UCLouvain

École polytechnique de Louvain

# A data-centric approach to information extraction of expenses

Authors: **Anthony Verriest, Marcin Wilk**
Supervisor: **Axel Legay**
Readers: **Louis Baudoux, Florian Daloze, Pierre Schaus**
Academic year 2022–2023
Master [120] in Computer Science

# Acknowledgements

We would like to express our gratitude to our thesis supervisor, Professor Axel Legay, for his guidance, support, and encouragement throughout our work. We are also grateful to the members of Odoo's In-App Purchase (IAP) team, Florian Daloze, and Louis Baudoux, for their valuable contributions and assistance in the development of this thesis and the source code. In a larger sense, we are grateful to the Odoo company for allowing us to have a place to work and a personal laptop to conduct the different experiments. Moreover, we are grateful to Bogdan Ginter for providing valuable grammatical and spelling feedback and suggestions on early drafts of this thesis. Finally, we would like to thank our friends and family for their support and encouragement during the writing of this thesis.

# Abstract

The study of automatic processing and comprehension of business documents is known as Document AI. It covers activities such as reading and evaluating a document's content. Working with business papers presents several difficulties, including dealing with their many formats and intricate layouts. This paper aims to review the state-of-the-art Deep Learning models that are used in the field of Document AI and the techniques employed to improve their performances. This work represents a concrete case of information extraction from expenses, done as part of a master's thesis in collaboration with Odoo. The employed methodology falls under the data-centric approach, which is a new arising method that advocates focusing on the data to a greater extent instead working heavily on the model. The Deep Learning models that will be presented all extend the pioneering BERT model, a predominant Natural Language Processing (NLP) model from recent years with the breakthrough of Transformers. Such models are firstly pre-trained with multiple types of data before being fine-tuned on a custom dataset, in this case, images of expenses. The creation of the specialized training dataset will be the core of the data-centric approach. This data-centric workflow is used to evaluate the performance of different models and identify the one that performs the best. Moreover, the model's predictions are checked for faults or inaccuracies to update and improve the image processing techniques. This will include an examination of the post-processing methods used to parse the predictions made by the models. Additionally, a discussion of the overall performance and the limitations of the data-centric approach will be initiated. In the end, the different business values that the best model leverages will be presented.

# Acronyms

**AI** Artificial Intelligence. 2, 7, 27, 29, 30, 34, 37, 43–45, 80, 81

**BERT** Bidirectional Encoder Representations from Transformers. 31–34, 38, 40, 41, 66

**BiACM** Bidirectional Attention Complementing Mechanism. 38

**BIO** Begin-Inside-Outside. 29, 57, 58

**BLSTM** Bidirectional Long Short-Term Memory Networks. 20

**BPTT** Back Propagation Through Time. 19

**CAI** Cross-modal Alignment Identification. 38

**CMYK** Cyan, Magenta, Yellow, Black. 6

**CNN** Convolutional Neural Networks. 16, 17, 26, 34, 35, 38, 67, 69, 70

**CORD** Consolidated Receipt Dataset for Post-OCR Parsing. 39–41, 56, 65, 78

**CTC** Connectionist Temporal Classification. 26

**DL** Deep Learning. 2, 3, 5, 11–13, 15, 29, 30, 62, 67, 80, 81

**FUNSD** Form Understanding in Noisy Scanned Documents. 39–41, 56

**IAP** In-App Purchase. 2, 41, 59, 69, 76, 78, 81

**IE** Information extraction. 5, 11, 27, 28, 80

**KPL** Key Point Location. 38

**LiLT** Language-independent Layout Transformer. 37, 38, 40, 41, 45, 51, 63, 66, 67, 69–72, 76–78, 80

**LSTM** Long short-term memory. 19–21

**MIM** Masked Image Modeling. 33, 34, 36

**ML** Machine Learning. 2, 5, 7–9, 11, 13, 28, 43–46, 67, 80, 81

**MLM** Masked Language Model. 32–34, 36

**MLP** Multi Layer Perceptron. 33

**MVLM** Masked Visual-Language Modeling. 38

**NER** Named-entity recognition. 29, 31, 40, 60, 66, 67, 77

**NLP** Natural Language Processing. 27–30, 32

**NSP** Next Sentence Prediction. 32

**OCR** Optical Character Recognition. 5, 25–27, 29, 32, 38, 45–48, 55, 58, 59, 63, 66, 69, 76, 78

**ReLU** Rectified Linear Unit. 13, 17

**RGB** Red, Green, Blue. 7, 17

**RNN** Recurrent Neural Network. 18, 19, 21, 26

**SGD** Stochastic Gradient Descent. 14–16

**ViT** Vision Transformer. 34, 67, 69

**WPA** Word-patch alignment. 33, 34, 36

# List of Figures

# List of Tables

# Contents

# Chapter 1

# Introduction

The world is rapidly becoming more digital. The rise of big data and new technologies provides more opportunities to unlock information from various sources for various purposes. For instance, invoices and receipts often contain the same essential information, but their layout and format vary greatly, making it difficult to build systems that automatically extract key information. Furthermore, most businesses use digital accounting software to manage their bookkeeping. This requires someone to manually input data from a physical expense into the software to store the details of each purchase. In this context, an expense is defined as a justification for the outflow of cash from a company to an individual such as receipts, parking tickets, or screenshots of Uber rides. Invoices, which are bills issued by a seller to a buyer for a sale transaction, can also be considered expenses and typically include a description of the goods or services being sold, the prices, and the terms of sale.

Odoo, a well-known Belgian suite of business management software containing diverse tools including CRM, ERP, and Accounting, allows their clients to encode any type of bill manually. To improve the client's experience, Odoo built a new system to automatically import the data from expenses into the client's database. For invoices, Odoo's internal solution works as expected, reaching nearly perfect accuracy in detecting the requested features, e.g. the supplier name, the invoice lines, the date, the due date, the invoice ID, the currency, the payment reference, and the IBAN account. All of this is done using a custom algorithm developed by Odoo. However, regarding expenses, their algorithm performs poorly.

Building an expense parsing system that can efficiently extract key features can be quite challenging, as expenses have all unique layouts and styles. In addition, expenses may be blurred, handwritten, rotated, cropped, or captured with a phone camera in different lighting and perspectives. The difficulties encountered by the

differences presented in the expenses are not the only obstacle. Collecting useful data is another critical aspect to consider when building effective Artificial Intelligence (AI) models. A data-centric approach focuses primarily on improving the quality of the acquired data to get the best performance from these AI models.

To address the challenge of this project, we collaborated with the Odoo In-App Purchase (IAP) team. We have worked at Odoo's headquarters in Grand-Rosière-Hôtômont twice a week and remotely the rest of the time. The project took five months to complete, with the first month being dedicated to understanding the work environment and research needed for the thesis. The following months were devoted to mastering the necessary tools, namely Hugging Face and OpenCV, which will be presented later, and developing the source code. We spent a significant amount of time understanding and using OpenCV for image manipulation and processing. Moreover, we also used the knowledge of our Machine Learning (ML) and Deep Learning (DL) courses at UCLouvain to overcome obstacles in our Machine Learning work. In particular, these courses allowed us to have a first look at some famous DL architectures as well as datasets handling. Finally, our software engineering courses taught us useful design patterns, including inheritance, which was implemented in the source code, and some organizational tools, such as Trello or Confluence, which were also used.

We mainly worked autonomously, sometimes in a pair programming way. Our work was done locally without pushing any code to Odoo's repository. The deployment will be done by the IAP team. We met with the IAP team leader on a weekly or biweekly basis to track our progress.

## 1.1 Thesis Objectives

To improve document understanding and build a high-performing Document AI model, incorporating multiple types of features from expenses can be beneficial, as text alone does not provide all the necessary information. For instance, new patterns could be detected in visual or positional features from different types of data. It is known as multimodal data.

This thesis will explore ways to effectively learn from multimodal data using a data-centric approach to essentially focus on the quality of the data rather than on the model tuning. The ultimate goal of this thesis is to find a way to improve the overall performance of Odoo's extract module regarding expense parsing and key information extraction of the total, the date and the currency fields. For this work, the expenses entered by Odoo's clients into the Accounting tool will be used as the data source. In addition to this general overview, more specific research questions

will be investigated.

1. Which multimodal extraction model will give satisfying results when considering expenses?
2. How the newly arising data-centric approach allows us to tackle such a problem?

## 1.2 Outline

The thesis's content can be roughly divided into three sections. The first section covers some fundamentals of Deep Learning and additional topics that are important for understanding the material in the thesis. It also presents some state-of-the-art DL models used for document understanding. The second section describes the methodology used to address the problem under study and explains the data-centric approach. The last section analyzes and discusses the results of the experiments, draws conclusions, presents the added business values to Odoo, and offers suggestions for further improvements.

# Chapter 2

# Background

## Contents

In this chapter, the core concepts will be reviewed to provide background knowledge for the reader. This will ensure that the following chapters are fully understood.

First of all, an introduction to the basics of a digital image as well as an explanation of its composition will be given. Then, different concepts such as Machine Learning (ML), Neural Networks, Optical Character Recognition (OCR), and Information extraction (IE) will be defined and described.

## 2.1 Digital image

Before explaining what an OCR is all about and the usage of images as input in a Deep Learning model, it is important to understand the underlying structure of a digital image.

Digitized images are representations of visual data stored and handled by computers as a set of numerical values. Images are translated into numbers by dividing them into small areas called pixels (picture elements).

As an example, Figure 2.1 shows a digital image of flowers.

Figure 2.1: A digital image of flowers taken from Stanford (2018)

By zooming in on the flower on the upper left of Figure 2.1, it becomes possible to see that it is made of many square pixels. This is illustrated in Figure 2.2

Figure 2.2: A zoomed digital image of flowers taken from Stanford (2018)

Each pixel is recorded as a number, or a small set of numbers, describing some characteristics of the pixel such as its brightness and color. These numbers are arranged in rows and columns corresponding to the vertical and horizontal positions of the pixels. (Encyclopedia, 2022)

In other words, pixels can be thought of as a grid of numbers, each entry identified by an (x, y) coordinate. For example, a 600 x 400 image is 600 pixels wide and 400 pixels high, for a total of 240,000 pixels. The more pixels an image has, the more information it contains and the more details it can represent. This grid representation is shown in Figure 2.3.

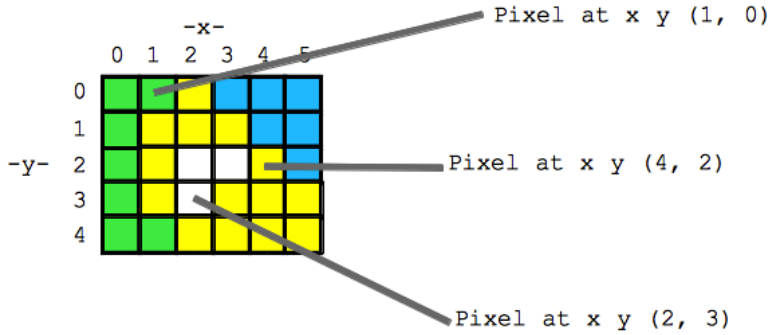



Figure 2.3: The image's grid representation taken from Stanford (2018)

One of the basic characteristics of a digital image is its type, or more specifically its color system. For instance, black-and-white images record only the intensity of light falling into the pixels. The Cyan, Magenta, Yellow, Black (CMYK) type

provides a four-color image representation. Another well-known system, the Red, Green, Blue (RGB) type, has a three-color representation.

As the RGB type is the most frequent one, it will be used in the following chapters.

In this color scheme, a pixel is described by a triplet (r, g, b) where the values of r, g, and b (which stand for Red, Green, and Blue) are located in the range from 0 to 255 (coded in 8 bits), with 0 meaning no light and 255 meaning maximum light. As an example, (red = 255, green = 150, blue = 0) is a color where red is its maximum value, green is its medium value and blue is not present at all. This color results in a shade of orange. When proceeding as such, any color can be formed as red, green, and blue color components (Stanford, 2018).

It is crucial to realize that a computer must first dissect an image into its component pixels to examine it. This procedure can make it difficult for a computer to recognize objects, shapes, or text in the image. Humans, on the other hand, can rapidly recognize these components in an image without having to break them down into individual pixels.

## 2.2 Machine Learning

### 2.2.1 Introduction

Humans learn from past experiences, whereas computers are programmed to follow instructions. Machine Learning (ML) is the intersection between the two where the machines are programmed in a way that allows them to learn from experience. For computers, past experience is data; thus, in short, a ML algorithm is an algorithm that can learn from data. In our case, the data are a set of pictures of expenses, i.e. receipts, invoices, etc. Machine Learning belongs to the Artificial Intelligence field. By taking a more formal definition of ML, Mitchell (1997) describes ML as a study of computer algorithms that improve automatically through experience. Following his definition:

"A computer program is said to learn from experience $\mathcal{E}$ with respect to some class of tasks $\mathcal{T}$ and performance measure $\mathcal{P}$, if its performance on tasks in $\mathcal{T}$, as measured by $\mathcal{P}$, improves with experience $\mathcal{E}$."

Often, ML tasks are described in terms of how the Machine Learning system should process an example. An "example", in the AI vocabulary, is a collection of features that have been quantitatively measured from some object or event needed for the Machine Learning system to process. An example is usually represented

as a vector $x \in \mathbb{R}^n$ where each entry $x_i$ of the vector is a feature. For instance, some features of an expense image will be the values of the pixels in the image. (Goodfellow et al., 2017)

Many kinds of well-known ML tasks have been studied. A small overview of these tasks is listed below.

- **Classification**: A task that assigns a category to each item.
- **Regression**: A task that assigns a real value for each item.
- **Clustering**: A task that divides a set of items into one or several homogeneous subsets.
- **Dimension reduction**: A task that tackles the problem of transforming an initial set of items into a lower-dimensional representation while preserving its properties.

To assess a ML algorithm's abilities, a quantitative measure of its performance must be devised. This performance measure $\mathcal{P}$ is usually specific to the task $\mathcal{T}$ that the system is performing.

For tasks such as classification, the accuracy of the model is often employed as a performance metric. The accuracy represents the proportion of examples for which the model produces the correct output. Other metrics, namely precision and recall, are also utilized to better understand the overall performance of the model. More concisely, considering a basic binary classification task where predicting whether an email is spam is referred to as positive and the opposite, as negative. The precision measures the percentage of the items that the system detected (i.e., that the system labeled) that are positive (positive according to the truth labels). On the other hand, recall measures the percentage of items present in the input that were correctly identified by the system (Jurafsky and Martin, 2021). Those metrics will be discussed in further chapters as these will be picked to assess the model's performances.

Typically, the main focus of a ML algorithm is to achieve high generalization on the data learned by the model. In other words, how well it performs on new unseen data, as this determines how well it will perform when deployed in the real world.

To increase our chances of having a good generalization, the data are divided into multiple sets. Each set is a collection of many examples, as defined previously. The reason to divide the dataset into multiple ones is to provide unseen data for the model at the testing phase. Thus, data are split into a training set to train the model and a testing set to assess its performance. It is important to split the dataset at the very beginning to avoid data leakage. Data leakage occurs when

information from outside the training data set is used to build the model; hence the testing set is no longer unseen from the model. Other sets, such as a validation set, can also be sampled depending on the needs of the task. This additional set will be examined in the methodology chapter.

ML algorithms are usually classified as either unsupervised learning, supervised learning, or reinforcement learning based on the type of experience $\mathcal{E}$ or feedback they are permitted to have during the learning process. Their main differences are listed below (Russell and Norvig, 2022).

- In supervised learning, the examples have a label, also called target, associated with it. It can be seen as a supervisor who shows the machine what to do in order to learn the mapping between the input and output.
- In unsupervised learning, only a set of input data without the corresponding target values is fed to the model. The goal is to discover regularities in the input, such as retrieving groups of similar inputs in the clustering task.
- In reinforcement learning, the feedback or experience takes the form of rewards that a computer program receives after each action it takes while interacting with a dynamic environment. The program, known as an agent, learns a policy of actions that maximize cumulative rewards to achieve a high-level goal. For example, finding the best path to escape a maze. In contrast to supervised learning, the learning algorithm is not given examples of optimal outputs but must instead discover them through trial and error by exploring its environment.

In the remainder of the thesis, only supervised learning information extraction strategies will be mentioned and proposed. Therefore, only supervised learning will be explained in more detail. Note that unsupervised and reinforcement learning still shares certain fundamental principles, tools, and approaches with the supervised scenario (Bishop, 2006).

### 2.2.2 Supervised learning and Generalization

Machine Learning (ML) is fundamentally concerned with generalizations. Standard supervised learning tasks consist of making accurate predictions about unseen examples using a finite sample of labeled examples. Typically, the problem is formulated by Russell and Norvig (2022) as selecting a function $h$ from a hypothesis space $\mathcal{H}$, which is a subset of the family of all functions. The chosen function is then employed to label all instances, including unseen examples.

Soon enough, a question arises: How to choose a good hypothesis from within the hypothesis space? With a large or complex hypothesis space, the learner can select a function that is consistent with the training set, i.e. a function that makes no mistakes on the training sample. Having a less complex space will incur some errors in the training set that may be unavoidable.

The criterion for evaluating the validity of a hypothesis is not how it performs on the training set, but rather how well it handles inputs it has not yet seen, as stated previously. It can be said that $h$ generalizes well if it accurately predicts the results of the test set.

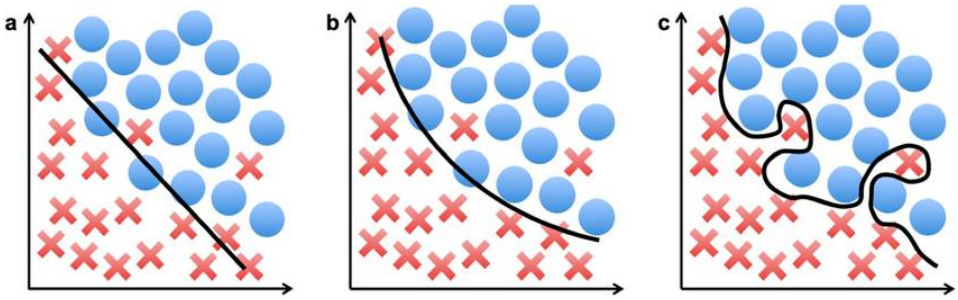



Figure 2.4: (a) An underfitting hypothesis. (b) An ideal hypothesis that identifies the underlying relationship of the data. (c) An overfitting hypothesis. All three figures are taken from Radiya-Dixit et al. (2017)

By sub-sampling the main training dataset, one way to examine hypothesis spaces is to look at the bias they impose (regardless of the training dataset) and the variance they produce (from one training sample to another). The bias, in this case, refers (loosely) to a predictive hypothesis's tendency to deviate from the expected value when averaged across different training samples. Variance, on the other hand, is referred to as the amount of change in the hypothesis caused by fluctuations in the training data.

For illustration purposes, let us take a basic supervised learning task of binary classification. The dataset is made up of examples labeled as either red crosses or blue circles.

Figure 2.4 (a) is referring to a case where the hypothesis is underfitting the data. The hypothesis is underfitting when it fails to find a pattern in the data. It is said that it has a high bias and low variance. On the other hand, in Figure 2.4 (c), the hypothesis is overfitting the data. It pays too much attention to the particular dataset on which it is trained, causing it to perform poorly on unseen data. The illustrated piecewise linear function has low bias and high variance; the shape of the function is driven by the data. (Russell and Norvig, 2022)

Generalization refers to the bias-variance trade-off: a choice between more complex, low-bias hypotheses that fit the training data well and simpler, low-variance hypotheses that may generalize better. This is exposed in Figure 2.4 (b).

## 2.3 Deep Learning

Deep Learning (DL) is a subset of Machine Learning (ML). There are many well-performing ML algorithms, including kernel-based and tree-based methods. However, when it comes to the field of Information extraction (IE), deep neural network approaches have become dominant in recent years. Hence, the methods employed in this thesis all fall under the Deep Learning paradigm. As such, a very brief and incomplete introduction will be given to bring the reader up to speed. Refer to Goodfellow et al. (2017) to have a more in-depth exposition.

DL, as a whole, is the belief that to solve complex data problems, the best approach is to use powerful deep ML models that have been trained on a large amount of data. Initially, this data is as unprocessed as possible.

Traditional ML techniques were limited in their ability to process natural raw data. For decades, building a pattern recognition or ML system required careful engineering and domain expertise to design a feature extractor that converted raw data (such as image pixel values) into a suitable internal representation or feature vector from which the learning subsystem, often a classifier, could detect or classify patterns in the input.

Deep Learning methods are representation-learning techniques that have multiple levels of representation. They are created by composing simple but non-linear modules. Each of those modules transforms the representation at one level (beginning with the raw input) into a representation of a higher, slightly more abstract level. Very complex functions can be learned by combining enough of these transformations.

This means that Deep Learning is, in fact, not conceptually bound to any particular ML model or learning algorithms. However, in practice, it is inextricably linked to neural networks and stochastic gradient descent (LeCun et al., 2015).

### 2.3.1 Neural networks overview

Neural networks are made up of a series of layers, each layer containing neurons that act as a small hypothesis function. These types of networks are often called feedforward neural networks. The objective of a feedforward network is to approximate some function $f^*$. For a classifier, as an example, $y = f^*(\boldsymbol{x})$ maps an input

$\boldsymbol{x}$ to a category $y$. A feedforward network, on the other hand, defines a mapping $y = f(x; \boldsymbol{\theta})$ and learns the parameter $\boldsymbol{\theta}$ values that result in the best function approximation.

Feedforward neural networks are referred to as networks because they are generally represented by combining many different functions. The model is linked to a directed acyclic graph that describes how the functions are assembled. Its composition is described as $f(\boldsymbol{x}) = f^{(l)}( \ ... \ f^{(2)}(f^{(1)}(\boldsymbol{x})) \ ... \ )$ where each function $f^{(i)}$ of this composition then corresponds to a layer of the network, and the depth of the model refers to the total length of this chain, $l$. From this terminology, the name Deep Learning arises. The final layer of a feedforward network is called the output layer $y$. The layers located between the input and output of the neural network are called hidden layers $h$ (Goodfellow et al., 2017).

An illustration of a feedforward neural network is given in Figure 2.5.

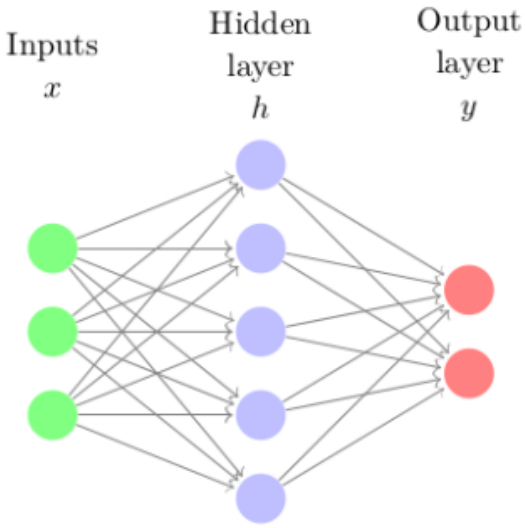



Figure 2.5: Example of a feedforward neural network taken from Dupont (2022)

Such learning hierarchies may include a consequent number of levels for input types, for instance, images, videos, or text. This implies that the number of layers may be quite large, exceeding 20 for some image recognition models such as ConvNet (LeCun et al., 2015) and a few hundred for some recent DL models.

This type of network can be intuitively understood by considering it as a linear model as described by Goodfellow et al. (2017). However, linear models also have the obvious flaw of being limited to linear functions. The model cannot understand

the interaction of two input variables. An extension to these linear models is necessary to represent non-linear functions of $\boldsymbol{x}$. To do so, a mapping of $\boldsymbol{x}$ to a transformed input can be applied as follows: $\boldsymbol{x} \rightarrow \phi(\boldsymbol{x})$, where $\phi$ is a non-linear transformation. One can think of $\phi$ as providing a set of features describing $\boldsymbol{x}$, or as providing a new representation for $\boldsymbol{x}$.

A DL algorithm attempts to learn the best mapping $\phi$. It is one of the key differences between Deep Learning and traditional Machine Learning. The latter typically relies on manually selected features, whereas the former discovers them on its own. In practice, the representation is often parameterized as $\phi(\boldsymbol{x}; \theta)$. An optimization algorithm can then learn the optimal parameters. The vast majority of neural networks use a parameterized linear transformation followed by a non-linear activation function.

The outputs $\boldsymbol{y}$ of a hidden layer having $\boldsymbol{x}$ as input are defined by $\boldsymbol{y} = g(\boldsymbol{W}^T\boldsymbol{x}+\boldsymbol{b})$ where $\boldsymbol{W}$ provides the weights of a linear transformation, $\boldsymbol{b}$ the biases and $g$ is the activation function (applied element-wise; note that the vector and scalar versions are used interchangeably).

In modern neural networks, the default recommendation for $g$ is to use the Rectified Linear Unit (ReLU) defined by the activation function $g(z) = max\{0, z\}$ where $z$ stands for $\boldsymbol{W}^T\boldsymbol{x} + \boldsymbol{b}$.

There exist other activation functions worth mentioning, more specifically, the sigmoid function defined by Mitchell (1997) as:

$$\sigma(z) = \frac{1}{1 + \exp(-z)} \tag{2.1}$$

Another popular activation function is the *softmax* function employed in the last layer of the majority of the neural network-based classifiers to represent the probability distribution over $n$ different classes. This function is given by Goodfellow et al. (2017) as:

$$\text{softmax}(z)_i = \frac{\exp(z_i)}{\Sigma_j \exp(z_j)} \tag{2.2}$$

### 2.3.2 Gradient-Based Learning

Having defined neural networks, it still requires a way of training these models by learning a set of parameters that produces good results. Therefore, a cost function $J$ is needed, when combined with gradient descent, it improves the network performance on its intended task.

Gradient descent is a method for minimizing an objective function $J(\theta)$ composed by the parameters of a model $\theta \in \mathbb{R}^n$. It proceeds by updating the parameters

in the gradient's opposite direction of the objective function $\Delta_\theta J(\theta)$. [1].

The size of the steps taken to reach a (local) minimum is determined by the learning rate $\eta$. In other words, the slope of the surface created by the objective function is descended until a valley is reached (Ruder, 2016).

### Gradient descent variants

Gradient descent has three variants that differ in how much data is utilized to compute the gradient of the objective function. It makes a trade-off between the accuracy of the parameter update and the time it takes to perform an update depending on the amount of available data.

Vanilla gradient descent, also known as batch gradient descent, computes the gradient of the cost function for the parameters $\theta$ for the entire training dataset.

$$\theta = \theta - \eta * \Delta_\theta J(\theta) \tag{2.3}$$

For convex error surfaces, batch gradient descent is guaranteed to converge to the global minimum, and for non-convex surfaces, to a local minimum. Still, in most cases, batch gradient descent can be very slow and intractable for datasets that do not fit in memory. This is because the gradients for the entire dataset need to be computed to perform just one update.

On the contrary, Stochastic Gradient Descent (SGD) updates the parameters for each training example $x^{(i)}$ and label $y^{(i)}$ :

$$\theta = \theta - \eta * \Delta_\theta J(\theta; x^{(i)}; y^{(i)}) \tag{2.4}$$

For large datasets, batch gradient descent performs redundant computations by recalculating gradients for similar examples before each parameter update. SGD eliminates this redundancy by performing one update at a time. However, the updates suffer from a high variance that makes the loss fluctuate heavily. This complicates the convergence to the minimum at the end of the optimization process.

Finally, the mini-batch gradient descent combines the best of both worlds by performing an update for each mini-batch of $n$ training examples:

$$\theta = \theta - \eta * \Delta_\theta J(\theta; x^{(i:i+n)}; y^{(i:i+n)}) \tag{2.5}$$

The mini-batch gradient reduces the variance of the parameter updates, which can lead to more stable convergence, and allows one to use advanced matrix

[1] $\theta$, in this case, refers to all the model parameters (weights and biases).

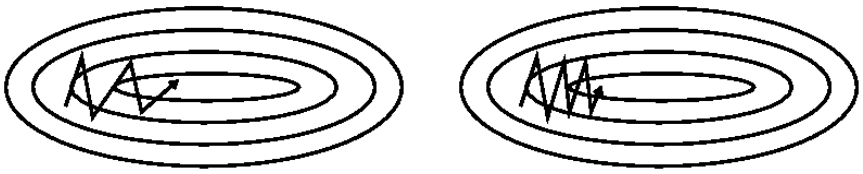
optimizations common in cutting-edge DL libraries, which makes computing the gradient for the mini-batch very efficient.

When training a neural network, the algorithm of choice is generally the mini-batch gradient descent, and the term SGD is commonly referred to when mini-batches are used.

However, vanilla mini-batch gradient descent does not guarantee good convergence and presents a few challenges that must be addressed when selecting an appropriate learning rate, adjusting the learning rate during training, or lastly avoiding getting trapped in their numerous suboptimal local minima (Ruder, 2016).

### Gradient descent optimization algorithms

While stochastic gradient descent is still a popular optimization strategy, learning with it can be time-consuming. The momentum method (Polyak, 1964) is intended to accelerate learning, particularly in the areas where the surface curves are much more steeply in one dimension than in another. In these scenarios, SGD oscillates across the ravine's slopes, making only hesitant progress along the bottom toward the local optimum (Ruder, 2016). The effect of the momentum is illustrated in Figure 2.6.



(a) SGD with momentum (b) SGD without momentum

Figure 2.6: Comparisons of SGD momentum taken from Ruder (2016)

Momentum can be interpreted as pushing a ball down a hill. As it rolls downhill, the ball gains momentum and becomes faster and faster. The momentum term increases for dimensions with gradients that point in the same direction and decreases for dimensions with gradients that change directions. As a result, there is a faster convergence and fewer oscillations (Ruder, 2016).

However, a ball that rolls down a hill, blindly following the slope, is highly unsatisfactory. Having a smarter ball, one that knows where it is going and knows when to slow down before the hill slopes up again is preferable. For this purpose, several adaptive learning rates were proposed in the literature, for instance, AdaGrad, Adadelta, or RMSprop.

In practice, the best overall choice is Adam, as stated by Kingma and Ba (2014) after empirical testing. The name Adam is derived from adaptive moment estimation. It is a method for efficient stochastic optimization that requires only first-order gradients and has low memory requirements. The method calculates individual adaptive learning rates for different parameters using estimates of the gradient's first and second moments. It combines the benefits of two other methods, as explained previously: AdaGrad, which works well with sparse gradients, and RMSProp, which works well in both online and non-stationary settings.

#### Back propagation

When implementing SGD or any other gradient-based optimizer on a neural network, the training loss gradient must be computed for the parameters of each layer. The chain rule principle is employed to compute the gradient one layer at a time. To that extent, back propagation is defined as a learning procedure that permits the adjustment of the weights of the neural network's connections to minimize a measure of the difference between the actual output vector of the network and the desired output (Rumelhart et al., 1986).

## 2.4 Main deep learning architectures

Now that neural networks were introduced, it is time to focus on some of the most popular and trending neural network architectures. Only a small batch of them will be introduced to easily digest the rest of the thesis.

### 2.4.1 Convolutional Neural Networks

Convolutional Neural Networks (CNN) or ConvNets is a type of neural network that is used to process data with a known, grid-like topology. It is intended to process data in the form of multiple arrays, such as a colored image composed of three 2D arrays containing pixel intensities in three different color channels. The term Convolutional Neural Networks refers to the use of a mathematical operation by the network: a convolution. A convolution is a subset of linear operations.

In functional analysis, the convolution is a mathematical operation on two functions $f$ and $g$ which produce a third function. The resulting function expresses how the shapes of these two functions interact. It is defined by the following integral:

$$(f * g)(t) \triangleq \int f(\tau)g(t-\tau)d\tau \tag{2.6}$$

which results in a "sliding" of the value $\tau$ of the function $g$ on the function $f$. The first argument is often referred to as the input and the second as the kernel. The output is called the feature map.

Convolutional networks are simply neural networks with at least one layer that uses convolution instead of general matrix multiplication. CNN covers four key ideas to exploit the properties of natural signals: local connections, shared weights, pooling, and the use of many layers (Delbrouck, 2019; Goodfellow et al., 2017; LeCun et al., 2015).

The architecture of a typical CNN is organized in several stages. The first few stages are made up of two types of layers: convolutional and pooling. A convolutional layer's units are organized in feature maps, within which, each unit is connected to local patches from the previous layer's feature maps via a set of weights known as a filter bank. Although the convolutional layer's role is to find local conjunctions of the preceding layer's features, the pooling layer's role is to combine semantically related features into a single feature. The relative positions of the components that make up a motif can differ slightly; hence, it is possible to accurately identify the motif by approximating the position of each feature. In a common pooling unit, the maximum of a local patch of units in a feature map is calculated. This locally weighted sum is then passed through a non-linear function, for instance, a ReLU activation function. A feature map's units all use the same filter bank. Different filter banks are employed by different feature maps in a layer.

An example of such a network is presented in Figure 2.7. It represents a convolutional network architecture that takes an RGB picture as input. In this case, an image of a Samoyed dog is taken as input.

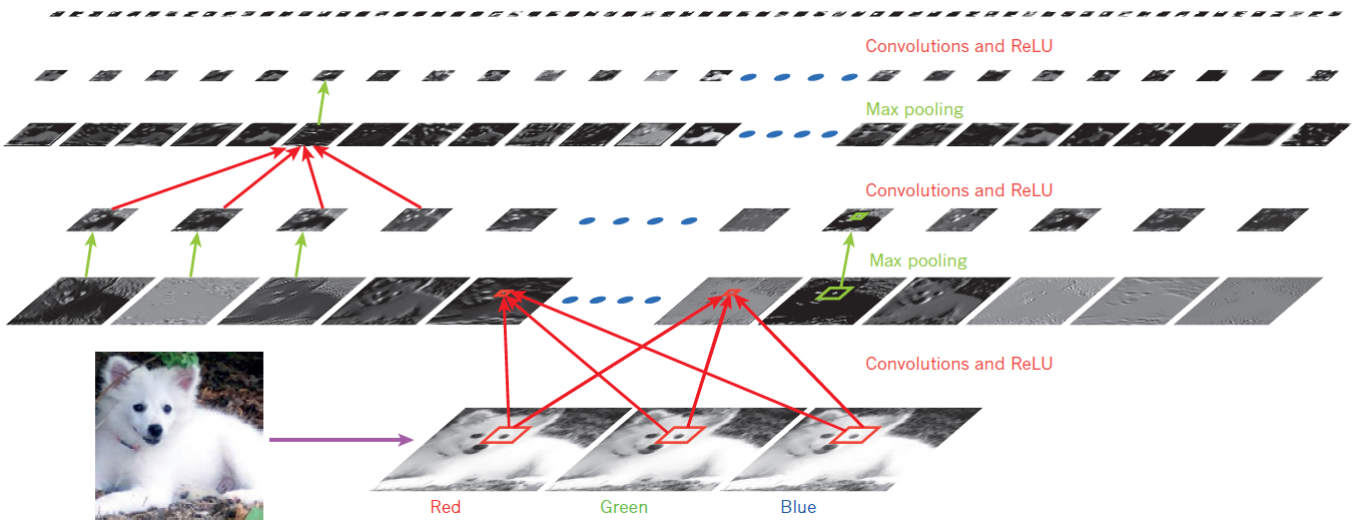



Figure 2.7: Example of a CNN taken from LeCun et al. (2015)

### 2.4.2 Recurrent Neural Network

Recurrent Neural Network (RNN) is a type of neural network where connections between nodes form a directed graph along a temporal sequence. It allows them to exhibit temporal dynamic behavior. Unlike feedforward neural networks, RNN can use its internal state, that is, memory, to process variable-length sequences of inputs (Lipton et al., 2015).

The main idea behind RNN is that the output from a given time step is fed back into the network as input for the next time step. This enables RNN to use their memory to process arbitrary sequences of inputs. In other words, RNN can employ the internal state to process inputs without a fixed length limit. The output of RNN is usually fed into a fully connected neural network. This type of network is illustrated in Figure 2.8.

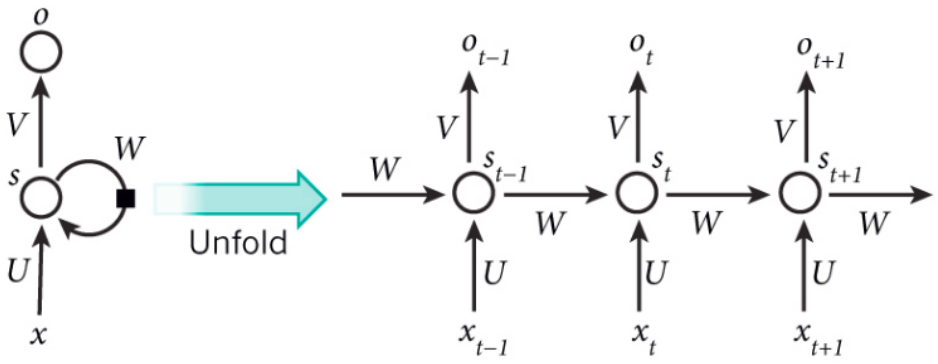



Figure 2.8: Unfolding in time of the RNN taken from LeCun et al. (2015)

More concisely, Figure 2.8 describes on its left side an RNN and on its right side the unfolding of such a network in time.

RNN weight matrices $(U; W; V)$ can be learned by unfolding the recurrence over time. This is known as Back Propagation Through Time (BPTT). The same weights are selected at each time step but $(x_t; h_t; y_t)$ evolves (Dupont, 2021). The network's equations are defined as follows :

$$\begin{cases} h_t = \tanh(Ux_t + Wh_t) \\ y_t = Vh_t \end{cases} \tag{2.7}$$

where $h_{t-1}$ represents the state at the previous step $(t-1)$, and is combined with $x_t$ to predict the current state $h_t$.

BPTT has a hard time converging for long-term dependencies present in long sequences. This behavior is known as exploding or vanishing gradients. The problem is due to continuous matrix multiplication from traversing the network. In basic mathematical algebra, a number greater than 1 with a high power will increase exponentially and a number lower than 1 with a high power will tend toward 0. Without going into further details, this illustrates the principle behind what causes a gradient to explode or to vanish on long sequences depending on the values of the weights. Activation functions will also play a role in the computation of the gradients.

### 2.4.3 Long short-term memory

Long short-term memory (LSTM) has been proposed to fix the issue of vanishing/exploding gradients. It utilizes purpose-built memory cells to store information. Those memory cells will improve the search and the use of the long-range context. Figure 2.9 illustrates a single LSTM memory cell.

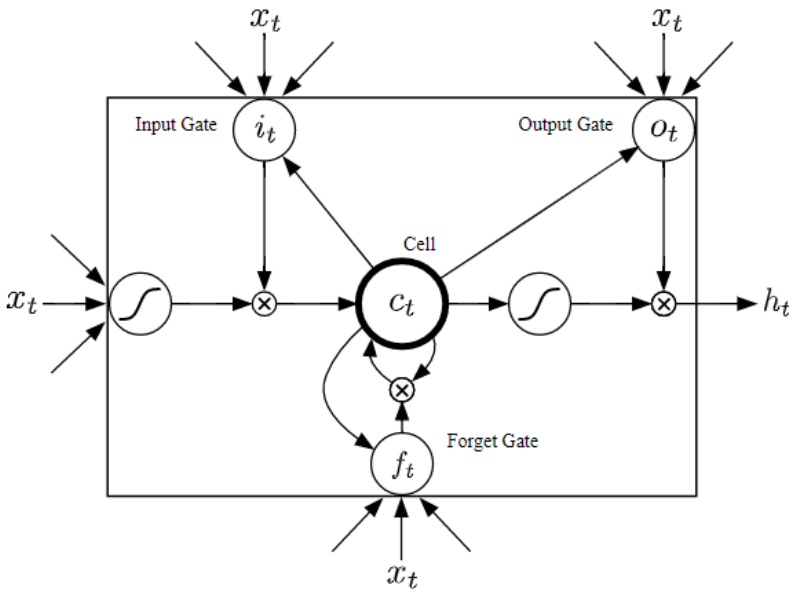



Figure 2.9: Overview of the LSTM taken from Graves et al. (2013)

The input gate $i_t$, output gate $o_t$, and forget gate $f_t$ all have an effect on the cell state:

$$\begin{cases} i_t = \sigma(W_{xi}x_t + W_{hi}h_{t-1} + W_{ci}c_{t-1}) \\ o_t = \sigma(W_{xo}x_t + W_{ho}h_{t-1} + W_{co}c_{t-1}) \\ f_t = \sigma(W_{xf}x_t + W_{hf}h_{t-1} + W_{cf}c_{t-1}) \\ c_t = f_t * c_{t-1} + i_t * \tanh(W_{xc}x_t + W_{hc}h_{t-1}) \\ h_t = o_t * \tanh(c_t) \end{cases} \tag{2.8}$$

For many sequence modeling tasks, it is advantageous to have access to both future and past contexts. Standard LSTM networks, on the other hand, process sequences in chronological order and ignore future context. Bidirectional Long Short-Term Memory Networks (BLSTM) networks extend unidirectional LSTM networks by adding a second layer in which hidden-to-hidden connections flow in the opposite temporal order. As a result, the model can use information from both the past and the future (Zhou et al., 2016).

### 2.4.4 Sequence to sequence models

As seen previously, neural networks can only be applied to problems whose inputs and targets can be encoded with vectors of fixed dimensionality. This represents a significant limitation because many important problems are best expressed with

sequences of lengths that are unknown in advance. Speech recognition and machine translation are such sequential problems. Sutskever et al. (2014) propose a new way to solve these types of problems by employing two LSTM working as an encoder and a decoder. More specifically, the idea is to read the input sequence one-time step at a time with one LSTM to obtain a large fixed-dimensional vector representation, and then use another LSTM to extract the output sequence from that vector. The second LSTM is similar to a Recurrent Neural Network language model but is conditioned on the input sequence. An example of this kind of network is represented in Figure 2.10. This model reads an input sentence "ABC" and produces "WXYZ" as the output sentence and stops after outputting the end-of-sentence token "<EOS>".

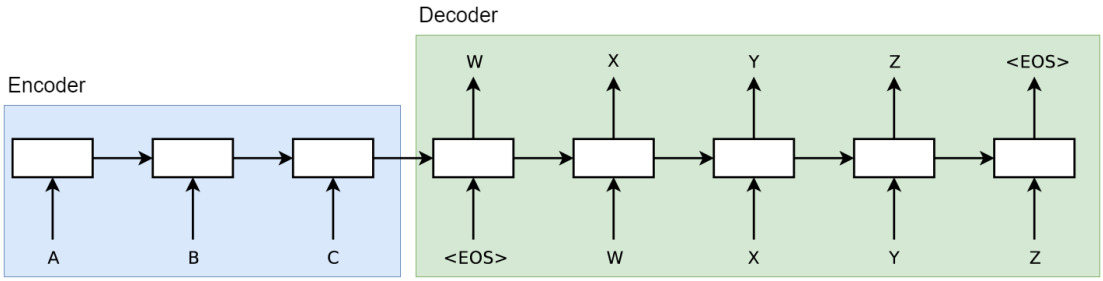



Figure 2.10: An example of a sequence to sequence model adapted from Sutskever et al. (2014)

However, the main issue with gradients remains. This type of architecture has the disadvantage of not being able to model long-term dependencies in the encoded vector.

### 2.4.5 Transformer based networks

Although LSTMs enable handling more distant information than RNNs, it does not completely solve the underlying problem: passing information through an extended series of recurrent connections that results in information loss and training difficulties. Furthermore, the inherently sequential nature of recurrent networks makes parallel computation difficult. These considerations resulted in the development of Transformers which is a new method of sequence processing that eliminates recurrent connections (Jurafsky and Martin, 2021).

An overview of the Transformer's architecture is illustrated in Figure 2.11.

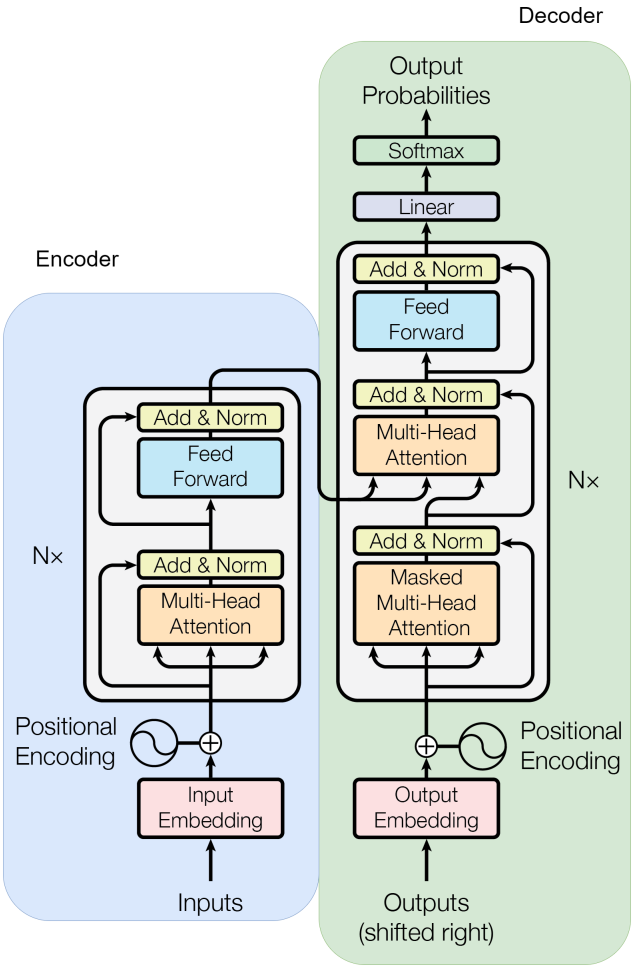



Figure 2.11: Transformer architecture adapted from Vaswani et al. (2017)

It is composed of an encoder and a decoder as a basic sequence-to-sequence model.

The encoder is first dependent on an input embedding layer. This embedding will allow semantically representing an input, i.e. a character, a word, or a sentence. There exist different ways to create these embeddings. An example of an embedding is word2vec, where words are converted to vectors enabling mathematical operations to be performed on them.

The Transformer architecture uses an attention mechanism that will allow one to pay attention to specific parts of the input that are considered important. An attention function is defined as mapping a query and a set of key-value pairs to an output, where the queries, keys, values, and output are all vectors. The output is computed as a weighted sum of the values, with the weight assigned to each value determined by a compatibility function of the query with the corresponding key

(Dupont, 2021; Vaswani et al., 2017).

Each input sequence gets a corresponding attention vector computed as follows.

$$\text{Attention}(Q, K, V) = \text{softmax}(\frac{QK^T}{\sqrt{d_k}})V \tag{2.9}$$

where $Q$, $K$, $V$ respectively Query, Key, and Value are computed using learnable weight matrices $W^Q$, $W^K$ and $W^V$, and $d_k$ is the dimension of the $K$ vector. After packing the input embedding into a single matrix $X$, the following equality is obtained:

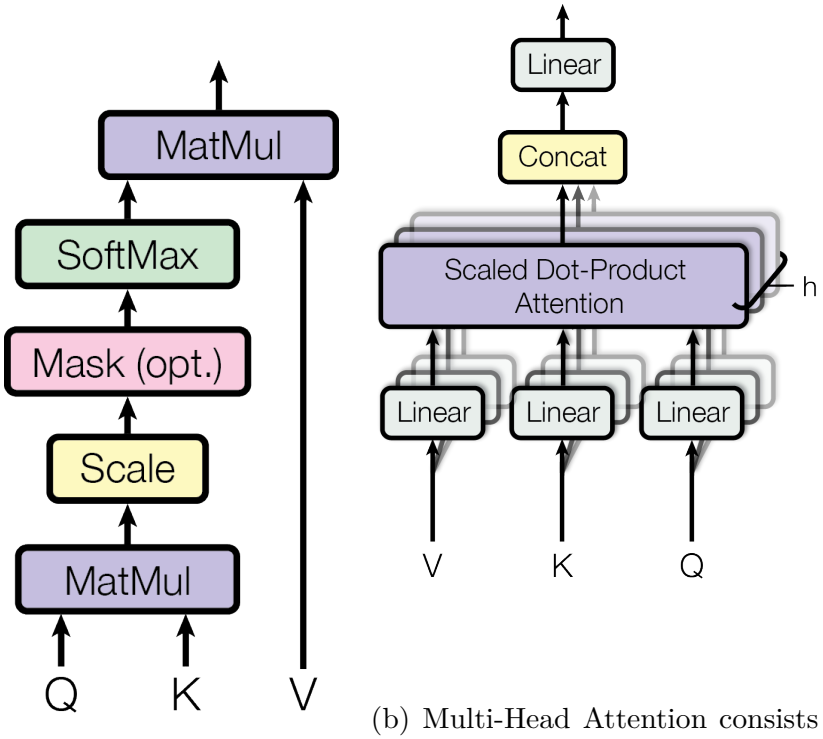
$$Q = W^Q X; K = W^K X; V = W^V X \tag{2.10}$$

This particular attention is called Scaled Dot-Product Attention and is represented in Figure 2.12a.

Moreover, Vaswani et al. (2017) advocates to use of multiple attention operations with several matrices $W^Q$, $W^K$, and $W^V$ repeated $h$ times calling it Multi-Head Attention. This type of attention is represented in Figure 2.12b. As different words in a sentence can relate to each other in different ways by having, for instance, syntactic, semantic, or discourse relationships, it will allow the model to jointly attend to the information from multiple representation subspaces at multiple positions.

The output of this layer is then fed to a forward layer, giving the final output of the encoder output.

The decoder, on the other hand, is composed of two different attention layers. It uses a Masked Multi-Head Attention layer. This masking, combined with the fact that the output embedding is offset by one position $i$, ensures that position predictions can only rely on known outputs at positions less than $i$. This layer is then followed by a Multi-Head Attention layer, which obtains its Query vector from the previous decoder layer while obtaining the Key and Value from the encoder output illustrated in Figure 2.11, followed by a feedforward layer similar to the encoder (Vaswani et al., 2017).




(a) Scaled Dot-Product Attention

(b) Multi-Head Attention consists of several attention layers running in parallel

Figure 2.12: Attention mechanism taken from Vaswani et al. (2017)

One essential concept that has been omitted so far is positional encoding. As the model contains no recurrence or convolution, the injection of some information about the relative or absolute position of the tokens in the sequence is required. It is done just after the input embedding in the encoder and the decoder. This encoding is given by the following formula :

$$\mathbf{PE}_{(pos,2i)} = \sin(pos/10000^{2i/d_{\text{model}}}) \tag{2.11}$$

where $pos$ is the position and $i$ is the dimension. That is, each dimension of the positional encoding corresponds to a sinusoid. The wavelengths form a geometric progression from $2\pi$ to $10000 * 2\pi$ (Vaswani et al., 2017).

## 2.5 Optical Character Recognition

The main characteristics of an image were illustrated previously. Mainly the diverse color systems of an image and how it is decomposed, i.e. by its pixels.

Now, let us take an interest in what an OCR is and what it does. This section contains some fundamental concepts of OCR required for a better understanding of the thesis.

### 2.5.1 Introduction

Optical Character Recognition (OCR) is the process of classifying optical patterns in a digital image that correspond to alphanumeric or other characters. OCR has gained in popularity in academic research, as well as in the industry in recent years. It has long been a man's ambition to create machines that can replicate himself. Human activities, such as reading documents that contain various types of text, are examples of human functions. In recent times, machines have become increasingly sophisticated. Giving reading capabilities to machines has progressed from a pipe dream to a reality thanks to the advancement of technology and reliable Optical Character Recognition (OCR) systems. There exist various techniques for automatic identification which cover the needs of different application areas. Some notable technologies and their applications worth mentioning apart from OCR are speech recognition, radio frequency, vision systems, magnetic stripe, bar code, magnetic ink, and optical mark reading. These technologies have been actively utilized in recent decades. However, since the main focus of the thesis is on expenses images, these types of technology will not be further discussed (Chaudhuri et al., 2017).

An OCR will be used to convert expenses' images containing textual content to text representations detected from the image. An example of this can be seen in the illustration 2.13. This is usually done by a two-phase process: text detection and recognition. Text detection tries to locate all text blocks in the image, while text recognition aims to understand and transcribe the text content and decode each character of the detected text.

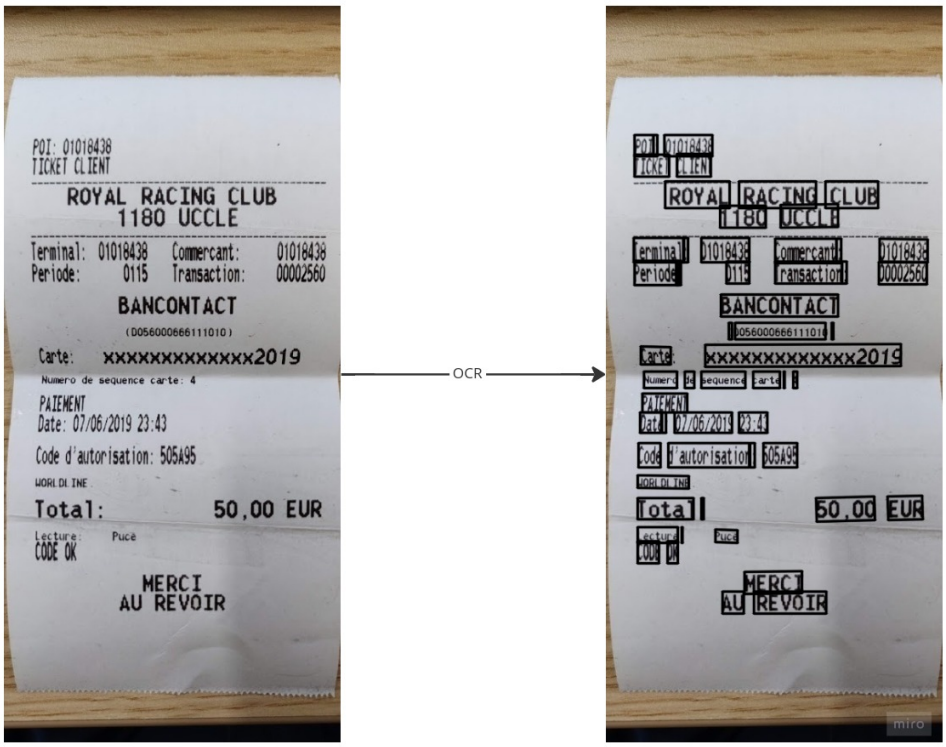



Figure 2.13: Application of an OCR process on an expense's image

Mostly, all text detection algorithms employ a U-net-type architecture. The U-net is a CNN developed at the University of Freiburg's Computer Science Department and initially put to use for biomedical image segmentation. The basic idea of this type of network is to add successive layers to a traditional contracting network, where pooling operations are replaced by upsampling operators. As a result, these layers improve the output resolution. Based on this information, a subsequent convolutional layer can learn to build a precise output (Ronneberger et al., 2015; Sicara, 2022).

On the other hand, the text recognition phase is usually performed using two main approaches, both of which utilize a CNN to pre-process the image and an RNN to decode the text. The first approach consists of using a CRNN (which is an RNN on top of a CNN) combined with a Connectionist Temporal Classification (CTC). The second approach uses a CNN with attention RNN.

### 2.5.2 Off-the-shelf solutions

In the majority of cases, it is better to choose an off-the-shelf OCR solution rather than to implement one from scratch. OCR is one of the most extensively researched

problems in Computer Vision and numerous ready-made solutions are available. Therefore, it was decided to go in that direction and leave OCR out of the scope of this thesis because it is a relatively distinct and standalone problem to solve.

Therefore, rather than explain the OCR architecture in detail, solutions that are already available will be discussed.

OCR support can generally be provided by two options: cloud-based solutions or programming language libraries. Cloud-based OCR engines provide an API that can be called in any programming language. These engines work by requesting an image that is sent through HTTP to an API endpoint. The recognized text blocks are then returned as a response. Google Vision OCR and Azure Cognitive Services Computer Vision are the most popular cloud APIs.

A programming language library is the second option. Tesseract is possibly the most widely used OCR library. It was developed historically by Google in 2006 and is now open source under the Apache license (Google, 2006).

The main distinction between the two options is that the cloud-based API provides the OCR engine via an HTTP API as said previously, whereas the programming language library is bundled into the program's source code. When deciding between the two, it's important to consider the specific needs and requirements of the application. Cloud-based APIs can be easily accessed from any programming language and may require minimal setup, but they may be slower and more costly compared to using a library. On the other hand, a library can be more challenging to maintain and update, and it may require additional dependencies based on the programming language.

## 2.6 Information extraction

In a large sense, automatic extraction of structured information from unstructured sources such as entities, relationships between entities, and their attributes is referred to as Information extraction (IE). This allows for much richer queries on the abundant unstructured sources rather than keyword searches alone. In general, the field of IE studies the extraction of structured data from unstructured documents. Extracting structured, machine-readable information from unstructured human communication is not an easy task. The path to glory, of course, would be if computers truly could understand unstructured human language but this essentially requires strong Artificial Intelligence (AI) capabilities (Sarawagi, 2008).

IE takes its roots in the Natural Language Processing (NLP) field. NLP is a field of study and application that investigates how computers can be employed to understand and manipulate natural language text or speech to accomplish useful tasks (Chowdhury, 2005). As most of IE is performed on natural language text, IE

frequently employs methods developed for Natural Language Processing (NLP).

Patterns, rules, and token classification are the two broad categories of approaches to IE.

Before exploring these approaches, let us focus on the main concept that these two share, namely entities.

Entities are typically noun phrases and comprise one to a few tokens in the unstructured text. Tokens in this context represent fragments of a text split. The most popular form of entity is "named entities", for example, names of persons, locations, and companies. However, the term named entity is commonly extended to include things that are not entities per se, including dates or times. In our case, the total, the date, and the currency will be tagged as named entities for an expense (Jurafsky and Martin, 2021).

### 2.6.1 Patterns and rules

Extraction tasks can be conveniently handled through a collection of rules which can be either hand-coded or learned from examples. Extraction by using rules-based methods is particularly useful when the task is controlled and well-behaved, e.g. the extraction of phone numbers or well-formatted dates. Usually, a rule can be represented as: regex(Token) $\rightarrow$ Entity. This means that on a token, a regular expression is fired and assigns the optimal label entity to it by detecting a pattern (Sarawagi, 2008).

Unfortunately, this type of proceeding has some limitations. Let us take an example to illustrate it. If someone wishes to extract the date "1-1-2022" and uses a regular expression in the form of "[0-9]{1,2}/[0-9]{1,2}/[0-9]{4}", there will be a match. But, it will not be the case for a date in the form of "January the 1st 2022".

Therefore, patterns and rules are easy yet very limited as IE methods. Especially in the case of expenses, where there are many varieties in formats.

### 2.6.2 Token classification

The other major approach to information extraction is token classification. This concept consists in using supervised Machine Learning to determine which tokens, usually words, represent which named entities. Nonetheless, to do so it is necessary to label each token.

### Named-entity recognition

The task of Named-entity recognition (NER) is described as finding and classifying the text segments into predefined classes that constitute an entity and labeling them as such. The standard approach to sequence labeling for a span-recognition problem as NER is Begin-Inside-Outside (BIO) tagging. It is a method that allows us to treat NER like a word-by-word sequence labeling task, via tags that capture both the boundary and the named entity type. To demonstrate BIO tagging, consider the following example: "Jane Foster lives in Chicago, Illinois.". The predefined classes are the person's name (PER) and location (LOC). After splitting the sentence into tokens at the word level, the tagged sentence is represented in Table 2.1 below (Jurafsky and Martin, 2021).

| **Words** | **BIO label** |
|---|---|
| Jane | B-PER |
| Foster | I-PER |
| lives | O |
| in | O |
| Chicago | B-LOC |
| Illinois | B-LOC |

Table 2.1: NER using BIO tagging

An "O" label indicates that the token is not assigned to an entity. A "B-" prefix indicates that the tagged token is the beginning of an entity. An "I-" prefix implies that the token belongs to the same entity as the previous token (prefixed either by "B-" or "I-").

From this example, it can be observed that the person's name is composed of two tokens but is assigned to one entity, while the location consists of two separate tokens each of which has been assigned to a distinct entity.

The NER approach will differ slightly if applied directly to expenses. Indeed, the extracted words from an OCR engine will not, in most cases, produce a coherent sequence of sentences. Additionally, Deep Learning Document AI models expect the input text to be in a two-dimensional format rather than a one-dimensional format as in most NLP tasks. These characteristics of expenses have to be taken into account when choosing the design of the model. Note that, with the exception of the encoding of layout information in the model's input, the task will not differ from a usual NER tagging task. The next chapter will focus on these models.

# Chapter 3

# Deep Learning Document AI

## Contents



Document AI, also known as Document Intelligence, refers to techniques for automatically reading, comprehending, and analyzing business documents. It is considered, nowadays, a critical research area in NLP and Computer Vision. Due to the diversity of layouts and formats, low-quality scanned document images, and the complexity of template structures, Document AI is a very challenging task and has attracted worldwide attention. The popularity of Deep Learning technologies in recent years has greatly advanced the development of Document AI, mainly document visual question answering, document image classification, document layout analysis, and document parsing (Cui et al., 2021).

This thesis will focus on the last two domains mentioned above which are document layout analysis and document parsing.

The task of determining the physical structure of a document, i.e., identifying the individual building blocks that make up a document, such as text segments, headers, and tables, is known as document layout analysis. This task is frequently approached as an image segmentation/object detection problem. The model produces segmentation masks/bounding boxes, as well as class names (Shah et al., 2022).

Document parsing or Key Information Extraction constitutes a step beyond layout analysis. It describes techniques for extracting entities and their relationships from large amounts of unstructured content in a document. In contrast to traditional approaches as NER, the document's construction transforms the text from a one-dimensional sequential arrangement to a two-dimensional spatial arrangement. As a result, text, visual, and layout features play a significant role in key information extraction. (Cui et al., 2021).

The next sections will describe two state-of-the-art models used in the document parsing domain.

## 3.1 LayoutLM: Pre-training of Text and Layout for Document Image Understanding

LayoutLM is a family of multimodal models based on Transformers. Heavily influenced by the BERT Transformer, this family is divided into multiple versions: LayoutLM, LayoutLMv2, and LayoutLMv3. LayoutLMv3, the latest successor of the LayoutLM type architecture, will be the main focus of this section as it is the most advanced version of LayoutLM. These models are intended to be used on English data. There exists also a model trained with multilingual data, LayoutXLM, but the core concept remains very similar between all these versions.

It is important to go into more detail on how the BERT Transformer works, as it is the backbone of the LayoutLM-type architecture.

### 3.1.1 BERT

Bidirectional Encoder Representations from Transformers (BERT) was published by Google in 2019 and built on the original Transformer architecture. As the name suggests, the goal was to train a language model to generate meaningful text sequences and token-level embeddings that can be used in a variety of downstream

tasks. Pre-trained BERT models were applied to many standard NLP tasks, resulting in state-of-the-art performances.

### 3.1.2 Pre-training on BERT

One of the main improvements of BERT over vanilla Transformers is its two-phase training strategy: pre-training and fine-tuning. The pre-training phase builds, in a self-supervised way, language understanding on a large dataset of unlabeled text with carefully selected pre-training objectives using Masked Language Model (MLM) which will be seen in further details later and Next Sentence Prediction (NSP) tasks. NSP task is employed to predict whether two sentences precede each other or not. The pre-trained BERT model is then refined on task-specific and much smaller datasets to extend the model, depending on the downstream task, thereby, saving time and requiring less labeled data (Devlin et al., 2018). Figure 3.1 shows an example of this process.

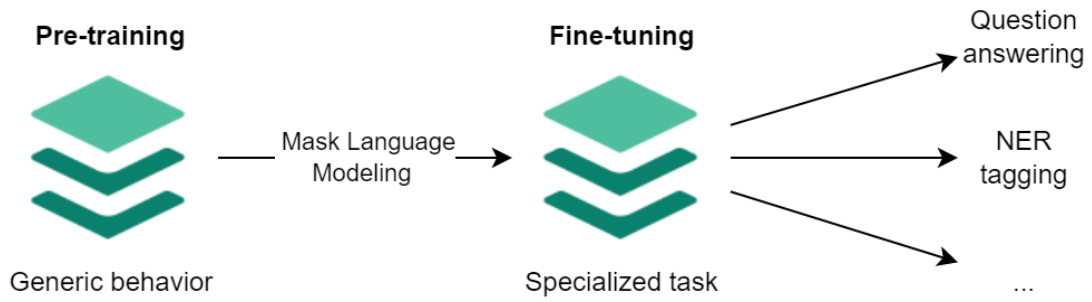



Figure 3.1: An example of BERT pre-training

### 3.1.3 Extending BERT

The LayoutLMv3 extends the BERT concepts just as BERT extends the original Transformer architecture. In particular, LayoutLMv3 extends BERT in three important ways, as mentioned by Huang et al. (2022):

**Layout Embedding** The BERT architecture had the concept of one-dimensional position embedding. LayoutLMv3, on the other hand, works with two-dimensional data. Therefore, it models spatial relationships between the text tokens. LayoutLMv3 extends the one-dimensional position embedding with a new two-dimensional layout embedding. This embedding refers to the bounding box coordinates of the text sequence, which is usually extracted with an off-the-shelf OCR toolkit.

**Pre-Training Objectives** LayoutLMv3 uses more appropriate objectives by extending the concepts of BERT's pre-training objectives.

The full pre-training objectives of LayoutLMv3 are defined as : $L = L_{MLM} + L_{MIM} + L_{WPA}$

- *Masked Language Model (MLM) :*

  The training goal for MLM is straightforward. The model is asked to predict a random subset of input tokens that have been masked. As a result, the model picks up on the bidirectional context around the masked words and learns to anticipate words (or tokens).

  The pre-training objective is to maximize the log-likelihood of the correct masked text tokens $y_l$ based on the contextual representations of corrupted sequences of image tokens $X^{M'}$ and text tokens $Y^{L'}$, where $M'$ and $L'$ represent the masked positions. The Transformer model's parameters are denoted with $\theta$ and minimize the subsequent cross-entropy loss:

$$L_{MLM}(\theta) = -\Sigma_{l=1}^{L'} log\ p_\theta(y_l \mid X^{M'}, Y^{L'}) \tag{3.1}$$

- *Masked Image Modeling (MIM) :*

  The block-wise masking approach is employed to randomly mask a portion of around 40% of the image tokens as part of the MIM objective, which is symmetric to the MLM objective. A cross-entropy loss serves as the motivation for the MIM objective, which is to rebuild the masked image tokens $x_m$ in the context of the surrounding text and image tokens.

$$L_{MIM}(\theta) = -\Sigma_{m=1}^{M'} log\ p_\theta(x_m \mid X^{M'}, Y^{L'}) \tag{3.2}$$

- *Word-patch alignment (WPA)*

  Each text word in a document relates to an image patch. This image patching will be explained in more detail later. At this stage, there is no explicit alignment learning between the text and image modalities because MIM and MLM are employed to randomly mask text and image tokens, respectively. To learn a precise alignment between text words and image patches, Word-Patch Alignment is needed. The goal of WPA is to determine whether a corresponding image patch of a text word is masked.

  For this purpose, a two-layer Multi Layer Perceptron (MLP) head is used that inputs contextual text and images, and outputs binary aligned/unaligned labels with a binary cross-entropy loss.

$$L_{WPA}(\theta) = -\Sigma_{l=1}^{L-L'} log\ p_\theta(z_l \mid X^{M'}, Y^{L'}) \tag{3.3}$$

where $L - L'$ is the amount of unmasked text tokens and $z_l$ is the binary label of the language token at position $l$.

**Visual embedding** Visual embeddings provide more relevant information to the input when working in a visual context. LayoutLMv3 extends the BERT architecture by supporting Word-patch alignment (WPA), as stated above.

### 3.1.4 ViT

LayoutLMv3 is also inspired by the Vision Transformer (ViT) and utilizes raw image patches from document images without any complex pre-processing steps such as page object detection. LayoutLMv3 learns images, text, and multimodal representations collaboratively in a Transformer model with unified MLM, MIM, and WPA objectives. This makes LayoutLMv3 the first multimodal pre-trained Document AI model that does not require CNNs for image embeddings, saving parameters, and eliminating the need for region annotations (Huang et al., 2022). Note that, for the other LayoutLM variants, a CNN is still implemented instead of a Transformer.

As seen in Figure 3.2, the ViT works by dividing an image into fixed-size patches, each of which is linearly embedded, adding position embeddings, and feeding the resulting vector sequence to a standard Transformer encoder. For instance, to perform a classification, the standard method of inserting an additional learnable "classification token" into the sequence is employed. Vaswani et al. (2017) was the inspiration for the illustration of the Transformer encoder, as stated by Dosovitskiy et al. (2020).

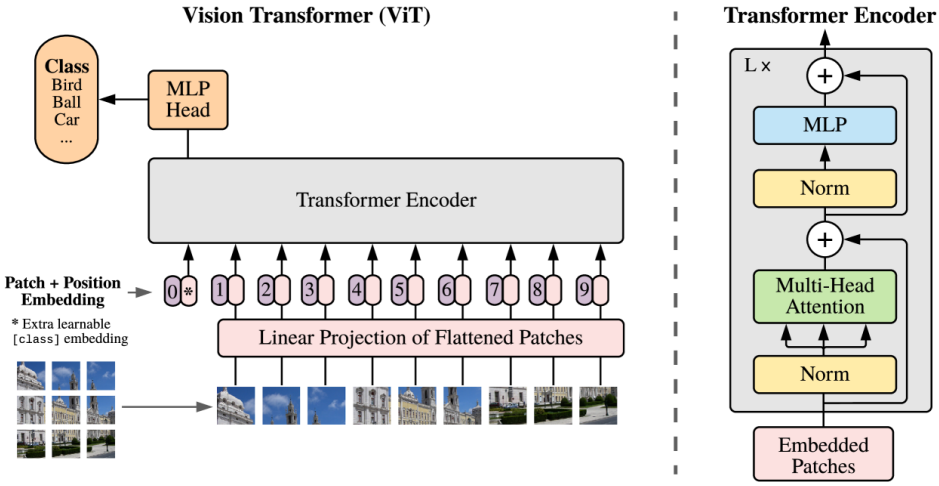



Figure 3.2: Overview of the ViT's architecture taken from Dosovitskiy et al. (2020)

### 3.1.5 Architecture

Let us now focus on LayoutLMv3's architecture more in-depth.

Firstly, it is important to determine the unique aspects of the architecture compared to existing works. This comparison can be divided into two features, as shown in Figure 3.3:

**Image embedding** LayoutLMv3 employs linear patches to alleviate the computational bottleneck of CNNs and eliminate the need for region supervision in object detector training.

**Retraining objectives on image modality** To capture high-level layout structures rather than noisy details, LayoutLMv3 learns to reconstruct discrete image tokens of masked patches rather than raw pixels or region features (Huang et al., 2022).

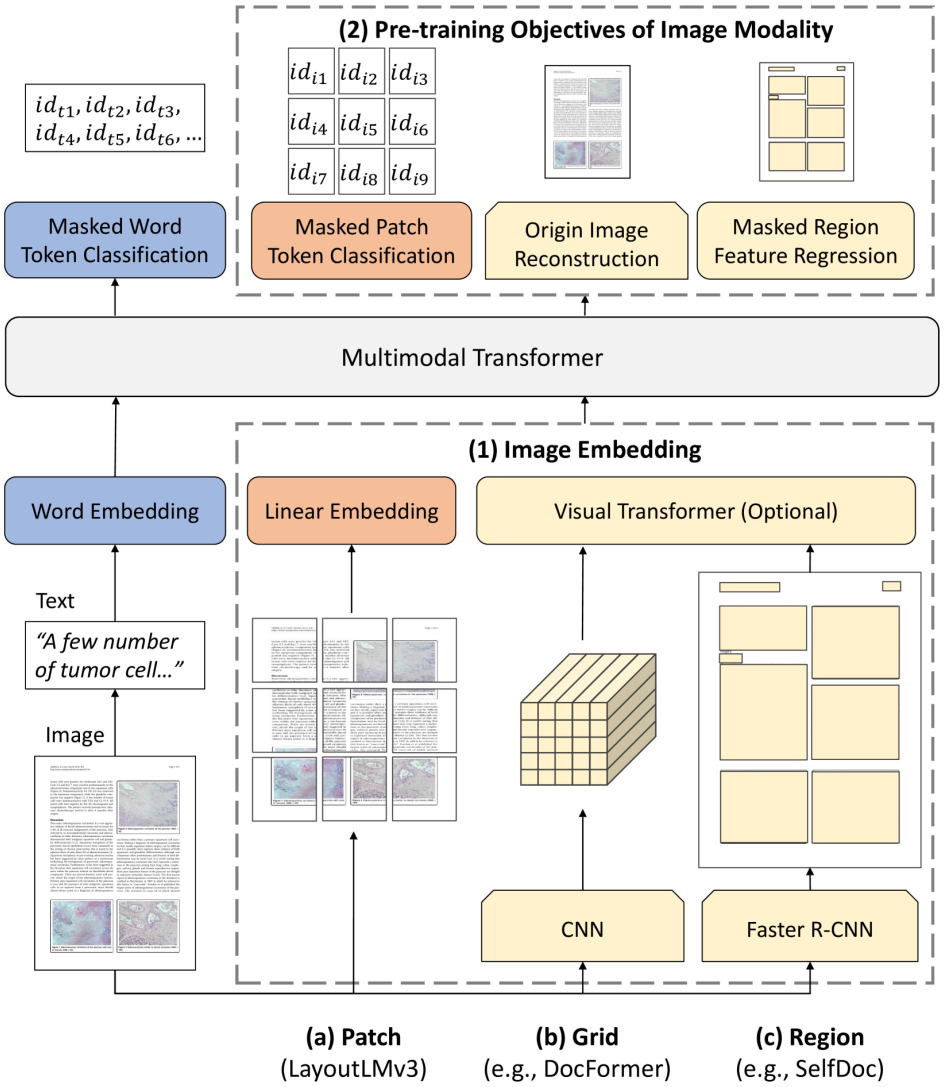



Figure 3.3: Comparisons of LayoutLM from existing works taken from Huang et al. (2022)

The high-level architecture of LayoutLMv3 is represented in Figure 3.4. Given an input document image, its corresponding text embedding and layout position information, the model takes the linear projection of image patches and word tokens and encodes them into contextualized vector representations. As mentioned previously, LayoutLMv3 is pre-trained with discrete token reconstructive MLM and MIM objectives. LayoutLMv3 is also pre-trained with a WPA objective to learn cross-modal alignment by predicting whether a text word's corresponding image patch is masked. "Seg" stands for segment-level positions. Special tokens

include "[CLS]," "[MASK]," "[SEP]," and "[SPE]" (Huang et al., 2022).

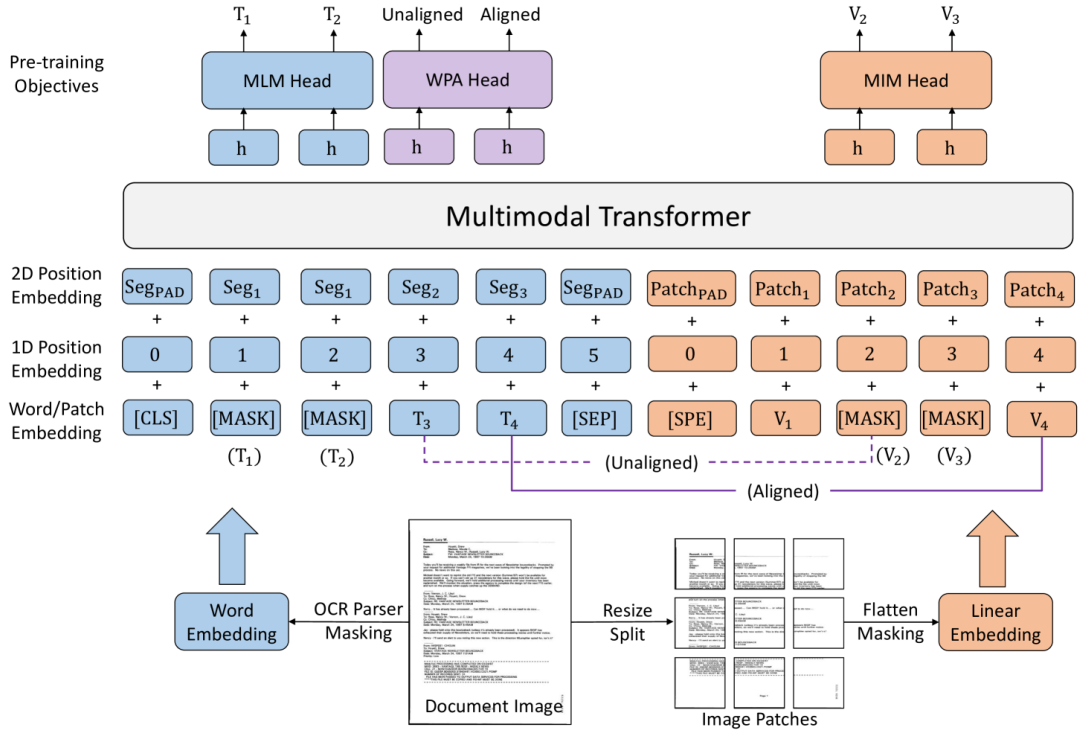



Figure 3.4: High-level LayoutLM's architecture taken from Huang et al. (2022)

## 3.2 LiLT: Language-Independent Layout Transformer

The majority of related models for Document AI tasks, including LayoutLM, only handle document data from the pre-training collection in one or more specific languages, usually English. Therefore, another model Language-independent Layout Transformer (LiLT) for structured document interpretation was proposed to solve this problem. With the appropriate pre-trained monolingual/multilingual textual models that are readily available, LiLT can be pre-trained on the structured texts of a single language before being immediately fine-tuned on other languages (Wang et al., 2022).

### 3.2.1 Architecture

An overview of the LiLT's architecture, which is similar in most parts to LayoutLM, can be observed in Figure 3.5 and is briefly explained below.

**Text embedding** All text strings from the OCR results are first tokenized and concatenated as a sequence.

**Layout embedding** In terms of layout flow, LiLT creates a 2D position sequence of the same length as the token sequence by using text bounding boxes.

**Visual embedding** Since there is no CNN decoder and no linear projection features of image patches, there is no image embedding.

### BiACM

To generate high-level enhanced features, text embedding and layout embedding are fed into their respective submodels. If the text and layout features at the encoder output were simply combined, the cross-modal interaction process would largely be ignored. The network must also thoroughly analyze them at an earlier stage. Therefore, a new Bidirectional Attention Complementing Mechanism (BiACM) is proposed to improve cross-modality interaction throughout the encoding pipeline.

### Pre-Training Objectives

The pre-training objectives are divided into three parts to guide the model to autonomously learn joint representations with cross-modal cooperation.

- Masked Visual-Language Modeling (MVLM): Similarly to BERT, MVLM masks some of the input tokens at random and asks the model to recover them across the entire vocabulary using the output encoded features, guided by a cross-entropy loss.
- Key Point Location (KPL): This task has been proposed to help the model better understand the layout information in structured documents. KPL divides the entire layout into several regions and masks some of the input bounding boxes at random. Using separate heads, the model must predict which regions the key points (top-left corner, bottom-right corner, and center point) of each bounding box belong to.
- Cross-modal Alignment Identification (CAI): The encoded features of the token box pairs that are masked are collected and then replaced (misaligned) or left unchanged (aligned) by MVLM and KPL. They are used to build an additional head to determine whether each pair is aligned. To accomplish this, the model must learn cross-modal perception capability. CAI is a binary classification task that employs a cross-entropy loss.

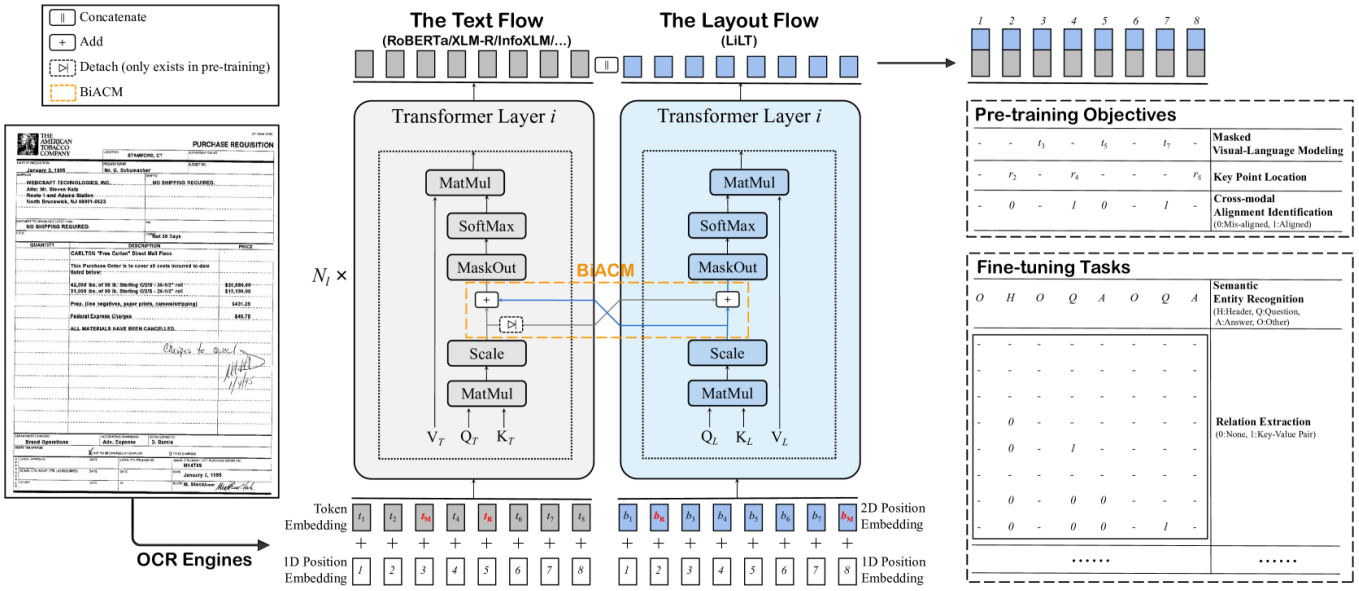



Figure 3.5: Overview of LiLT's architecture taken from Wang et al. (2022)

## 3.3 Models' performance on form and receipt understanding task

Form and receipt comprehension tasks necessitate the extraction and structuring of textual content from the documents. These tasks are a sequence-labeling problem in which each word is tagged with a predefined label. Considering that these types of tasks are similar to the expense annotation task, the best-performing models will be carried out to assess the performance of the Odoo expense dataset.

Experiments collected from Huang et al. (2022) and Wang et al. (2022) are conducted on the Form Understanding in Noisy Scanned Documents (FUNSD) and Consolidated Receipt Dataset for Post-OCR Parsing (CORD) datasets.

The FUNSD is a noisy scanned form understanding dataset sampled from the RVL-CDIP. The FUNSD dataset includes 199 documents that have detailed annotations for 9,707 semantic entities. The semantic entity labeling task on the FUNSD dataset refers to assigning a label to each semantic entity from "question", "answer", "header", or "other". On the other hand, CORD is a receipt key information extraction dataset with 30 semantic labels defined under 4 categories such as the total or subtotal. It contains 1,000 receipts of 800 training, 100 validation, and 100 test examples (Huang et al., 2022).

### 3.3.1 Comparison of different models

Having defined the datasets employed for performance assessment, it is time to select different models to conduct the tests. The task considered will be a classification task of NER-tagging. As a baseline model, the BERT model was used. Then, different variants of the architecture of the previously presented models were tested on the CORD and FUNSD datasets.

#### F1 score

As previously introduced, there are several metrics worth considering when dealing with a classification problem. When evaluating the performance of a classification model, the F1 score is a metric that is often considered. This metric combines precision and recall, with a higher score indicating better performance. As mentioned before, precision is the proportion of correct positive predictions, while recall is the proportion of actual positive examples that were predicted correctly. The F1 score is calculated by taking the harmonic mean of precision and recall, which puts more emphasis on lower values.

#### Performances comparisons

The results are represented in Figure 3.1. The retrieved results are for the "BASE" version of the models for simplicity, which refers to the model version with a restrained number of parameters. The F1 scores are sorted in ascending order and are in percent. All LayoutLM variants that were presented were considered. In addition, two LiLT variants were tested. Firstly, the LiLT-RoBERTa (standing for Robustly Optimized BERT Pre-training Approach) which uses an existing pre-trained RoBERTa textual model on an English base. Then, the LiLT-InfoXLM which, on the other hand, uses an off-the-shelf pre-trained InfoXLM textual model on multilingual data.

| Model | F1 on CORD ↓ | F1 on FUNSD ↓ |
|---|---|---|
| BERT | 89.68 | 60.26 |
| LayoutLM | 94.72 | 78.66 |
| LayoutXLM | 94.81 | 80.34 |
| LayoutLMv2 | 94.95 | 82.76 |
| LiLT-InfoXLM | 95.77 | 85.86 |
| LiLT-RoBERTa | 96.07 | 88.41 |
| LayoutLMv3 | **96.56** | **90.29** |

Table 3.1: Models' performance comparison taken from Huang et al. (2022); Wang et al. (2022) conducted on the CORD and FUNSD datasets

It can be observed that the LayoutLMv3 model stands out and is the best-performing LayoutLM variant model on the CORD and FUNSD datasets. The second-best performing model is the LiLT-RoBERTa and is only $\sim 0.5$ % away from LayoutLMv3 on the CORD dataset and $\sim 2$ % on the FUNSD dataset.

From the performances shown, it appears that using multimodal inputs with pre-trained Transformer-based models is quite effective on document understanding tasks compared to the baseline model BERT.

### 3.3.2 Models considered for expenses' information extraction

Based on the results shown in Table 3.1, Transformers for document understanding tasks prove to be an effective strategy. Therefore, the two models that performed the best were selected to be tested on the Odoo expense dataset for performance assessment, that is, the **LayoutLMv3** model and **LiLT with RoBERTa** pre-training. Additionally, two other models were chosen, namely the multilingual variation of the previously cited models: the **LayoutXLM** and **LiLT-InfoXLM**. As the expense dataset contains multi-lingual expenses those models might perform as well as the best ones.

Now that the two main models have been studied and the best instances have been considered, the methodology that will be used to fine-tune and evaluate the different models will be described. Among the different models considered, only one model will be chosen to be utilized in production by the Odoo IAP team. In the next chapters, the term LayoutLM will refer to the LayoutLM variants considered (i.e. LayoutLMv3 and LayoutXML), unless stated otherwise. The same applies for LiLT.

# Chapter 4

# Methodology

## Contents

After having introduced the background knowledge needed along with the selection of the Document AI's models, it is time to go on practical aspects. Therefore, this chapter is dedicated to explaining the methodology used in our thesis. It is made up of three main parts. Firstly, the data handling will be explained. This part represents the overall data workflow. How the data, from the expense to the annotation stage, is handled to get the best performance out of our model. Then, the image-processing techniques that were applied to the images before feeding the model will be discussed. Finally, the overall experimental setup will be exposed.

## 4.1 Machine Learning development

The goal of this section will be to explain the contrast between the well-established model-centric approach employed in the ML research and the newly arising approach called data-centric.

### 4.1.1 Model-centric approach

As mentioned in the last chapters, AI systems are created through the use of code (which implements a learning algorithm) and data (which are used to train the system). For many years, the standard AI approach was to download a dataset and work on the code. This type of approach is called model-centric. It entails conducting experimental research to improve the performance of the ML model. This requires selecting the best model architecture and training process from a variety of options. The primary objective of this approach is to work on code while keeping the data the same, as seen in the illustration 4.1 (Ng, 2021).

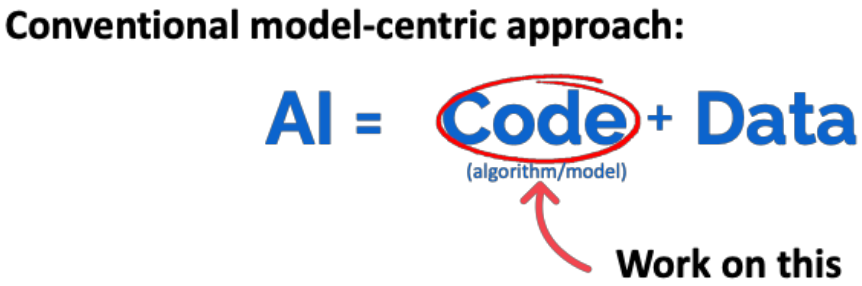



Figure 4.1: The model-centric approach taken from Ng (2022)

According to Ng (2021), the majority of AI applications are currently model-centric. He suspects that one possible reason for this is that the AI industry pays close attention to academic research on models (more than 90% of research papers in this domain are model-centric). It is due to the difficulty of creating large datasets that can become widely accepted standards. As a result, the AI community considers model-centric Machine Learning to be more promising. In that manner, data are often overlooked while focusing on the code, and data collection is viewed as a one-time event.

### 4.1.2 Data-centric approach

In contrast to the model-centric approach, the data-centric approach entails systematically changing/improving datasets to increase the accuracy of Machine Learning applications. The primary goal of this approach is to work with the data as seen in the illustration 4.2.

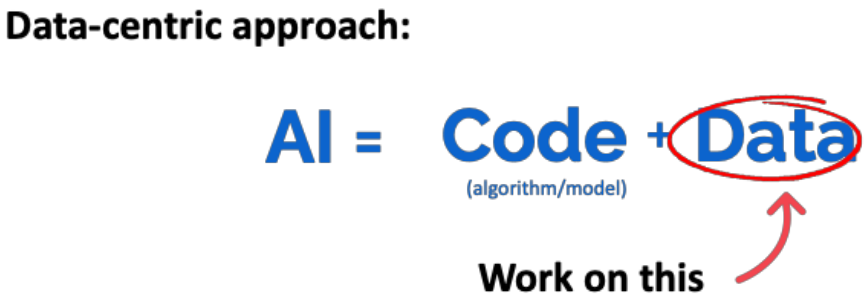



Figure 4.2: The data-centric approach taken from Ng (2022)

The model-centric methodology works well with clean annotated data. However, in practice, most of the acquired data are noisy. Noisy data can be caused by

hardware, wrong inputs, human errors, etc. For example, a common issue lies in the complexity of building a well-annotated dataset. Two people might have different views when annotating a specific data object. This non-negligible difference will impact the model during the training phase. To counteract this, research of new models and fine-tuning them was the right solution (Ng, 2021).

A new emerging idea to tackle this problem, instead of continuously improving the model, is a data-centric approach to building AIs. This solution is mainly supported by Ng (2021), a globally recognized AI scientist. Rather than extensively working on the model, the focus is on the data. The goal is to remove all unnecessary noise and feed clean data to the model. This approach offers better performances and saves training time. Additionally, the model can be reused for new tasks because it is not as dependent on the dataset as in a model-centric approach.

## 4.2 Related tools

This section is dedicated to the presentation of the main tools used in the experimental process.

### 4.2.1 OpenCV

OpenCV is a well-known Apache 2 licensed open source computer vision library written in C++ and accessed through Python. It has several distinct image-handling data structures and routines that can roughly carry out any computer vision process on images. The library is extensively put to use in this thesis to manage the photos and run pre-processing operations on the images such as blur, binarization, etc. (OpenCV, 2022b). This library was selected as it is popular, well documented and highly optimized for real-time image processing.

### 4.2.2 Google Cloud Vision

Google Vision OCR was chosen as the off-the-shelf OCR as it is utilized by Odoo to extract textual information from expenses.

### 4.2.3 Hugging Face

Hugging Face is a well-known community and data science platform that provides tools for users to build, train, and deploy ML models based on open source code and technologies (Hugging Face, 2022). Its libraries, especially the transformers library, are employed in this thesis, as it provides off-the-shelf pre-trained transformers such as LayoutLM and LiLT implementations.

### 4.2.4 Weights & Biases

Weights & Biases is a Machine Learning online platform. This platform is primarily utilized to track and record experiments and model performances. The use of Weight & Biases was motivated by its native integration with Hugging Face.

## 4.3 Data-centric workflow

The following section outlines the data-centric workflow used to improve the performance of the model.

### 4.3.1 Prior data cleaning

The initial expense dataset obtained from the Odoo backend contained 2264 expenses. From this dataset, a subset of expenses was selected for further analysis. It was filtered using the following process.

- All PDF files were discarded and only pictures were retained.
- Pictures that did not represent expenses were also removed.

### 4.3.2 Overall workflow

Once only photos of expenses remain, their annotation from the OCR was added to the dataset. Given this dataset, the new objective is to get high overall performances on our model. The workflow used to achieve it will be described in this section and is illustrated in Figure 4.3 below.

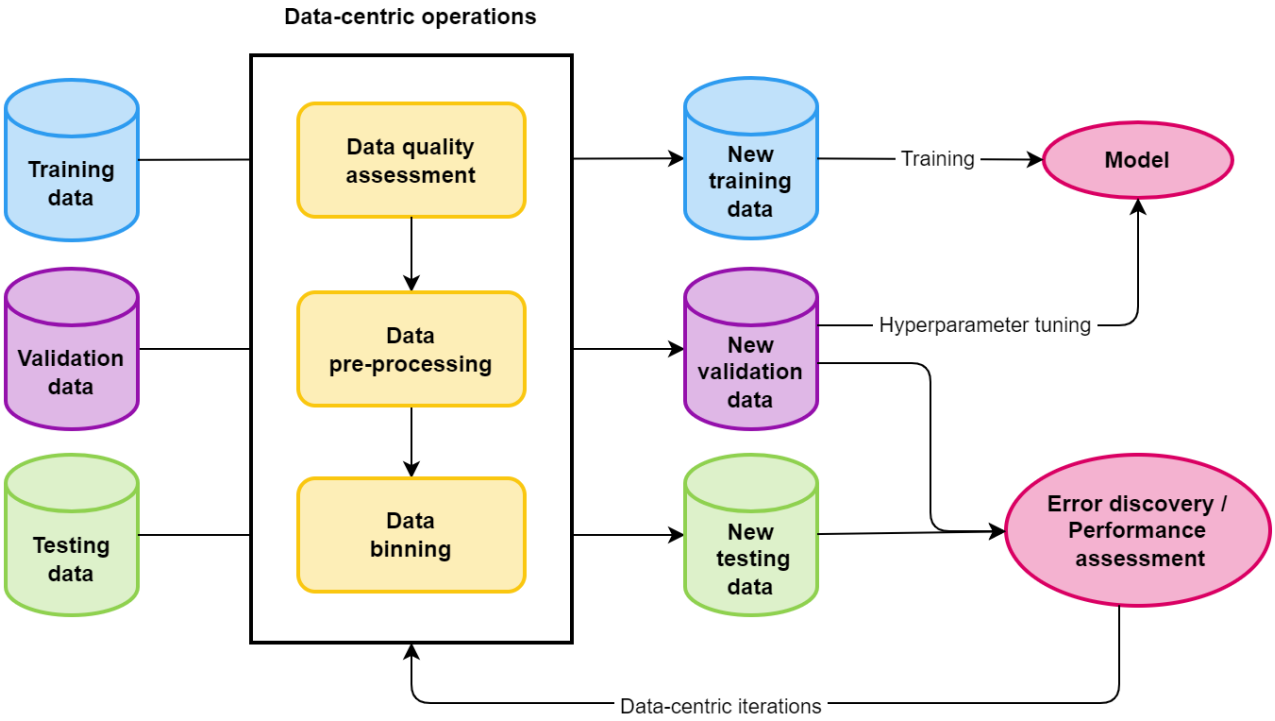



Figure 4.3: The data-centric workflow inspired from Mazumder et al. (2022)

The goal of the data-centric workflow is to build high-quality datasets to feed the model. Starting on the left side of Figure 4.3, the dataset is first split into 3 sets, training, validation, and testing. These sets and their role have been explained previously. The technique used to split the dataset will be covered in more detail in the following chapter.

After obtaining these sets, various data-centric operations are carried out on them to generate new, processed sets. The first operation in the data-centric pipeline is quality assessment. This operation will be responsible for fixing the major mistakes that were found such as the lack of total in the OCR parsing. All those images will be discarded. Then, multiple pre-processing techniques are applied to the data. This pre-processing step allows us to test several combinations of image processing techniques and improve the overall performance. The entire image processing pipeline, including the selection of specific steps, will be discussed in the following section. Finally, the dataset is divided into different bins. Binning is the process of splitting data into several distinct bins or ranges. This can be useful for a variety of purposes, including visualization (creating a histogram) and grouping together comparable data. At first glance, binning might seem insignificant, but it allows us to gain better insights into problematic pictures. The different types of

bins being considered will be explained in more detail later.

Once processed, the training set is used to train the model, and the validation set is used to fine-tune the model. Both the validation set and the testing set enable error discovery and performance assessment.

Finally, during the data-centric iterations, the model's overall and specific performance will be assessed by the bins. Each iteration of the image processing pipeline will attempt various combinations of processing approaches to enhance these performances. On the other hand, data binning will include removing bins from the training and validation sets throughout iterations to try to improve the performance by lowering noise. These procedures take a long time to complete. The best-performing model will ultimately be chosen after these procedures are iteratively performed to maximize the performance of the model. New training sets and validation sets will be generated at each iteration unlike the test set, which will remain unchanged, except for the application of the image processing techniques.

## 4.4 Image processing techniques

This section will explain the pre-processing techniques taken into consideration for images and their implementation, which are crucial aspects of this thesis.

### 4.4.1 Rotating

The rotation technique's objective is to rotate every image such that it is displayed in portrait mode. This is done to ensure that the images are orientated consistently and to prevent coping with various layout angles. The strategy implemented is based on Odoo's existing algorithm, which will establish the rotational angle required to straighten the bounding box retrieved from the OCR. This is shown in Figure 4.4.

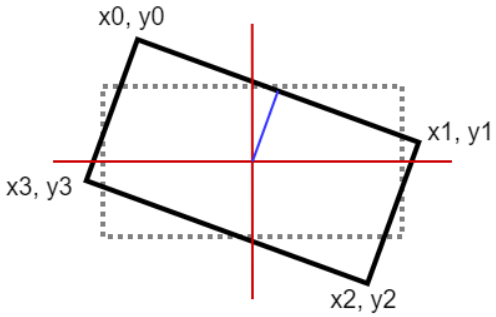



Figure 4.4: Rotation of a bounding box

To this extent, the arctangent is determined by using the following formula: $\arctan(y0 - y1, x1 - x0)$. The bounding box's coordinates ($x0, y0$, and $x1, y1$) are used as inputs into the formula, and the output is the angle of rotation required to straighten the bounding box.

After taking the angle of rotation of each bounding box, the median angle of rotation for the image is determined. That way the rotation that results is accurate and takes into account any differences in the bounding boxes' angles. The image and all of its bounding boxes are then rotated by an angle that is opposite to the main angle of rotation. Thereby, the image and all of its bounding boxes are essentially rotated to be displayed in portrait mode.

### 4.4.2 Cropping

Cropping is the process of removing unwanted areas of an image. Cropping the image allows us the remove the unused background information as excess noise may be harmful to the model's performance.

The first approach for cropping images was to perform image segmentation with Hough lines which are usually used in detecting straight lines in images. The problem with this approach is that it is not perfect. As shown in Figure 4.5b, folds will also be taken into account and the lines are not always drawn if the contrast between the expense and the background is not high enough. Additionally, such cropping requires more processing time.

A better solution is to get the lowest and highest x and y from all the bounding box coordinates. In this way, it becomes possible to get a contour box that includes all other bounding boxes. An example of such contouring is shown in Figure 4.5c. All that is left to do is to crop the image to the size of the contour box and translate the bounding boxes accordingly. The only drawback of this solution is that the background may also contain textual information. For such instances, the lowest and highest coordinates will be outside of the expense area.


(a) Original (b) Hough lines (c) Contour box

Figure 4.5: Expense localization techniques

### 4.4.3 Blurring

It may seem counter-intuitive to blur an image because it degrades the image's quality. However, blurring can make other pre-processing methods function better. It is due to the fact that blurring smooths the image by eliminating superfluous elements. This simplification can make it easier for algorithms to analyze and understand the content of the image.

### 4.4.4 Binarizing

Binarization is a technique to convert a grayscale image into a black-and-white image. Its purpose is to eliminate noise and reduce the computational load. To do so, the image must first be put into a grayscale format by the OpenCV library.

### 4.4.5 Thinning

The goal is to achieve the skeletonization of the text. The idea is to reduce the text width to a very small width to get the text skeleton. This step allows for the standardization of all textual information.

### 4.4.6 Morphological operations

The most common morphological operations are dilation and erosion. It is performed on binarized images. Dilation increases the white region in the image or the size of the foreground object, and erosion erodes the boundaries of the foreground object

(OpenCV, 2022a). Erosion might remove noise from the image. For instance, small black dots on the expense would be eroded away. On the other hand, dilation could improve the shape of text expense by adding pixels to it.

### 4.4.7 Normalization

Normalization is another important step to ensure layout standardization. It is mandatory for both LayoutLM and LiLT. The bounding box coordinates are normalized between a range of 0 and 1000 given the formulas: $(1000 * (x/\text{image\_width}))$ and $(1000 * (y/\text{image\_height}))$ for the x and y coordinates respectively.

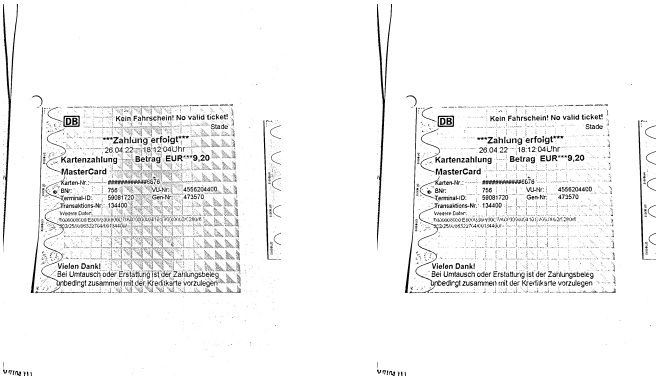
### 4.4.8 Resizing

Again to ensure standardization, resizing is a well-known step to put all images to the same dimensions and reduce training time.

## 4.5 Pre-processing pipeline

Thus far, the workflow and the pre-processing techniques have been described but the best pre-process pipeline has yet to be found.

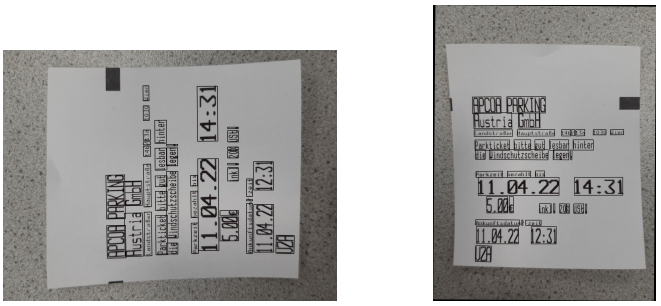
After analyzing LayoutLMv3's authors Huang et al. (2022) source code, their model has been pre-trained with a binarization technique. It uses an adaptive threshold from OpenCV with the block size parameter set to 45 and the C parameter set to 11. The C parameter is a constant that is subtracted from the mean or weighted sum of the neighborhood pixels. These parameters were kept for the experiments. Furthermore, blurring, with a kernel size of 3 by 3, was found to increase the performance of the binarization step. An example of this improvement is shown in Figure 4.6. It can be seen that the image is lighter and has better contrast between the text and the background. These two techniques are therefore coupled and put into the pipeline.




(a) Without blurring (b) With blurring

Figure 4.6: Improvement after binarization

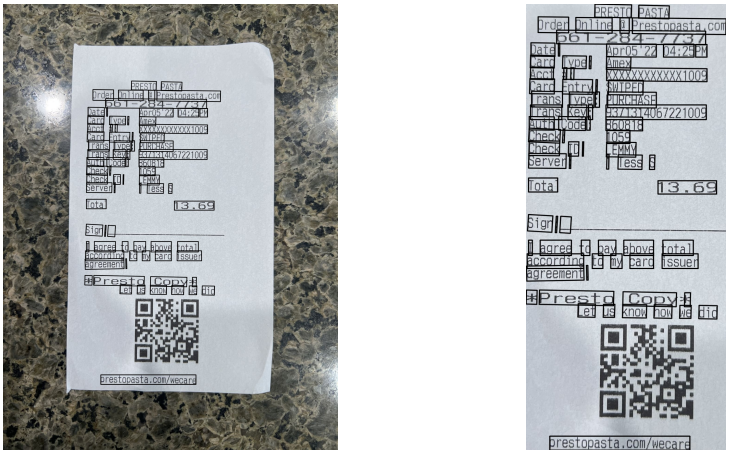
As the dataset employed for pre-training the model was too heavy to analyze and because the models use layout information, it has been decided, with minimal testing, to add rotation to the pipeline as it was quite easy and fast to implement. An example of rotation is illustrated in Figure 4.7. All the images are rotated, even those that are slightly disoriented.



(a) Without rotation (b) With rotation

Figure 4.7: An example of rotation

Cropping, shown in Figure 4.8, was more challenging. Cropping will get rid of the image background by cropping everything outside the image contour box. It requires more testing to see if it does improve the models' performances. This technique represented thus a point of uncertainty and will be further tested in the next chapter.



(a) Without cropping (b) With cropping

Figure 4.8: An example of cropping

Afterward, the thinning technique was dropped as the results provided were not satisfying enough in our opinion. Even though it performs skeletonization successfully, it drastically lowers the image quality. An example of thinning is illustrated in Figure 4.9.


(a) Without thinning (b) With thinning

Figure 4.9: An example of thinning

Morphological operations were also dropped because they did not produce a significant effect on images.

Finally, as normalization is a mandatory process required by the models, it has been added to the pipeline. Furthermore, resizing is already done internally by the LayoutLM models. The images are resized to a 224 x 224 pixels scale. This scale may seem quite small but it should be taken into consideration that the

models have access to the token's words. The images are only used to make a correspondence between the tokens and the layout positions.

The final pipeline employed for the experimentation is illustrated in Figure 4.10. The structure consists of two branches, one for each model. The crop technique is annotated with a question mark as it needs further testing to be proven effective.

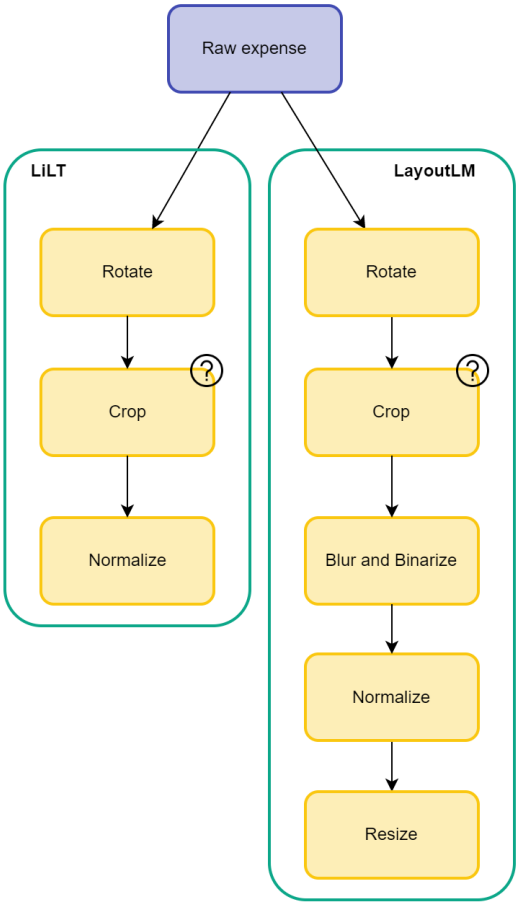



Figure 4.10: Image processing pipeline

### 4.5.1 Data binning

As stated previously, data binning consists of splitting data into several distinct bins. This allows us to regroup the same kinds of expenses together and investigate in more dept those that could lead to errors.

The distribution of each bin when considering the entire expense dataset is shown in Figure 4.11. There are a total of ten bins, and a brief description of each one is provided below.

- Regular: This represents "regular" expenses without special cases.
- Two expenses: This bin refers to pictures of two expenses side by side. Multiple expenses on a single picture can lead the model in the wrong direction.
- Textual background: This bin represents expenses with text in the background. For instance, a picture of an expense taken on an open book. As said before, cropping relies on backgrounds without text areas to work. Moreover, even without cropping, textual backgrounds add noise.
- Handwriting: This bin refers to expenses with handwritten text on it. The OCR has more difficulties reading handwritten text and this may lead to spelling mistakes.
- Damaged: All the damaged expenses with blurry pictures or cut expenses are regrouped in this bin. Note that the required information such as total, date, and currency is still visible. Damaged expenses do not contain all the structure of an expense which can be harder to extract information.
- Colored: This bin regroups expenses printed on colored paper. Even with binarization, some special colors will remain and add unnecessary noise.
- Screenshot: This bin represents pictures of expenses that are taken from a computer screen. A picture of a screen is of very low quality.
- Overwriting: All the expenses that have a stamp on them are regrouped in this bin. Stamps make it more difficult for the OCR to get the right information.
- Special tokens: This bin regroups expenses with exotic alphabets such as Arabic or Chinese. Another problem for the OCR which is heavily trained in Latin alphabets.
- Long text: This bin regroups expenses with elongated text. These expenses contain very long descriptions, which will add noise and be truncated by the model. This truncation might cut off the total or other fields of the expense.

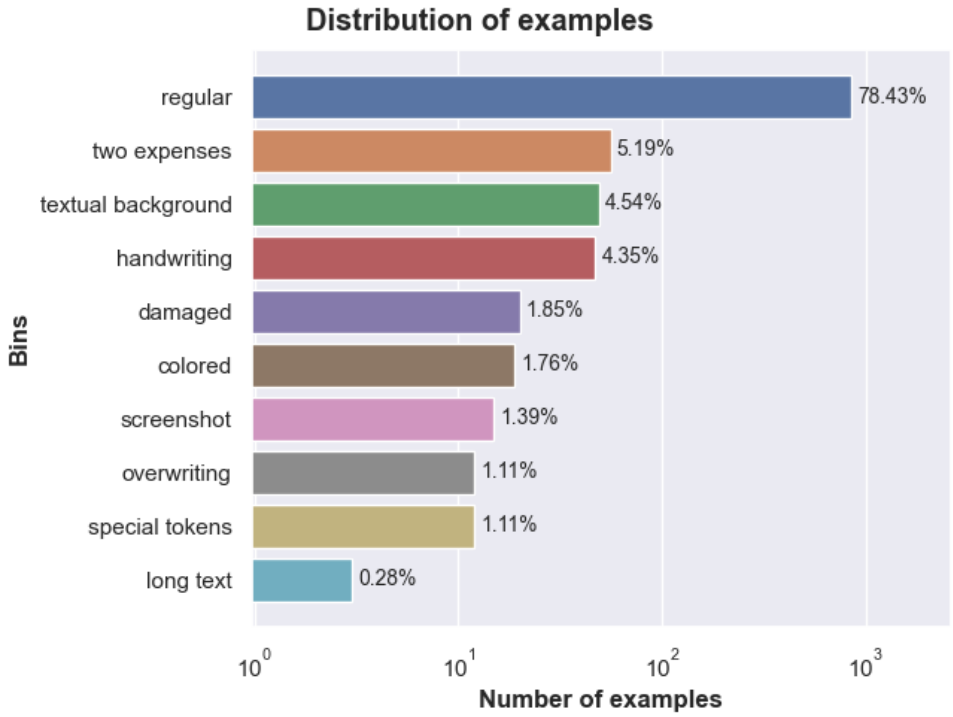



Figure 4.11: Data distribution

It stands out from Figure 4.11 that roughly 22% of the expenses contain some characteristics that may cause errors in the model's predictions. Note that the bins are mutually exclusive. If an expense contains several types of errors, it would be removed to avoid data duplication. Once the dataset is cleaned and binned, it is composed of 1080 expenses.

The experiments conducted in the next chapter will take these bins into account.

### 4.5.2 Limitations of the data

The limitations and discrepancies of Odoo's expense dataset are inferred from comparisons with the CORD and FUNSD datasets. Firstly, the dataset is relatively small bin-wise, which may result in poor performance because there is not enough data to fine-tune the model. An idea to improve this could be to augment the data using affine and perspective transformations on the images to alter their rotation, scale, and orientation. However, implementing these augmentations methods was not considered in the current work and would need to be done in future research. Additionally, there is a difference in the language used in the expense dataset compared to the other two datasets. The expense dataset is mostly composed of French written expenses, with a minority in English and Spanish, and a few other

languages. Moreover, the data received from Odoo are extracted in a particular interval of time. Users sometimes upload multiple times the same kind of expense (e.g. a specific parking ticket), which might lead the model to be biased towards those specific types of data more than what it should be on those extracted in a real-world normal (i.e. Gaussian) distribution setting.

### 4.5.3 Annotations

As mentioned above, the fields that need to be extracted are the total, the date, and the currency. In each case, the corresponding BIO tags are B-TOTAL, I-TOTAL, B-DATE, I-DATE, and B-CURR. The currency exists only in a one-token format setting across the dataset thus the I-CURR label is not mandatory. The annotation process was performed in a semi-automatic way. First, a sample of the dataset was selected and annotated by hand. This sample was then employed to train a base model (LayoutLMv3 was chosen for this purpose). The base model was then used to predict the labels for the rest of the dataset. Finally, the entire annotated dataset was corrected by hand to fix any annotation errors made by the model. Figure 4.12 below, shows the difference between a person who is annotating and a machine. Manual labeling is time-consuming, and logic gets lost over the process. Programmatic labeling, on the other hand, uses a labeling function. Hence, the outputs of this function follow a consistent pattern, and thus, the labels produced are less prone to annotation errors.

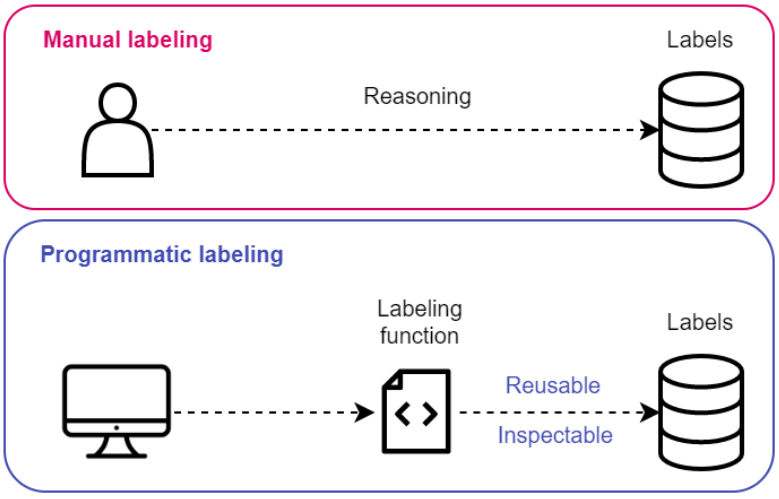



Figure 4.12: Labeling process inspired from Prakash (2022)

## 4.6 Expense information extraction workflow

Having defined the overall methodology used, let us turn to an example of how the field predictions for an expense sample will be extracted. This section briefly explains the handling of the inputs and the outputs of the models by detailing the different pre-processing and post-processing steps involved. Then, a concrete example of information extraction from an expense image sample is presented.

### 4.6.1 Data handling and prediction

After getting the OCR annotation from Google Vision, the images are passed through the image processing pipeline presented earlier. Those data will then be fed to the model. The inputs will or will not include an image, depending on the model considered.

Regarding the model's prediction, it is usually simple to convert the model's raw outputs into its final output values. However, to recover unstructured data from an expense, additional post-processing procedures are required.

During the output predictions of the model, each token in the input is given a label from a list of predetermined BIO tagging labels, as previously mentioned. A probability distribution over these classes is the model's output for each token. The label for the token with the highest probability is then chosen.

#### Tokens decoding

Each model encodes the words given as input into tokens. To generate the correct predictions and the corresponding bounding boxes, only tokens that represent the start of a word are considered. For this purpose, offset mapping is used, provided by the model's processor, to identify which tokens correspond to sub-words.

#### Parsing the outputs

To create a single value for each label (one value for total, date, and currency), the corrected predictions must be processed and the BIO tags must be collapsed. After that, each token has one of the TOTAL, DATE, or CURR labels. Then, a post-processing function specific to the label is applied to this list of candidates to produce the final value. The specific rules used for post-processing depend on the label to be processed. For the total, a regular expression is first used to remove any irrelevant tokens and extract only the total omitting any noise. Then from the possible several candidates, the most frequent is selected and parsed to a floating-point number. If there is no single most frequent total, the highest total

is chosen. For the date, a similar procedure is used. The date is parsed, with a regular expression using a library, to the format "YYYY-mm-dd" and the most frequent candidate is chosen. In the case of ties, the most recent date is taken instead. The same goes for the currency except for the parsing, which uses Odoo in-house parsing by comparing the candidates to hard-coded currency symbols. Additionally, as Google OCR provides the languages of the expense, in case of a missing currency, it is possible to retrieve a currency based on the language of the expense. This approach was dropped as the Odoo IAP team wished not to assign a currency by default if there was none.

### 4.6.2 A concrete example

In this section, the information extraction process from the raw inputs to the final post-processed model output will be shown. For this purpose, let us take one image from the test set and try to extract the three relevant fields total, date, and currency.

The input to the model is made up of the tokens and bounding boxes retrieved from Google OCR after feeding an image. For this particular example, 118 tokens and bounding boxes are retrieved. Figure 4.13 represents the image sample.


Figure 4.13: Image sample

The model outputs a list of length 118 composed of the corresponding NER-tagging to the tokens.

After proceeding to the post-processing of the output explained in the section 4.6.1, the three parsed fields: 47.8, '2022-01-03', 'EUR' are retrieved.

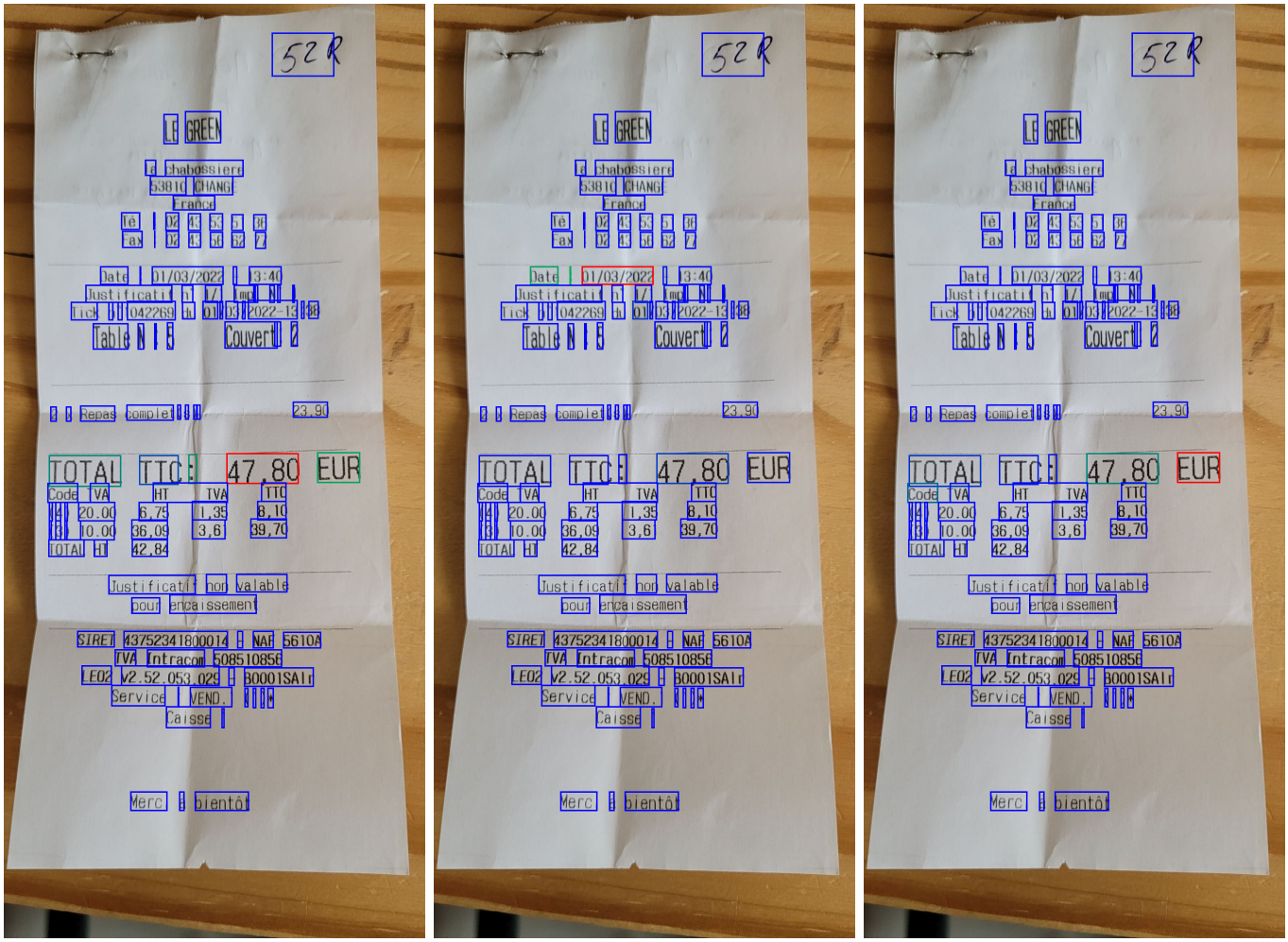
### Attention visualization

For recall, the attention mechanism in a Transformer is a way for the model to selectively focus on certain parts of the input sequence when processing it. Attention visualization is then a way to visualize the parts of the input sequence the model is focusing on at each step in the processing.

During the information extraction process, the model may focus more on certain words in the input, based on the context and the task at hand. For example, if the goal is to extract a total amount, the model might pay extra attention to words like "TOTAL," "TOTAAL" (in Dutch), or "jumla" (in Swahili). Attention visualization allows us to see which words the model is paying more attention to and it can help us understand how the model is arriving at its predictions. To achieve this, the attention weights from the output can be extracted for specific tokens. For instance, in a total extraction task, the token of interest is "47,80". For each token, the weight assigned to it with respect to the token "47,80" can be retrieved. Since there can be multiple attention heads, the mean of all of them will be taken. To gain a better understanding, let us color the bounding boxes that correspond to the tokens. This is shown in Figure 4.14.

The colors are based on a heatmap ranging from blue, which represents either no weight or a slight weight, to red, which represents a high weight. Green is located in the middle and represents a medium weight. In Figure 4.14a, attention visualization is shown for the total extraction task. Attention visualization for the date extraction task is shown in Figure 4.14b, and attention visualization for the currency extraction task is shown in Figure 4.14c.

For Figure 4.14a it can be seen that the model pays close attention to the tokens "47.80" and to the tokens "TOTAL", ":" and "EUR" when the total amount needs to be found. In fact, it is similar to what humans would do to perform the same task. Someone would first search for the token "TOTAL" or any translation and find the total amount on the same line. Similarly, in Figure 4.14b, the model puts more weight on the tokens "Date", ":" and "01/03/2022" when extracting the date. Finally, in Figure 4.14c, the model focuses on tokens such as "EUR" and "47,80" to find the currency.




(a) Attention with respect to the total

(b) Attention with respect to the date

(c) Attention with respect to the currency

Figure 4.14: Attention visualization

## 4.7 Experimental setup

This section provides a high-level overview of the technical environment in which the experiments were carried out. The following hardware and software were used to gather experiment results for this thesis.

### 4.7.1 Models configurations

As all the models are available as "off-the-shelf" models, ready to be fine-tuned and tested, there are only a few needed configurations. To begin with, the epoch refers to a single pass through the entire training dataset. This means that each training example will be used once per epoch. After each epoch, the model weights are updated based on the learning algorithm. Typically, multiple epochs are employed when training a Deep Learning model to improve its accuracy. The number of epochs to use is often a hyperparameter that is determined through experimentation (Rosebrock, 2017) and will be fixed to 7, in our case, for all the tested models.

The second hyperparameter that will be manually set is the learning rate. To recall, the learning rate is a hyperparameter that determines the step size at which the model's weights are updated during training. It is a scalar value that is often set between 0 and 1 and determines how quickly the model learns (Goodfellow et al., 2017). This hyperparameter will be set to $1e^{-5}$ or $5e^{-5}$, depending on the model considered.

### 4.7.2 Hardware and Software Specifications

The experiments were conducted in the Google Colaboratory environment with the following specifications checked through shell commands.

- GPU: One Tesla T4 with 16GB GDDR6 VRAM
- CPU: One single-core hyper-threaded Xeon Processor @2.00GHz
- RAM: ∼13 GB available
- Disk: ∼100 GB Available

Due to the restrictions of the Google Colaboratory free plan, which only permitted a small amount of GPU utilization of about 3 to 5 hours, testing and training our models in this environment was thus quite difficult. This significantly reduced the number of tests that could be run.

### 4.7.3 Python Package

Having convenient development and testing settings are essential as it allows less overhead and saves valuable time. In this regard, a dedicated pip package has been created with all the required dependencies to run our models on the Google Colaboratory environment. The composition of this package is briefly explained below.

- A dataset builder: This file is responsible for creating the datasets for the experiments.
- A dataset wrapper: This file is a wrapper for the Hugging Face datasets module to build, pre-process and load our zipped dataset into memory.
- Model, LayoutLM, and LiLT classes: The model is the parent class of the LayoutLM and LiLT classes. They have everything in common except for the tokenizer as LiLT does not need visual input. LayoutLM and LiLT are the classes used to create an instance of the model.
- Pre-processing classes: It represents all the files responsible for the pre-processing of the expense images. It includes rotation, cropping, binarization, and normalization techniques.
- Pre-process and post-process utility classes: Pre-process handles the reading of the OCR parse, the image, and the pre-processing techniques applied to it. It is called by the dataset wrapper. On the other hand, the post-process handles the parsing of the total, the date, and the currency to a standardized format after the prediction of the model. This post-process will be explained further in the next chapter.
- A compose class: A pipeline that will automatically run the pre-processing classes on the data. It is called by the pre-processing module.
- A tester: It runs multiple tests to ensure that all the images and bounding boxes transformations are correctly done.
- A setup: This file contains the dependencies required to run and log our experiments.

The package presented above will essentially be used for testing and training the model. The final model that will be provided to Odoo will be put in another package that includes everything needed to run predictions. To briefly described, the package will contain a model folder that contains the exported model after training. A model class is used to instantiate the model folder and will only contain the logic required to generate the predictions. It will also have the same pre-processing, pre-process/post-process utilities, and compose classes as the previous package. Finally, a requirements file will be added to ensure that all dependencies needed to run the model are correctly installed.

# Chapter 5

# Experiments

## Contents



Given all the relevant background information and the established methodology, the results of the experiments carried out on Odoo's multilingual expense dataset are presented in this chapter.

In the first part of this chapter, the training of the Hugging Face models and the construction of the initial datasets will be described in detail. This covers a discussion on the different limitations of the models, as well as determining which model will be chosen for further experimentation.

Once the most appropriate model has been chosen, the next step will be to tune the model using the data-centric workflow. This means that multiple training sets will be created for different types of experiments.

Finally, the evaluation of the model will be described. This will include a discussion of the overall performance of the model compared to the baseline model which will be introduced later. Altogether, this chapter will provide a comprehensive overview of the experiments conducted on the expense dataset and the results obtained.

## 5.1 Training the information extraction models

As stated previously, the thesis's focus is to train models in a data-centric way. It means that the data are of high importance and that the model/code is considered to a lesser extent. The reason for this approach is that, when focusing solely on the model, the significance of the data can be overlooked. Therefore, a hybrid approach that takes both the data and the model into account is usually the best option. However, the priority given to the data versus the model can vary depending on the specific application at hand (Patel, 2022).

The goal of this thesis is to primarily focus on the data while only considering two hyperparameters: the number of epochs and the learning rate. This focus on the data and the hyperparameters will help ensure that the models are trained effectively and accurately.

### 5.1.1 Building the datasets

As explained before, three types of datasets need to be created, the training set, the test set, and the validation set.

The datasets will be created iteratively in a data-centric way, as mentioned in the previous chapter. This is done in a stratified process. For the test set, each bin is sampled with 10% of the initial dataset. The validation dataset is built in the same way as the test set and also uses a 10% split. The remaining data are our training set. Ten percent was chosen because the CORD dataset, which contains 1000 examples, uses this split.

For example, the dataset containing all the bins is split into a training set of 867 examples, a validation set of 100 examples, and a test set of 113 examples, each set containing a fraction of the distinct bins.

The datasets will change when considering different models. Indeed, the LayoutLM-type model takes as an input an image with the corresponding tokens and bounding box coordinates extracted with the OCR engine. In addition, when training, it takes the NER-tagged tokens corresponding to the tokens being labeled as explained previously. On the other hand, the LiLT-type model does not take an image as input, only the tokens, the bounding box coordinates, and the NER-tagged tokens. Note that LiLT still needs the image dimensions for the pre-processing.

Concisely, the features of the dataset can be described as follows. A sample (which corresponds to an initial image of an expense) is composed of an **id** which is a string id taken from the Odoo backend. Then, the **tokens** are a list of strings corresponding to the output of the OCR engine. The **bboxes** (bounding boxes) is a list of integers that initially corresponds to the four coordinates (x,y) of a token in the expense (taking into account that the upper-left corner consists of the origin). Those coordinates are then pre-processed to only keep the upper-left and lower-right corners, which is the mandatory format for each model. The **ner_tags** is a list composed of one of the six class labels considered, i.e. O, B-TOTAL, I-TOTAL, B-DATE, I-DATE, and B-CURR for each token. Finally, the **image** is added depending on the model.

### 5.1.2 Fine-tuning the models

All models that are considered extend the initial BERT model. Having this in mind, the models are first pre-trained with a lot of different types of data before being fine-tuned on the custom dataset presented. LayoutLMv3 is pre-trained with a large IIT-CDIP dataset to learn a universal representation for multiple document tasks. The IIT-CDIP Test Collection 1.0 is a large-scale scanned document image dataset with around 11 million images that can be divided into 42 million pages. Only 11 million of them are used to train LayoutLMv3 (Huang et al., 2022). The same pre-training is performed on the LiLT model (Wang et al., 2022) and on the LayoutXML model which, in addition, is initialized with the InfoXML model for cross-lingual language model pre-training. However, the LayoutLMv3 model's core purpose is to be used on an English-written text. LiLT, even when being pre-trained on the IIT-CDIP dataset can be adapted to additional languages. It is the first language-independent solution for structured document interpretation in this regard.

The pre-trained models are taken as it is from the Hugging Face's hub. For the fine-tuning part, the input needs first to be encoded in a specific way. The encoded input is made up of **pixel_values** which corresponds to a batch of document images. This is exclusive to the LayoutLMv3 model, as LayoutXLM uses a resized

image for its CNN-based visual encoder. As mentioned above, the LayoutLMv3 model architecture is made up of a ViT that divides each image into patches of a specific shape. Then, the **input_ids** are the indices of input sequence tokens in the vocabulary. During pre-processing of the model, a tokenizer is used to turn words, word-level bounding boxes, and optional word labels into token-level input ids that are taken from a restricted vocabulary. The **boxes** and **labels** correspond to the bounding boxes coordinates and the NER tagging respectively. Finally, an additional feature is added, namely the **attention_mask**. The attention mask is a binary tensor that indicates the position of the padded indices so that the model does not pay attention to them. As all the encoded image tokens have different lengths, they cannot be combined in the same tensor. A unique sequence length has to be set to pad to the length of a small-sized encoding or to truncate to a larger encoding (Hugging Face, 2022).

The models are then trained on the training set while tuned on the validation set. Each model has a number of epochs fixed to a maximum of 7 with an early stopping with the patience parameter set to 3. The early stopping consists in halting the learning when the model performance is not improving considering a metric (usually the validation loss is considered but in this case, the metric F1 is used). A learning rate of $5e^{-5}$ is used for the LiLT model and $1e^{-5}$ for the LayoutLMv3 model.

Another aspect that needs to be taken into consideration is the fact that the tokenizer of the two models only accepts a list of tokens of a maximum size of 512. If an encoding has more than 512 tokens, it will be truncated. This configuration will limit the model to predict the expense's image which includes limited text.

K-fold cross-validation is a popular technique in Machine Learning for assessing a model's performance. The data are divided into k subsets, and the model is trained on k-1 of the subsets while the remaining subset is used as a validation set. This procedure is carried out k times, with each iteration using a different subset as the validation set. The model's performance is then estimated using the average score over all k iterations. This strategy makes for a more thorough evaluation method than employing only one validation set because it allows the use of all data for both training and validation (Russell and Norvig, 2022).

Deep Learning can require a lot of resources when training several models using k-fold cross-validation. Consequently, it is sometimes advised to employ only a hold-out set for validation, which is made by choosing a random subset of the training data, as presented previously (Lyashenko, 2022). Therefore, the following experiment processes did not employ the k-fold cross-validation strategy

for several reasons. First, as there are not many resources available in the work environment (i.e. Google Colaboratory), each training run is a time-limited process. Furthermore, when employing k-fold cross-validation, the tiny amount of data in each bin made it difficult to divide across the different folds. Instead, multiple training sets were constructed using various random states to ensure unbiased results and prevent overfitting.

## 5.2 Model comparisons

Having explained how the data will be handled and fed to the models, let us compare the performances of the different models as well as their limitations.

### 5.2.1 Token level F1 score comparison when training on the whole training set

Table 5.1 represents the performances retrieved by running the models on the test set first by training them on the entire training set. This was managed by logging all the data to the Weights & Biases platform and manually encoding them in percent. Because the training set is relatively small, the "BASE" version of the models was employed for training. The best results are shown in bold column-wise (for a metric like F1, the higher the score, the better). Here, for simplicity, each model was trained using one training set generated with one random state.

| Model | F1 | F1 Total | F1 Date | F1 Curr |
|---|---|---|---|---|
| LayoutLMv3 | 94 | 82.7 | 92.2 | 98.8 |
| LayoutXLM | **94.8** | **84.3** | **93.1** | 99.3 |
| LiLT-RoBERTa | 93.7 | 79.7 | 91.7 | **99.7** |
| LiLT-InfoXLM | 92.7 | 75.2 | 91.1 | 99.6 |

Table 5.1: Token level F1 score of the models

Table 5.1 shows that there are no significant performance differences between the different models. Overall, it seems that the performance of the models is comparable one to one another, with no particular model significantly surpassing the others.

### 5.2.2 Comparing the execution time

Another aspect worth considering is the execution time of each model. As the future model will be used by customers to retrieve relevant fields from expenses,

its execution time is crucial. Indeed, customers value the time of execution since it has an impact on their user experience. Clients may experience frustration and delays in their work or other activities if the Odoo's OCR service takes a long time to complete the information retrieval. Furthermore, having an adequate execution time was also a requirement of Odoo. That is why the times of loading and prediction of all the models will be compared. The results of the execution time for predicting one image are presented in Table 5.2. The results are shown in seconds, and the best results are in bold by considering each column (for execution time, the shorter the time, the better). The pre-processing and the post-processing are included in the prediction time.

| Model | Loading | Prediction | Total |
|---|---|---|---|
| LayoutLMv3 | **13.395** | 7.211 | **20.607** |
| LayoutXLM | 55.191 | 8.835 | 64.025 |
| LiLT-RoBERTa | 13.538 | 7.409 | 20.947 |
| LiLT-InfoXLM | 22.720 | **6.679** | 29.340 |

Table 5.2: Models' execution time

The LayoutLMv3 model, closely followed by the LiLT-RoBERTa model, has the quickest overall execution time, according to Table 5.2. Despite having top tier performance, the LayoutXLM model now takes the longest to execute, taking more than three times as long as the LayoutLMv3 model.

It is important to remember that the tests were performed in the Google Colaboratory environment; thus, there might have been more fluctuations than with a typical on-premise execution. Therefore, the small differences between the top three algorithms can be neglected. The LayoutXLM model being substantially slower than the others is the only notable conclusion from the presented table. This could be explained by the fact that it uses a CNN instead of a Vision Transformer to extract visual features.

### 5.2.3 Short license comparison

Licenses should not remain unchecked when choosing a model. It is crucial because Odoo's IAP team plans to leverage the predictions of the chosen model for commercial use. Before proceeding, it is helpful to explore the two types of software licenses that are important to take note of while also defining a software license.

Any legally binding document that specifies how software can be used and distributed is known as a software license. It specifies the obligations and limitations of both parties to the agreement while also granting users the ability to access and

use the program. Additionally, provisions governing fair use, liability restrictions, warranties, disclaimers, and protection against intellectual property infringement, may be included in software licenses. Depending on the conditions for software copying or distribution, licenses can be classified as proprietary, free, or open source (Lutkevich and Lebeaux, 2021).

Any program intended to use should have a license carefully considered, especially if it is intended for business use, as is the case in this thesis. Users are free to use, modify, and distribute the software under the terms of the MIT license, which is a permissive open source license. This indicates that users are free to use the software for any purpose, including profitable ones, without obtaining authorization or paying a charge. Although, they must give credit to the original authors and include a copy of the license with the software (Open Source Initiative, 2022). The content of the LiLT project itself is licensed under the MIT license.

A stricter license known as Creative Commons Attribution-NonCommercial-ShareAlike 4.0 International (CC BY-NC-SA 4.0) allows others to reuse and modify the work as long as they credit the authors, do not use it for commercial gain and distribute any modifications made under the same license. The content of the LayoutLM project itself is licensed under this specific license. This means that if someone wishes to use the LayoutLM type model for business, he must get the authors' permission and may have to pay a charge. Before utilizing any software, it is crucial to read and comprehend the conditions of the license, as failing to do so may have legal repercussions (Commons, 2022).

### 5.2.4 Choosing the most adapted model

After reviewing the execution times and the licenses mentioned above, it has been decided to focus on the **LiLT-RoBERTa** model for further experimentation.

The LiLT-RoBERTa model was chosen for several reasons. For starters, it has a commercial license, which is critical for the commercialization potential of Odoo's product. Second, it trains faster than the LayoutXLM model, which employs a CNN decoder. Then, it has one of the shortest overall execution time among the considered models. Finally, unlike the LayoutXLM model, the LiLT-RoBERTa model did not necessitate plenty of resources to be run which could lead to memory overflow. Overall, these factors influenced the decision to conduct further research with the LiLT-RoBERTa model.

## 5.3 Model tuning with the data-centric workflow

Several types of experiments will be considered to optimize the model. In the first experiment, the focus will lie on image pre-processing techniques to improve overall performance. This experiment will focus on an ablation study conducted on the cropping processing technique. This method was identified as an area of uncertainty in the previous chapter.

The second experiment that will be conducted is an ablation study on data binning. As seen in the already presented Figure 4.11, the majority of the bins represent only a small subset of the entire dataset. Previous experiments involving the removal of individual bins from the training set yielded insignificant results, so in this experiment, only two training sets will be considered: the full training set and a version that only includes the regular bin.

### 5.3.1 Cropping experiment: An ablation study of the cropping processing technique

For the cropping experiment, four LiLT models will be trained on different datasets. Two of these models will be trained with cropping included in the image processing pipeline, with one using all of the dataset and the other using a version of the dataset with the textual background removed. This bin was considered because it contains expenses with text in the background, and removing those expenses could eliminate noise and increase the overall performance. The other two models will be trained with cropping added to the pipeline similarly. Table 5.3 shows the results of this experiment.

| Iteration | F1 | F1 Total | F1 Date | F1 Curr |
|---|---|---|---|---|
| All bins with cropping | 92.9 | 76.4 | **91.8** | 99.3 |
| All bins without cropping | **93.7** | **79.7** | 91.7 | **99.7** |
| Removing textual background bin with cropping | 93.2 | 76.7 | 91.8 | **99.7** |
| Removing textual background bin without cropping | **93.5** | **78.7** | **92.4** | 99.6 |

Table 5.3: Results from the cropping ablation experiment

When examining Table 5.3, it appears that cropping does not improve the performance on the test set. To decrease the bias caused by the data, different random states could be used for the training and validation split, as stated previously. However, it has been decided unnecessary since the data are already of the same type and that there is no significant variation in the results. Nevertheless, this will be implemented in the next experiment. As shown in Table 5.3, the re-

sults are insignificant, so this pre-processing technique will not be used in the future.

While performing all the iterations, it is useful to check if our model can generalize well on unseen data. Therefore, it is critical to monitor the difference between the training and validation losses because it can reveal how well the model adapts to new data. If the difference is consistently large, it may be worth looking into it further to identify and address the cause. In general, overfitting and underfitting must be avoided at all costs. It is convenient to take a look at the train and validation losses comparison plots such as in Figure 5.1 which represents an example for the "all bins" iterations.


Figure 5.1: Losses comparison

There are a few notable observations to be made from Figure 5.1. First, by keeping an eye on the gap between the training and validation losses, it can be noticed that the gap is narrow; thus it can be assumed that there is no overfitting or underfitting. Second, the losses represented are not perfectly aligned. This is because both losses do not have the same purpose. The training loss is used to evaluate the model's performance on the training data and to guide the optimization process. The validation loss is used to evaluate the model's generalization performance on unseen data, as well as to tune the model's hyperparameters. The loss is calculated on different datasets, therefore, the non-alignment. Finally, one can ask himself why the validation loss is much lower than the training loss before the convergence of the training loss. It is due to the optimizer used by the LiLT model. Indeed, LiLT uses an Adam optimizer with weight decay. A form

of regularization, such as weight decay, can help reduce overfitting and improve generalizations. This can lead to a lower validation loss compared to training loss. The model was trained with early stopping and thus stopped at 4.5 epochs, as seen in the figure. From the training, the best model is selected based on the F1 score. The red dotted line represents at which epoch the best model is chosen.

### 5.3.2 Binning experiment: An ablation study on data binning

As said previously, the ablation study will not just be taking out one bin of the training set at a time. Instead, two separate training sets will be compared to each other in the form of two iterations, one that includes all bins and one that only includes the regular bin. According to our theory, the training set that only includes the regular bin will perform better than the entire training set. In that manner, five runs with various random sets for each to assure fairness will be conducted to verify or disprove this hypothesis. The results of this experiment can be seen in Table 5.4.

Table 5.4 groups all runs according to their corresponding random states. The highest score for each random state is highlighted in bold column-wise. To make it easier to understand, a particular emphasis will be placed on the mean and standard deviation of each training set across the different random states. Tables 5.5 and 5.6 show the mean and standard deviation of each training set, respectively.

| Random state | Bins | F1 | F1 Total | F1 Date | F1 Curr |
|---|---|---|---|---|---|
| 12 | All bins | 92.8 | 76.9 | 88.7 | 99.5 |
| 12 | Only regular bin | **98.1** | **94.2** | **96.8** | **99.9** |
| 17 | All bins | **93.0** | **78.9** | 92.1 | **99.0** |
| 17 | Only regular bin | 91.8 | 69.8 | **92.4** | **99.0** |
| 31 | All bins | 92.0 | 68.1 | **92.6** | 98.7 |
| 31 | Only regular bin | **92.3** | **72.1** | 91.8 | **98.8** |
| 42 | All bins | **93.7** | **79.7** | 91.7 | **99.7** |
| 42 | Only regular bin | 93.4 | 77.7 | **92.3** | 99.4 |
| 51 | All bins | 93.6 | 80.7 | 91.9 | 98.9 |
| 51 | Only regular bin | **94.7** | **83.3** | **93.9** | **99.0** |

Table 5.4: Results from the data binning ablation experiment

| Bins | F1 Mean | F1 Total Mean | F1 Date Mean | F1 Curr Mean |
|---|---|---|---|---|
| All bins | 93.02 | 76.86 | **91.4** | 99.16 |
| Only regular bin | **94.06** | **79.42** | **91.4** | **99.22** |

Table 5.5: Mean of the experiment's results

| Bins | F1 Std | F1 Total Std | F1 Date Std | F1 Curr Std |
|---|---|---|---|---|
| All bins | 0.61 | 4.55 | 1.38 | 0.38 |
| Only regular bin | **2.25** | **8.75** | **1.82** | **0.39** |

Table 5.6: Standard deviation of the experiment's results

From the above tables, several conclusions can be drawn. First, when using only the regular bin as the training set, an improvement in performance on average can be seen. However, it is worth noting that this training set also exhibits a significantly larger standard deviation compared to the other training set, particularly when it comes to the total field. Despite the higher average performance, the large standard deviation suggests that the results may be less consistent or trustworthy when using this training set.

To gain a better understanding of the results, let us perform a paired t-test on the various bins using these results. The t-test will be used to determine whether there is a statistically significant difference between the model prediction when the two different training sets are considered. The t-test will output a p-value, which represents the probability that the difference between the models' scores is due to chance. If the p-value is below a certain threshold (usually 0.05) for a certain bin, then the conclusion can be drawn that there is a significant difference between the models' predictions for this particular bin. This test will determine whether there is a substantial change in the model's predictions across the two iterations. A dependent t-test for paired samples is utilized because it takes into account the fact that the data from the two iterations are related. Indeed, the second iteration's data are a subset of the first iteration's data.

First of all, let us look at Figure 5.2 which represents the mean F1 score of the two iterations in the different bins. It can be seen that the difference in performance is not significant, but the overall best-performing iteration is the only regular bin, as previously mentioned.

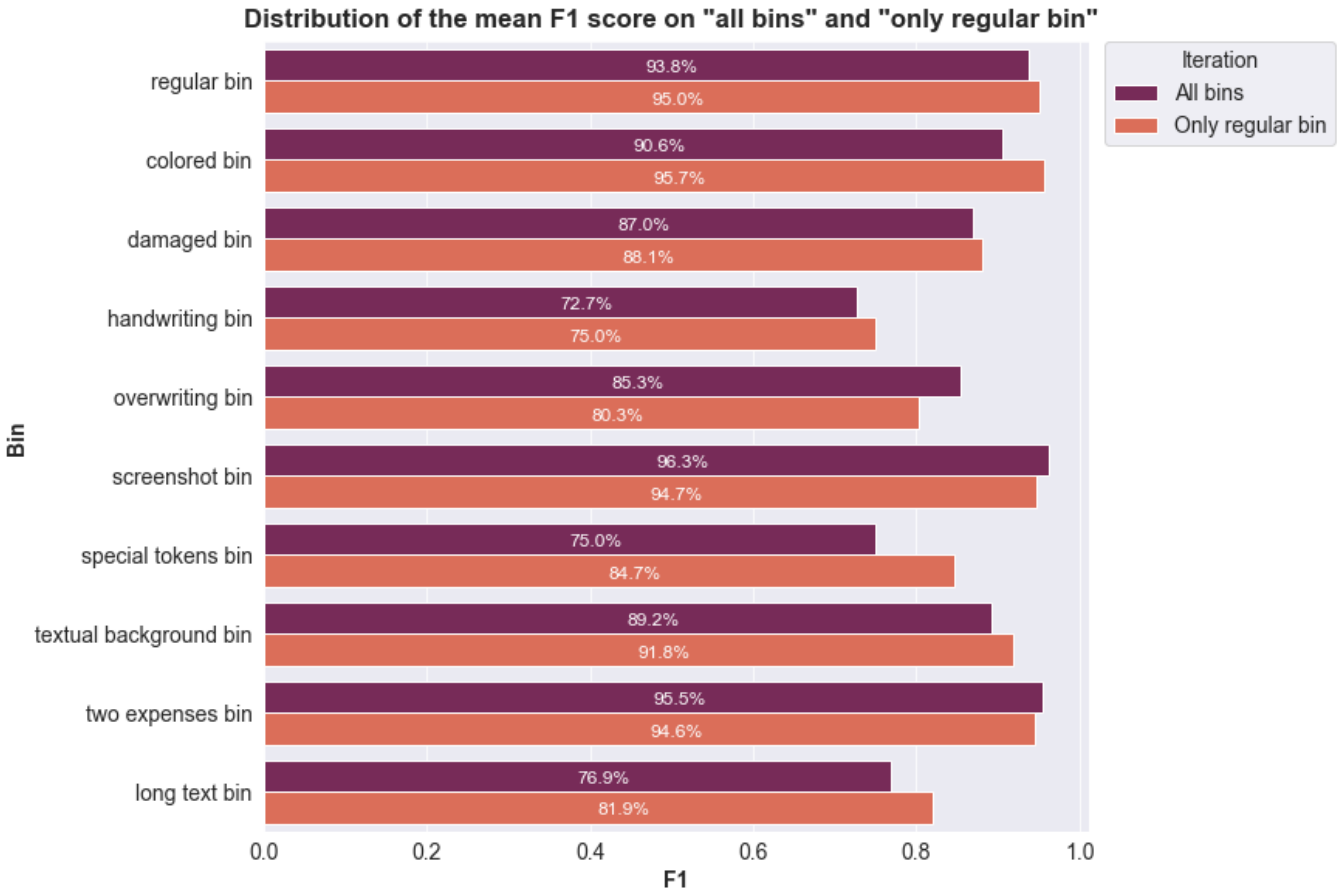



Figure 5.2: Distribution of the mean F1 score across two iterations

The p-values obtained from a paired t-test on each bin are shown in Figure 5.3. It shows that only the "textual background" bin has a p-value below the 0.05 cutoff. This indicates that the model predictions from the two iterations for this bin deviate statistically significantly from each other. Therefore, the null hypothesis can be rejected. It can be assumed that the model trained on the only regular bin has a higher F1 score than the other model for this particular bin. A similar conclusion cannot be drawn for the other bins because the p-values are higher than the cutoff.

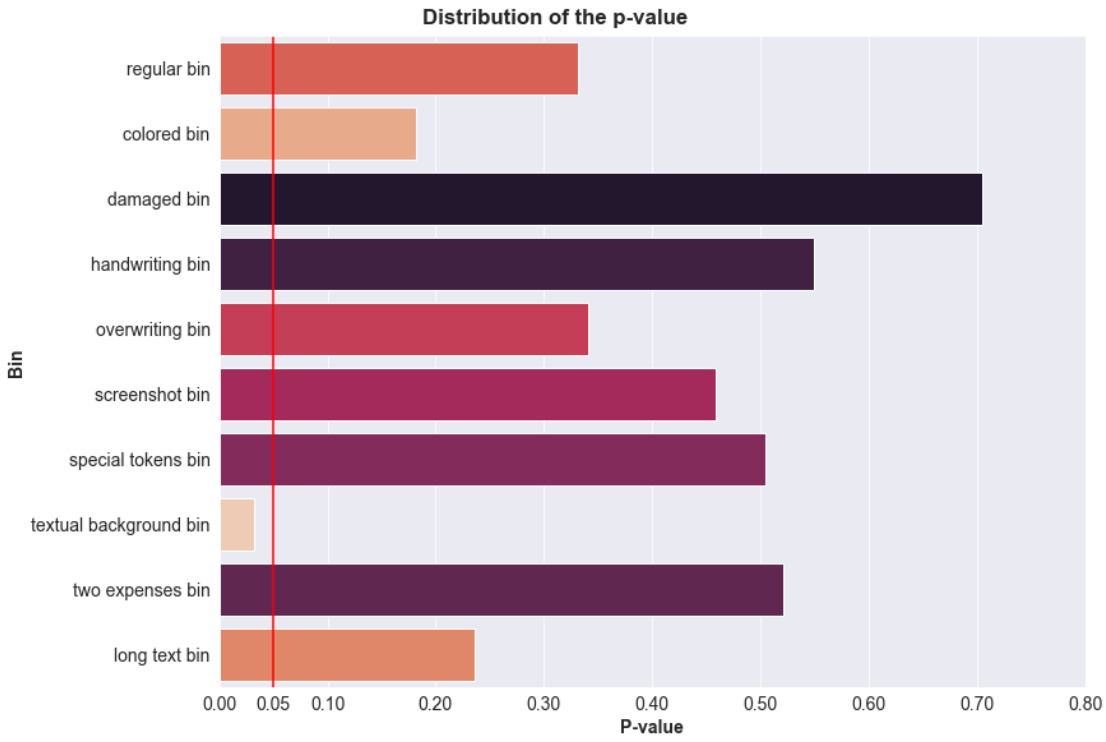



Figure 5.3: Distribution of P-values across different bins

## 5.4 Model evaluation

Now that the different experiments have been discussed it is time to see the LiLT model performance compared to the current model, which is used by Odoo and will represent our baseline model. As the main focus of our collaboration with Odoo is to improve the performance of their expense extraction tool, a particular effort will be made to outperform the baseline model.

### 5.4.1 Baseline model

To extract structured data from unstructured documents, such as expenses, Odoo's IAP team has created a custom algorithm that was mentioned before. This algorithm's performance assessment on the test set will be taken as the baseline. Its workflow will be briefly examined. When a client's document is received, the first step is to use OCR to extract the textual information of the image, which is handled by the Google Vision API. Odoo's custom algorithm then extracts the different fields from the detected text, using indicators, for instance, neighboring words, the position on the page, the font size, the syntax, etc. to determine

whether a word could be a field of the expense. Finally, the algorithm ensures that the fields are consistent with each other and adhere to the corresponding field types.

This baseline model outputs the three considered fields without performing NER-tagging, as the LiLT model does. The post-processing presented earlier is then useful because it allows for common ground between the two models and allows for performance comparisons.

### 5.4.2 Overall accuracy

The NER-tagged test set, which has been post-processed in the same manner as previously described, serves as the ground truth for this evaluation. The three relevant fields (total, date, and currency) are extracted for each sample in the test set. The post-processed outputs of the various models will then be compared to these fields, with accuracy serving as the metric for this comparison. This metric was chosen over the F1 score because the NER-tagging fixed label categories were removed during the post-processing step. As a result, the only way to compare the models' performance to the ground truth is to compute the models' accuracy based on whether they correctly "hit" or "miss" the relevant fields.

The models considered in the accuracy assessment are the LiLT models trained on "all bins" and on "only regular bin".

Table 5.7 shows the calculated accuracy for all the models. The mean of the different random states training sets for the LiLT models was taken. It can be seen that all of the LiLT models outperformed the baseline model. The field "total" was found, approximately, 25% more often than with the baseline model. The date and the currency were found more than 35% more frequently. Note that for the baseline model, the date and the currency have a different post-processing method. Therefore, these accuracies could be higher than those shown in Table 5.7.

As the two LiLT model's performances are quite close, it cannot be said for sure which one is better than the other. Because the models are already pre-trained with a huge amount of data, the performance discrepancies from the fine-tuning employing the entire dataset or a big enough subset are neglectable. The model that performed the best overall was selected as the final model to be used by the Odoo services.

| Model | Accuracy Total | Accuracy Date | Accuracy Curr |
|---|---|---|---|
| Odoo baseline | 61.1 | 57.1 | 63.9 |
| All bins model | 85.4 | 92.7 | 99.1 |
| Only regular bin model | **87.0** | **93.6** | **99.5** |

Table 5.7: Final accuracy comparison to the baseline model

The results demonstrate that the models were able to learn complex relationships from the expense dataset and effectively extract important information from them. Although the results were not as good as those reported in the LiLT paper on the CORD dataset (considering the F1 score), they are more than satisfactory enough. It is likely that by improving the pre-processing or having more quality data, the results presented on the CORD dataset would have been similar to those presented in this thesis.

Even with a limited number of samples, the LiLT model proved its ability to generalize well on multilingual data. Hypothetically, it might be interesting to look at the overall performance of the dataset in language bins to see if there are any notable performance differences.

## 5.5 Leveraging LiLT to boost Odoo business value

In the end, the model trained on the only regular bin is chosen as it performed slightly better on average and it performed better on the "textual background" bin. This decision was taken in concordance with the IAP team. The final model will be deployed by the team as it is out of the scope of this thesis.

Once deployed on the OCR service, the performance provided by the developed model will significantly improve the accuracy of the current Odoo module running in production. As a result, the customer experience and Odoo's service offerings will be enhanced. Better quality predictions will also help build customers' trust which is an important aspect to consider. Indeed, the amount of money a customer is willing to spend and his retention is tightly coupled to the performances provided by the service. By improving the quality of the module, it becomes possible for Odoo to strengthen its current pricing plan. Furthermore, Odoo includes a variety of tools that make it a powerful all-in-one platform for business management. By adding a well-performing document parser it will give Odoo a competitive advantage over other accounting and services businesses. Finally, the information extraction of expenses can also be automated or streamlined to process a batch of expenses that would otherwise require time and costs to manually encode.

## 5.6 Future experiments

Getting a look at the interpretability of the predictions is another research area. Shapley values are a method to try to answer that question. This technique is model-agnostic, which means that it can run on any model. Simply said, a Shapley value is the average of the marginal contribution of a feature value across all possible coalitions (Molnar, 2022). Coalitions refer to permutations in the sample values. Unfortunately, due to time constraints, this possibility will remain unexplored.

# Chapter 6

# Conclusion

The information extraction of expenses by adopting a data-centric approach is the main goal of our thesis. Information extraction refers to the automatic extraction of structured information from unstructured text data. In our case, this data involve images of expenses. An expense is defined as a justification for the outflow of cash from a company to an individual. In this thesis, the expenses entered by Odoo's clients into the Accounting tool are the ones used as the data source. To achieve high performances on the retrieval of information, a data-centric workflow is implemented. A data-centric approach entails systematically improving datasets to increase the accuracy of Machine Learning applications.

The final pre-processing pipeline built through data-centric iterations, applied on expenses to enhance the data, is composed of two techniques, namely, rotation and normalization. Processed and quality data is then fed to state-of-the-art Deep Learning Document AI models that parse the expenses and extract the total, the date and the currency as required by Odoo. A Deep Learning Document AI model is composed at its core of a Transformer. Transformers are a particular kind of DL model that uses a self-attention mechanism to process sequential input data which allows it to handle long-range dependencies. Transformers have the possibility of being pre-trained on large amount of data and then fine-tuned to a specific task. From the multiple pre-trained models, LiLT-RoBERTa is chosen as the main model.

The data-centric approach allows us to conduct relevant experiments through an iteration process and by dividing the dataset by different types of bins. Moreover, this binning permits performance assessment and error discovery. The experiments reveal that there is a significant difference, a 25 to 35 percent increase, between Odoo's current algorithm and the fine-tuned LiLT model. After the post-processing phase, the newly developed model has an extraction mean accuracy of 87.0, 93.6 and 99.5 percent for the total, the date and the currency respectively across five different runs.

Furthermore, there is no significant variation between a model trained on all

the dataset and a model trained on only clean data except for those containing textual background noise, where the latter performed better. Clean data is a filtered subset of the whole dataset which represents only regular cases of expenses. Therefore, with the approval of the IAP team, the model selected to replace Odoo's current algorithm is the model trained on only clean data containing regular types of expenses.

To conclude, a dedicated python package is built to give the IAP team all the required tools to run the model in production. Once deployed this model will improve the customer's experience, Odoo's service offering and it will provide Odoo a competitive advantage over other accounting and service businesses.

## 6.1 Future improvements

There are several interesting areas for further investigation in this work. One possibility would be to collect more quality data to perform more detailed tests and improve model training. This can be achieved through data augmentation by altering the perspective from which the expense pictures are taken, as well as by striving for higher data quality by encouraging users to take clearer, higher-quality pictures rather than rushing to snap a quick photo. Another area of interest would be addressing the truncation problem, as the model's tokenizer is only able to process a limited number of tokens at once. One way to tackle this issue could be to use a sliding window approach. This will allow breaking up larger lists of tokens into smaller segments and predicting one segment at a time.

## 6.2 Final thoughts

The running and sound working of the models was possible throughout several crucial processes, including data collection and data labeling, which took a huge amount of time as each expense label had to be verified by hand. Furthermore, the search for the best image pre-processing techniques and the search for the best network architecture was also not a straightforward task. In the end, we were able to enhance the model's performance by applying different data-centric strategies such as binning to satisfy the demands of Odoo.

Even though we did not build a Deep Learning model from scratch as it would take a far more considerable amount of time, we gain a worthwhile and educational experience by fine-tuning an already implemented model. Through this master thesis, we were able to learn more about Machine Learning and Deep Learning in practice and use what we learned throughout our course at UCLouvain from the taken AI and software engineering courses to solve a challenging problem.

# Bibliography


Christopher M. Bishop. *Pattern recognition and machine learning*. Springer, 2006.

Arindam Chaudhuri, Krupa Mandaviya, Pratixa Badelia, and Soumya Ghosh. Optical Character Recognition Systems for Different Languages with Soft Computing. *Springer*, 2017. doi: 10.1007/978-3-319-50252-6.

Gobinda G. Chowdhury. Natural language processing. *Annual Review of Information Science and Technology*, 37(1):51–89, 2005. doi: 10.1002/aris.1440370103.

Creative Commons. Creative Commons License Deed, 2022. URL https://creativecommons.org/licenses/by-nc-sa/4.0. [Online; accessed 15-November-2022].

Lei Cui, Yiheng Xu, Tengchao Lv, and Furu Wei. Document AI: Benchmarks, Models and Applications, 2021. URL https://arxiv.org/abs/2111.08609.

Jean-Benoit Delbrouck. Réseau de neurones, 2019. URL https://jbdel.github.io/teaching/index.html.

Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding, 2018. URL https://arxiv.org/abs/1810.04805.

Alexey Dosovitskiy, Lucas Beyer, Alexander Kolesnikov, Dirk Weissenborn, Xiaohua Zhai, Thomas Unterthiner, Mostafa Dehghani, Matthias Minderer, Georg Heigold, Sylvain Gelly, Jakob Uszkoreit, and Neil Houlsby. An Image is Worth 16x16 Words: Transformers for Image Recognition at Scale, 2020. URL https://arxiv.org/abs/2010.11929.

Pierre Dupont. LINFO2263: Lecture notes in Computational Linguistics, 2021.

Pierre Dupont. LINFO2262: Lecture notes in Machine Learning, 2022.

Encyclopedia. Digital Images. https://www.encyclopedia.com/computing/news-wires-white-papers-and-books/digital-images, 2022. [Online; accessed 6-October-2022].

Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep learning*. The MIT Press, 2017.

Google. Announcing Tesseract OCR, 2006. URL https://developers.googleblog.com/2006/08/announcing-tesseract-ocr.html. [Online; accessed 12-November-2022].

Alex Graves, Abdel-rahman Mohamed, and Geoffrey Hinton. Speech Recognition with Deep Recurrent Neural Networks, 2013. URL https://arxiv.org/abs/1303.5778.

Yupan Huang, Tengchao Lv, Lei Cui, Yutong Lu, and Furu Wei. LayoutLMv3: Pre-training for Document AI with Unified Text and Image Masking, 2022. URL https://arxiv.org/abs/2204.08387.

Daniel Jurafsky and James H. Martin. *Speech and language processing*. Pearson, 2021.

Diederik P. Kingma and Jimmy Ba. Adam: A Method for Stochastic Optimization, 2014. URL https://arxiv.org/abs/1412.6980.

Yann LeCun, Y. Bengio, and Geoffrey Hinton. Deep Learning. *Nature*, 521:436–44, 2015. doi: 10.1038/nature14539.

Zachary C. Lipton, John Berkowitz, and Charles Elkan. A Critical Review of Recurrent Neural Networks for Sequence Learning, 2015. URL https://arxiv.org/abs/1506.00019.

Ben Lutkevich and Rachel Lebeaux. What is a software license? everything you need to know, 2021. URL https://www.techtarget.com/searchcio/definition/software-license. [Online; accessed 15-November-2022].

Vladimir Lyashenko. Cross-validation in Machine Learning: How To Do It Right, 2022. URL https://neptune.ai/blog/cross-validation-in-machine-learning-how-to-do-it-right. [Online; accessed 4-January-2023].

Mark Mazumder, Colby Banbury, Xiaozhe Yao, Bojan Karlaš, William Gaviria Rojas, Sudnya Diamos, Greg Diamos, Lynn He, Douwe Kiela, David Jurado, David Kanter, Rafael Mosquera, Juan Ciro, Lora Aroyo, Bilge Acun, Sabri

Eyuboglu, Amirata Ghorbani, Emmett Goodman, Tariq Kane, Christine R. Kirkpatrick, Tzu-Sheng Kuo, Jonas Mueller, Tristan Thrush, Joaquin Vanschoren, Margaret Warren, Adina Williams, Serena Yeung, Newsha Ardalani, Praveen Paritosh, Ce Zhang, James Zou, Carole-Jean Wu, Cody Coleman, Andrew Ng, Peter Mattson, and Vijay Janapa Reddi. DataPerf: Benchmarks for Data-Centric AI Development, 2022. URL https://arxiv.org/abs/2207.10062.

Tom Michael Mitchell. *Machine learning*. McGraw-Hill, 1997.

Christoph Molnar. Interpretable machine learning, 2022. URL https://christophm.github.io/interpretable-ml-book/shapley.html. [Online; accessed 4-january-2023].

Andrew Ng. A chat with Andrew on MLOps: From model-centric to data-centric AI, 2021. URL https://www.youtube.com/watch?v=06-AZXmwHjo. [Online; accessed 12-November-2022].

Andrew Ng. The data-centric AI approach: Tips and data labeling examples, 2022. URL https://landing.ai/tips-for-a-data-centric-ai-approach. [Online; accessed 12-November-2022].

Open Source Initiative. The MIT License, 2022. URL https://opensource.org/licenses/MIT. [Online; accessed 15-November-2022].

OpenCV. Morphological transformations, 2022a. URL https://docs.opencv.org/4.x/d9/d61/tutorial_py_morphological_ops.html. [Online; accessed 4-January-2023].

OpenCV. About OpenCV, 2022b. URL https://opencv.org/about. [Online; accessed 15-November-2022].

Harshil Patel. Data-centric approach vs model-centric approach in machine learning, 2022. URL https://neptune.ai/blog/data-centric-vs-model-centric-machine-learning. [Online; accessed 15-November-2022].

B.T. Polyak. Some methods of speeding up the convergence of iteration methods. *USSR Computational Mathematics and Mathematical Physics*, 4(5): 1–17, 1964. ISSN 0041-5553. doi: https://doi.org/10.1016/0041-5553(64)90137-5. URL https://www.sciencedirect.com/science/article/pii/0041555364901375.

Arjun Prakash. Building resilient applications using data centric AI, 2022. URL https://www.youtube.com/watch?v=hmE7Cv4Ci7A. [Online; accessed 15-November-2022].

Evani Radiya-Dixit, David Zhu, and Andrew Beck. Automated Classification of Benign and Malignant Proliferative Breast Lesions. *Scientific Reports*, 2017. doi: 10.1038/s41598-017-10324-y.

Olaf Ronneberger, Philipp Fischer, and Thomas Brox. U-Net: Convolutional Networks for Biomedical Image Segmentation, 2015. URL https://arxiv.org/abs/1505.04597.

Adrian Rosebrock. *Deep Learning for Computer Vision*. PyImageSearch, 2017.

Sebastian Ruder. An overview of gradient descent optimization algorithms, 2016. URL https://arxiv.org/abs/1609.04747.

David E. Rumelhart, Geoffrey E. Hinton, and Ronald J. Williams. Learning representations by back-propagating errors. *Nature*, 323:533–536, 1986.

Stuart J. Russell and Peter Norvig. *Artificial Intelligence: A modern approach*. Pearson, 2022.

Sunita Sarawagi. Information extraction. *Foundations and Trends in Databases*, 1: 261–377, 2008. doi: 10.1561/1500000003.

Rajiv Shah, Niels Rogge, Florent Gbelidji, and Nicholas Broad. Accelerating document AI, 2022. URL https://huggingface.co/blog/document-ai. [Online; accessed 12-November-2022].

Sicara. OCR in natural images SOTA in text detection and recognition, 2022. URL https://www.sicara.fr/blog-technique/ocr-text-detection-recognition. [Online; accessed 15-November-2022].

Stanford. Image-1 Introduction to digital images, 2018. URL https://web.stanford.edu/class/cs101/image-1-introduction.html. [Online; accessed 6-October-2022].

Ilya Sutskever, Oriol Vinyals, and Quoc V. Le. Sequence to Sequence Learning with Neural Networks, 2014. URL https://arxiv.org/abs/1409.3215.

Hugging Face. Hugging Face - the AI community building the future., 2022. URL https://huggingface.co. [Online; accessed 15-November-2022].

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser, and Illia Polosukhin. Attention is All You Need, 2017. URL https://arxiv.org/pdf/1706.03762.pdf.

Jiapeng Wang, Lianwen Jin, and Kai Ding. LiLT: A Simple yet Effective Language-Independent Layout Transformer for Structured Document Understanding, 2022. URL https://arxiv.org/abs/2202.13669.

Peng Zhou, Wei Shi, Jun Tian, Zhenyu Qi, Bingchen Li, Hongwei Hao, and Bo Xu. Attention-Based Bidirectional Long Short-Term Memory Networks for Relation Classification. In *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics (Volume 2: Short Papers)*, pages 207–212. Association for Computational Linguistics, 2016. doi: 10.18653/v1/P16-2034. URL https://aclanthology.org/P16-2034.

**UNIVERSITÉ CATHOLIQUE DE LOUVAIN**
**École polytechnique de Louvain**
Rue Archimède, 1 bte L6.11.01, 1348 Louvain-la-Neuve, Belgique | **www.uclouvain.be/epl**