FACULTY OF APPLIED SCIENCES
UNIVERSITY
OF WEST BOHEMIA

DEPARTMENT OF
COMPUTER SCIENCE
AND ENGINEERING

Master's Thesis

Multimodal recognition of historical named entities

Jiří Trefil

PILSEN, CZECH REPUBLIC

2024

FACULTY OF APPLIED SCIENCES
UNIVERSITY
OF WEST BOHEMIA

DEPARTMENT OF
COMPUTER SCIENCE
AND ENGINEERING

Master's Thesis

Multimodal recognition of historical named entities

Bc. Jiří Trefil

Thesis advisor
Doc. Ing. Pavel Král, Ph.D.

PILSEN, CZECH REPUBLIC 2024

Citation in the bibliography/reference list:
TREFIL, Jiří. *Multimodal recognition of historical named entities*. Pilsen, Czech Republic, 2024. Master's Thesis. University of West Bohemia, Faculty of Applied Sciences, Department of Computer Science and Engineering. Thesis advisor Doc. Ing. Pavel Král, Ph.D.

ZÁPADOČESKÁ UNIVERZITA V PLZNI
Fakulta aplikovaných věd
Akademický rok: 2024/2025

ZADÁNÍ DIPLOMOVÉ PRÁCE

(projektu, uměleckého díla, uměleckého výkonu)

Jméno a příjmení:	**Bc. Jiří TREFIL**
Osobní číslo:	**A22N0060P**
Studijní program:	**N0613A140040 Softwarové a informační systémy**
Téma práce:	**Multi-modální rozpoznávání historických pojmenovaných entit**
Zadávající katedra:	**Katedra informatiky a výpočetní techniky**

Zásady pro vypracování

1. Seznamte se s úlohou rozpoznávání pojmenovaných entit (NER) a metodami pro multi-modální zpracování dat s důrazem na textovou a obrazovou modalitu.
2. Prostudujte existující datové sady pro historické NER obsahující více modalit.
3. Z dodaných dat vytvořte anotovaný dataset pro multi-modální rozpoznávání historických pojmenovaných entit.
4. Navrhněte a vytvořte prototyp systému pro multi-modální historické NER s využitím neuronových sítí.
5. Prototyp otestujte na dvou vybraných datových sadách.
6. Zhodnoťte dosažené výsledky a navrhněte další možná rozšíření.

Rozsah diplomové práce: **doporuč. 50 s. původního textu**
Rozsah grafických prací: **dle potřeby**
Forma zpracování diplomové práce: **tištěná/elektronická**
Jazyk zpracování: **Angličtina**

Seznam doporučené literatury:

dodá vedoucí diplomové práce

Vedoucí diplomové práce: **Doc. Ing. Pavel Král, Ph.D.**
Katedra informatiky a výpočetní techniky

Datum zadání diplomové práce: **9. září 2024**
Termín odevzdání diplomové práce: **15. května 2025**

L.S.

Doc. Ing. Miloš Železný, Ph.D.
děkan

Doc. Ing. Přemysl Brada, MSc., Ph.D.
vedoucí katedry

V Plzni dne 30. září 2024

Declaration

I hereby declare that this Master's Thesis is completely my own work and that I used only the cited sources, literature, and other resources. This thesis has not been used to obtain another or the same academic degree.

I acknowledge that my thesis is subject to the rights and obligations arising from Act No. 121/2000 Coll., the Copyright Act as amended, in particular the fact that the University of West Bohemia has the right to conclude a licence agreement for the use of this thesis as a school work pursuant to Section 60(1) of the Copyright Act.

V Plzni, on 14 February 2024

. . . . . .

Jiří Trefil

Abstract

This thesis focuses on multimodal named entity recognition (NER). It explains the main issues in this domain, the common problems associated with natural language processing and the motivation behind this task. It also presents three datasets from this research field, two publicly available and one self-created. The first two of the mentioned datasets were created by collecting tweets from the social media network Twitter, while the third dataset was created from a collection of books on the history of the Czech-German territory.
Most of the research in this area has focused exclusively on text modality. Multimodal recognition has recently gained popularity due to the large amount of data and its availability. The aim of this work is to determine whether multimodal NER would yield better results than unimodal one. Three unique neural network architectures have been proposed that use specific modules to process text and image data. A total of 23 unique topologies were developed for the experiments, mainly using recurrent neural networks and transformer architecture. Large language models, namely GPT-4o and Llama 3.1 were also used.
Experiments have shown that a multimodal processing can in fact improve performance of recognition in some cases.

Abstrakt

Tato diplomová práce se zabývá multimodálním rozpoznáváním pojmenovaných entit. Práce vysvětluje problematiku v této oblasti, časté problémy spojené při zpracování přirozeného jazyka a také důvody, proč se touto úlohou zabývat. Dále představuje tři datové sady z této domény, dvě veřejně dostupné a jednu vlastnoručně vytvořenou. První dvě zmíněné sady vznikly kolekcí tweetů ze sociální sítě Twitter, třetí datová sada byla vytvořena ze souboru pěti knih o historii českoněmeckého území.
Většina výzkumu v této oblasti se soustředila výlučně na textovou modalitu. Multimodální rozpoznávání nabývá v poslední době na popularitě, zejména díky velkému objemu dat a jejich dostupnosti. Cílem této práce je zjistit, zda multimodální přístup rozpoznávání pojmenovaných entit přinese lepší výsledky než jejich unimodální zpracování. Jsou navrženy tři unikátní architektury neuronových sítí, které používají specializované moduly na zpracování textu a obrázků. Během této práce vzniklo celkem 23 unikátních topologií, které používají zejména rekurentní neuronové sítě a transformer architekturu. Použity byly také Velké jazykové modely, konkrétně

GPT-4o a Llama 3.1.
Experimenty prokázaly, že multimodální zpracování v některých případech pomůže zvýšit úspěšnost rozpoznávání.

Keywords

natural language processing • named entity recognition • BERT • Llama 3.1 • ViT • multimodal named entity recognition • python

Acknowledgement

I would like to thank to my supervisor, Pavel Král Doc. Ing. Ph.D. for guidance, general advice and regular communication during the development of this thesis.

Contents

Introduction 1

Named entity (NE) recognition is a task in the domain of natural language processing (NLP). Named Entity is a real world object which can be denoted with a **proper name** such as people, organizations and more. Most of the research conducted in this domain focused on a singular modality - text. This unimodal approach works but does not handle *ambiguity* well. For example a follow sentence, *We visited Washington in spring.*, *Washington* is denoted with a proper name therefore it is a named entity. However is not clear if the sentence refers to the city or the person.
Multimodal approaches, i.e using more than one modality (text and image for this thesis) offer a solution for this problem. Using additional modality, for example image data, ambiguity can be mitigated. If the sample sentence mentioned would have a picture of the city of Washington D.C. the ambiguity is solved.
The purpose of this master's thesis is to identify if multimodal approaches are better than unimodal. To achieve this goal a system for multimodal named entity recognition with text and image modalities will be designed and implemented. For this purpose a deep neural networks with Transformer based architecture, recurrent neural networks and Large language models (LLM) will be used. Multimodal and unimodal solutions will be compared with each other.
The structure of this thesis is following. First NER is introduced with historical evolution in this task with common challenges and the difference between multimodal and unimodal approach. Relevant work in the field of Multimodal named entity recognition (MNER) is mentioned next. The fourth chapter dives deeply into modern machine learning and its fundamentals. The following chapters goes deeply into concrete topology of neural networks that were used in conducted experiments.
Chapter 6 describes the statistics datasets that are used in the experiments. Two publicly available, Twitter2015 and Twitter2017 and one manually created. The manual creation of the dataset led to implementation of useful tool for parsing scanned pages of books. Chapter 7 focuses on problem space of MNER. It states requirements for MNER and proposes three distinct designs for viable system. It also delves deeply into the complications associated with multimodal solutions and the challenges of data preprocessing. Chapter 8 contains the architecture of proposed solutions. It

also contains detailed information about used optimizer and scheduler and tool for dataset creation. Chapter 9 describes metric used for evaluation of experiments. Most importantly, this chapter contains the results achieved on mentioned datasets. Chapter 10 contains conclusion and suggests further research and improvements in this domain.

Recognition of named entities 2

The following chapter will briefly describe the motivation behind named entity recognition. It will state what named entities are, more specifically what *historical named entities* are. Various approaches on NER are described, notably statistical approach with CRF and machine learning methods. Information retrieval is mentioned as a use-case of NER. Lastly it will define the **key difference** between multi-modal and unimodal recognition of said entities.

2.1 Named entities

Named entity (NE) is a real-world object which can be denoted with a *proper name*. A proper name is a noun that identifies a single entity and is used to refer to that entity as distinguished from a common noun. For example the noun "Gates" refers to a name, not "gates" as the structures or abstract terms. For more detailed description of a NE refer to [1, 2].
This means that a NE can be used to retrieve information about a subject, place or a person and "filter out" the unwanted documents. Named entities in text are shown in Figure 2.1.
This masters thesis focuses on historical named entities. The named entities are extracted from historical books.

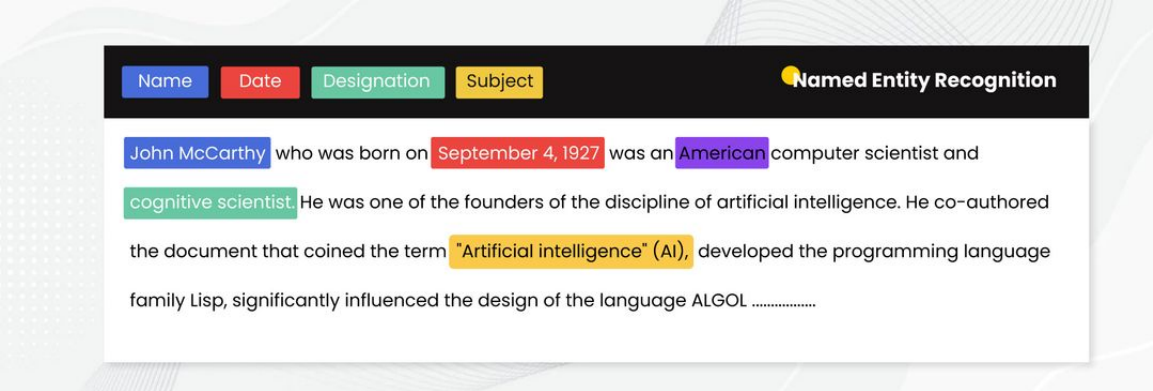

Figure 2.1: Named entities recognized in text [3].

Named entities recognized in text with four predefined categories: *Name, Date, Designation, Subject*.

2.2 Named Entity Recognition

Named entity recognition (NER) can be looked at as *sequence labeling problem* given sequence input $X = (x_1, ..., x_n)$ the task is to predict $Y = (y_1, ..., y_n)$ where x_i is part of a sequence (usually a single *word*) and y_n is named entity.
NER corresponds to the identification of **named entities in texts**, generally of the types *Person, Organisation and Location*. Such entities act as referential anchors which underlie the semantics of texts and guide their interpretation [4]. Figure 2.2 shows overall architecture of an information retrieval system.

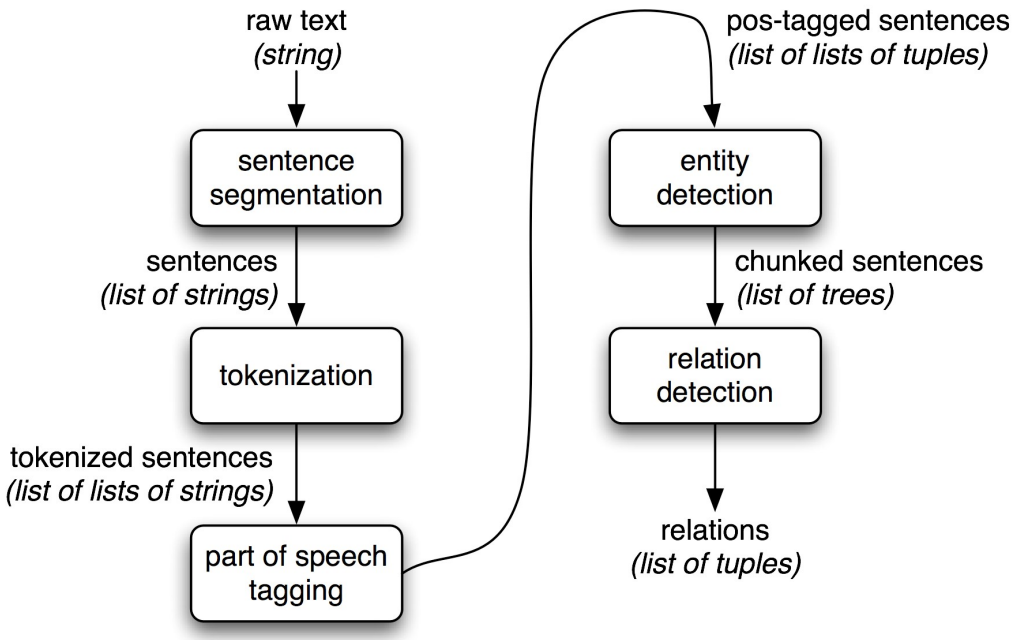

Figure 2.2: Architecture of information extraction system [5].

In the first step, raw data is segmented into individual sentences which are further processed in the second step, **tokenization**. The process of tokenization breaks the sentences into individual words. The words are in the following step processed by **part of speech tagging (POS)** which labels each word in a sentence with its corresponding part of speech (noun, adjective, etc.). In the forth step, this preprocessed text is then searched for **named entities** and the last step detects relationships between them and creates a knowledge graph.

NER methods can be classified into :

- Rule-based Methods
- Statistical Methods
- Machine Learning Methods
- Hybrid Methods

2.2.1 BIO format

BIO[1] format is a tagging scheme used in NER to label tokens in a sequence. It stands for:

- **B**egging - indicates the beggining of a named entity.

[1] formally Inside–outside–beginning

- **I**nside - indicates that the token is a named entity, but is not a first token.
- **O**utside - indicates that the token is not a named entity. This label is usually denoted as OTHER.

For example a sentence *Barack Obama was in Hawaii.* in BIO format can be seen in Table 2.1.

Table 2.1: BIO format

Token	Label
Barack	B-PERSON
Obama	I-PERSON
was	OTHER
in	OTHER
Hawaii	B-LOCATION

2.2.2 Rule-based method

This process is based on an *expert* creating a *set of rules*, ie. creating a *formal grammar* [6] which will be able to recognize **named entities** in text.
Example of formal grammar is shown in Figure 2.3. The **major** disadvantage is scaling and maintaining a robust set of rules.
Examples of this can be *regular expression* which are very efficient and finding "non-complex" entities, for example for phone numbers.
`^(\+\d{1,2}\s)?\(?\d{3}\)?[\s.-]\d{3}[\s.-]\d{4}$` this regular expression will be able to find phone numbers in unstructured text, however finding complex named entities, such as Person or Organization is next to impossible because of *ambiguity*.

2.2.3 Machine Learning Methods

This method is very popular in modern day and age, especially after the "renaissance era" that deep learning [7] started.
The process of machine learning in general consists of:

- **Task description** - The goal of the model, for example NER.
- **Dataset** - The data is a set of tuples $\{x, y\}$ where x is the data itself (for example sentence) and y is the label for classification (for example named entities in said the sentence).
- **Classifier** - A discriminative model that outputs probablity distribution of Y (classes) given X (input data).

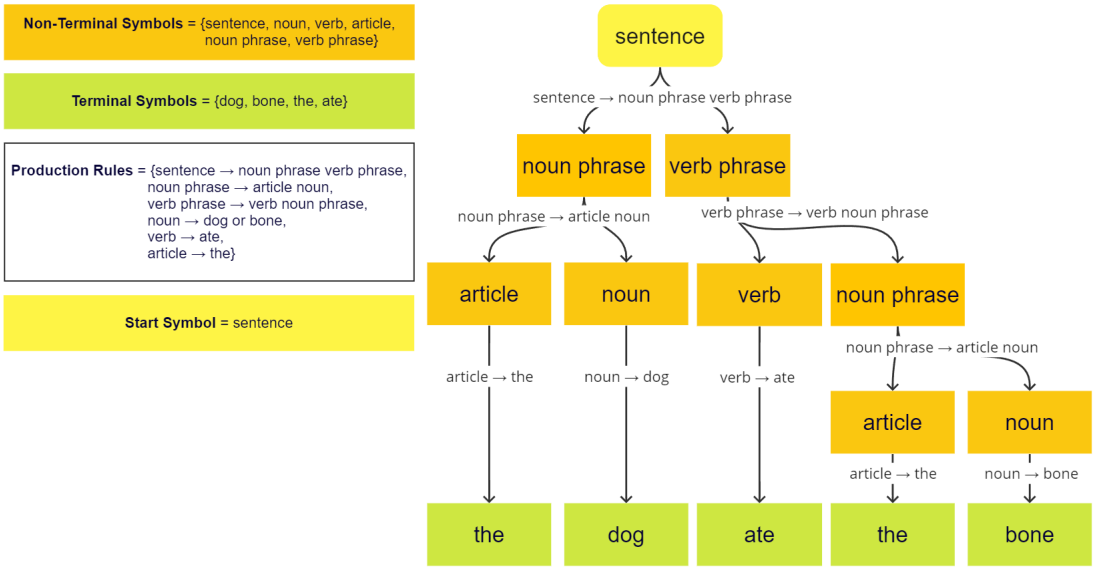

Figure 2.3: Example of formal grammar that *generates* sentences [3].

- **Training** - The process of iterating over training data to train the model i.e. acquire knowledge.
  In this phase, the model performs classification with some numerically measurable error. The main purpose of this process is to **minimize** this error (or *loss value*) by adjusting parameters of the model via *backpropagation* [7].
- **Validation** - Subset of the dataset which is used to validate the performance of the model. This can be used to stop the training process early if the model is not learning or to tweak models hyper parameters (learning rate α, etc.) for optimal performance.
- **Inference** - The process of using a trained model to make predictions or decisions based on new, **unseen** data.

Neural networks (described in Chapter 4 and 7) utilize machine learning to obtain knowledge. They are the only machine learning algorithm used in implemented system with the exception of CRF.

2.2.4 Statistical methods

Transitioning from manual rules, statistical methods employ models like Hidden Markov Models or Conditional Random Fields. They predict named entities based on likelihoods derived from training data.

2.2.4.1 Generative and discriminative models

Generative models represent the relationship between observed data and classes (i.e. the distribution of all possible pairs of X and Y given model parameters θ) $P(X, Y|\theta)$. This can be used to generate various combinations (X, Y) for given X, θ.
Discriminative models model conditional distribution of the output Y given X as an input and model parameters θ => $P(Y|X, \theta)$. This is what **classifiers** do. Note that discriminative model can be derived from generative model, for more on that refer to [8].

2.2.4.2 Hidden Markov Model

Hidden Markov Model (HMM) is a statistical model that assumes the presence of *hidden states* governing the observable sequences.
In NER, the hidden states correspond to the entity tags (e.g., "B-PERSON" for the beginning of a person's name, "I-PERSON" for inside a persons name, "I-LOC" and "B-LOCATION" equivalently for location.) and the observed data are the words in the text.
For example a sentence: *"John Doe lives in New York."* is split into tokens:
[John, Doe, Lives,in, New, York]
and the *hidden states* are:
[B-PERSON,I-PERSON,O,O,B-LOCATION,I-LOC].
HMM would estimate the probability of this sequence by modeling:

- Transition Probabilities: The probability of moving from one tag to another (e.g., from "B-PERSON" to "O").
- Emission Probabilities: The probability of a word being emitted from a specific tag (e.g., the word "John" given the tag "B-PERSON").

HMM works well for sequential data but **assumes that each tag only depends on the previous tag and enforces 1:1 relation between** X **and** S. This assumption is limiting when working with natural languages which are contextually dependent. The other limitation lies in the transition probabilities. They are static.
This means that the transition probability, for example: $P(S_1|S_2)$ where S_2 is *adjective* and S_1 is a *noun*, will not change throughout the sequence. This seems a little naive and restrictive for NER. Due to all these problems, HMM was not used in conducted experiments. Figure 2.4 shows a structure of an HMM.

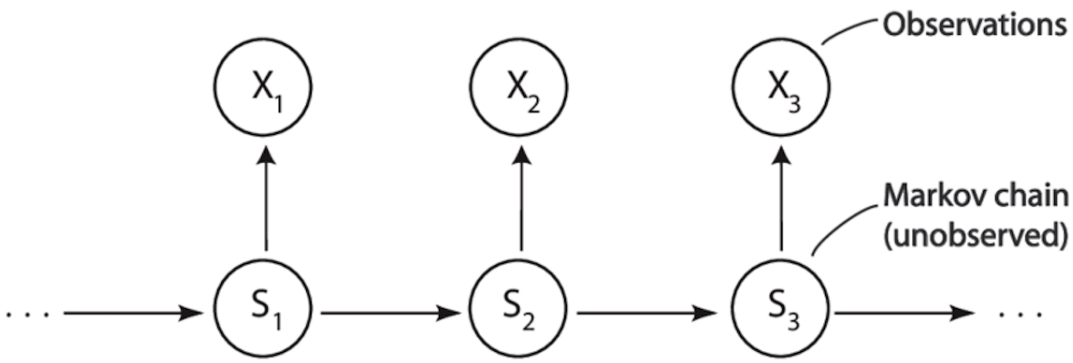

Figure 2.4: Hidden Markov Model structure [9].

X_1 to X_3 are the observed data - words in sentence in case of NLP. S_1 to S_3 are the hidden states. Nach arrow indicating transition from one node to another has a weight that corresponds to the probability of this transition.
Arrow between S_i and S_{i+1} and its associated weight is called *transition probability*. Arrow between S_i and x_i is called *emission probability*.

2.2.4.3 Conditional Random Fields

Conditional Random Fields (CRF) models the conditional probability of the label sequence given the observation sequence, rather than joint probabilities. This difference in modeling is commonly expressed as *discriminative model* vs. *generative model*.
The very general nature of a CRF eliminates restriction of an HMM but is more computionaly expensive. This means the we can have relation between various hidden states S and observed data X and even between the hidden states themselves. In *Linear chain CRF* [10] which is commonly used, only transitions between states Y_i, Y_{i-1} are allowed. In [11] Laffety et. al define the probability of a particular label sequence Y given observation sequence X to be a normalized product of potential functions.
The probability can be expressed (simplified version by [10] but equivalent to the one by Laffety et. al):

$$P(y|X,\theta) = \frac{1}{Z(X)} \exp(\sum_{i}^{n} \sum_{j} \theta_j f_j(y_{i-1}, y_i, X, i))$$

where θ_j is trainable weight for feature function f_j
$f_j(y_{i-1}, y_i, X, i)$ is a concrete feature function with parameters:

- y_{i-1} - previous hidden state
- y_i - current state
- X - the *entire* observed sequence
- i - the *position* in the sequence
- $Z(X)$ - the partition function that ensures the distribution sums to 1 over all possible label sequences.

Two important observations:

1. The entire observed sequence X impacts the predicted states Y.
2. The *position* i in the sequence has impact as well.

Figure 2.5 shows the difference between HMM and Linear chain CRF connections.

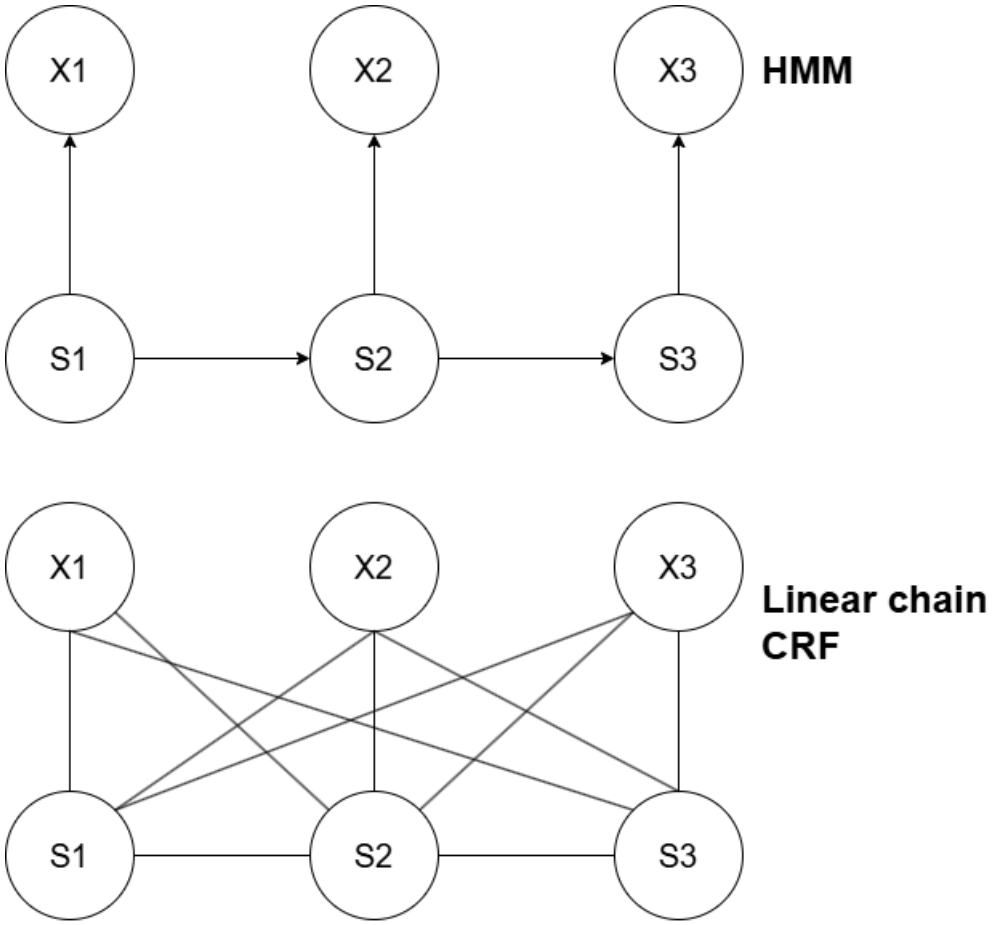

Figure 2.5: Linear chain CRF and HMM.

With CRF the number of connections increased and $X_{i=1,..,n}$ directly impact the hidden state S_i.

For further details about CRF refer to [12, 13, 11].

2.3 NER challenges

While Named entity recognition is a powerful technique, it comes with challenges and limitations. These include:

- **Language-Specific Challenges** - NER performance can vary across different languages due to differences in grammar, syntax, and entity naming patterns.
- **Ambiguity** - Words may represent different entities in different contexts (e.g., "Amazon" could be a company or a river)

- **Entity Variability** - Entities may appear in different forms (e.g., "U.S.", "United States", "USA").

These challenges are being tackled by new research and advancements in machine learning. For further reading on this research refer to: [14, 15, 16] as only a few of these challenges will be described for context.
Ambiguity can be reduced by using contextual models or additional modalities [17]. Image associated with some sentence can help with this problem on a condition that the image is related to the text.

2.4 Multimodal versus unimodal approach

Multimodal models process data $X = (x_1, ..., x_i)$ where each x_i is data sample of **different** type. In this thesis the models are *bimodal*, they process textual and visual data. Unimodal models process data $X = (x)$.
Text models were traditionally used for NER. Modern approaches try to combine multiple modalities into one model for richer representation of the entities. For tasks where visual information is relevant, multimiodal approach yields better results than unimodal [18].
The topic of multimodal named entity recognition (MNER) is covered in [19].
Figure 2.6 shows a generic multimodal model, while Figure 2.7 shows unimodal model.

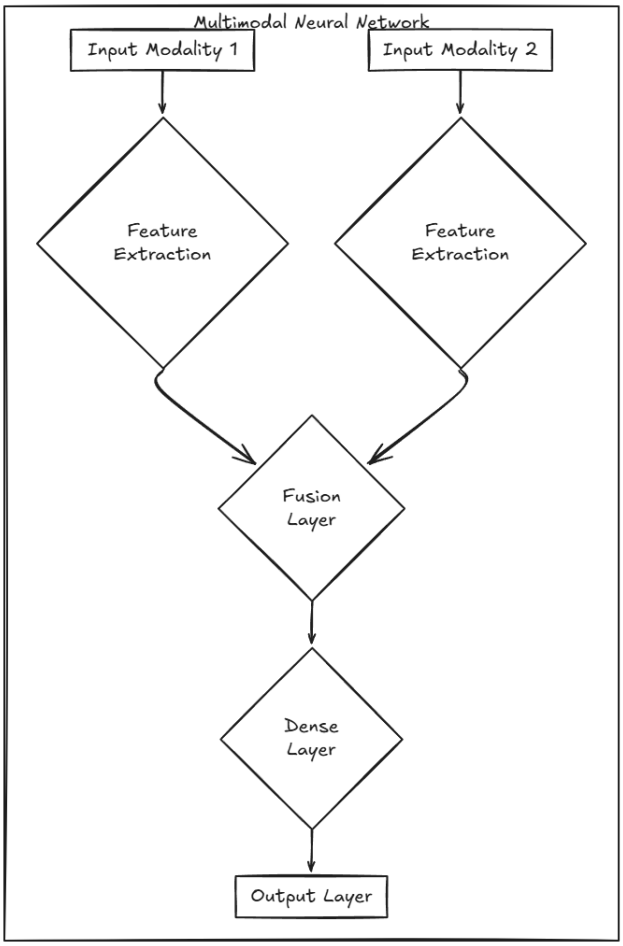

Figure 2.6: Multimodal model

Input Modality 1 is for example the text data. *Input Modality 2* can be image data. *Feature Extraction* is performed by models which extract information from raw data. *Fusion Layer* is a layer that fuses together features from multiple modalities. This layer can be a simple vector concatenation or more complex approach with concatenation and several layers for further processing of the fused vector. Complexity of this layer depends on the use case. *Dense Layer* is a fully connected layer (details in Chapter 4.2) which makes the final prediction.

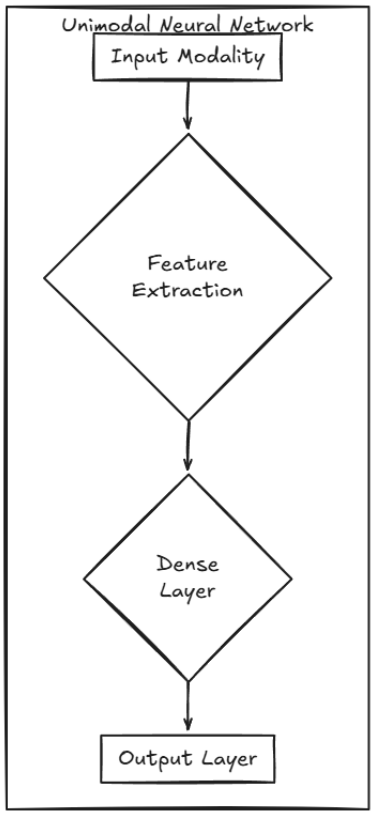

Figure 2.7: Unimodal model

Purpose of the layers is them same as in Figure **??**.

Relevant work 3

This chapter briefly explain previous work on MNER.

3.1 Named entity and relation extraction with multi-Modal retrieval

The goal of this article was to leverage relevant image information to improve performance of NER and relation extraction (RE). This paper produced a framework called MoRe.

> "MoRe contains a text retrieval module and an image-based retrieval module, which retrieve related knowledge of the input text and image in the knowledge corpus respectively. Next, the retrieval results are sent to the textual and visual models respectively for predictions. Finally, a Mixture of Experts (MoE) module combines the predictions from the two models to make the final decision" [20].

The last sentence in this citation is crucial; instead of fusing the modalities together into one prediction, as shown in Figure 2.6, each model predicts *named entities* from a given modality. MoE module is then trained to correctly combine these predictions together for the optimal performance. Figure 3.1 shows this approach in a clear way. MoE module accepts the *conditional distribution* from each modality, $P_{\theta_z}(y|x, I, Z_t)$ for text and $P_{\theta_z}(y|x, I, Z_I)$ for images and makes the **final prediction** $P(y|x, I)$[1]. The authors demonstrated through empirical evidence that integrating knowledge from text-based and image-based retrieval modules significantly improves the performance of MNER and RE tasks.

[1] Z_t and Z_I is extracted knowledge from Text Retrieval System and Image Retrieval System respectively which are concatenated with input *sentence* x.

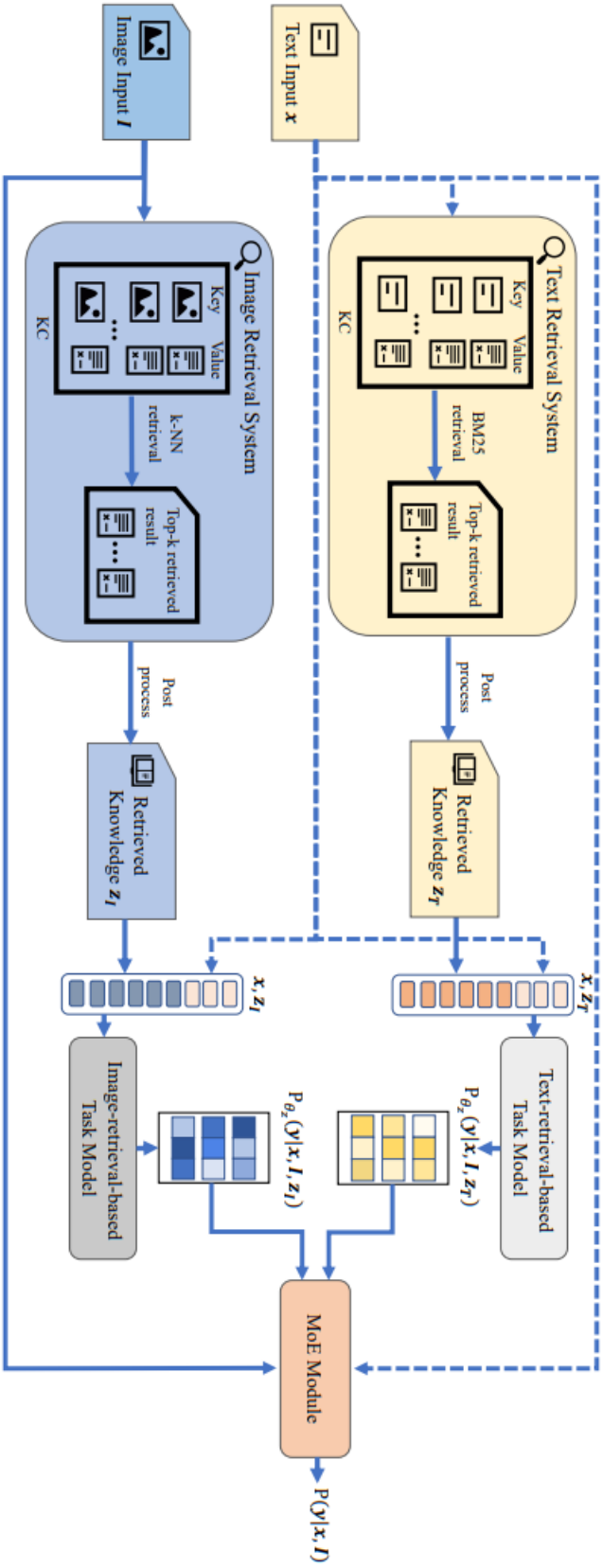

Figure 3.1: MoRe architecture [20].

3.2 ITA: Image-text alignments for multi-modal named entity recognition

The authors proposed ITA, **I**mage-**T**ext-Alignment framework.

> ITA converts an image into visual contexts in textual space by multi-level alignments [21].

They process the image and convert it to textual representation. The process is shown in Figure 3.2. This approach removes the problem of working with raw image data (pixels) in MNER but introduces problems with visual data processing. This approach is fundamentally different from their later work [20] where they opted to work with image data for disambiguation.

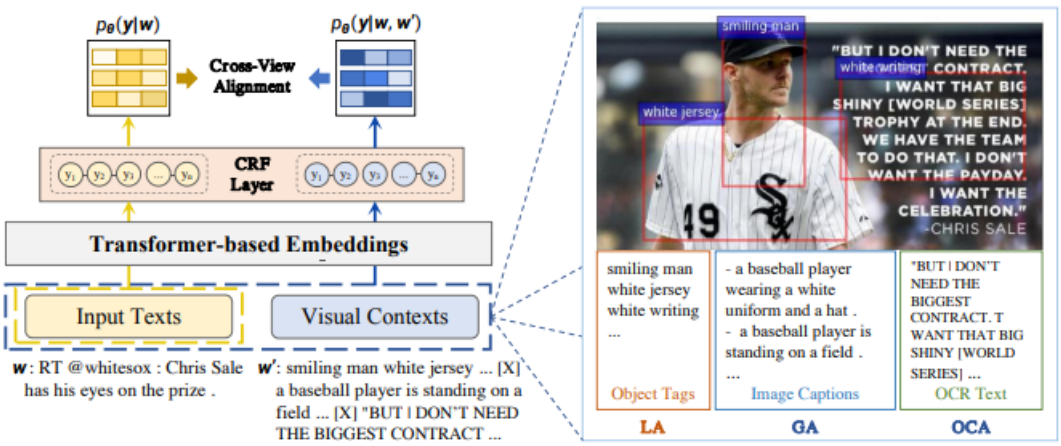

Figure 3.2: ITA architecture [21].

ITA aligns images into object tags, captions, and OCR text, treating them as visual context. They are fused with input text and processed via transformer-based embeddings. The cross-view alignment module minimizes the distribution gap between cross-modal and text-only representations.

Workflow of ITA can be summarized as follows:

1. Object detector extracts object tag and corresponding image regions (orange and red boxes in Figure 3.2).
2. Image captioning model predicts image captions (blue box).
3. OCR is used to read text in image (green box).

4. Input text and extracted visual context is passed through the network and final prediction is made.

3.3 Improving multimodal named entity recognition via entity span detection with unified multimodal transformer

The authors propose a multimodal interaction module to obtain both image-aware word representations and word-aware visual representations. Prior models generated word representation insensitive to the visual context. Another shortcoming the authors noted is that there is no mechanism to correct bias introduced by the image. This bias makes the model focus only on the main entity in the picture and ignore the rest [22].
Unified Multimodal Transformer model was introduced to solve these issues. The architecture is shown in Figure 3.3.

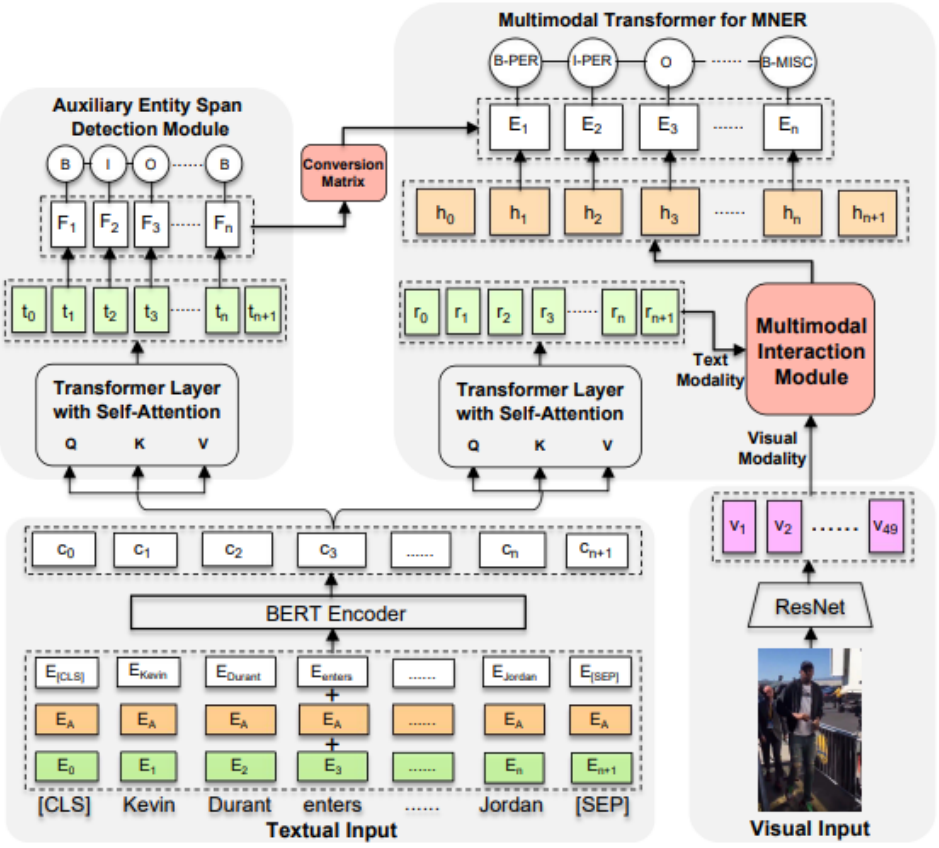

Figure 3.3: Unified Multimodal Transformer architecture [22].

Starting from the bottom, the model first extracts *contextualized word representation* and *visual block* representation from the input sentence and image respectively using BERT and ResNet. On the right side of the architure, self attention layer is used to extract *textual hidden representations*, denoted as $r_0, .., r_{n+1}$. This representation is then used in the *Multimodal Interaction Block* together with visual block to capture the cross modality dynamics, denoted as $h_0, ..., h_{n+1}$. The cross modality features are then passed to the CRF layer to produce label for each word.
To address the visual bias, a text-only module ESD (left side of the architecture) with another CRF for span label prediction. A conversion matrix encodes dependency relations between corresponding label dependencies from ESD to MNER, so that the entity span prediction from ESD has influence on the final MNER label predicted for each word [22].

Machine learning fundamentals

4

This chapter describes fundamental ideas behind machine learning.

4.1 Basic building blocks

The purpose of this section is to explain how (on the abstract level) any machine learning model acquires knowledge in supervised learning environment. The following subsections introduce simple models on which these ideas can be described.

4.1.1 Loss function

Loss function is a function that maps an *event* to a real number representing a cost associated with the event. In the training process the goal is to minimize this function by adjusting model parameters θ.
For linear regression an example of loss function can *mean squared error* (MSE).

$$J(\theta) = \frac{1}{n} \times \sum_{i}^{n} (Y^i - Y^i_{predicted}))^2$$

MSE is a smooth function. This means that it can be used as a part of backpropagation. Generally, any function that is *differentiable* can be used as a loss function - this is a strict requirement for backpropagation [23].
In this thesis, *Negative Log Likelihood* and *Cross-entropy* loss function will be used [24].

4.1.2 Linear regression

Linear regression is a statistical model that assumes linear dependency between the *dependent* variable Y and one or more[1] explanatory variable X and is used to predict continuous numerical values based on the input features.
Regression as a machine learning task is not related to this thesis, therefore it will not

[1] when $X = (x_1, x_2, x_3, ..., x_n)$ the model is referred to as generalized linear model (GLM).

be explained it in detail. The main distinction from classification is that the predicted samples are not inherently discrete (as is always the case with classification). To formally describe the difference, regression can be seen as:
$model(X, \theta) => Y \in R$
whereas classification can be seen as:
$model(X, \theta) => Y \in N$
where θ is model hypothesis (i.e. the parameters[2]). Model can be formally described as:
$Y = \theta^T \times X$ where $\theta = (\theta_0{}^3, \theta_1, ..., \theta_n)$. How to correctly place the regressor into given problem space is the task of *least squares method* ([25]).

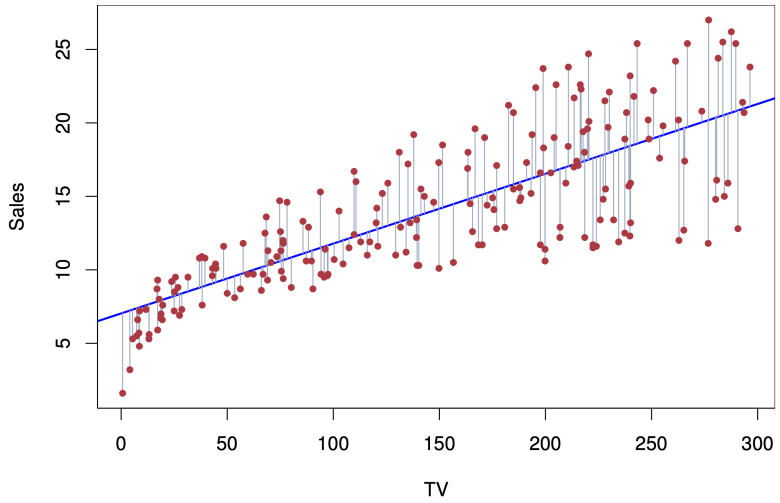

Figure 4.1: Linear regression with a regressor [26].

Red points are the data samples, blue line is the regressor. MSE is used to find the optimal position of the regressor.

4.1.3 Logistic regression

Logistic regression is a statistical model for binary classification tasks. The model models log-odds of an event (dependent variable Y) as *linear combination* of one or more independent variables (X).
An explanation of a logistic regression can begin with an explanation of the standard logistic function. The logistic function is a *sigmoid function* (shown in Figure 4.2), which takes input t and outputs a value $y \in < 0, 1 >$ [27]. This function can be used to model probability of class Y for data sample X.
Logistic regression model can be declared as: $P(Y = 1|X) = \sigma(\theta^T \times X)$

[2] For linear regression its 2, θ_0 and θ_1, bias and slope respectively

[3] θ_0 is a virtual parameter that is always equal to 1. It's point is to align X and θ for matrix multiplication.

$\sigma(T)$ is a sigmoid function defined as:

$$\sigma(T) = \frac{1}{1 + e^{-T}}$$

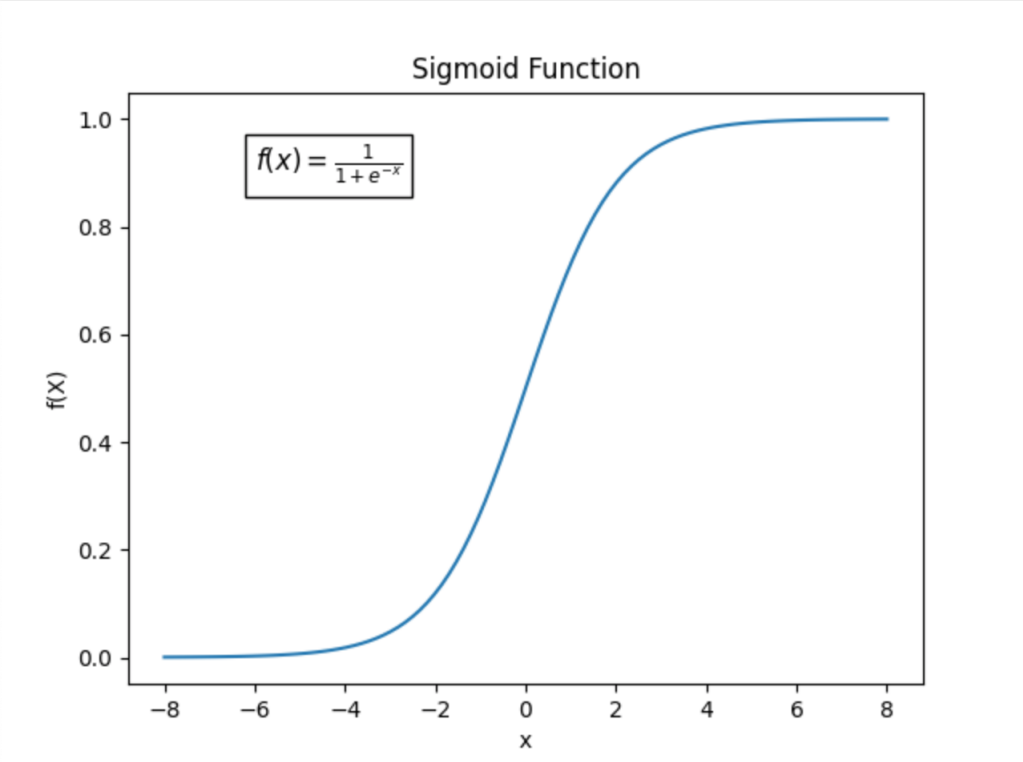

Figure 4.2: Sigmoid function [28]

By extending the regression models with a sigmoid function, these models can now be used for classification. Sigmoid function compresses any real-valued output into the range $< 0, 1 >$ (probability distribution).
Activation functions are necessary not only to project values into probability distribution but to introduce **non-linearity** to deep neural networks. Without activation function any neural network could solve only linearly separable problems [29].

4.2 Multilayer perceptron

Linear regression with activation function on their own can only solve binary problems. For NLP and other fields this is not enough since the nature of the problems is rarely binary. Multilayer perceptron (MLP) is a model which has:

- Input layer - takes input data and passes them into the first hidden layer.

- Hidden layer - performs object representation.
- Output layer - predicts the final class.

MLP is a **feedforward neural network**, meaning that there are no cycles in the graph (in contrast to **recurrent neural networks** which uses cycles). Furthermore, the MLP showed in Figure 4.3 is a **dense** model. This means that each neuron in a layer i recieves signal from each neuron in previous layer $i-1$.
Dense model captures complex nonlinear relationship between input x and desired output y. With that advantage it is necessary to state that they suffer more from "learning" noise than sparse neural networks [30]. Forward pass in an MLP can be formally expressed as:

$$h^1{}_i = \sigma^1(\sum_j^n \omega_{ij}^1 \times x_j + b_i^1)$$

for input layer. $h^1{}_i$ refer to units in the first (input) layer, σ^1 is activation function of the first layer. The body of the activation function is a sum over all connections (synapses) and their weights multiplied with the input data.
For hidden layer:

$$h^k{}_i = \sigma^k(\sum_j^n \omega_{ij}^k \times h_i^{k-1} + b_i^k)$$

The only difference is that instead of using raw input data x_j, activation of previous layer h_i^{k-1} is used. For output layer the expression is the same; still a linear combination of nonlinear output of previous layer with weights stored in synapses.

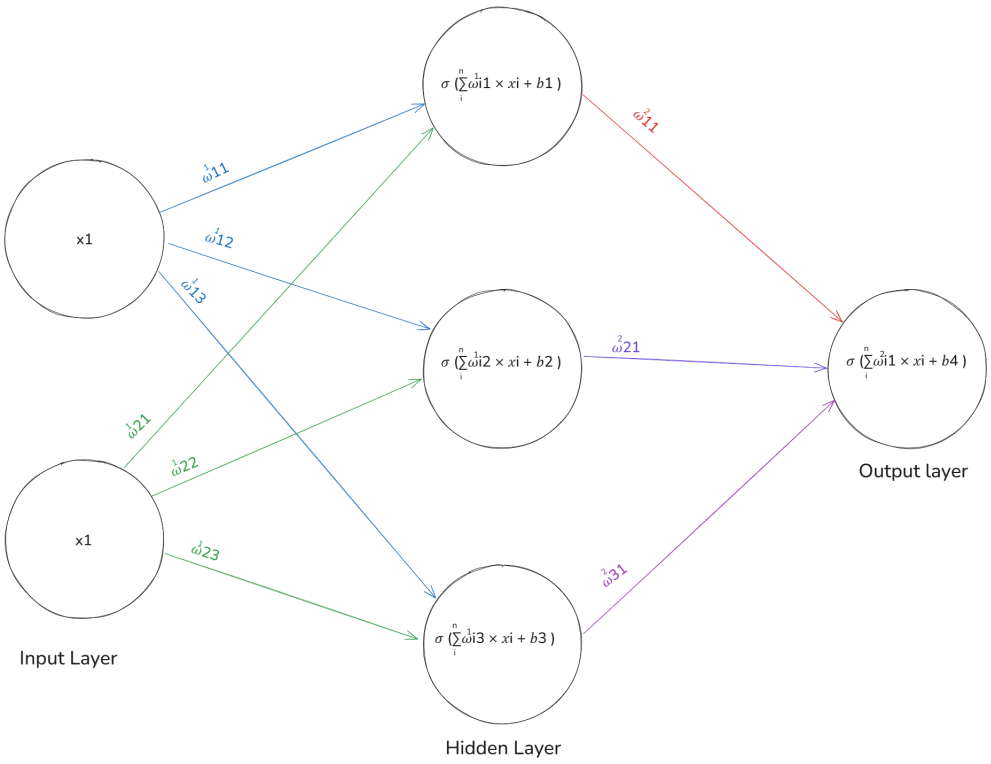

Figure 4.3: Forward pass in multilayer perceptron.

MLP with one input layer, denoted as x_1, x_2, one hidden layer with 3 neurons and one output layer. This picture shows quite nicely that that a neuron is just a *linear regression* and an activation function σ.

The forward pass produces desired output Y. In case of NER Y is a vector of predicted entities.

4.3 Optimization

Optimizing the neural network is a key step in the training process. Without it, the network would not gain knowledge from provided information.

4.3.1 Backpropagation

Backpropagation is the most common algorithm used in training phase to compute *gradient* of a *loss function* for each layer with respect to the weights of the network for a single input–output example. This computation is done from *output layer* to *input layer*, one layer at a time to avoid redundancy.
The algorithm is often tightly coupled with another algorithm which is used to up-

date parameters, commonly stochastic gradient descent which is described further. The algorithm can be broken down to these following steps:

1. forward pass - pass the input data X to recieve output $Y_{predicted}$.
2. compute loss - compare $Y_{predicted}$ with Y_{true} to compute *loss* value.
3. calculate gradients of the loss function with respect to each weight and bias

The gradient calculation uses chain rule for efficient computation. Since the gradients are calculated from output layer L there is no need to compute gradients more than once for each pass or to compute intermediate state. That is because layer L_{i-1} only affects *loss* of layer L_i and it does so linearly.
[31] describes the entire backpropagation process in detail.

$$Cost = C(Y, f^L(\omega^L \times f^{L-1}(\omega^{L-1} ... f^1(\omega^1 \times X))))$$

The formula above computes the *loss* for one forward pass (given target value Y). This value is then propagated through the network to compute error of each neuron. Error of output layer is computed as a *partial derivative* of the loss function with regards to output layer:

$$OutputCost = \frac{\partial C}{\partial a^L}$$

where a^L is the output of the activation function in layer L. To compute error of weights of previous (hidden) layer $L - 1$:

$$\frac{\partial C}{\partial \omega^{L-1}} = \frac{\partial C}{\partial a^L} \times \frac{\partial a^L}{\partial z^L} \times \frac{\partial z^L}{\partial a^{L-1}} \times \frac{\partial a^{L-1}}{\partial z^{L-1}} \times \frac{\partial z^{L-1}}{\partial \omega^{L-1}}$$

this is the application of chain rule method which propagates all the way to the input layer. The weighted input of layer L is denoted as z^L. Figure 4.4 visualizes backward pass through one singular neuron. This backward pass computes gradient based on the error of this neuron.
Gradients are used to adjust model parameters. This can be done for example with gradient descent (GD) algorithm which will be explained further alongside with some variations of GD, stochastic gradient descent (SGD) and AdamW. SGD and primarily AdamW played immense part in conducted experiments. For more information and properties of this algorithm refer to [23, 29].

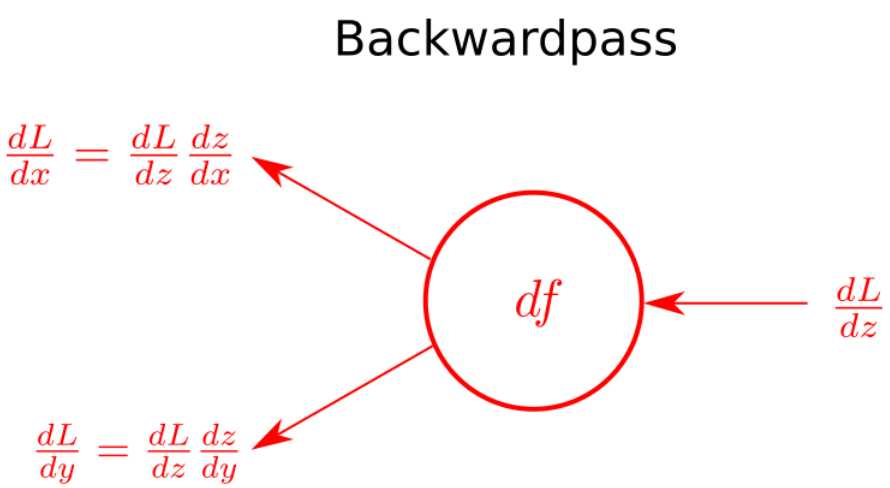

Figure 4.4: Backward pass in neuron [32].

Notation is different, z_j is an activation function.

4.3.2 Gradient descent

The goal of this algorithm is to determine a set of parameters θ with minimal error between the predicted $Y_{predicted}$ and actual values Y_{true}. This is accomplished by computing the *gradient* of the loss function with respect to the model parameters and updating the parameters accordingly.
Gradient of multi-variable function F at point ω_i is a partial derivation with respect to it's input. This implies that F or any loss function has to be smooth.
GD is based on an observation that $F(x)$ decreases **fastest** if the negative value of gradient $\Delta F(\omega_i)$ is subtracted from ω_i. This can be expressed as:

$$\omega_{i+1} = \omega_i \times -\alpha \times \Delta F(\omega_i)$$

This is the connection between gradient descent and backpropagation. Gradient descent updates parameters of model with gradients computed by backpropagation algorirthm. Parameter α is called *learning rate* and **needs** to be set correctly.[4] The issues of gradient techniques are described further in the chapter.Figure 4.5 shows gradient descent iterations and how the computed *loss* is smaller with each iteration when α is set correctly.Figure 4.5 visualizes gradient descent. The visualized loss function is very simple - this "bucket" shape is ideal for GD algorithm because it can reach global optima without getting stuck in any local optima. For further details about this algorithm and it's properties and prerequisites refer to [33].

[4]There is no one single value that would be a perfect learning rate. This value must be empirically chosen based on model's architecture and loss function.

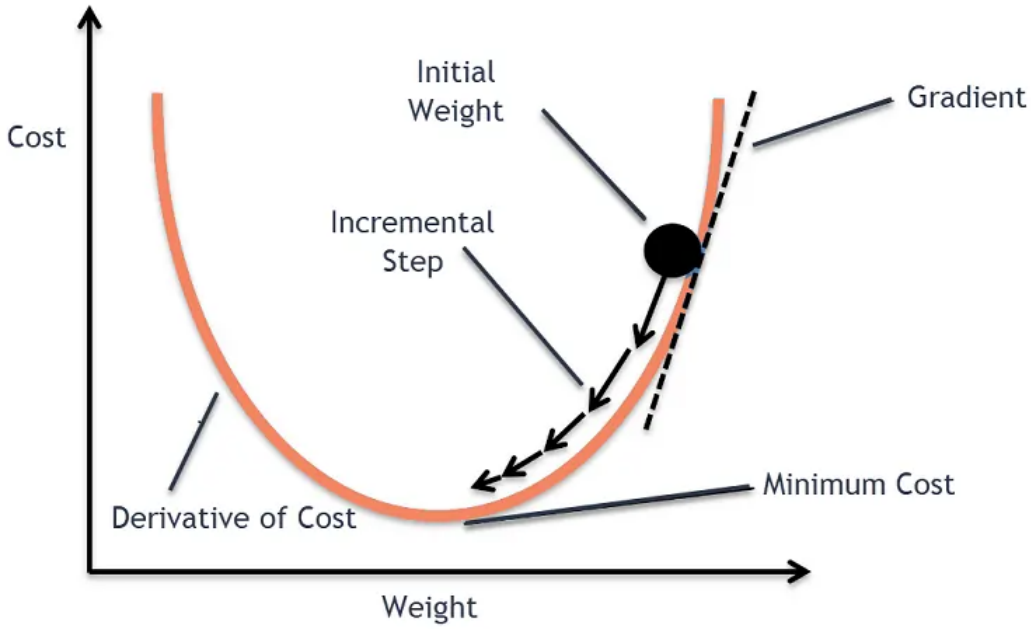

Figure 4.5: Gradient descent visualization [34].

Loss is sometimes refered to as "cost". As is this the case on this Figure.

4.3.2.1 Stochastic gradient descent

Gradient descent computes the gradient of the loss function $F(\theta)$ with respect to the parameters θ on the entire training dataset.
SGD performs a parameter update for randomly picked sample (x_i, y_i) from training dataset [35]. This simplification is useful for datasets with large redundancy (low variability) or volume[5].

$$\omega_{i+1} = \omega_i \times -\alpha \times \Delta F(\omega_i; x_i, y_i)$$

This results in a faster (but not as smooth) convergence. This algorithm is also more efficient with regards to computational power since it does not compute gradient for all data points (x_i, y_i). The difference in convergence is shown in Figure 4.6.

[5]Training can become a bottlenck with GD. SGD can help with this.

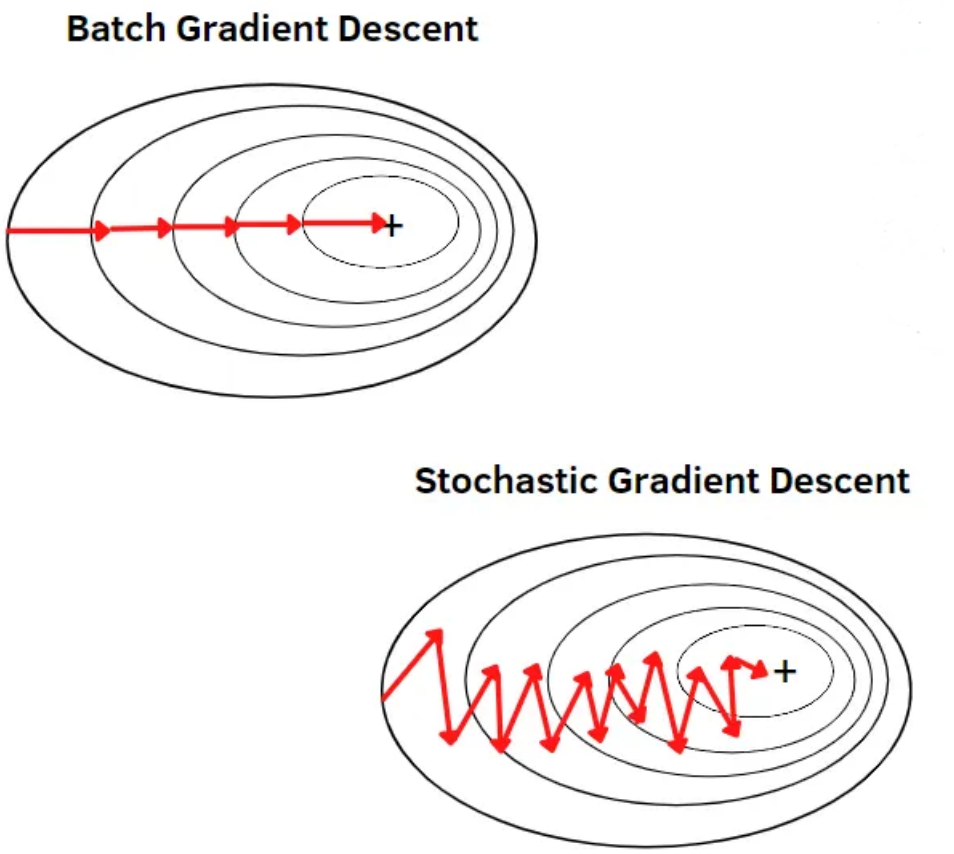

Figure 4.6: Classical gradient descent (denoted as batch gradient descent) convergence vs. SGD [36].

4.3.2.2 Momentum

Before delving into Adam and AdamW respectively it is important to mention *momentum*. As mentioned before, (S)GD suffer from getting "stuck" in local minima and not being able to "climb" out. Another issue are saddle points [37]. These points create plateaus where gradient updates are small.
Momentum speeds up converge and addresses the issue of oscillation around local minima and compensates small gradient changes in saddle points. *Classical momentum* [38] accumulates a decaying sum (with decay factor μ[6]) of the previous updates into a momentum vector m. This vector replaces the original gradient step. This improvement changes the original equation to:

$$m_i = \mu \times m_{i-1} - \alpha \times \Delta F(\omega_i)$$

$$\omega_{i+1} = \omega_i + m_i$$

4.3.2.3 Adam(W)

Momentum fixes some of the problems of GD, but there is one more. The learning rate. α is set as a constant for all parameters of the models. In complex neural network architecture, there is a need to change some parameters more than others [39]. Very influential extension of SGD is Adam and AdamW respectively. For deep explanation of Adam, refer to [40] as the algorithm will be explained very briefly here.
Adam is an optimization algorithm that adapts the learning rate α for each parameter $omega_i$. It does this by computing *momentum* and running average of squared gradients (RMSprop). The idea is that the *momentum* helps convergence and *RMSprop* helps adapt each parameter's learning rate. The issue with Adam itself is that it couples regularization term (usually L_2) to the loss function. Adding this term to the loss affects the adaptive learning rates, which can hinder optimal convergence.
AdamW [41] is an improvement on Adam algorithm that decouples *weight decay* (special form of L_2 regularization) from gradient update.

4.3.3 Struggles of gradient based learning

Some common problems associated with GD were already mentioned in the optimization section (local minima, saddle points, ...). In this section the list will be extended by other important ones which can occur.

[6] How strongly should previous iteration contribute to current update.

4.3.3.1 Overfitting and underfitting

Overfitting is a state where the model is too complex for given a dataset [42] or too many training iterations were performed. In other words, the model memorizes the data set and is not able generalize. In practictal context, the model will perform very well on training data but poorly on validation and evaluation data.
Underfitting occurs when mathematical model is not able to capture the underlying data structure. This can happen if the model itself is too simple and some parameters that would be present in a correct model are missing [42]. These states can be seen in Figure 4.7.

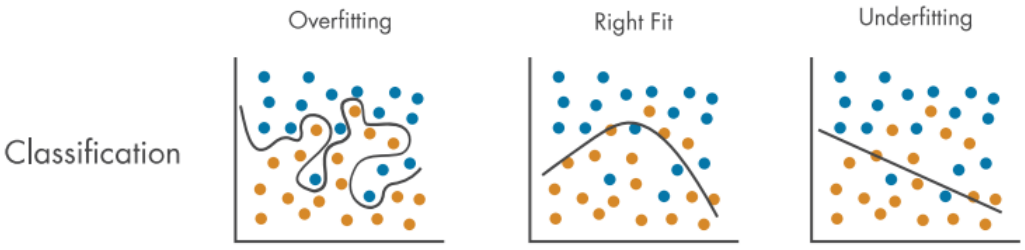

Figure 4.7: The various states of decision boundary [43].

Overfitting happens when the model memorizes the training dataset and is not able to generalize. Underfitting happens when the model was no able to generalize from training data. This means that the dataset either very poor in quality or the training was stopped too early.

4.3.3.2 Dataset imbalance

A balanced dataset is characterized by a uniform distribution of class labels. In reality, this is rarely the case. Dataset is imbalanced when one or more classes are (significantly) underrepresented compared to others. This imbalance can lead to a biased model. The classifier becomes overly tuned to the majority class and not being able to capture the significant patterns in minority classes [44].
To combat this heavy class inbalance it is absolutely necessary to adjust gradient computation. The more frequent the class, the less impact should it have in the training. The formula to compute these weights is a heuristic inspired by [45]

$$W_c = \frac{N}{(m \times freq_c)}$$

where c is concrete class, N is number of samples and $freq_c$ occurrence of c in the entire dataset. The result W is a vector of weights for each individual class. This vector is then used to adjust values of given gradients. Refer to [46] for different methods.

4.3.3.3 Exploding gradient and gradient clipping

The exploding gradient problem is a challenge encountered during the training of deep neural networks, particularly in the context of gradient-based optimization methods [47]. In backpropagation, if the gradient values would be too large and they would be propagated backwards to another layers that multiples them. This can lead to a several issues:

- Numerical instability - the gradient value wont fit into classical 64 or even 128 bit variable, causing the model to diverge.
- Overshooting - with large updates the model might overshoot minima.
- Model divergence - the model might be updating weights too aggressively. Any acquired knowledge to this point could be overwritten by large weight update.

This problem often occurs for two reasons. Complex architecture of the model and unbounded activation functions, such as ReLU, GeLU and others which are commonly used.
Recurrent neural networks famously suffered from this issue (and from vanishing gradient issue) - [48].
The solution to this problem (among other) is to *clip gradients*. The idea is extremely simple and can be expressed as:

$$g = \begin{cases} \text{clip} & \text{if } \Delta F(\omega_i) > \text{clip} \\ \Delta F(\omega_i) & \text{otherwise} \end{cases}$$

where $clip$ is a constant, for example 1.

4.3.3.4 Overwriting existing knowledge

This problem is very interesting and closely related to large weight updates. The learning rate parameter α, even if set conservatively for given model architecture (for example 1×10^{-5}, can after few epochs be too large and may start to overwrite existing knowledge in the network. This problem can be compensated by using so called *learning rate schedulers* [49]; more on them in Sec. 8.4.4.
Another more robust approach is Elastic Weight Consolidation [50].

Neural network topologies

5

This chapter describes the topologies of neural networks which will be used in the conducted experiments. It is focused on main ideas behind the topologies, not the concrete implementation details of given models.

5.1 Convolutional neural network

Convolutional neural network (CNN) is a feedforward neural network that learns features via kernel optimization. This network has been applied to process and make predictions from many different types of data including text, images and audio [7]. CNNs are tightly coupled with the term **deep learning** where each hidden layer serves a different purpose. Stacked on each other, they perform a sophisticated feature extraction. An example can be image processing. One convolutional layer can specialize in edge detection. The following layer can specialize in face detection (building from the extracted edges) and so on. CNN is comprised of three types of layers:

- Convolutional layer - consists of *kernels* which are used to connect concrete neurons to concrete regions of input signal.
- Pooling layer - simply perform downsampling along the spatial dimensionality of the given input.
- Dense layer - works the same as in any other neural network.

These layers stacked together form a CNN [51] as shown in Figure 5.1.

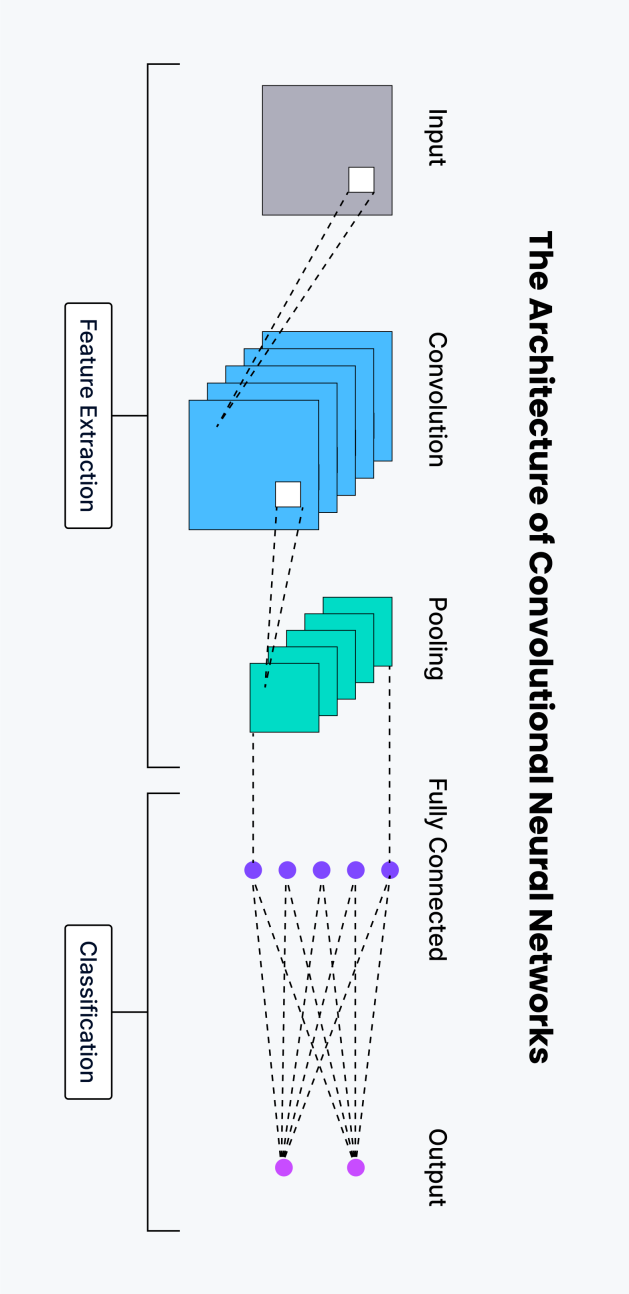

Figure 5.1: Convolutional neural network [52].

5.1.1 Convolutional layer

The layers parameters focus around the use of learnable **kernels**. When the data arrives at a convolutional layer, the layer moves each filter across the spatial dimensionality of the input.

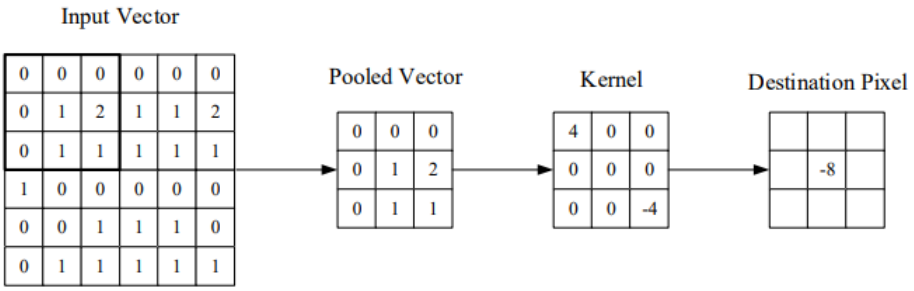

Figure 5.2: Kernel demonstration [51].

Kernel moves along the input matrix and computes values for given areas (pooled vectors). These kernels together with pooling layers are used to extract specific features (edges, shapes etc).

A dimensionality after applying kernel is computed as:

$$W' = \frac{input\_size - kernel\_size + 2 \times padding}{stride} + 1$$

where $input\_size$ is a 2D vector of input dimension and kernel_size is a size of kernel. If W' is not an integer the that means that *stride* was not set correctly. Padding is used when the kernel "moves out" of input vector (usually the padding strategy is to add spacce) and stride is the size of step on the input vector.

5.1.2 Pooling layer

There are usually two main pooling methods, *average pooling* and *max pooling*. Pooling layers are defined as a relatively small matrix with dimension $(width, height)$, for example $(2, 2)$.
The matrix then travels around feature map computed by kernels and reduces dimensionalityby applying a reduction rule.
Max pooling would use **maximum pooling** strategy. It picks the maximum value from the $(width, height)$ subspace while **average pooling** would take the average value [53]. The difference in approach can be seen in Figure 5.3.

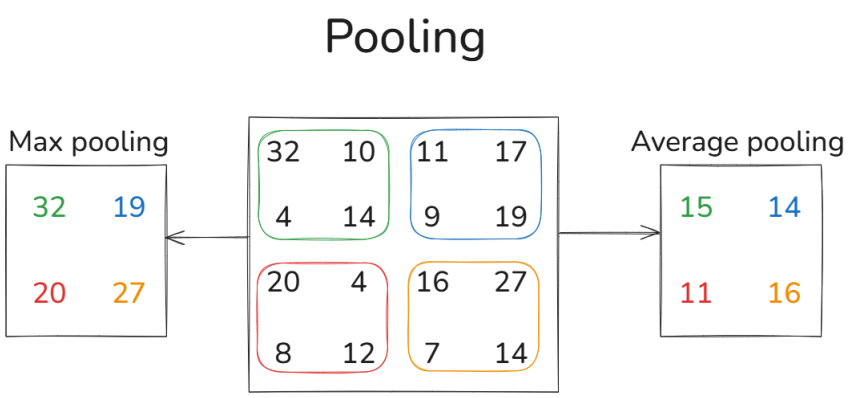

Figure 5.3: Pooling strategies.

5.2 Recurrent neural network

Recurrent neural networks (RNN) are feedforward neural networks augmented by the inclusion of edges that span adjacent time steps, introducing a notion of time to the model. Like feedforward networks, RNNs may not have cycles among conventional[1] edges [54]. However, edges that connect adjacent time steps, called recurrent edges, form cycles. This includes cycles of length one that are self-connections from a node to itself across time [54]. These cycles are important as they represent "memory" of the network and allow it to process sequential data better than standard feedforward neural networks.
RNN can be expressed by these two equations:[2]:

$$h^t = a(\omega^{hx} \times x^t + \omega^{hh} \times x^{t-1} + b^h)$$

$$y^t = \sigma(\omega^{yh} \times h^t + b^y)$$

The first equation is the pass between recurrent layers (hidden states). h^t is a hidden node value at time t, x^t is the input at time t and ω^{hx} are the weights between hidden node and input (i.e the synapse) and ω^{hh} is the *recurrent connection*. Activation function a can be any non-linear function. RELu, tanh or GELu are commonly used. b^h is the bias of hidden node h. The second equation the activation of the neural network.
RNNs suffer from a lot of problems. Gradient explosion was mentioned in Sec. 4.3.3.3. Another issue[3] is *vanishing gradient* which has the same cause but the gradient value "vanishes" to 0. Vanilla RNNs are rarely used for any task and other, more

[1] Recurrent edges span through different timesteps. Usually from timestamp t to $t + 1$. They do not span in the same timestep t.
[2] Recall linear regression and softmax; the equation is similiar but includes recurrent edges
[3] For more information about the problems with RNNs check [55].

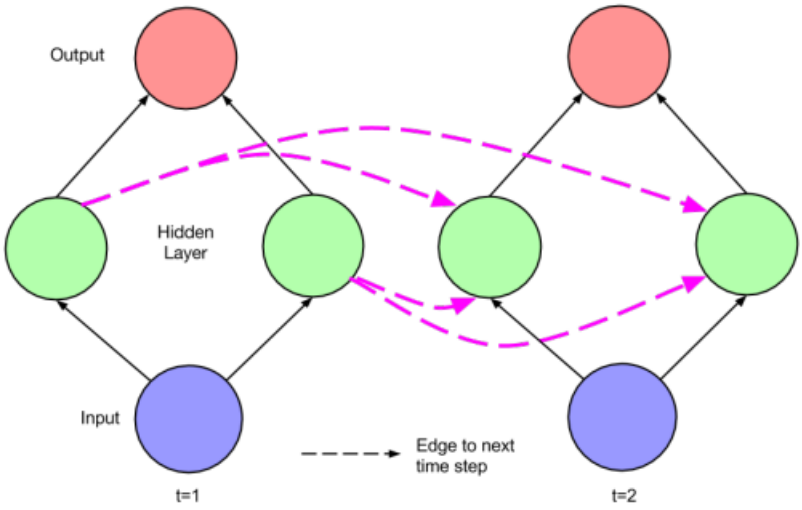

Figure 5.4: Unfolded RNN with connections [54].

The pink arrows recurrent connections between step $t = 1$ and $t = 2$.

robust variations are preferred, such as GRU [56] or LSTM [57] since they generally perform better.
LSTM and its extension - BILSTM - will be mentioned further as it was used widely in conducted experiments for sequence modeling. I will not mention GRU since it was not used.
Figure 5.4 shows RNN and connections between layers.

5.2.1 Long-Short Term Memory

Long-Short Term Memory (LSTM) is a modern take on RNN architecture. In [57] Hochreiter the memory cell, a unit of computation that replaces traditional nodes in the hidden layer of a network. With these memory cells, networks are able to overcome difficulties with encountered by earlier recurrent networks [54] - such as vanishing gradient. Instead of ordinary nodes, LSTM replaces them with *memory cells* denoted as c. These cells are nodes with self-connected recurrent edge of fixed weight. This means that gradient can pass through these cells many times without vanishing or exploding.

5.2.1.1 LSTM elements

LSTM consists of these elements (The notation uses vectors, meaning that g contains values of all input nodes in given layer.):

- *Input node* - this unit, denoted as g takes activation of input layer x^t and previous hidden state h^{t-1}. This is nothing new from a standard RNN. Weighted sum of these parameters is propagated forward to the network.
- *Input gate* - gating is a distinct feature of LSTM. Gate is a *sigmoid* unit that takes input x^t and h^{t-1}. Gate has values between 0 and 1; if the value is 0 then no flow is passed through. Gate value of 1 means that the entire flow is passed. Value of input gate is further denoted as i.
- *Internal state* - this is node in every memory cell with linear activation function. This node has self-connected recurrent layer with fixed unit weight. This state is updated as:

  $$s^t = g^t \times i^t + s^{t-1}$$

  where s^t is the internal state at time t, g^t is the input node at time t, s^{t-1} is the internal state at time $t-1$.
- *Forget gate* - these gates f were introduced as a means of "flushing" the contents of internal state. This is useful for long running networks as it can prevent overfitting and other related issues. With these gates, the internal state is updated as:

  $$s^t = g^t \times i^t + f^t \times s^{t-1}$$

  forget gate at time t is denoted as f^t.
- *Output gate* - The value of memory cells is the value of internal state s and the value of output gate o (computed from x^t and h^{t-1}). Value of hidden layer at t is then computed as:

  $$h^t = \sigma(s^t) \times o^t$$

  where σ is an activation function, for example *tanh*. o^t is the output gate at time t.

LSTM memory cell with operations can be seen in Figure 5.5.

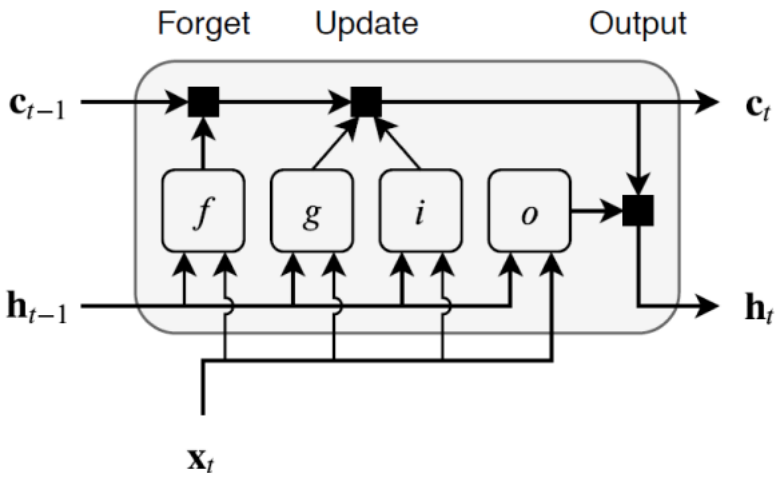

Figure 5.5: LSTM memory cell [58].

Memory cell, internal state is denoted as c.

5.2.1.2 Bidirectional LSTM

Enhancement of LSTM network proposed in [59]. Information from future, $t+1$ and past $t-1$ impacts the current input t. This is different from previous implementations because only past information impacted current input. This trait is very useful for sequence labeling tasks (such as NER) and NLP in general since it allows a model to better understand language. BILSTM architecture can be seen in Figure 5.6.

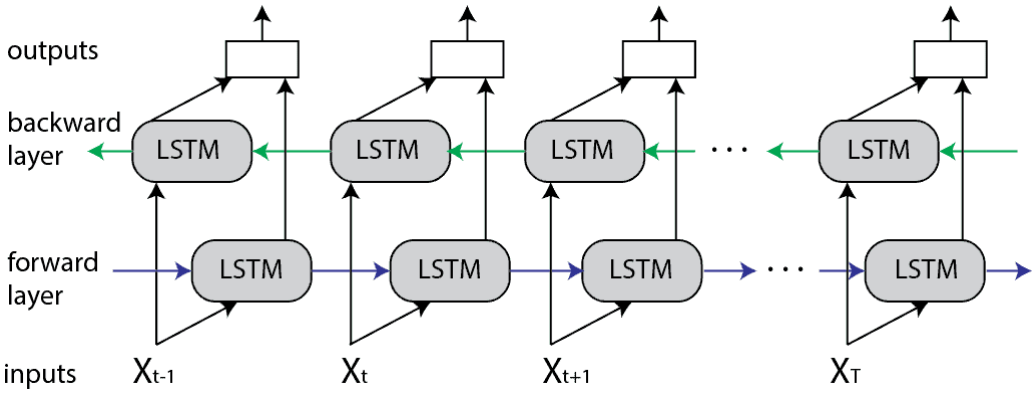

Figure 5.6: Bidirectional LSTM [60].

The backward propagation adds context for better language understanding (in case of NLP).

5.3 Transformers

Transformer is a deep learning architecture that is based on multi-head *self-attention mechanism*. Self-attention idea was first proposed in [61].
Before delving deeper into transfomers, it is important to understand what is sub-optimal with RNN's. The sequential computations. Recall the hidden state h^t which depends on h^{t-1} and so on. Although this problem has been addressed by some clever optimization techniques ([62, 63]) the sequential[4] nature remains. Transformers process data in parallel. The last major architecture difference worth mentioning is that transformers do not have recurrent edges meaning that they do not suffer from related problems.

5.3.1 General architecture

Most competitive neural sequence transduction models have an encoder-decoder structure. The **encoder** maps an input sequence of symbol representations $x = (x_1, ..., x_n)$ to a sequence of continuous representations $Z = (z_1, ..., z_n)$. Given Z, the **decoder** then generates an output sequence $Y = (y_1, ..., y_m)$ of symbols one element at a time. At each step the model is auto-regressive, consuming the previously generated symbols as additional input when generating the next [61].
In simpler terms, encoder stack and the decoder stack each have their corresponding embedding layers for their respective inputs X and Z. Finally, output layer generates the final output Y.
Encoder
Encoder is composed of stack of N identical layers. Each layer also has two sub-layers with multi-head self-attention mechanism and feedforward neural network. Each of the two sub-layers have a residual connection[5] with normalization. Function of encoder can be summarized [64] as this:

1. Self-attention layer computes relationship between all words
2. Feed forward network applies non-linear transformation
3. Apply residual connection and layer normalization for stabilization

Encoder only architecture (BERT specifically) is widely used in conducted experiments for feature extraction of text features. Multi-head attention blocks were used in proposed architecture in chapter 8 for fusing modalities.
Decoder

[4] Meaning that to compute hidden state h^t, h^{t-1} needs to be computed first.
[5] Residual connection is the technique used to retain information by skipping layers. For example output of layer with residual connection can be computed as $Y = ((\omega^i \times X) + X) + b^i$

The decoder is composed of a stack of N identical layers. In addition to the two sub-layers, which are identical to the sub-layers of encoder, the decoder inserts a third sub-layer. This sub-layer performs multi-head attention over the output of the encoder stack.
The attention layer which accepts output of the encoder stack is modified to prevent positions from attending to subsequent positions. This means that output on position i is only affected by tokens less than i, i.e the tokens before the currently predicted token. The encoder-decoder architecture can be seen in Figure 5.7.

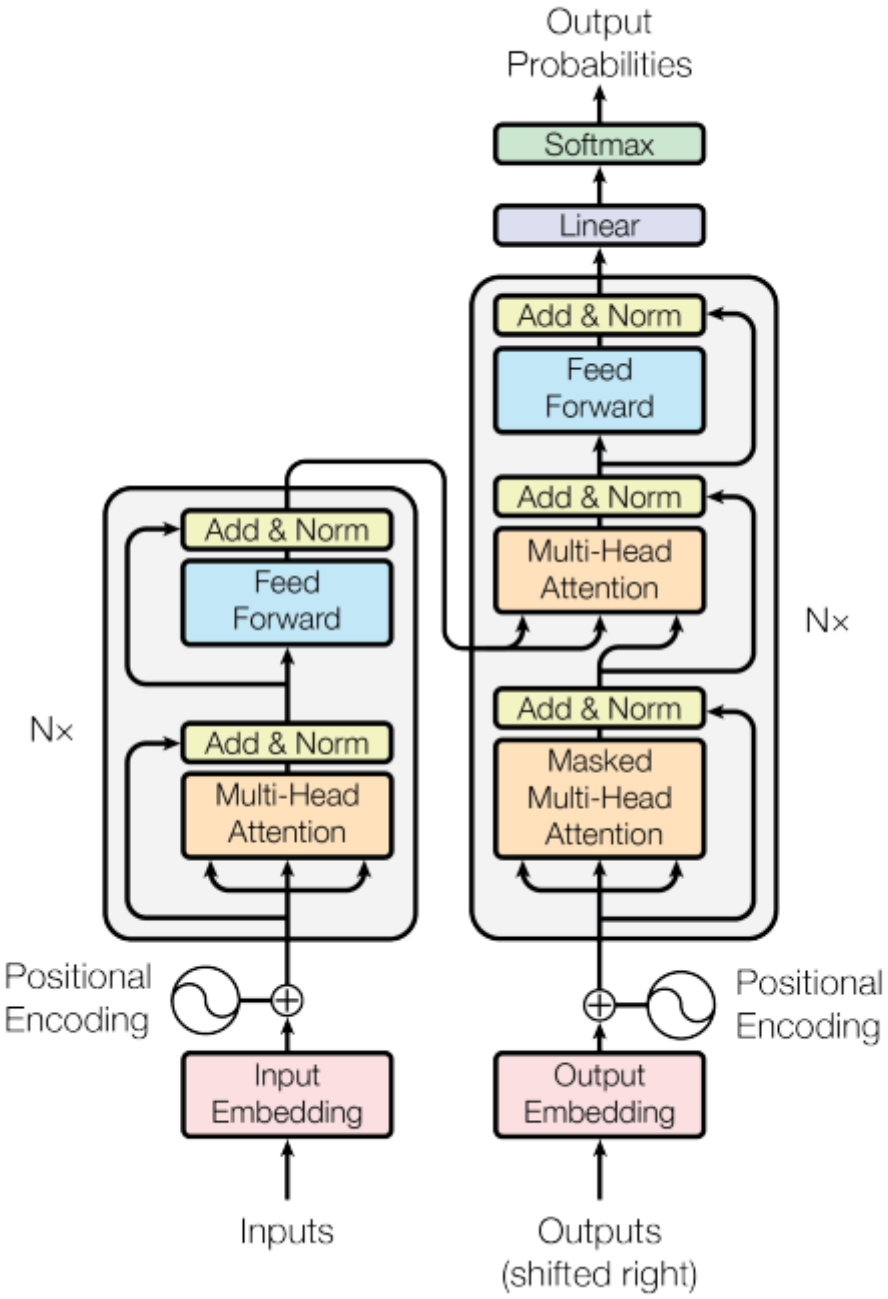

Figure 5.7: The Transformer - model architecture [61]. The left block represents the *encoder*. *Decoder* is on the right.

Positional Encoding captures the location of each token in the sequence.
One pass through this encoder decoder outputs a probability distribution of next token in the sequence.

5.3.1.1 Tokenization and encoding

There are several ways for the tokenization, which can be as simple as *space tokenization*[6] or more complex that rely on *rules*.
A commonly used tokenizer algorithm is *Byte-pair encoding* introduced in [65] which can operate on raw text input or it can be used with pre-tokenization. This input is then used as an input to the *encoder layer*.

5.3.1.2 Attention, why it matters?

The purpose of attention block is to encode contextual information to tokens. In other words, to encode token's surroundings into the token.
Before **contextual embedding** (which is what BERT does for example) the dominant representation of words was **word-embeddings**, for example word2vec [66]. The main problem with word-embedding is that it **ignores** contextual information (the *meaning* derived from words surrounding). In practice, this means that for the sentence: *"I **left** the house early and walked **left** to my favourite pub."* the word **left** will be represented by the same *embedding*.
Contextual representation (i.e attention based) fixes this problem by learning **sequence-level semantics** by considering the sequence of all words in the document. In practice, this means that the word **left** would be represented by **different** embedding due to the surroundings.
The authors of [61] define attention as:

$$Attention(Q, K, V) = softmax(\frac{Q \times K^T}{\sqrt{d_k}}) \times V$$

where Q, K, V are called **Query vector**, **Key vector**, **Value vector**. Vectors Q[7] and K are used to compute *attention pattern*. To breakdown the formula:

$$AttentionPattern = \frac{Q \times K^T}{\sqrt{d_k}}$$

computes the *attention pattern*, i.e how relevant the words are to each other (for example adjectives are more relevant to nouns then verbs). d_k is used for normalization to prevent very high or low values. Attention values are then converted to probabilities via *softmax* function, this ensures that all attention weights are in the interval $(0, 1)$ and introduces non-linearity.
Vector V contains rich contextual information about the token and is multiplied by

[6] Splitting the input into token by whitespaces. This of course has a lot of issue for exotic languages (Chinese) and does not handle punctuation well.

[7] To compute vector Q and K a transformation of original embedding is needed, such as: $Q = E \times W_Q$ and $K = E \times W_K$ where W_Q and W_K are trainable weights. Same logic applies for V.

attentions, $softmax(AttentionPattern) \times V$.
The vectors which are the results of this multiplication are then summed together to produce the final *attended representation*.

5.3.2 Vision transformers

Transformers found widespread usage in NLP tasks. In [67] the authors experimented with attention blocks for image processing. The authors replaced convolution layers (i.e kernels) with self-attention blocks which yielded competitive results with other deep neural networks (such as ResNet [68]) while requiring fewer flops[8].
In [69] vision transformer (ViT) was introduced as an alternative to CNN architecture. The model follows *encoder* only architecture.
In Section 5.3.1 tokenization and encoding was mentioned which is required for encoder layer. ViT uses tokenizer which splits the image into **patches**, a matrix with shape:

$$Patch = (P, P, C)$$

where P is the size of patch and C is the number of channels. The patches **do not** overlap. This patch is then flattened to a vector $v^{P \times P \times C}$.
This vector is then used as an input to a *trainable* linear layer that projects in to fixed-sized space. Output of this layer is a **sequence** of patch embedding which are used in the encoder layer.
The architecture of ViT and the entire process described above can be seen in Figure 5.8.

[8] Floating point operations per second

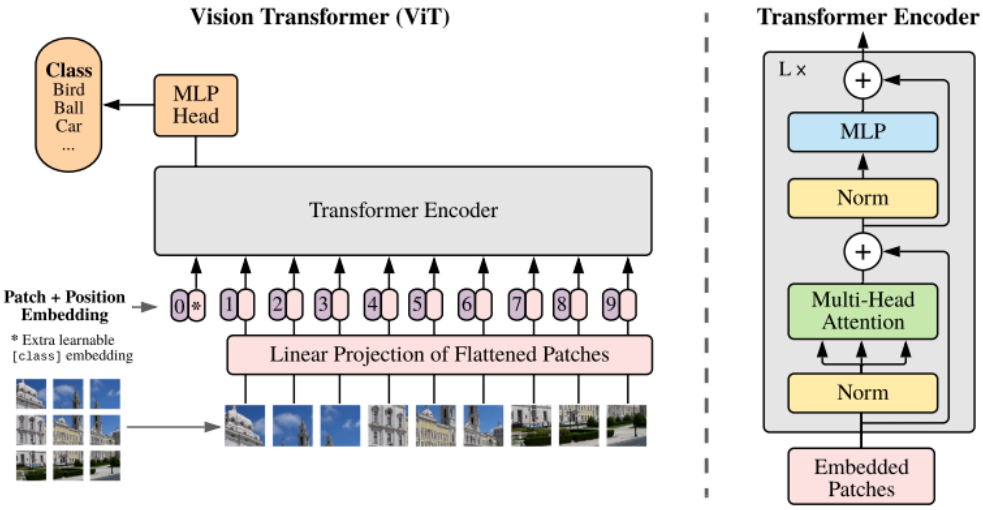

Figure 5.8: Architecture of ViT [69].

As can be seen in the picture, the main task the authors solved is how to *tokenize* images for transformer encoder. Otherwise the architecture of the encoder is similar to the one presented in [61].

5.3.3 BERT

Bidirectional Encoder Representations from Transformers (BERT) was introduced in [70]. It is designed to understand the context of words in a sequence by leveraging the bidirectional capabilities of the Transformer architecture.
BERT uses a masked-language modeling (MLM) pretraining objective. To quote the authors of [70]

> *"The masked language model randomly masks some of the tokens from the input, and the objective is to predict the original vocabulary id of the masked word based only on its context. Unlike left-to right language model pre-training, the MLM objective enables the representation to fuse the left and the right context, which allows us to pretrain a deep bidirectional Transformer."*

Unlike previous unidirectional methods, BERT processes text bidirectionally (same improvement as in the BILSTM architecture). This allows the model to understand the full context of the sequence. BERT is an **encoder-only**[9] transformer architecture.
The differences which are needed to be point out from generic Transformer are:

[9] same as ViT since both models are designed to understand context i.e extract features, not generate sequences.

- **Tokenization** - BERT uses WordPiece [71] tokenizer to effectively represent words.
- **Embedding layer** - alongside token embedding and positional embedding, BERT utilizes a *segmentation layer* which is used to differentiate between segments in sentence. This enables the model to learn relationship between different parts of the sentence. The input embeddings are the sum of the token embeddings, the segmentation embeddings and the position embeddings [70].

The encoder stack itself follows the design in Figure 5.7.
BERT was a core *text module*, specifically BERT-LARGE which has a context size 1080 (standard BERT has size of 768).

5.3.4 Large language models and Llama 3.1

Llama [72] is a Large Language Model. LLMs are transformer based models with many (billions) parameters trained on massive corpora of texts. These models have shown the ability to perform new tasks from textual instructions or from a few examples. These so called *few-shot properties* appear when scaling a model to sufficient size [73].
This observation claims that scaling is everything. However in [74] this was adjusted by a claim that smaller models (but still large enoughh) with **more** data samples perform better than bigger models with fewer data samples[10].
The main complexity in LLMs in general lies in their size and the trainig process, neither is relevant to this thesis therefore it will not be explained; refer to [72] and [75] for details.
Unlike BERT, Llama is a **decoder only** transformer, meaning that the primary objective of these model is to generate[11] tokens. Decoder-only models were not designed for NER or other classification tasks, but it is possible to use them for such purposes.

[10] [72] the authors also created several smaller models which perform very well.
[11] Recall the purpose of discriminative and generative models 2.2.4.1

Dataset 6

This chapters explores the datasets used for training and inference. It also provides relevant statistics for design decision, be it in regards to neural network architecture or data processing.
One of the main points of this thesis was to explore relevant datasets with **historical named entities**. To the best of our knowledge no dataset matching this criteria is available. We have looked for other datasets for MNER and found Twitter2017 [76] and Twitter2015[77][1]. We have also created a custom dataset, HiCBaM, from historical books.

6.1 Twitter 2017

Twitter 2017 (T17) is constructed from tweets on social media platform Twitter (now X). Tweets are valuable for MNER because they can consist of text and images. This dataset contains 9 distinct labels: *B-PER*, *I-PER*, *B-MIS*, *I-MIS*, *B-ORG*, *I-ORG*, *B-LOC*, *I-LOC*, *O*[2]. The dataset consists of 8576 tweets with 17496 unique tokens and 24021 images. The dataset itself is split into three subsets: *train*, *validation*[3], *test*. Train subset contains 80% of the tweets and remaining 20% is split evenly amongst validation and test. Label distribution in the entire dataset can be seen in Figure 6.1.

[1] Both datasets are publicly available on github.
[2] label "OTHER" - not an entity
[3] This subset serves many purposes in machine learning, for example to stop training to prevent overfitting and to tune hyperparameters.

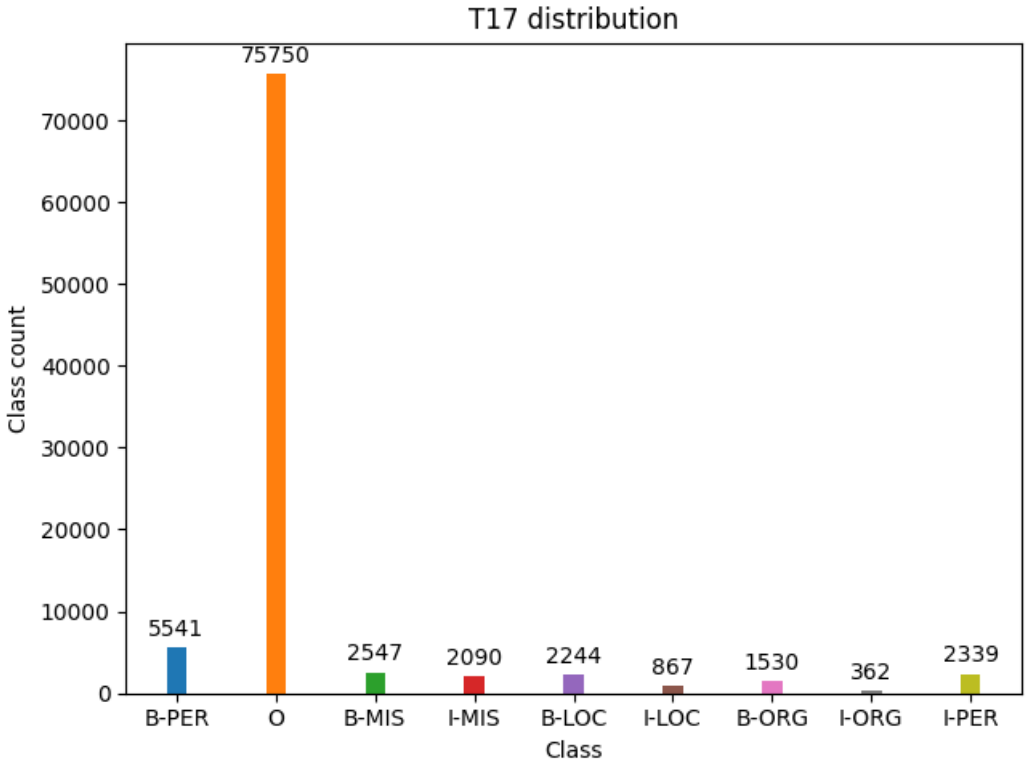

Figure 6.1: Twitter17 class distribution in the entire dataset.

As can be seen on the graph, the label distribution is imbalanced. Label *O* is heavily represented while label *I-ORG* is rarely present.

The distribution in train, validation and test subsets can be seen in Table 6.1.

Table 6.1: T17 statistics for train, validation, and test subsets.

	Train	**Validation**	**Test**
Tweets	6856	860	860
Tokens	74697	9596	8977
Unique Tokens	14899	3580	3576
Images	19177	2388	2456
Labels	74697	9596	8977
Entities	17520	1800	1441

6.1.1 Dataset example

A data sample contains three parts. Text, annotated entities and image reference. An example can look like this: *HB local. We love freedom here in HB, B-LOC O O O O O O O O B-LOC* and the image reference: *7196_0.jpg*. The image can be seen in Figure 6.2.

Figure 6.2: Image associated with given tweet

In this example, the image itself carries no information related to Huntington Beach. It looks like generic suburb in America.

6.2 Twitter 2015

Twitter 2015 (T15) preceded T17 but is constructed just the same with a few difference.

1. Data corruption - there are small hunders of tweets which refer to image that is not present in the dataset. These tweets were not used in model training as they are not multimodal.
2. T15 is larger and contains 10000 tweets.
3. Only 1 image is associated with each tweet.
4. *MIS* label from T17 is called *OTHER*. This means that there are two labels, *OTHER* and *O* which mean different things. For sanity, all *OTHER* labels were mapped to *MIS* in the preprocessing phase.

There is one more small technical detail, the T17 is stored in json [78] format while T15 in stored in conll [79] format. In the preprocessing phase, cconll format is migrated to json.
T15 consists of 10000 tweets with 37106 unique tokens and 10000 images. The dataset is also split into three parts: train, validation and test. The split ratio is different from T17. Train subset contains 70% and remaining 30% is evenly split between validation and test subsets. The label distribution in dataset can be seen in Figure 6.3.

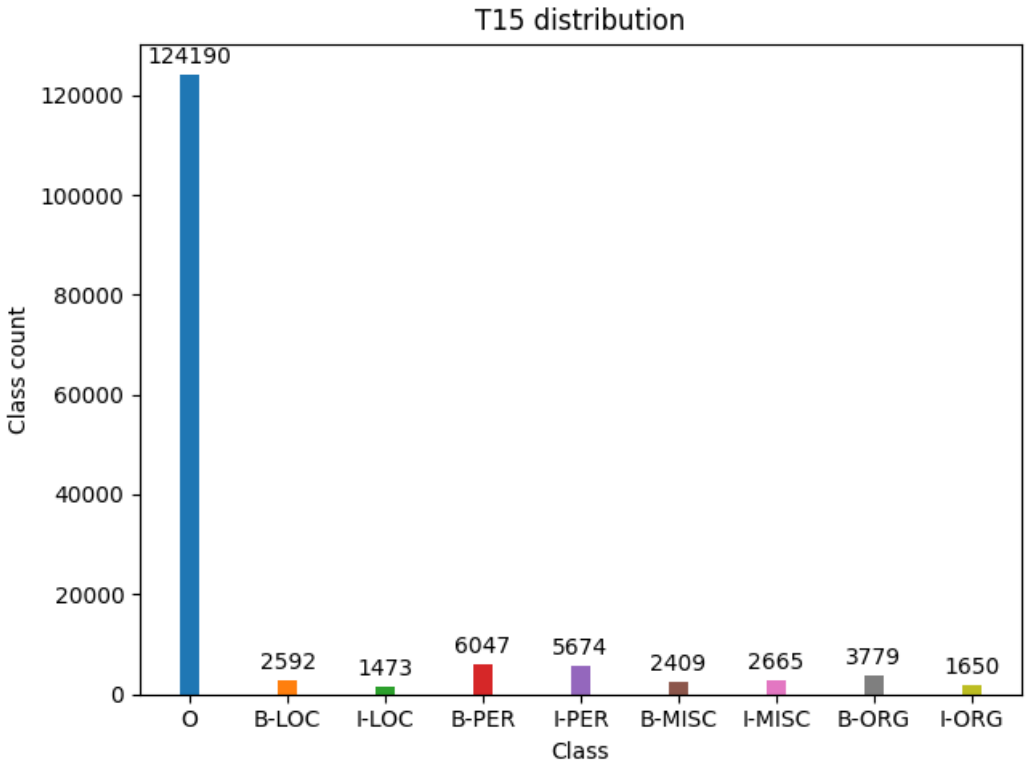

Figure 6.3: Twitter15 class distribution in the entire dataset.

Same as T17, this dataset suffers from huge class imbalance which needs to be addressed while designing solution.

The distribution in train, validation and test subsets can be seen in Table 6.2.

Figure 6.4: Image associated with given tweet.

Unsurprisingly the tweets are quite political.

Table 6.2: T15 statistics for train, validation, and test subsets.

	Train	Validation	Test
Tweets	7 000	1 500	1 500
Tokens	105 542	22 699	22 238
Unique Tokens	22 262	7 430	7 414
Images	7 000	1 500	1 500
Labels	105 542	22 699	22 238
Entities	18 473	3 851	3 390

6.2.1 Dataset example

The format is the same as in T17: sentence, entities and image reference. Sample from this dataset:
RT @ Mahaveerm _ : Pak is Home 2, O O O O O B-LOC O O O and the *image reference: 797970.jpeg*. The image can be seen in Figure 6.4.

6.3 Twitter observations

The datasets are inbalanced with "O" being the dominant class. This is to be expected. Social media posts are very noisy. In Section 4.3.3.2 weight computation for classes is mentioned to prevent overfitting which any model will be inclined to do.
Noise in text data can be addressed by *stemming* [80] and rule based text filtering. Implementation of data transformation and potential downfalls of this approach are mentioned in chapter 7.
All tweets are in English language so any language model does not need to be multilingual.
The dataset split (80% – 10% – 10% train-validation-test) is peculiar. Test set usually is not as big as validation set, a common split is (70% – 10% – 20%). This split performs a more robust evaluation of model. in Figure **??** there are only 44 samples[4] of class *I-ORG* which will heavily impact computed metrics.

6.4 Historical Czech-Bavarian multimodal NER dataset

This new dataset, in short denoted as HiCBaM (sometimes referred to as SOA), was created from books with Czech-Bavarian theme. The concrete books used to construct this dataset were:

- Heitmatbuch MARIENBAD Stadt und Land - a book about Mariánské Lázně.
- Ortschronik Maschowitz - a book about life in towns Ortschronik and Maschowitz.
- Heimat in Böhmem - book about Bohemia.
- Heimatbuch, subtitled Gemeinde Plöß mit Wenzeldorf, Rappauf und Straßhütte - local history of Wenzeldorf, Rappauf and Straßhütte.
- Heimatskunde politischen Bezirkes Falkenau - a book about political district Falkenau.

The dataset was constructed from books provided by *Porta fontium*[5].
The construction of the dataset was done manually. Over 1000 pages were searched for suitable data (image with a caption) for MNER dataset.
HiCBaM dataset contains 9 distinct labels: *B-PER, I-PER, B-MIS, I-MIS, B-BUI*[6], *I-BUI, B-LOC, I-LOC, O*.

[4] This could be the reason why the authors in 3.1 pretrained model on custom dataset.
[5] `https://www.portafontium.eu/`
[6] BUI stands for BUILDING.

HiCBaM consists of 254 sentences with 876 unique tokens and 254 images. The dataset is split into three parts: train, validation and test. Train part contains 70% of the data, validation contains 10% and 20% creates test part. The label distribution in dataset can be seen in Figure 6.5.

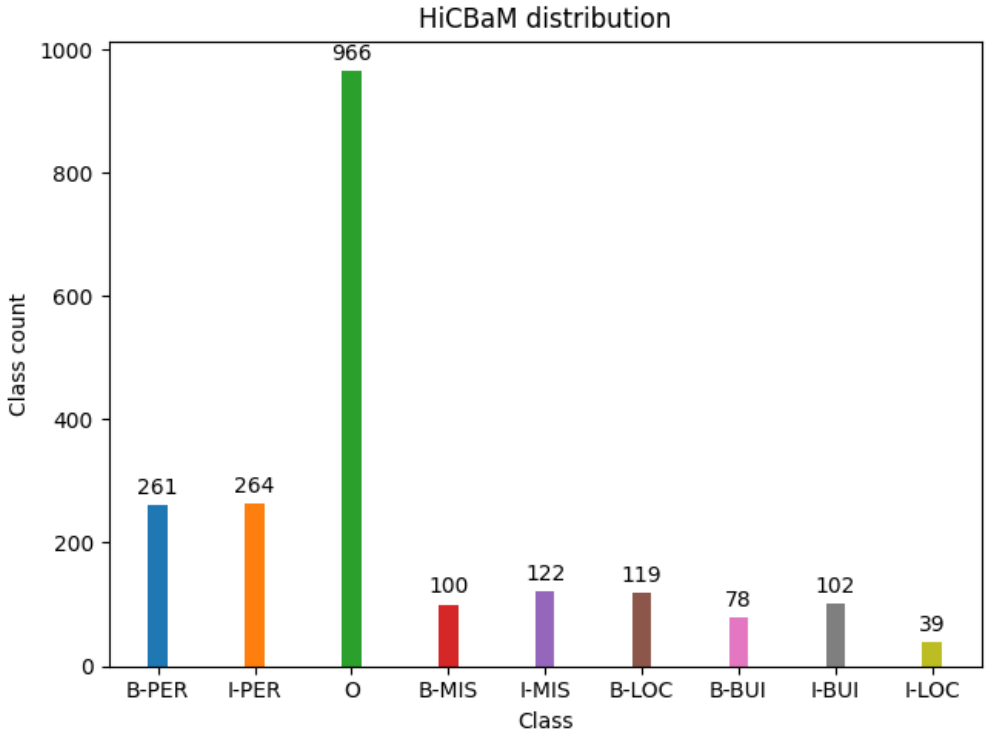

Figure 6.5: HiCBaM class distribution in the entire dataset.

The distribution in train, validation and test subsets can be seen in Table 6.4.

Table 6.3: HiCBaM statistics for train, validation, and test subsets.

	Train	Validation	Test
Sentences	228	26	51
Tokens	1 453	142	287
Unique Tokens	804	142	287
Images	228	26	51
Labels	1 453	187	411
Entities	775	94	216

6.4.1 Dataset construction

After searching through the books, a *tool* was created to convert snippets into data line in the dataset. The dataset is stored in *json* format as *T17* and *T15*.

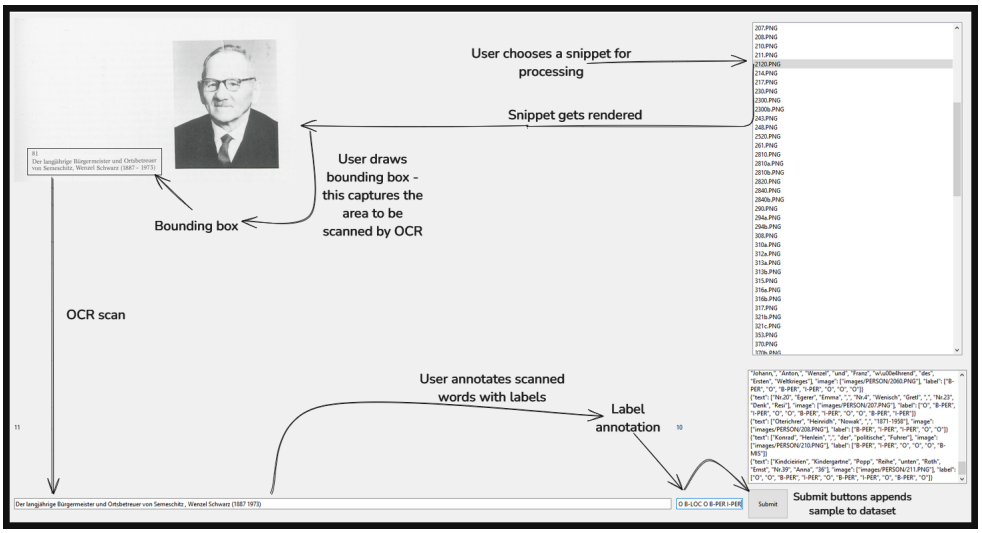

Figure 6.6: Main screen of data annotation tool.

The arrows from top to left visualize workflow of the application. The UI is very raw.

The snippets have low resolution and are in hand-written German so the Optical character recognition (OCR) [81] has to be very robust.
Two main OCR's we have experimented with were *Tesseract* [82] and *EasyOCR* [83]. Tesseract performed poorly for provided snippets and EasyOCR had amazing performance therefore it was chosen as a backbone of the application[7]. To connect a user with OCR and a formatting tool to produce json lines a simple UI tool was created to simplify this process. Data annotation tool is written using QT framework with python flavour[8] to keep the solution lightweight. Architecture of Data annotation tool is described in Chapters 7 and 8.

[7] Further referred to in the text as "Data annotation tool".
[8] pyqt6 https://www.pythonguis.com/

Problem analysis and design

7

This chapter describes several aspects of multimodal named entity recognition. From design decisions regarding data preprocessing pipeline, neural network design and training loop setup. Furthermore the chapter describes encountered issues of technical character and proposed solutions.
For implementation specifics, refer to Chapter 8.

7.1 MNER requirements

As mentioned in Sec. 2.4, MNER involves multiple modalities. This creates a requirement that a system performing MNER must process text and image data.
There are two possible ways how to handle this situation. A monolith like design (see Figure 7.1) where one module processes image and text data (this is the case of GPT-4o [84]) or a modular solution (shown in Figure 7.2) with specialized modules, which is used by authors of [20, 21, 22] and many more.
The benefit of the first approach is a tight fusion between text and image data. This property can theoretically help with data disambiguation since the data from both modalities are processed in the same module with no need to combine them later. This comes with a high cost. Firstly, a specialized pre-trained modules can not be used (like Llama, BERT, ViT). Secondly, any change in modality would make the entire system obsolete due to the tight coupling of modalities.
Second approach with higher granularity does not suffer from above mentioned problems. A system following this design has a crucial component, *fusion layer* (on Figure 7.2 the fusion is a simple concatenation. Refer to Sec. 7.4 for complex fusion strategies). This layer combines extracted features from text and image data. Using this pattern enables the use of pre-trained large models for feature extraction which is a massive advantage.
Both designs were used in experiments. GTP-4o has multimodal abilities. The model is monolithic[1] and can be prompted to perform MNER. All other neural networks

[1] GPT-4o is a monolith unlike GPT-4-Vision which is modular.

were implemented as a modular system. The design difference can be seen in Figures 7.1 and 7.2.

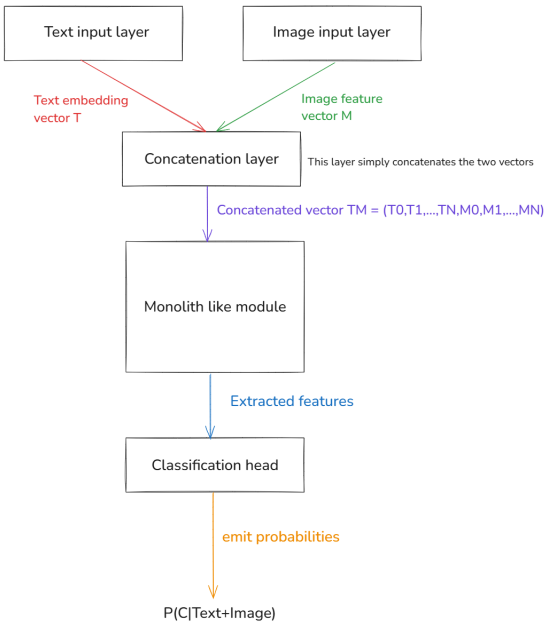

Figure 7.1: Monolith MNER system architecture.

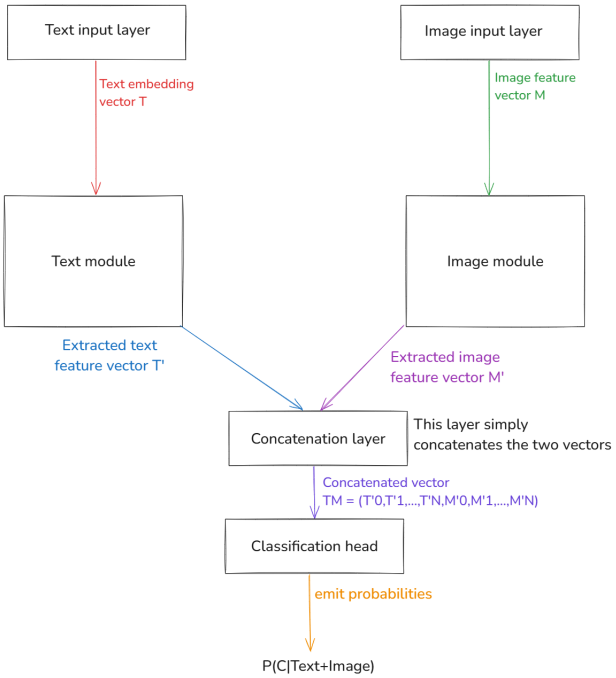

Figure 7.2: Modular MNER system architecture.

7.2 Modality modules

This section describes the design for each individual module. It also mentions important aspect of selected models which will have an impact on the overall results.

7.2.1 Text module

BILSTM, BERT[2] and Llama 3.1[3] are chosen as text modules in the proposed modular architecture.
BILSTM has shown a great success in NLP field and language understanding so it sets a reasonable baseline for a text module. BILSTM architecture has been somewhat succeeded by transformers and BERT has become a go-to standard for text feature extraction. BERT is also bidirectional which enables to perform rich feature extraction.
LLMs like Llama and GPT are *decoders* and not *encoders*, but without *language modeling head* they can be used for feature extraction. The models are massively larger - BERT Large has 340×10^6 parameters while Llama 3.1 has 8×10^9 parameters. However, Llama is not bidirectional and was fine tuned for text *generation*, not *text understanding* like BERT. Because of that, BERT can still outperform Llama even when the size difference is large.
T17 and T15 datasets are in English. HiCBaM dataset is in German. The sentences are not morphologically complex since they originate from social media and image captions.

7.2.2 Vision module

Image data from T15 and T17 are not always relevant to the text as people can add any image to the tweets. HiCBaM dataset has images directly related to the text data. The role of this module in the entire system is to provide additional features to help with ambiguity problem but *text module* is still the dominant component.
Vision transformer and convolutional neural networks are the perfect adepts for this role as they both process images but with a different approach.
Output of ViT are patch-wise[4] features. However, this output is not wanted as it would enforce the text module to align with image features. Image features should align with the text features. A *pooled* output (for example a mean across all patches) is required to provide features for the image in its entirety. CNN does not have this problem as it provides features for the entire image.
AlexNet CNN implementation provides a rich feature extraction on image data and

[2] BERT Large specifically with 1024 hidden size.
[3] Llama 3.1 8B is chosen due to hardware constraints.
[4] For visualisation check Figure 5.8.

just like ViT, it has the same requirement for input image, the $(width \times height)$ must be of the size (224×224).

7.3 Dataset preprocessing

Both image modules have requirements for image dimensions - this is a requirements that needs to be addressed in preprocessing. Text data from tweets are very noisy and contain meaningless symbols which can stripped to improve performance of text module.
BERT and Llama are pretrained and use tokenizers to handle text data without any preprocessing. **Any stemming** on the text data will be harmful. In contrast with BILSTM where stemming will be helpful due to the fact that it reduces the vocabulary size of the embedding layer.
This observation enforces one last constraint on the preprocessing, any processing must be optional.

7.3.1 Preprocessing Twitter dataset

The Twitter15 and Twitter17 are similar so the preprocessing requirements are the same.

Text filtering using rule-based methods, such as *regular expressions* is beneficial since it removes some noise from the text. An example from T15 dataset here **??** there are "words" that carry no informational value: @, _, and 2, but are associated with label (*O*). Removing them lowers the occurrence of dominant class *0* without harming BERT's or Llama's performance. Both *image modules* in architecture (on Figure 7.2) require resizing the images to (224×224) dimension. For GPT-4o no preprocessing is needed.

7.3.2 Mapping Twitter15 to Twitter17

As mentioned in Sec. 6.2, T15 dataset is in *conll* format and not json. The class naming is also a bit confusing.
During preprocessing it is suitable to map conll to json schema so a one singular format is used throughout the system. Another quality of life change is to map class *B-OTHER* to *B-MIS* since they represent the same objects and model can benefit from *transfer learning*[5] [85].

[5] In this context that means to train the model on either T15 or T17 and use this model as a base for finetuning on the other dataset.

7.4 Fusion layer

In modular system, *fusion layer* is a component that fuses (merges the distinct features together to create one representation) text with image.
The two main approaches, linear concatenation and cross-attention fusion, used in proposed architectures are mentioned in upcoming subsections. **Linear fusion layer** is simple. Each module extracts features which are then concatenated (*Concatenation layer* can be see in Figure 7.2). This vector is then passed to *dense layer* with activation function σ. Design of this fusion layer can be seen in Figure 7.3.

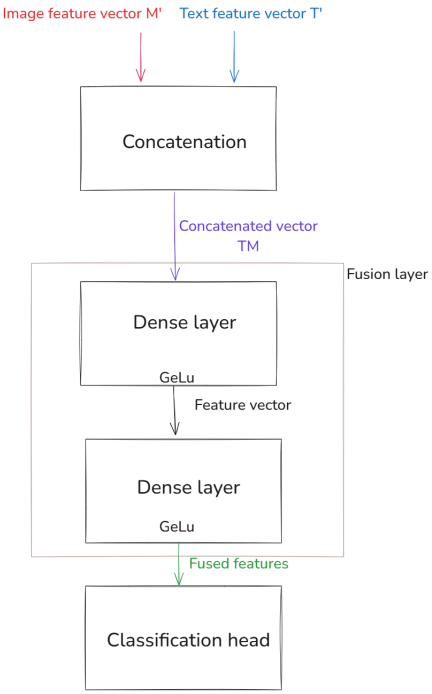

Figure 7.3: Linear fusion layer.

Attention based fusion layer is more complex. It extracts how does different modalities attend to each other. For example to extract "Which part of image are relevant to my text" and vice versa.
The attended features are then concatenated and passed to classification head. Design of this layer can be seen Figure 7.4.

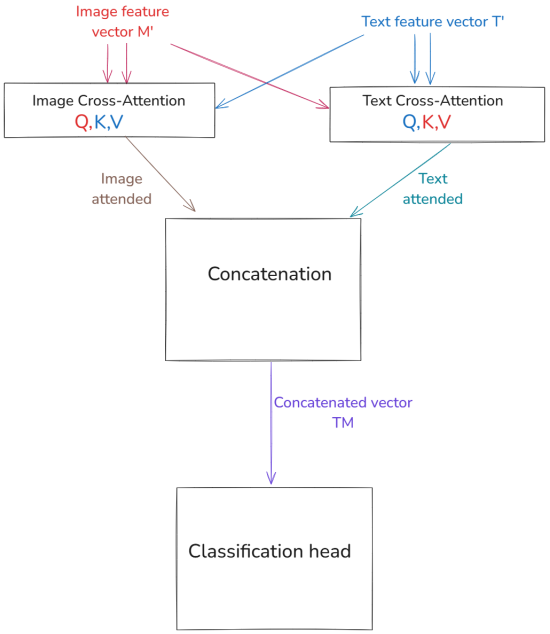

Figure 7.4: Cross attention based fusion.

7.5 Classification head

Classification head is a simple dense layer just the same as *output layer* mentioned in Chapter 5.
In preliminary experiments, MoE was tried and did not work well. Dataset is not large enough to train the module as a classification head like Wang et. al did in [20].

7.6 Activation functions

Activation functions are necessary for non-linear relationships between the layers. For example in Sec. 7.4 a *GeLU* [86] activation function is used. This activation function is preferred over ReLU for large models. ReLU does not activate neurons under a threshold value.
The red "bubble" in the Figure 7.5 shows this difference in function behaviour. While *ReLU* does **not** activate neurons values $x <= 0$, GeLU does if they are close to 0. This is beneficial for larger models. In the Figure 7.6 a derivation of the functions is visualized. GeLU is smoother than ReLU which allows a more stable gradient computation and weight updates via backpropagation.

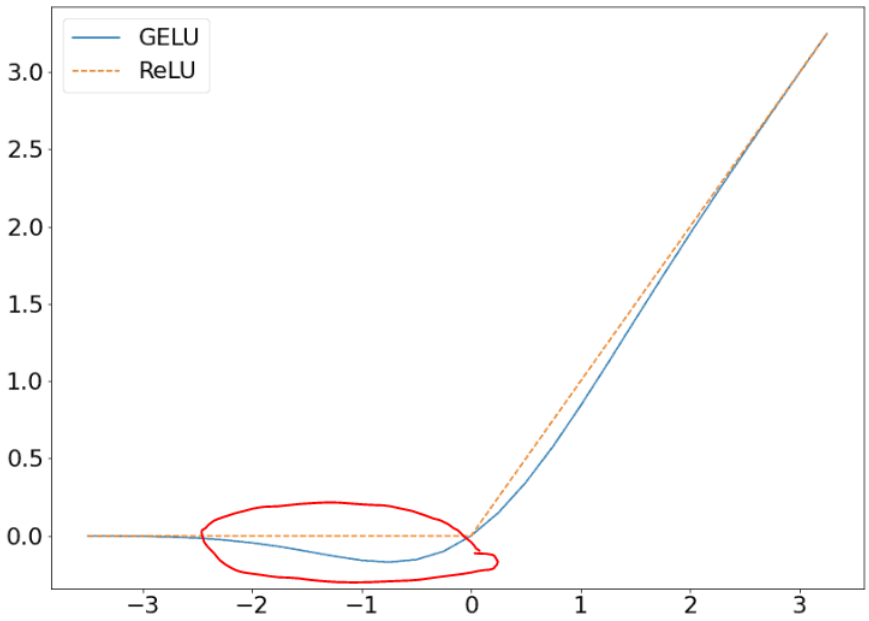

Figure 7.5: ReLU v. GeLU [87].

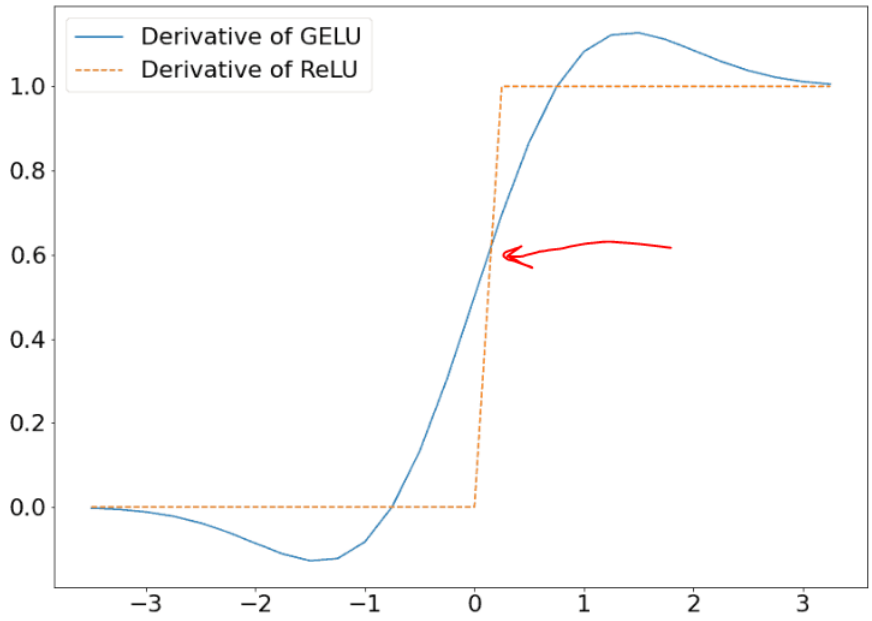

Figure 7.6: Derivation of GeLU and ReLU [87].

The red arrow points to a big difference in smoothness. ReLU is very steep which can harm gradient based optimization, such as backpropagation for large models.

However, ReLU is used as activation function in CNN.
Softmax [29] activation function has found an application in *Partial prediction model* as a *normalization* element (see Sec. 8.2.3).

Implementation 8

This chapter is focused on technical details and issues faced during implementation of proposed solutions in the previous chapter.
It breaks down the concrete implementation of ETL [88] for data preprocessing and loading using coroutines due to volume of the data. Implementation of proposed architectures in previous chapter is described after that.
For comparison, unimodal (text only, image only) model is also implemented to verify that image data improve model's performance. After that, *training* and *inference* loops implementation is briefly mentioned since it contains some minor tweeks worth mentioning alongside a variety of *scheduler* used for finetuning.
The last part focuses on technical difficulties faced with LLMs on premise and ChatGPT *data access object* (DAO) implementation.

8.1 ETL implementation

ETL designed in Chapter 7 is implemented as a asynchronous producer-consumer solution using coroutines. The implementation itself has two variations, *data loader* and *data preprocessor*. Both of these modules can use *processor* on image or text data. Conceptually, *data loader* loads data from storage into memory and optionally performs some processing (application of individual processors). *Data preprocessor* loads the original dataset (T17, T15) and uses transformation[1] to convert it to "new" dataset which is then stored on filesystem.
Figure 8.1 shows a preprocessor design and Figure 8.2 shows the loader.

8.1.1 Data processors

Data processor are a big part of ETL solution.
Processors are injected into Loader via constructor and are *sequentially* applied on text, image or label data.

[1] Both ViT and CNN have input layer with dimesion (224 × 224). Image processor is used to resize images to this dimension.

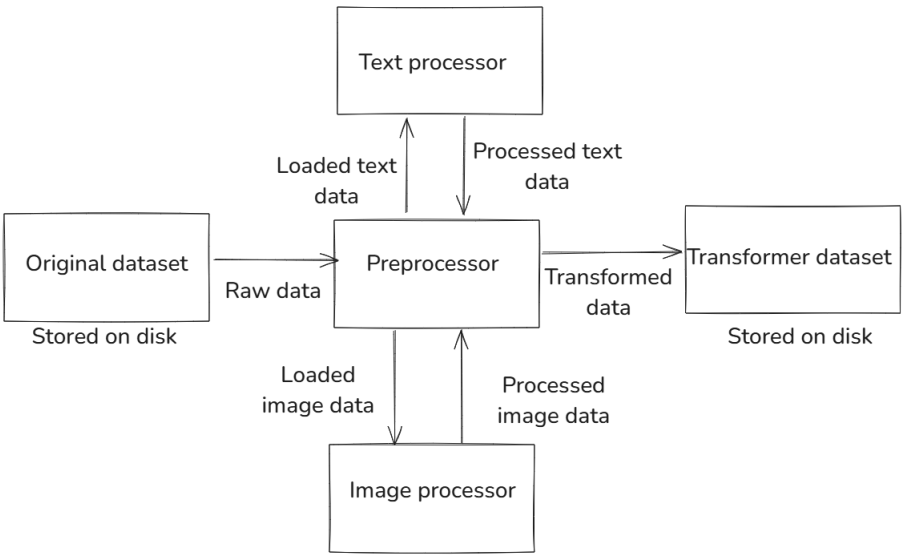

Figure 8.1: Preprocessor.

T15 and T17 preprocessors both use processor to resize images. T15 also uses a label mapping processor to convert "OTHER" entity class to "MIS".
HiCBaM dataset has no preprocessor since the dataset was constructed manually and images were resized in the process.

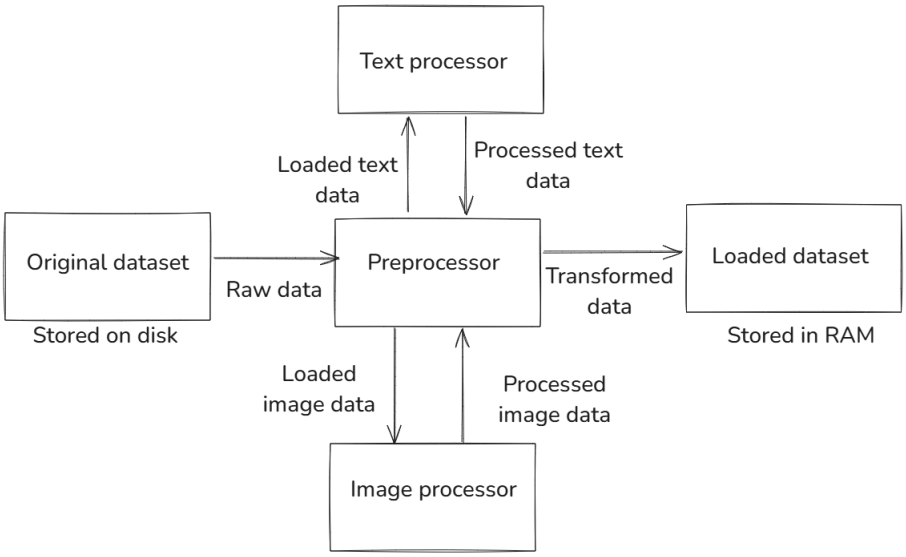

Figure 8.2: Loader.

Loader has no inherit processor. For LSTM training a stemming data processor is used to reduce the vocabulary size.

An UML digram (Attachment 10) shows class and module dependencies in more detail.

8.2 Multimodal models

This section describes implementation of proposed architectures. BERT and ViT are used as a *text module* and *vision module*. The embedding size from *BERT* and *ViT* as shown in Figures have different differention dimension from other models. Concretely, *Llama* has hidden size of 4096, LSTM 600 and CNN also has hidden size of 4096.
The three models described are implemented as a solution for MNER. The models share similarities, such as aggressive down-projections and activation functions, but the core of the models i.e the *fusion mechanism* is different for each model.

8.2.1 Cross-attention model

The architecture of cross attention model is shown in Figure 8.3. The core idea behind this model is to teach *attention* layers which part of text is relevant to which part of image and vice versa. For example if there is a distinct feature in the text, such as the word *Obama* and ex-president Obama is on the picture, the *Text cross attention* layer should learn that.
Text module is implemented as Llama, BERT or BILSTM. Vision module is implemented as CNN or VIT. *Text hidden size* dimension is 4096, 1024 and 1024 for Llama, BERT and BILSTM respectively. *Visual hidden size* dimension is 768[2] for ViT and 4096 for CNN.

[2]The features extracted by ViT are of dimension $(batch, 16, 768)$ since ViT extract features for each patch in image. This output is averaged over the second dimension to extract features for the image as a whole.

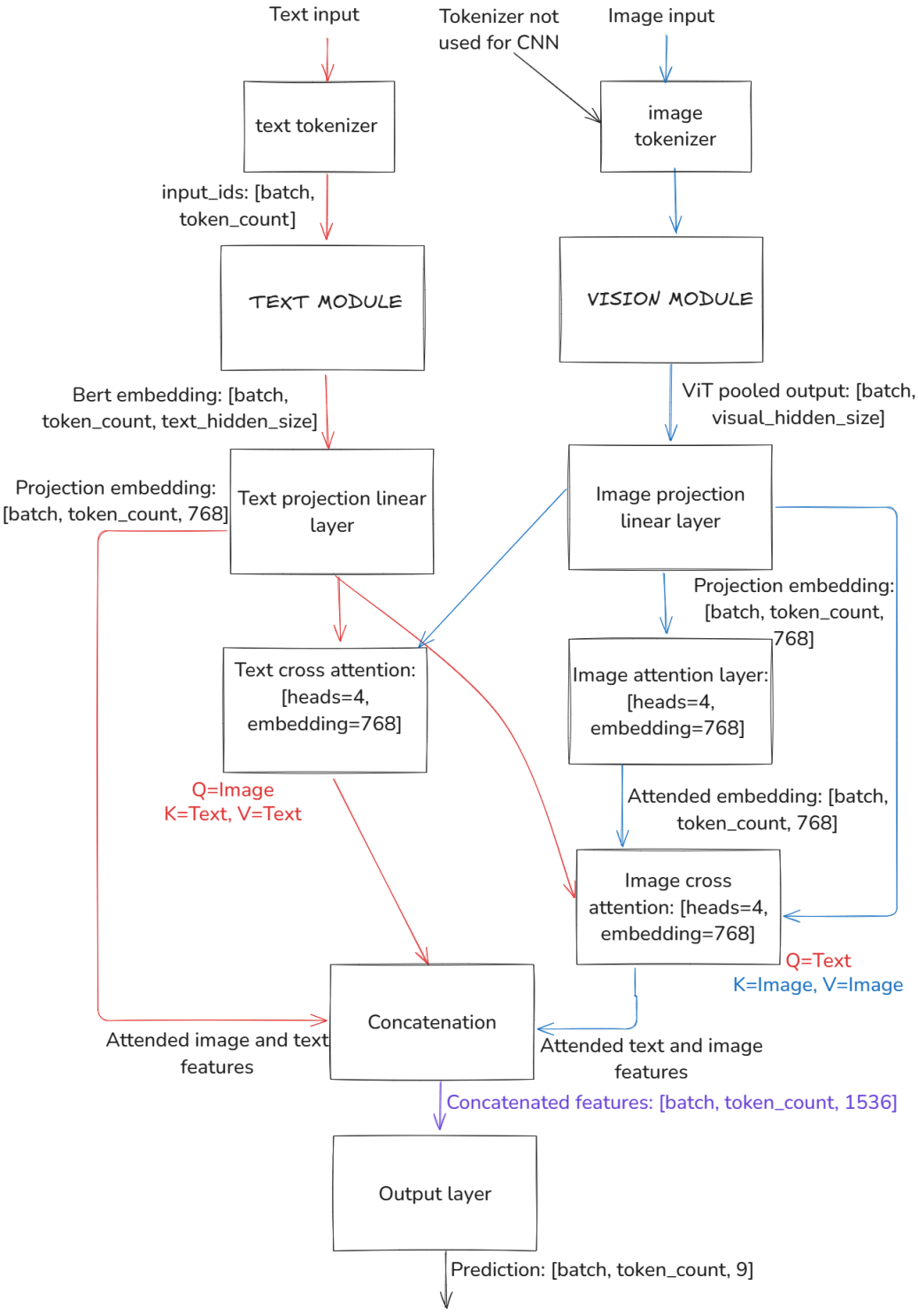

Figure 8.3: Cross attention model architecture.

Residual connections enrich the the attended features from attention layers. Element wise addition is performed before the feature concatenation.

8.2.2 Linear fusion model

The idea behind this architecture is simple. Combine extracted features from *text module* and *vision module* with linear layers. The main idea behind this implementation is:

- Reduce embedding space - *text projection* and *image projection* layer reduce the space from R^{hidden_size} to $R^{hidden\_size/2}$. This will force the model to choose only the relevant extracted features from the original. space[3].
- The features are then concatenated and passed into another linear layer, creating a non-linear transformation.
- The fused features create sequential data which are passed into single-layer BILSTM which provides feature extraction in the fused space.
- CRF is used to compute *loss* value using *negative log likelihood* function.

Using aggressive down projection such as these via linear layers proved very efficient for this task. CRF with performed generally better than cross-entropy loss in conducted experiments for this architecture. The usage of BILSTM is not typical, it helped with performance of this topology in a small way (roughly 2% on *macro f1*). This might be due to the sequential nature of text data in the fused embedding.
The topology of this model can be seen in Figure 8.4.

[3] hidden_size is **different** for text and image models.

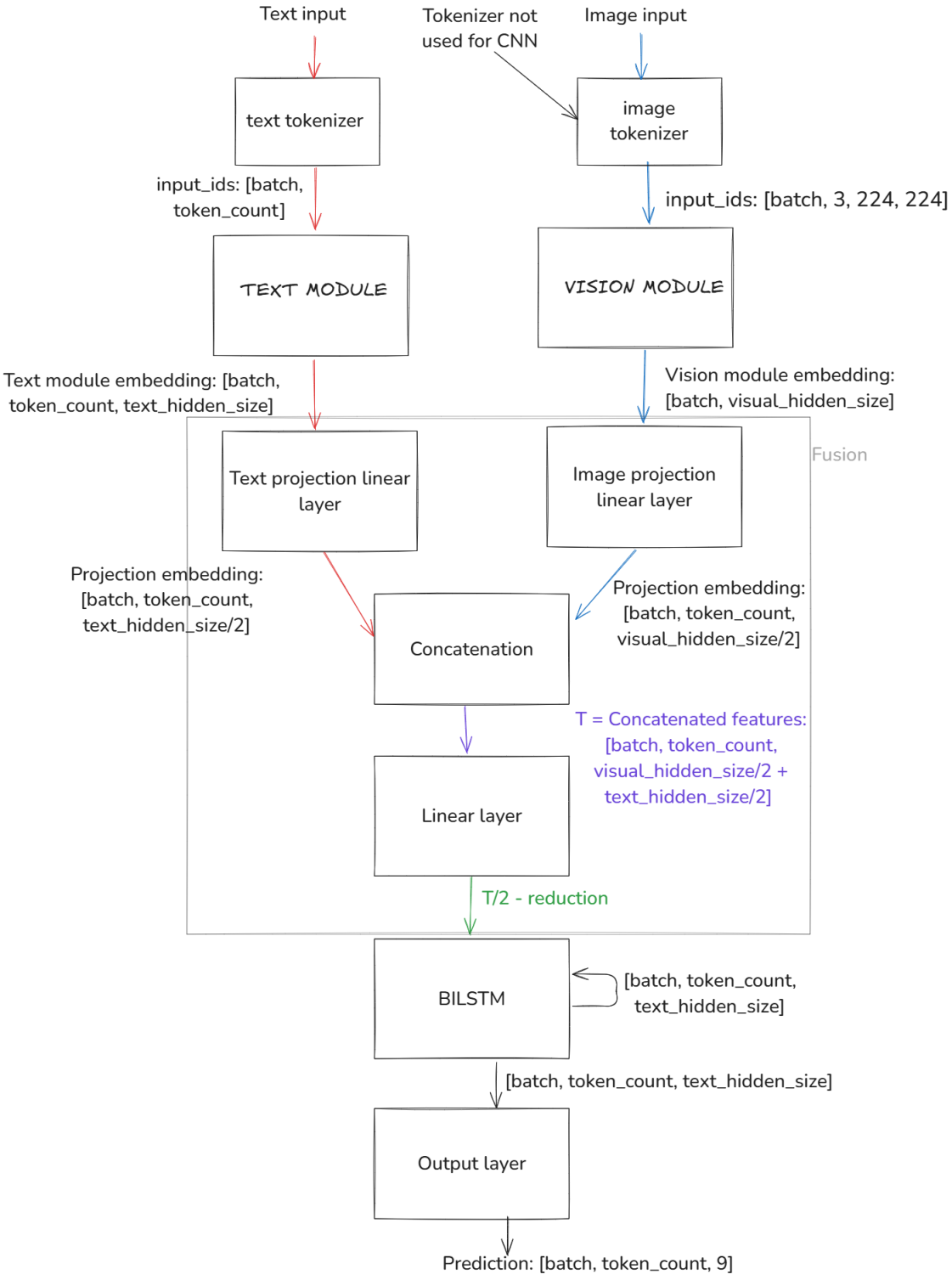

Figure 8.4: Linear fusion architecture.

Linear fusion might be misleading - the fusion itself is non-linear but is implemented using dense layers, denoted as *linear* in pytorch.

8.2.3 Partial prediction model

This architecture can be seen as two classifiers with simple dense layer on top for final prediction. The idea is that *text module* and *vision module* both make a priori predictions. These two predictions are used by a dense layer for posteriori prediction. The architecture of this model can be seen in Figure 8.5

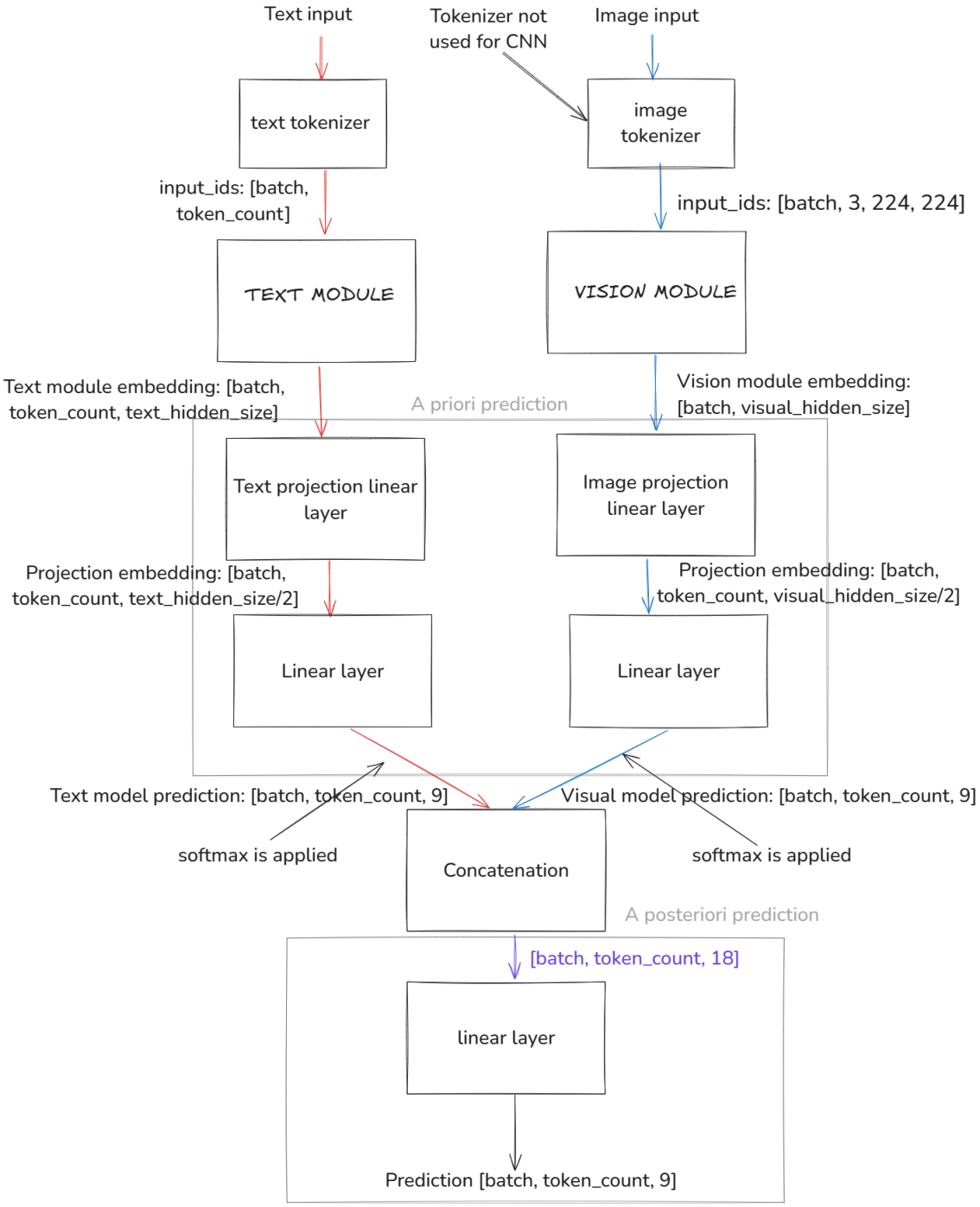

Figure 8.5: Partial prediction architecture

8.3 Unimodal models

To compare the proposed multimodal solution, experiments were conducted with *text-only* and *image-only* models.

8.3.1 Text model

NER is primarily a text-based task. Language models, such as BERT and Llama should perform well even without additional features provided by *vision module*. Transformer based solutions utilize CRF and BILSTM. The reasoning behind topology design is the same as with multimodal topologies. Down-projection with GeLU with the added "benefit" of not having to include a fusion mechanism to combine modalities.

8.3.1.1 Transformer based

The topology shown in Figure 8.6 uses Llama and BERT as a text module. Llama and BERT have their own tokenizer.

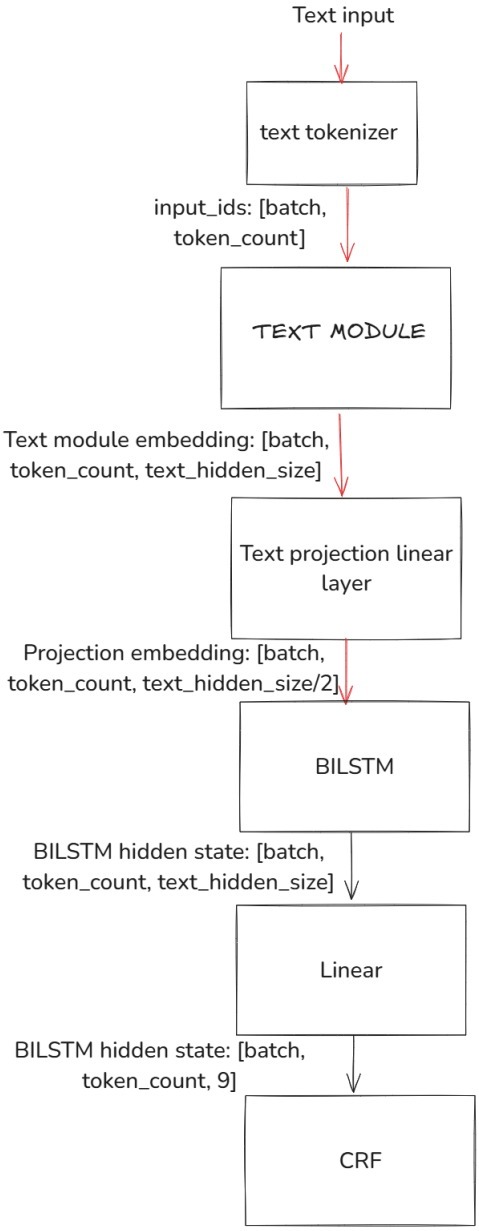

Figure 8.6: Text only model architecture for transformer models.

The tokenizers of Llama and BERT have the same API but different outputs. This means that stripping the extra *cls* tokens before making prediction is different for each model. Everything else is shared.

8.3.1.2 BILSTM based

Only notable difference from transformer model is the tokenizer. LSTM's tokenizer maps individual words in sentence to embedding space. *Token_count* is therefore always equal to the lenght of the sentence. The architecture is shown in Figure 8.7. The notable difference in BILSTM solution is the tokenizer and the embedding layer. As mentioned previously, text filtering is applied before using this text module. That is because the embedding layer is a trainable associative map that perform projection of a word to embedding space. Reducing the size of vocabulary is therefore benefitial for training.

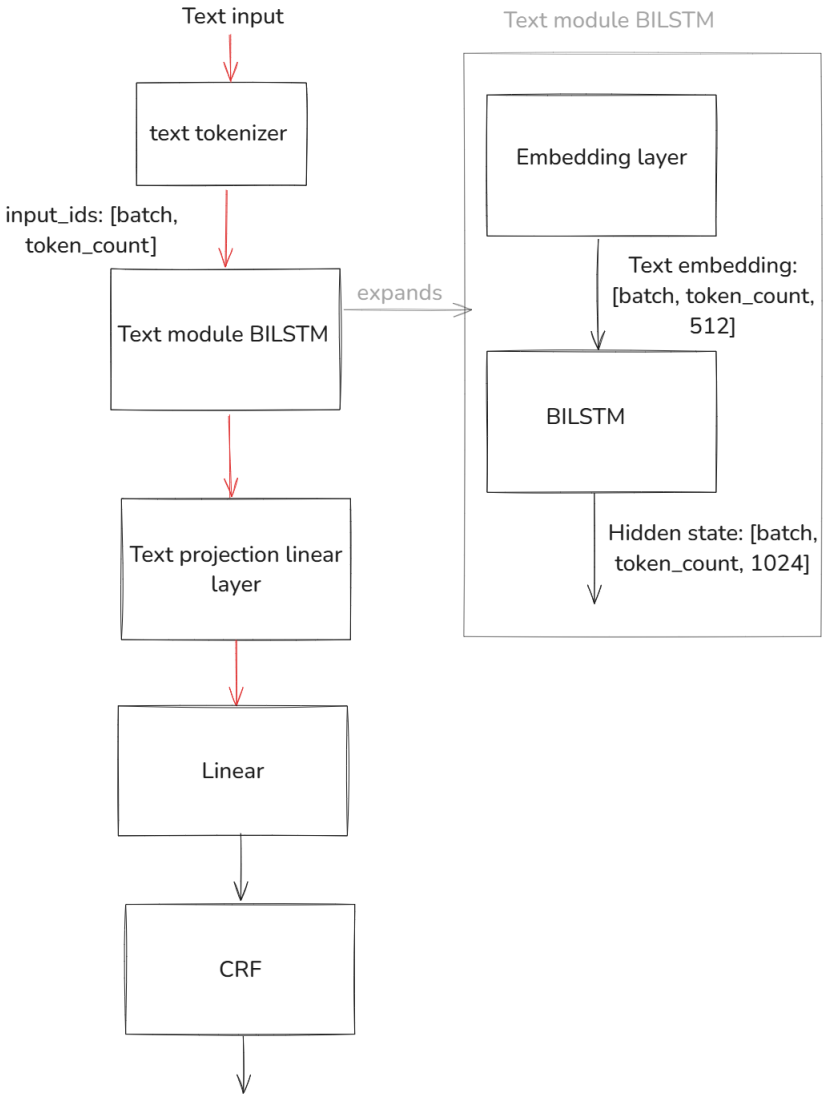

Figure 8.7: Text only model architecture for BILSTM model.

The *Embedding layer* used in BILSTM module maps distinct words to embedding space.

8.3.2 Image model

Implemented *vision modules* extract features of the image as a whole. Predicting various labels from one or two image features is not expected to work well. Visual module is used to extract features from the image which are then passed to *up-projection down-projection* layer. This layer is primarily used as a trainable adapter. This neural network architecture is shown in Figure 8.8

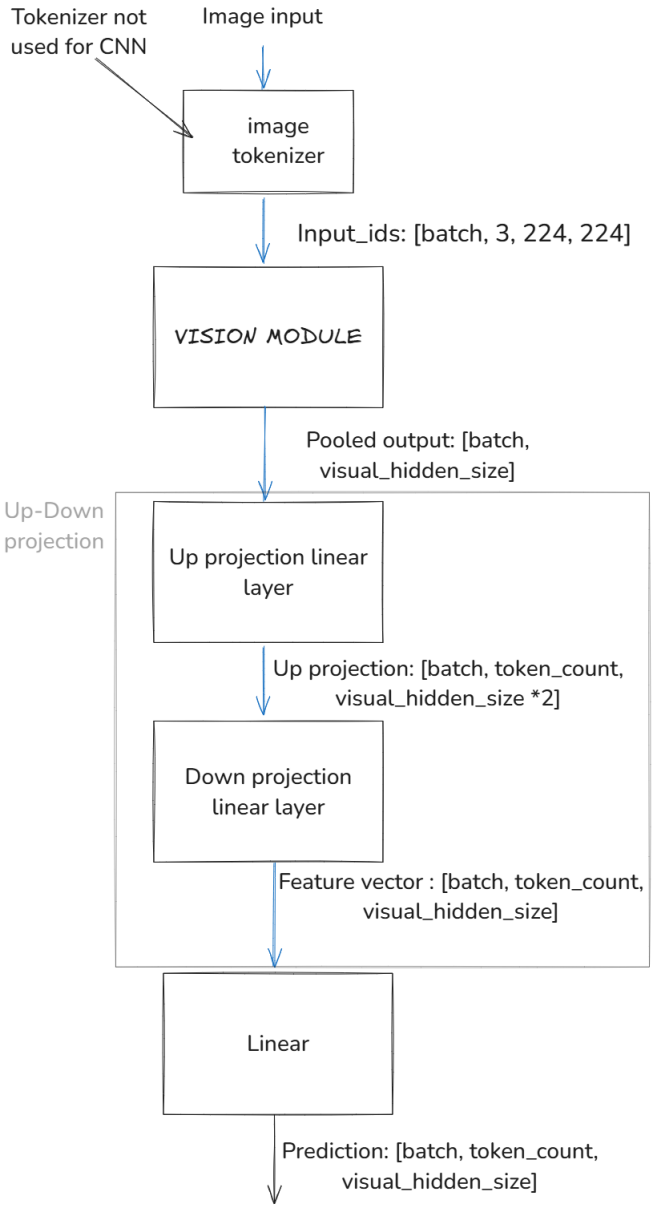

Figure 8.8: Image classifier with up-projection down-projection layer.

Image classifier uses cross entropy as a loss function.

8.4 Training the models

Training consisted of 3 main components:

- Model - the concrete model for training.
- Optimizer - a function that updates parameters of the model.
- Scheduler - a function that *changes* the learning rate α during the training.

To breakdown the training algorithm i.e what each component does:

1. Text and image (if model is multimodal) data are passed forward through the model. This computes the model prediction $Y_{predicted}$.
2. *Loss* value is computed using loss function with $Y_{predicted}$ and Y_{true}. *Negative log likelihood* for models **with** CRF and *cross entropy* for models without the CRF layer. This is the "error" of the model as a whole.
3. Gradients are computed using backpropagation algorithm and subsequently, parameters are updated using the optimizer.
4. Scheduler is used to alter the learning rate.

These steps are performed for each data sample in the training subset. This iteration over the entire training subset is performed N[4] times or until *early-stop condition* is met.

8.4.1 Validation

This subset is used to test if model is able to generalize during training. If the performance is getting worse on the validation data (the data model has not seen) then the model is clearly overfitting to the training set.
Another usage of this subset is to finetune hyperparameters of the model - for example the learning rate α and potentially to perform *early stop* if condition is met.

8.4.2 Early stopping

Early stopping is a technique that is used to stop the training if model is not getting better for M[5] epochs. A common practice is to stop model training if the *loss* value increases M times on the validation subset.
Experiments with this approach were tried but they were unsatisfactory. For MNER

[4] N is commonly called *epoch*.
[5] M is commonly called *patience*.

tasks, *loss* did not serve as a reliable indicator of model performance. The models were performing worse on the main metric *-macro f1 score-* very quickly but the overfitting which was the cause of that occurred was not as prevalent in *loss*. For example, the model's macro F1 score dropped from 75% to 70%, despite a lower *loss* value.
Based on this observation an early stopping mechanism on the *validation macro f1 score* rather than *loss* was implemented and used.

8.4.3 Inference

Inference loop is generally the same as validation loop but serves a different purpose. To verify real performance of the model. Parameters of the model are not updated in this stage. The evaluation subset is usually bigger than validation set for a more thorough evaluation.

8.4.4 Schedulers

Experiments with various schedulers were conducted. The two schedulers that performed well were: *warm cosine restarts* and *plateau* scheduler. *Linear* and *cosine* schedulers were tried as well but they performed worse for every situation.

8.4.4.1 Warm cosine restart scheduler

Warm cosine restarts [89] is an extension of cosine scheduler. The idea is to reset the learning rate α to original value after a number of steps (usually denoted as T_i). Before this reset, α is being lowered after each step.
To express this more formally,

$$\alpha_t = \alpha_{min} + \frac{1}{2} \times (\alpha - \alpha_{min}) \times (1 + \cos(\pi \times \frac{T_{cur}}{T_i}))$$

Where α_{min} is the lower bound of learning rate. T_{cur} is the current step and T_i is the total number of steps to be performed before restart. T_i value increases after restart. These restarts are very beneficial since they allow the model to escape local minima. This is exceptional for heavily imbalanced datasets such as T15 and T17.
The period between restarts increases after each restart. This prevents the model from overwriting existing knowledge and thus avoids so called *catastrophic forgetting*.
The learning rate values through the training are visualised in Figure 8.9.

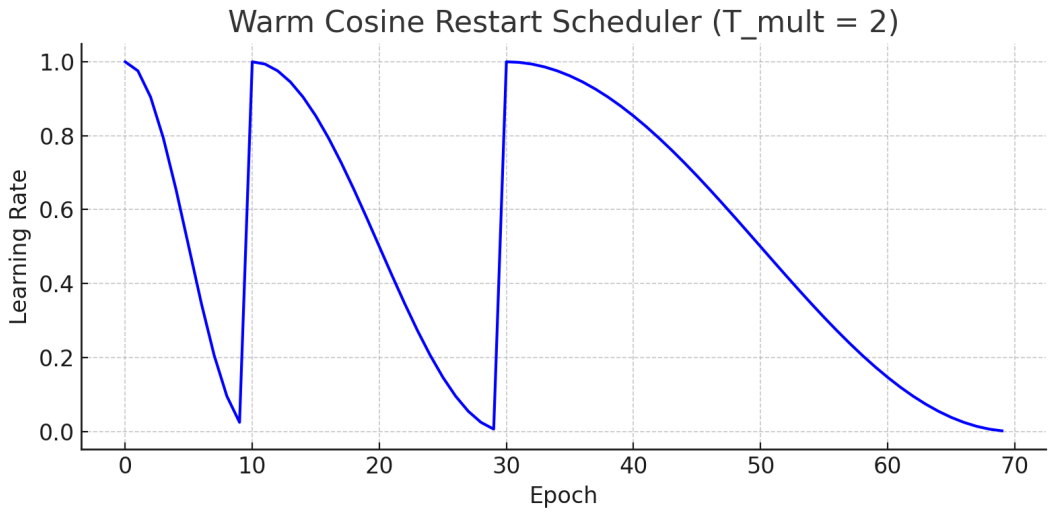

Figure 8.9: Warm cosine restarts.

T_mult is the factor by which T_i is multiplied after each restart.

$T_0 = 5$, $T\_mult = 2$ and $\alpha_{min} = 1 \times 10^{-6}$ with *AdamW* optimizer was used in the experiments.

8.4.4.2 Plateau scheduler

Value α is still getting lowered but by a constant value k and only when plateau is reached. Plateau is reached when a metric does not improve by a *threshold* value after *patience* steps. This simple scheduler was used in conducted experiments since it outperformed all the other schedulers and performed pretty much the same as *Warm cosine restart scheduler*. The simplicity of this scheduler is the reason why it was preferred.

8.4.5 AdamW optimizer.

As mentioned in Sec. 4.3.2.3, AdamW plays a vital role in training phase. The experiments with SGD with Nesterov's momentum were tried but the updates of parameters were too slow.

Due to many complex components in the implemented models, mainly due to issues tied with Llama 3.1 the setup of the optimizer is non-trivial. The best results were achieved using setup in Table 8.1 for BERT and 8.2 for Llama 3.1.

The experiments were conducted with more aggressive learning rates for BILSTM (text module) but they did not improve performance of the model so the same setup as for BERT was used.

Table 8.1: BERT AdamW optimizer setup.

Component	Learning rate α
BERT/BILSTM	8×10^{-6}
Visual module	8×10^{-5}
Fusion layer	5×10^{-4}
Inner BILSTM	3×10^{-4}
CRF	2×10^{-4}
Dense layers	1×10^{-5}

Stable fine-tuning of Llama is difficult as the model is very sensitive to (even small) learning rate changes. The setup below works well and the model does not overfit.

Table 8.2: Llama 3.1 AdamW optimizer setup

Component	Learning rate α
Llama 3.1	2×10^{-5}
Visual module	2×10^{-5}
Fusion layer	2×10^{-5}
Inner BILSTM	2×10^{-5}
CRF	2×10^{-5}
Dense layers	1×10^{-5}

8.5 Storing and versioning of the models

Models are implemented in python using pytorch library. Pytorch can save the entire model into their custom *pth* format. However, this approach is not recommended by the authors of the library and is marked as deprecated. The preferred way is to store the *state dictionary*[6], a data structure which maps a *layer name* to parameters of the layer.

This approach has a very obvious benefit - it saves only the relevant information after finetuning (the weights) and therefore fewer bytes are needed so it saves space. The disadvatange however is the fact that the saved state dictionaries has to be versioned along with the model architecture. If the architecture changes the state dictionary can not be loaded. The layers of the model will not match stored layers in the file.

This means that versioning scales very poorly since one has to use two different

[6]As of writhing this thesis a RCE vulnerability was discovered for this approach.

tools and synchronize them. For example one can use GIT for versioning code and a persistent storage, a filesystem with directory hierarchy that matches particular branches or object storage, such as *S3* with different bucket names.
GIT branches and filesystem hierarchy was used as a solution for the conducted experiments.

8.6 The LLM problems

The problem with running LLMs locally is their size. Llama 3.1 uses *float16* data type which requires 2 bytes. With Llama 3.1 8B, to load the entire model requires $8 \times 10^9 \times 2$ bytes, or 16 GB VRAM on GPU. Running instances of models with this size (and 8 billion parameters is relatively small) is very expensive.
This problem can be mitigated using *quantization* and *parameter efficient fine-tuning* (PEFT) using Low-Rank adaptation approach (LORA) [90] technique.

8.6.1 Quantization

Quantization [91] reduces the bit space of a model's parameter from higher bit-length (float32 - 32 bits) to lower bit-length (float16 - 16 bits or even float8). This lowers the accuracy of the pretrained model due to the loss of data in reduction. On the upside, this technique makes the models viable for finetuning locally on reasonable hardware.

8.6.2 PEFT and LORA

PEFT is a framework for fine-tuning large pre-trained models by updating only a small number of parameters. This makes training cheaper and faster. LORA is the implementation of this concept that was used in conducted experiments.
What LORA does is relatively simple at its core but made possible due to several key observations from the authors, which are described in [90].
Low-rank adaptation is a technique which breaks down one large matrix (in LLM context the weights of layers) and expresses them using two matrices **smaller** (rank decomposition) matrices.
For example, original matrix $A = (1024 \times 1024)$ has to store a lot of numbers - specifically 1048576. Most LLMs store their weights on 4 bytes, so that is 4194304 bytes, 4 GB, for just one layer.
What low-rank adaptation does is that it breaks down the original matrix and expresses it as a multiplication of the smaller one. $A_1 = (32 \times 1), A_2 = (32 \times 1))$ are the smaller matrices and their product has the **same** shape as the original matrix: $A = A_1 \times A_2^T$. Only 64 numbers need to be stored. This of course comes with the cost of potential loss of accuracy.

> *"LoRA allows us to train some dense layers in a neural network indirectly by optimizing rank decomposition matrices of the dense layers' change during adaptation instead, while keeping the pre-trained weights frozen."*

[90].
LORA config used in conducted experiments with Llama 3.1 are listed in Table 8.3.

Table 8.3: Llama 3.1 LORA configuration.

Parameter	Value
R	2
Lora α	16
Lora dropout	0.1

R is the rank of decomposition matrices. Lora α is learning rate used to update values in said matrices. Dropout is used for regularization.

8.6.3 Stabling Llama

Models are saved as a state dictionary as mentioned before. By using PEFT and quantization the model after finetuning can not be saved and loaded as is. The LORA layers first needs to be merged with the original model which can then be stored.
An issue we have encountered with *pytorch* and *peft*, concretely pytorch version 2.5.1 and peft version 0.14.0, libraries is that the function which merges LORA adapters to the original model simply does not work. After saving state dictionary and subsequently loading it, the model does not retain any knowledge previously learned.

8.7 ChatGPT connector

OpenAI offers an inference as a service. Their new[7] models have multimodal capability. Experiments were conducted using GPT-4o on T17 dataset to see how well the models perform.
A connector using OpenAI's sdk in python was implement to use the model. The usage of SDK itself is relatively easy since the APIs are designed well. The only **prerequisite** for using the model is API key which has to be purchased.
The main issue with the model is its nature. Since GPTs are generative models, they tend to generate text that was not present in the original prompt or use different labels than allowed. The first error with extra words is relatively easy to solve by

[7] At the time of writing, GPT-4o and 4o-mini

simply filtering them out. The second issue is more difficult to solve. The only reasonable way to solve this issue is to *re-prompt* the model and tell it to fix the mistake by using only allowed labels. But if it happens again, how many times should one retry before "giving up"? Due to this potential "cycle of doom" any responses what contained different labels were filtered out from the evaluation. These errors very rarely, so from the 860 test samples a maximum of 10 were filtered out.
We have used this *system (nowadays called develop) prompt*:
"Perform Multimodal Named Entity Recognition and return only these classes.: B-PER, I-PER, B-MIS, I-MIS, B-ORG, I-ORG, B-LOC, I-LOC, O. Assign label to every word in the sentence. Use json format with 'entities' key for predicted classes."
The user prompt was the tweet itself with associated image.

8.8 Data annotation tool

The annotation tool is a desktop application built on top of *Qt* framework which provides cross-platform abstraction layer. Python flavour of this framework was selected for implementation to keep all the solutions in one language.

8.8.1 Architecture

The application is *event driven*. All desktop applications are powered using signals (or messages on windows) on the OS level. That is why all reasonable frameworks have very strong support for event driven approach.
The core of the application is a custom class, EventEmitter which serves as a lightweight event bus. The provided API offers two functions:

- *listen_to_channel(channel: string, callback: Callable(AbstractMessage))* - adds a subscriber to channel
- *emit_message_to_channel(channel: string, message: AbstractMessage)* - AbstractMessage is an interface. Currently, two implementations of this interface are provided, *ByteMessage* and *StringMessage*.

The main philosophy is that EventEmitter is injected to any component that needs to communicate with other parts of the application. No direct communication between components should ever happen and components should not be dependent on each other.
With this design pattern, EventEmitter becomes a single point of failure in the application. The application is very small so scaling is not an issue there this does not matter. What **matters a lot** is that the messages are consumed on different thread, a worker thread, rather than on main thread as that will lead to UI freezes with

computationally intensive tasks, such as OCR. A usage of this design is shown in Figure 8.10. The main usecase of the application is to draw a bounding box on an image which is then processed by OCR. Text retrieved by OCR is then rendered to an input (as shown in Figure 6.6).

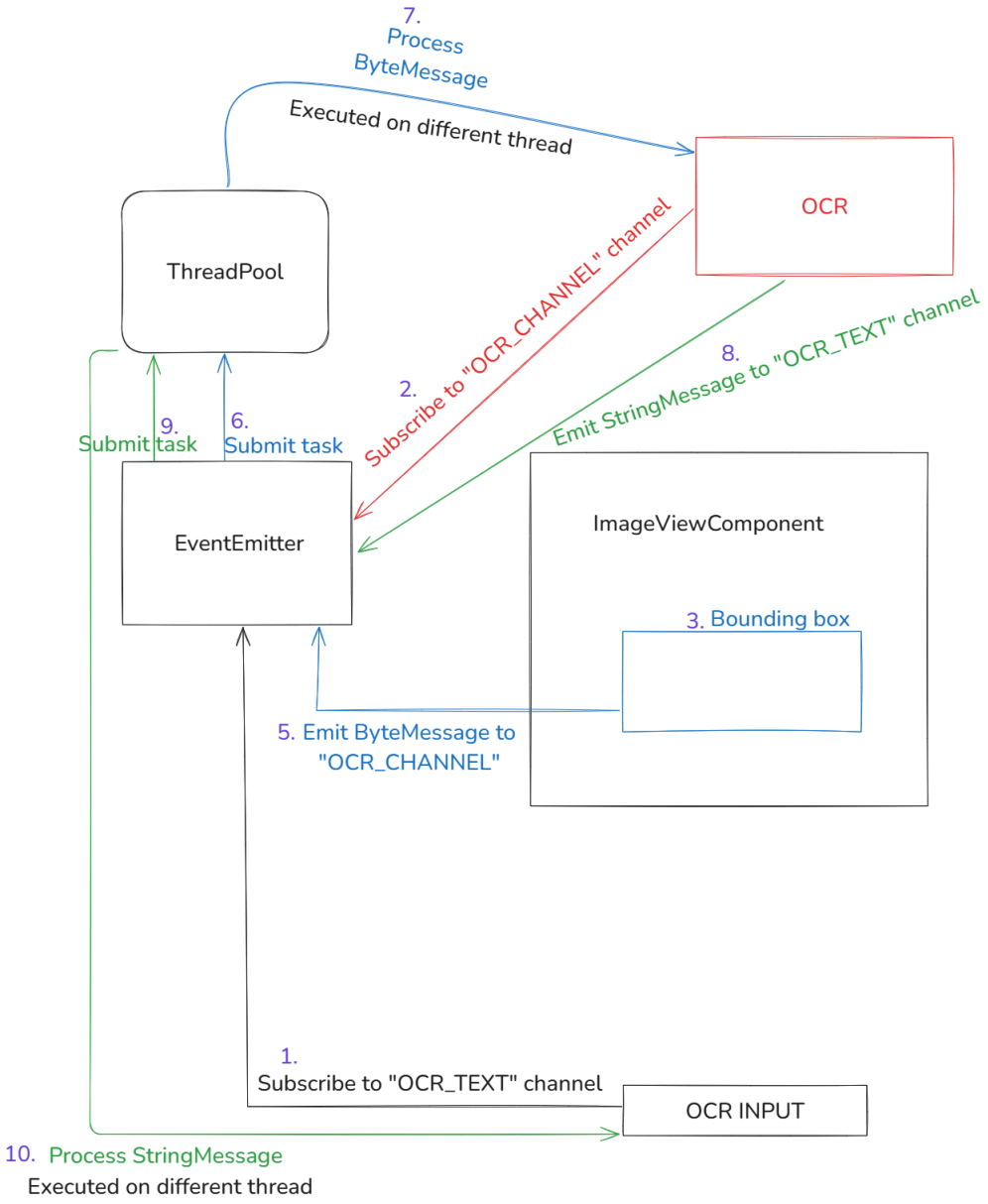

Figure 8.10: Component communication between input element, OCR and ImageViewComponent throught channels.

The purple numbers around the text indicate the order of operations in the workflow. Submit task is the process of submitting a function to a thread pool. Worker thread then invokes the function.

Experiments 9

This chapter contains the information on metric used for evaluation, the training process itself and lastly the achieved results.

9.1 Metrics

This section breaks down the used metrics and reasoning behind using *macro F1 score*.

9.1.1 Problem with accuracy

For prediction systems, accuracy is a famously terrible metric. When one looks at the formula:

$$\text{Accuracy} = \frac{Y_{true}}{Y_{total}}$$

Where Y_{true} are all the correctly predicted labels during classification and Y_{total} are all labels. A system that would for example predict only the majority class O, which represents around 80% of T17 would have an Accuracy of 80%.
This leads to a false assumption that the system works well while it does not since it never predicts any other class.

9.1.2 F1 Score

F1 score [92] fixes this issue. It is calculated as a harmonic mean of *precision* and *recall*.

$$\text{F1} = 2 \times \frac{\text{Precision} \times \text{Recall}}{\text{Precision} + \text{Recall}}$$

If precision or recall is low, the overall F1 score is low.

9.1.3 Precision

Precision is the ratio of true positive predictions to the total number of positive predictions made by the model. This measures how many of the predicted positive instances are actually correct.

$$\text{Precision} = \frac{\text{True Positives}}{\text{True Positives} + \text{False Positives}}$$

9.1.4 Recall

Recall is the ratio of true positive predictions to the total actual positives. It measures how well the model identifies all relevant instances.

$$\text{Recall} = \frac{\text{True Positives}}{\text{True Positives} + \text{False Negatives}}$$

9.1.5 Macro F1 Score

The Macro F1 Score is the unweighted mean of F1 scores calculated for each class. Macro F1 treats all classes equally, regardless of their occurrence. For example in Figure 6.1, class *I-ORG* will have the same value as *B-PER* even when the occurrence of *B-PER* is larger.
This fact makes Macro F1 Score a good metric for imbalanced sequence labeling tasks, such as MNER.

$$\text{Macro F1} = \frac{1}{N} \sum_{i=1}^{N} \text{F1}_i$$

where N is the number of classes and F1_i is the F1 score for class i.

9.1.6 Micro F1 Score

Micro F1 Score calculates the F1 score globally by aggregating the contributions of all classes.

$$\text{Micro Precision} = \frac{\sum \text{TP}}{\sum \text{TP} + \sum \text{FP}} \qquad \text{Micro Recall} = \frac{\sum \text{TP}}{\sum \text{TP} + \sum \text{FN}}$$

$$\text{Micro F1} = 2 \times \frac{\text{Micro Precision} \times \text{Micro Recall}}{\text{Micro Precision} + \text{Micro Recall}}$$

Micro F1 is actually **equal** to **accuracy** for tasks, where a sample belongs exactly to one class. This is the case for MNER. Micro F1 is therefore not a good metric for this task.

9.2 Evaluation approach

Therefore, all the published results are evaluated using *macro F1 score*. There are several valid ways how to compute macro F1 for (M)NER. First and the most strict one is the *token-wise* macro F1. The tokenizer splits the individual words in a sentence to byte sequences. This means that the encoded sentence is longer than the original one. The second step is to *align labels* i.e. assign a label of the original word to each byte sequence originated from the word. For example a sentence:
Hello, Steven Wozniak! O B-PER I-PER
could be tokenized and aligned as:
He ll o , St ev en Woz ni ak! O O O O B-PER B-PER B-PER I-PER I-PER I-PER.
This is how F1 is computed and presented further.
The second approach is more relaxed. The idea is to strip the **BIO** format to the core entity. B-PER, I-PER just becomes PER and so on. The evaluation using this metric was performed and the results were better by roughly 4% but the stripping of positional information from the labels is unwanted. The first evaluation approach is used.

9.3 Multimodal models

This section summarizes the achieved results with the three proposed model architectures for multimodal NER.

9.3.1 HiCBaM results

The results are depicted in Table 9.1, divided by the model architecture. The model architecture with *ViT* as *Vision module* outperforms the plain CNN. BERT being an older and much smaller model than Llama still outperforms it (not to mention the cost to run BERT vs Llama). This is surprising but understandable. BERT has bidirectional capability which enhances feature extraction. The simplest *Linear fusion model (seen on Figure 8.4)* perform very well and outperforms the other architectures. This is also somewhat expected since the model is "simple" and simpler models generally perform better on sparse data. BILSTM does not work very well in general. Using pre-trained embedding vectors might improve the results but it is highly unlikely it would achieve results close to BERT.

Table 9.1: HiCBaM results.

Architecture	Text+Vision module	Macro F1 score
Cross attention model	BERT+ViT	**75.86%**
	BERT+CNN	68.17%
	Llama+ViT	60.71%
	Llama+CNN	58.97%
	BILSTM+ViT	58.51%
	BILSTM+CNN	54.26%
Linear fusion model	BERT+ViT	**77.20%**
	Llama+ViT	76.00%
	Llama+CNN	73.65%
	BERT+CNN	69.57%
	BILSTM+CNN	54.38%
	BILSTM+ViT	36.42%
Partial prediction model	BERT+ViT	**62.95%**
	Llama+ViT	55.11%
	Llama+CNN	53.65%
	BERT+CNN	40.45%
	BILSTM+ViT	31.86%
	BILSTM+CNN	21.21%

Figures 9.1, 9.2 and 9.3 below visualize the training process of the models with the best performance on this[1] dataset. *Cross attention model - BERT+ViT, Linear fusion model - BERT+ViT* and *Partial prediction model - BERT+ViT* respectively.

[1]The training progress is similar for T15 and T17. Additional figures are therefore not included for these datasets.

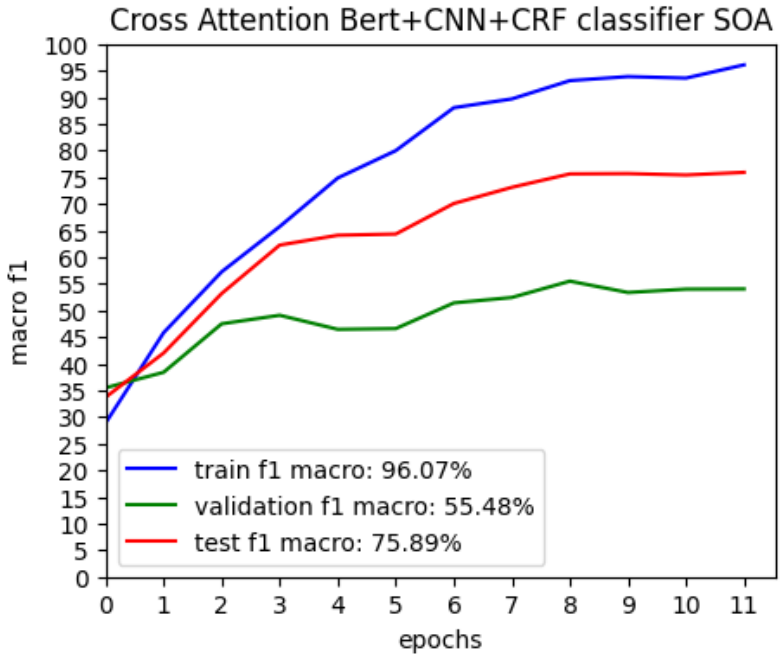

Figure 9.1: Training process of *Cross attention model - BERT+ViT*.

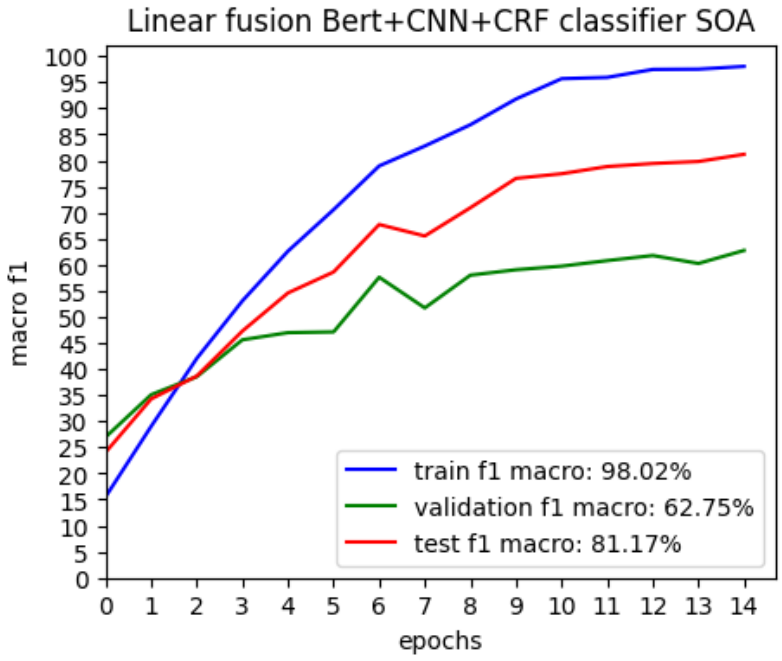

Figure 9.2: Training process of *Linear fusion model - BERT+ViT*.

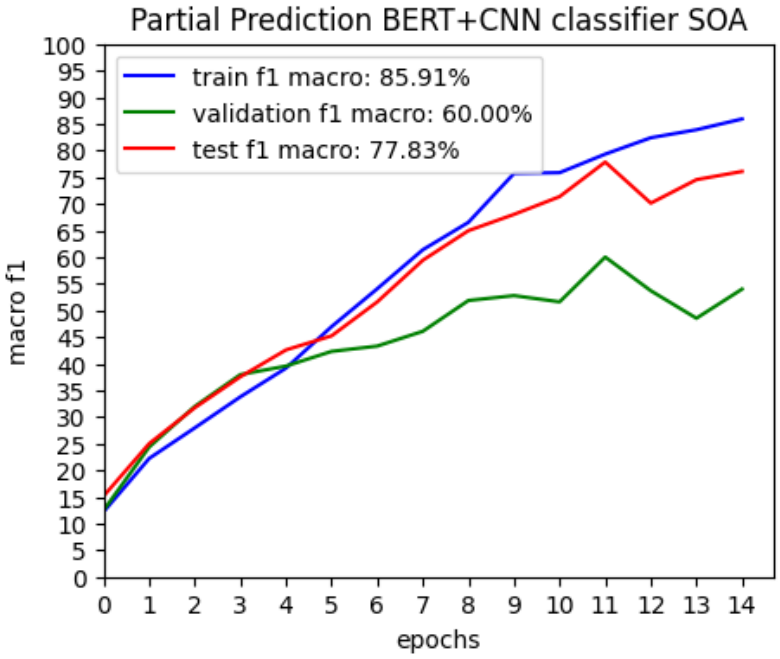

Figure 9.3: Training process of *Partial prediction model - BERT+ViT*.

9.3.2 T15 results

The results are in Table 9.2. Models with BERT as a text module achieve the best results. The results with BERT are comparable across all three network architectures. Where BERT based neural nets are comparable, both Llama and BILSTM differ significantly. Cross attention and Partial prediction architectures struggle with this dataset. Linear fusion performs better but still not as good as BERT.
Attention layers in Cross attention model help BILSTM with feature extraction and performs significantly better than the other variants. It is worth nothing that Partial prediction model does not work with BILSTM. The model *underfits*, i.e does not learn anything.

Table 9.2: T15 results.

Architecture	Text+Vision module	Macro F1 score
Cross attention model	**BERT+ViT**	**75.93%**
	BERT+CNN	73.17%
	Llama+CNN	58.84%
	Llama+ViT	58.44%
	BILSTM+ViT	35.53%
	BILSTM+CNN	34.09%
Linear fusion model	**BERT+ViT**	**75.32%**
	BERT+CNN	74.61%
	Llama+ViT	66.31%
	Llama+CNN	65.98%
	BILSTM+ViT	37.63%
	BILSTM+CNN	33.82%
Partial prediction model	**BERT+ViT**	**74.93%**
	BERT+CNN	74.74%
	Llama+CNN	56.61%
	Llama+ViT	54.25%
	BILSTM+ViT	12.77%
	BILSTM+CNN	10.33%

9.3.3 T17 results.

The results are depicted in Table 9.3.
The task of NER is difficult and T17 dataset is very noisy. State of the art[2] model, GPT-4o-mini performs relatively poorly with only 54.31% macro F1 score. BERT performs the best by a large margin. CNN outperforms ViT in multiple cases. Partial prediction architecture with BILSTM does not work on this dataset. The model *underfits*. Learning rate α was modified, both increased and decreased, but it did not help with the training. Further optimization could solve this issue, but BILSTM does not perform very well on this task in general. This process would yield a diminishing value.

Table 9.3: T17 results

Architecture	Text+Vision module	Macro F1 score
Cross attention model	**BERT+CNN**	**74.17%**
	BERT+ViT	72.63%
	Llama+ViT	62.09%
	Llama+CNN	62.39%
	BILSTM+ViT	44.35%
	BILSTM+CNN	40.42%
Linear fusion model	**BERT+ViT**	**74.35%**
	BERT+CNN	73.10%
	Llama+ViT	68.26%
	Llama+CNN	68.48%
	BILSTM+ViT	40.06%
	BILSTM+CNN	39.96%
Partial prediction model	**BERT+CNN**	**74.47%**
	BERT+ViT	73.20%
	Llama+ViT	52.24%
	Llama+CNN	55.55%
	BILSTM+ViT	9.56%
	BILSTM+CNN	15.21%
GPT-4o-mini		54.31%

[2]At the time of writing this thesis

9.4 Unimodal models

The experiments were performed using the text-only and the image-only models and serve the purpose of comparison with multimodal models.

9.4.1 Image only models

In these experiments, the text is disregarded and the entities are predicted only from the image or images. As expected and as can be seen in Table 9.4, the image only architecture does not work well for MNER.

Table 9.4: Image only results.

Dataset	Model	Macro F1 score
HiCBaM	**ViT**	**18.66%**
	CNN	4.5%
T15	**ViT**	**10.08%**
	CNN	8.29%
T17	**ViT**	**11.65%**
	CNN	10.27%

9.4.2 Text only models

Text only modality experiment represent a standard way of doing NER. It seems that BERT is able to learn all the relevant information from text only and does not benefit from additional modality in a significant way. The experiments with Llama model are more interesting. Llama does not learn from the text data alone as well as BERT and using image modality can help to improve performance. BILSTM underperforms as a text only classifier across all three datasets but additional modality also helps the model achieve better results. The results are still not breath taking.

9.4.2.1 HiCBaM results

The results of text only classification are depicted in Table 9.5. BERT model performs very well over other models and is in fact better than multimodal solution. It seems that BERT is able to learn all the relevant information from the text only and on this particular dataset and using a *vision module* worsens the results.
When it comes to Llama and LSTM, both of these models benefit from image modality. When comparing *Text classifier - Llama* model to the *Linear fusion model-Llama+ViT*, the downgrade in performance is 16.33% which is significant. Same applies for BILSTM model, but the solution using BILSTM in general underperform.

Table 9.5: HiCBaM text results

Model	Macro F1 score
Text classifier - BERT	**80.94%**
Text classifier - Llama	59.67%
Text classifier - BILSTM	23.73%

9.4.2.2 T15 results

The results of text only classification are displayed in Table 9.6. Compared to multimodal solution, BERT performs the same. It is apparent that the model is able to learn from text only data and does not benefit from additional modality when it comes to this dataset. It is worth reminding, that **only one** image is associated with each tweet whereas several words compose a tweet. This means that the text modality is much bigger.
Llama based model performs better across the board without image model. This is very interesting since the model does not learn from text as well as BERT does. It was expected that the additional modality will help the model to perform better but it is not the case. Simpler text model, BILSTM still benefits greatly from additional modality.

Table 9.6: T15 text results.

Model	Macro F1 score
Text classifier - BERT	**75.68%**
Text classifier - Llama	67.55%
Text classifier - BILSTM	42.10%

9.4.2.3 T17 results

The results of text only classification are displayed in Table 9.7. On this dataset, the results are more interesting. A multimodal model, *Partial prediction model - BERT+ViT* outperforms a text only BERT. Although the difference is slight, it appears that the image data which are present in bigger volume (recall that each tweet in this dataset has associated 1 to 5 images) improve the overall performance. Llama benefits from image modality even more. When it comes to *Linear fusion model - Llama+ViT*, the performance on validation subset is **better** by 5.36%.

Table 9.7: T17 text results.

Model	Macro F1 score
Text classifier - BERT	**72.62%**
Text classifier - Llama	69.05%
Text classifier - BILSTM	41.04%

Conclusion 10

This thesis focused on the task of multimodal *historical* named entity recognition, addressing the integration of **textual** and **visual** information to enhance performance of the system. Due to the lack of available datasets for this tasks HiCBaM dataset was created from various historical books using a custom data parsing tool. Two more datasets were used in conducted experiments, Twitter2017 and Twitter2015 for robust evaluation of implemented system.

Three architectures of neural networks were implemented - *Cross attention*, *Linear fusion* and *Partial prediction* model with **different** fusion strategies. These solutions can use various text modules and vision modules to extract information from text and images. We leveraged this fact by using three text modules - **BERT**, **Llama 3.1 8B** and **BILSTM** and two vision modules - **VIT** and **CNN**.

To evaluate the benefit of multimodality, unimodal and multimodal configurations were compared. The results demonstrate that multimodal systems **can outperform** unimodal baselines, though not uniformly. Notably, on the Twitter2017 dataset, the **BERT+ViT Partial Prediction model** consistently outperformed the unimodal BERT-based approach. The experiments using state-of-the-art ChatGPT-4o-mini with multimodal abilities were also conducted and results achieved were inferior to the implemented solutions by a large margin.

Possible extension of this work is to experiment on different datasets or to perform better data preprocessing of used datasets to improve performance. BERT and VIT perform very well for the given task but all the solutions struggle with minority classes. All the implemented models are modular, meaning that *text* and *visual* modules are easily replaceable by different or better model. HiCBaM dataset is small and could be extended by processing more books to make it more robust. Lastly, the custom data parsing tool ("Data annotation tool") developed during this project is minimalistic; further improvements, particularly to the UI/UX, would significantly increase its usability and value for future use.

List of Abbreviations

Abbreviation	Definition
LLM	Large Language Model
NE	Named entity
NER	Named entity recognition
NLP	Natural Language Processing
POS	Part of speech tagging
HMM	Hidden Markov Models
CRF	Conditional Random Fields
IE	Information extraction
IR	Information retrieval
MNER	Multimodal named entity recognition
RE	Relation extraction
MoE	Mixture of Experts
ITA	Image-Text-Alignment framework
MSE	Mean squared error
MLP	Multilayer perceptron
GD	Gradient descent
SGD	Stochastic gradient descent
RMSprop	Running average of squared gradients
LSTM	Long-Short Term Memory
CNN	Convolutional neural network
GPT	Generative pretrained transformer
RNN	Recurrent neural networks
BILSTM	Bidirectional Long-Short Term Memory
ViT	Vision transformer
T17	Twitter17
T15	Twitter15
OCR	Optical character recognition
ETL	Extract-Transform-Load
DAO	Data access object
PEFT	Parameter efficient fine-tuning
LORA	Low-Rank adaptation approach

Bibliography

1. NADEAU, David; SEKINE, Satoshi. A Survey of Named Entity Recognition and Classification. *Lingvisticae Investigationes*. 2007, vol. 30, no. 1, pp. 3–26.
2. KRIPKE, Saul. *Identity and Individuation*. Identity and Necessity. Ed. by MUNITZ, Milton K. New York: New York University Press, 1971.
3. GAUR, Aman. *What is Named Entity Recognition in Natural Language Processing?* MarkovML, 2022. Available also from: `https://www.amygb.ai/author/aman-gaur`.
4. EHRMANN, Maud; HAMDI, Ahmed; LINHARES PONTES, Elvys; ROMANELLO, Matteo; DOUCET, Antoine. Named Entity Recognition and Classification on Historical Documents: A Survey. *ACM Computing Surveys*. 2022, vol. 55, no. 1, pp. 1–41.
5. FATHIMA, Shaistha. *Importance of Named Entity Recognition (NER) in NLP*. MarkovML, 2024. Available also from: `https://www.markovml.com/blog/named-entity-recognition-ner`.
6. JOHNSON, Maggie; ZELENSKI, Julie. *Formal Grammars*. Standford University, 2012. Available also from: `https://web.stanford.edu/class/archive/cs/cs143/cs143.1128/handouts/080%20Formal%20Grammars.pdf`.
7. LECUN, Yann; BENGIO, Yoshua; HINTON, Geoffrey. Deep learning. *nature*. 2015, vol. 521, no. 7553, pp. 436–444.
8. TU, Zhuowen. Learning Generative Models via Discriminative Approaches. In: *2007 IEEE Conference on Computer Vision and Pattern Recognition*. 2007, pp. 1–8. Available from DOI: `10.1109/CVPR.2007.383035`.
9. SCOYOC, Amy Van. A brief primer on Hidden Markov Models. 2022. Available also from: `https://dlab.berkeley.edu/news/brief-primer-hidden-markov-models`.
10. MCCALLUM, Andrew. Efficiently Inducing Features of Conditional Random Fields. *CoRR*. 2012, vol. abs/1212.2504. Available from arXiv: `1212.2504`.

11. LAFFERTY, John; MCCALLUM, Andrew; PEREIRA, Fernando, et al. Conditional random fields: Probabilistic models for segmenting and labeling sequence data. In: *Icml*. Williamstown, MA, 2001, vol. 1, p. 3. No. 2.

12. KLEIN, Dan; SMARR, Joseph; NGUYEN, Huy; MANNING, Christopher D. Named entity recognition with character-level models. In: *Proceedings of the seventh conference on Natural language learning at HLT-NAACL 2003*. 2003, pp. 180–183.

13. BORTHWICK, Andrew Eliot. *A maximum entropy approach to named entity recognition*. New York University, 1999.

14. YANG, Binxia; LUO, Xudong. Recent Progress on Named Entity Recognition Based on Pre-trained Language Models. In: *2023 IEEE 35th International Conference on Tools with Artificial Intelligence (ICTAI)*. IEEE, 2023, pp. 799–804.

15. MALMASI, Shervin; FANG, Anjie; FETAHU, Besnik; KAR, Sudipta; ROKHLENKO, Oleg. MultiCoNER: A large-scale multilingual dataset for complex named entity recognition. *arXiv preprint arXiv:2208.14536*. 2022.

16. CHEN, Xilun; AWADALLAH, Ahmed Hassan; HASSAN, Hany; WANG, Wei; CARDIE, Claire. Multi-source cross-lingual model transfer: Learning what to share. *arXiv preprint arXiv:1810.03552*. 2018.

17. MOON, Seungwhan; NEVES, Leonardo; CARVALHO, Vitor. Multimodal Named Entity Recognition for Short Social Media Posts. In: WALKER, Marilyn; JI, Heng; STENT, Amanda (eds.). *Proceedings of the 2018 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long Papers)*. New Orleans, Louisiana: Association for Computational Linguistics, 2018, pp. 852–860. Available from DOI: 10.18653/v1/N18-1078.

18. HUANG, Yu et al. *What Makes Multi-modal Learning Better than Single (Provably)*. 2021. Available from arXiv: 2106.04538 [cs.LG].

19. WU, Zhiwei et al. Multimodal representation with embedded visual guiding objects for named entity recognition in social media posts. In: *Proceedings of the 28th ACM International Conference on Multimedia*. 2020, pp. 1038–1046.

20. WANG, Xinyu et al. Named entity and relation extraction with multi-modal retrieval. *arXiv preprint arXiv:2212.01612*. 2022.

21. WANG, Xinyu et al. ITA: Image-text alignments for multi-modal named entity recognition. *arXiv preprint arXiv:2112.06482*. 2021.

22. YU, Jianfei; JIANG, Jing; YANG, Li; XIA, Rui. Improving multimodal named entity recognition via entity span detection with unified multimodal transformer. In: Association for Computational Linguistics, 2020.

23. HECHT-NIELSEN, Robert. Theory of the backpropagation neural network. In: *Neural networks for perception*. Elsevier, 1992, pp. 65–93.

24. MARWALA, Tshilidzi; MBUVHA, Rendani; MONGWE, Wilson Tsakane. *Hamiltonian Monte Carlo methods in machine learning*. Elsevier, 2023.

25. MILLER, Steven J. The method of least squares. *Mathematics Department Brown University*. 2006, vol. 8, no. 1, pp. 5–11.

26. EDU, Stanford. *Linear regression*. 2022. Available also from: `https://web.stanford.edu/class/stats202/notes/Linear-regression/Simple-linear-regression.html`. Accessed: 2025-04-29.

27. HOSMER JR, David W; LEMESHOW, Stanley; STURDIVANT, Rodney X. *Applied logistic regression*. John Wiley & Sons, 2013.

28. YANIHA. Sigmoid Activation Function. *Mathematics Department Brown University*. [N.d.]. Available also from: `https://www.codecademy.com/resources/docs/ai/neural-networks/sigmoid-activation-function`.

29. BENGIO, Yoshua; GOODFELLOW, Ian; COURVILLE, Aaron, et al. *Deep learning*. Vol. 1. MIT press Cambridge, MA, USA, 2017.

30. HAN, Song et al. Dsd: Dense-sparse-dense training for deep neural networks. *arXiv preprint arXiv:1607.04381*. 2016.

31. DAVE BERGMANN, Cole Stryker. *What is backpropagation?* IBM, 2022. Available also from: `https://www.ibm.com/think/topics/backpropagation`.

32. VERCACA. *NN-Backpropagation*. 2023. Available also from: `https://github.com/Vercaca/NN-Backpropagation?tab=readme-ov-file`. Accessed: 2025-04-26.

33. RUDER, Sebastian. An overview of gradient descent optimization algorithms. *arXiv preprint arXiv:1609.04747*. 2016.

34. CRYPTO01. *Gradient Descent Algorithm: How Does it Work in Machine Learning?* 2025. Available also from: `https://www.analyticsvidhya.com/blog/2020/10/how-does-the-gradient-descent-algorithm-work-in-machine-learning/`. Accessed: 2025-04-26.

35. BOTTOU, Léon. Stochastic Gradient Descent Tricks. 2012. Available also from: `https://www.microsoft.com/en-us/research/wp-content/uploads/2012/01/tricks-2012.pdf`.

36. GEOFFREY HINTON, Based on work of. *Gradient Descent*. ML explained, 2024. Available also from: `https://ml-explained.com/blog/gradient-descent-explained#gradient-descent-variants`.

37. ANTON, Howard; BIVENS, Irl; DAVIS, Stephen. Calculus: multivariable version. (*No Title)*. 2002.

38. POLYAK, Boris T. Some methods of speeding up the convergence of iteration methods. *Ussr computational mathematics and mathematical physics*. 1964, vol. 4, no. 5, pp. 1–17.

39. DONG, Xiaoyi et al. *CLIP Itself is a Strong Fine-tuner: Achieving 85.7% and 88.0% Top-1 Accuracy with ViT-B and ViT-L on ImageNet*. 2022. Available from arXiv: 2212.06138 [cs.CV].

40. KINGMA, Diederik P; BA, Jimmy. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*. 2014.

41. LOSHCHILOV, Ilya; HUTTER, Frank. Decoupled weight decay regularization. *arXiv preprint arXiv:1711.05101*. 2017.

42. EVERITT, Brian S. *The Cambridge dictionary of statistics*. Cambridge, 2009.

43. MATHWORKS. *What Is Overfitting?* 2022. Available also from: https://www.mathworks.com/discovery/overfitting.html. Accessed: 2025-04-26.

44. HE, Haibo; GARCIA, Edwardo A. Learning from imbalanced data. *IEEE Transactions on knowledge and data engineering*. 2009, vol. 21, no. 9, pp. 1263–1284.

45. KING, Gary; ZENG, Langche. Logistic regression in rare events data. *Political analysis*. 2001, vol. 9, no. 2, pp. 137–163.

46. KOTSIANTIS, Sotiris; KANELLOPOULOS, Dimitris; PINTELAS, Panayiotis, et al. Handling imbalanced datasets: A review. *GESTS international transactions on computer science and engineering*. 2006, vol. 30, no. 1, pp. 25–36.

47. PYKES, Kurtis. *The Vanishing/Exploding Gradient Problem in Deep Neural Networks*. towardsdatascience, 2020. Available also from: https://towardsdatascience.com/the-vanishing-exploding-gradient-problem-in-deep-neural-networks-191358470c11/.

48. PASCANU, Razvan; MIKOLOV, Tomas; BENGIO, Yoshua. On the difficulty of training recurrent neural networks. In: *International conference on machine learning*. Pmlr, 2013, pp. 1310–1318.

49. DARKEN, Christian; CHANG, Joseph; MOODY, John, et al. Learning rate schedules for faster stochastic gradient search. In: *Neural networks for signal processing*. Citeseer, 1992, vol. 2, pp. 3–12.

50. KIRKPATRICK, James et al. Overcoming catastrophic forgetting in neural networks. *Proceedings of the national academy of sciences*. 2017, vol. 114, no. 13, pp. 3521–3526.

51. O'SHEA, Keiron; NASH, Ryan. An introduction to convolutional neural networks. *arXiv preprint arXiv:1511.08458*. 2015.
52. COLLECTIVE, zilliz. *What is a Convolutional Neural Network? An Engineer's Guide*. zilliz. Available also from: `https://zilliz.com/glossary/convolutional-neural-network`.
53. CIREGAN, Dan; MEIER, Ueli; SCHMIDHUBER, Jürgen. Multi-column deep neural networks for image classification. In: *2012 IEEE conference on computer vision and pattern recognition*. IEEE, 2012, pp. 3642–3649.
54. LIPTON, Zachary C; BERKOWITZ, John; ELKAN, Charles. A critical review of recurrent neural networks for sequence learning. *arXiv preprint arXiv:1506.00019*. 2015.
55. HOCHREITER, Sepp. Bridging long time lags by weight guessing and. *Spatiotemporal models in biological and artificial systems*. 1997, vol. 37, p. 65.
56. CHUNG, Junyoung; GULCEHRE, Caglar; CHO, KyungHyun; BENGIO, Yoshua. Empirical evaluation of gated recurrent neural networks on sequence modeling. *arXiv preprint arXiv:1412.3555*. 2014.
57. HOCHREITER, Sepp; SCHMIDHUBER, Jürgen. Long short-term memory. *Neural computation*. 1997, vol. 9, no. 8, pp. 1735–1780.
58. NAN. *What Is Long Short-Term Memory (LSTM)*. Mathworks, 2020. Available also from: `https://www.mathworks.com/discovery/lstm.html`.
59. SCHUSTER, Mike; PALIWAL, Kuldip K. Bidirectional recurrent neural networks. *IEEE transactions on Signal Processing*. 1997, vol. 45, no. 11, pp. 2673–2681.
60. ZVORNICANIN, Enes. *Differences Between Bidirectional and Unidirectional LSTM*. baeldung, 2025. Available also from: `https://www.baeldung.com/cs/bidirectional-vs-unidirectional-lstm`.
61. VASWANI, Ashish et al. Attention is all you need. *Advances in neural information processing systems*. 2017, vol. 30.
62. SHAZEER, N et al. The sparsely-gated mixture-of-experts layer. *Outrageously large neural networks*. 2017.
63. KUCHAIEV, Oleksii; GINSBURG, Boris. Factorization tricks for LSTM networks. *arXiv preprint arXiv:1703.10722*. 2017.
64. DOSHI, Ketan. *Transformers Explained Visually (Part 1): Overview of Functionality*. Towards data science, 2020. Available also from: `https://towardsdatascience.com/transformers-explained-visually-part-1-overview-of-functionality-95a6dd460452/`.

65. SENNRICH, Rico; HADDOW, Barry; BIRCH, Alexandra. Neural machine translation of rare words with subword units. *arXiv preprint arXiv:1508.07909*. 2015.

66. MIKOLOV, Tomas; CHEN, Kai; CORRADO, Greg; DEAN, Jeffrey. Efficient estimation of word representations in vector space. *arXiv preprint arXiv:1301.3781*. 2013.

67. RAMACHANDRAN, Prajit et al. Stand-alone self-attention in vision models. *Advances in neural information processing systems*. 2019, vol. 32.

68. HE, Kaiming; ZHANG, Xiangyu; REN, Shaoqing; SUN, Jian. Deep residual learning for image recognition. In: *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2016, pp. 770–778.

69. DOSOVITSKIY, Alexey et al. An image is worth 16x16 words: Transformers for image recognition at scale. *arXiv preprint arXiv:2010.11929*. 2020.

70. DEVLIN, Jacob; CHANG, Ming-Wei; LEE, Kenton; TOUTANOVA, Kristina. Bert: Pre-training of deep bidirectional transformers for language understanding. In: *Proceedings of the 2019 conference of the North American chapter of the association for computational linguistics: human language technologies, volume 1 (long and short papers)*. 2019, pp. 4171–4186.

71. WU, Yonghui et al. *Google's Neural Machine Translation System: Bridging the Gap between Human and Machine Translation*. 2016. Available from arXiv: 1609.08144 [cs.CL].

72. TOUVRON, Hugo et al. *LLaMA: Open and Efficient Foundation Language Models*. 2023. Available from arXiv: 2302.13971 [cs.CL].

73. CHUNG, Hyung Won et al. Scaling instruction-finetuned language models. *Journal of Machine Learning Research*. 2024, vol. 25, no. 70, pp. 1–53.

74. HOFFMANN, Jordan et al. *Training Compute-Optimal Large Language Models*. 2022. Available from arXiv: 2203.15556 [cs.CL].

75. GRATTAFIORI, Aaron et al. The llama 3 herd of models. *arXiv preprint arXiv:2407.21783*. 2024.

76. LU, Di; NEVES, Leonardo; CARVALHO, Vitor; ZHANG, Ning; JI, Heng. Visual attention model for name tagging in multimodal social media. In: *Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*. 2018, pp. 1990–1999.

77. WANG, Yue; LI, Jing; LYU, Michael R; KING, Irwin. Cross-media keyphrase prediction: A unified framework with multi-modality multi-head attention and image wordings. *arXiv preprint arXiv:2011.01565*. 2020.

78. CROCKFORD, Douglas; MORNINGSTAR, Chip. Standard ECMA-404 The JSON Data Interchange Syntax. In: 2017. Available from DOI: 10.13140/RG.2.2.28181.14560.

79. SANG, Erik F; DE MEULDER, Fien. Introduction to the CoNLL-2003 shared task: Language-independent named entity recognition. *arXiv preprint cs/0306050*. 2003.

80. JACOB MUREL PH.D., Eda Kavlakoglu. *What is stemming?* IBM, 2024. Available also from: https://www.ibm.com/think/topics/stemming.

81. MORI, Shunji; NISHIDA, Hirobumi; YAMADA, Hiromitsu. *Optical character recognition*. John Wiley & Sons, Inc., 1999.

82. SMITH, Ray. An overview of the Tesseract OCR engine. In: *Ninth international conference on document analysis and recognition (ICDAR 2007)*. IEEE, 2007, vol. 2, pp. 629–633.

83. AI, Jaided. *EasyOCR - Ready-to-use OCR with 80+ supported languages* [https://github.com/JaidedAI/EasyOCR]. 2020. Accessed: 2025-04-10.

84. OPENAI. *GPT-4o System Card* [https://arxiv.org/abs/2410.21276]. 2024. Accessed: 2025-05-10.

85. WEISS, Karl; KHOSHGOFTAAR, Taghi M; WANG, DingDing. A survey of transfer learning. *Journal of Big data*. 2016, vol. 3, pp. 1–40.

86. HENDRYCKS, Dan; GIMPEL, Kevin. Gaussian error linear units (gelus). *arXiv preprint arXiv:1606.08415*. 2016.

87. THOMAS CARR, Milos Simic. GELU Explained. 2025. Available also from: https://www.baeldung.com/cs/gelu-activation-function.

88. THEODOROU, Vasileios; ABELLÓ, Alberto; LEHNER, Wolfgang; THIELE, Maik. Quality measures for ETL processes: from goals to implementation. *Concurrency and computation: practice and experience*. 2016, vol. 28, no. 15, pp. 3969–3993.

89. LOSHCHILOV, Ilya; HUTTER, Frank. *SGDR: Stochastic Gradient Descent with Warm Restarts*. 2017. Available from arXiv: 1608.03983 [cs.LG].

90. HU, Edward J. et al. *LoRA: Low-Rank Adaptation of Large Language Models*. 2021. Available from arXiv: 2106.09685 [cs.CL].

91. JACOB, Benoit et al. *Quantization and Training of Neural Networks for Efficient Integer-Arithmetic-Only Inference*. 2017. Available from arXiv: 1712.05877 [cs.LG].

92. OPITZ, Juri; BURST, Sebastian. *Macro F1 and Macro F1*. 2021. Available from arXiv: 1911.03347 [cs.LG].

List of Figures

List of Tables

Attachment 1: User manual

Following chapter contains information on running the two main applications, **Multimodal-named-entity-recognition** and **Cutie_parser**.

Multimodal-named-entity-recognition

To run the core of this thesis please follow this guide. A prerequisite for running any of the application is a Hugging Face account. Skip the first section if you already have an account.

Setting up Hugging Face account

Join the Hugging Face community here. After successful registration, confirm email. With confirmed email, generate fine-grained access token. It is **necessary** to check the
Read access to contents of all public gated repos you can access option.
The token is used for **authentication** and **authorization** meaning that it must be a fine-grained token. Hugging Face uses Read/Write only for authorization. **Save** this token.

Access to Llama 3.1 8B model

Request access to Llama model. This is **needed** otherwise Llama model will not work in experiments.

Anaconda environment

For quality of life use Anaconda's environments to avoid version war with old versions of python and libraries.
In the root directory of the application run
`bash create_conda_env.sh`
`conda activate multimodal_named_entity`

`pip3 install -r requirements.txt`
Running the commands above should produce working environment with necessary dependencies.

Downloading datasets

To follow the enforced archive structure, refer to directory **Input_data**. In the directory run the shell script:
`bash download_dataset.sh`
This will download and unzip Twitter17, Twitter15 and HiCBaM datasets.

Running preprocessor

Refer back to root directory of the application, Multimodal-named-entity-recognition.
In there, refer to `src` directory.
Run the shell script which runs the preprocessor `bash preprocessor.sh`
After the script is completed, a directory `Multimodal-named-entity-recognition/dataset` will contain preprocessed datasets.

Running neural networks

In the root directory, paste your hugging face access token to `token` file. Beware of extra `\n` character at the end of the token in the file. Any whitespace has to be removed otherwise the authentication will fail.
To run all[1] the experiments, run the following command `python3 experiments.py`

Configuration file

If you wish to modify which experiments, modify the `experiments_config.json` file. Refer to `README.md` in the same directory for description of attributes.

Cutie_parser

Running this application is much simpler. Refer to the main directory of the application - **Cutie_parser**.
Run `pip3 install -r requirements.txt`
After all necessary dependencies are installed, run `python3 main.py`
The application accepts one argument *input_dir* which should be a directory with images. Default value is the directory `dummy_data` located in the root folder of this application with some sample images.

[1] Runtime is roughly 8 days.

Attachment 2: ETL UML