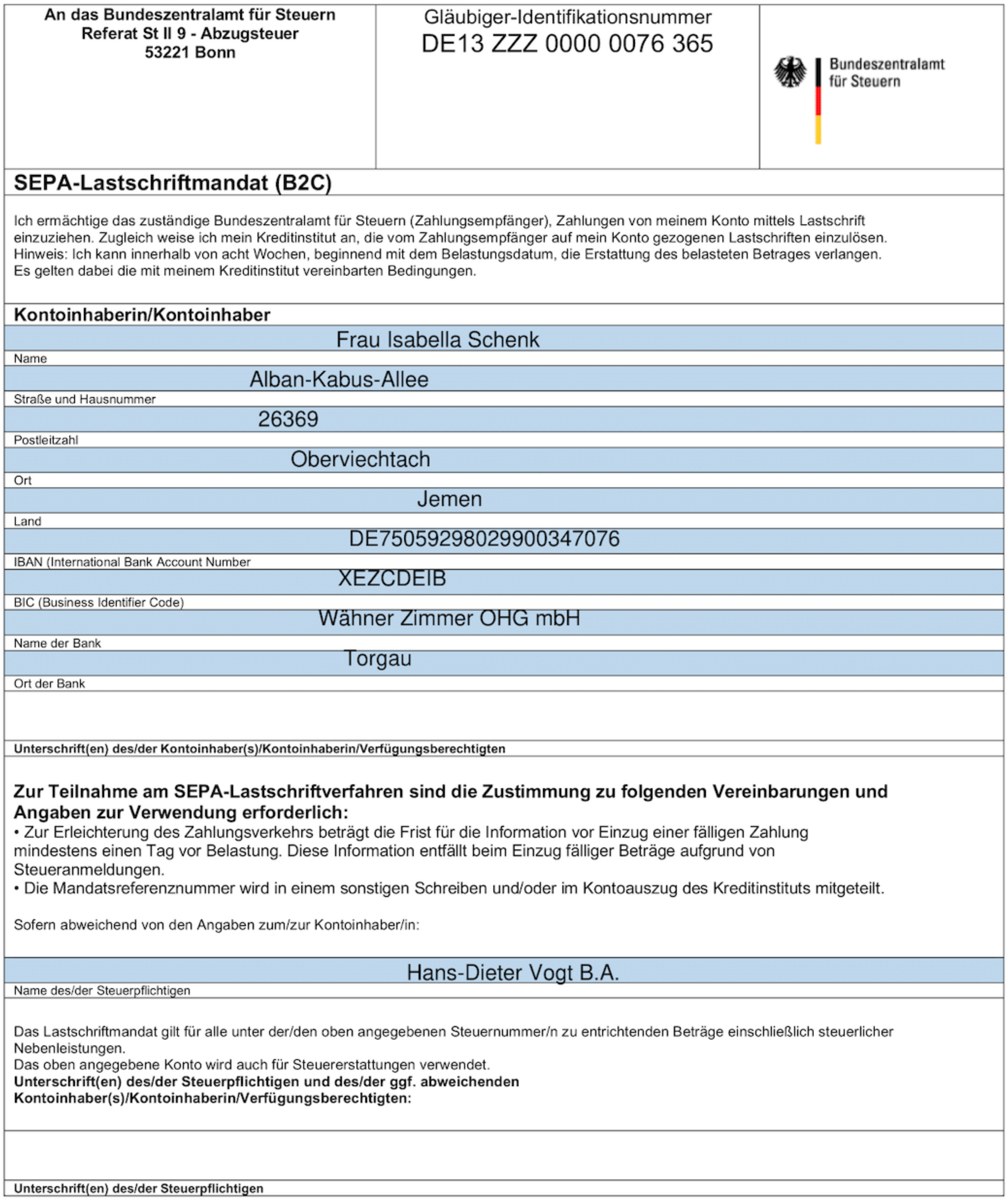
(c) tile-03


kfragen bitte Ansprechpartne
Sevim Hermi
+49(0) 336514953
ımmer
:ungen:

(d) tile-04

Figure 4.7: Some sample of tiles in different forms with 300 dpi

Figure 4.8 shows a sample filled document image from synthetic dynamic content.



An das Bundeszentralamt für Steuern **Referat St II 9 - Abzugsteuer** **53221 Bonn**	Gläubiger-Identifikationsnummer DE13 ZZZ 0000 0076 365	Bundeszentralamt für Steuern

SEPA-Lastschriftmandat (B2C)

Ich ermächtige das zuständige Bundeszentralamt für Steuern (Zahlungsempfänger), Zahlungen von meinem Konto mittels Lastschrift einzuziehen. Zugleich weise ich mein Kreditinstitut an, die vom Zahlungsempfänger auf mein Konto gezogenen Lastschriften einzulösen.
Hinweis: Ich kann innerhalb von acht Wochen, beginnend mit dem Belastungsdatum, die Erstattung des belasteten Betrages verlangen.
Es gelten dabei die mit meinem Kreditinstitut vereinbarten Bedingungen.

Kontoinhaberin/Kontoinhaber

Frau Isabella Schenk
Name

Alban-Kabus-Allee
Straße und Hausnummer

26369
Postleitzahl

Oberviechtach
Ort

Jemen
Land

DE75059298029900347076
IBAN (International Bank Account Number

XEZCDEIB
BIC (Business Identifier Code)

Wähner Zimmer OHG mbH
Name der Bank

Torgau
Ort der Bank

Unterschrift(en) des/der Kontoinhaber(s)/Kontoinhaberin/Verfügungsberechtigten

Zur Teilnahme am SEPA-Lastschriftverfahren sind die Zustimmung zu folgenden Vereinbarungen und Angaben zur Verwendung erforderlich:
• Zur Erleichterung des Zahlungsverkehrs beträgt die Frist für die Information vor Einzug einer fälligen Zahlung mindestens einen Tag vor Belastung. Diese Information entfällt beim Einzug fälliger Beträge aufgrund von Steueranmeldungen.
• Die Mandatsreferenznummer wird in einem sonstigen Schreiben und/oder im Kontoauszug des Kreditinstituts mitgeteilt.

Sofern abweichend von den Angaben zum/zur Kontoinhaber/in:

Hans-Dieter Vogt B.A.
Name des/der Steuerpflichtigen

Das Lastschriftmandat gilt für alle unter der/den oben angegebenen Steuernummer/n zu entrichtenden Beträge einschließlich steuerlicher Nebenleistungen.
Das oben angegebene Konto wird auch für Steuererstattungen verwendet.
Unterschrift(en) des/der Steuerpflichtigen und des/der ggf. abweichenden Kontoinhaber(s)/Kontoinhaberin/Verfügungsberechtigten:

Unterschrift(en) des/der Steuerpflichtigen

Figure 4.8: A sample filled document image from synthetic dynamic content

Chapter 5

Experiments and Result

5.1 Model Architecture

As mentioned before, a CycleGAN contains four models: two generators and two discriminators. Let's take a closer look at the architecture of the generators. CycleGAN generators can take one of two forms: U-Net or ResNet.

5.2 The generator U-Net

As shown in Figure 5.1, a U-Net is composed of two halves: the down-sampling half, which compresses input images while increasing the number of channels, and the up-sampling half, which expands representations while reducing the number of channels.

U-Net includes skip connections that enable information to bypass certain parts of the network and flow to later layers. In each subsequent layer of the down-sampling half, the model captures the "what" of the images but loses information on the "where". At the apex of the U, the feature maps will have learned a contextual understanding of what is in the image but have limited knowledge of its location.

The critical aspect of the U-Net architecture is during the up-sampling phase, where we upscale back to the original image size. We reintroduce the spatial information that was lost during down-sampling into each layer. This is why skip connections are necessary; they allow the network to blend high-level abstract information obtained during down-sampling with specific spatial information fed back from previous layers in the network.

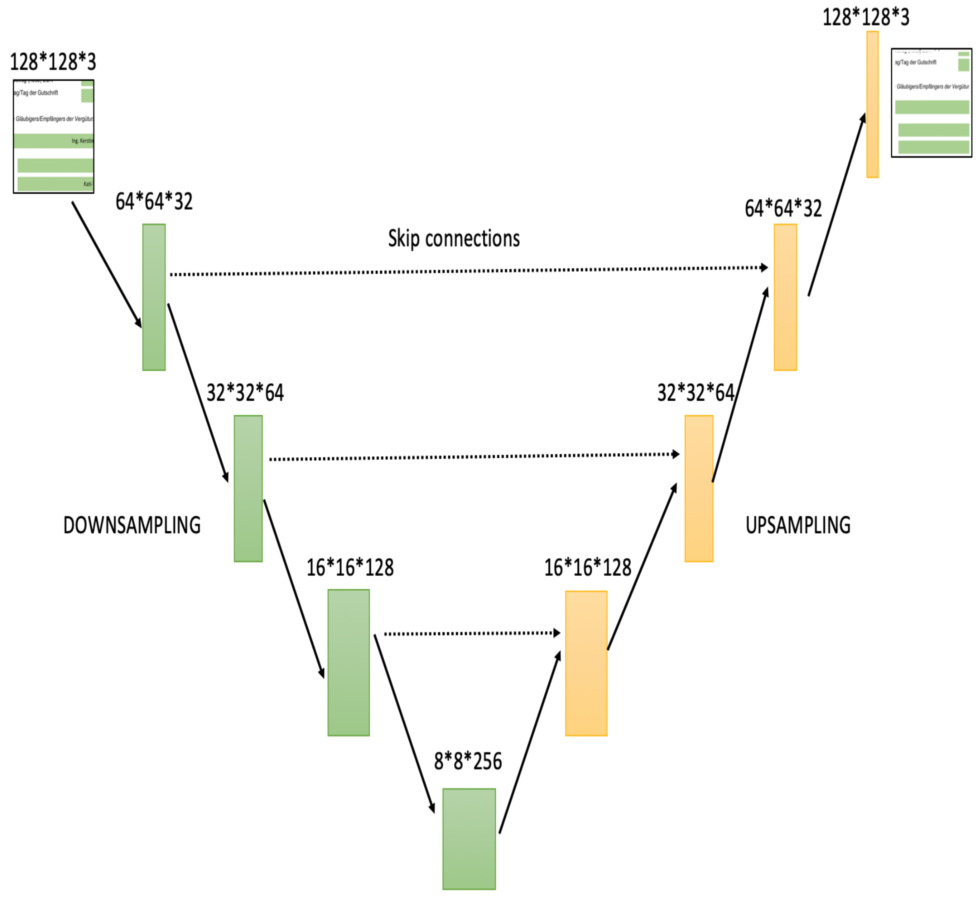


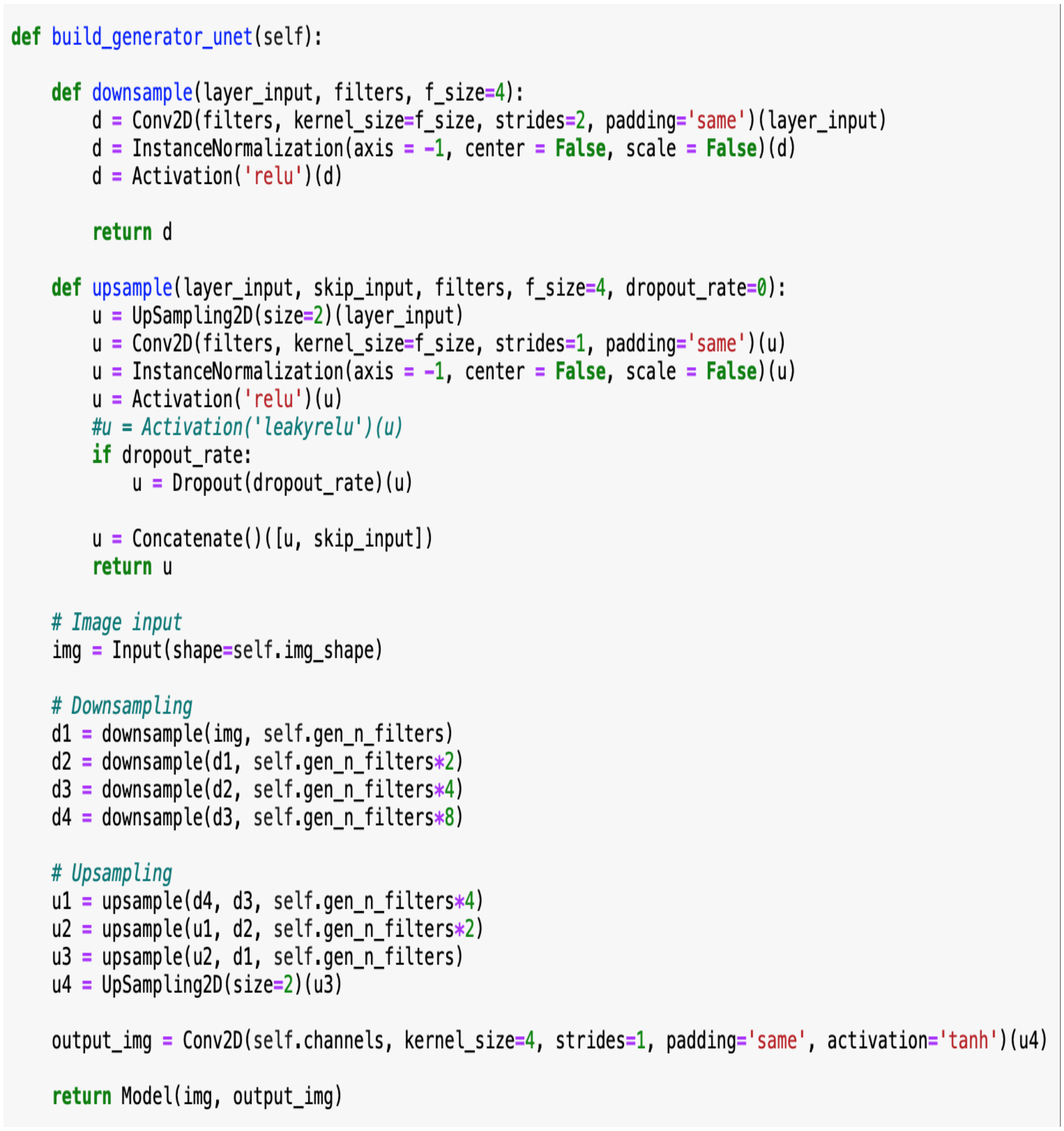
Figure 5.1: The U-Net architecture diagram

To build skip connections, we will need to introduce a new type of layer: 'Concatenate'. The concatenate layer joins a set of layers together along an axis. In the U-Net, there are concatenate layers used to connect up-sampling layers to an equivalently sized layer in the down-sampling part of the network. The layers are joined together along the channels dimension. There are no weights to be learned in a Concatenate layer; they are used to glue previous layers together.

This generator also contains another new layer type: InstanceNormalization. The generator of the CycleGAN uses InstanceNormalization layers rather than Batchnormalization layers. An InstanceNormalization layer normalizes every single observation individually, rather than as a batch. It doesn't require mu and sigma parameters to be calculated as a running average during training since, at test time, the layer can normalize per instance in the same way as it does at the train time. The means and standard deviations used to normalize each layer are calculated per channel and per observation.

For the InstanceNormalization layers in this network, there are no weights to learn because we do not use scaling (gamma) or shift (beta) parameters. We now have everything we need to build a

U-Net generator in Keras, as shown in Figure 5.2.



```python
def build_generator_unet(self):

    def downsample(layer_input, filters, f_size=4):
        d = Conv2D(filters, kernel_size=f_size, strides=2, padding='same')(layer_input)
        d = InstanceNormalization(axis = -1, center = False, scale = False)(d)
        d = Activation('relu')(d)

        return d

    def upsample(layer_input, skip_input, filters, f_size=4, dropout_rate=0):
        u = UpSampling2D(size=2)(layer_input)
        u = Conv2D(filters, kernel_size=f_size, strides=1, padding='same')(u)
        u = InstanceNormalization(axis = -1, center = False, scale = False)(u)
        u = Activation('relu')(u)
        #u = Activation('leakyrelu')(u)
        if dropout_rate:
            u = Dropout(dropout_rate)(u)

        u = Concatenate()([u, skip_input])
        return u

    # Image input
    img = Input(shape=self.img_shape)

    # Downsampling
    d1 = downsample(img, self.gen_n_filters)
    d2 = downsample(d1, self.gen_n_filters*2)
    d3 = downsample(d2, self.gen_n_filters*4)
    d4 = downsample(d3, self.gen_n_filters*8)

    # Upsampling
    u1 = upsample(d4, d3, self.gen_n_filters*4)
    u2 = upsample(u1, d2, self.gen_n_filters*2)
    u3 = upsample(u2, d1, self.gen_n_filters)
    u4 = UpSampling2D(size=2)(u3)

    output_img = Conv2D(self.channels, kernel_size=4, strides=1, padding='same', activation='tanh')(u4)

    return Model(img, output_img)
```

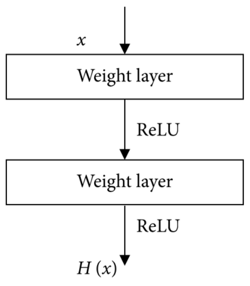
Figure 5.2: Python code in Keras for building the U-Net generator

As you can see in Figure 5.2, the generator consists of two halves:

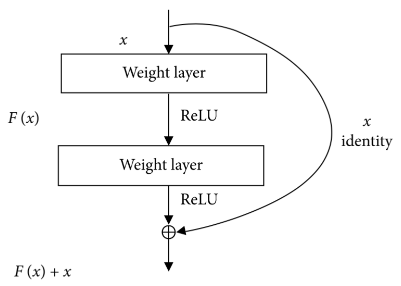
1. **Down-sampling blocks:** First, we down-sample the image using Conv2D layers with a stride of 2. The down-sampling blocks contain 4 steps, at each step:
    - The number of filters doubles.
    - The size of the kernel is constant and equal to 4.
    - The value for the stride is equal to 2, which means the height and width of the output tensor will be half the size of the input tensor.
    - The padding parameter is equal to 'same', which means that the parameter pads the input data with zeros.
    - Apply the InstanceNormalization layer.
    - The activation function should be 'relu'.
2. **Upsampling blocks:** Then we upsample to return the tensor to the same size as the original image. The up-sampling blocks include 4 steps, at each step:
    - Use the UpSample2D layer with a size equal to 2. It upsamples an input to double each row and column.
    - The number of filters is halved.
    - The size of the kernel is constant and equal to 4.
    - The value for the stride is equal to 1, which means the height and width of the output tensor will be equal to the size of the input tensor.
    - The padding parameter is equal to 'same', which means that the parameter pads the input data with zeros.
    - Apply the InstanceNormalization layer.
    - The activation function should be 'relu'.
    - If the dropout-rate parameter is equal to 0, ignore the Dropout layer; otherwise, apply the Dropout layer.
    - The last layer of the up-sampling blocks should be the Concatenate layer, which gives the network the U-Net architecture.

5.3 The generator ResNet

The general CNN network shown in Figure 5.3(a) receives the input x and produces the output H(x) through two weighted layers. This output is then passed on as input to the next layer. Figure 5.3(b) illustrates the architecture of the ResNet, which employs a short cut connection to link the input of a layer directly to its output. Although it is a simple network, ResNet delivers superior performance.
The ResNet architecture is similar to that of a U-Net in that it enables information from earlier layers in the network to bypass one or more layers. However, instead of connecting layers from the down-sampling part of the network to corresponding up-sampling layers to form a U-shape, a ResNet is constructed from residual blocks stacked on top of one another, with each block featuring a skip connection that adds the input and output of the block before passing it on to the next layer. Figure 5.3(b) shows a single residual block.



(a)



(b)

Figure 5.3: The architecture of the networks: (a) CNN and (b) ResNet. The output in CNN is the input of the next layer. The input in ResNet connects directly to the output of the layer

In our CycleGAN, the "weight layers" in the diagram are convolutional layers with instance normalization. In keras, a residual block can be coded as shown in Following Figure 5.4.

```
def residual(layer_input, filters):
    shortcut = layer_input
    y = ReflectionPadding2D(padding =(1,1))(layer_input)
    y = Conv2D(filters, kernel_size=(3, 3), strides=1, padding='valid')(y)
    y = InstanceNormalization(axis = -1, center = False, scale = False)(y)
    y = Activation('relu')(y)

    y = ReflectionPadding2D(padding =(1,1))(y)
    y = Conv2D(filters, kernel_size=(3, 3), strides=1, padding='valid')(y)
    y = InstanceNormalization(axis = -1, center = False, scale = False)(y)

    return add([shortcut, y])
```

Figure 5.4: Python code in Keras : a residual block

In each residual layer:

- The first layer is Reflection padding, which reflects the row into the padding.
- Conv2D filters extend through the three channels in an image, with different filters for each channel. The kernel size is constant and equal to 3.
- An InstanceNormalization layer is then applied.
- The activation function used is 'relu'.
- The second layer is a Reflection padding layer.
- The second Conv2D layer uses different filter values, with a kernel size of 3.
- A second InstanceNormalization layer is then applied.

On the other side of the residual blocks, the ResNet generator also includes down-sampling and up-sampling layers. The overall architecture of the ResNet and its code are shown in the figures.

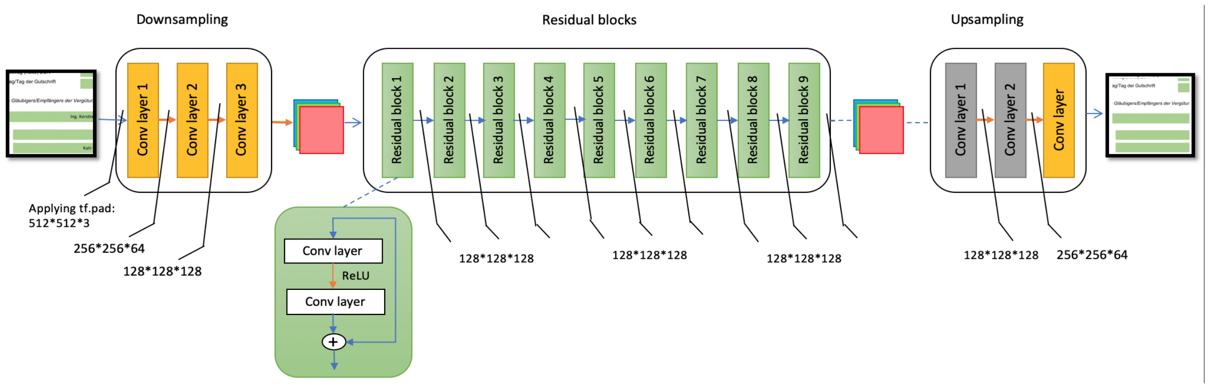
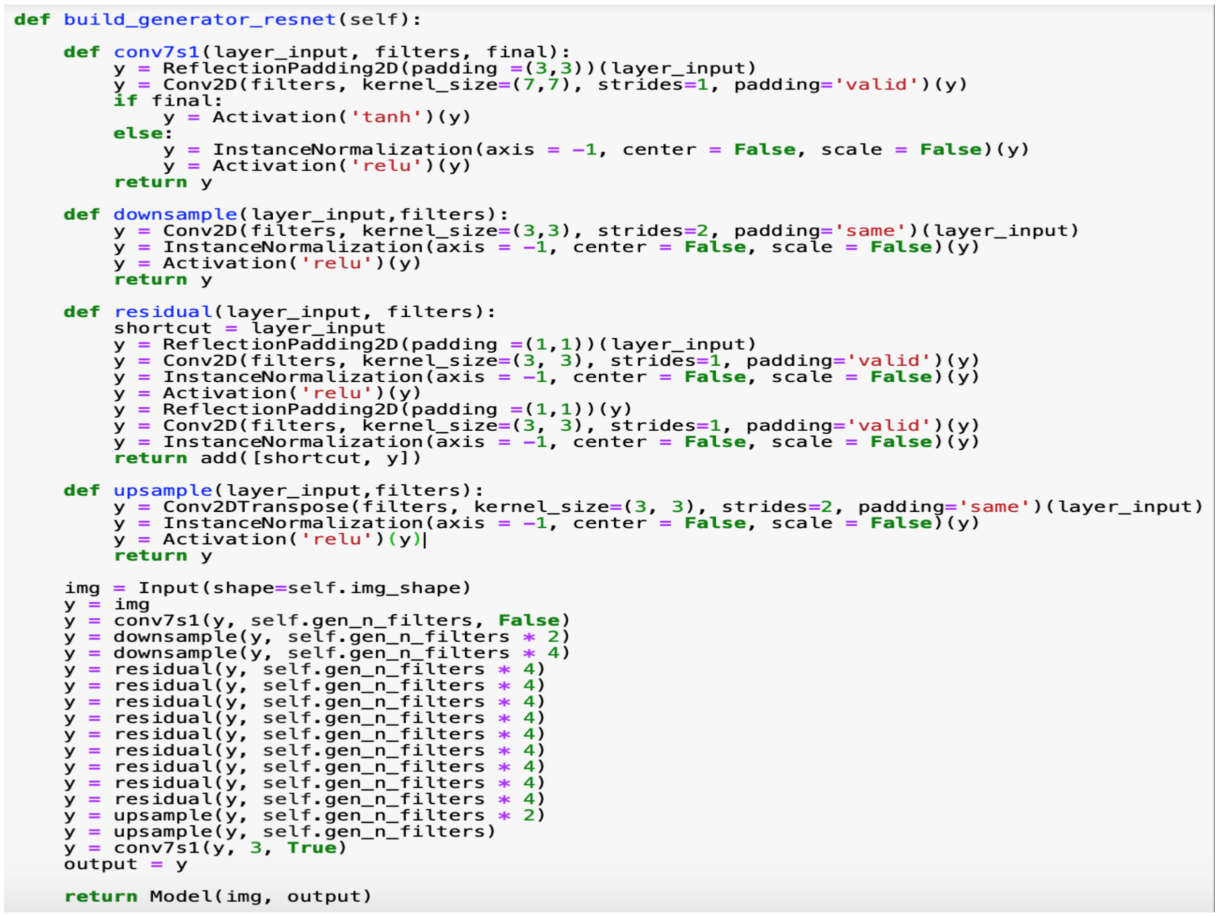


Figure 5.5: A ResNet Generator



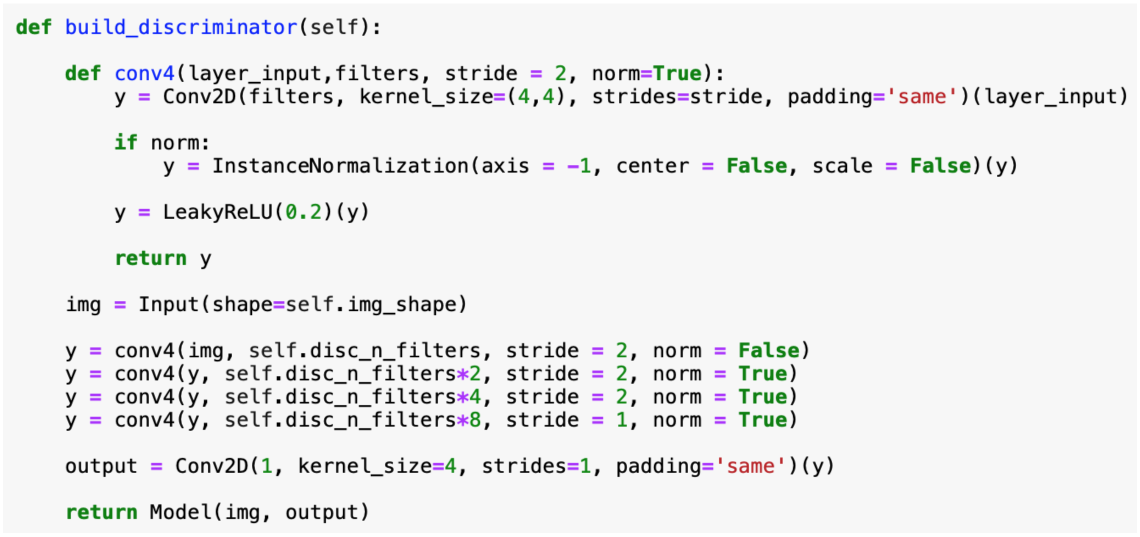
```
def build_generator_resnet(self):

    def conv7s1(layer_input, filters, final):
        y = ReflectionPadding2D(padding =(3,3))(layer_input)
        y = Conv2D(filters, kernel_size=(7,7), strides=1, padding='valid')(y)
        if final:
            y = Activation('tanh')(y)
        else:
            y = InstanceNormalization(axis = -1, center = False, scale = False)(y)
            y = Activation('relu')(y)
        return y

    def downsample(layer_input,filters):
        y = Conv2D(filters, kernel_size=(3,3), strides=2, padding='same')(layer_input)
        y = InstanceNormalization(axis = -1, center = False, scale = False)(y)
        y = Activation('relu')(y)
        return y

    def residual(layer_input, filters):
        shortcut = layer_input
        y = ReflectionPadding2D(padding =(1,1))(layer_input)
        y = Conv2D(filters, kernel_size=(3, 3), strides=1, padding='valid')(y)
        y = InstanceNormalization(axis = -1, center = False, scale = False)(y)
        y = Activation('relu')(y)
        y = ReflectionPadding2D(padding =(1,1))(y)
        y = Conv2D(filters, kernel_size=(3, 3), strides=1, padding='valid')(y)
        y = InstanceNormalization(axis = -1, center = False, scale = False)(y)
        return add([shortcut, y])

    def upsample(layer_input,filters):
        y = Conv2DTranspose(filters, kernel_size=(3, 3), strides=2, padding='same')(layer_input)
        y = InstanceNormalization(axis = -1, center = False, scale = False)(y)
        y = Activation('relu')(y)
        return y

    img = Input(shape=self.img_shape)
    y = img
    y = conv7s1(y, self.gen_n_filters, False)
    y = downsample(y, self.gen_n_filters * 2)
    y = downsample(y, self.gen_n_filters * 4)
    y = residual(y, self.gen_n_filters * 4)
    y = residual(y, self.gen_n_filters * 4)
    y = residual(y, self.gen_n_filters * 4)
    y = residual(y, self.gen_n_filters * 4)
    y = residual(y, self.gen_n_filters * 4)
    y = residual(y, self.gen_n_filters * 4)
    y = residual(y, self.gen_n_filters * 4)
    y = residual(y, self.gen_n_filters * 4)
    y = residual(y, self.gen_n_filters * 4)
    y = upsample(y, self.gen_n_filters * 2)
    y = upsample(y, self.gen_n_filters)
    y = conv7s1(y, 3, True)
    output = y

    return Model(img, output)
```

Figure 5.6: Python code in Keras for building the ResNet Generator

As you can see in Figure 5.6, in order to build the ResNet generator, we need four functions: conv7s1, down-sample, residual, and up-sample. This generator contains 9 residual layers, 2 down-sample layers, 2 up-sample layers, and 2 conv7s1 layers.

5.4 The Discriminator

The discriminator in CycleGAN that we will build outputs an 8*8 single-channel tensor instead of a single number.
The reason for this is that CycleGAN inherits its discriminator architecture from a model called a PatchGAN. In PatchGAN, the discriminator divides the image into overlapping square patches and predicts whether each patch is real or fake instead of predicting for the image as a whole. Therefore, the output of the discriminator is a tensor that contains the predicted probability for each patch, rather than just a single number.
Using a PatchGAN discriminator has the benefit that the loss function can measure how well the discriminator distinguishes images based on their style, rather than their content. For making decisions, the discriminator must use the style of the patch instead of its content. The Keras code for building the discriminators is provided in the following figure.



```
def build_discriminator(self):

    def conv4(layer_input,filters, stride = 2, norm=True):
        y = Conv2D(filters, kernel_size=(4,4), strides=stride, padding='same')(layer_input)

        if norm:
            y = InstanceNormalization(axis = -1, center = False, scale = False)(y)

        y = LeakyReLU(0.2)(y)

        return y

    img = Input(shape=self.img_shape)

    y = conv4(img, self.disc_n_filters, stride = 2, norm = False)
    y = conv4(y, self.disc_n_filters*2, stride = 2, norm = True)
    y = conv4(y, self.disc_n_filters*4, stride = 2, norm = True)
    y = conv4(y, self.disc_n_filters*8, stride = 1, norm = True)

    output = Conv2D(1, kernel_size=4, strides=1, padding='same')(y)

    return Model(img, output)
```

Figure 5.7: Python code in Keras for building the discriminator

As you can see in Figure 5.7, the Python code in Keras for building the discriminator includes the following:

- A CycleGAN discriminator is a series of convolutional layers, all with instance normalization, except for the first layer.

- The final layer is a convolutional layer with only one filter and no activation function.

5.5 Experiments

5.5.1 Libraries

The experiments were implemented using the Python programming language, and various libraries were used for the implementation.

- To generate dynamic content, the ’faker’ library was used.
- For editing and manipulating PDF files, the following libraries were used: ’PyPDF4’, ’reportlab’, ’reportlab.pdfgen’, and ’pdfrw’.
- The major libraries used for creating the model in the deep neural network were:
  - CV2
  - Keras
  - Keras.layers linke: Input, Dense, Reshape, Flatten, Dropout, Concatenate, BatchNormalization, Activation, ZeroPadding2D, Add, LeakyReLU, UpSampling2D, Conv2D, Conv2DTranspose
  - keras.models like: Sequential, Model
- The major libraries used for reading and processing images were:
  - keras.preprocessing.imag like: ImageDataGenerator, load-img, save-img, img-to-array
  - Pandas
  - Numpy
  - walk, getcwd
  - scipy
  - glob
  - os
  - keras.backend
  - imageio
  - keras.utils like to-categorical
- The major libraries used for plotting data and saving results as CSV files were:
  - ExcelWriter
  - Tensorflow
  - matplotlib.pyplot

5.5.2 Hardware for training dataset

The experiment were conducted on a GeForce RTX 2080 Ti GPU with 10 GB of RAM.

5.5.3 Preparing the dataset

As explained in Sections 4.2.3 and 4.2.4 respectively, after filling out the forms and converting them from PDF to PNG, the dataset is ready to be fed into the Deep Neural Network model. However, the forms/images have a large size of 1700x2200 pixels, therefore the images are processed in tiles of size 128*128 pixels. The size of forms/images is not a multiple of 128, so first, I have to center the images on the canvas and pad them with white pixels all around the image. Then, the images are converted into tiles of size 128*128.

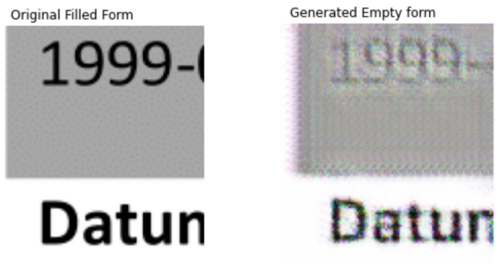
5.5.4 Experiments

The experiments were implemented using the Python programming language. For the first experiment, the U-Net generator was used to train the dataset. The normal U-Net architecture is shown in Figure 5.1. In order to use the U-Net generator, certain hyper-parameters had to be set. The values for the hyper-parameters can be seen in Table 5.1:

Table 5.1: The hyper parameters value for U-Net Generator with new tile size

The name of hyper parameters	Value
The name of generator	U-Net
The size of the image	(128*128*3)
The number of Epochs	200
Learning rate	0.0002
The number of filters for discriminator	32
The number of filters for generator	32

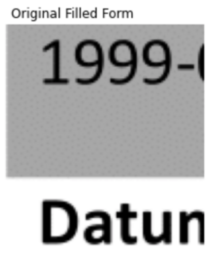
After training the dataset with the above hyper-parameters, the results should appear as in the Figure 5.8. The figure on the left shows the filled tile consisting of the 'Datum' label with its corresponding dynamic content, while on the right side is the corresponding generated empty form by the Template U-Net model with SSIM loss. These predictions reveal a major issue in the training and predictions, which is described below:



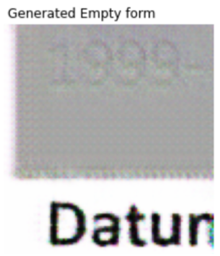
(a) Original fill tile **(b)** Predicted empty tile

Figure 5.8: Predicted empty tile with U-Net model, epochs=200, SSIM=0.0022585

- Due to the small number of epochs, the model was not trained well. Therefore, the first solution would be to increase the number of epochs from 200 to 400. In the Figure 5.9, on the right side, you can see the generated empty tile that was trained with the U-Net model and 400 epochs.



(a) Original fill tile



(b) Predicted empty tile

Figure 5.9: Predicted empty tile with U-Net model, epochs=400, SSIM=0.0009435

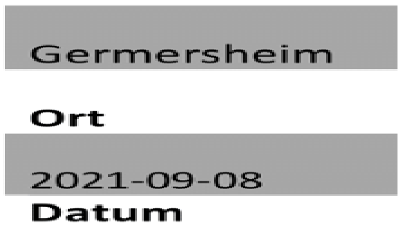
By comparing the SSIM loss values in Figures 5.8 and 5.9, you can see that after 400 epochs, the SSIM loss is less than that of 200 epochs. However, this prediction has a major issue; it cannot detect and erase dynamic contents. Therefore, I decided to use the ResNet model architecture for the next experiment.

To implement the second model, I used the same dataset. The size of each tile was set to 128*128, and the ResNet generator architecture is shown in Figure 5.5. In order to use the ResNet generator, I had to set new hyper-parameters, which are listed in Table 5.2:

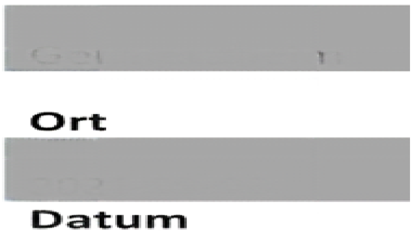
Table 5.2: The hyper parameters value for ResNet Generator

The name of hyper-parameters	Value
The name of generator	ResNet
The size of the image	(128*128*3)
The number of Epochs	400
Learning rate	0.0002
The number of filters for discriminator	64
The number of filters for generator	32
Batch Size	32

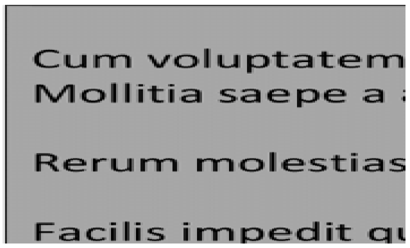
After training the dataset with the hyperparameters in table 5.2, you can see the results in Figure 5.10.



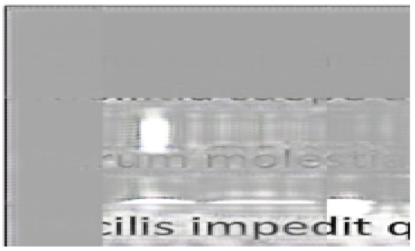
(a)



(b)



(c)



(d)

Figure 5.10: Predicted empty tiles with ResNet model, Epochs=400

The figures on the left, 5.10(a), 5.10(c), are filled tiles consisting of the labels 'Datum', 'Ort', and 'Description', with their corresponding dynamic contents. On the right side are the corresponding generated empty tiles by the ResNet model with SSIM loss. The above predictions demonstrate two major issues in the training and predictions, which are described below:

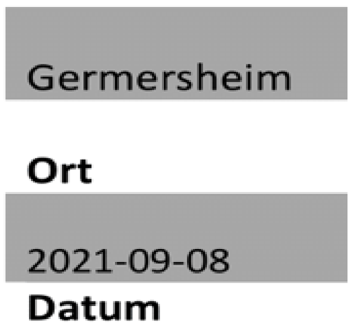
- The model can easily recognize dynamic contents that are not text and can erase them, but not completely. These dynamic contents are in a single word, such as date or city name. As you can see in Figure 5.10(b), the first character of the city name wasn't completely deleted.
  - Possible solution: Adjusting some hyper-parameters.
- The model could not completely erase the contents of the text-based dynamic contents due to a lack of variety in the training data, as shown in Figure 5.10(d).
  - Possible solution: Adding some forms with dynamic content that is text-based and spans more than one word. For example, adding forms with several sentences.

After increasing the number of forms and using new hyper-parameters, the dataset was retrained. You can see the new values of the hyper-parameters in table 5.3.

Table 5.3: The hyper parameters value for ResNet Generator

The name of hyper parameters	Value
The name of generator	ResNet
The size of the image	(128*128*3)
The number of Epochs	500
Learning rate	0.0002
The number of filters for discriminator	64
The number of filters for generator	32
Batch Size	32

However, it may be helpful to add more information about what is shown in Figure 5.11, such as whether it is the result of the ResNet model with the updated hyper-parameters and if it addresses the issues previously mentioned.

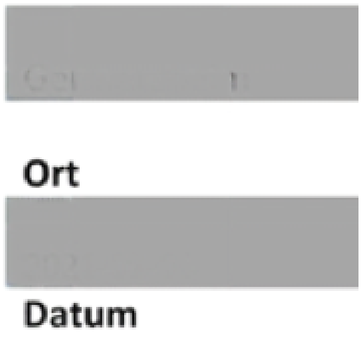

Germersheim

Ort

2021-09-08

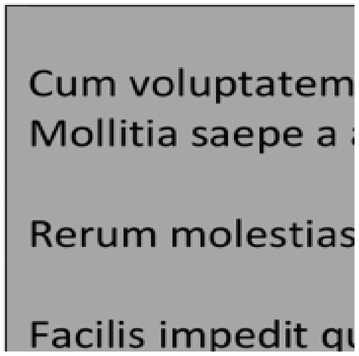
Datum

(a)


Ort

Datum

(b)


Cum voluptatem
Mollitia saepe a

Rerum molestias

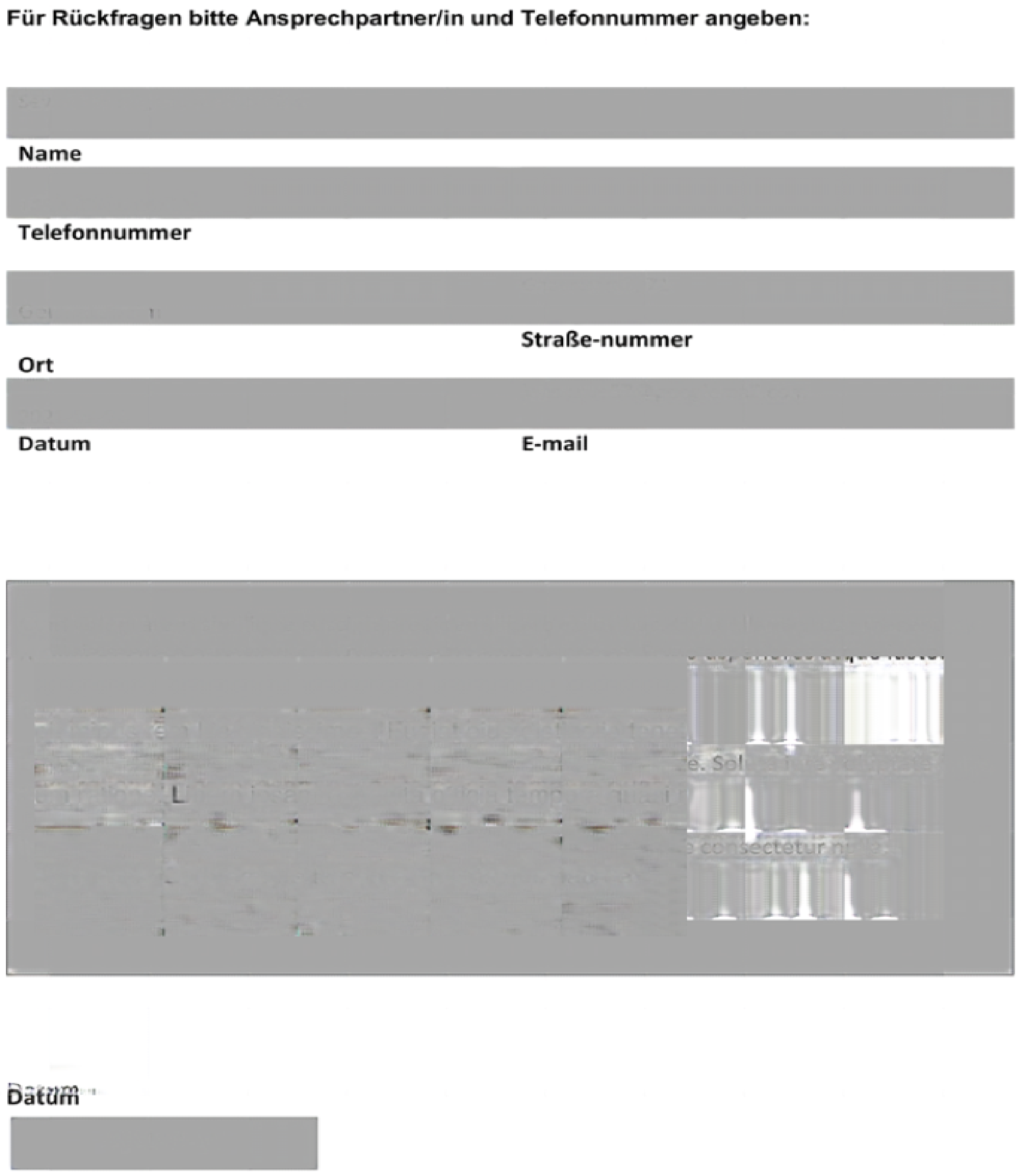
Facilis impedit qu

(c)

(d)

Figure 5.11: Predicted empty tiles with ResNet model, Epochs=500

After rejoining the tiles generated by the ResNet model, Figure 5.12 displays a complete empty form that has been successfully generated.



Für Rückfragen bitte Ansprechpartner/in und Telefonnummer angeben:

Name

Telefonnummer

Straße-nummer

Ort

Datum

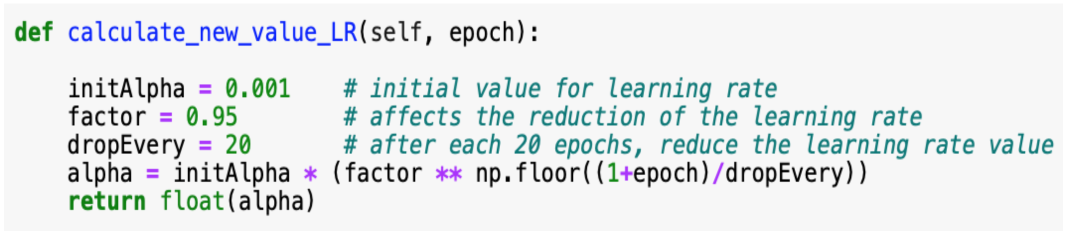
E-mail

Datum

Figure 5.12: Rejoining the tiles of a form

As shown in Figure 5.12, the ResNet model due to the wide range of training data, can distinguish dynamic content and partially remove it. However, there is still a challenge in reconstructing the background. Therefore, instead of using a constant learning rate, I utilized a learning rate scheduler.

In the Figure 5.13, you can see the code of Learning rate scheduler.



```python
def calculate_new_value_LR(self, epoch):

    initAlpha = 0.001    # initial value for learning rate
    factor = 0.95        # affects the reduction of the learning rate
    dropEvery = 20       # after each 20 epochs, reduce the learning rate value
    alpha = initAlpha * (factor ** np.floor((1+epoch)/dropEvery))
    return float(alpha)
```

Figure 5.13: Python code in Keras for Learning Rate Scheduler

To calculate the new value of the learning rate after every 20 epochs, the following steps were taken:

- Initialize the first value for the learning rate.
- Determine the extent to which the learning rate should be reduced.
- Define a new parameter for reducing the learning rate value after every 20 epochs.
- Calculate the new value of the learning rate.

After modifying the code and implementing the learning rate scheduler along with new hyper-parameters, the dataset was trained. The updated values of the hyper-parameters can be found in Table 5.4.

Table 5.4: The new hyper parameters value for ResNet Generator

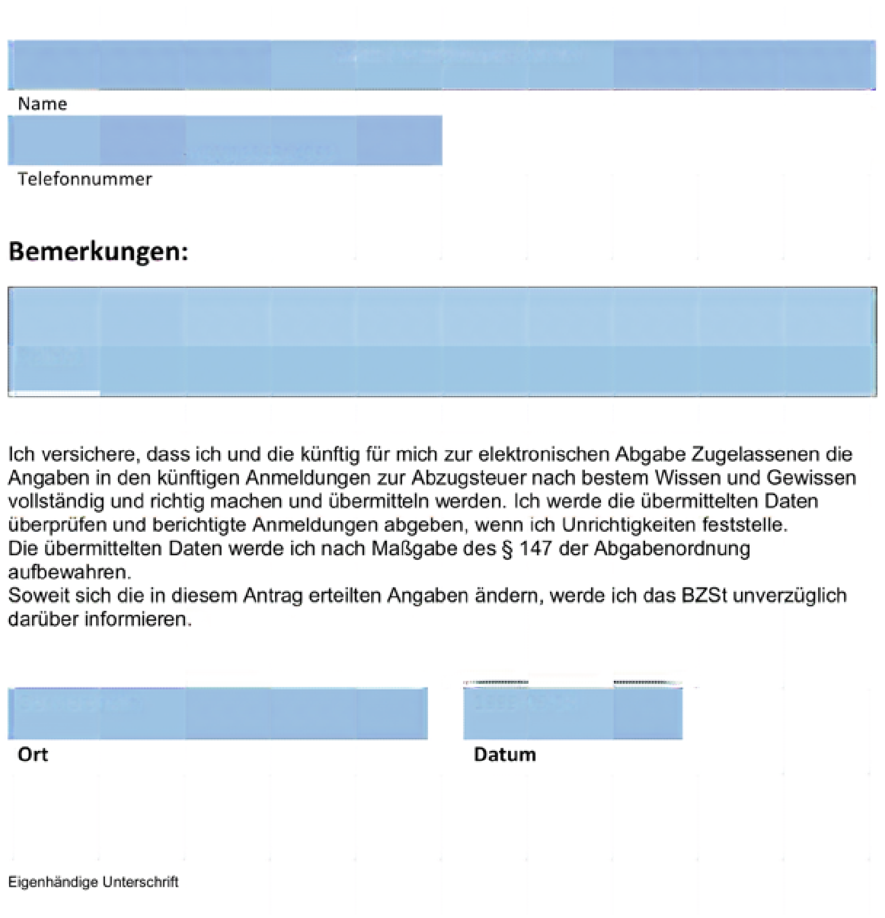
The name of hyper parameters	Value
The name of generator	ResNet
The size of the image	(128*128*3)
The number of Epochs	500
Learning rate	Learning Rate Scheduler
The number of filters for discriminator	64
The number of filters for generator	32
Batch Size	32

After predicting the tiles and assembling them, the resulting complete form can be seen in Figure 5.14. The above predictions highlight two major issues in the training and prediction process, which are described below:

- The use of different colors for coloured forms results in tiles being predicted with different colors. Each tile lacks information about the colors of its neighbouring tiles.

  Possible solution:

- Change the tile size.
- Modify the evaluation method during training by using complete forms instead of individual tiles.


Name

Telefonnummer

Bemerkungen:

Ich versichere, dass ich und die künftig für mich zur elektronischen Abgabe Zugelassenen die Angaben in den künftigen Anmeldungen zur Abzugsteuer nach bestem Wissen und Gewissen vollständig und richtig machen und übermitteln werden. Ich werde die übermittelten Daten überprüfen und berichtigte Anmeldungen abgeben, wenn ich Unrichtigkeiten feststelle.
Die übermittelten Daten werde ich nach Maßgabe des § 147 der Abgabenordnung aufbewahren.
Soweit sich die in diesem Antrag erteilten Angaben ändern, werde ich das BZSt unverzüglich darüber informieren.

Ort

Datum

Eigenhändige Unterschrift

Figure 5.14: Predicted empty form with new using learning rate scheduler

After modifying the code and using tiles with a new size of 512*512*3, along with new hyper-parameters, the dataset was trained. The updated values of the hyper-parameters can be found in Table 5.5.

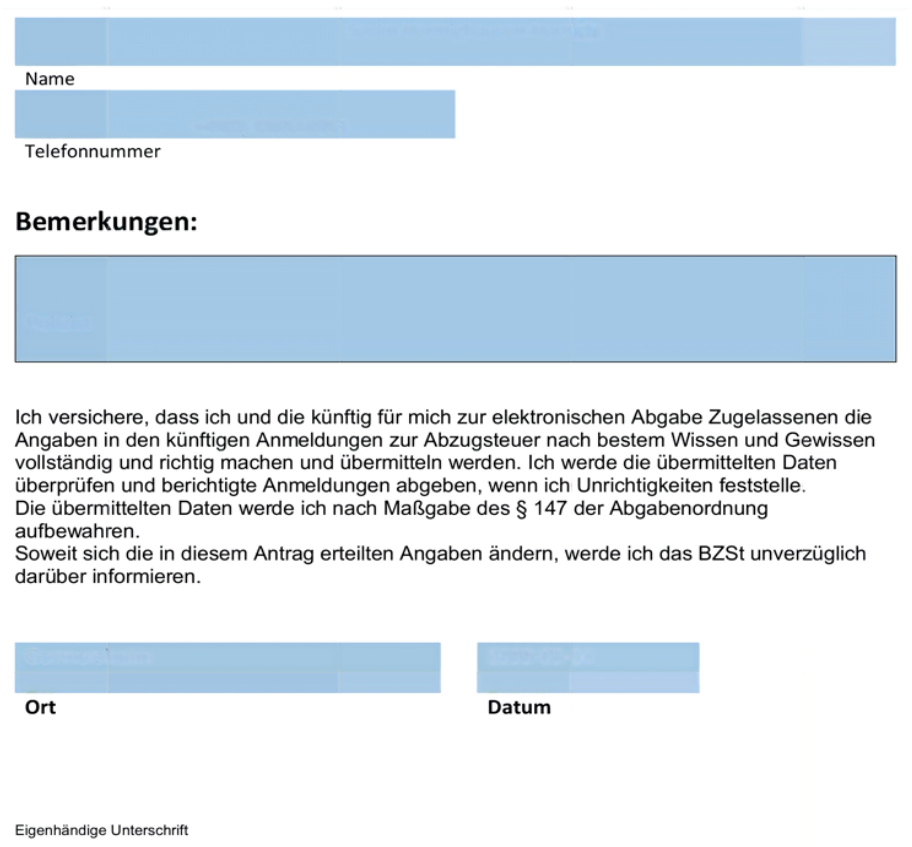
However, due to the increase in the tile size from 128*128*3 to 512*512*3, the maximum batch size that can be used is 5, limited by the available RAM.

After predicting the tiles and assembling them, the resulting complete form can be seen in Figure 5.15.

After predicting the tiles and joining them, you can find the predicted whole form in Figure 5.15.

Table 5.5: The new hyper parameters value for ResNet Generator with new tile size

The name of hyper parameters	Value
The name of generator	ResNet
The size of the image	(512*512*3)
The number of Epochs	500
Learning rate	Learning Rate Scheduler
The number of filters for discriminator	64
The number of filters for generator	32
Batch Size	5



Name

Telefonnummer

Bemerkungen:

Ich versichere, dass ich und die künftig für mich zur elektronischen Abgabe Zugelassenen die Angaben in den künftigen Anmeldungen zur Abzugsteuer nach bestem Wissen und Gewissen vollständig und richtig machen und übermitteln werden. Ich werde die übermittelten Daten überprüfen und berichtigte Anmeldungen abgeben, wenn ich Unrichtigkeiten feststelle.
Die übermittelten Daten werde ich nach Maßgabe des § 147 der Abgabenordnung aufbewahren.
Soweit sich die in diesem Antrag erteilten Angaben ändern, werde ich das BZSt unverzüglich darüber informieren.

Ort **Datum**

Eigenhändige Unterschrift

Figure 5.15: Predicted empty form with new value of tile (512*512*3)

5.6 Result

You can find the report and result of experiments in the table 5.6:

Table 5.6: The report and result of experiments

Generator-type	Tile Size	Epochs	Learning-Rate	D-Filters	G-Filters	SSIM score
U-Net	(128*128*3)	200	0.0002	32	32	0.02251853
U-Net	(128*128*3)	400	0.0002	32	32	0.00943532
Res-Net	(128*128*3)	400	0.0002	64	32	0.00027418
Res-Net	(128*128*3)	500	0.0002	64	32	0.00015080
Res-Net	(128*128*3)	500	Learning-Rate-scheduler	64	32	0.00007836
Res-Net	(512*512*3)	500	Learning-Rate-scheduler	64	32	0.00003809

As shown in Table 5.6, the best SSIM score is achieved in the last row. The summary of the findings is presented below:

- Experiments that utilized the Res-Net Generator produced better empty forms compared to those using the U-Net generator, and resulted in a significant drop in the SSIM score.
- Increasing the tile size from 128*128 to 512*512 led to a significant reduction in the batch size.
- Decreasing the batch size resulted in a substantial increase in memory usage.
- After implementing the learning rate scheduler, the SSIM score between the generated and original empty forms decreased significantly.
- Increasing the number of filters for the Discriminator improved the SSIM score.

Chapter 6

Future Work

6.1 Future Work

The proposed solution can be modified in several ways, including:

- Changing the dataset from color to black and white
- Modifying the generator type and tile size of the forms
- Altering the loss function used to evaluate the results
- Preparing forms in different languages to expand the dataset

Chapter 7

Appendix A

7.1 Implemented Code

```

from __future__ import print_function, division
import scipy
import scipy.misc
from keras_contrib.layers.normalization.instancenormalization
    import InstanceNormalization
from keras.layers import Input, Dense, Reshape, Flatten, Dropout,
    Concatenate
from keras.layers import BatchNormalization, Activation,
    ZeroPadding2D, Add
from keras.layers.advanced_activations import LeakyReLU, ELU
from keras.layers.convolutional import UpSampling2D, Conv2D,
    Conv2DTranspose
from keras.layers.merge import add
#from layers import ReflectionPadding2D
from keras.models import Sequential, Model
from keras.initializers import RandomNormal
from keras.optimizers import Adam
from keras import backend as K
#from loaders import DataLoader
from keras.utils import plot_model
import datetime
import matplotlib.pyplot as plt
import sys
import numpy as np
import os
import pickle as pkl
import random

from collections import deque


class CycleGAN():
    def __init__(self, input_dim, learning_rate, lambda_validation,
                 lambda_reconstr, lambda_id, generator_type,
                 gen_n_filters, disc_n_filters, buffer_max_length =
     50):

        self.input_dim = input_dim
        self.learning_rate = learning_rate
        self.buffer_max_length = buffer_max_length
        self.lambda_validation = lambda_validation
        self.lambda_reconstr = lambda_reconstr
        self.lambda_id = lambda_id
        self.generator_type = generator_type    # unet or resnet
        self.gen_n_filters = gen_n_filters      # the number of 
    filter for generator
        self.disc_n_filters = disc_n_filters    # the number of 
    filter for discriminator

        # Input shape
        self.img_rows = input_dim[0]
        self.img_cols = input_dim[1]
        self.channels = input_dim[2]
        self.img_shape = (self.img_rows, self.img_cols, self.
    channels)

        self.d_losses = []
        self.g_losses = []
        self.epoch = 0

        self.buffer_A = deque(maxlen = self.buffer_max_length)
        self.buffer_B = deque(maxlen = self.buffer_max_length)

        # Calculate output shape of D (PatchGAN)
        patch = int(self.img_rows / 2**3)
        self.disc_patch = (patch, patch, 1)

        self.weight_init = RandomNormal(mean=0., stddev=0.02)

        self.compile_models()


    def compile_models(self):

        # Build and compile the discriminators
        self.d_A = self.build_discriminator()
        self.d_B = self.build_discriminator()

        self.d_A.compile(loss='mse', optimizer=Adam(self.
    learning_rate, 0.5), metrics=['accuracy'])
        self.d_B.compile(loss='mse', optimizer=Adam(self.
    learning_rate, 0.5), metrics=['accuracy'])


        # Build the generators
        if self.generator_type == 'unet':
            self.g_AB = self.build_generator_unet()
            self.g_BA = self.build_generator_unet()
        else:
            self.g_AB = self.build_generator_resnet()
            self.g_BA = self.build_generator_resnet()

        # For the combined model we will only train the generators
        self.d_A.trainable = False
        self.d_B.trainable = False

        # Input images from both domains
    img_A = Input(shape=self.img_shape)
    img_B = Input(shape=self.img_shape)

    # Translate images to the other domain
    fake_B = self.g_AB(img_A)
    fake_A = self.g_BA(img_B)

    # Discriminators determines validity of translated images
    valid_A = self.d_A(fake_A)
    valid_B = self.d_B(fake_B)

    # Translate images back to original domain
    reconstr_A = self.g_BA(fake_B)
    reconstr_B = self.g_AB(fake_A)

    # Identity mapping of images
    img_A_id = self.g_BA(img_A)
    img_B_id = self.g_AB(img_B)


    # Combined model trains generators to fool discriminators
    self.combined = Model(inputs=[img_A, img_B],
                          outputs=[valid_A, valid_B,
                                   reconstr_A, reconstr_B,
                                   img_A_id, img_B_id])

    self.combined.compile(loss=['mse', 'mse',
                                'mae', 'mae',
                                'mae', 'mae'],
                        loss_weights=[self.lambda_validation,
self.lambda_validation,
                                        self.lambda_reconstr,
self.lambda_reconstr,
                                        self.lambda_id, self.
lambda_id ],
                        optimizer=Adam(0.0002, 0.5))

    self.d_A.trainable = True
    self.d_B.trainable = True


def build_generator_unet(self):

    def downsample(layer_input, filters, f_size=4):
        d = Conv2D(filters, kernel_size=f_size, strides=2,
padding='same')(layer_input)
        d = InstanceNormalization(axis = -1, center = False,
scale = False)(d)
        d = Activation('relu')(d)

        return d

    def upsample(layer_input, skip_input, filters, f_size=4,
dropout_rate=0):
        u = UpSampling2D(size=2)(layer_input)
        u = Conv2D(filters, kernel_size=f_size, strides=1,
padding='same')(u)
```

```python
            u = InstanceNormalization(axis = -1, center = False, scale = False)(u)
            u = Activation('relu')(u)
            if dropout_rate:
                u = Dropout(dropout_rate)(u)

            u = Concatenate()([u, skip_input])
            return u

        # Image input
        img = Input(shape=self.img_shape)

        # Downsampling
        d1 = downsample(img, self.gen_n_filters)
        d2 = downsample(d1, self.gen_n_filters*2)
        d3 = downsample(d2, self.gen_n_filters*4)
        d4 = downsample(d3, self.gen_n_filters*8)

        # Upsampling
        u1 = upsample(d4, d3, self.gen_n_filters*4)
        u2 = upsample(u1, d2, self.gen_n_filters*2)
        u3 = upsample(u2, d1, self.gen_n_filters)

        u4 = UpSampling2D(size=2)(u3)
        output_img = Conv2D(self.channels, kernel_size=4, strides=1, padding='same', activation='tanh')(u4)

        return Model(img, output_img)


    def build_generator_resnet(self):

        def conv7s1(layer_input, filters, final):
            y = ReflectionPadding2D(padding =(3,3))(layer_input)
            y = Conv2D(filters, kernel_size=(7,7), strides=1, padding='valid', kernel_initializer = self.weight_init)(y)
            if final:
                y = Activation('tanh')(y)
            else:
                y = InstanceNormalization(axis = -1, center = False, scale = False)(y)
                y = Activation('relu')(y)
            return y

        def downsample(layer_input,filters):
            y = Conv2D(filters, kernel_size=(3,3), strides=2, padding='same', kernel_initializer = self.weight_init)(layer_input)
            y = InstanceNormalization(axis = -1, center = False, scale = False)(y)
            y = Activation('relu')(y)
            return y

        def residual(layer_input, filters):
            shortcut = layer_input
            y = ReflectionPadding2D(padding =(1,1))(layer_input)
            y = Conv2D(filters, kernel_size=(3, 3), strides=1,
padding='valid', kernel_initializer = self.weight_init)(y)
    y = InstanceNormalization(axis = -1, center = False,
scale = False)(y)
    y = Activation('relu')(y)

    y = ReflectionPadding2D(padding =(1,1))(y)
    y = Conv2D(filters, kernel_size=(3, 3), strides=1,
padding='valid', kernel_initializer = self.weight_init)(y)
    y = InstanceNormalization(axis = -1, center = False,
scale = False)(y)

    return add([shortcut, y])

  def upsample(layer_input,filters):
    y = Conv2DTranspose(filters, kernel_size=(3, 3),
strides=2, padding='same', kernel_initializer = self.
weight_init)(layer_input)
    y = InstanceNormalization(axis = -1, center = False,
scale = False)(y)
    y = Activation('relu')(y)

    return y


  # Image input
  img = Input(shape=self.img_shape)

  y = img

  y = conv7s1(y, self.gen_n_filters, False)
  y = downsample(y, self.gen_n_filters * 2)
  y = downsample(y, self.gen_n_filters * 4)
  y = residual(y, self.gen_n_filters * 4)
  y = residual(y, self.gen_n_filters * 4)
  y = residual(y, self.gen_n_filters * 4)
  y = residual(y, self.gen_n_filters * 4)
  y = residual(y, self.gen_n_filters * 4)
  y = residual(y, self.gen_n_filters * 4)
  y = residual(y, self.gen_n_filters * 4)
  y = residual(y, self.gen_n_filters * 4)
  y = residual(y, self.gen_n_filters * 4)
  y = upsample(y, self.gen_n_filters * 2)
  y = upsample(y, self.gen_n_filters)
  y = conv7s1(y, 3, True)
  output = y


  return Model(img, output)


def build_discriminator(self):

  def conv4(layer_input,filters, stride = 2, norm=True):
    y = Conv2D(filters, kernel_size=(4,4), strides=stride,
padding='same', kernel_initializer = self.weight_init)(
layer_input)

```

```
        if norm:
            y = InstanceNormalization(axis = -1, center = False, scale = False)(y)

        y = LeakyReLU(0.2)(y)

        return y

    img = Input(shape=self.img_shape)

    y = conv4(img, self.disc_n_filters, stride = 2, norm = False)
    y = conv4(y, self.disc_n_filters*2, stride = 2, norm = True)
    y = conv4(y, self.disc_n_filters*4, stride = 2, norm = True)
    y = conv4(y, self.disc_n_filters*8, stride = 1, norm = True)

    output = Conv2D(1, kernel_size=4, strides=1, padding='same', kernel_initializer = self.weight_init)(y)

    return Model(img, output)

def train_discriminators(self, imgs_A, imgs_B, valid, fake):

    # Translate images to opposite domain
    fake_B = self.g_AB.predict(imgs_A)
    fake_A = self.g_BA.predict(imgs_B)

    self.buffer_B.append(fake_B)
    self.buffer_A.append(fake_A)

    fake_A_rnd = random.sample(self.buffer_A, min(len(self.buffer_A), len(imgs_A)))
    fake_B_rnd = random.sample(self.buffer_B, min(len(self.buffer_B), len(imgs_B)))

    # Train the discriminators (original images = real / translated = Fake)
    dA_loss_real = self.d_A.train_on_batch(imgs_A, valid)
    dA_loss_fake = self.d_A.train_on_batch(fake_A_rnd, fake)
    dA_loss = 0.5 * np.add(dA_loss_real, dA_loss_fake)

    dB_loss_real = self.d_B.train_on_batch(imgs_B, valid)
    dB_loss_fake = self.d_B.train_on_batch(fake_B_rnd, fake)
    dB_loss = 0.5 * np.add(dB_loss_real, dB_loss_fake)

    # Total disciminator loss
    d_loss_total = 0.5 * np.add(dA_loss, dB_loss)

    return (
        d_loss_total[0]
        , dA_loss[0], dA_loss_real[0], dA_loss_fake[0]
        , dB_loss[0], dB_loss_real[0], dB_loss_fake[0]
        , d_loss_total[1]
        , dA_loss[1], dA_loss_real[1], dA_loss_fake[1]
        , dB_loss[1], dB_loss_real[1], dB_loss_fake[1]
    )

def train_generators(self, imgs_A, imgs_B, valid):

    # Train the generators
    return self.combined.train_on_batch([imgs_A, imgs_B],
                                         [valid, valid,
                                         imgs_A, imgs_B,
                                         imgs_A, imgs_B])


def train(self, data_loader, run_folder, epochs, test_A_file,
test_B_file, batch_size=1, sample_interval=50):

    start_time = datetime.datetime.now()

    # Adversarial loss ground truths
    valid = np.ones((batch_size,) + self.disc_patch)
    fake = np.zeros((batch_size,) + self.disc_patch)

    for epoch in range(self.epoch, epochs):
        for batch_i, (imgs_A, imgs_B) in enumerate(data_loader.
load_batch(batch_size)):

            d_loss = self.train_discriminators(imgs_A, imgs_B,
valid, fake)
            g_loss = self.train_generators(imgs_A, imgs_B,
valid)

            elapsed_time = datetime.datetime.now() - start_time

            # Plot the progress
            print ("[Epoch %d/%d] [Batch %d/%d] [D loss: %f,
acc: %3d%%] [G loss: %05f, adv: %05f, recon: %05f, id: %05f]
time: %s " \
                % ( self.epoch+1, epochs,
                    batch_i, data_loader.n_batches,
                    d_loss[0], 100*d_loss[7],
                    g_loss[0],
                    np.sum(g_loss[1:3]),
                    np.sum(g_loss[3:5]),
                    np.sum(g_loss[5:7]),
                    elapsed_time))

            self.d_losses.append(d_loss)
            self.g_losses.append(g_loss)
            '''
            # If at save interval => save generated image
samples
            if batch_i % sample_interval == 0:
                self.sample_images(data_loader, batch_i,
run_folder, test_A_file, test_B_file)
                self.combined.save_weights(os.path.join(
run_folder, 'weights/weights-%d.h5' % (self.epoch)))
                self.combined.save_weights(os.path.join(
run_folder, 'weights/weights.h5'))
```

```
                #self.save_model(run_folder)

            '''
        self.epoch += 1

def sample_images(self, data_loader, batch_i, run_folder, test_A_file, test_B_file):

    r, c = 2, 4

    for p in range(2):

        if p == 1:
            imgs_A = data_loader.load_data(domain="A", batch_size=1, is_testing=True)
            imgs_B = data_loader.load_data(domain="B", batch_size=1, is_testing=True)
        else:
            imgs_A = data_loader.load_img('%s/%s' % (data_loader.dataset_name, test_A_file))
            imgs_B = data_loader.load_img('%s/%s' % (data_loader.dataset_name, test_B_file))

        # Translate images to the other domain
        fake_B = self.g_AB.predict(imgs_A)
        fake_A = self.g_BA.predict(imgs_B)

        # Translate back to original domain
        reconstr_A = self.g_BA.predict(fake_B)
        reconstr_B = self.g_AB.predict(fake_A)

        # ID the images
        id_A = self.g_BA.predict(imgs_A)
        id_B = self.g_AB.predict(imgs_B)

        gen_imgs = np.concatenate([imgs_A, fake_B, reconstr_A, id_A, imgs_B, fake_A, reconstr_B, id_B])

        # Rescale images 0 - 1
        gen_imgs = 0.5 * gen_imgs + 0.5
        gen_imgs = np.clip(gen_imgs, 0, 1)

        titles = ['Original', 'Translated', 'Reconstructed', 'ID']
        fig, axs = plt.subplots(r, c, figsize=(25,12.5))
        cnt = 0
        for i in range(r):
            for j in range(c):
                axs[i,j].imshow(gen_imgs[cnt])
                axs[i, j].set_title(titles[j])
                axs[i,j].axis('off')
                cnt += 1
        fig.savefig(os.path.join(run_folder ,"images/%d_%d_%d.png" % (p, self.epoch, batch_i)))
        plt.close()


```

```python
    def plot_model(self, run_folder):
        plot_model(self.combined, to_file=os.path.join(run_folder ,'viz/combined.png'), show_shapes = True, show_layer_names = True)
        plot_model(self.d_A, to_file=os.path.join(run_folder ,'viz/d_A.png'), show_shapes = True, show_layer_names = True)
        plot_model(self.d_B, to_file=os.path.join(run_folder ,'viz/d_B.png'), show_shapes = True, show_layer_names = True)
        plot_model(self.g_BA, to_file=os.path.join(run_folder ,'viz/g_BA.png'), show_shapes = True, show_layer_names = True)
        plot_model(self.g_AB, to_file=os.path.join(run_folder ,'viz/g_AB.png'), show_shapes = True, show_layer_names = True)


    def save(self, folder):

        with open(os.path.join(folder, 'params.pkl'), 'wb') as f:
            pkl.dump([
                self.input_dim
                ,  self.learning_rate
                ,  self.buffer_max_length
                ,  self.lambda_validation
                ,  self.lambda_reconstr
                ,  self.lambda_id
                ,  self.generator_type
                ,  self.gen_n_filters
                ,  self.disc_n_filters
                ], f)

        self.plot_model(folder)


    def save_model(self, run_folder):


        self.combined.save(os.path.join(run_folder, 'model.h5'))
        self.d_A.save(os.path.join(run_folder, 'd_A.h5'))
        self.d_B.save(os.path.join(run_folder, 'd_B.h5'))
        self.g_BA.save(os.path.join(run_folder, 'g_BA.h5'))
        self.g_AB.save(os.path.join(run_folder, 'g_AB.h5'))

        pkl.dump(self, open(os.path.join(run_folder, "obj.pkl"), "wb"))

    def load_weights(self, filepath):
       self.combined.load_weights(filepath)
```

Bibliography

[1] Tursun, Osman et al.
"MTRNet++: One-stage Mask-based Scene Text Eraser." . Computer Vision and Image Understanding. 201 (2020): 103066.

[2] Li, X., Chan Lu, D. Cheng, Wei-Hong Li, Mei Cao, Bo Liu, Jiechao Ma and W. Zheng.
"Towards Photo-Realistic Visible Watermark Removal with Conditional Generative Adversarial Networks. ICIG (2019).

[3] Nakamura, Toshiki, Anna Zhu, Keiji Yanai and S. Uchida.
"Scene Text Eraser." 2017 14th IAPR International Conference on Document Analysis and Recognition (ICDAR) 01 (2017): 832-837.

[4] Huang, W. R., Yike Qi, Qianqian Li and Jonathan Degange.
"DeepErase: Weakly Supervised Ink Artifact Removal in Document Text Images." 2020 IEEE Winter Conference on Applications of Computer Vision (WACV) (2020): 3511-3519.

[5] Ibtehaz, N. and Mohammad Sohel Rahman.
"MultiResUNet : Rethinking the U-Net Architecture for Multimodal Biomedical Image Segmentation." Neural networks : the official journal of the International Neural Network Society 121 (2020): 74-87 .

[6] Mazaheri, Ghazal, Niluthpol Chowdhury Mithun, Jawadul H. Bappy and A. Roy-Chowdhury.
"A Skip Connection Architecture for Localization of Image Manipulations.". CVPR Workshops (2019).

[7] David Foster, *Generative Deep Learning, Teaching Machines to paint, write, compose and play*, First edition, July 2019.

[8] Different types of neural networks in Deep Learning,
`https://www.analyticsvidhya.com/blog/2020/02/cnn-vs-rnn-vs-mlp-analyzing-3-types-of-neural-networks-in-deep-learning/`

[9] S. Albawi, T. A. Mohammed and S. Al-Zawi. *"Understanding of a convolutional neural network,"*. 2017 International Conference on Engineering and Technology (ICET), 2017, pp. 1-6, doi: 10.1109/ICEngTechnol.2017.8308186.

[10] N. Aloysius and M. Geetha, *"A review on deep convolutional neural networks,"* · 2017 International Conference on Communication and Signal Processing (ICCSP), 2017, pp. 0588-0592, doi: 10.1109/ICCSP.2017.8286426.

[11] A. Elhassouny and F. Smarandache, *"Trends in deep convolutional neural Networks architectures: a review,"*. 2019 International Conference of Computer Science and Renewable Energies (ICCSRE), 2019, pp. 1-8, doi: 10.1109/ICCSRE.2019.8807741.

[12] Liu Q. et al. (2017), *"A Review of Image Recognition with Deep Convolutional Neural Network."* In: Huang DS., Bevilacqua V., Premaratne P., Gupta P. (eds) Intelligent Computing Theories and Application. ICIC 2017. Lecture Notes in Computer Science, vol 10361. Springer, Cham. https://doi.org/10.1007/978-3-319-63309-1-7.

[13] W. James Murdocha, C. Singhb, K. Kumbiera, R. Abbasi-Aslb, and B. Yua, *"Definitions, methods, and applications in interpretable machine learning "*, 2019

[14] Deep Learning Neural Network,
`https://www.google.com/url?sa=trct=jq=esrc=ssource=webcd=ved=2ahUKEwj -9uGW17PyAhWhhf0HHXQrD78QFnoECBoQAwurl`

[15] Comparing Between CNN, RNN, ANN
`https://www.analyticsvidhya.com/blog/2020/02/cnn-vs-rnn-vs-mlp-analyzing-3-types-of-neural-networks-in-deep-learning/`

[16] Z. Pan, W. Yu, X. Yi, A. Khan, F. Yuan and Y. Zheng, *"Recent Progress on Generative Adversarial Networks (GANs): A Survey,"* in IEEE Access, vol. 7, pp. 36322-36333, 2019, doi: 10.1109/ACCESS.2019.2905015.

[17] Peter Flach, *"Machine Learning: The Art and Science of Algorithms That Make Sense of Data, isbn: 9780511973000. doi: 10.1017/CBO9780511973000 (cit. on pp. 12, 14),"* Jan. 2012.

[18] Aurelien Geron, *"Hands-On Machine Learning with Scikit-Learn, Keras, and TensorFlow, 2nd Edition, "* 2019. isbn: 9781492032649 (cit. on pp. 22,24,25,29).

[19] Alex Sherstinsky, *"Fundamentals of Recurrent Neural Network (RNN) and Long Short-Term Memory (LSTM) network, "* In: Physica D: Nonlinear Phenomena 404 (Mar. 2020), p. 132306. doi: 10.1016/ j.physd.2019. 132306 (cit. on p. 22).

[20] Tom Heskes, Bert Kappen, *"Learning processes in neural networks, "* In:Physical review. A 44 (Sept. 1991), pp. 2718–2726. doi: 10.1103/PhysRevA. 44.2718 (cit. on pp. 22, 23).

[21] Z. You, B. Xu., *"Improving training time of deep neural network with asynchronous averaged stochastic gradient descent, "* In: The 9th International Symposium on Chinese Spoken Language Processing. 2014, pp. 446–449.

[22] Anand Kumar Maurya, *"A supervised CNNs based approach to restore empty form images from machine-printed filled form images, "* In Association with Lang.Tec, Semantic Text Processing 20146 Hamburg, Germany , 2021.