

MASTER'S THESIS 2023

Using Transformers to Extract Date Expressions From Receipts

Axel Leander

Elektroteknik
Datateknik

ISSN 1650-2884

LU-CS-EX: 2023-33

DEPARTMENT OF COMPUTER SCIENCE

LTH | LUND UNIVERSITY



EXAMENSARBETE
Datavetenskap

LU-CS-EX: 2023-33

**Using Transformers to Extract Date
Expressions From Receipts**

Extrahering av datumsuttryck ur kvitton
med hjälp av maskininlärning

Axel Leander

Using Transformers to Extract Date Expressions From Receipts

Axel Leander
tfy13ale@student.lu.se

August 7, 2023

Master's thesis work carried out at
the Department of Computer Science, Lund University.

Supervisor: Pierre Nugues, Pierre.Nugues@cs.lth.se

Examiner: Flavius Gruian, flavius.gruian@cs.lth.se

Abstract

The extraction of date expressions in documents is a time-consuming task if done manually. For computers, automating this process can be achieved through various methods. In this thesis, I compare a regular expression system that extracts date expressions from receipts to a machine learning system based on the transformer architecture. The idea is that a machine learning system has the possibility of being more robust to spelling errors and OCR misses. The machine learning system consists of two pre-trained transformers. To train the machine learning models, I collected training data from the regular expression system. I also generated synthetic data to complement the training dataset. The final performance of the machine learning system reached a test set accuracy of 94.62% compared to the regex systems 96.79%. While having a slightly inferior accuracy, the machine learning system was occasionally able to ignore OCR misses and extract date expressions on formats with variations relative to the training data.

Keywords: Natural Language Processing, Transformers, Data Extraction, Data Generation, Receipts

Acknowledgements

This work was made possible by a company in Stockholm, Sweden which is referred to in the report as *The Company* in cursive style. Special thanks to Pierre Nugues at LTH and Jevgenij Tsoi for support and guidance during the project.

Contents

1	Introduction	9
1.1	Motivation	11
1.1.1	Research Questions	11
1.2	Proposed System	11
2	Background	13
2.1	Tasks in Natural Language Processing	13
2.2	From Images to Text: Optical Character Recognition	14
2.3	Regular Expressions	14
2.4	Tokenization	15
2.5	Vectorization & Embeddings	15
2.6	The Transformer	16
2.6.1	Attention	17
2.6.2	Encoder-Decoder	18
2.6.3	Positional Encodings	19
2.7	CANINE	19
2.7.1	Sequence Classification	20
2.7.2	Token Classification	21
2.8	T5: Text-to-Text Transfer Transformer	21
2.8.1	mT5	22
2.8.2	byT5	23
2.9	Transfer Learning	23
2.10	Model Parallelism	24
2.11	CRISP-DM	24
3	Previous Work	27
4	Dataset	29
4.1	Exploratory Data Analysis	29
4.1.1	Purchase Date	29

4.1.2	Total Amount	30
4.1.3	VAT	31
4.2	Datasets From the Regex Module	31
5	Method	33
5.1	Processing & Building Dataset to Fine-Tune machine Learning Models	33
5.1.1	Date Classification Dataset	33
5.1.2	Generating a Synthetic Dataset	36
5.2	Potential System Architecture	37
5.3	Sequence Classification With CANINE	38
5.4	Token Classification With CANINE	39
5.5	Formatting Dates by Fine-Tuning T5	39
5.5.1	mT5	40
5.5.2	byT5	40
5.6	Complete Pipeline	41
5.7	Attempted Improvements for the Complete Pipeline	42
5.7.1	Data-Augmentation: "date"	42
5.7.2	Multiple Dates per Sequence	43
5.7.3	Improving the Post Processing	43
5.7.4	Token Classification Dataset	43
5.7.5	Fine-tune CANINE Without Synthetic Data	44
6	Results	45
6.1	Sequence Classification With CANINE	45
6.2	Token Classification With CANINE	46
6.2.1	Training on the Date Classification Dataset	46
6.2.2	Training on the Token Classification Dataset	47
6.2.3	Training Without Synthetic Data	47
6.3	Formatting Dates by Fine-Tuning T5	48
6.3.1	T5	48
6.3.2	mT5	48
6.3.3	byT5	48
6.4	Complete Pipeline	49
7	Discussion	53
7.1	Observations From Results	53
7.1.1	Sequence Classification	53
7.1.2	Token Classification	53
7.1.3	Formatting Date Expressions With byT5	54
7.1.4	Post Processing Step	54
7.2	Notable Details	55
7.3	Limitations	55
7.3.1	Dataset	55
7.3.2	Hardware	56
8	Conclusion & Future Work	57

References

59

Chapter 1

Introduction

Within most companies, employees can make purchases that are reimbursed by the company. For example, if an employee makes a trip to meet with a potential client, the company may pay for the travel expenses, the hotel, and some restaurant visits. To keep track of the expenses that the employee makes, s/he will save the receipts and hand them into their company's financial department. The financial department will then post these expenses for accounting purposes. In smaller companies, these events may be handled manually through spreadsheets, where expenses are typed in and paid out, and posted once a month. However, when companies scale and the number of receipts grows, systems to handle these expenses may be required.

Systems that handle expense reporting are often automated in many ways and may utilize machine learning in different capacities. This helps the financial department of companies to make the posting of expenses faster. *The Company*, where this thesis was carried out, produces one such system which is sold to other companies to help them post expenses for their employees. With their specific system, employees can report expenses by uploading an image of a receipt. What information a receipt is required to contain may differ from country to country. The system implemented by *The Company* handles receipts from various countries but to give an example, In Sweden a receipt has to contain:

- Company Name, including contact information
- List of products, including price per product.
- Total price, including VAT
- Date of sale.
- Date of when the receipt is written.

How this information is presented on the receipt is not determined, and the format can vary a lot.

After an image of a receipt is uploaded to *The Company*, the expense report software attempts to extract the following information:

- *Total* (Amount) of the purchase.
- *VAT* (Amount) of the purchase, which may be tax deductible.
- *Currency* of the purchase.
- *Date* of when the purchase was made.
- *Country* of where the purchase was made.
- *Category* which corresponds to which account the company should book the expense to.

Figure 1.1 illustrates an example of how this extraction may look to a customer.

Purchase date
2023-04-07

Country
Sweden

Currency
Swedish kronor


Category
Kollektivtrafik & sparkcykel

Total (SEK) VAT (SEK)
285.0 6.0

Purpose

Save expense

No transactions +



The image shows a receipt from Hallandsstrafiken. At the top, there is a QR code. Below it, the text reads 'ENKEL BILJETT' and 'vuxen 2'. The price is listed as 332:00 kr, with a group discount of -47:00 kr, resulting in a total price of 285:00 kr. The receipt is valid from 2023/04/07 09:28 to 2023/04/07 13:13, starting from Göteborg C and ending at Falkenberg station. Payment was made via BANKKORT (285:00 kr). The receipt also includes contactless payment details and a card number: 642fc6389a438606724a0721.

Figure 1.1: Example of uploaded receipt in the app.

Currently, these values are extracted from the receipt image by utilizing machine learning and rule-based systems.

1.1 Motivation

In the system, which is currently in use at *The Company*, the first step of the expense reporting software is to convert the images to text data by performing optical character recognition (OCR). The resulting text is then used by multiple subsequent modules. Currently, to extract the *date* value from the receipts text data, *The Company* uses a regular expression module to find all text sequences on the receipt which may refer to a date. The potential dates are then given to a machine learning model to classify which date, if multiple dates were given, was the date of purchase.

Regular expression (regex) is a rule-based system that matches patterns in text strings. It can be utilized for extracting data from documents with well-defined structure. However, if the structure of the data to be extracted is not in a rigid format, regular expressions quickly struggle. For this project, the dataset available and the task to be solved in practice is that of receipts from different countries and industries, making the dataset very diverse and the regular expression technique could potentially have shortcomings. This has made *The Company* interested in investigating if machine learning techniques can extract this information more efficiently instead.

1.1.1 Research Questions

Specifically the questions I am investigating in this thesis are:

- How to construct a machine-learning system that identifies and extracts date expressions from the text content of a receipt?
- While the dataset may be large and diverse, it may not be properly/fully labeled for this task. As such I will also investigate the second question: Can we improve the performance of the system by generating synthetic data to train on?

1.2 Proposed System

In this thesis, I constructed a system of two machine-learning models based on the transformer architecture. The first model identified which parts of the text content of the receipt contain date expressions by performing token classification. The second transformer model takes the date expressions and formats them to a standardized format (ISO8601, 2019) by viewing the task as a sequence-to-sequence text generation problem. Figure 1.2 shows an image of a receipt and how the system works. On a subset of the dataset provided by *The Company*, this system finds the correct date, i.e. the purchase date, on 94.6% of receipts compared to the regex modules 96.79% on the same subset.

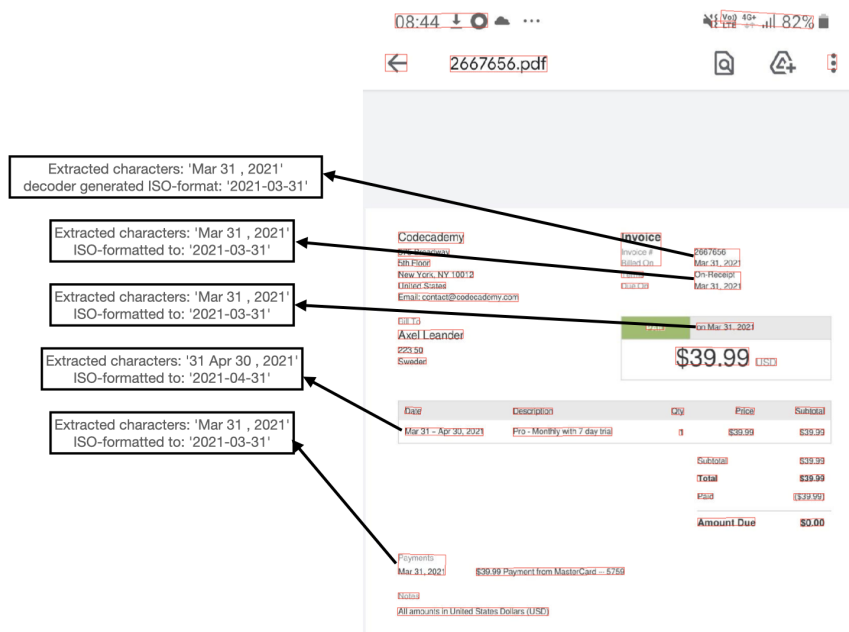


Figure 1.2: A receipt with bounding boxes of OCR segments. Dates are extracted by the first transformer model and converted to ISO format by the subsequent model.

Chapter 2

Background

This chapter gives a theoretical background and describes some relevant concepts and theories which are used in this thesis.

2.1 Tasks in Natural Language Processing

Natural language processing (NLP) is an interdisciplinary research domain that utilizes Computer Science, Statistics & Machine Learning, etc. to enable computers to understand human languages e.g. English, Swedish & French, etc.

Applying machine learning for NLP tasks can be done in various ways. Some problem formulations which will be applied in this thesis are:

Sequence Classification: The machine learning model is provided with a text sequence. The complete sequence is classified into a range of predefined categories. An example of sequence classification is sentiment analysis where a model can be trained to, for example, categorize movie reviews as either ‘positive’ or ‘negative’.

Token Classification: Similar to sequence classification. However, the difference is that each word, or character, in the input text sequence is classified individually. This task is also referred to as named entity recognition (NER) as it is often used for identifying what words, or characters, of an input text sequence describe entities such as person names, locations, organizations, etc. These entities are usually labeled with a BIO-scheme. The individual tokens are tagged in this scheme to identify the *beginning* (B), *inside* (I) & *outside* (O) of the entities. The BIO-scheme was first introduced by Ramshaw and Marcus (1995).

Sequence-to-Sequence Generation: The machine learning model generates a new output text sequence based on the input text sequence. The strength of this task lies in the fact that the input and output sequence lengths can be different from each other. This task

can be utilized in translation, for example, from English to Swedish. Summarizing text documents is another example where the sequence-to-sequence task can be applied.

These tasks have in common that the machine learning model takes input in the form of text. In contrast, the dataset available in this thesis is a collection of images. As such, the first step before applying any of these NLP tasks will be to convert these images to text.

2.2 From Images to Text: Optical Character Recognition

Consider the case where we have an image of a text document. For this situation, a common approach is to first extract the text contents from the image. This is called *Optical character recognition* (OCR). Then, a computer can read the characters in the image. The concept has been around for a very long time but has become more useful in recent decades. OCR rose in popularity as computers dominate most fields and converting paper documents to a digital format is common practice (Eikvil, 1993).

Modern OCR systems rely on deep learning to recognize text in fuzzy images and handwritten text. In this project, I used one such system produced by Google (2023) which is applied to the images of receipts to extract their text content.

2.3 Regular Expressions

Once we have recognized the characters with the OCR module, *regular expressions* (regex) can be applied to interpret patterns in the text. It is a rule-based system that extracts specific patterns from sequences of characters (Kleene, 1951). When using regular expressions we define a pattern that we want to match in a sequence of characters. For example, consider the string "I'm typing this the 13th of March 2023". If we want to extract all numbers from the string we can use a regular expression pattern that finds all numbers which would then match and extract "13" & "2023". The patterns for a regular expression can match any given set of characters, for the example above we match any continuous sequence of digits $[0-9]^+$.

Regular expression patterns can also be built to match different combinations of multiple character sets, hence building very complex matching patterns. An example of matching a combination of sets could be to find any date written in ISO-standard format. A date written in ISO format can be represented as YYYY-MM-DD, which can be found in a text by a regex pattern that matches:

- a continuous sequence of four digits
- followed by a dash
- followed by a continuous sequence of two digits
- followed by a dash
- followed by a continuous sequence of two digits.

The regular expression module which is currently in production at *The Company* uses a variety of patterns like this one to extract date expressions from receipts on different formats.

2.4 Tokenization

In contrast to regex, a machine learning system requires that the inputs are in a numeric format. The conversion from text to a numeric representation first needs to be *tokenized* i.e. breaking the text string into tokens. There are two trivial ways of tokenizing a text sequence, word-level and character-level:

character-level tokenization: the sequence is split on every single character which is tokenized individually.

word-level tokenization: the sequence is split on every word. Every word is then tokenized as a single token.

Both variations have pros and cons. For example, while character-level tokenization creates a small index for the computer to keep track of it makes the input sequences for a machine-learning model longer, and vice versa for word-level tokenization.

A popular alternative to word & character tokenization that has arisen is subword tokenization which has various implementations (Sennrich et al., 2016; Kudo and Richardson, 2018). The overall idea of subword tokenization is to combine common character sequences and tokenize them together, while more uncommon character sequences are split. This results in a reasonably sized vocabulary comprising a combination of characters, common words & sub-words. Figure 2.1 illustrates an example of subword tokenization, where the color boxes represent where the input sequence is split into individual tokens.



Many words map to one token, but some don't: indivisible.

Figure 2.1: Example of subword tokenization, using a tokenizer publicly available at Open AI.

However, in some situations, subwords may not be the ideal choice as word-level and subword tokenizations are somewhat sensitive to noise like typos, spelling mistakes, and OCR failures. In the case of multilingual problems, the vocabulary can also become larger than desired when using word or subword tokenization. Hence, in some situations, a character-level or even byte-level tokenization may be preferable (Xue et al., 2022).

In this thesis, I utilized subword-, character- & byte-level tokenization for the different models.

2.5 Vectorization & Embeddings

When the text sequence has been tokenized, we can associate each token to a corresponding integer index.

The most trivial way of providing these inputs to a machine-learning model would be to replace the text sequence with the corresponding sequence of token indices. However, this

will result in a major problem where the machine learning model will treat tokens differently depending on which value they were assigned in the index. For instance, we can imagine a case of word-level tokenization where the two words “Swedish” and “computer” are tokenized to the respective integers 100 & 101, they will look similar to the model as the numbers are close to each other.

A simple solution to these arbitrary dependencies between tokens when they are provided as integer indices is “one-hot encoding”. This is a vectorization technique where each token in the corpus is given its own input dimension, making all tokens orthogonal to each other. However, this is not optimal for a larger corpus of several thousand different words as the input space becomes very large and sparse. Another issue with the one-hot encoding is that having tokens be orthogonal to each other may not be optimal either. In reality, words have semantic relations to each other, i.e. they’re not orthogonal.

A solution to this problem is dense vector embeddings which instead map each token to a continuous vector representation. The idea is that having a reasonable number of dimensions in a continuous vector representation allows for complex relations between tokens that are semantically coherent. These continuous vector representations are trained by neural networks and were first proposed by Mikolov et al. (2013).

In modern machine learning models, these dense vector representations are often trained in tandem with the model. In a larger machine learning model, these dense vector representations can either be trained separately or together with the rest of the model. Figure 2.2 shows an example of dense vector representations for words that are trained separately. In this figure, the semantic relationship between words is illustrated by showing the similarity in the arithmetic difference between word forms for different words.

2.6 The Transformer

Recurrent neural networks (RNN) were previously a very common architecture for sequential data, such as text. Cho et al. (2014) applied the concept of having separate encoder and decoder parts of an RNN network. This introduced the possibility of providing an input sequence to the encoder and producing a new output sequence from the decoder of arbitrary length. This was a useful architecture for the task of machine translation.

The initial version of the encoder-decoder RNN architecture struggled with the processing of longer sequences. This was due to the fact that the input sequence was compressed to a fixed-length representation by the encoder. The encoded vector was then provided to the decoder to generate the corresponding output sequence. To combat the issue of compressing the input vector to a fixed length representation, the concept of “attention” was introduced by Bahdanau et al. (2015).

RNNs still had the drawback compared to other neural network architectures of being very inefficient to train. The transformer (Vaswani et al., 2017) threw away the recurrent part of the architecture while keeping the attention mechanism, making it more efficient to train as it could parallelize more of the computation. Today the RNN architecture has in large part been out-competed by the transformer for NLP tasks.

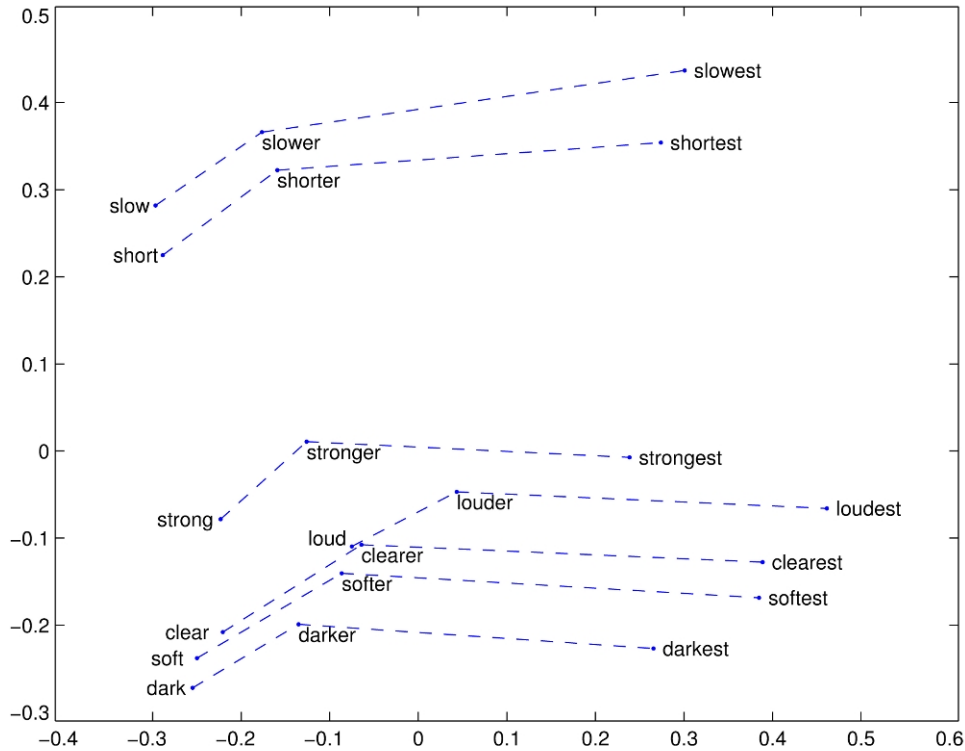


Figure 2.2: Semantic relationship between different words and their word forms in a 2D projection of word embeddings trained by Pennington et al. (2014)

2.6.1 Attention

The attention mechanism is central to the transformer architecture and is implemented as:

$$\text{Attention}(K, Q, V) = \text{softmax}\left(\frac{QK^T}{\sqrt{d_k}}\right)V \quad (2.1)$$

where K , Q & V are outputs from previous linear layers and denote *Key*, *Query* & *Value* respectively. $\sqrt{d_k}$ is a scaling factor depending on the size of K & Q . The reasoning behind the scaling factor is that for larger matrices the dot product grows large and the softmax function produces very small gradients. The general idea of the attention equation is that it gives each input token a numeric weight of how much it should 'attend' to the other tokens of the input sequence.

The attention mechanism can be, and usually is, split into what is referred to as *Multi-head attention*. This means that the *Key*, *Query* & *Value* are projected, with individual projections, to a number of heads where the attention mechanism is applied separately. The output from the different attention heads is then concatenated and fed through another linear layer to complete the algorithm. A reason for using Multi-head attention is that all the heads can be computed in parallel.

2.6.2 Encoder-Decoder

The original transformer from Vaswani et al. (2017) had the complete architecture as Figure 2.3. In the figure, the encoder is the left half of and the decoder is the right half of the image. However, a transformer does not need to have both of these parts. In fact, a transformer can consist of an encoder only, a decoder only, or an encoder-decoder. The choice of the architecture will depend on the nature of the tasks it should solve. Popular models of all three of these alternatives include:

- BERT (Devlin et al., 2019) which is an encoder only model,
- Generative Pre-trained Transformer 2 (GPT-2) (Radford et al., 2018) which is a decoder only model and
- Text-to-Text Transfer Transformer (T5) (Raffel et al., 2019) which is an encoder-decoder model.

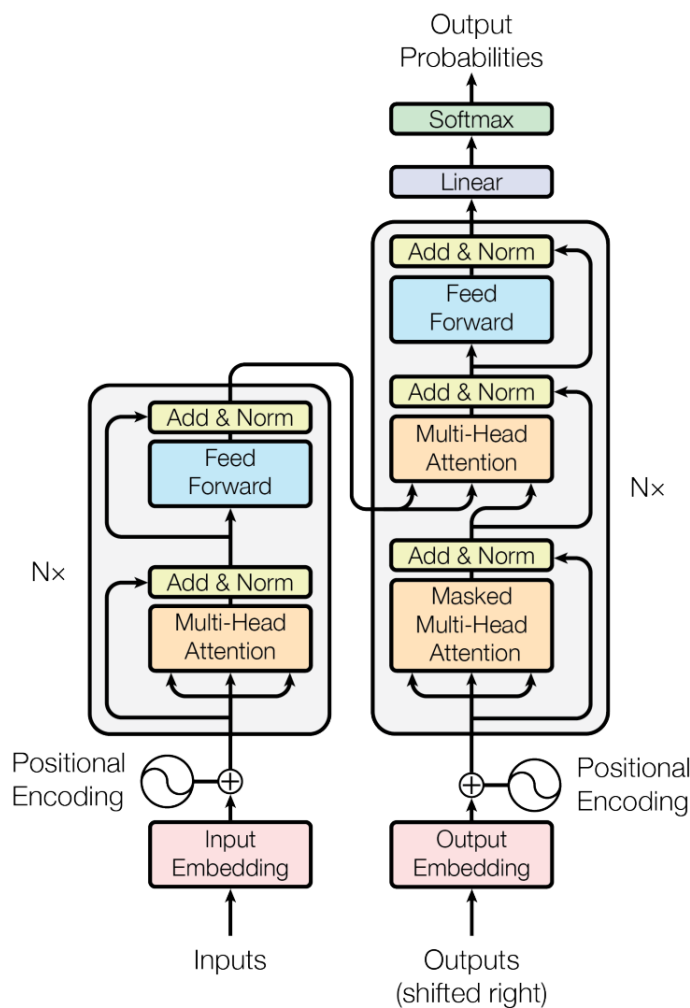


Figure 2.3: The neural architecture of the transformer as it was originally proposed. Image from Vaswani et al. (2017).

The Attention mechanism can be used in different ways depending on if it is applied in an encoder or a decoder part of the transformer. In an encoder the keys, queries and values are outputs from three linear layers with respective weight matrices W_k , W_q & W_v all operating on the same input vector x , this is usually referred to as self-attention and can be thought of as 'how much attention does the input sequence pay to its own respective parts?'.

When the attention mechanism is implemented in a decoder it is usually done in two different ways according to Figure 2.3 with one respective important adjustment in both cases. In the case of Masked Multi-Head Attention, the tokens are prevented from paying attention to subsequent tokens in the sequence, hence attention is only allowed to 'flow' backward in this case. Then there is the second Attention head in the decoder, see Figure 2.3, where the keys and values are produced by the encoder part of the transformer while the query comes from the Masked Multi-Head Attention layer in the decoder itself. This is usually referred to as cross-attention as it applies the attention mechanism between two different inputs, combined from the encoder and decoder.

2.6.3 Positional Encodings

Another feature to note on the transformer architecture is the positional encoding from Figure 2.3. The input is given as a sequence where the order of the input is relevant, hence the positional encoding modifies the input embeddings such that the same tokens don't look identical to the model if it occurs in different positions of the input sequence. The positional encoding is produced by adding a sine & cosine function to even & odd embedding dimensions respectively where the frequency is dependent on the token position. Equation 2.2 shows the exact formula that was used by Vaswani et al. (2017) and an example visualization is given in Figure 2.4.

$$PE(pos, 2i) = \sin\left(\frac{pos}{10000^{\frac{2i}{d_{model}}}}\right) \quad (2.2)$$

$$PE(pos, 2i + 1) = \cos\left(\frac{pos}{10000^{\frac{2i}{d_{model}}}}\right)$$

Since the inception of the transformer architecture, modifications have been proposed to the architecture with relatively small success. In some instances, modifications may show improvements to performance but as Narang et al. (2021) argue, most of these modifications do not seem to transfer across tasks/domains.

2.7 CANINE

CANINE is an encoder transformer model and is available for both sequence and token classification. It is pre-trained on the multilingual Wikipedia dataset which consists of 104 languages. The model is available optimized with either subword loss (CANINE-S) or character-level loss (CANINE-C). Both versions of the model use character-level tokenization and have 121 million parameters.

As character-level tokenization produces a longer input sequence for the transformer stack, this results in a model which is computationally expensive to train and use. To mitigate

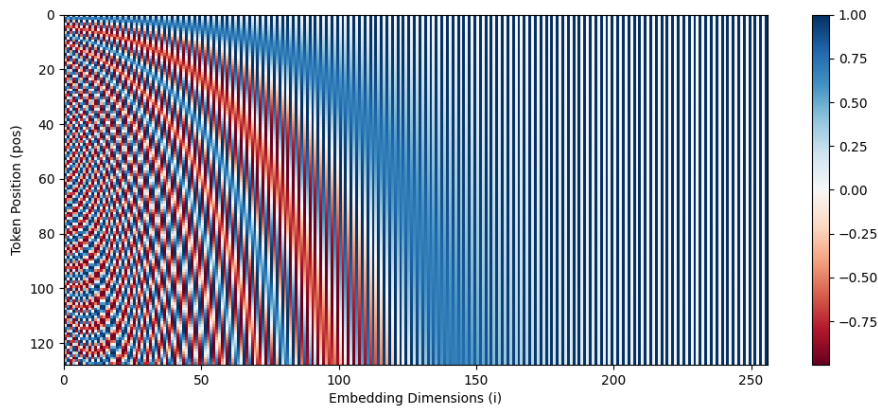


Figure 2.4: A visualization of positional encodings. Token Position & Embedding Dimensions are arbitrarily chosen for clear visualization.

this consequence, CANINE is built with a down-sampling convolutional layer before the transformer stack and a corresponding up-sampling convolutional layer after. This results in a model which the authors argue is computationally efficient while maintaining character-level tokenization.

2.7.1 Sequence Classification

When using CANINE for sequence classification, the hidden state from the last layer in the transformer stack representing the “beginning of sentence” token (often designated as **[CLS]**) is extracted as shown in Figure 2.5. The extracted hidden state is given to an LM-head classifier with a task-specific number of output nodes. For our case, this is a binary classification task of labeling a sequence as containing a date expression or not, and as such only 1 output node is required.

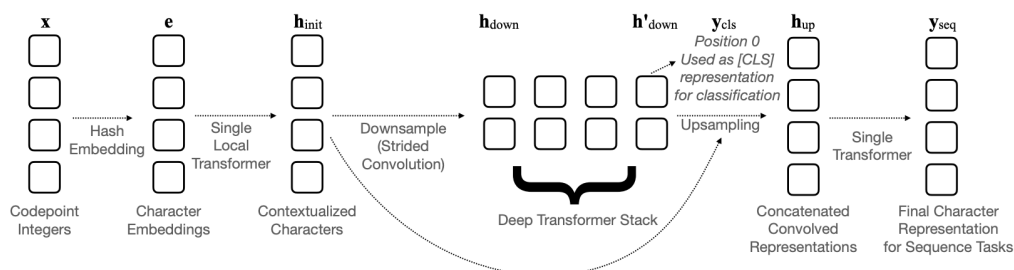


Figure 2.5: neural architecture of CANINE. y_{cls} shows the state which is extracted when using CANINE for sequence classification. After Clark et al. (2021).

2.7.2 Token Classification

In the case of token classification, CANINE works similarly to when it's used for sequence classification. The difference is that the representation of the [CLS] token is not extracted but rather the representation for all tokens in the \mathbf{y}_{seq} layer. The representation from these tokens is then similarly given to a classifier which outputs a prediction for each token. As we used a BIO-scheme for classifying if a token belongs to a date expression or not, this results in three possible output labels per token.

2.8 T5: Text-to-Text Transfer Transformer

This model was introduced by Raffel et al. (2019) at Google. It is built as a sequence-to-sequence, encoder-decoder transformer. The model takes a text string input and generates a text string output as the name suggests. Figure 2.6 shows how the model is built to work as a unifying framework that can be applied to solve many different NLP tasks.

The authors proposed five versions of the model with varying numbers of parameters ranging from 60 million to 11 billion. For their largest model, they reported state-of-the-art performance on multiple tasks including SQuAD, superGLUE, and MultiRC, demonstrating strong capabilities of general language understanding. I intend to use this transformer as a conventional sequence-to-sequence model, to generate the ISO format for a date expression given an input string containing an arbitrarily formatted date expression.

T5 is pre-trained on “Colossal Clean Crawled Corpus” (C4) which is a data set consisting of 750 gigabytes of English text data based on Common Crawl. It was trained on a denoising objective, which means that for an input sequence a number of tokens are masked and the objective for the model is to predict the masked tokens; see Figure 2.7. This was done by randomly replacing 15% of tokens in an input sequence with a specified placeholder token and letting the model predict the tokens that were removed. Note that the authors of T5 call this denoising but it is the same as masked language modeling, described in section 2.9.

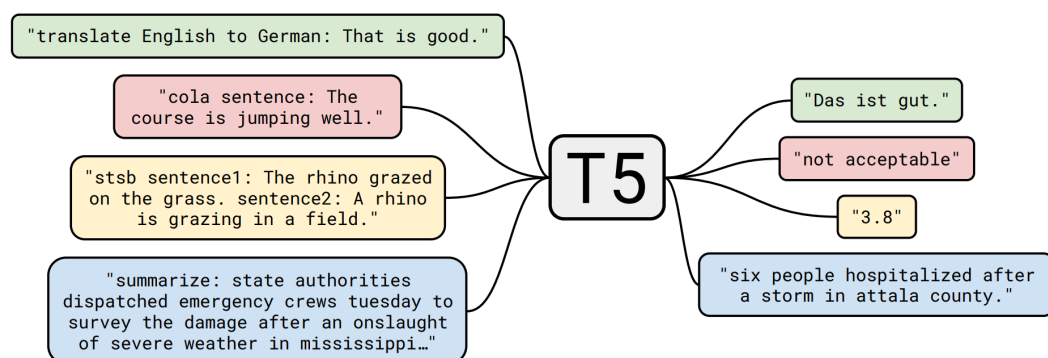


Figure 2.6: An illustration of how T5 is intended to work as a unifying framework for different NLP tasks.

There are also two subsequent models based on the original T5 architecture which are of interest to this project. *mT5* (Xue et al., 2020) and *byT5* (Xue et al., 2022).

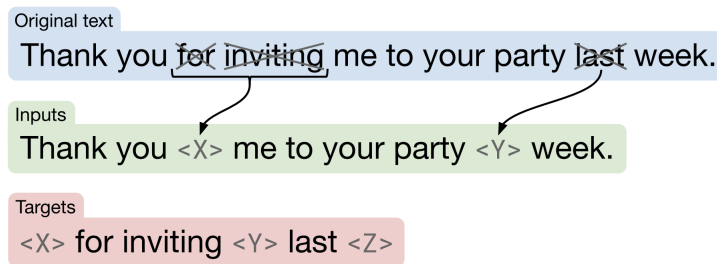


Figure 2.7: Illustration of the pre-training objective for training T5 model family. After Raffel et al. (2019).

2.8.1 mT5

About one year after T5 was created, another team at Google Research created mT5 (Xue et al., 2020), a multilingual counterpart. This family of models is built to be similar in architecture to the original T5 and with the same goal in mind of creating a unifying model for most NLP tasks. Like the original T5, they made 5 versions with different numbers of parameters.

To train mT5, the authors used a variation of the C4 dataset named mC4. This dataset contains text in 101 different languages. Unsurprisingly the most frequent language in the mC4 dataset is English. To mitigate the fact that some languages are much more common in the dataset than others, the less common languages are oversampled and vice versa more common languages are undersampled. Figure 2.8 shows the probability of sampling different languages depending on their frequency in the mC4 dataset. The probability of sampling a language is proportional to the number of examples in that language to the power of α , i.e. $P(L) \propto L^\alpha$ where L is the number of examples for a given language.

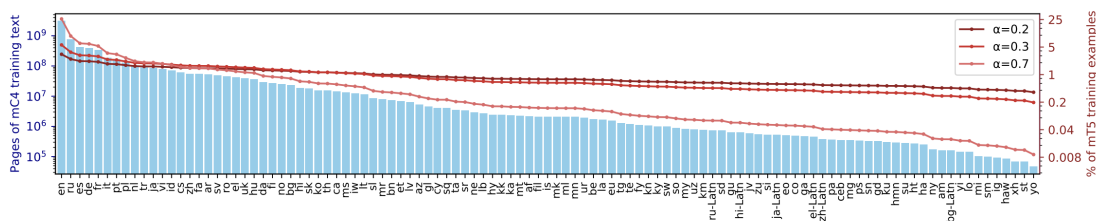


Figure 2.8: Language distribution of the mC4 dataset. the red lines show the sampling rate of over the languages while the blue histogram shows the occurrence of documents from each respective language. For the published models, the authors chose $\alpha = 0.3$ for the sampling rate. Image from Xue et al. (2020).

Because the model is multilingual, the vocabulary sizes of these models are larger, 250,000 for mT5 compared to 32,000 for the original T5. As a consequence of this, the number of parameters for the mT5 models is also larger, ranging from 300 million to 13 billion.

2.8.2 byT5

The idea of this model is to address issues with subword tokenization which it does by converting all input text to its byte-level representation before feeding it into the model. One trade-off with this approach is that the byte-level sequence length is longer than the subword tokenized sequence length, resulting in the input lengths being on average 4 times that of mT5 depending on the language. Because of this, the authors expect the model to perform well on shorter sequences, which generally fits our task well in this thesis.

Another consequence of the byte-level inputs, which the authors noted is that the model required a larger encoder, resulting in an overall larger model. Similarly to T5 and mT5, they released 5 models ranging from 300 million to 13 billion parameters. Note that this range of model sizes is the same for byT5 as for mT5. While byT5 has a deeper encoder stack compared to mT5 the byte-level vocabulary is much smaller than the subword vocabulary, which compensates and saves a lot of parameters in the input embeddings. The family of byT5 models is pre-trained on the mC4 dataset like mT5, also making them multilingual.

2.9 Transfer Learning

The concept of transfer learning in machine learning can loosely be expressed as a model that is trained on some, often general, task and then reused and trained for another more specific. This is referred to as pre-training and fine-tuning.

This is widely applied for transformer models in natural language processing (Wolf et al., 2019) and as research is showing, scaling up the size of models generally gives improved performance for NLP models (Raffel et al., 2019). Hence a reason for using transfer learning is that training a very large model from scratch can be computationally expensive and using a pre-trained model often saves time and resources.

A more formal definition of transfer learning is given by Zhuang et al. (2019) in the paper "A Comprehensive Survey on Transfer Learning which states:

Definition 1. *Given some/an observation(s) corresponding to $\mathbf{m}^S \in \mathbb{N}^+$ source domain(s) and task(s) (i.e., $\{(\mathcal{D}_{S_i}, \mathcal{T}_{S_i}) | i = 1, \dots, \mathbf{m}^S\}$) and some/an observation(s) about $\mathbf{m}^T \in \mathbb{N}^+$ target domain(s) and task(s) (i.e., $\{(\mathcal{D}_{T_j}, \mathcal{T}_{T_j}) | j = 1, \dots, \mathbf{m}^T\}$), transfer learning utilizes the knowledge implied in the source domain(s) to improve the performance of the learned decision functions $f^{T_j} (j = 1, \dots, \mathbf{m}^T)$ on the target domain(s).*

A couple of general tasks for pre-training a model for text processing include *masked language modeling* (MLM) or auto-regressive generation. These tasks are unsupervised and can therefore be trained on very large datasets.

- **Masked language modeling:** From a tokenized input text, a certain percentage of tokens are masked with a placeholder token. The model is tasked with predicting what the specific tokens that were removed were. This task is sometimes also referred to as 'denoising'.
- **Auto-regressive generation:** In an input text, a continuous subsequence is given to the model. The model is tasked with predicting which token comes after the given input subsequence in the text.

In this thesis, there are two somewhat different ways I am going to utilize transfer learning. The first way is to load an encoder model and further feed the hidden states output to a *LM-head* (linear model head) which I train for our specific task. In this setup, I can either choose to train the parameters of the base encoder model, i.e. fine-tune it, or save the computational cost and only train the LM-head on top of the transformer stack. The second way I am going to utilize transfer learning is on encoder-decoder models where no layers will be added to the model but rather fine-tune the pre-trained transformer stack to our specific task.

2.10 Model Parallelism

As shortly mentioned above it has been shown that scaling up model size tends to give better performance for machine learning models in general and not least for language models. As a natural consequence of this fact, together with computing power evolving according to Moore's Law (Moore, 1965), machine learning models have generally grown larger and larger in size over the years. Especially in recent years, the number of parameters in language models has grown significantly. This was demonstrated quantitatively by Villalobos et al. (2022) in the article *Machine Learning Model Sizes and the Parameter Gap*, where they analyze the trend in model size of different machine learning fields over the years.

With the "Parameter Gap" the authors refer to that there seem to be fewer models trained in the range between 20 and 70 billion parameters than both the ranges below and above this interval, which is illustrated in Figure 2.9. They hypothesize that this gap in the number of parameters is due to the fact that training models beyond 20 Billion parameters require multiple techniques of model parallelism to train effectively.

In this thesis, I am limited in model size by the fact that I want to avoid model parallelism as much as possible due to time constraints during implementation and the fact that it's not desirable to use such a powerful computer during inference for *The Company*.

Even though the models that will be used are significantly smaller than 20 billion parameters the GPUs I have available will run out of memory before reaching this order of magnitude in parameters. Among other things, during training the computer also has to keep track of optimizer states and gradients beyond just the model parameters, which saturates the available memory.

2.11 CRISP-DM

In this thesis, I have access to a dataset and a regex module from *The Company* which is intended to be used when constructing a machine-learning system to extract date expressions from receipts. How this system should be designed is not entirely decided beforehand, but rather is to be determined during the process. CRISP-DM (Wirth and Hipp, 2000) (Cross-Industry Standard Process for Data Mining) is a framework that can be utilized for this.

Figure 2.10 shows the reference model presented by Wirth and Hipp (2000). This model highlights the most important dependencies between the different stages of a data mining project. The action taken at each stage of this figure can also be dependent on the previous step. This means that parts of the methodology for the project can be adapted during the

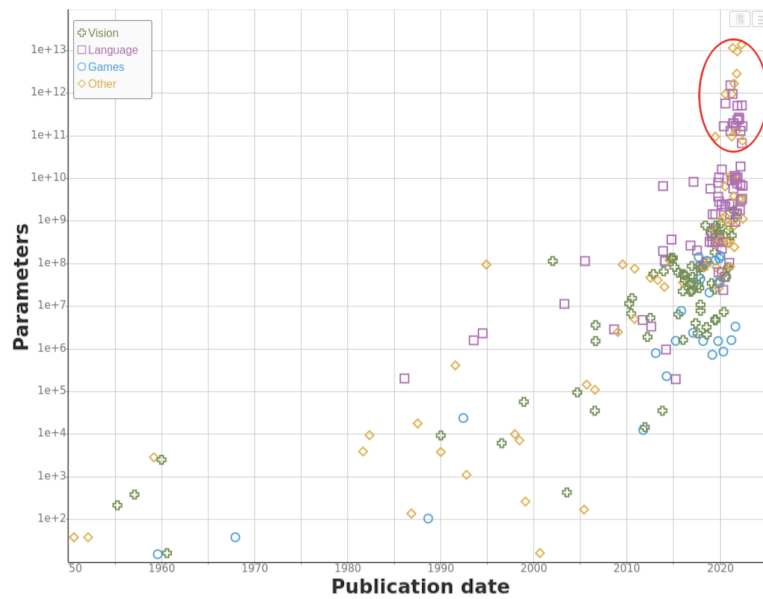


Figure 2.9: Number of parameters in published machine learning models for different fields over the years. Illustrating the hypothesized parameter gap. Image from Villalobos et al. (2022).

process and informed by partial results.

The model proposed in the paper is represented in Figure 2.10. The authors emphasize that the sequence of each part in the figure is not strict but provides a general guideline for Data Mining. The focus for this thesis will mainly lie in the phases of *Data Understanding*, *Data Preparation*, *Modelling*, & *Evaluation* for building and improving the machine learning system. The *Business Understanding & Deployment* part of the model are outside the scope of this thesis and will be considered up to *The Company*.

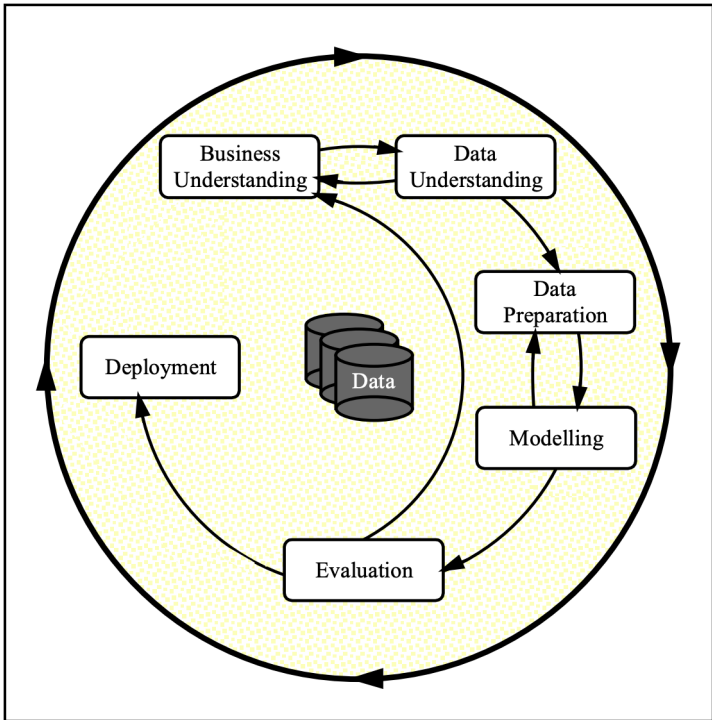


Figure 2.10: CRISP-DM reference model for Data Mining. Image from Wirth and Hipp (2000)

Chapter 3

Previous Work

There are several examples of attempts and solutions to identify dates from natural text on arbitrary formats. For instance, HeidelTime (Strötgen and Gertz, 2010) is a rule-based system that tags temporal expressions in natural text according to the TIMEX3 annotation standard (Boguraev et al., 2002). In their 2010 paper, the authors discuss the alternatives of either building a rule-based or a machine-learning system. However, they decided to go with the rule-based approach due to the relative simplicity of formatting the values of date expressions from rule-based extraction compared to machine learning.

Since 2010, there have been huge advancements in the field of machine learning. Vaswani et al. (2017) introduced the transformer in the paper “Attention is all you need”. It was initially proposed as a machine-learning architecture for translation. Due to the success of the transformer architecture, it was quickly adopted for various other *natural language processing* (NLP) tasks. Later the transformer has also seen successes in other domains such as computer vision where transformer-based models have achieved *state of the art* (SOTA) results in tasks such as *optical character recognition* (OCR) for example Li et al. (2021). In this thesis, however, I will work with transformers on text input to find date expressions on the text data from receipts.

“BERT got a Date” by Almasian et al. (2021) is a recent attempt for identifying dates in natural text which utilize a transformer. In this work, they used the encoder part of the architecture and attempted to solve the problem with two different approaches. In the first approach, they used token classification to classify each token of an input sequence to either belong to a TIME, SET, DURATION, or DATE with a standard BIO-scheme. For example, in an input sequence containing a date expression the BIO-scheme could tag it as follows:

Remember,	remember	the	fifth	of	November
O	O	O	B-DATE	I-DATE	I-DATE

In the second approach, they annotated sequences with the TIMEX3 format. This was performed by viewing the problem as sequence-to-sequence text generation. Annotating

the same example string as above with the TIMEX3 format transforms it into "Remember remember the <TIMEX3 type="date">fifth of November</TIMEX3>". They achieved their best performance with the sequence-to-sequence, text generation solution. This work partially solves our problem with two different approaches, however, we are not only interested in identifying dates in this thesis but also extract the exact date value.

The model in this work was based on the successful transformer BERT (Devlin et al., 2019) which reported many SOTA results. Although it did not report SOTA results for the CoNLL-2003 Shared Task: Language-Independent Named Entity Recognition (Tjong Kim Sang and De Meulder, 2003), it came close. It has also later been shown that the transformer architecture generally performs well on Fine-Grained Named Entity Recognition (FN-NER) (Lothritz et al., 2020). Following these results I will also use the transformer architecture to approach the problem in a similar manner.

Seker and Ahn (2022) describes another study entitled "A generalized framework for recognition of expiration dates on product packages using fully convolutional networks" that not only identifies dates but also extracts their exact values on multiple formats from images. In their approach, they use image data as input to train a three-step pipeline of convolutional neural networks (CNNs) to identify expiration dates on product packages. The three steps were as follows:

1. The first network identifies the region of the image which contains the mention of an expiration date,
2. The second network identifies which sub-parts of the identified region belongs to the mention of a year, month & day, and
3. The last network identifies the values of the three sub-parts for a year, month & day respectively.

Unlike this work, I will use text data directly instead of the corresponding image, I will also rely on the transformer architecture rather than CNNs. However, similar to this work I draw inspiration from the possibility of splitting the task into sub-tasks for a series of machine learning models.

Chapter 4

Dataset

As of January 2023, *The Company* has a dataset of 852,333 receipts. Each receipt is initially obtained as a camera image or print screen which is sent to Google (2023) from which a `.json`-file is returned with OCR elements. Each element has a position on the receipt defining a bounding box. These bounding boxes are categorized into *word*, *paragraph*, *block* & *page* elements. Figure 4.1 shows an illustration of two receipts where the bounding boxes of *word*, *paragraph* & *block* elements are drawn. The elements are structured such that each *page* element encloses a number of *block* elements which in turn encloses a number of *paragraph* elements which lastly encloses a number of *word* elements.

Each receipt has also been manually annotated with six labels and their corresponding values: [*Total Amount*, *VAT*, *Purchase Date*, *Country*, *Currency*, *Category*]. In some cases, one or more values on a receipt may be missing and as such the receipt is only partially labeled. The three latter labels are used for classifying receipts as a whole while the former three are used to classify a bounding box on the receipt containing the labeled data. In this thesis, we will focus on the labels corresponding to the classification of bounding boxes and the *Purchase Date*-label specifically.

4.1 Exploratory Data Analysis

Currently *The Company* uses regex modules to extract features from the OCR results of each receipt to find candidates for *Amount*, *VAT*, and *Date*. In this section, I will use the regex module which identifies date expressions to evaluate how it performs on the dataset from *The Company* for finding the labeled purchase date.

4.1.1 Purchase Date

From the dataset, there are 57,554 receipts where the label for a purchase date is missing. These receipts were excluded from the evaluation in this section. I used the regex code that

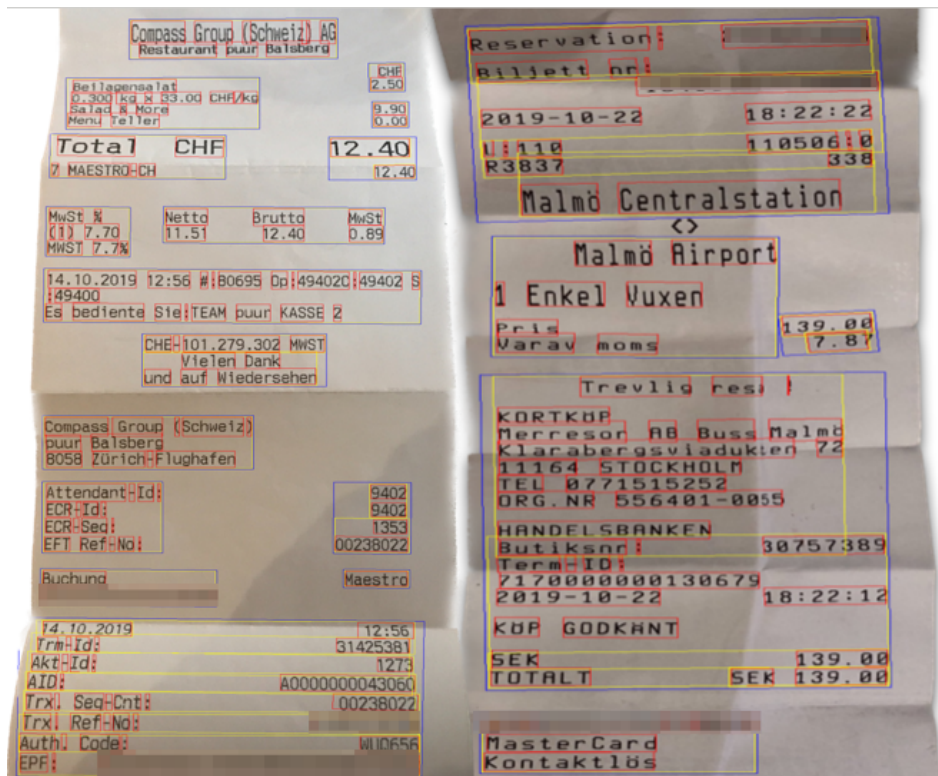


Figure 4.1: An example image of the dataset where the bounding boxes containing ocr elements are illustrated. The Blue boxes are **block elements**, the yellow boxes are **paragraph elements** and the red boxes are **word elements**. Any personal information, or information that can be used to extract personal information, has been redacted.

The Company currently has in production to calculate whether or not any OCR element on a receipt was parsed to a date matching that of the labeled purchase date. For the $852,333 - 57,554 = 794,779$ tested receipts, I found that for 63,038 receipts the regex code could not find a date matching the manually labeled purchase date resulting in a failure rate of 7.93%. I further inspected the receipts where no correct date could be found, to get an understanding of where the failure was caused. These failures could generally be categorized into three different types:

- The manually produced label for the receipt was wrong.
- OCR misses. A few examples which were identified were: “5” in one image was wrongly identified as “7” and in another image “11” was identified as “M”.
- The date is written in a format that is not recognized by the regex module.

4.1.2 Total Amount

From the dataset, there are 25,288 receipts where the label for the total amount is missing. These receipts were excluded from the evaluation in this section. Again I used the regex that *The Company* currently has in production to calculate whether or not any OCR element on

a receipt was parsed to a number that could be interpreted as an amount. In the dataset, $852,333 - 25,288 = 827,045$ receipts had a labeled total amount. Among them, I found that for 8,783 receipts, the regex output did not match the manually labeled total amount. This resulted in a failure rate of 1.06%. As the failure rate of the regex module is significantly lower for the total amount compared to the purchase date, I did not investigate it further.

4.1.3 VAT

Lastly, I did the same procedure for the VAT label for which 382,402 out of 852,333 receipts were missing a labeled VAT amount. For the remaining $852,333 - 382,402 = 469,931$ receipts which have a labeled VAT amount, I found that the regex module did not find the correctly labeled VAT amount on 17,380 receipts, resulting in a failure rate of 3.72%.

4.2 Datasets From the Regex Module

Each of these three regex modules serves two purposes. They both classify a bounding box to potentially contain the labeled *Total Amount*, *VAT*, or *Purchase Date* and then also extracts/parses the value.

In this thesis, I aim to replace the regex module that identifies date expressions with a transformer-based machine learning system. To this goal, I need to process the dataset provided by *The Company* for the machine learning models. This processing will consist of labeling sequence of text from the receipts in the dataset for both sequence and token classification. Performing this will, in large part, utilize the existing regex module.

An important feature of the regex module is that it attempts to match *word* elements as date expressions from a set of defined regular expression patterns. However, there is also some logic built into the module which concatenates up to four *word* elements within a text paragraph to match as a date expression if none was identified from individual *word* elements. Ignoring this, I will refer to the regex module as identifying date expressions from *word* elements in the rest of this report for the sake of simplicity.

Chapter 5

Method

In this chapter, I outline how I built a machine-learning system to extract date expressions and format them according to the ISO standard.

First I process the dataset provided by *The Company* for the machine learning models as well as construct a purely synthetic dataset. Then I outline the two possible architectures for my system which will be considered. I then performed some fine-tuning tests on different versions of CANINE & T5 to make an informed guess as to which architecture to implement. Lastly, When my system is implemented I inspect the results and produce some attempts at improving it.

All pre-trained transformer models are downloaded from huggingface transformers library as their PyTorch version. For fine-tuning, I used a virtual machine on Amazon Web Services, EC2. The virtual machine had 4 Tesla T4 GPUs, resulting in a total of 16x4GB VRAM.

The T5 model families consist of 5 model sizes in each family. These are named as *small*, *base*, *large*, *xl* and *xxl*. I used *base*, *base* and *small* for T5, mT5 and byT5 respectively.

5.1 Processing & Building Dataset to Fine-Tune machine Learning Models

To train the machine learning models I needed a dataset labeled for their respective tasks. To this end, I both processed the receipts provided by *The Company* and also produced a synthetic dataset. Both these datasets are labeled for sequence and token classification.

5.1.1 Date Classification Dataset

I constructed this dataset by using the regex module to label where date expressions were found on the OCR response of receipts. For this dataset, each individual token was labeled as

either belonging to a date expression or not. Each sequence was also labeled for sequence classification i.e. the sequence either contains a date expression or not, with the ISO-formatted date as the label.

The dataset was constructed by the following procedure where I applied the regex module to sequences of *word* elements in the receipt:

1. I gathered a set of positively labeled examples of date expressions that the regex module had identified.
2. For each date expression identified by the regex module, I concatenated a left and right context of up to a maximum of 64 characters in total. For this sequence, the regex module produced the ISO-formatted date as the label and each token from the identified date expression was also individually classified as belonging to a date.

For negatively labeled examples which do not contain a date expression:

1. I removed the *word* elements which the regex module identified as date expressions and split the remaining OCR response on these indices to a list.
2. Sequences were constructed from the list by randomly selecting a *word* element and padding to a max length of 64 characters or until the list of *word* elements was empty. For these sequences, all tokens were individually labeled as *O* for 'outside'.

I now had a dataset of 3,986,293 sequences. From these 2,165,992 were positively labeled and contained a date expression and 1,820,301 were negatively labeled. As plotted in Figure 5.1, there are some differences in length distributions between the positively and negatively labeled examples due to how the dataset was built.

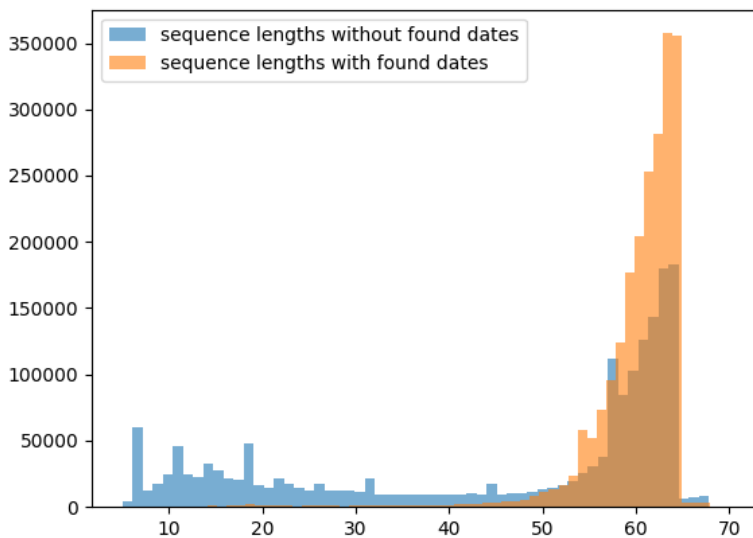


Figure 5.1: Histogram showing the distribution of sequence lengths for both sequences containing dates and those that do not.

For this dataset, I also inspected the distribution of lengths for the date expressions within the sequences. This is illustrated in Figure 5.2. It is worth noting that these numbers are influenced by how the regular expression patterns are constructed and may not be an exact correspondence to how long the 'actual'/'true' date expressions in the dataset are. Nonetheless, I will assume that this is still relatively representative even if not exactly correct.

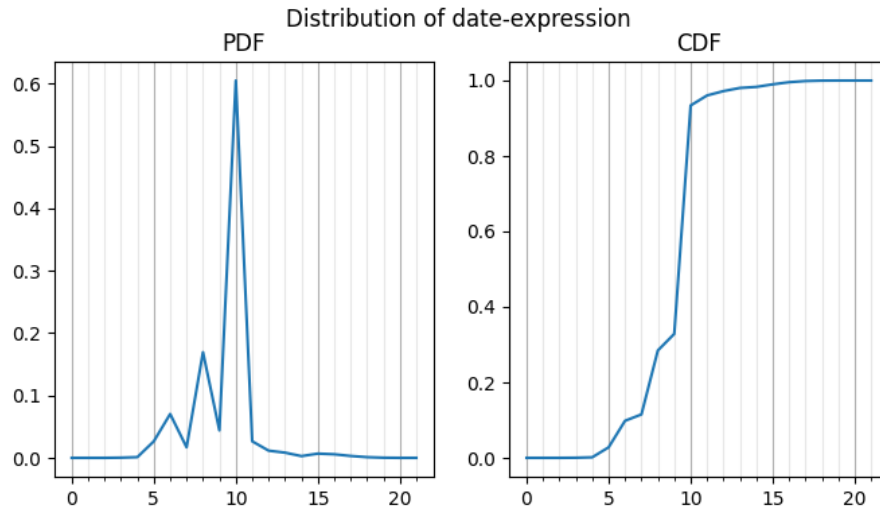


Figure 5.2: The lengths, in the number of characters, of date expressions that are identified by the regex module. PDF is the probability distribution of a date expression having a specific length. Similarly, CDF represents the cumulative distribution.

I saved this dataset to roughly 1,100 `.csv`-files containing approximately 4,000 lines each. When storing this dataset I separated the part of the sequence which was padded from the original *word* element. This way I had access to both the full sequences and the *word* element which the sequence was constructed around, for each example. One of these files was removed from the dataset to be manually reviewed. The removed file contained 4,571 sequences from which 1,003 examples were manually inspected and labeled. Table 5.1 shows the results of this manual inspection. This gives an estimate of the regex modules' performance on finding date expressions, and will also be used as a testset when evaluating the machine learning models.

Date?	precision	recall	F1-score
Yes	0.94	1	0.97
No	1	0.94	0.97

Table 5.1: Results for 1003 manually labeled sequences to evaluate the performance of the regex module.

When inspecting the sequences I discovered some systematic problems with the regex module. A clear problem was that the module often recognized addresses as dates. This was due to the regex module matching months by either a number from 1 to 12 or by looking in a file of month names and abbreviations. In Italian January is called *gennaio* and abbreviated as '*gen*' which caused the module to match all Swedish addresses ending in the suffix '*vägen*'

as months. If the street name is also accompanied by a street number within the range of 1 to 31 it caused the regex to identify a date. For instance, the address 'Möllevångsvägen 31 222 40 Lund' is matched by the module as the date 2023-01-31. The year can be inferred by the module from the metadata of when the receipt image is uploaded.

5.1.2 Generating a Synthetic Dataset

To create a synthetic dataset I primarily used the library Faker (Pua, 2022). I used the following set of generating functions from the library to synthetically generate data:

- `address`
- `bban`
- `ean`
- `administrative_unit`
- `currency`
- `url`
- `ascii_company_email`
- `company_name`
- `ascii_email`
- `phone_number`
- `words`

The functions were selected by items that I, provided with advice from *The Company*, deemed most likely to find on a receipt.

Faker can generate data with different *locale* settings. Locale is a combination of language and territory, for example, the default locale in Faker is "en_US" for English + United States. However, the library is not fully implemented for all available locales. For example, it generates *lorem ipsum* text when the words function from a specified locale is not implemented.

The procedure for generating synthetic data was as follows:

1. The generator was instantiated with a randomly selected locale.
2. To generate an example I started with a 50% chance to select a function from Faker called `date_between` and label the sequence as a positive example. Otherwise, I selected a random function from the listed generator functions above and labeled the sequence as a negative example.
3. For the positive examples the `date_between` returns a python `datetime`-object. To format this string to an arbitrary format I used the package Babel (Koskela et al., 2023) and the function `format_date`. With this, we could select custom formats or use Babel's default formats for writing date expressions that are locale-specific.
4. To extend the initially selected element to a full sequence I gave the generator a 40% chance to pad the generated sequence with a 50/50% chance of pre/post-padding, followed by another 30%, 20% & 10% chance to pad the sequence again resulting in generated sequences of 1 to 5 elements.

As the real data also have access to a reference date referring to when the receipt image was uploaded, I generated another date with a random offset between `[0, 90]` days after the positive labeled examples, and for the negative examples, the reference date was simply randomly sampled. The selection of this offset was chosen as we assume that most receipts are uploaded to the system within 90 days of the purchase, however, this was not investigated but

Generated sequence	label date	reference date	locale
paezguillermo@gomez.net 3 de marzo de 2022 alias	2022-03-03	2022-04-19	es_CO
Mart 11 https://www.koruturk.org/	2024-03-11	2022-03-12	tr_TR
soluta dignissimos New York	None	2022-05-10	sq_AL
7522758688093	None	2019-08-08	or_IN
lazura22@ilikov.info YMIU01400468967145	None	2032-09-21	bg_BG
fsimmons@blair-pope.info Τετάρτη 4 Αυγούστου 2032	2032-08-04	2032-09-04	e1_CY
5.10.2019 12637 Şinasi Points Türkburgh, IA 64728	2019-10-05	2019-10-19	tr_TR
7239,75Comorian franc 6334088601649	None	2033-01-20	ne_NP
arbejder specialist nylig	None	2031-07-27	da_DK
30 maijs 5703934261978	2037-05-30	2037-07-06	lv_LV

Table 5.2: Instances of generated synthetic data with corresponding labels. The synthetic data was also labeled for token classification which is not shown in this table.

rather guessed by *The Company* from experience. Table 5.2 shows a few examples of generated data points.

Similarly to the dataset of sequences built from the OCR elements I generated a synthetic data batch and plotted a histogram to get an idea of how the length distribution of the data points compared to the other datasets. In Figure 5.3 we can see that the sequences in this dataset have different length distributions to the one in Figure 5.1.

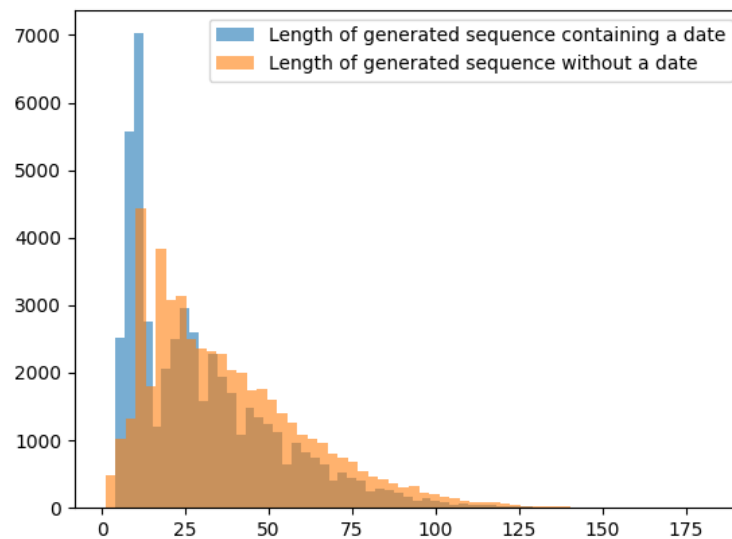


Figure 5.3: Histogram plot of the length distributions of sequences for the generated synthetic dataset.

5.2 Potential System Architecture

I considered two potential architectures for the system. Both the date classification dataset from Section 5.1.1 and the synthetic dataset from Section 5.1.2 can produce either a single

OCR element/generative function or a complete sequence comprised of multiple OCR elements/generative functions. From this fact, I can either apply:

- **Sequence classification** followed by a sequence-to-sequence model generating the ISO formatted date from the full sequence if a date expression was found.
- **Token classification** followed by a sequence-to-sequence model generating the date from the tokens classified as a date expression from an input sequence.

Consider the example of *I'm typing this on the 13th of March 2023*. The first potential architecture would ideally identify this as containing a date and as such send the whole sequence to the subsequent sequence-to-sequence model in hopes that it will generate the target sequence "2023-03-13". The second potential architecture would ideally, from the same input sequence, identify *13th of March 2023* as the part of the sequence which describes a date and hence send this sub-sequence to the subsequent sequence-to-sequence model in hopes that it will generate "2023-03-13".

Figure 5.4 illustrates the two potential architectures in broad terms. Note that for the token classification alternative, there is a possibility of including a post-processing step. Consider the previously mentioned example of *I'm typing this on the 13th of March 2023*. The token classification model might miss-classify a single character and label *13th of & March 2023* as two separate date expressions. A post-processing step could possibly correct errors such as this by implementing some relatively simple logic.

In both cases, the different versions of CANINE and T5 will be optimized separately to then be applied in sequence for the full system. Beyond this, I also have a couple of different datasets which can be applied differently to the separate transformer models. This results in a large space of possibilities where I will have to make design choices when constructing my system.

Using CRISP-DM as inspiration, I have a framework for selecting a design for the complete system which I aim to build. This framework can also be applied after the system is designed in attempts at improving it. Potential improvements can be performed by inspecting examples where the system fails to extract the correct date expression, and then adjusting whichever part of the system seems to cause the systematic error.

5.3 Sequence Classification With CANINE

As our dataset is multilingual and we preferred a character-level model, we found the CANINE transformer by Clark et al. (2021) from Google Research to fit these preferences.

I used CANINE as described in Section 2.7.1 for classifying if a sequence of characters contained a date or not. The dataset used here was the one produced in Section 5.1.1. The transformer parameters themselves were not fine-tuned but rather the LM-head on top of the transformer was trained to classify the input sequences.

Two classifiers were trained here, one when only using the *word* element which the regex module had classified as a date, and the other when using the full text sequences. For these experiments, I used the CANINE-S version optimized on subword loss.

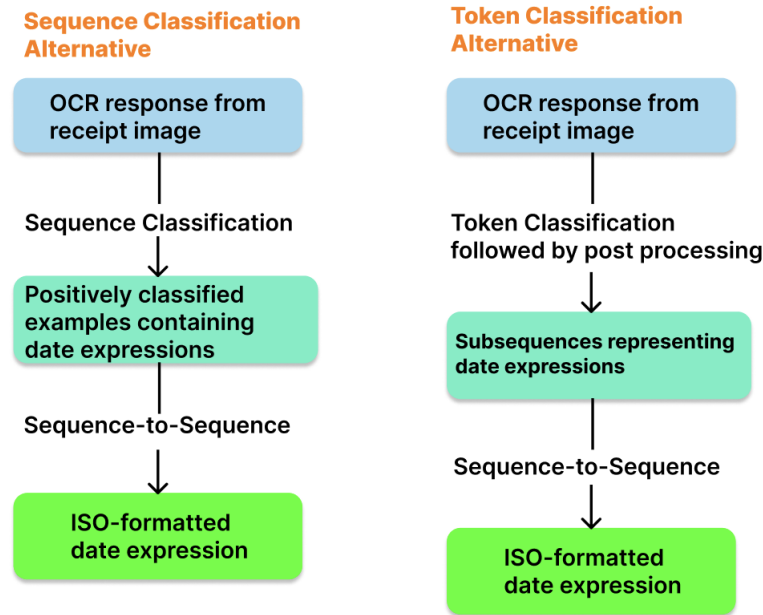


Figure 5.4: Flowchart illustration, outlining the two potential architectures I will consider for building the complete system.

5.4 Token Classification With CANINE

For these experiments, I used the datasets from Section 5.1.1 and synthetic training data from Section 5.1.2. I used CANINE for token classification as described in Section 2.7.2. For this task, I trained an LM-head, as well as fine-tuning the parameters of the transformer stack at the same time. I also used both versions of the base model i.e. CANINE-S and CANINE-C for this task. The LM-head had three output labels following the BIO-scheme:

- O: all tokens not belonging to a date expression
- B-DATE: token signifying the start of a date expression
- I-DATE: tokens inside a date expression

These labels were used to find subsequences of tokens in the input sequence which the model identified as belonging to a date expression.

5.5 Formatting Dates by Fine-Tuning T5

As an initial test T5-base was fine-tuned on the dataset from Section 5.1.1. I used 100,000 examples for training and 10,000 for validation. The task was viewed as a sequence-to-sequence problem. The input sequence was the *word* elements which contained a date on some arbitrary format and the target sequence was the corresponding ISO-formatted date. I also used the full sequences of the dataset and not only the *word* elements which contained the date expression as input with the same target sequence.

Given a *word* element containing a date expression in an arbitrary format, the model attempts to output a string of the corresponding ISO-formatted date. i.e. in the format of YYYY-MM-DD. It can be very hard for the model to distinguish between two dates if switching the numbers representing month and day also produces a valid date. for example, if a *word* element contains the string *Date: 03.11.2022*, it is ambiguous whether it aims at *11th March* or *3rd November* of 2022. This can depend on which country a receipt is from, where conventions of date formats may differ. Because of this reason, I measured the accuracy of the model on:

- the year (YYYY) part of the date matched the labeled year.
- the month (MM) part of the date matched the labeled month.
- the day (DD) part of the date matched the labeled day.
- the full date of the output matched the labeled ISO-date.
- the full date of the output matched the labeled ISO-date after a post-processing step where the day and month are switched such that both YYYY-MM-DD & YYYY-DD-MM are tested.

For the dataset from Section 5.1.1 the corresponding uploaded image to the software system contains a *reference_date* from when it was received and also a *language_code* for which language Googles OCR response suspects the text to be in. For the synthetic data, the reference date was produced as described in Section 5.1.2, and the language code was extracted from the locale attribute of the instantiated Faker-object. I used this data to enhance the input sequences to T5, giving it further context. The input sequences now contained a prefix which, for example, could look as the following:

```
[reference_date = '2021-01-20', language_code = 'en']
```

5.5.1 mT5

The same fine-tuning was performed on mT5-base as for T5-base where the input sequences were enhanced with *language_code* & *reference_date*. The number of parameters for T5-base & mT5-base are 220 & 580 million respectively. Because of this difference, I could not load the full mT5-base into a single GPU as previously done with T5-base. For mT5-base this was solved relatively easily with *model parallelism*. The model was split into 4, roughly, equally large parts and distributed over the 4 available GPUs.

5.5.2 byT5

The number of parameters in the respective model sizes of byT5 are the same as for mT5. However, with byT5 the same solution to perform *model parallelism* was not possible due to implementation differences on huggingface. Because of time constraints, I chose to use byT5-small instead of byT5-base which is 300 million parameters compared to 580 million. Hence I could fit the full model into a single GPU.

First I performed the same fine-tuning procedure for byT5 as for mT5. Then I also investigated the impact of the synthetic data during training. This was performed by inserting different fractions of synthetic data shuffled into the training dataset from Section 5.1.1.

5.6 Complete Pipeline

Based on the results of Tables 6.11, 6.12 & 6.13 the formatting of an input sequence seems to be significantly easier for a model when the input sequence is shorter and contains less ‘noise’/‘context’ around the characters which represents a date expression. In practice the post-processing step of trying to switch the digits expressing the day and month is always tried, hence the post-processed metric is the one I value most here. Because of this and also the first results of token classification with CANINE from Table 6.3, I decided that going forward I would use the “Token Classification Alternative” architecture of the full system, illustrated in Figure 5.4.

To get a system that could extract ISO-formatted dates from the text content of a receipt I now used CANINE-S for token classification and byT5-small. CANINE first identified where in a sequence of characters a date was found and then byT5 transformed the date expression to our preferred ISO format. The input sequences to CANINE were segmented by splitting the full OCR-response of the receipts on *paragraph* level. Because the accuracy on chunks of the CANINE model was not optimal, see Table 6.5, a post-processing step was introduced between CANINE for token classification and byT5. The post-processing procedure was initially as follows:

- For a sequence of characters identified as a date, which is at most 20 characters long in total. Check if another identified date is less than 4 characters away.
- If this other sequence of characters is identified as a date with less than 4 characters in total: merge the two sequences of dates to a single date, including the characters between them classified as ‘O’.
- recursively repeat pattern until no more merges of identified dates occur.

The numbers in this algorithm were chosen from manual inspection of results from CANINE for token classification but without any quantitative analysis.

Figure 5.5 shows a full representation of all the steps of the system from the uploaded receipt image to the extracted dates found on the text contents of the image.

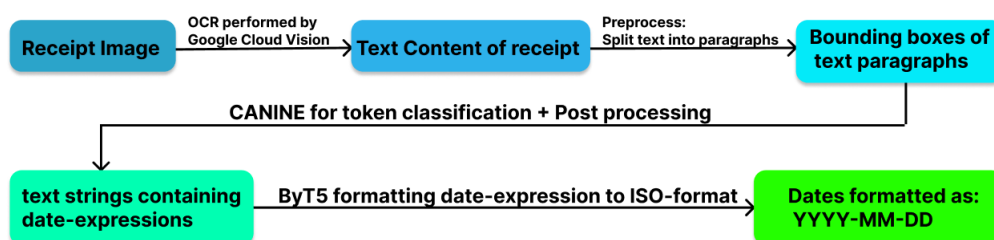


Figure 5.5: Flowchart representation of the components of the built system.

Figure 5.6 shows an example of how the final system worked. In the image on the right side, we can see that the system extracted a wrong date from a very unusual format. The receipt from this example is Dutch and the wrongly extracted date is from the text line “AS-tijd 29.01. 08:59”. This date format excludes a representation for the year and ‘confuses’ CANINE,

which misses the first character of the day and includes a character from the representation of the time of day.

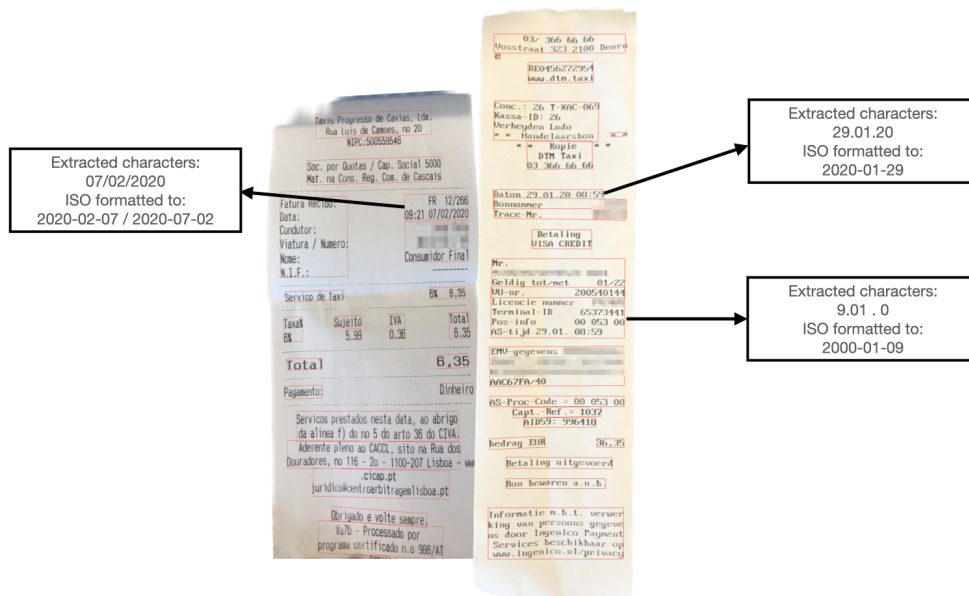


Figure 5.6: A couple of receipts where the dates are extracted using the built transformer system. Any personal information, or information that can be used to extract personal information, has been redacted.

5.7 Attempted Improvements for the Complete Pipeline

I inspected the failures of the transformer system for systematic errors which could possibly be solved by changing the post-processing, synthetic data generation, etc. The results of how these attempts affected the accuracy of the system can be found in Table 6.15

5.7.1 Data-Augmentation: "date"

I found that CANINE, somewhat counterintuitively, ignore dates that are found after the word "date" is explicitly written out. This was attempted to counteract by modifying the synthetic training data to sometimes write out the word "date" explicitly in front of a date expression.

5.7.2 Multiple Dates per Sequence

The training data for CANINE only contained either zero or one date expression per input sequence, possibly making it overfitted to only find a maximum of one date expression per input sequence. This problem is exemplified in Figure 1.2 where CANINE recognizes "Mar 31 - Apr 30, 2021" as a single date. An attempt to make CANINE more susceptible to finding multiple dates in an input sequence was made by modifying the synthetic dataset to possibly contain up to three dates per input sequence.

5.7.3 Improving the Post Processing

While I had a post-processing step after CANINE that merged subsequences tagged as date expressions which were suspected to belong to the same date expression, the model also showed problems of missing trailing characters of date expressions. This was attempted to counter by including another post-processing step where all date expressions found by CANINE are extended to be at least 8 characters long or include the full input sequence if it was shorter than 8 characters. I also tried extending to 10 characters which were initially chosen by inspecting Figure 5.2, but this gave very poor results (84.17% system accuracy) and was scaled back.

In addition to this, I also made the merging algorithm a bit more generous, increasing the size of the smallest possible date expression, to five instead of four, before merging it with an adjacent found date expression. Figure 5.2 that there should be very few date expressions with fewer than five characters.

5.7.4 Token Classification Dataset

CANINE for token classification in the complete system now used the *paragraphs* in the OCR response as input sequences. These input sequences are not segmented in the same manner as the dataset described in Section 5.1.1. I built another dataset which splits the OCR response on *paragraph*-level and matches how I decided to input the text sequences to CANINE in the system. This dataset might also solve the above mentioned problem of CANINE being overfitted to only find at most one date expression on each input sequence, as this dataset can contain multiple dates on each input sequence.

The procedure for building this dataset was as follows:

1. Extract a list of all *word* elements from the OCR response.
2. Create a corresponding list of token labels of the same length as the full text which by default was set to 'O' for "other" according to the standard BIO-scheme.
3. Apply the regex module to the OCR response, resulting in a subset of *word* elements which the module identified to represent a date expression. I found the corresponding index in the labels list and flipped the values from O to 'B-DATE' & 'I-DATE'.
4. Split the full text and token labels on the indices where the *paragraph* changed in the OCR response.

Corresponding to Figure 5.1, this dataset had length distributions shown in Figure 5.7. We can note that the number of sequences not containing a date expression is significantly larger than the number of that sequences that contain date expressions. This dataset contains 16,065,509 sequences from which 1,197,907 contains an identified date expression.

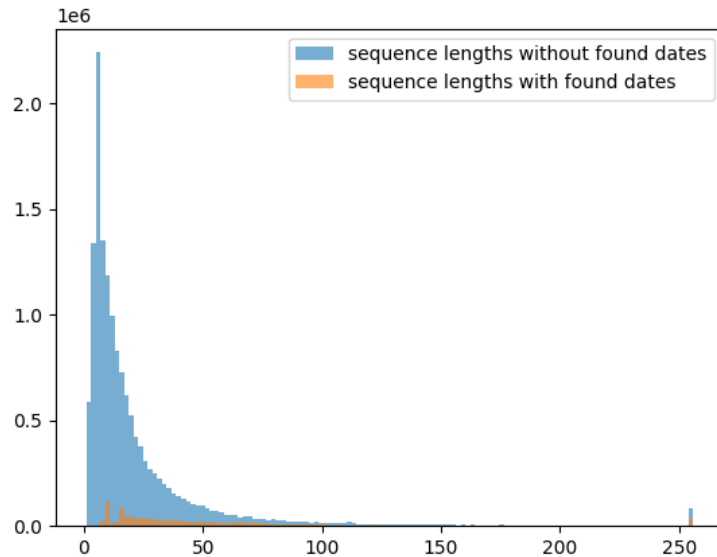


Figure 5.7: Length distribution of sequences in the dataset specifically built for token classification. Note the 'peak' at sequence length 256, all sequences longer than 256 are gathered at this length to avoid further squeezing the x-axis of the graph. During training, this is also the max sequence length to which inputs are truncated.

5.7.5 Fine-tune CANINE Without Synthetic Data

The proportion of generated vs real data which CANINE for token classification was fine-tuned on, was decided from the results of fine-tuning byT5 for formatting date expressions, i.e. another task. To ensure that this proportion was optimal for the token classification step as well, I removed the synthetic data to fine-tune CANINE again. Due to time constraints, this was not as thoroughly tested as with byT5, and only 100% real data versus the original 50% real data + 50% synthetic data was tested.

Chapter 6

Results

In this chapter, I first present all the training results of fine-tuning versions of CANINE and T5. I then present the results from putting CANINE-S for token classification and byT5 together to evaluate how they perform versus the regex module in finding the correct date on a whole receipt, as described in Section 5.6.

6.1 Sequence Classification With CANINE

For this section, CANINE for sequence classification was evaluated on the manually labeled 1003 sequences from Section 5.1.1. The classifiers trained as an LM head on CANINE as described in Section 5.3 gave the following results:

Date?	precision	recall	F1-score
Yes	94%	100%	97%
No	100%	94%	97%

Table 6.1: Evaluation of sequence classification when predicting if a *word* element contained a date expression or not on the testset produced in Section 5.1.1.

Date?	precision	recall	F1-score
Yes	78%	80%	79%
No	82%	80%	81%

Table 6.2: Evaluation of sequence classification when predicting if a full sequence contained a date expression or not on the test set produced in Section 5.1.1.

6.2 Token Classification With CANINE

This section is not evaluated on the manually labeled sequences as they were not labeled on token level. For evaluating CANINE for token classification I extracted a subset of 10,000 sequences instead from the training dataset, which were not included during training. The results of this section are hence presented with the regex module as a reference for ground truth.

6.2.1 Training on the Date Classification Dataset

The results of fine-tuning CANINE with an LM-head on top of the transformer stack with the dataset from Section 5.1.1 combined with synthetic data are presented in Tables 6.3 & 6.4.

BIO-label	precision	recall	F1-score
Out	99%	99%	99%
B-DATE	88%	87%	87%
I-DATE	89%	86%	87%

Table 6.3: Evaluation of token classification for CANINE-S optimized on subword loss. Trained on 500,000 examples from Section 5.1.1 and 500,000 synthetic examples.

BIO-label	precision	recall	F1-score
Out	99%	99%	99%
B-DATE	87%	86%	86%
I-DATE	89%	85%	87%

Table 6.4: Evaluation of token classification for CANINE-C optimized on character loss. Trained on 500,000 examples from Section 5.1.1 and 500,000 synthetic examples.

Although the difference in results between Tables 6.3 and 6.4 are very small I decided to only use CANINE-S (optimized on subword loss) further.

CANINE-S was also evaluated using conllevl to measure the performance on classifying chunks, i.e. a sequence of characters representing a full date expression, in comparison to table 6.3 & 6.4 where the metrics are measured on individual characters.

precision	recall	F1-score
76.35%	78.39%	77.36%

Table 6.5: Chunk-score measured by conllevl. The same fine-tuned version of CANINE-S is evaluated in Table 6.3 on each token individually.

6.2.2 Training on the Token Classification Dataset

The results of fine-tuning CANINE-S with the dataset specifically made for token classification as described in Section 5.7.4 combined with synthetic data, are presented in table 6.6 & 6.7.

BIO-label	precision	recall	F1-score
Out	100%	100%	100%
B-DATE	95%	96%	96%
I-DATE	98%	97%	97%

Table 6.6: Evaluation of token classification for CANINE-S optimized on subword loss. Trained on 500,000 examples from Section 5.7.4 and 500,000 synthetic examples.

precision	recall	F1-score
89.90%	93.87%	91.85%

Table 6.7: Chunks-score from conllevl. Evaluating CANINE-S for token classification on the dataset from Section 5.7.4 and synthetic data.

6.2.3 Training Without Synthetic Data

The results of fine-tuning CANINE-S without any generated synthetic data, i.e. only the dataset from Section 5.7.4 are presented in Tables 6.8 and 6.9.

BIO-label	precision	recall	F1-score
Out	100%	100%	100%
B-DATE	95%	96%	96%
I-DATE	98%	97%	97%

Table 6.8: Evaluation of token classification for CANINE optimized on subword loss. Trained on 1,000,000 examples from the dataset described in Section 5.7.4.

precision	recall	F1-score
94.04%	94.95%	94.49%

Table 6.9: Chunks-score from conllevl, evaluating CANINE-S for token classification on the dataset from Section 5.7.4.

6.3 Formatting Dates by Fine-Tuning T5

When evaluating the different fine-tuned versions of T5 I used the manually labeled 1003 sequences from Section 5.1.1. The results in this section measure how accurately T5 generates the ISO formatted date for the manually labeled testset.

6.3.1 T5

The initial test of fine-tuning T5 to format a date expression to ISO format is presented in Table 6.10.

No post processing	accuracy on full date	70.54%
	accuracy on year (YYYY)	97.42%
	accuracy on month (MM)	75.27%
	accuracy on day (DD)	73.76%
Including switched DD & MM	full date YYYY-MM-DD or YYYY-DD-MM	93.54%
	accuracy on month (MM)	98.71%
	accuracy on day (DD)	97.20%

Table 6.10: Results for the initial test of fine tuning T5-base on 100,000 examples for 1 epoch.

The subsequent test of training on 100,000 examples again with the input sequences enhanced with reference date and locale can be found in Table 6.11.

6.3.2 mT5

The multilingual version of T5, i.e. mT5, was fine-tuned in a similar manner with input sequences enhanced with reference date & locale data. The results of this experiment are presented in Table 6.12.

6.3.3 byT5

On byT5 I first fine-tuned the model in the same manner as mT5. These results are presented in Table 6.13. Further, I investigated the impact of generated synthetic data when ISO-formatting dates with byT5. These results are presented in Table 6.14.

	Accuracy	<i>word element</i>	Full sequence
No post processing	full date YYYY-MM-DD	72.26%	70.10%
	year (YYYY)	98.71%	98.49%
	month (MM)	75.49%	80.21%
	day (DD)	74.62%	73.12%
Including flipped DD & MM	full date YYYY-MM-DD or YYYY-DD-MM	94.84%	84.30%
	month (MM)	98.71%	96.77%
	day (DD)	97.85%	87.74%

Table 6.11: Results of fine-tuning T5-base on 100,000 examples for 1 epoch. training on both *word* elements and the full length sequences. Sequences were prefixed with *reference_date* and *language_code*.

	Accuracy	<i>word element</i>	Full sequence
No post processing	full date YYYY-MM-DD	70.75%	70.97%
	year (YYYY)	98.70%	98.92%
	month (MM)	74.62%	80.43%
	day (DD)	73.12%	73.76%
Including flipped DD & MM	full date YYYY-MM-DD or YYYY-DD-MM	95.27%	84.94%
	month (MM)	99.35%	96.13%
	day (DD)	98.06%	89.24%

Table 6.12: Results of fine tuning mT5-base on 100,000 examples for 1 epoch. training on both *word* elements and the full length sequences. Sequences were prefixed with *reference_date* and *language_code*.

6.4 Complete Pipeline

A sample of 10,000 receipts was used for evaluation in this section. They were selected such that CANINE or byT5 had not been trained on content from these specific receipts. In this section, I measure the accuracy of the complete system similar to how the regex module was evaluated in Section 4.1.

The initial system found the correct date on 9,183 out of the 10,000 receipts, i.e. 91.83% accuracy. Evaluating the regex module on the same 10,000 receipts it found the correct date on 9679, i.e. 96.79% accuracy.

Improvements to the system were attempted as described in Section 5.6. The results of these changes together with the original results and comparisons to the regex module are found in Table 6.15.

For the evaluation of the complete system where my performance achieves a 94.62% ac-

	Accuracy	<i>word element</i>	Full sequence
No post processing	full date YYYY-MM-DD	75.05%	77.85%
	year (YYYY)	98.92%	97.20%
	month (MM)	78.49%	88.82%
	day (DD)	77.63%	81.94%
	Including flipped DD & MM	full date YYYY-MM-DD or YYYY-DD-MM	95.25%
	month (MM)	99.14%	94.84%
	day (DD)	98.05%	87.32%

Table 6.13: Results of fine tuning by T5-small on 100,000 examples for 1 epoch. training on both *word elements* and the full length sequences. Sequences were prefixed with *reference_date* and *language_code*.

	Accuracy on	10% / 111k	20% / 125k	30% / 142k	40% / 166k	50% / 200k	100% / 100k
No post processing	full date YYYY-MM-DD	68.17%	66.45%	62.80%	68.39%	70.96%	94.41%
	year (YYYY)	98.92%	98.92%	98.92%	98.92%	98.92%	98.49%
	month (MM)	71.18%	68.81%	65.60%	71.18%	73.76%	97.42%
	day (DD)	70.75%	68.38%	65.16%	70.75%	73.33%	96.99%
	Including flipped DD & MM	full date YYYY-MM-DD or YYYY-DD-MM	95.70%	95.70%	95.70%	95.70%	95.70%
	month (MM)	99.14%	99.14%	99.14%	99.14%	99.14%	98.28%
	day (DD)	98.50%	98.50%	98.50%	98.50%	98.50%	97.63%

Table 6.14: Results of fine tuning by T5-small on different fractions of added generated data. Note that the header of each column denotes how many percent of the data was generated and that it corresponds to the amount of real data being fixed to 100,000 examples.

accuracy, I compared the misses to the regex module. This resulted in a failure on 538 receipts for the transformer system and 321 for the regex module. Out of these, I found that for 7 receipts the transformer system had found the correct date while the regex module had failed. From these 7 receipts which the transformer system found the correct date, 5 contained OCR misses which caused the regex module to fail. For the other 2, the date expressions were written on formats that were not matched by the regex module, and hence formats that CANINE had not seen during training either. Figure 6.1 shows two examples of these receipts where my system extracts the correct purchase date while the regular expression module fails.

Baseline system			Accuracy
regex module	(full dataset)		92.07%
regex module	(testset)		96.79%
Constructed transformer-based system			
Encoder	Modification to the encoder	Enc./Dec.	
CANINE	base	ByT5	91.83%
CANINE	(Data-augmentation: explicit “date”)	ByT5	90.15%
CANINE	(multiple dates per generated sequence)	ByT5	91.53%
CANINE	(IPP)	ByT5	92.07%
CANINE	(IPP & TCD)	ByT5	94.23%
CANINE	(IPP, TCD & -GD)	ByT5	94.62%

Table 6.15: Evaluations of the complete system with different iterative modifications tried. The accuracy represents the fraction of receipts for which each system found the correctly labeled date for a testset of 10,000 receipts. The first line is an exception where the evaluation is taken from Section 4.1 on the full dataset.

IPP = More inclusive post processing.

TCD = token classification dataset.

-GD = no generated synthetic data for training CANINE.

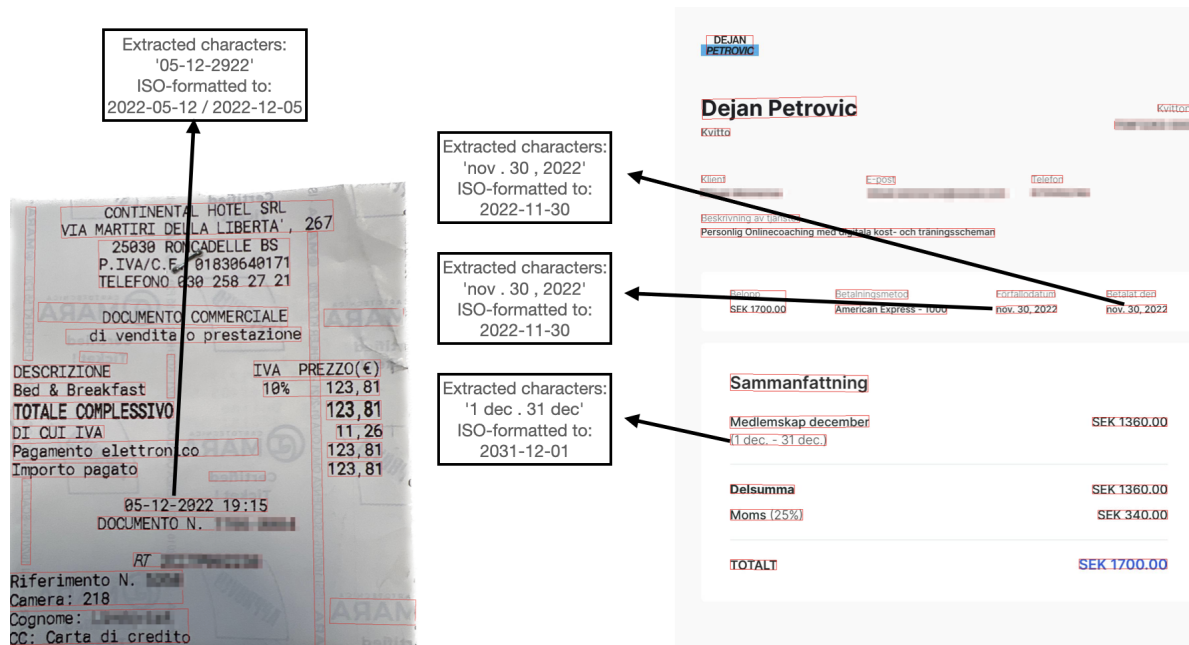


Figure 6.1: Illustration of where my system extracts the correct purchase date for two receipts where the regular expression module fails.

Chapter 7

Discussion

In general, the results show that the constructed machine learning system can achieve results reasonably similar to the regex module, in identifying and extracting date expressions. In this chapter follows a more detailed discussion of the results of the different parts of the system. Further, I comment on some details which are not covered by the results directly. Lastly, I discuss some limitations that were encountered while implementing the system.

7.1 Observations From Results

7.1.1 Sequence Classification

From the sequence classification with CANINE, it seems that when only using the OCR elements which the regex module has classified as a date or not, it is quite a low effort to reproduce the same behavior as the regex module with a transformer model. This can be seen by comparing Tables 6.1 & 5.1 which produce identical results. However when including context around these OCR elements to train on the full sequences, the dataset becomes more noisy and the metrics suffer substantially as presented in Table 6.2.

7.1.2 Token Classification

The reason for changing the dataset when training CANINE for token classification was to make the inference data match the training data. The general idea was that the context surrounding the dates found by the regex module will be more relevant when defined by the *paragraph* elements of the OCR-response as this has a more geometric logic on the receipt image, compared to padding *word* elements to a sequence length threshold. Padding *word* elements to a threshold has the possibility to span over several paragraphs and possibly make the context less relevant. While a major factor for using machine learning models is the

ability to generalize, having the training data match the inference data to an as large extent as possible had a significant positive impact here.

The synthetic data was also found to not help when training CANINE for token classification. I had more control over the synthetic data in the sense that I could generate date expressions on formats that I knew the regex did not capture. However, the context surrounding the date expressions was randomly generated from a set of generator functions. I suspect that the context tokens surrounding the date expressions are seen as noise to the model and as such are very low-quality data which impairs the overall model performance.

7.1.3 Formatting Date Expressions With byT5

In Table 6.14 we see the impact of adding synthetic data when formatting dates with byT5. The fraction of synthetic versus real data seems to have some effect on whether or not the model is confused about which part of the string belongs to the month and day. This confusion does not matter much for the system as the post-processing code is used to switch the day and month during inference to test if both “YYYY-MM-DD” & “YYYY-DD-MM” produces a valid date expression. Comparing Tables 6.13 & 6.14 there is a small gain in performance for byT5 when including some synthetic data. Although it is hard to tell exactly how much synthetic data should be included as most fractions gave the same performance on the metric of “full date YYYY-MM-DD or YYYY-DD-MM”, which was the primary decision-making metric.

No further investigations were made here but I suspect that the remaining 4.3% of the wrongly formatted dates are mostly the same examples for all fine-tuned models with different fractions of synthetic data. A possible explanation could be that the transformer model is trained on data expressions that are extracted from the regex module as well as synthetic examples generated from Faker and formatted by Babel. These formats may have some discrepancies to what was manually labeled and used as ground truth, resulting in the 95.7% accuracy for most tests in Table 6.14.

7.1.4 Post Processing Step

Date expressions are often written together with a time of day. For example, consider the expression “27 Apr 2023 11:23”, where “11:23” is not part of the date expression, but rather represents the time of day. This is a problem, especially for the post-processing step I introduced between CANINE and byT5, where I sometimes extend identified date expression lengths. When the date expression lengths are extended I introduce the possibility of including the characters representing the time of day written after the date to byT5. This is not optimal and may result in byT5 formatting the wrong date. On the other hand not extending the found date expression lengths has the converse problem of possibly missing the last characters of a date expression, i.e. CANINE predicts ‘O’ when it should predict ‘L-DATE’. Consequently, byT5 isn’t provided the full date expression with all the information needed to format the correct date. I found that extending the found date expressions identified by CANINE to 8 characters was the ‘sweet spot’ between these two effects.

However, when inspecting the failures of my system, the issue of extracting the time of day is still a typical error for which I did not find a good solution. Figure 5.6 also exemplifies

this problem. The post-processing step might not be the best solution for this problem either, rather a dataset better labeled for the task may solve this issue.

7.2 Notable Details

In Table 6.15 we can observe a significant difference in accuracy for the regex module depending on if it was evaluated on the same testset as the transformer system or the full dataset. This is caused by a difference in the processing of the dataset. As mentioned in Section 4.1 I removed examples that were missing the labeled feature for the three respective labels which were investigated. However when building the dataset for training I removed all examples which were missing any of these three labels. This results in a bias for the produced datasets which are possibly more 'clean' than the data which might be encountered for inference in a production setting. As such the performance of 94.62% for the transformer should only be compared to the 96.79% accuracy of the regex module, evaluated on the same dataset.

In this thesis, I focused on performance in terms of finding the right answer, i.e. the purchase date on an image of a receipt. However, it should be noted that the constructed transformer system is significantly slower compared to running the regex module. For the testset of 10,000 receipts, the regex module runs for approximately one minute while the transformer system takes over an hour. While these numbers were not measured precisely and are dependent on implementation I find the difference significant enough to be mentioned. The inference of the transformer system was done on a CPU machine and can be optimized to some extent by implementing batching and running it on a GPU machine instead. Nevertheless, the built transformer system processes a full receipt in under a second with the current implementation, which can be considered an acceptable delay.

7.3 Limitations

In this thesis, I had to work with training data for a machine learning system that was not specifically labeled for the problem. This is a common problem in machine learning which impacts the results.

The available hardware during the project was also a limiting factor. As mentioned in Chapter 2, scaling language models has been shown to generally improve performance. In this thesis, I chose models which I could, mostly, fit into a single GPU with 16GB of VRAM.

7.3.1 Dataset

The available dataset was labeled as an input image of a receipt having a corresponding purchase date. This means that performing token classification had to be labeled by some automated procedure during the project. The solution I found to this was to use the regex module in production to label character sequences as part of a date expression and also generate synthetic data. Both the regex module and the generation of synthetic data follow strict rules. The idea was that the models could utilize their pre-training to generalize from the strict rules of the produced training dataset during fine-tuning. However, these rules may also just

be replicated by the machine learning model without much generalization. Specifically a few problems I noticed with our datasets after fine-tuning were:

- Regex-generated dataset: Training a machine learning model on the datasets from Sections 5.1.1 & 5.7.4 will likely make it reproduce the same systematic errors as the regex-module itself. For instance, a systematic error was false positives of addresses as mentioned in Section 5.1.1. One idea of producing a synthetic dataset was to offset systematic errors such as this by producing similar data with the opposite label.
- synthetically generated dataset: While this gave the possibility of generating locale-dependent date expressions on arbitrary formats to teach the models to recognize it was very hard to produce surrounding context data in the input sequence that made sense and not just produce extra noise for the model to filter.

7.3.2 Hardware

As already mentioned, the system created is significantly slower compared to the regex-module which it's compared to. However, if performance in terms of accuracy alone is to be optimized it is possible to use larger transformer models in hopes of reaching better results. For instance, I used byT5-small which could have been exchanged for a larger version in the same model family. CANINE has no larger version in its family but other, larger, models for token classification could have been tested.

Strubell et al. (2019) produced a study of energy consumption and CO₂ emissions from training large machine learning models. They found that training a large transformer model can produce CO₂ emissions comparable to that of airplane travel. In this thesis, I fine-tune already pre-trained transformer models which removes a large amount of computational cost. Even so, the computational power required for the system I built is significantly larger than the regex module which it is compared to. As such, further attempts at improving my system beyond the accuracy of the regex module by possibly utilizing even larger transformer models should not be done uncritically.

Chapter 8

Conclusion & Future Work

To answer the first research question of How I constructed a machine learning system to extract dates on arbitrary formats from natural text, specifically pertaining to receipts. I chose to construct a solution consisting of two transformer models connected in sequence. The first machine learning model consisted of an encoder model of the transformer architecture which performed token classification. After the first model was a post-processing that attempts to breach the gap between the character level accuracy and chunk accuracy. Lastly, the system consisted of an encoder-decoder model which generated an ISO formatted date expression from the expression identified by the first model.

The system was compared to a regex module which *The Company* currently has in production. The regex module was also used to construct the training data for the transformer models. The final accuracy of the transformer system was 94.62% compared to the regex-modules 96.79%.

Regarding the second research question , generating synthetic data was a harder task than expected. In this thesis, I found some minor success in using synthetically generated date expressions for byT5 to ISO format. For the token classification model however it had a negative effect on the accuracy. While it gave very small successes in this work, I suspect that it can be further improved on to achieve better results. Approaches could include more integration with the real datasets, where manipulating date expressions found by the regex module in sequences to include formats that the regex module doesn't catch. Another data augmentation that could be tested is to insert OCR failures that have been observed manually into the training data.

Due to the time constraints of this thesis, I only tested CANINE & different versions of T5. Other models could possibly be fine-tuned and evaluated. However, the choice of CANINE was in part due to it being a tokenization-free, character-level model which made it easier to implement for token classification.

Further, having a manually labeled dataset, containing date expressions on a variety of formats could potentially improve performance.

For a small number of receipts, the system found the correct date while the regex module failed. I inspected these receipts individually to find typical examples of OCR failures. Hence the built transformer system achieves an inferior accuracy compared to the regex module. The potential upside of using a machine learning system instead is that it shows some indications to generalize and eliminate the shortcomings of the regex module stated in Section 4.1.

The potential for machine learning systems to replace some regular expression tasks seems to be possible but for this specific task, more improvements will have to be made. Resource efficiency should also be considered, both for economic and environmental reasons. With this in mind, regular expression techniques have an advantage compared to large machine learning systems utilized in this thesis.

References

- Almasian, S., Aumiller, D., and Gertz, M. (2021). BERT got a date: Introducing transformers to temporal tagging. *CoRR*, abs/2109.14927.
- Bahdanau, D., Cho, K., and Bengio, Y. (2015). Neural machine translation by jointly learning to align and translate. In Bengio, Y. and LeCun, Y., editors, *3rd International Conference on Learning Representations, ICLR 2015, San Diego, CA, USA, May 7-9, 2015, Conference Track Proceedings*.
- Boguraev, B., Castaño, J., Gaizauskas, R., Ingria, B., Katz, G., Knippen, B., Littman, J., Mani, I., Pustejovsky, J., Sanfilippo, A., See, A., Setzer, A., Saurí, R., Stubbs, A., Sundheim, B., Symonenko, S., and Verhagen, M. (2002). A formal specification language for events and temporal expressions. url: http://timeml.org/site/publications/timeMLdocs/timeml_1.2.1.html.
- Cho, K., van Merriënboer, B., Gulcehre, C., Bahdanau, D., Bougares, F., Schwenk, H., and Bengio, Y. (2014). Learning phrase representations using RNN encoder–decoder for statistical machine translation. In *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 1724–1734, Doha, Qatar. Association for Computational Linguistics.
- Clark, J. H., Garrette, D., Turc, I., and Wieting, J. (2021). CANINE: pre-training an efficient tokenization-free encoder for language representation. *CoRR*, abs/2103.06874.
- Devlin, J., Chang, M., Lee, K., and Toutanova, K. (2019). BERT: pre-training of deep bidirectional transformers for language understanding. In Burstein, J., Doran, C., and Solorio, T., editors, *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, NAACL-HLT 2019, Minneapolis, MN, USA, June 2-7, 2019, Volume 1 (Long and Short Papers)*, pages 4171–4186. Association for Computational Linguistics.
- Eikvil, L. (1993). Optical character recognition. *citeseer.ist.psu.edu/142042.html*, 26.
- Google (2023). Cloud vision API. url: <https://cloud.google.com/vision/docs/libraries>.

- ISO8601 (2019). International organization for standardization. url: <https://www.iso.org/obp/ui/#iso:std:70907:en>.
- Kleene, S. C. (1951). Representation of events in nerve nets and finite automata. *Princeton University Press*.
- Koskela, A. et al. (2023). Babel, speaking your language. url: <https://babel.pocoo.org/en/latest/>.
- Kudo, T. and Richardson, J. (2018). SentencePiece: A simple and language independent subword tokenizer and detokenizer for neural text processing. In *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing: System Demonstrations*, pages 66–71, Brussels, Belgium. Association for Computational Linguistics.
- Li, M., Lv, T., Cui, L., Lu, Y., Florêncio, D. A. F., Zhang, C., Li, Z., and Wei, F. (2021). Trocr: Transformer-based optical character recognition with pre-trained models. *CoRR*, abs/2109.10282.
- Lothritz, C., Allix, K., Veiber, L., Bissyandé, T. F., and Klein, J. (2020). Evaluating pretrained transformer-based models on the task of fine-grained named entity recognition. In *Proceedings of the 28th International Conference on Computational Linguistics*, pages 3750–3760, Barcelona, Spain (Online). International Committee on Computational Linguistics.
- Mikolov, T., Chen, K., Corrado, G., and Dean, J. (2013). Efficient estimation of word representations in vector space. In Bengio, Y. and LeCun, Y., editors, *1st International Conference on Learning Representations, ICLR 2013, Scottsdale, Arizona, USA, May 2-4, 2013, Workshop Track Proceedings*.
- Moore, G. E. (1965). Cramming more components onto integrated circuits. *Electronics*, 38(8).
- Narang, S., Chung, H. W., Tay, Y., Fedus, L., Fèvry, T., Matena, M., Malkan, K., Fiedel, N., Shazeer, N., Lan, Z., Zhou, Y., Li, W., Ding, N., Marcus, J., Roberts, A., and Raffel, C. (2021). Do transformer modifications transfer across implementations and applications? In Moens, M., Huang, X., Specia, L., and Yih, S. W., editors, *Proceedings of the 2021 Conference on Empirical Methods in Natural Language Processing, EMNLP 2021, Virtual Event / Punta Cana, Dominican Republic, 7-11 November, 2021*, pages 5758–5773. Association for Computational Linguistics.
- Pennington, J., Socher, R., and Manning, C. D. (2014). Glove: Global vectors for word representation. In Moschitti, A., Pang, B., and Daelemans, W., editors, *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing, EMNLP 2014, October 25-29, 2014, Doha, Qatar, A meeting of SIGDAT, a Special Interest Group of the ACL*, pages 1532–1543. ACL.
- Pua, A. (2022). Faker. url: <https://faker.readthedocs.io/en/master/>.
- Radford, A., Wu, J., Child, R., Luan, D., Amodei, D., and Sutskever, I. (2018). Language models are unsupervised multitask learners.

-
- Raffel, C., Shazeer, N., Roberts, A., Lee, K., Narang, S., Matena, M., Zhou, Y., Li, W., and Liu, P. J. (2019). Exploring the limits of transfer learning with a unified text-to-text transformer. *CoRR*, abs/1910.10683.
- Ramshaw, L. and Marcus, M. (1995). Text chunking using transformation-based learning. In *Third Workshop on Very Large Corpora*.
- Seker, A. C. and Ahn, S. C. (2022). A generalized framework for recognition of expiration dates on product packages using fully convolutional networks. *Expert Systems with Applications*, 203:117310.
- Sennrich, R., Haddow, B., and Birch, A. (2016). Neural machine translation of rare words with subword units. In *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics, ACL 2016, August 7-12, 2016, Berlin, Germany, Volume 1: Long Papers*. The Association for Computer Linguistics.
- Strötgen, J. and Gertz, M. (2010). HeidelTime: High quality rule-based extraction and normalization of temporal expressions. In *Proceedings of the 5th International Workshop on Semantic Evaluation*, pages 321–324, Uppsala, Sweden. Association for Computational Linguistics.
- Strubell, E., Ganesh, A., and McCallum, A. (2019). Energy and policy considerations for deep learning in NLP. In *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*, pages 3645–3650, Florence, Italy. Association for Computational Linguistics.
- Tjong Kim Sang, E. F. and De Meulder, F. (2003). Introduction to the CoNLL-2003 shared task: Language-independent named entity recognition. In *Proceedings of the Seventh Conference on Natural Language Learning at HLT-NAACL 2003*, pages 142–147.
- Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., Kaiser, L., and Polosukhin, I. (2017). Attention is all you need. *CoRR*, abs/1706.03762.
- Villalobos, P., Sevilla, J., Besiroglu, T., Heim, L., Ho, A., and Hobbhahn, M. (2022). Machine learning model sizes and the parameter gap. *CoRR*, abs/2207.02852.
- Wirth, R. and Hipp, J. (2000). Crisp-dm: towards a standard process model for data mining. In *Proceedings of the 4th international conference on the practical applications of knowledge discovery and data mining*, pages 29–39.
- Wolf, T., Debut, L., Sanh, V., Chaumond, J., Delangue, C., Moi, A., Cistac, P., Rault, T., Louf, R., Funtowicz, M., and Brew, J. (2019). Huggingface’s transformers: State-of-the-art natural language processing. *CoRR*, abs/1910.03771.
- Xue, L., Barua, A., Constant, N., Al-Rfou, R., Narang, S., Kale, M., Roberts, A., and Raffel, C. (2022). Byt5: Towards a token-free future with pre-trained byte-to-byte models. *Trans. Assoc. Comput. Linguistics*, 10:291–306.
- Xue, L., Constant, N., Roberts, A., Kale, M., Al-Rfou, R., Siddhant, A., Barua, A., and Raffel, C. (2020). mt5: A massively multilingual pre-trained text-to-text transformer. *CoRR*, abs/2010.11934.
-

Zhuang, F., Qi, Z., Duan, K., Xi, D., Zhu, Y., Zhu, H., Xiong, H., and He, Q. (2019). A comprehensive survey on transfer learning. *CoRR*, abs/1911.02685.

EXAMENSARBETE Using Transformers to Extract Date Expressions From Receipts**STUDENT** Axel Leander**HANDLEDARE** Pierre Nugues (LTH)**EXAMINATOR** Flavius Gruian (LTH)

Extrahering av datumsuttryck ur kvitton med hjälp av maskininläring

POPULÄRVETENSKAPLIG SAMMANFATTNING Axel Leander

Utläggredovisning och bokföring är tidskrävande uppgifter som har potential att automatiseras med verktyg så som maskininläring. I detta arbetet byggde jag ett system av transformers-modeller för att extrahera datumsuttryck ur kvitton.

Att extrahera data ur dokument är mycket tidskrävande om det görs manuellt. För en dator finns det potential att automatisera och göra processen snabbare. Regelbaserade verktyg, som Regular Expression (regex) kan användas för att matcha och extrahera data ur text som följer ett specifikt definierat mönster. Vanlig data att extrahera med hjälp av regex kan vara telefonnummer, mail-adresser m.m. Det kan dock uppstå problem av matchningen vid förekomst av stavfel och liknande. En annan källa till fel uppstår vid skanning av pappersdokument till digitalt format där optical character recognition (OCR) används och det förekommer avläsningsfel.

I mitt examensarbete undersökte jag extraktion av datumsuttryck från kvitton med hjälp av maskininläring. Kombinationen av att datum kan skrivas på många olika format tillsammans med problemet av OCR-avläsningsfel gör att regex potentiellt inte är en optimal strategi. Jag designade ett system där två maskin-inlärningsmodeller av transformer-arkitekturen används i sekvens. Den första modellen extraherar vad den identifierar som ett potentiellt uttryck för ett datum och skickar vidare till den andra modellen som genererar datumet på standardiserat ISO-format.

Ett vanligt problem med maskininläring är tillgång till stora mängder relevant träningsdata. Därför undersöktes även möjligheten att generera syntetisk data för att träna maskin-inlärningsmodellerna.

Resultaten visade att systemet kan identifiera datumsuttryck där OCR-avläsningen identifierat enskilda tecken fel. Samt kunde modellen identifiera datumsuttryck på format med små variationer från vad den tränats på. Generering av syntetisk data visade en svag förbättring för genereringsmodellen men lyckades inte nyttjas för att förbättra klassificeringsmodellen. Trots maskin-inlärningsmodellernas förmåga att generalisera till nya format och förbise OCR-avläsningsfel presterade referenssystemet, byggt på regular expression, överlag lite bättre.

