Democratic and Popular Republic of Algeria

Ministry of Higher Education and Scientific Research

Engineering School of

Computer Sciences and Digital Technologies

End-of-Studies Project

For the award of the State Engineer's Degree in Computer Science

Specialization: Data Science and Artificial Intelligence

Image Text Similarity using Deep Learning Object Detection and Word Spotting Approach

---

Authors:

Billal MOKHTARI

Lilia MAHDID

Defended in front of the jury composed of:

Mr. Bellal HAFHOUF	MAB	ESTIN	*President*
Mme. Meroua DAOUDI	MCB	ESTIN	*Examiner*
Mr. Abdelkader TAGMOUNI	PhD Student	ESTIN	*Examiner*
Mr. M'Hamed Amine HATEM	PhD Student	ESTIN	*Examiner*
Pr. KHERBACHI Hamid	PROF	ESTIN	*Supervisor*

Promotion: 2024/2025

Dedication

To the most precious person in my life, my source of motivation, the reason I am here today, the one who gave me life, taught me, dressed me, fought for me, held me, prayed for me, kissed me, but most importantly loved me unconditionally. The one who awaited this day as eagerly as I did, to my lifelong friend and beloved mother.

To my idol in this life, my light in the darkness, my strength and pillar in all the obstacles I face, to my shield, my dear father who was always by my side to encourage me and say "you can do it."

To my dear brother Walid, with whom I shared all my childhood, the one who encouraged me and was always proud of me.
To my cherished little sisters: Nouara, Melissa, and Alicia, who have supported me more than anyone at every stage, who have been there from the start, for all their love and care.

To my grandparents and guardian angels, may God welcome them into His vast paradise excet Setti Fadhma, may Allah have mercy on her, for their support despite the little they understood of what I was doing.

To my dear former high school physics teacher, Mrs. Sofia, who has always been with me and has has grown to feel like an older sister to me. I deeply value her presence in my life.

To my best friend Hafidha, who accompanied me throughout this journey and turned all the difficult moments into beautiful memories. Despite the physical distance between us, it has never hindered our friendship; in fact, it has strengthened our bond even more.

And I dedicate this work especially to my project partner, with whom I accomplished this work.

To all the people dear to my heart.

To you, all the years I've spent studying, working hard, and achieving success. Thank you for being a part of my life.

I dedicate this modest work.

- Lilia

Acknowledgment

First and foremost, **w**e thank **ALLAH**, the Almighty, for granting us the courage, willpower, and patience necessary to complete this work.

We would like to express our deep gratitude to our parents, both our mother and father, and to all the members of our family for their unwavering support and encouragement throughout this journey.

We extend our heartfelt thanks to the LIMOS laboratory (Laboratoire d'Informatique, de Modélisation et d'Optimisation des Systèmes) in Clermont-Ferrand, France, for providing an excellent research environment and the resources needed for this project.

We are immensely grateful to our supervisor, Mr. KHERBACHI Hamid, Professor at ESTIN, for his invaluable guidance, unwavering support, and confidence in us. It has been a great pleasure and honor to be a part of your team.

We also wish to thank the Higher National School of Computer Science and Digital Technologies (ESTIN - Bejaia) for the enriching education and invaluable experiences provided during the years we spent there.

We express our sincere thanks to the members of the jury for the honor they bestow upon us by taking the time to read and evaluate this work.

Lastly, we wish to thank everyone who contributed, directly or indirectly, to the realization of this project.

Abstract

With the fast expansion of Deep Learning, multi-modal models have become increasingly popular for tasks requiring complex data inputs. Content generation—such as image, video, or text generation—as well as recent object detection and segmentation methods, frequently use Large Language Models (LLMs). This project focuses on enhancing image and text similarity measures, aiming to improve the *CLIP (Contrastive Language-Image Pre-training)* method by examining the impact of object semantics on image descriptions. Our approach, named *ODITS (Object Driven Image and Text Similarity)*, uses the CLIP model pre-trained with the ViT-B/32 architecture, which is subsequently fine-tuned for our specific purposes. We evaluated the performance of the fine-tuned model using modified metrics, selecting the optimal checkpoint based on precision to minimize false associations between descriptions and images. Our findings indicate that this optimal checkpoint is 10% more precise than the original checkpoint. The weights from this model will be integrated into ODITS's shared components with CLIP, providing a robust starting point for further optimization. The research component of the ODITS model, including theoretical and preliminary analysis, is also discussed, providing insights into its potential and areas for future development.

Keywords : Image and Text Similarity, Multi-Modal models, CLIP, ODITS, Zero-Shot Learning, Large Language Models, Object Detection, Image Segmentation, Text Recognition and Spotting

Contents

List of Figures

List of Tables

List of Acronyms

AI Artificial Intelligence

CRN Character Recognition Network

ESTextSpotter Explicit Synergy-based Text Spotting Transformer

FCN Fully Convolutional Network

GO Go Token

IoU Intersection over Union

LN Layer Normalization

MAE Masked Autoencoders

MLP Multi-Layer Perceptron

MSA Multi-Head Self-Attention

NLP Natural Language Processing

REM Receptive Enhancement Module

RES-5 Feature Map

SAM Segment Anything Model

STR Scene Text Recognition

ViT Vision Transformer

ViTSTR Vision Transformer Scene Text Recognition

VLC Vision-Language Communication Module

General introduction

Background and Context

The project focuses on finding the best way to describe images with text. The goal is to measure how well an image corresponds to a piece of text (or vice versa). It combines both *Computer Vision* and *Natural Language Processing (NLP)*. These fields were developed separately until researchers began to explore the possibilities of merging them. This convergence is driven by several major factors:

- **Advancements in Deep Learning Architectures:** Traditional deep learning techniques, such as *Convolutional Neural Networks (CNNs)* by O'shea and Nash [2015] used extensively in Computer Vision, and *Recurrent Neural Networks (RNNs)* by Schmidt [2019] used in NLP, have limitations due to their assumptions about data. Researchers have since developed more sophisticated architectures and mathematical models to capture data dependencies more effectively. These newer models, known as transformers, were introduced by Vaswani et al. [2017] and are based on the concept of *Attention*.

- **Large-Scale Datasets:** Advances in storage, data collection technologies, and database concepts have led to the creation of larger datasets, both in terms of the number of examples (vertical scale) and the number of features (horizontal scale). Various mathematical concepts, such as Graph Theory, have been explored to represent the hierarchical nature of this data.

- **Multi-modal Learning:** These advancements have prompted researchers to develop models that can process multiple input types and produce multiple outputs. Consequently, new components have been designed to learn cross-features between data of different types, such as images and text. An example is the *Feature Enhancer Layer* used in **GroundingDINO**, introduced by Liu et al. [2023].

- **Transfer Learning:** Transfer learning involves transferring knowledge from one dataset to another. One challenge in model optimization is selecting hyperparameters that lead to the best local optima. Pre-trained models, which have already learned useful characteristics from other datasets, can be adapted for new tasks. By using these pre-trained models with the same weights and modifying them according to the specific task, we can achieve more efficient training. For more information, see the survey on Transfer Learning by Zhuang et al. [2020].

The integration of image and text data opens up several possibilities across different industries. For instance, some known search engines like Google and Bing enable users to describe the content and texture of the images they are looking for, rather than

searching for correspondences between texts. Consequently, this approach enhances search accuracy and user experience. Moreover, sufficiently trained models can be used to associate significant text with an image, which not only enhances the model itself but also facilitates the creation of new models. Additionally, the article "*CLIP in Medical Imaging: A Comprehensive Survey*" by Zhao et al. [2023b] examines the use of CLIP, as explained by Radford et al. [2021], for tasks such as image retrieval, report generation, and zero-shot learning, thereby demonstrating its potential to improve diagnostic accuracy and efficiency in medical imaging. Furthermore, using image-text similarity, we can extract meta-knowledge that assists in other machine learning tasks like classification, highlighting the broader applicability and impact of these advancements. In the realm of autonomous driving, integrating image and text data can enhance the vehicle's ability to understand and interpret complex scenes by associating textual descriptions with visual elements (traffic signs), thereby improving decision-making and safety.

The development of image-text similarity techniques has undergone significant advancements over the years. The early stages in the 1990s to 2000s focused on text-based image retrieval (TBIR) and content-based image retrieval (CBIR). These methods were relatively basic and relied on manual annotation and visual content indexing Stafylopatis and Likas [1992], Smeulders et al. [2000]. The 2010s saw the emergence of bag-of-words models and latent semantic analysis Sivic and Zisserman [2003], Monay and Gatica-Perez [2003], followed by the deep learning revolution, which introduced convolutional neural networks (CNNs) for images and recurrent neural networks (RNNs) for text Krizhevsky et al. [2017]. This period also saw the development of deep image-text embedding models Frome et al. [2013]. The late 2010s to present has been marked by the introduction of advanced techniques such as attention mechanisms, transformers, joint embedding models, CLIP, ALIGN, multimodal transformers, and a focus on zero-shot learning and generalization Vaswani et al. [2017], Radford et al. [2021], Jia et al. [2021].

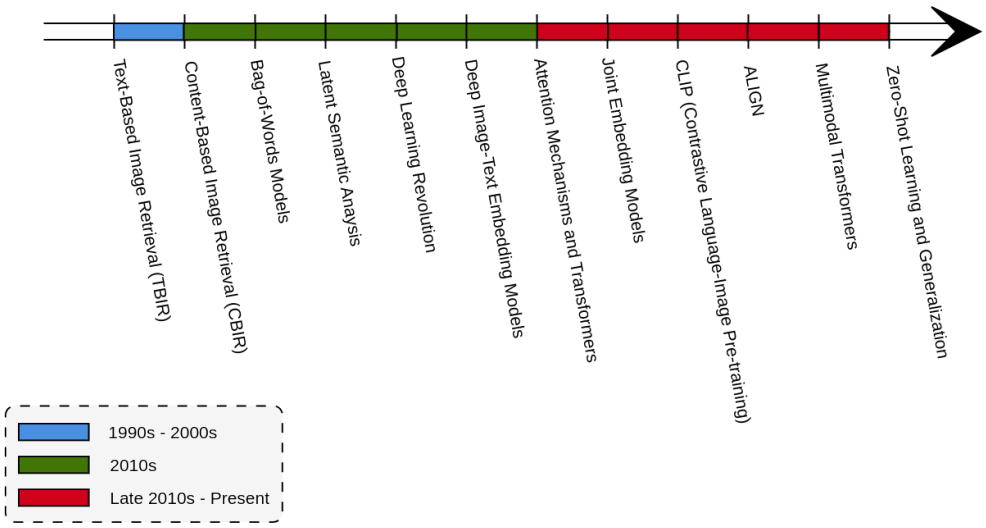


Figure 1: Historical Roadmap of Image-Text Similarity Techniques. **(source: compiled by us)**

This figure illustrates the progression of image-text similarity techniques, highlighting key advancements and their corresponding time periods. The timeline is divided into three main periods: the 1990s to 2000s, which include text-based and content-based image

retrieval methods; the 2010s, which introduced bag-of-words models, latent semantic analysis, and deep learning methods; and the late 2010s to present, characterized by sophisticated models like CLIP, ALIGN, and multimodal transformers.

Problem Statement

Zhao et al. [2023b] have designed an efficient technique to describe images using a set of texts. In this project, our goal is to understand the boundaries and limitations of previously developed methods, particularly CLIP. We aim to go beyond simple image and text similarity by investigating the effect of the presence of objects within an image. Specifically, we want our model to focus on the meaning of each object and how it is described, providing a better interpretation of the image by considering the dependencies between the objects and their associated descriptions. We strive to enhance the explainability of our model by identifying the most relevant objects that maximize the similarity between the text and the image. Here are some questions we aim to answer and key points to consider in this project:

- How can we evaluate CLIP and its fine-tuned models using the pre-trained ViT-B/32 checkpoint?
- Does the nature of the dataset impact the optimization process?
- In real-time applications, minimizing false descriptions is crucial. For instance, in autonomous driving, we do not want the model to confuse a speed limit sign with a stop sign. Therefore, precision is the most important metric for us. Given that CLIP's outcomes are probabilities, how can we adapt these metrics for probabilistic outputs, and what are their interpretations ?
- How do we select the best checkpoint among various options?
- How can we theoretically ensure that our proposed model will function as intended? What kind of certification can we provide to support this?
- Are there efficient approaches for image segmentation and detection in the current literature?

The most challenging aspect of our model lies in its time complexity, particularly if we intend to deploy it on embedded systems. This complexity arises from the necessity of employing two separate models for effective prediction: one for object segmentation and another is ODITS 3. Certain features of this project will not be included in the current presentation due to the substantial time and resource investments they require. Instead, these aspects will be outlined as areas for future exploration and development.

Objectives and Goals

As mentioned in the previous section, the objectives of our project can be categorized into *short-term goals* and *long-term goals*. The short-term goals relate to what we implemented and the results achieved within the scope of this project, whereas the long-term goals serve as future research directives.

Short-Term Objectives

Our short-term objectives focus on defining the main modalities of the described problem, specifying the desired outcomes, testing our hypotheses, and determining how our efforts contribute to solving the problem statement. These objectives include:

- Evaluate the CLIP method, using the *ViT-B/32* architecture with optimized weights as explained in 2, on the SVT (Street View Text) dataset 2.1.1. We will then fine-tune it by adjusting hyperparameters to achieve multiple checkpoints that yield better results than the vanilla model.
- Design customized metrics for the evaluation process to address the problem of CLIP's outcome being a distribution of density.
- Focus on customized precision as a key metric for evaluating our models. In addition to accuracy and the F1-Score, we aim to minimize false positives. We will verify that the selected checkpoint has the highest accuracy, precision, recall, and F1-Score compared to the non-fine-tuned checkpoint. Our goal is it to have the highest precision and F1-Score among all fine-tuned models. Target a 10% increase in Precision.
- Design a theoretical model called *ODITS (Object-Driven Image and Text Similarity)* 3, which includes additional components to capture dependencies between the entire image, the objects extracted from it, and the text.
- Detect every object in the image using the *SAM (Segment Anything Model)* by Kirillov et al. [2023]. We will adjust multiple parameters of this model to ensure that the number of objects in each image follows a Gaussian density distribution (neither skewed to the left nor to the right). The Anderson-Darling statistical test will be used to verify the normality of the distribution at a significance level of at least $\alpha = 5\%$.

Long-Term Objectives

Given that *ODITS* 3 is a novel technique, it will be subjected to a comprehensive scientific evaluation. We will formulate hypotheses, implement the initial version, interpret the results, and study its limitations until we achieve a valuable product. The objectives are similar to the short-term goals, but with a focus on long-term advancements. This method should outperform the previous CLIP method, which relies solely on text to describe images. Specifically, it should improve previously described metrics.

- Enhance the accuracy of matching images with textual descriptions by integrating object detection and embeddings.
- Aim to be at least 1× more efficient than the existing CLIP method. Efficiency here is defined as the model's ability to achieve higher accuracy with fewer images processed. The higher the efficiency, the fewer images are needed to reach the same accuracy level, indicating a more efficient learning process. The Figure 2 shows Comparison of Zero-Shot ImageNet accuracy across different models and number of images processed.

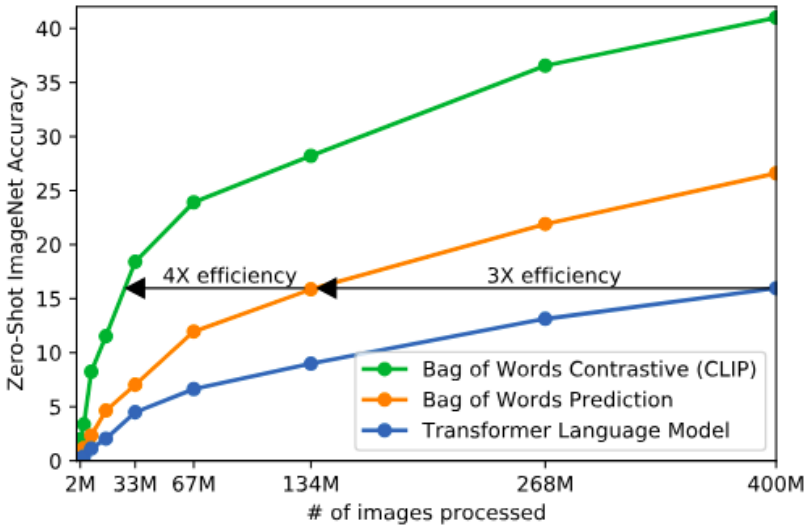


Figure 2: Comparison of Zero-Shot ImageNet Accuracy Across Different Models and Number of Images Processed. **(source: Zhao et al. [2023b])**

- Ensure the model can generalize well across various datasets and domains without significant performance degradation. We will start optimization by leveraging some of the pre-existing optimized weights available in both the existing architecture and our novel approach.
- Develop the model to be efficient enough for real-time applications, aiming for processing times under 100 milliseconds per image-text pair on embedded system GPUs. This will enable its use in real-world systems like autonomous driving or surveillance.

Methodology

Our approach is divided into two dependent parts. In the first part, we take the CLIP model and fine-tune it 10 times, changing the number of epochs. The training data has 100 pairs of (image, text) examples. This number is sufficient as it has already proven its efficiency without fine-tuning. The losses, hyperparameters, and system resources are tracked with *Neptune*. Each time we record a reduction in the loss, we save the corresponding checkpoint. Checkpoints are not saved at each epoch due to storage limitations, as we only have 100GB available with the free version of Neptune. Each model is tracked in a single experiment object called a **Run**. We evaluate all the models on an evaluation set of 250 examples using the metrics presented in 2.3. The results are also saved in Neptune to have a centralized repository where any data can be easily retrieved by the user. The model with the best checkpoint is deployed on a Streamlit server (we use local addresses for the moment).

The second part of the approach uses **Object Driven Image and Text Similarity (ODITS)** 3. Some ODITS components are the same as the CLIP ones, with additional components responsible for capturing the features between objects and their associated descriptions in the image. We load the previously saved best checkpoint's weights from the Neptune platform to the identical components. For the others, we use one of the uniform initialization methods such as **Xavier**, **Kaiming**, or simulation methods like

MRG32k3a & MRG32kp, **Mersenne Twisters (MT19237, SFMT, MTGP) & WELL families of generators**, or **MLFG_6331_64** introduced respectively by Glorot and Bengio [2010], He et al. [2015], L'Ecuyer [2015], Matsumoto and Nishimura [1998], L'Ecuyer [1999], which are known as the best pseudo-random number generators at the moment. Alternatively, we can use the default *PyTorch* initialization. ODITS will undergo a complete process of training, evaluation, and testing until we get a valid model. Once again, the intermediate results are saved in Neptune. A valid iteration is considered completed when the model is trained, evaluated, and tested properly. The model takes three types of inputs: the image and its derived objects extracted using the Segment Anything Model (SAM), and the text. Two alternatives could be considered:

- Compute and save beforehand all the required data for the training time to avoid losing time in computing them during training, like SAM outcomes. The advantage is that we gain in time complexity but require a lot of storage.
- Compute the necessary data during training. The disadvantage is that the training will last longer, but we gain in storage because we store the intermediate results in the CPU or GPU memory.

Figure 3 shows the overall pipeline of our approach.

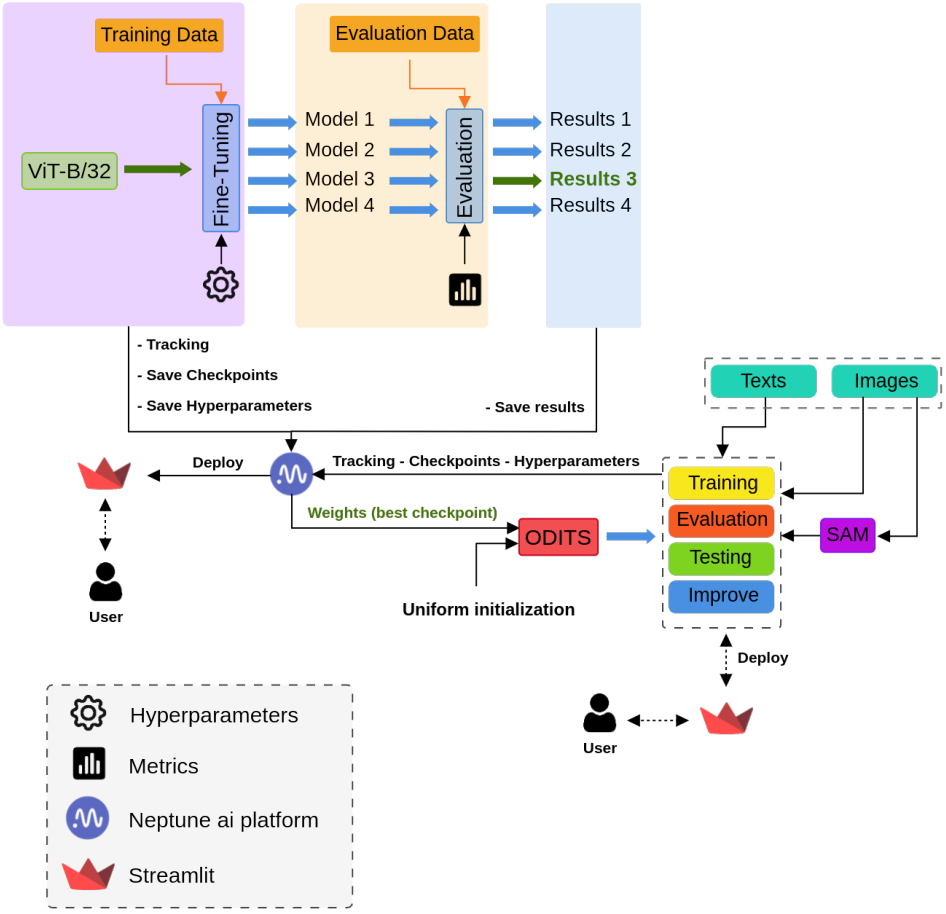


Figure 3: Overall Schema of our Approach **(source: compiled by us)**

Structure of the Report

The project's report is divided into three chapters and a conclusion.

In the first chapter, 1, we define some general concepts necessary to understand the technical aspects of the project. The concepts are presented in a logical order. We then present the State-of-the-Art (SOTA) by discussing previous related works in greater detail. This chapter consists of three main elements: object detection and segmentation techniques, where we cite GroundingDINO and the Segment Anything Model (SAM); text recognition and spotting methods, such as ViTSTR, ESTextSpotter, and DeepSolo; and image and text similarity, focusing particularly on CLIP.

In the second chapter, 2, we evaluate existing image-text similarity approaches, using CLIP as our case study. We first present the dataset used for this purpose statistically. We describe the experimental setup and the environment in which we ran our experiments. We then explain the evaluation metrics, modified to account for the outputs in the format of probability density. Finally, we present the results of using CLIP with the ViT-B/32 checkpoint, tested on 250 examples, both without fine-tuning and with fine-tuning, to select the checkpoint that provides the best outcomes based on the customized metrics we introduced.

In the last chapter, 3, we explicitly explain our research work called *Object Driven Image and Text Similarity*. We provide mathematical notation and detail the three main parts of the model: the Encoding Phase, the Object-Text Similarity Matrix, and Cross-Feature Learning. Preliminary results are presented in the final section of the chapter, including an analysis of segmented images, the distribution of objects in each image in the dataset, and a theoretical comparison of our approach to the CLIP method.

Finally, the conclusion summarizes everything we have done in this project. We outline the research directives and highlight the aspects we will focus on to achieve the long-term objectives as future work.

Chapter 1

Litterature Review

Introduction

This chapter explores contemporary techniques in object detection, segmentation, text spotting, and recognition. We begin by defining fundamental concepts that provide a basis for understanding the transition from fine-grained to coarse elements. Subsequently, we trace the evolution of Machine Learning paradigms and Deep Learning architectures, discussing the advancements that have enabled these techniques. Furthermore, we examine the role of recent language models in visual feature extraction, emphasizing their transformative impact on these fields.

We provide a detailed examination of these methods, including *GroundingDINO* by Liu et al. [2023], which builds upon the *DINO* model and its enhanced version *DINOv2*, developed by Caron et al. [2021] and Oquab et al. [2023] respectively. This model processes images and textual prompts to draw bounding boxes around objects, deriving descriptions from the text. Additionally, we explore Open-Set object detection models as discussed in Section 1.1.

Our analysis includes the Segment Anything Model (SAM) by Kirillov et al. [2023] for segmentation tasks. SAM, based on transformers, segments images using prompts such as points, bounding boxes, or textual descriptions. It extracts the necessary information from these inputs to accurately segment the relevant objects.

Advancements in scene text recognition (STR) leverage transformers, notably the Vision Transformer (ViT) by Atienza [2021], which surpasses traditional deep networks through extensive pre-training on large datasets. ViT models transform input images into patches, convert them into 1D vector embeddings, and predict character sequences using a transformer encoder. *DeepSolo* by Ye et al. [2023] integrates text detection and recognition, employing a fully convolutional network to detect text regions and a character recognition network for text within those regions, utilizing multi-scale features and Bezier curves for accurate text spotting. *ESTextSpotter* by Huang[1*] et al. [2023] employs a synergy-based transformer framework combining detection and recognition queries processed through task-aware decoders and vision-language communication modules for robust text instance detection and recognition.

Moreover, *CLIP* by Radford et al. [2021], a model trained on 400 million image-text pairs, facilitates zero-shot learning by associating visual concepts with natural language, enabling competitive performance without additional training on new tasks. CLIP's method significantly improves performance on various datasets, demonstrating the efficacy of using natural language to specify visual concepts, particularly in data-scarce scenarios.

These models represent the state-of-the-art in integrating language and vision, pushing the boundaries of what is possible in object detection, segmentation, and text recognition.

1.1 General Concepts

Before delving into the core of the methods, we first explain some concepts and terms that might be important to understand the content of this project. Note that the following definitions are only valid in the context of Deep Learning.

- **Object detection**: it is a *Computer-Vision* paradigm that aims to develop learning techniques to identify existing objects in the image, localizing them and classify them.
- **Segmentation**: it's viewed as a fine-grained approaches because each pixel in the image is classified among a set ob objects. We distinguish two categories of segmentation :
  - *Semantic Segmentation*: This method assumes there is at most one instance of each object type. For example, if there are two people in the image, it will detect them together and label them both as "person." Formally, each pixel x_{ij} is labeled with $z_{ij} \in \mathcal{C}$, where $\mathcal{C} = \{c_1, \ldots, c_n\}$ is a set of classes. The number of classes n can be either finite or infinite, depending on the approach used.
  - *Instance Segmentation*: Contrary to the previous approach, this method goes into more detail to extract each instance of an object separately. For example, the first person will be labeled as **('person', 1)**, and the second as **('person', 2)**. Formally, each pixel is associated with a label consisting of the object type (the class) and the object identifier (the instance). We denote these labels as $z_{ij} \in \mathcal{C} \times \mathbb{N}$.
- **Closed-set object detection**: The number of classes is fixed, and the classes are predefined. This approach is based on classical model architectures such as *Fast R-CNN*, *Faster R-CNN*, *Mask R-CNN*, *PointRend*, *OneFormer*, and *YOLO*, introduced by Girshick [2015], Ren et al. [2015], He et al. [2017], Kirillov et al. [2020], Jain et al. [2023], and Redmon et al. [2016], respectively.
- **Open-set object detection**: This is a modern paradigm based on *zero-shot learning*. It uses transformers and, more specifically, Large Language Models (LLMs). We present two methods in section 1.2: *GroundingDINO*, proposed by Liu et al. [2023], for object detection, and *Segment Anything*, proposed by Kirillov et al. [2023], for segmentation.
- **Few-Shot Learning (FSL)**: Few-shot learning is a machine learning paradigm that enables a pre-trained model to learn new categories from only a few examples of each. This approach often overlaps with the concept of meta-learning.
- **One-Shot Learning (OSL)**: One-shot learning is often illustrated using *Siamese Networks*. Traditionally, to train a model to recognize faces, we would need thousands of examples for each face. However, we rarely have that many examples of a

single person. One-shot learning addresses this by training the model to recognize similarities between pairs of faces rather than memorizing the faces themselves. This way, the model only needs one example of each face.

- **Zero-Shot Learning (ZSL):** Unlike few-shot and one-shot learning, zero-shot learning doesn't require any training examples to learn new concepts. It addresses the limitations of traditional object detection, which requires a predefined set of classes to make predictions. Modern techniques have been developed to enable models to predict categories that were not included in the training data.
- **Large Language Models (LLMs):** According to Zhao et al. [2023a], large language models (LLMs) refer to Transformer language models that contain hundreds of billions (or more) of parameters, which are trained on massive text data, such as GPT-3, PaLM, Galactica, and LLaMA. LLMs exhibit strong capacities to understand natural language and solve complex tasks (via text generation). To have a quick understanding of how LLMs work, this part introduces the basic background for LLMs, including scaling laws, emergent abilities and key techniques.
- **End-to-End Text Spotting**: This approach integrates text detection and recognition into a unified model, simplifying the identification and reading of text within images in different contexts.

1.2 State of the Art

In this section, we present Deep Learning methods to detect and segment objects and backgrounds in images by merging some novel techniques. We use *GroundingDINO*, an open-set method introduced by Liu et al. [2023], to detect object bounding boxes at certain confidence levels. We also use the *Segment Anything Model*, introduced by Kirillov et al. [2023], to segment every object in the image at the pixel level.

Almost all the methods used are based on recent approaches like transformers and Large Language Models (LLMs).

1.2.1 GroundingDINO: Marrying DINO with Grounded Pre-Training for Open-Set Object Detection

GroundingDINO is an *open-set object detector* that is able to detect new objects, unlike closed-set methods that require the training on predefined objects. It detects arbitrary objects specified by human language inputs. It's a *dual-encoder-single-decoder* architecture since it takes bi-modal inputs, texts and images and returns only the language-aware object region embeddings or bounding-boxes, that are classified into new categories with associated probabilities.

The figure 1.1 shows the architecture of GroundingDINO. The model processes both text and image inputs, enhances their features using cross-attention mechanisms, and decodes these features to produce final outputs. The outputs are evaluated using contrastive and localization losses. It is constituted of three principle parts:

1. **Backbone**: the image backbone encodes the input images into vanilla image features while the text backbone encodes the input texts into vanilla text features.

2. **Neck**: feature enhancement is performed on both vanilla image and text features. Each feature enhancement layer is a stack of self-attention and text-to-image cross-attention layers. The inputs are fed as Query, Key and Value to the self-attention layers. However, in cross-attention layers, one time images are fed as query and texts as Keys and values, another time it's the contrary. This technique helps both inputs to share the information between each others.

3. **Head**: At this stage, only important features should be selected as candidates for decoder queries. These latents are calculated to select the maximum values in each of the image and text embedding matrices. Let $M_{\text{image}} \in \mathbb{R}^{m \times n_{\text{image}} \times d}$ and $M_{\text{text}} \in \mathbb{R}^{m \times n_{\text{text}} \times d}$ be the updated image feature matrix and the updated text feature matrix, respectively, where m is the number of examples in the batch, n_{image} and n_{text} are respectively the number of image and text tokens, and d is the dimension of each token.

   The logits are computed as:

$$\text{logits} = M_{\text{image}} \cdot M_{\text{text}}^{T} \in \mathbb{R}^{m \times n_{\text{image}} \times n_{\text{text}}}.$$

   We only select the maximum values in each feature matrix according to:

$$\max_{i} \text{logits} \in \mathbb{R}^{m \times n_{\text{image}}},$$

   where i represents the index along the last dimension. The reason Liu et al. [2023] have chosen to eliminate the dimension representing the text rather than the one representing the images is that the processing will be applied on images. The logits will be fed into a *decoder layer* that first passes through Self-Attention, then image and text Cross-Attention that take the updated image an text features respectively. These updated cross-modality queries represent regions of interests or the bounding boxes.

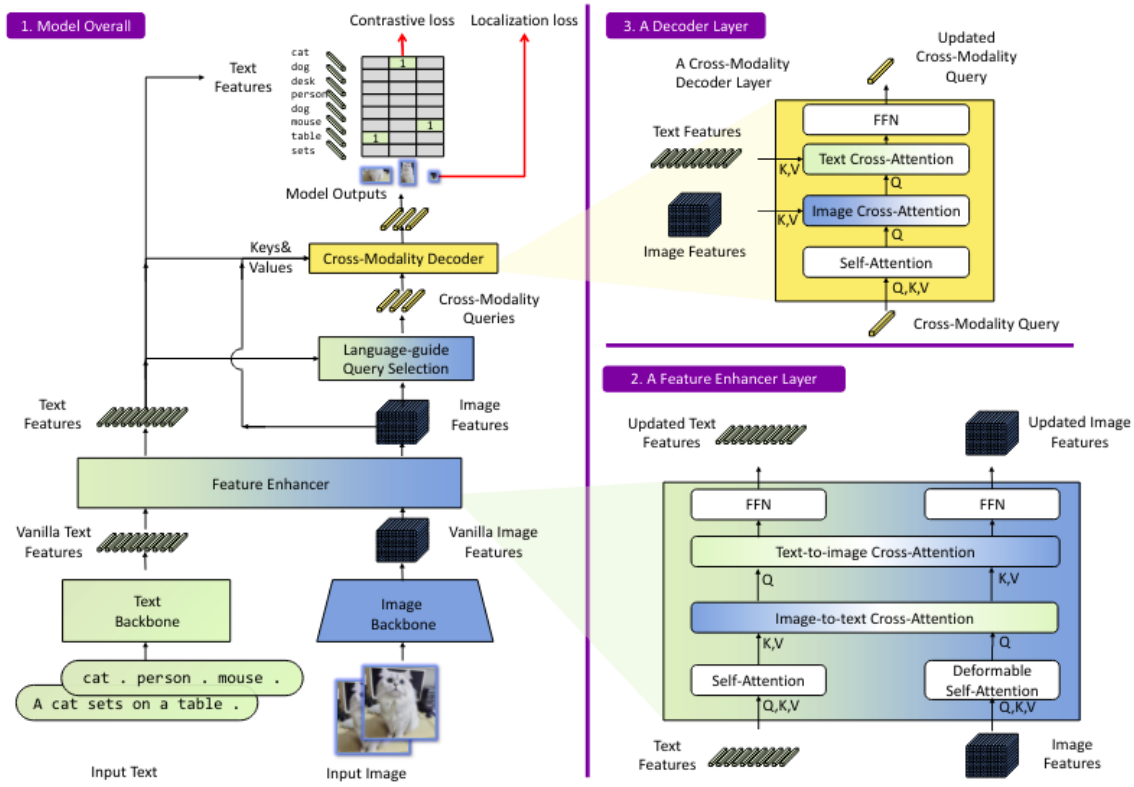


Figure 1.1: Overview of the model architecture. **(source: Liu et al. [2023])**

There exist two levels of text featuring. We can either encode the whole sentence as one feature or each word independently to represent category names for example. The problem with the second approach is that we may have unnecessary data dependency. To resolve this problem, Liu et al. [2023] have proposed a technique called *Sub-Sentence Level Text Feature* which uses attention masks to block some unwanted attentions.

Loss Function. Liu et al. [2023] have used $L1$ loss and GIOU loss for bounding box regressions. Contrastive Loss is used between text tokens and image bounding boxes for classification.

1.2.2 Segment Anything

Segment Anything is a concept designed by Kirillov et al. [2023] to segment any foreground or background in images based on text prompts inspired by large language models. It is built on three principal pillars: a promptable segmentation task that takes a prompt and outputs a set of segmentation masks; a model capable of supporting *flexible prompts*, *real-time* inference, and a*mbiguity-aware* segmentation; and training on a very large, high-quality dataset. Since classical segmentation datasets are not suitable for this type of model, the authors developed their own data engine to collect and annotate data. In this section, we discuss these points in detail.

- **Task**: the promptable segmentation task takes a prompt, which can be categorized into two sets: sparse prompts, such as points, boxes, or text, and dense prompts, or masks. It returns a set of valid masks regardless of the prompt type. Although this can lead to ambiguity, such as identifying "parts of a human," the task should output subparts, parts, or the whole object. The model is pre-trained using a *zero-shot*

transfer paradigm, meaning it does not require any examples to learn from for making inferences. In fact, for instance segmentation, we can combine promptable segmentation with existing object detection methods without requiring additional pre-training.

- **Model**: the *Segment Anything Model* (SAM) architecture is designed to perform promptable segmentation tasks with remarkable efficiency and flexibility, leveraging its three primary components: the image encoder, the prompt encoder, and the mask decoder. At the core of SAM, the image encoder processes the input image to create a high-dimensional embedding that encapsulates the critical features of the image, typically utilizing a Vision Transformer (ViT) pre-trained with Masked Autoencoders (MAE) to manage high-resolution images effectively. Complementing this, the prompt encoder handles various types of prompts, such as points, boxes, text, and masks, converting them into embeddings that can explicitly interact with the image embedding. This encoder employs distinct mechanisms for different prompts—points and boxes are encoded with positional encodings, text prompts use an off-the-shelf text encoder, and masks are processed through convolutions. The mask decoder then plays a crucial role in integrating these embeddings to produce segmentation masks. It achieves this by combining the image and prompt embeddings through a modified *Transformer decoder block*, which processes the combined information to output a set of valid masks. Designed for real-time inference, the mask decoder can generate multiple valid masks for a single prompt, effectively addressing ambiguities by providing several possible segmentation outcomes. The final output consists of these valid segmentation masks, each accompanied by a score indicating its confidence or quality, thus enabling the model to handle ambiguous prompts proficiently. The explicit integration of the image encoder, prompt encoder, and mask decoder in the SAM architecture ensures that the model delivers high-quality, promptable segmentation results across various types of input prompts, making it a versatile and powerful tool for image segmentation tasks. The figure 1.2 shows the SAM Architecture components. The subfigure on the left is an overview of the Segment Anything Model (SAM) architecture highlighting the three primary components: the image encoder, the prompt encoder, and the mask decoder. The image encoder processes the input image to create a high-dimensional embedding, while the prompt encoder handles various types of prompts such as points, boxes, text, and masks, converting them into embeddings. These embeddings are then integrated by the mask decoder, which uses a modified Transformer decoder block to process the combined information and output a set of valid masks. While the subfigure on the right, provides detailed view of the mask decoder in the SAM architecture. It consists of multiple attention layers: image-to-token attention, token-to-image attention, and self-attention layers, along with multi-layer perceptrons (MLP) and convolutional transformations. These layers process the image embedding and the prompt tokens to generate output tokens, which are then used to produce the final segmentation masks through a dot product operation. Additionally, the decoder computes Intersection over Union (IoU) scores for each mask to indicate its confidence or quality. This detailed mechanism ensures that the mask decoder can handle real-time inference and produce accurate segmentation results, even for ambiguous prompts.

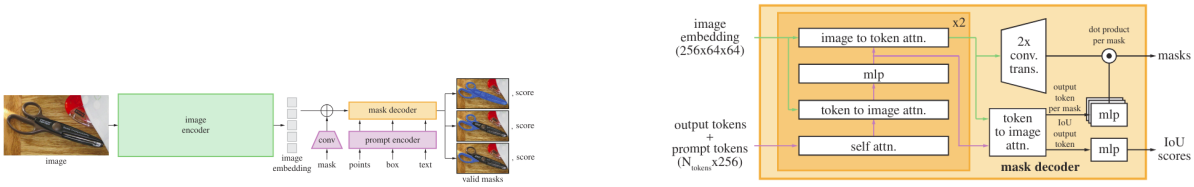


Figure 1.2: SAM Architecture components. **(source: Kirillov et al. [2023])**

Data. The authors developed a specialized data engine to collect data from various locations around the world, collaborating with photographers to obtain high-resolution images, which were then resized to 1024×1024 pixels. They gathered 11 million images, from which they extracted over 1.1 billion masks. Data annotation was performed in three stages:

1. *Assisted-manual*: This initial stage is like classical data annotation. It is crucial as the model is not yet capable of generating valid masks independently. At this stage, the model uses ViT-B and the number of masks generated per image increased from 20 to 44.

2. *Semi-automatic*: This stage involves a combination of automatically predicted masks and model-assisted annotation. The number of masks per image increased from 44 to 72, prompting modifications to the model architecture, including the adoption of ViT-H. The model was trained six times in total during this phase.

3. *Fully-automatic*: This stage became feasible because of two factors:

   – the model's improved ability to detect a sufficient number of masks, which progressively enhanced the model.

   – he incorporation of ambiguity-aware capabilities, allowing the model to identify valid masks even in ambiguous scenarios. A 32×32 regular grid of points was used to prompt each region of the image, along with additional small crops to focus on smaller objects. Confident masks were selected using IoU, and only stable masks were chosen based on their probability map threshold at $0.5 - \sigma$ and $0.5 + \sigma$, ensuring consistent mask production.

1.2.3 Scene Text Recognition with Vision Transformer

Like the progress in Natural Language Processing (NLP), transformers have proven to be a game-changer in the field of sequence modeling and prediction. This innovation leverages parallel self-attention and prediction, resulting in the development of fast and efficient models. While current transformer-based Scene Text Recognition (STR) models still involve a Backbone and a Transformer Encoder-Decoder, recent advancements, such as the Vision Transformer (ViT), have shown that it's possible to outperform deep networks like ResNet and EfficientNet for image classification on datasets like ImageNet1k only by relying on the transformer encoder but pre-trained on extensive datasets such as ImageNet21k and JFT-300M. The figure 1.3 shows the Network Architecture of ViTSTR:

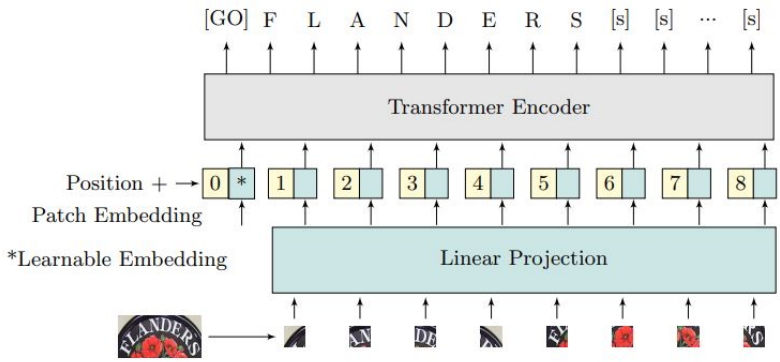


Figure 1.3: Network Architecture of ViTSTR **(source: Atienza [2021])**

In ViTSTR Atienza [2021], the output vector corresponds to the [GO] token. Furthermore, instead of just extracting one output vector, we extract multiple feature vectors from the encoder. The number is equal to the maximum length of text in our dataset plus two for the [GO] and [s] tokens.
Note that: The [GO] token is used to mark the beginning of the text prediction and [s] to indicate the end or a space. [s] is repeated at the end of each text prediction up to the maximum sequence length to mark that nothing follows after the text characters. An input image is first converted into patches. The patches are converted into 1D vector embeddings (flattened 2D patches). As input to the encoder, a learnable patch embedding is added together with a position encoding for each embedding. The network is trained end-to-end to predict a sequence of characters. [GO] is a pre-defined start of sequence symbol while [s] represents a space or end of a character sequence. The figure below 1.4 shows the High-Level Architecture of L-Layer Transformer Encoder Block:

Let

$$\mathbf{z}_0 = \left[\mathbf{x}_{\text{class}}; \mathbf{x}_p^1\mathbf{E}; \mathbf{x}_p^2\mathbf{E}; \ldots; \mathbf{x}_p^N\mathbf{E}\right] + \mathbf{E}_{\text{pos}}, \tag{1.2.1}$$

the patch embedding inputs where $\mathbf{E} \in \mathbb{R}^{P^2C\times D}$ and $\mathbf{E}_{\text{pos}} \in \mathbb{R}^{(N+1)\times D}$ which go through Layer Normalization (LN). Then, the Self-Attention layer (MSA) determines the relationships between feature vectors, the output of the MSA block is:

$$\mathbf{z}'_l = MSA\left(LN\left(\mathbf{z}_{l-1}\right)\right) + \mathbf{z}_{l-1}, \tag{1.2.2}$$

for $l = 1 \ldots L$.L is the depth or the number of encoder blocks. A transformer encoder is made of a stack of L encoder blocks. The output of MSA is concatenated with the previous output of LN, then the result passes through an LN again and Multi-Layer Perceptron (MLP) to perform feature extraction, the output of MLP is:

$$\mathbf{z}_l = MLP\left(LN\left(\mathbf{z}'_l\right)\right) + \mathbf{z}'_l, \tag{1.2.3}$$

for $l = 1 \ldots L$. Finally, the head is made of a sequence of linear projections forming the word prediction:

$$\mathbf{y}_i = \text{Linear}\left(\mathbf{z}_L^i\right), \tag{1.2.4}$$

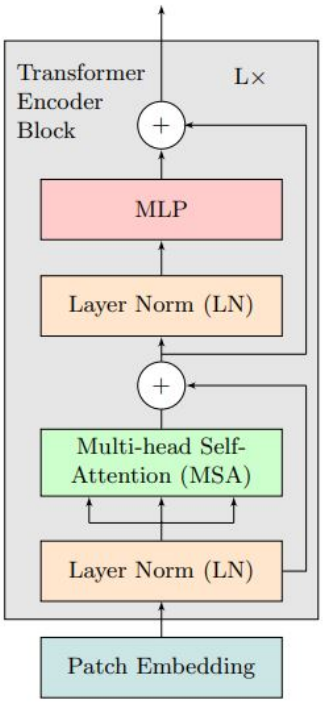


Figure 1.4: L-Layer Transformer Encoder Block Architecture **(source: Vaswani et al. [2017])**

1.2.4 End-to-end Text Spotting with DeepSolo

DeepSolo, introduced by Ye et al. [2023], is an end-to-end text spotting approach that integrates scene text detection and recognition into a unified framework. It uses a fully convolutional network to detect text regions and a character recognition network to recognize the text within those regions.

Since we are using transformers, we will first extract flattened multi-scale features from the image and for this purpose, we will use a backbone, we will then add positional embeddings to the result. After that, the transformer encoder will output embeddings. A lot of Bezier center curves are generated using four Bezier control points, then the top scored K-curves are selected to be sampled with N points in which coordinate points encode positional queries which will be added to learnable content queries to form composite queries which will be the input of the Transformer Decoder

Select the top-K Bezier Curve Proposals. Let

$\hat{p_i} = (\hat{p}_{ix}, \hat{p_i y}) \in [0,1]^2$ the 2D normalized coordinates of the curve i. $\delta p_{ix_0,y_0,\ldots,x_3,y_3}$ the predicted offsets by 3-layer MLP in which the last layer is 8 dimensions since we have 4 2D-points to predict their positions. The Bezier control points are $BP_i = \{\overline{bp}_{i0}, \overline{bp}_{i1}, \overline{bp}_{i2}, \overline{bp}_{i3}\}$ are predicted by

$$\overline{bp}_{i_j} = \left(\sigma\left(\Delta p_{ix_j} + \sigma^{-1}\left(\hat{p}_{ix}\right)\right), \sigma\left(\Delta p_{iy_j} + \sigma^{-1}\left(\hat{p}_{iy}\right)\right)\right). \tag{1.2.5}$$

This equation is used to score the curves. The reason the authors have incorporated the term $\sigma^{-1}\left(\hat{p}_{ix}\right)$ is that the sigmoid function, generally, takes an input not of probability nature. So to transform them from probabilities to real numbers, they have applied the **inverse of sigmoid**.

Point query modelling. We then have to select K-top curves according to previously calculated probabilities. The N points around the curves are uniformly generated using *Bernstein Polynomials Method*. While the normalized point coordinates are of the shape $K \times N \times 2$ in each image, the positional queries P_q are of the shape $K \times N \times 256$ which are generated by $P_q = MLP(PE(Coords))$. The composite queries $Q_q = P_q + C_q$ are formed by the positional queries P_q and content queries C_q which are generated using learnable embeddings. These last and content queries generated from the L-layer transformer encoder will be both fed into the decoder which will have to predict four tasks:

1. Instance classification
2. Character classification
3. Center curve points
4. Boundary points

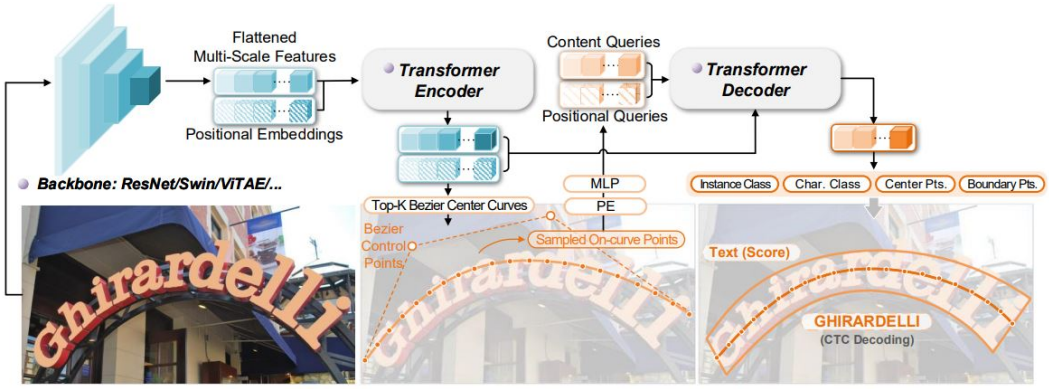


Figure 1.5: The architecture of DeepSolo **(source: Ye et al. [2023])**

Figure 1.5 presents DeepSolo architecture, let's explain how does this work:

1. Input image: DeepSolo takes an input image containing scene text as its input.
2. Text detection: The first step is to detect the text regions in the input image. DeepSolo uses a fully convolutional network (FCN) to predict a binary mask indicating the presence of text at each pixel location. This FCN is trained using a pixel-wise binary cross-entropy loss function.
3. Text recognition: Once the text regions have been detected, the next step is to recognize the text within those regions. DeepSolo uses a character recognition network (CRN) to perform this task. The CRN takes as input a cropped and resized version of each text region and outputs a sequence of character probabilities. The CRN is trained using a sequence cross-entropy loss function.
4. Joint training: The text detection and recognition networks are trained jointly using a multi-task loss function that balances the detection and recognition objectives. The multi-task loss function is a weighted sum of the pixel-wise binary cross-entropy loss and the sequence cross-entropy loss.
5. Routing mechanism: DeepSolo incorporates a novel routing mechanism that enables the recognition network to attend to different parts of the input feature map. This routing mechanism allows the recognition network to handle text of varying lengths and aspect ratios.

6. Output: The final output of DeepSolo is a set of bounding boxes and corresponding text labels for each detected text region in the input image.

1.2.5 End-to-end Scene Text Spotting with ESTextSpotter: Explicit Synergy in Transformer

Here, we will present an Explicit synergy-based Text Spotting Transformer framework, termed ESTextSpotter Huang[1*] et al. [2023], figure 1.6 illustrates the framework of the proposed ESTextSpotter.

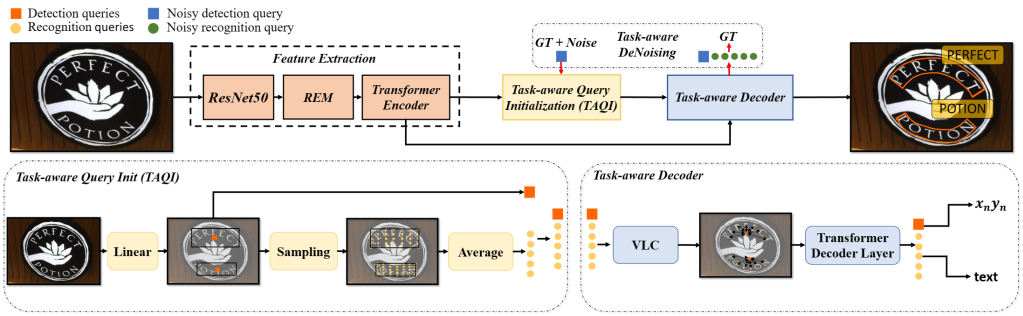


Figure 1.6: The framework of the proposed ESTextSpotter. **(source: Huang[1*] et al. [2023])**

Let's explain the architecture of this solution. First of all, the image will pass through a feature extraction process which consists of ResNet50 as the backbone, receptive enhancement module (REM), and Transformer encoder. The output of Resnet50 (feature map res_5) is sent to the REM which downsamples it, then the output of this REM is fed into the Transformer Encoder to extract important features, and then the result is sent to the Task-aware Query Initialization ($TAQI$). Now, we will have to generate detection and recognition queries in this phase, so, starting from generating text classification scores using the linear layer and then choosing the top N-scored features, these features are sent into a linear layer to initialize proposals for detection queries. For recognition queries, we sample the features es $\mathbf{F_p} \in \mathbb{R}^{N\times H\times T\times C}$ from the proposals and average over the height dimension to initialize recognition queries. Here we have: N represents the maximum number of predictions identified in DETR Carion et al. [2020], while T represents the length of recognition queries, and C represents the feature dimension. Finally $TAQI$ encodes detection and recognition queries,$\mathbf{G} \in \mathbb{R}^{N\times C}$ and $\mathbf{R} \in \mathbb{R}^{N\times T\times C}$ respectively. The next step is to send the task-aware queries $\mathbf{S} \in \mathbb{R}^{N\times(T+1)\times C}$ (including both detection and recognition queries) to the task-aware decoder which enhances the explicit synergy between task-aware queries using a vision language communication. The Language Conversion extracts semantic features in recognition queries and then maps them into language vectors $\mathbf{L} \in \mathbb{L}^{N\times(T+1)\times C}$, here is the definition:

$$\mathbf{L} = \text{cat}\left(\mathbf{G}, \mathbf{W_2 P}\right), \tag{1.2.6}$$

where $\mathbf{W}_1 \in \mathbb{R}^{C\times U}$ and $\mathbf{W}_2 \in \mathbb{R}^{U\times C}$ represent trainable weights.U indicates the character class number. cat is the concatenation operation. Then the task-aware queries $\mathbf{S}$

and language vectors $\mathbf{L}$ are sent to a vision-language attention module, which is formalized as:

$$\begin{aligned} \mathbf{M}_{ij} &= \begin{cases} 0, & i \neq j, \\ -\infty, & i = j. \end{cases} \\ \mathbf{F} &= \text{softmax}\left(\frac{(\mathbf{S}+\text{PE}(\mathbf{S}))(\mathbf{L}+\text{PE}(\mathbf{L}))^T}{\sqrt{D}} + \mathbf{M}\right)\mathbf{L}. \end{aligned} \tag{1.2.7}$$

PE indicates the position encoding used by DETR Carion et al. [2020]. The attention mask $\mathbf{M}$ is designed to prevent the queries from over-focusing "itself".

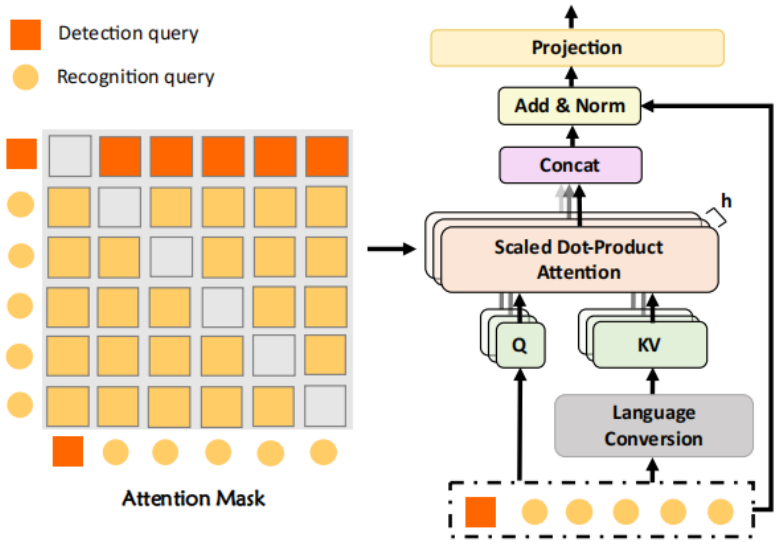


Figure 1.7: Vision language communication module **(source: Huang[1*] et al. [2023])**

Explication: The language conversion extracts the semantic features in recognition queries. Then the visual and semantic information can interact in a cross-modal perspective. h is the number of parallel attention heads.
After the exchange of vision language information, the task-aware queries are fed into the Transformer Decoder to extract the position and shape information of text instances from the detection queries and understand the semantic and positional information of characters from the recognition queries, here we can say that both detection and recognition queries are helping each other to achieve the best result.
The approach to generate detection and recognition results. The detection queries are sent into two-feed forward layers, one predicts the proposals: (x, y, h, w), while the other predicts Z polygon offsets $(\Delta x_1, \Delta y_1, \Delta x_2, \Delta y_2, \ldots, \Delta x_Z, \Delta y_Z)$ based on the center point (x, y) of each proposal. Z is 16 . Reconstructing the polygon can be formulated as:

$$\begin{aligned} x_i &= x + \Delta x_i, \\ y_i &= y + \Delta y_i. \end{aligned} \tag{1.2.8}$$

This method allows us to predict the detection outcome using detection queries without needing serial control points or freezing the model weights for training a segmentation head. Once the decoding process is completed, the recognition queries can efficiently gather

character features. To achieve this, a linear layer is used to transform the recognition queries into characters, this article Zhang et al. [2022] explains the method.

1.2.6 Learning Transferable Visual Models From Natural Language Supervision

Traditional computer vision models require large, labeled datasets for training which are time-consuming and expensive to create, in addition, they limit the models to a fixed set of object categories. These models are often struggling to generalize to new, unseen categories without additional labeled data. CLIP leverages a different approach by learning from a large dataset of 400 million (image, text) pairs collected from the internet. This method allows the model to understand and reference visual concepts through natural language by using visual concepts described in everyday language, enabling zero-shot learning [1] to handle new tasks without needing any additional training specific to those tasks.

The CLIP approach as it's presented in 1.8 uses the names of all classes in a dataset as text pairings and predicts the most probable (image, text) pair, by computing the feature embeddings of images and texts using their respective encoders and calculating the cosine similarity between these embeddings. The similarities are then scaled by a temperature parameter and normalized into z probability distribution with a softmax function. This approach serves as the backbone for visual feature extraction while the text encoder generates a classifier based on the text descriptions of visual concepts.

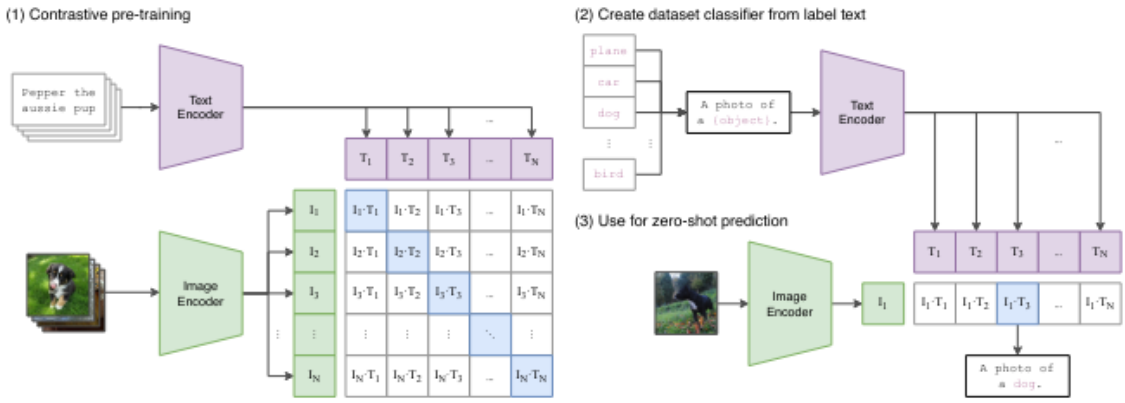


Figure 1.8: Summary of CLIP approach **(source: Zhao et al. [2023b])**

Unlike standard image models that train an image feature extractor and a linear classifier together to predict labels, CLIP jointly trains an image encoder and a text encoder to identify the correct pairings (image, text). During testing, text encoder synthesizes a zero-shot linear classifier by embedding the names or descriptions of the classes in the target dataset.

1. **Contrastive Pre-Training** The model learns to associate images with their descriptive texts.

[1] Zero-shot learning (ZSL) is a machine learning technique where a model can recognize and understand new, unseen tasks or categories without having been explicitly trained on them beforehand. It does this by leveraging knowledge from related tasks and generalizing from existing data.

(a) **Input Data**: An image and its corresponding descriptive text are fed into the model.

(b) **Text Encoder**: The descriptive text is processed by the text encoder to produce a text embedding T_i.

(c) **Image Encoder**: The image of the dog is processed by the image encoder to produce an image embedding I_i.

(d) **Contrastive Learning**: The model learns by maximizing the similarity (using cosine similarity) between the matching (image, text) embeddings (e.g., I_i and T_i) and minimizing the similarity between non-matching pairs (e.g., I_i and T_j for $j \neq i$).

2. **Create Dataset Classifier from Label Text** Class text descriptions are embedded to form a classifier

(a) **Label Text**: The names or descriptions of the classes (e.g., "plane", "cat", "dog", "bird") are defined in a template format (e.g., "A photo of a {object}").

(b) **Text Encoder**: Each class description is processed by the text encoder to produce a set of text embeddings $T_1, T_2, \ldots, T_N$ corresponding to each class.

3. **Use for Zero-Shot Prediction** The model predicts the class of a new image by comparing its embedding with the class text embeddings.

(a) **Image Encoding**: A new image (e.g., a photo of a dog) is processed by the image encoder to produce an image embedding I_j.

(b) **Similarity Calculation**: The image embedding I_j is compared to all the class text embeddings $T_1, T_2, \ldots, T_N$ using cosine similarity.

(c) **Prediction**: The class with the highest similarity score (e.g., $I_j \cdot T_{dog}$) is selected as the predicted label for the image.

Experiments and Results

CLIP conducted many experiments on different datasets including the evaluation of its zero-shot classification performance across various datasets and tasks, the most notable comparison is with Visual N-Grams, where CLIP improved the accuracy on ImageNet from 11.5% to 76.2%, matching the performance of fully supervised ResNet-50 without using labeled training examples. It was also tested on 27 different datasets, outperforming the fully supervised logistic regression classifier trained on ResNet-50 features in many other cases.

The competitive results of CLIP are shown with fully supervised models in a zero-shot setting. In terms of few-shot learning [2], CLIP's zero-shot classifier matched the performance of a 4-shot linear classifier and nearly matched the best results of a 16-shot

[2]Few-Shot Learning (FSL) is a Machine Learning framework that enables a pre-trained model to generalize over new categories of data (that the pre-trained model has not seen during training) using only a few labeled samples per class.

linear classifier across various models. These findings suggest that CLIP's approach of using natural language to specify visual concepts can effectively compete with traditional supervised and few-shot learning methods, particularly in scenarios where labeled data is scarce.

Conclusion

In this chapter, we presented the state-of-the-art (SOTA) methods in object detection, segmentation, text recognition, and text spotting. We began by showcasing the GroundingDINO model, a top object detector capable of identifying new objects not included in the training data or predefined list. Following this, SAM, a highly effective segmentation model, can accurately segment objects in images using various techniques (such as text prompts or training on extensive high-quality datasets). Next, we delved into the realm of vision transformers, which have become very popular nowadays, with the ViTSTR model, recognizing the text in images using only one transformer encoder. Also, we have presented two end-to-end scene text spotting methods that are DeepSolo and ESTextSpotter. both of them are based on transformers architecture. Finally, we have presented the CLIP model, emphasizing the importance of integrating transformers and large language models like CLIP.

By exploring fundamental concepts and advanced strategies, we have demonstrated the impact of these technologies on various fields, showcasing the progress and potential of combining visual and textual data to enhance performance and introduce new capabilities in computer vision tasks.

Chapter 2

Evaluation of Current Approaches and Their Limitations

Introduction

The aim of this chapter is to show the outcomes of using the CLIP method before and after fine-tuning it with the SVT (Street View Text) Dataset. We will begin by giving a statistical overview of the dataset, made up of pairs of images and words that express meanings. Next, we will discuss the resources utilized for training and testing our models on this dataset. The model creates a probability density distribution, requiring metrics like accuracy, precision, recall, and F1-Score. Additionally, we assume that all correct words have an equal chance of being detected, shown as a uniform distribution. This assumption justifies the use of an entropy measure to assess the uniformity of word detection probabilities, with the logarithmic base matching the number of accurately predicted words.

The CLIP model, explained in the previous Chapter 1, is built using the Vision Transformer (ViT) architecture by Fu [2022], Ruan et al. [2022]. It considers images as sequences of tokens processed by transformer layers. In our study, we work with the ViT-B/32 version, which is defined by specific parameters like embedding dimension, image resolution, and the number of vision layers. The model's accuracy in generating text predictions is vital for real-world use. Our primary goal is to assess how effective fine-tuning the CLIP model on the SVT dataset can be. We want to find the best setup that improves precision while delivering consistent performance across different measures. This involves:

- **Qualitative Analysis:** Visually inspecting top predicted words for selected images.
- **Quantitative Analysis:** Conducting a comprehensive evaluation using adjusted metrics and comparing performance across different checkpoints.

By integrating these components, this chapter endeavors to offer a comprehensive understanding of the CLIP model's capabilities and its relevance to the SVT dataset. This endeavor lays the groundwork for future exploration and advancement in image and text similarity tasks.

2.1 CLIP's Application to SVT Data

In this section, we will discuss how we applied the CLIP method to the SVT dataset, starting with a description of the dataset itself, followed by a detailed explanation of our methodology.

2.1.1 Exploratory Data Analysis

In this section, we present the results of the exploratory data analysis (EDA) conducted on the Street View Text (SVT) dataset. This analysis aims to provide insights into the distribution and frequency of words in the dataset, as well as other relevant characteristics of the data.

Data Description

The Street View Text (SVT) dataset consists of 647 word patches cropped from Google Street View.Each image is associated with a 50-word lexicon. These images contain a variety of text instances with different fonts, sizes, colors and orientations, providing a robust testbed for test detection and recognition methods. Image text in this data often comes from business signage and business names are easily available through geographic business searches. These factors make the SVT suited for word spotting in the wild where the goal is to identify words from a given street-view image.

So here, the dataset that we have contains:

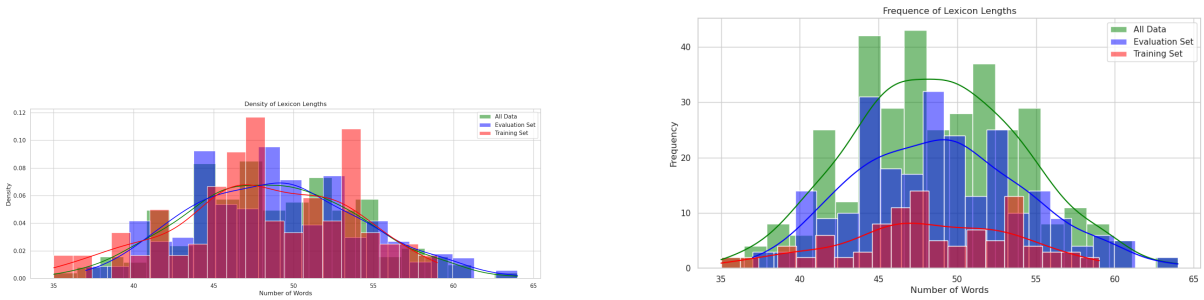
- **Training set:** A subset of 100 images linked with their lexicons
- **Evaluation set:** A subset of the SVT dataset contains 249 images paired with the adequate text

Lexicon Overlap

We also examined the overlap of words between the training and evaluation sets. The training set contains **2545 train unique words**, while the evaluation set contains **4282 eval unique words**. Notably, there are **1464 common words** to both sets, resulting in a total vocabulary of **5363 all unique words** across the entire dataset. This overlap indicates a degree of similarity in the lexicons used in both subsets, which could be advantageous for training and evaluating word recognition models.

Lexicon Lengths

In this analysis, we examined both the density and the frequency of the number of words in each lexicon. Each image in the dataset is associated with a unique set of words, and we aimed to understand how the number of words varies and how frequently these different numbers of words counts appear across both the training and evaluation sets. As illustrated in Figure 2.1a, the lexicons show a significant variation in length, with a distribution pattern that appears to follow a normal curve. Additionally, we looked at the frequency of these word counts to identify any common patterns or trends in the dataset as it is shown in 2.1b.



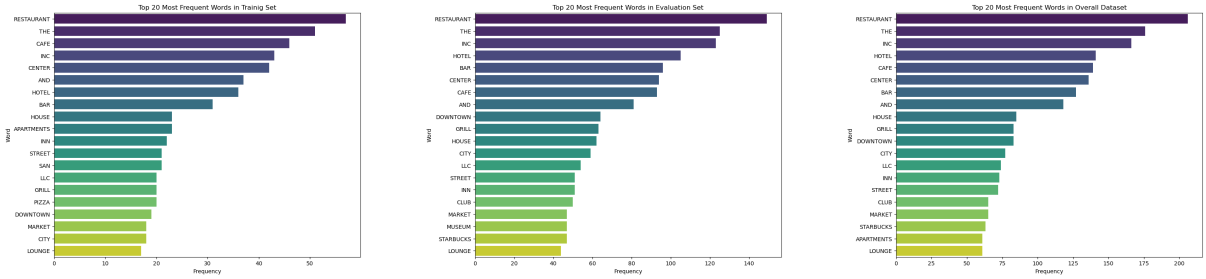
(a) Density of Lexicon Lengths

(b) Frequency of Lexicon Lengths

Figure 2.1: Analysis of Lexicon Lengths: Density and Frequency in Training, Evaluation, and Overall Dataset **(source: compiled by us)**

Word Frequency Analysis

We performed a frequency analysis to identify the most common words in the dataset. Figures 2.2a and 2.2b show the top 20 most common words in the training and evaluation sets, respectively.



(a) Top 20 Most Common Words in Training Set

(b) Top 20 Most Common Words in Evaluation Set

(c) Top 20 Most Common Words in Overall Dataset

Figure 2.2: Comparison of Most Common Words in Training, Evaluation, and Overall Datasets **(source: compiled by us)**

From these results, we have obtained "RESTAURANT", "THE", "INC", "HOTEL", "CAFE", "CENTER", "BAR", and "AND" as the most frequent words in both the training and evaluation sets. Consequently, they also occupy the top positions in the overall dataset, being among the top 8 most common words. This suggests a significant consistency in word frequency across the different datasets.

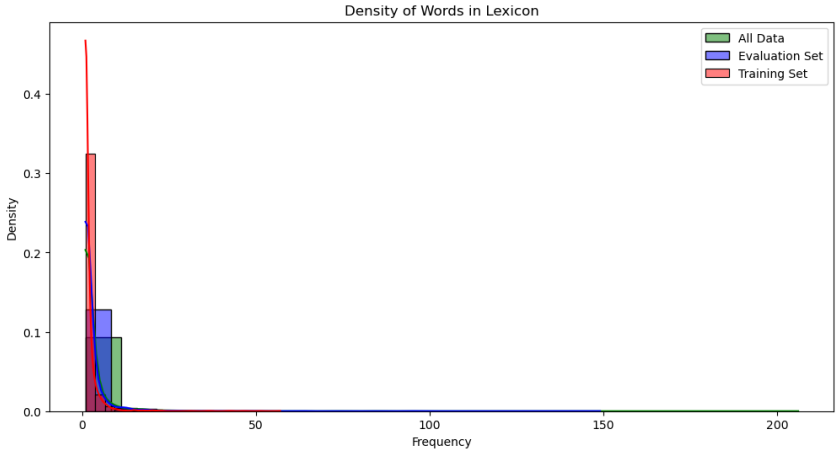


Figure 2.3: Density of word frequencies **(source: compiled by us)**

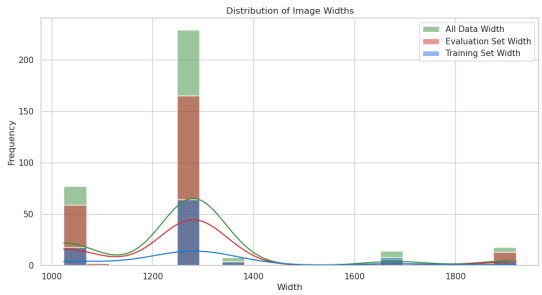
This figure (2.3) shows that the frequency of words in our lexicon varies significantly. Over 5,000 words share the same frequency throughout the dataset, indicating that our data is unbalanced. Specifically, we found 5,265 words with frequencies ranging from 0 to 20. This large difference in frequency distribution explains the uneven density of word frequencies. Consequently, only a few words have frequencies between 20 and 206. This pattern is consistent across both the training and evaluation datasets. More details are shown in the tale below 2.1.

Statistic	**Value**
Count	5363.000
Mean	3.166
Standard Deviation	8.098
Minimum	1.000
25th Percentile	1.000
50th Percentile (Median)	1.000
75th Percentile	3.000
Maximum	206.000

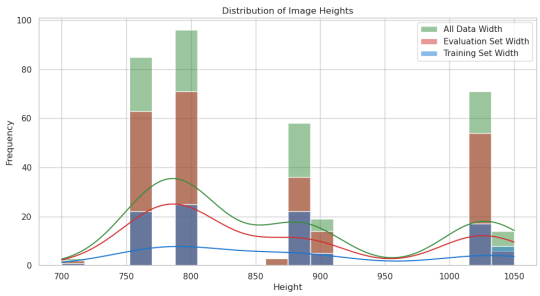
Table 2.1: Summary Statistics about words frequency distribution

Image Analysis

In addition to the text data, we analyzed the image dimensions in the dataset. Figures 2.4a and 2.4b show the distribution of image widths and heights, respectively. These distributions help us understand the variability in image sizes within the dataset.



(a) Distribution of Image Widths



(b) Distribution of Image Heights

Figure 2.4: Distribution of Image Dimensions in the Dataset **(source: compiled by us)**

As we can see from the analysis, the image dimensions do not follow a normal distribution. There is considerable variation in both image widths and heights across the dataset. To handle this, CLIP model uses preprocessing steps to standarize the image dimensions and to ensure that all images are adjusted to a uniform size or resolution. Upon reviewing the preprocessing code, it was confirmed that the resizing step is applied to all images after loading, the $image\_size = (224, 224)$.

2.2 Experimental Setup

Integrating CLIP with the SVT dataset involved fine-tuning the model to handle the unique challenges presented by the dataset. Still, before this, we have started by testing the dataset with CLIP model without fine-tuning to see the results it would give us and compare these results to those we would get after the fine-tuning. This section outlines the steps taken in to prepare out experimental setup, and the system resources.

2.2.1 Word-Image Association Pipeline

Our main objective was to create our vocabulary, the list of unique words appearing overall the dataset, then create a matrix where each row represents an image while each column represents a word from our vocabulary. Inside this matrix, we fill in the probabilities showing how each word is connected in meaning to the image it corresponds to.

To get started, we built a list of words that represent our vocabulary by carefully going through our data. Let Once this list was ready, we moved on to the testing phase. Here, we fed in the images and the list of words. The result of this process was the creation of a matrix that neatly shows how each word in our vocabulary is related in meaning to each image.

The table 2.2 explains our mathematical formulation for the image-text association pipeline that we have used: Let $I \in \mathbb{R}^{m \times n_w \times n_h \times n_c}$ where m denotes the number of examples and $n_w \times n_h \times n_c$ is the shape of each image.

W be the set of words in the vocabulary, and we can represent the number of words in the vocabulary as $|W|$. For each image i, let W_i^+ denote the set of existing words (lexicons) already present in our dataset for that image, and let $W_i^- = W - W_i^+$ represent the set of words that do not exist in this image. Here, "existing" does not imply that the words appear in the image but rather that they have a semantic relationship with the image.

Annotation	Explanation
$I \in \mathbb{R}^{m \times n_w \times n_h \times n_c}$	Tensor representing m examples of images with dimensions $n_w \times n_h \times n_c$.
W	Set of words in the vocabulary.
$\|W\|$	Total number of words in the vocabulary.
W_i^+	Set of existing words in the dataset for image i.
W_i^-	Set of words not existing in image i.
P	Matrix representing probabilities of word-image semantic associations.
$\mathbb{P}(j \in W^+ \| I^{(i)}) = p_{i,j}^+ = \frac{1}{\|W^+\|}$	Probability of word j existing in image i (uniform distribution $\sim \mathcal{U}(0,1)$).
$\mathbb{P}(j \in W^- \| I^{(i)}) = p_{i,j}^- = 0$	Probability of non-existing word j in image i.
$\hat{\mathbb{P}}(j \in W^+ \| I^{(i)}) = \hat{p}_{i,j}^+$	True positive probability for word j in image i.
$\hat{\mathbb{P}}(j \in W^- \| I^{(i)}) = \hat{p}_{i,j}^-$	False negative probability for word j in image i.
$\sum_{j=1}^{\|W^+\|} \hat{p}_{i,j}^+ + \sum_{j=1}^{\|W^-\|} \hat{p}_{i,j}^- = 1$	Sum of probabilities per prediction is equal to 1.

Table 2.2: Mathematical Annotations and Explanations

Let P be the matrix representing the probabilities of each word's semantic association with each image. **Initially**, we assume that our existing words follow a uniform distribution, meaning they are all detected with equal probability. Let $\mathbb{P}(j \in W^+ | I^{(i)}) = p_{i,j}^+ = \frac{1}{|W^+|}$ where $i \in I, j \in W^+$ represent this probability, where $|W^+|$ represents the number of existing words. Since non-existing words are not detected, $|W^-| = 0$, and their probability $\mathbb{P}(j \in W^- | I^{(i)}) = p_{i,j}^- = 0$. **During prediction**, we have created an other annotation, where we denote the true positives as $\hat{\mathbb{P}}(j \in W^+ | I^{(i)}) = p_{i,j}^+$ and the false negatives as bellow: $\hat{\mathbb{P}}(j \in W^- | I^{(i)}) = p_{i,j}^-$. Note that the sum of predicted probabilities is equal to 1 since the CLIP model uses the softmax function to compute the probability over the set of words for each image.

The matrix P can be represented as:

$$P = \begin{bmatrix} p_{1,1} & p_{1,2} & \cdots & p_{1,|W|} \\ p_{2,1} & p_{2,2} & \cdots & p_{2,|W|} \\ \vdots & \vdots & \ddots & \vdots \\ p_{|I|,1} & p_{|I|,2} & \cdots & p_{|I|,|W|} \end{bmatrix}$$

p_{ij} = Probability of word j being semantically related to image i

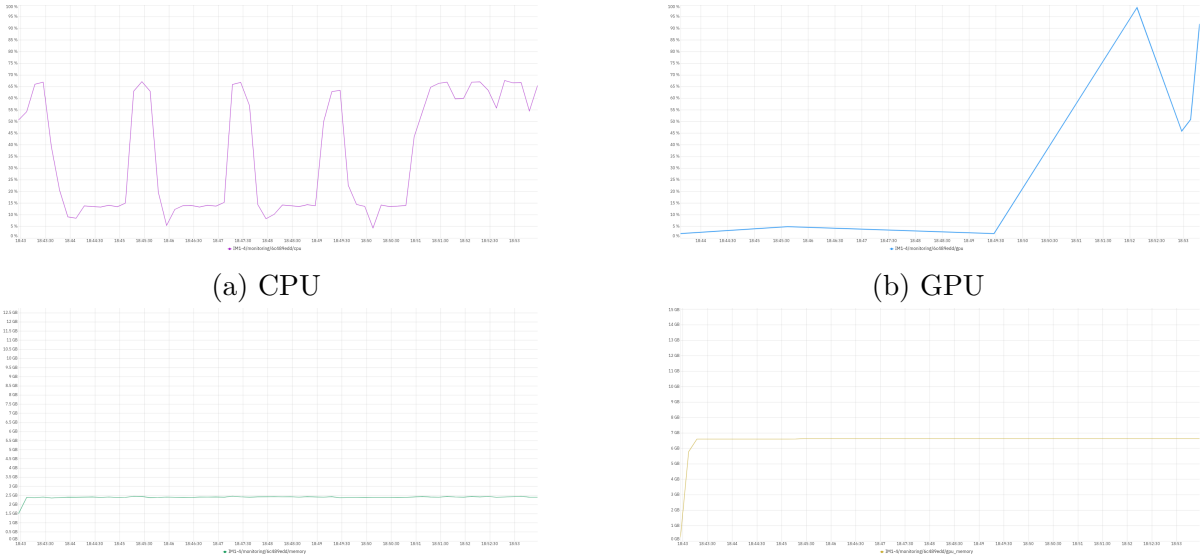
2.2.2 System Resources

We conducted our studies on Colab's 15GB *T4 GPU* which is mainly used by **PyTorch**[1] to train the model. Additionally, it is equipped with *Intel Xeon CPU with 2 vCPUs (virtual CPUs)* and 13GB of RAM to perform intermediate processings and 78.2GB of storage which we used to clone our project to it and the relative data described in 2.1.1.

[1]Machine Learning library developped by Paszke et al. [2019], that demonstrates it's possible to have both ease of use and high performance. It uses a straightforward, Python-friendly style that makes coding and debugging simple. PyTorch works well with other scientific computing libraries and efficiently supports hardware accelerators like GPUs.

For the evaluation, for each image, $n = 4282$ text logits are computed with softmax, this makes the computations harder on the CPU since it communicates with RAM of only 12.7GB. That is why we used the GPU also for the evaluation. The Figure 2.5 shows the resource utilization such as CPU, GPU, memory and GPU memory, during the training time.

- **CPU**: The CPU utilization shows cyclic patterns, fluctuating between 65-70% and nearly 5%. This pattern likely corresponds to the periodic handling of specific tasks such as data loading from the disk drive and pre-processing before feeding the data to the model. The end of the training phase requires more computations, which is reflected in increased CPU usage due to extra tasks like model saving.
- **GPU**: GPU utilization starts very low, around 0-5%, and then suddenly spikes to nearly 100% towards the end of the monitoring period. While the exact reason for this spike is unclear, it might be due to the high computational energy required as the parameters converge to a local minimum.
- **Memory**: Memory usage remains stable at around 2.5GB during training, indicating a constant quantity of loaded data at each step.
- **GPU Memory**: GPU memory usage remains constant during training as it stores both the model and the data required for processing. The size of the loaded data to the GPU Memory is also constant.



(a) CPU

(b) GPU

(c) Memory

(d) GPU Memory

Figure 2.5: Resource Utilization during the Training Time **(source: compiled by us)**

2.2.3 Tools and Technologies

Several tools were used in our project for simulation. These tools include programming languages, libraries, frameworks, and environments supporting different implementation parts.

Python is an interpreted, multi-paradigm, and cross-platform programming language. It supports imperative structured, functional, and object-oriented programming.

PyTorch is an open-source machine learning library for Python, developed by Meta, based on Torch. PyTorch allows for the necessary tensor computations, particularly for deep learning.

Pandas is a library written for the Python programming language, enabling data manipulation and analysis. It offers data structures and operations for manipulating numerical tables and time series.

Google Colab is a free, cloud-based Jupyter notebook environment that supports free GPU and TPU acceleration for machine learning models.

Streamlit is an open-source app framework for Machine Learning and Data Science teams. It allows for the quick creation of custom web applications for data exploration and model deployment using simple Python scripts.

Neptune.ai is a metadata store for MLOps, built for teams that run a lot of experiments. It allows users to log, store, display, organize, compare, and query all their MLOps metadata.

2.3 Performance Metrics

2.3.1 Accuracy, Precision, Recall and F1-score

The performance metrics that we have used to evaluate the first results are defined as follows:

$$\text{Accuracy} = \frac{\text{TP} + \text{TN}}{\text{TP} + \text{TN} + \text{FP} + \text{FN}} \tag{2.3.1}$$

$$\text{Precision} = \frac{\text{TP}}{\text{TP} + \text{FP}} \tag{2.3.2}$$

$$\text{Recall} = \frac{\text{TP}}{\text{TP} + \text{FN}} \tag{2.3.3}$$

$$\text{F1-score} = \frac{2 \times \text{Precision} \times \text{Recall}}{\text{Precision} + \text{Recall}} \tag{2.3.4}$$

Where:

- TP = True Positives
- TN = True Negatives
- FP = False Positives

- FN = False Negatives

Before using these metrics, we considered adjusting **TP, TN, FP** and **FN**. Since our dataset does not align accurately with conventional methods, we assumed a default value of 1 for existing words and 0 for non-existing words. Two approaches were used to evaluate these metrics:

1. **First Approach**
   We have established a method for interpreting the predictions to determine if a word exists based on its probability. To accomplish this, we must define a threshold. The challenge lies in determining an appropriate threshold value. To address this challenge, we devised a formulation:

$$\text{threshold}_{(i,j)} = \frac{1}{|W_i^+|} - \epsilon_j \tag{2.3.5}$$

   where $|W_i^+|$ represents the number of positive instances for image i and ϵ_j is a small adjustment factor for word j. The intuition behind using this method is that initially, we assumed that the actual data has a uniform distribution. However, we recognize that in real-world testing, this is rarely the case or highly improbable. Therefore, we added a margin of error, ϵ_j, to account for this discrepancy.

   We computed the optimal thresholds for each label (word) by aligning the predicted scores with the true labels, computing the Receiver Operating Characteristic (ROC) curve, and finding the threshold that maximizes Youden's J statistic. The process involves:

   - Extracting true labels y_{true} and predicted scores y_{scores} for each label.
   - Computing the ROC curve, which plots the True Positive Rate (TPR) against the False Positive Rate (FPR) at various threshold settings. The TPR and FPR are defined as follows:

$$\text{TPR}_j = \frac{\text{TP}_j}{\text{TP}_j + \text{FN}_j} \tag{2.3.6}$$

$$\text{FPR}_j = \frac{\text{FP}_j}{\text{FP}_j + \text{TN}_j} \tag{2.3.7}$$

   - Calculating Youden's J statistic, defined as:

$$J_j = \text{TPR}_j - \text{FPR}_j \tag{2.3.8}$$

   - Identifying the optimal threshold θ_j^* that maximizes J_j:

$$\theta_j^* = \arg\max_{\theta} \left(\text{TPR}_j(\theta) - \text{FPR}_j(\theta)\right) \tag{2.3.9}$$

     Here, θ_j^* represents our ϵ_j.

2. **Second Approach** Our approach considers the existence of text detection models. With some redefinitions, we will use the same notation as defined in table 2.2 for images, texts, and their probabilities.

There exist $|W| = 5363$ words in the vocabulary. CLIP measures the similarity between each word and the given image and returns a normalized exponential function which represents a density distribution function given by $\text{softmax}(x_i) = \frac{e^{x_i}}{\sum_{j=1}^{n} e^{x_j}}$. We consider the following notation to use in our modified metrics:

$$\mathbf{TP}_i = \sum_{j=1}^{|W_i^+|} \frac{\hat{p}_{i,j}^+}{\max(\hat{p}_{i,j}^+)} \tag{2.3.10}$$

$$\mathbf{TN}_i = \sum_{j=1}^{|W_i^-|} (1 - \frac{\hat{p}_{i,j}^-}{\max(\hat{p}_{i,j}^-)}) \tag{2.3.11}$$

$$\mathbf{FP}_i = \sum_{j=1}^{|W_i^-|} \frac{\hat{p}_{i,j}^-}{\max(\hat{p}_{i,j}^-)} \tag{2.3.12}$$

$$\mathbf{FN}_i = \sum_{j=1}^{|W_i^+|} (1 - \frac{\hat{p}_{i,j}^+}{\max(\hat{p}_{i,j}^+)}) \tag{2.3.13}$$

The intuition behind using this formula for $\mathbf{TN}_i$ and $\mathbf{FN}_i$ is based on the assumption that the probability of prediction for non-existing words is zero. In other words, we sum only the instances where the probability of prediction is zero, as we are interested in counting the true negatives or false negatives. For example, if we have three true negatives, this means that there are three instances where the model correctly predicted the absence of certain words (non-existing words), with the corresponding probabilities being zero.

2.3.2 Normalized Similarity-Entropy Measure

We propose *Normalized Similarity-Entropy Measure* to evaluate our models. The main idea is to evaluate how our probability distribution deviates from the true distribution. Given that the real distribution is uniform, entropy is a suitable metric because it increases as the distribution becomes more uniform.

The entropy loss for a sample is defined as:

$$\text{Entropy}_i = -\sum_{i=1}^{n} p_i \log_n(p_i) \tag{2.3.14}$$

To ensure we have a proper probability distribution, we normalize this by using a logarithm with an appropriate base, given that $p_i \sim \mathcal{U}(0, 1)$.

Figure 2.6 illustrates the entropy loss function with a base of 2. In our scenario, we are dealing with one distribution but two classes. Therefore, we aim to create two density distributions, focusing on the one that interests us at the moment.

This approach allows us to measure how well our predicted distribution aligns with the actual uniform distribution, ensuring that our model's outputs are as informative and accurate as possible.

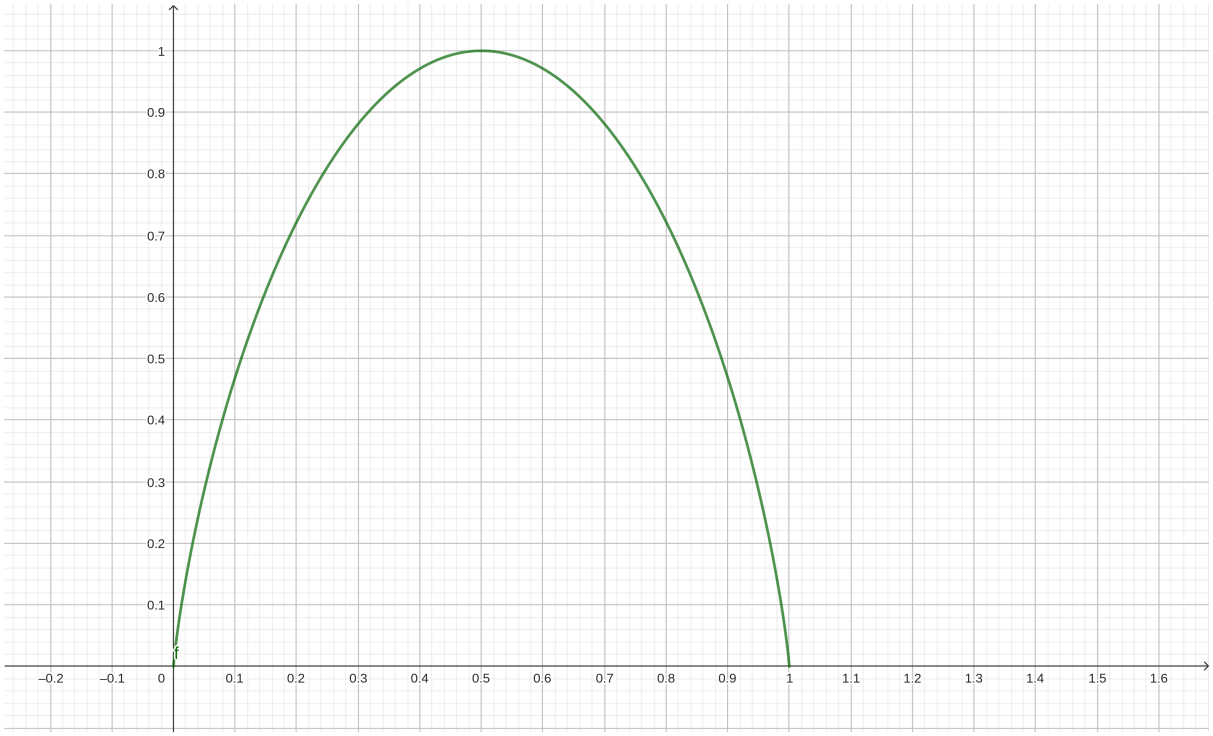


Figure 2.6: Entropy Function **(source: compiled by us)**

Let $\hat{p}_{i,j}^{norm+} = \frac{\hat{p}_{i,j}^{+}}{\sum_{i=1}^{|W_i^+|} \hat{p}_{i,j}^{+}}$ be the normalized probabilities. The reason for doing so, is to isolate the effect of the occurrence frequency from the evaluation. This gives us the following equation:

$$\text{Entropy}_i = -\sum_{j=1}^{|W_i^+|} \hat{p}_{i,j}^{norm+} \log_{|W_i^+|} \left(\hat{p}_{i,j}^{norm+}\right) \tag{2.3.15}$$

2.4 Results

2.4.1 Without Fine-Tuning

Checkpoint

The CLIP model architecture with which we tested the SVT dataset is represented in Table 2.3. The checkpoint used is ViT-B/32 which refers to a specific variant of the Vision Transformer (ViT) model. It is broken down into parts:

1. **ViT**: This stands for Vision Transformer. It's a type of neural network architecture specifically designed for processing images. Unlike traditional convolutional neural networks (CNNs), which operate on image patches, ViT treats images as sequences of tokens and processes them using transformer layers.

2. **B**: The letter "B" represents the model size or capacity. In the ViT architecture, model variants are denoted by letters such as "B", "L", "H", etc. Each letter typically corresponds to a specific combination of model width, depth, and other architectural parameters.

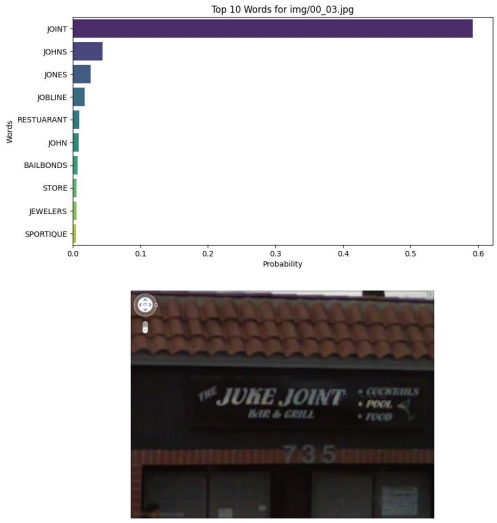
3. **32**: This number indicates the size of the patches used to divide the input images. In this case, "32" means that the input image is divided into patches of size 32x32 pixels, as explained in Table 2.3.

Table 2.3: Default parameters for the ViT-B/32 model in CLIP

Parameter	**Documentation**	**Value**
Embedding dimension	This is the size of the vector representations for both text and image features. In this case, it is set to 512	512
Image Resolution	The resolution to which all input images are resized. For the ViT-B/32 model, this resolution is 224x224 pixels.	224
Vision Layers	This represents the number of layers in the vision transformer model. ViT-B/32 has 12 layers.	12
Vision Width	The width of the layers in the vision transformer, which is set to 768.	768
Vision Patch Size	The size of the patches that the image is divided into before being fed into the transformer. Each patch is 32x32 pixels.	32
Context Length	The maximum number of tokens that the text transformer can process. This is set to 77 tokens.	77
Vocabulary Size	The number of unique tokens in the tokenizer's vocabulary. The vocabulary size is 49408.	49408
Transformer Width	The width of the text transformer layers, set to 512.	512
Transformer Heads	The number of attention heads in the multi-head attention mechanism of the text transformer, set to 8.	8
Transformer Layers	The number of layers in the text transformer model, which is set to 12.	12

Interpretation

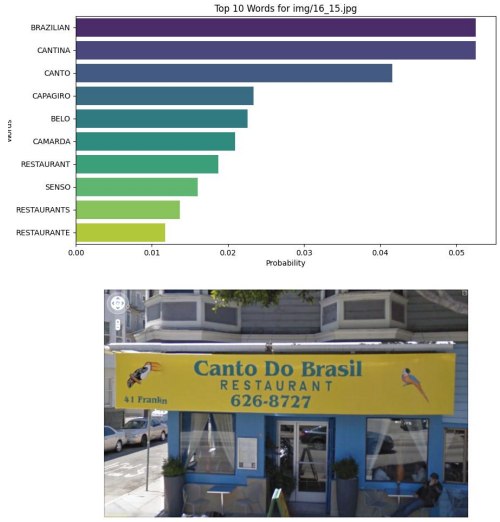
The main reason for this study is to develop an intuition about how the model works and how well the results align with reality. For each image, we present the top 10 predicted words that best describe it. This step is considered important before conducting quantitative analysis because a model that performs well statistically may not necessarily be effective in real-world environments. We selected some images and visually examined whether the predictions correspond to the actual labels in the evaluation set. Figure 2.7 illustrates the top 10 words that best describe each image in the evaluation set; for this purpose, we only took 4 of them.



(a) 00_03

(b) 01_17

(c) 02_13



(d) 16_15

Figure 2.7: Top 10 predicted words for a batch of images extracted from the evaluation set **(source: compiled by us)**

- **00_03:**
  - **Top Predicted Word:** `JOINT`: This word has the highest probability, indicating a strong prediction by the model.
  - **Other Significant Words:** `JOHNS`, `JONES`, `JOBLINE`, `RESTAURANT`: These words have significantly lower probabilities compared to `JOINT`.

  - **Visual Confirmation:** The actual image confirms the prediction. The word `JOINT` appears prominently on the sign, making it the most legitimate description of the image and validating the high probability assigned to it by the model.

- **01_17:**

  - **Top Predicted Word:** `CARROLL`: This word has the highest probability, indicating a strong prediction by the model.

  - **Other Significant Words:** `HEADQUARTERS, CARVEL, CENTRAL, CARLTON`: These words also have notable probabilities but are significantly lower than `CARROLL`.

  - **Visual Confirmation:** The actual image of `CARROLL STREET BAKERY` confirms the prediction. `CARROLL` is the most prominent word on the sign, which aligns with the model's prediction.

- **02_13:**

  - **Top Predicted Word:** `SOPRA`: This word has the highest probability, showing a strong prediction.

  - **Other Significant Words:** `SOLEFOOD, SOIFER, SOPRANO, SOTO`: These words have significant probabilities but are much lower than `SOPRA`.

  - **Visual Confirmation:** The actual image shows `Sotto Sopra`, confirming the top prediction of `SOPRA` by the model. The high probability assigned to `SOPRA` is validated by the text on the sign.

- **16_15:**

  - All the words are detected to be relatively important.

  - The predictions seem to have a coherent meaning.

This interpretation is not sufficient and not complete to say if the model works or not. The comparison to the evaluation set gives the results shown in the Table 2.4. It shows that out of 10 words, only two of them appear in the ground truth.

00_03	JUKE	JOINT	
01_17	CARROLL		
02_13	SOPRA	RESTAURANT	
16_15	BRAZILIAN	CANTO	RESTAURANT

Table 2.4: Top predicted words for selected images from the evaluation set

2.4.2 With Fine-Tuning

training

We took the ViT-B/32 checkpoint and we fine-tuned it on 100 images. The algorithm is the following:

Algorithm 1 Training CLIP

Require: Paired dataset of images $\{I_i\}_{i=1}^N$ and texts $\{T_i\}_{i=1}^N$
Require: Pre-trained image encoder f_I and text encoder f_T
Require: Temperature parameter τ
Require: Learning rate η

1. Initialize image encoder parameters θ_I and text encoder parameters θ_T
2. **while** *not converged* **do**
   (a) Sample a batch of B image-text pairs $\{(I_b, T_b)\}_{b=1}^B$
   (b) Compute image embeddings $v_i = f_I(I_i; \theta_I)$ for each image I_i
   (c) Compute text embeddings $v_t = f_T(T_i; \theta_T)$ for each text T_i
   (d) Normalize embeddings $v_i \leftarrow \frac{v_i}{\|v_i\|}$ and $v_t \leftarrow \frac{v_t}{\|v_t\|}$
   (e) Compute similarity matrix $S_{ij} = \frac{v_i \cdot v_t}{\tau}$ for all $i, j \in \{1, \ldots, B\}$
   (f) Compute the contrastive loss:

$$\mathcal{L}_I = -\frac{1}{B}\sum_{i=1}^{B} \log \frac{\exp(S_{ii})}{\sum_{j=1}^{B} \exp(S_{ij})} \tag{2.4.1}$$

$$\mathcal{L}_T = -\frac{1}{B}\sum_{i=1}^{B} \log \frac{\exp(S_{ii})}{\sum_{j=1}^{B} \exp(S_{ji})} \tag{2.4.2}$$

   (g) Compute total loss: $\mathcal{L} = \frac{1}{2}(\mathcal{L}_I + \mathcal{L}_T)$
   (h) Compute gradients $\nabla_{\theta_I}\mathcal{L}$ and $\nabla_{\theta_T}\mathcal{L}$
   (i) Update parameters:

$$\theta_I \leftarrow \theta_I - \eta \nabla_{\theta_I}\mathcal{L} \tag{2.4.3}$$

$$\theta_T \leftarrow \theta_T - \eta \nabla_{\theta_T}\mathcal{L} \tag{2.4.4}$$

end

The temperature parameter τ is a hyperparameter that controls the scale of the logits before applying the softmax. Lower temperatures result in sharper probability distributions. The contrastive loss $\mathcal{L}$ is computed for both the image-to-text and text-to-image directions to ensure that the embeddings are aligned in both modalities. The

Figure 2.8 shows the evolution of Losses (image loss, text loss and overall loss) during CLIP fine-tuning. Below the explanation of each hyperparameter.

- **Batch Size**:
  - Batch size is the number of training samples used in one iteration of model training. A larger batch size can provide more stable gradient estimates but requires more memory.
- **Optimizer**:
  - The algorithm used to adjust the model parameters in order to minimize the loss function. Adam (Adaptive Moment Estimation) is a popular optimizer that combines the advantages of two other extensions of stochastic gradient descent: AdaGrad and RMSProp. The Adam optimizer updates the parameters using the following formulas:

$$
\begin{aligned}
m_t &= \beta_1 m_{t-1} + (1-\beta_1) g_t \\
v_t &= \beta_2 v_{t-1} + (1-\beta_2) g_t^2 \\
\hat{m}_t &= \frac{m_t}{1-\beta_1^t} \\
\hat{v}_t &= \frac{v_t}{1-\beta_2^t} \\
\theta_t &= \theta_{t-1} - \frac{\alpha \hat{m}_t}{\sqrt{\hat{v}_t} + \epsilon}
\end{aligned}
$$

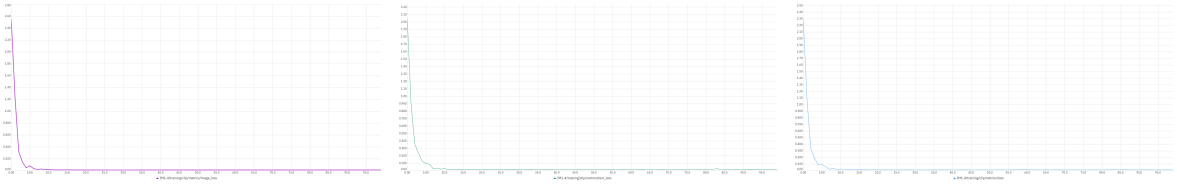
    Where:

    * m_t and v_t are the first and second moment estimates.
    * β_1 and β_2 are the exponential decay rates for the moment estimates.
    * g_t is the gradient at time step t.
    * α is the learning rate.
    * ϵ is a small constant for numerical stability.
- **Learning Rate**:
  - The learning rate is the step size at each iteration while moving toward a minimum of the loss function. A smaller learning rate means smaller updates to the model's parameters.
- **Betas (β_1, β_2)**:
  - These are the exponential decay rates for the moment estimates in the Adam optimizer. β_1 is usually close to 1 (e.g., 0.9), and it represents the decay rate for the first moment estimate (mean of gradients). β_2, also close to 1 (e.g., 0.98), represents the decay rate for the second-moment estimate (uncentered variance of gradients).

- **Epsilon** ϵ:
  - This is a small constant added to the denominator to improve numerical stability in the Adam optimizer. It prevents any division by zero during the optimization process.
- **Weight Decay**:
  - Weight decay is used to prevent overfitting by adding a penalty to the loss function based on the magnitude of the model weights. A larger weight decay can reduce overfitting but may also lead to underfitting. The penalty term is added as follows:

$$L_{\text{total}} = L_{\text{original}} + \lambda \sum_i \theta_i^2$$

    Where:
    - L_{total} is the total loss.
    - L_{original} is the original loss.
    - λ is the weight decay factor.
    - θ_i are the model parameters.



(a) Image Loss (b) Text Loss (c) Image-Text Similarity Loss

Figure 2.8: Training Losses **(source: compiled by us)**

Hyperparameters	
Initial Checkpoint	**ViT-B/32**
Number of Epochs	**100**
Batch Size	**128**
Optimizer	**Adam**
Learning Rate	5×10^{-5}
(β_1, β_2)	$(0.9,\ 0.98)$
ϵ	10^{-6}
Weight Decay	0.2
Image Loss	**Cross-Entropy**
Text Loss	**Cross-Entropy**

Table 2.5: CLIP Fine-Tuning Hyperparameters

We fine-tuned the model across 8 checkpoints, each with the same hyperparameters as shown in **Table 2.5**, but with varying numbers of epochs. An epoch represents a

complete pass through the training dataset during the training process. Increasing the number of epochs can potentially enhance model performance but also carries the risk of overfitting. Therefore, we chose to evaluate our model by varying only this hyperparameter. The consistency of the other hyperparameters, which are known to yield the optimized ViT-B/32 model, ensures a reliable basis for our evaluation.

2.4.3 Comparative Analysis

We took 8 checkpoints trained during some number of epochs. The reason of doing that is related to the fact that we don't really have reference to choose the number of epochs that lets the model to be trained well and without overfitting unlike other hyperparameters. For the evaluation we used metrics seen in section 2.3. The Table 2.6 shows the results of ViT-B/32 at different epochs of Fine-Tuning in terms of **Accuracy**, **Precision**, **Recall**, **F1-Score** and **Uniformity** measures. Sometimes using only the accuracy doesn't give us enough information about the model's performances as we have earlier seen in 2.1.1 that some words are more frequent than others. The question we should ask in order to choose which metric is important and which checkpoint is more suitable for our study is related to what we want to optimize. If we want to maximize the number of significant texts, we consider the **Recall** metric. Otherwise, if we are interested more in reducing the number of insignificant texts that don't really describe the images, then we should maximize the **Precision**.If both of them are important, so we use the **F1-Score** or weighted F1-Score if one of them is more or less important than another.

	Accuracy	**Precision**	**Recall**	**F1-Score**	**Uniformity**
ViT-B/32	**0.986354**	**0.192545**	0.041989	0.064774	**0.335908**
ViT-B/32 (+ 1)	**0.989814**	0.234369	**0.036764**	0.060481	0.250820
ViT-B/32 (+ 2)	0.989699	0.252353	0.037998	**0.060288**	**0.240897**
ViT-B/32 (+ 3)	0.989755	0.276059	0.039749	0.062460	0.250511
ViT-B/32 (+ 4)	0.989400	0.275336	0.040184	0.062257	0.264263
ViT-B/32 (+ 5)	0.989430	0.275841	0.042163	0.064984	0.283437
ViT-B/32 (+ 7)	0.989304	0.264843	0.040815	0.061682	0.270127
ViT-B/32 (+ 20)	0.989431	**0.293133**	0.042821	**0.065624**	0.291966
ViT-B/32 (+ 40)	0.988204	0.245166	**0.046244**	0.063810	0.322949

Table 2.6: Performance Metrics for ViT-B/32 at Various Epochs of Fine-Tuning

In our, we are normally interested in maximizing precision since we don't want many wrong texts that don't really explain the image's semantic, to be present in the predictions. We can assess three checkpoint in which each one gives the highest value in some metric.

- **ViT-B/32 (+1):** This configuration achieves the highest accuracy of 0.989814. However, this metric is not particularly significant due to the imbalance in the word distribution. Additionally, the standard deviation of accuracy, $\sigma_{\text{Accuracy}} = 0.001116$, indicates that accuracy is not the primary factor influencing the results.
- **ViT-B/32 (+20):** This model achieves the highest Precision and F1-Score, which are approximately 0.293133 and 0.065624, respectively. These metrics suggest that this checkpoint represents the optimal configuration for the objectives of our study, as it balances precision and overall performance effectively.

- **ViT-B/32 (+40):** This model records the highest recall at 0.046244. While this metric is impressive, the model might have been selected if it balanced other performance metrics equally well. The higher recall indicates a better identification of true positives, but the trade-offs with precision and F1-Score must also be considered.

Conclusion

Before we began working on the various tasks, we considered using existing models to gain an initial understanding of how our application might appear. For this, we have systematically explored the application and enhancement of the CLIP model in the context of the SVT (Street View Text) dataset. We started with a statistical overview of the SVT dataset to understand its characteristics and the challenges that we can have during the application.

Our experimental setup detailed the resources and methodologies used for training and testing the models, emphasizing the metrics including accuracy, precision, recall, F1-score, and the normalized similarity-entropy measure justified by the assumption of equal detection probability for all correct words, represented by a uniform distribution.

The main objective was to assess the impact of fine-tuning the CLIP model on this dataset to improve its performance in text recognition tasks. We used the ViT-B/32 checkpoint, defined by its default parameters. Our primary aim was to determine the effectiveness of this fine-tuning in improving precision and maintaining consistent performance across various measures since we have already tried the model without fine-tuning.

The analysis was bifurcated into qualitative and quantitative aspects. The qualitative analysis involved visually inspecting the top predicted words for selected images, providing insight into the model's interpretability and relevance. The quantitative analysis encompassed a thorough evaluation using refined metrics, facilitating a comparative assessment of performance across different checkpoints.

The obtained results showed notable improvements in the model's performance post-fine-tuning, as evidenced by enhanced accuracy, precision, recall, and F1-score metrics. The comparative analysis between the model's performance with and without fine-tuning underscored the efficacy of the fine-tuning process. The normalized similarity-entropy measure further validated the uniformity and reliability of the word detection probabilities.

The insights gained from this study pave the way for future research and advancements in image-text similarity. By integrating both qualitative and quantitative evaluations, we have laid a robust groundwork for further exploration in the field of computer vision and text recognition.

Chapter 3

Object Driven Image and Text Similarity Method

Introduction

In this chapter, we described our enhanced method which takes into account each of images, their objects to measure their similarities with the input text. For that, we have used SAM introduced by Kirillov et al. [2023] to segment the images, CLIP by Radford et al. [2021] to measure image and text similarities and finally used a component called Feature Enhancer Layer taken from GroundingDINO developed by Liu et al. [2023] to exchange information between different inputs (images and texts).

This chapter is divided into three sections. We first introduce the method known as Object-Driven Image and Text Similarity (ODITS) and outline its three main components. In the second section, we showcase the preliminary results obtained, including the evaluation metrics employed, the interpretation, an essential aspect that addresses the question **"why does it work?"**. We also conduct a statistical analysis on the data such as images and the distribution of probability density of segmented objects. These elements are important to design the architectural details of the model such as the optimization process. For example, if there is data imbalancing, we use weighted loss functions rather than others.

Finally, we theoretically compare our method to previously obtained ones and propose guidelines to tackle future product enhancements.

3.1 Methods

Object-Driven Image and Text Similarity (ODITS) is a framework developed by our team to consider the presence of objects in images and their relationships with text. This framework prioritizes the most semantically significant elements. The architecture of our model is divided into three main components, as illustrated in Figure 3.1.

- **Encoding Phase**

  In this phase, images, text in the vocabulary, and object images are encoded. Images are resized to 1024×1024 to accurately detect objects. The vocabulary is tokenized using the CLIP tokenizer. Images and texts are then encoded using the CLIP image encoder and CLIP text encoder, respectively, according to $E_X = \text{CLIP}_{\text{image}}(X)$ and

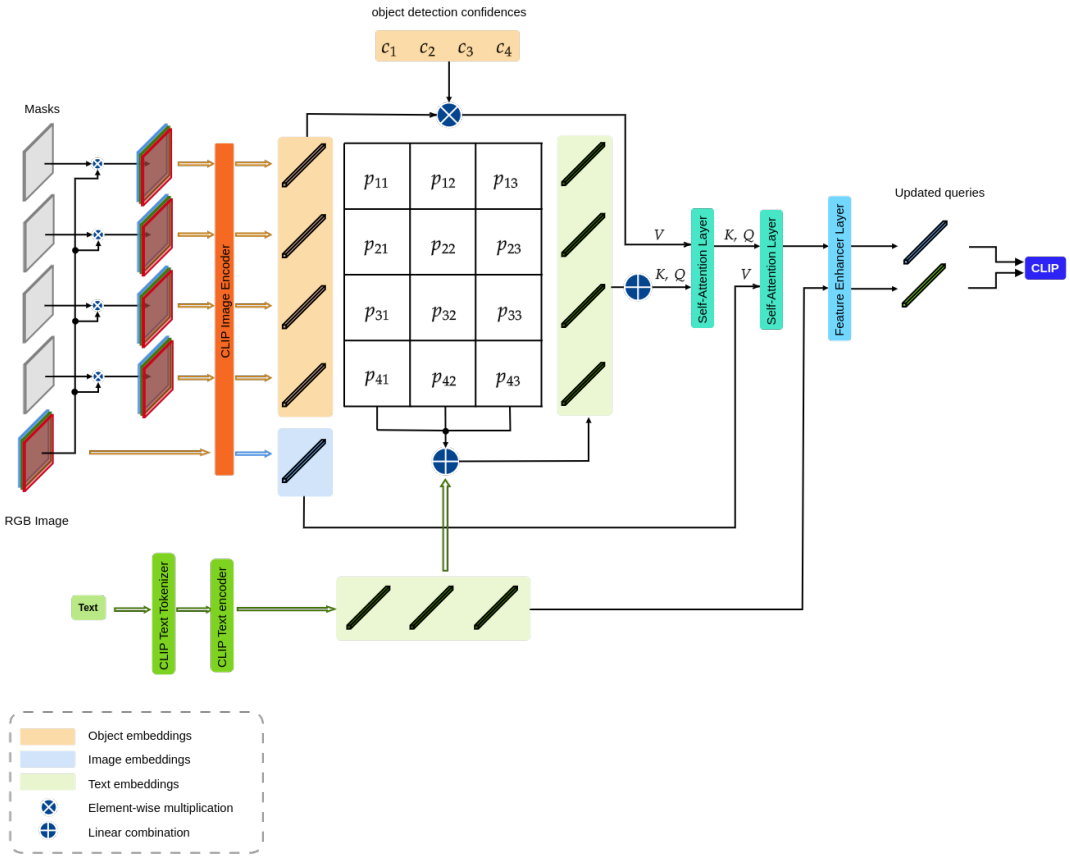


Figure 3.1: Object-Driven Image and Text Similarity (ODITS) model architecture. **(source: compiled by us)**

$E_T = \text{CLIP}_{\text{text}}(W)$. Objects require processing before encoding since segmentation returns only the mask, not the texture. Masks are applied to the image to extract objects, which are then encoded as $o^{(i)} = \text{CLIP}_{\text{image}}(M^{(i)} \circ X^{(i)})$.

For instance, consider an image containing various objects such as a cat, a dog, and a tree. Each object is detected and segmented, and these segments are then encoded separately to capture the unique features of each object.

- **Object-Text Similarity Matrix**

  The Object-Text Similarity matrix is computed using CLIP. It assigns a probability for each pair of (image object)-word to determine their correlation. For each image, we construct a similarity-weighted text embedding vector according to:

$$(E^S_{M^{(i)}})^{(j)} = \sum_{k=1}^{n} p^{(i)}_{jk} E^{(k)}_T$$

  These vectors highlight the importance of words describing each object. For example, if an image contains a cat, the similarity matrix will determine the relevance of words such as "feline," "pet," or "animal" to this object, and compute embeddings accordingly.

- **Cross-Feature Learning**

Inspired by the method in Chapter 1 and GroundingDINO by Liu et al. [2023], cross-correlations between prompts and images improve object localization from text descriptions. Our method measures correlations between images, objects, and words to determine image-text similarity.

Objects are detected in image $X^{(j)}$ with probabilities conf_j, which regularize the object embeddings $E_{M^{(i)}}$. To achieve two encoding vectors, one for the image and one for the text, we aggregate image and object embeddings into one component via a Self-Attention Layer:

$$\text{softmax}\left(\frac{QK^T}{\sqrt{d}}\right)V$$

where $V^{(i)} = \sum_{j=1}^{K_i} c_j E^{(j)}_{M^{(i)}}$ and $K^{(i)} = Q^{(i)} = \sum_{j=1}^{K_i} (E^S_{M^{(i)}})^{(j)}$. The output, a linearly combined vector of objects containing word importance, is fed into another Self-Attention Layer to extract information about the image and its object relationships. Here, $V = E_X$ and $K = Q$ represent the previous output.

Finally, we learn the cross image-object and text feature cross-attentions so that each vector incorporates information from the other. The Feature Enhancer Layer architecture is similar to Figure 1.1. These outputs are fed into the CLIP model to learn cosine similarities between the updated image and text embeddings.

For example, when processing an image of a cat sitting under a tree, the model integrates features from both the cat and the tree with relevant textual descriptions, enhancing the overall understanding of the scene.

Before delving into the details, we establish the mathematical notation used throughout this chapter shown in the Table 3.1.

Table 3.1: Mathematical notation of ODITS model

	Notation
$X \in \mathbb{R}^{N \times C \times W \times H}$	Set of RGB images, where N is the number of examples, W, H, and $C = 3$ are the width, height, and channels. The i-th example is $X^{(i)}$.
$\mathcal{M} = \{M^{(i)} \in \{0,1\}^{K_i \times W \times H}\}_{i=1}^{N}$	Set of binary masks. Each image has K_i objects.
$\mathcal{O} = \{o^{(i)} \in \mathbb{R}^{K_i \times C \times W \times H}\}_{i=1}^{N}$	Set of objects segmented from each image.
$\text{conf}_j = (c_1, \ldots, c_{K_i})$, $\forall j = 1, N$	Object detection probabilities for each image j.
$W = \{w_1, \ldots, w_n\}$	Vocabulary (set of words), where n is its size.
$E_X \in \mathbb{R}^{N \times d}$	Image embeddings.
$E_T \in \mathbb{R}^{n \times d}$	Text embeddings.
$E_{M^{(i)}} \in \mathbb{R}^{K_i \times d}$	Object embeddings for each image.

Continued on next page

	Notation
$E_{\cdot}^{U}, E_{\cdot}^{S}$	Updated embeddings and similarity-weighted embeddings, respectively.
$\mathcal{P}^{(i)} \in [0,1]^{K_i \times n}$	Object-Text Similarity matrix.
$p_{jk}^{(i)}$	Similarity between the j-th object of the i-th image and the k-th word of the vocabulary.
$\circ$	Binary operator representing element-wise multiplication.
$\text{CLIP}_{\text{image}}$	CLIP image encoder
$\text{CLIP}_{\text{text}}$	CLIP text encoder
CLIP	CLIP model that takes both of image and text as input and returns the similarity between them

3.2 Optimization Details and Reproducibility

Our approach is an improvement of the CLIP method, meaning there are some architectural components shared between CLIP and ODITS. Although we could initialize the weights uniformly, it is unnecessary to retrain the entire model since we have already fine-tuned CLIP on the SVT dataset. Instead, we load the shared components' weights into ODITS and uniformly initialize the other components, such as the two additional Self-Attention layers and the Feature Enhanced Layer.

An experiment is considered **reproducible** if different teams can achieve the same results using the identical setup and conditions. This should not be confused with **replicability**, which refers to the ability of an experiment or study to be duplicated by different researchers using their own equipment and methods to achieve the same results. **Repeatability**, on the other hand, refers to the ability to consistently reproduce the same results when the same experiment is conducted by the same team using the same methods, equipment, and conditions over multiple trials. This ensures the reliability of the results within a single study.

We have made our experiments reproducible by following a specific framework and several steps. It is well-known that achieving identical learning curves each time an experiment is redone is not trivial. To make this possible, we need to start from the same point. In other words, the user should initialize the model with the same weights as we did. The ODITS model is divided into two components:

- **Shared Components**: These are the same as in CLIP. We load these weights from the previously fine-tuned checkpoint.
- **Non-Shared Components**: These are initialized using ***Pseudo-Random Number Generators (PRNGs)***. Generally, we use uniform initialization to help mitigate issues of vanishing and exploding gradients. Examples include **Xavier**, **Kaiming**, and other methods such as **MRG32k3a & MRG32kp**, **Mersenne Twisters (MT19937, SFMT, MTGP) & WELL families of generators**, or **MLFG_6331_64**, as introduced by Glorot and Bengio [2010], He et al. [2015], L'Ecuyer [2015], Matsumoto and Nishimura [1998] and L'Ecuyer [1999] respectively. These are considered the best pseudo-random number generators currently available.

– **Xavier Initialization**: The Xavier initialization uses a uniform distribution within the range:

$$W \sim \text{Uniform}\left(-\frac{\sqrt{6}}{\sqrt{n_{\text{in}} + n_{\text{out}}}}, \frac{\sqrt{6}}{\sqrt{n_{\text{in}} + n_{\text{out}}}}\right)$$

– **Kaiming Uniform Initialization**: The Kaiming uniform initialization uses a uniform distribution within the range:

$$W \sim \text{Uniform}\left(-\sqrt{\frac{6}{n_{\text{in}}}}, \sqrt{\frac{6}{n_{\text{in}}}}\right)$$

– **Mersenne Twister (MT19937)**: The Mersenne Twister algorithm is defined by a recurrence relation:

$$x_n = (x_{n-r} \oplus (x_{n-s} \text{ and mask})) \text{ where } r = 19, s = 7$$

– **MRG32k3a**: A combined multiple recursive generator defined by:

$$x_n = (a_1 x_{n-1} + a_2 x_{n-2} + a_3 x_{n-3}) \mod m$$

where a_1, a_2, a_3 are specific constants and m is the modulus.

– **MLFG_6331_64**: A multiplicative lagged Fibonacci generator defined by:

$$x_n = (x_{n-6331} \times x_{n-1}) \mod 2^{64}$$

Figure 3.2 shows the pipeline for saving the initial weights to the Neptune.ai platform, allowing them to be directly loaded into the ODITS model for reproducibility.

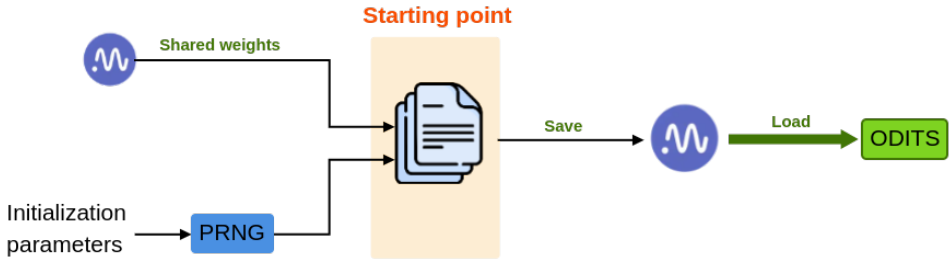


Figure 3.2: ODITS Weight Initialization Pipeline **(source: compiled by us)**

This alone is not sufficient to ensure determinism. Fortunately, **PyTorch** allows us to use deterministic algorithms by adding the following instructions at the beginning of the code:

```
1 torch.use_deterministic_algorithms(True)
```

Therefore, this solution works only on CPU. In order to have reproducibility on GPU. In the following Python code snippet using PyTorch, we configure cuDNN (CUDA Deep Neural Network library) for deterministic behavior during training. Setting `torch.backends.cudnn.benchmark` to `False` ensures that cuDNN selects deterministic algorithms, though this may impact performance. Conversely, `torch.backends.cudnn.deterministic` is set to `True` to enforce the use of deterministic convolution algorithms, potentially slowing down performance. However, it's important to note that complete determinism in the training process cannot be guaranteed if other libraries employing nondeterministic algorithms are utilized alongside PyTorch.

```
torch.backends.cudnn.benchmark = False
torch.backends.cudnn.deterministic = True
```

3.3 Results

3.3.1 Preliminary Results of Our Method

SAM Model

To segment the images, we used the 'SamAutomaticMaskGenerator' with its default hyperparameters that are summarized in Table 3.2.

Table 3.2: Hyperparameters for SamAutomaticMaskGenerator

Parameter	Value	Description
Model	Sam	The SAM model to use for mask prediction.
Points per Side	32	The number of points sampled along one side of the image, total points being points_per_side^2. If None, point_grids must provide explicit point sampling.
Points per Batch	64	The number of points run simultaneously by the model. Higher numbers may be faster but use more GPU memory.
Prediction IoU Threshold	0.88	A filtering threshold in [0,1], using the model's predicted mask quality.
Stability Score Threshold	0.95	A filtering threshold in [0,1], using the stability of the mask under changes to the cutoff used to binarize the model's mask predictions.
Stability Score Offset	1.0	The amount to shift the cutoff when calculating the stability score.
Box NMS Threshold	0.7	The box IoU cutoff used by non-maximal suppression to filter duplicate masks.
Crop N Layers	0	If >0, mask prediction will be run again on crops of the image. Sets the number of layers to run, where each layer has 2^{i_layer} number of image crops.
Crop NMS Threshold	0.7	The box IoU cutoff used by non-maximal suppression to filter duplicate masks between different crops.

Continued on next page

Table 3.2 – continued from previous page

Parameter	Value	Description
Crop Overlap Ratio	0.341	Sets the degree to which crops overlap. In the first crop layer, crops will overlap by this fraction of the image length. Later layers with more crops scale down this overlap.
Crop N Points Downscale Factor	1	The number of pointsperside sampled in layer n is scaled down by $crop\_n\_points\_downscale\_factor^n$.
Point Grids	None	A list over explicit grids of points used for sampling, normalized to [0,1]. The nth grid in the list is used in the nth crop layer. Exclusive with points_per_side.
Minimum Mask Region Area	0	If >0, postprocessing will be applied to remove disconnected regions and holes in masks with area smaller than min_mask_region_area. Requires opencv.
Output Mode	binary_mask	The form masks are returned in. Can be 'binary_mask', 'uncompressed_rle', or 'coco_rle'. 'coco_rle' requires pycocotools. For large resolutions, 'binary_mask' may consume large amounts of memory.

To implement our proposed solution, we generated object detection annotations using the SAM model. We extracted annotations for each image in the training set and combined them as depicted in Figure 3.3.

Once this is completed, we count the number of segmented objects for each image and create a distribution for both frequency and density, as depicted in the figure 3.6 below:

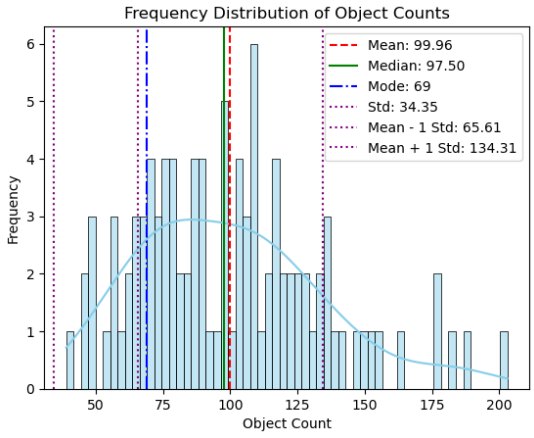


Figure 3.4: Frequency distribution

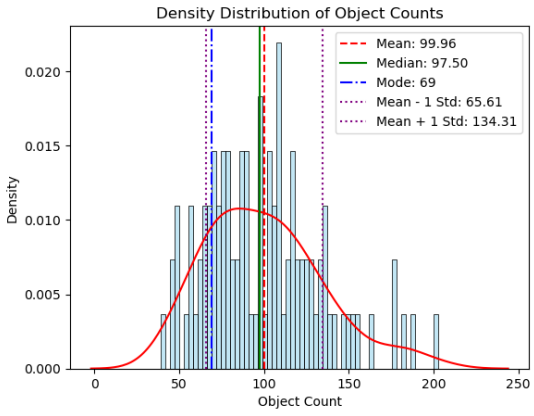


Figure 3.5: Density distribution

Figure 3.6: Frequency and Density Distributions **(source: compiled by us)**

Analysis of Object Count Distributions

- **Figure 3.4** shows the frequency distribution of object counts. The histogram represents the frequency of object counts across different bins, and a KDE (Kernel Density Estimate) curve is overlaid to show the distribution's shape. Key statistics such as the mean, median, mode, and standard deviation are marked on the plot.

(a) Original Image 1

(b) Segmented Image 1

(c) Original Image 2

(d) Segmented Image 2

(e) Original Image 3

(f) Segmented Image 3

Figure 3.3: Comparison of Original and Segmented Images **(source: compiled by us)**

- **Figure 3.5** illustrates the density distribution of object counts. This histogram, combined with a density curve, provides a clearer view of the distribution's shape and smoothness. Important statistical indicators are also highlighted on this plot.

We applied a hypothesis test to determine if the distribution fits a normal distribution. Observing the right skewness in the distribution, we also considered the possibility of an exponential distribution.

Let's define the hypotheses as follows:

- Null Hypothesis (H_0): The data follow an exponential distribution.
- Alternative Hypothesis (H_1): The data do not follow an exponential distribution.

The decision rule for the Anderson-Darling test at a significance level of $\alpha = 0.05$ is:

$$\text{Reject } H_0 \text{ if Test Statistic} > \text{Critical Value}_{0.05} \tag{3.3.1}$$

Otherwise, do not reject H_0.

The results of the hypothesis test are summarized in the following table:

Table 3.3: Hypothesis Test Results

Statistic	**Value**
Test Statistic	20.4421
Critical Value (95% Confidence)	1.333
Conclusion	The null hypothesis can be rejected.

3.3.2 Comparison with the Existing CLIP Method

As previously discussed, the CLIP model is designed to analyze both text and images to determine image-text similarity, essentially capturing the semantic connection between text and visual content. In our proposed solution, ODITS, we suggest incorporating segmented objects from the images into the CLIP model. This approach shifts the focus from image-text similarity to object-text similarity. Our method proves to be more effective due to the detailed segmentation of objects within images. By leveraging the SAM model to produce object masks, we are able to accurately extract the individual objects while preserving vital image information. Furthermore, by inputting the complete original image into our model, we ensure that the role of CLIP remains unchanged, with the added benefit of enriching the process by including object information. Moreover, our goal is to combine the GroundingDINO model with SAM in order to identify objects by name. This will enhance our understanding of the key objects in the image that serve to bridge the gap between text and visual content, thus interpreting these outcomes significantly contributes to explainable AI.

Detailed Comparison between ODITS and CLIP

The table below 3.4 compares ODITS and CLIP, focusing on their features and capabilities. ODITS, which is built for advanced tasks that integrate images and text, includes specialized functions like object segmentation, self-attention mechanisms, and handling imbalanced data. While CLIP can measure image-text similarity and perform zero-shot learning, it does not have many of the advanced features that ODITS offers. This comparison helps clarify the strengths and unique aspects of each model, providing a better understanding of their differences and how they can be used.

Table 3.4: Comparison between ODITS and CLIP

Feature	ODITS	CLIP
Object Segmentation	✓	✗
Image-Text Similarity Measurement	✓	✓
Integration of Feature Enhancer Layer	✓	✗
Usage of SAM for Image Segmentation	✓	✗
Cross-Feature Learning Capability	✓	✗
Self-Attention Mechanisms Implementation	✓	✗
Object-Text Similarity Matrix Calculation	✓	✗
Encoding of Images and Texts	✓	✓
Handling Imbalanced Data with Weighted Loss Functions	✓	✗
Requirement of Preprocessing for Object Segmentation	✓	✗
Zero-Shot Learning Capability	✓	✓
Use of Contrastive Pre-Training	✓	✓
Modular Architecture for Extensibility	✓	✗
Use of GroundingDINO for Object Identification	✓	✗
Flexibility in Handling Various Input Formats	✓	✓
Explainability of Model Predictions	✓	✗

Enhanced Explanation and Analysis

Object Segmentation and Identification

- ODITS: Integrates SAM for precise object segmentation and GroundingDINO for object identification, allowing the model to understand and isolate individual objects within an image. This enhances the granularity of the analysis and improves object-text similarity measurements.
- CLIP: Focuses on the overall image and text similarity without explicit object segmentation, leading to a broader but less detailed understanding of the visual content.

Feature Enhancement and Cross-Feature Learning

- ODITS: Incorporates a feature enhancer layer and uses self-attention mechanisms to improve cross-feature learning, thereby enhancing the interaction between image and text features.
- CLIP: Relies on the encoded features from the text and image encoders without additional feature enhancement, potentially limiting the depth of feature interaction.

Explainability and Interpretability

- ODITS: By using object segmentation and identification, ODITS provides more interpretable and explainable predictions, as it can highlight specific objects and their corresponding textual descriptions.
- CLIP: While effective, CLIP's predictions are less interpretable as they do not explicitly link objects within images to their textual counterparts.

Handling Imbalanced Data

- ODITS: Implements weighted loss functions to handle imbalanced data, ensuring better performance across diverse datasets.
- CLIP: Does not explicitly address data imbalance, which may affect performance on skewed datasets.

Flexibility and Extensibility

- ODITS: Designed with a modular architecture, allowing easy integration of new components such as different segmentation models or feature enhancers.
- CLIP: While flexible, its architecture is less modular, making it more challenging to extend with additional functionalities without significant modifications.

Conclusion

In this final chapter, we introduced and detailed the Object-Driven Image and Text Similarity (ODITS) method, highlighting its components and enhancements over existing methods. We presented it theoretically and explained its operational mechanics. We also discussed the encoding phase in detail and introduced the Object Text Similarity matrix, highlighting its added value to the existing approach. Furthermore, we elaborated on the cross-feature learning and provided the optimization details.

Following this, we presented our preliminary results, showcasing the outcomes from object segmentation using the SAM model. We analyzed these results comprehensively and compared our solution, ODITS, with the existing approach, CLIP. This comparative analysis with underscored the advantages of ODITS in terms of explainability, feature learning, and handling imbalanced data. This sets a strong foundation for future enhancements and applications of the ODITS framework.

General Conclusion

Recent advancements in object detection, text recognition, and text spotting have demonstrated significant importance in these fields. Numerous studies have sought to implement these techniques on images, particularly on scene images containing text. They have used methods to detect and recognize text in these images, as well as to establish similarity between existing and non-existing text in images, and to analyze the semantics of the image itself, similar to the approach taken by CLIP model. However, the precision of these methods using only the text and the provided image is limited, as the model lacks comprehensive information about the image. To address this issue, we propose the inclusion of object-text similarity to enhance the existing method. Rather than clearly considering the image and attempting to find the relationship between it and the text, we aim to explore the relationship between the text and each object within the image.

In this project, we began by discussing the GroundingDINO model, a cutting-edge object detector capable of identifying novel objects not present in the training data. We then explored the SAM model, which excels in segmenting objects in images through various techniques, including text prompts and training on large, high-quality datasets. Additionally, we delved into the realm of vision transformers, highlighting the ViTSTR model for its proficiency in text recognition using a single transformer encoder. We also examined two advanced scene text spotting methods, DeepSolo and ESTextSpotter, both of which leverage transformer architectures. Finally, we underscored the significance of integrating transformers and large language models by presenting the CLIP model.

Our exploration included a systematic application and enhancement of the CLIP model within the context of the Street View Text (SVT) dataset. We provided a statistical overview of the SVT dataset, outlining its characteristics and potential challenges. Our experimental setup detailed the resources and methodologies used for training and testing, emphasizing critical metrics such as accuracy, precision, recall, F1-score, and the normalized similarity-entropy measure.

The primary objective was to evaluate the impact of fine-tuning the CLIP model on the SVT dataset to enhance its performance in text recognition tasks. Using the ViT-B/32 checkpoint with default parameters, we conducted a thorough analysis, both qualitative and quantitative. The qualitative analysis involved visually inspecting top predicted words for selected images, while the quantitative analysis encompassed refined metrics for a comparative assessment. The results demonstrated significant improvements in the model's performance post-fine-tuning, with enhanced accuracy, precision, recall, and F1-score metrics. The normalized similarity-entropy measure further validated the uniformity and reliability of the word detection probabilities.

Expanding upon this groundwork, we presented our approach named Object-Driven Image and Text Similarity (ODITS), outlining its theoretical basis and functional mechanisms. We extensively examined the encoding stage and introduced the groundbreaking Object Text Similarity matrix, enhancing current methodologies significantly. Additionally, we delved into cross-feature learning and supplied optimization specifics. Initial findings illustrated the efficacy of ODITS in object segmentation with the SAM model. By conducting thorough scrutiny, we showcased the benefits of ODITS compared to CLIP, especially regarding interpretability, feature acquisition, and managing unbalanced data.

Perspectives

These perspectives will guide our future research, aiming to refine the ODITS framework and establish it as a leading solution in the field of object-text similarity in images and computer vision:

- **Implementation of the Overall Architecture**: We will implement the full ODITS architecture and rigorously test its performance. This will include optimizing the integration of object-text similarity to challenge and surpass the results obtained by the CLIP model.
- **Application to Other Benchmark Datasets**: To validate the robustness and generalizability of ODITS, we plan to apply our method to additional benchmark datasets. This will help in assessing its effectiveness across various contexts and types of scene images.
- **Enhancing Interpretability**: We will focus on enhancing the interpretability of our model, highlighting the importance of explainable AI. By showing how objects and text are linked within images, we aim to provide deeper insights and more intuitive understanding of the results, thereby increasing the transparency of the model.
- **Optimization and Fine-Tuning**: Continued efforts will be directed towards optimizing the model's parameters and fine-tuning the architecture to ensure maximal performance and efficiency.

Bibliography

R. Atienza. Vision transformer for fast and efficient scene text recognition. In *International conference on document analysis and recognition*, pages 319–334. Springer, 2021.

N. Carion, F. Massa, G. Synnaeve, N. Usunier, A. Kirillov, and S. Zagoruyko. End-to-end object detection with transformers. In *European Conference on Computer Vision*, pages 213–229. Springer, 2020.

M. Caron, H. Touvron, I. Misra, H. Jégou, J. Mairal, P. Bojanowski, and A. Joulin. Emerging properties in self-supervised vision transformers. In *Proceedings of the IEEE/CVF international conference on computer vision*, pages 9650–9660, 2021.

A. Frome, G. S. Corrado, J. Shlens, S. Bengio, J. Dean, M. A. Ranzato, and T. Mikolov. Devise: A deep visual-semantic embedding model. In C. Burges, L. Bottou, M. Welling, Z. Ghahramani, and K. Weinberger, editors, *Advances in Neural Information Processing Systems*, volume 26. Curran Associates, Inc., 2013. URL `https://proceedings.neurips.cc/paper_files/paper/2013/file/7cce53cf90577442771720a370c3c723-Paper.pdf`.

Z. Fu. Vision transformer: Vit and its derivatives. *arXiv preprint arXiv:2205.11239*, 2022.

R. Girshick. Fast r-cnn. In *Proceedings of the IEEE international conference on computer vision*, pages 1440–1448, 2015.

X. Glorot and Y. Bengio. Understanding the difficulty of training deep feedforward neural networks. In *Proceedings of the thirteenth international conference on artificial intelligence and statistics*, pages 249–256. JMLR Workshop and Conference Proceedings, 2010.

K. He, X. Zhang, S. Ren, and J. Sun. Delving deep into rectifiers: Surpassing human-level performance on imagenet classification. In *Proceedings of the IEEE international conference on computer vision*, pages 1026–1034, 2015.

K. He, G. Gkioxari, P. Dollár, and R. Girshick. Mask r-cnn. In *Proceedings of the IEEE international conference on computer vision*, pages 2961–2969, 2017.

M. Huang[1*], J. Zhang[2*], D. Peng[1], H. Lu[3], C. Huang[2], Y. Liu[3], X. Bai[3], and L. Jin[1]∗. Estextspotter: Towards better scene text spotting with explicit synergy in transformer. August 2023.

J. Jain, J. Li, M. T. Chiu, A. Hassani, N. Orlov, and H. Shi. Oneformer: One transformer to rule universal image segmentation. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 2989–2998, 2023.

C. Jia, Y. Yang, Y. Xia, Y.-T. Chen, Z. Parekh, H. Pham, Q. Le, Y.-H. Sung, Z. Li, and T. Duerig. Scaling up visual and vision-language representation learning with noisy text supervision. In *International conference on machine learning*, pages 4904–4916. PMLR, 2021.

A. Kirillov, Y. Wu, K. He, and R. Girshick. Pointrend: Image segmentation as rendering. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pages 9799–9808, 2020.

A. Kirillov, E. Mintun, N. Ravi, H. Mao, C. Rolland, L. Gustafson, T. Xiao, S. Whitehead, A. C. Berg, W.-Y. Lo, et al. Segment anything. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pages 4015–4026, 2023.

A. Krizhevsky, I. Sutskever, and G. E. Hinton. Imagenet classification with deep convolutional neural networks. *Communications of the ACM*, 60(6):84–90, 2017.

P. L'Ecuyer. Random number generation with multiple streams for sequential and parallel computing. In *2015 Winter Simulation Conference (WSC)*, pages 31–44. IEEE, 2015.

S. Liu, Z. Zeng, T. Ren, F. Li, H. Zhang, J. Yang, C. Li, J. Yang, H. Su, J. Zhu, et al. Grounding dino: Marrying dino with grounded pre-training for open-set object detection. *arXiv preprint arXiv:2303.05499*, 2023.

P. L'Ecuyer. Tables of maximally equidistributed combined lfsr generators. *Mathematics of computation*, 68(225):261–269, 1999.

M. Matsumoto and T. Nishimura. Mersenne twister: a 623-dimensionally equidistributed uniform pseudo-random number generator. *ACM Transactions on Modeling and Computer Simulation (TOMACS)*, 8(1):3–30, 1998.

F. Monay and D. Gatica-Perez. On image auto-annotation with latent space models. In *Proceedings of the eleventh ACM international conference on Multimedia*, pages 275–278, 2003.

M. Oquab, T. Darcet, T. Moutakanni, H. Vo, M. Szafraniec, V. Khalidov, P. Fernandez, D. Haziza, F. Massa, A. El-Nouby, et al. Dinov2: Learning robust visual features without supervision. *arXiv preprint arXiv:2304.07193*, 2023.

K. O'shea and R. Nash. An introduction to convolutional neural networks. *arXiv preprint arXiv:1511.08458*, 2015.

A. Paszke, S. Gross, F. Massa, A. Lerer, J. Bradbury, G. Chanan, T. Killeen, Z. Lin, N. Gimelshein, L. Antiga, et al. Pytorch: An imperative style, high-performance deep learning library. *Advances in neural information processing systems*, 32, 2019.

A. Radford, J. W. Kim, C. Hallacy, A. Ramesh, G. Goh, S. Agarwal, G. Sastry, A. Askell, P. Mishkin, J. Clark, et al. Learning transferable visual models from natural language supervision. In *International conference on machine learning*, pages 8748–8763. PMLR, 2021.

J. Redmon, S. Divvala, R. Girshick, and A. Farhadi. You only look once: Unified, real-time object detection. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 779–788, 2016.

S. Ren, K. He, R. Girshick, and J. Sun. Faster r-cnn: Towards real-time object detection with region proposal networks. *Advances in neural information processing systems*, 28, 2015.

B.-K. Ruan, H.-H. Shuai, and W.-H. Cheng. Vision transformers: state of the art and research challenges. *arXiv preprint arXiv:2207.03041*, 2022.

R. M. Schmidt. Recurrent neural networks (rnns): A gentle introduction and overview. *arXiv preprint arXiv:1912.05911*, 2019.

Sivic and Zisserman. Video google: A text retrieval approach to object matching in videos. In *Proceedings ninth IEEE international conference on computer vision*, pages 1470–1477. IEEE, 2003.

A. W. Smeulders, M. Worring, S. Santini, A. Gupta, and R. Jain. Content-based image retrieval at the end of the early years. *IEEE Transactions on pattern analysis and machine intelligence*, 22(12):1349–1380, 2000.

A. Stafylopatis and A. Likas. Pictorial information retrieval using the random neural network. *IEEE Transactions on Software Engineering*, 18(7):590, 1992.

A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, L. Kaiser, and I. Polosukhin. Attention is all you need. *In Advances in Neural Information Processing Systems*, page 6000–6010, 2017.

M. Ye, J. Zhang, S. Zhao, J. Liu, T. Liu, B. Du, and D. Tao. Deepsolo: Let transformer decoder with explicit points solo for text spotting. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 19348–19357, 2023.

X. Zhang, Y. Su, S. Tripathi, and Z. Tu. Text spotting transformers. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2022.

W. X. Zhao, K. Zhou, J. Li, T. Tang, X. Wang, Y. Hou, Y. Min, B. Zhang, J. Zhang, Z. Dong, et al. A survey of large language models. *arXiv preprint arXiv:2303.18223*, 2023a.

Z. Zhao, Y. Liu, H. Wu, Y. Li, S. Wang, L. Teng, D. Liu, X. Li, Z. Cui, Q. Wang, et al. Clip in medical imaging: A comprehensive survey. *arXiv preprint arXiv:2312.07353*, 2023b.

F. Zhuang, Z. Qi, K. Duan, D. Xi, Y. Zhu, H. Zhu, H. Xiong, and Q. He. A comprehensive survey on transfer learning. *Proceedings of the IEEE*, 109(1):43–76, 2020.