

UNIVERSITÀ
DEGLI STUDI
DI PADOVA



DIPARTIMENTO
DI INGEGNERIA
DELL'INFORMAZIONE

MASTER THESIS IN COMPUTER ENGINEERING

Visual Language Models: an in-depth exploration of ViLT

MASTER CANDIDATE

Giacomo Gonella

Student ID 2056116

SUPERVISOR

Prof. Giorgio Satta

University of Padova

CO-SUPERVISOR

Prof. Loris Nanni

University of Padova

ACADEMIC YEAR
2023/2024

To my family

Abstract

The relentless surge in human-AI interaction in recent years has propelled Natural Language Processing (NLP) and Computer Vision (CV) to the forefront of this transformative evolution. The advent of LLMs in the former field has revolutionized the global landscape, while the latter remains a pivotal component in the ongoing pursuit of automating diverse aspects of our daily lives.

This thesis delves into the exploration of the Vision Language (VL) field, tracing its evolution from inception and subsequently narrowing its focus to specific aspects. Specifically chosen for its simplicity and demonstrated competitive results, the ViLT model serves as the starting point for conducting tests and experiments. The primary aim is to reveal the intricate relationship between words and images.

The analysis extensively delves into the model, subjecting it to rigorous testing across key downstream tasks characteristic of VLMs. The central focus of this research involves a comprehensive examination of challenges impacting the model, with a particular emphasis on the object counting task. Various techniques are employed in the proposed solutions, including leveraging datasets, modifying phrases to generate new instances, utilizing a zero-shot segmenter to enhance the model's inference, adapting the model's input reception using a Convolutional Neural Network (CNN) for improved feature extraction, and culminating in the implementation of an advanced training technique. Each of these aspects serves as a topic for discussion.

This dissertation serves as a key reference for discussions and further exploration within the VL domain. It stands as a comprehensive investigation, highlighting the intricacies of the field by uncovering diverse challenges and possibilities derived from various obtained results. The findings contribute not only to the comprehension of potential issues in the VL domain but also lay the foundation for subsequent investigations and advancements in this evolving field.

Sommario

L'incessante aumento dell'interazione uomo-IA negli ultimi anni ha spinto il Natural Language Processing (NLP) e la Computer Vision (CV) in prima linea in questa evoluzione trasformativa. L'avvento dei LLM nel primo campo ha rivoluzionato il panorama globale, mentre il secondo rimane una componente fondamentale nella continua ricerca di automatizzare diversi aspetti della nostra vita quotidiana.

Questa tesi si addentra nell'esplorazione del campo della VL, tracciandone l'evoluzione fin dall'inizio e restringendo successivamente l'attenzione ad aspetti specifici. Scelto appositamente per la sua semplicità e per i risultati competitivi dimostrati, il modello ViLT serve come punto di partenza per condurre test ed esperimenti. L'obiettivo principale è quello di rivelare l'intricata relazione tra parole e immagini.

L'analisi approfondisce ampiamente il modello, sottoponendolo a test rigorosi su compiti chiave caratteristici dei VLM. L'obiettivo centrale di questa ricerca è l'esame completo delle sfide che il modello deve affrontare, con particolare attenzione al compito di conteggio degli oggetti. Le soluzioni proposte utilizzano diverse tecniche, tra cui lo sfruttamento di dataset, la modifica di frasi per generare nuove istanze, l'utilizzo di un segmentatore zero-shot per migliorare l'inferenza del modello, l'adattamento della ricezione di input del modello utilizzando una CNN per migliorare l'estrazione delle feature e culminando nell'implementazione di una tecnica di addestramento avanzata. Ognuno di questi aspetti costituisce un argomento di discussione.

La presente dissertazione costituisce un riferimento fondamentale per le discussioni e le ulteriori esplorazioni nell'ambito della VL. Si tratta di un'indagine completa, che mette in luce le complessità del campo scoprendo diverse sfide e possibilità derivanti dai vari risultati ottenuti. I risultati contribuiscono non solo alla comprensione di potenziali problemi nel dominio VL, ma gettano anche le basi per successive indagini e progressi in questo campo in evoluzione.

Contents

List of Figures	xi
List of Tables	xiii
List of Code Snippets	xv
List of Acronyms	xvii
1 Introduction	1
1.1 Motivations	1
1.2 History of Vision Language	2
1.2.1 First steps	2
1.2.2 Vision language joint representation	3
1.2.3 Future of field	8
1.3 ViLT	9
1.3.1 ViLT architecture	9
1.3.2 Pre-training	11
1.3.3 Implementation	12
2 The Counting Problem	15
2.1 NLVR2	15
2.1.1 Origin of dataset	16
2.1.2 Experiments	17
2.1.3 Errors	19
2.2 Counting in VLM	21
2.2.1 Other approaches	22
2.2.2 Counting Probe Dataset	22

CONTENTS

3	Visual Entailment	25
3.1	Dataset and experiments	25
3.1.1	Description	26
3.1.2	Results	27
3.1.3	Errors	28
3.2	Counting Probe VE	29
3.2.1	Results	30
3.2.2	Balanced dataset	31
3.2.3	Augmented dataset	32
3.3	Segmentation	33
3.3.1	Improving with zero-shot segmentators	34
3.3.2	Counting with segmented images	35
3.3.3	Errors	37
4	Visual Question Answering	39
4.1	Experiments	39
4.1.1	VQAv2	40
4.1.2	Counting probe VQA	41
4.1.3	VQAv2 counting	43
4.2	CNNs	45
4.2.1	Linear projection	46
4.2.2	Reshaping	48
4.2.3	Errors	49
5	Generative VLMs	51
5.1	Counting via MLM	51
5.1.1	Dataset and experiments	52
5.1.2	Errors	53
5.2	Visual Instruction Tuning	54
5.2.1	LLaVA	55
6	Low-Rank Adaptation	59
6.1	What is LoRA	59
6.1.1	Method	60
6.1.2	Results	61
6.2	Experiments	62
6.2.1	Visual Entailment	63

6.2.2	Visual Entailment Counting	64
7	Conclusions and Future Works	67
A	Datasets management	71
A.1	Image paths	71
A.2	NLVR2	73
A.3	VE	74
A.4	LM	75
A.5	VQAv2	76
A.5.1	Standard	76
A.5.2	ResNetV2 features	77
A.5.3	SqueezeNet features	78
A.6	DataLoader	79
B	Use of SAM	81
C	ViLT-CNN hybrid models	83
C.1	Linear Projection	83
C.2	Reshaping	84
C.2.1	ResNetV2	84
C.2.2	SqueezeNet	86
D	Use of LoRA	89
	References	91
	Acknowledgments	95

List of Figures

1.1	Example of instances from Vision Question Answering (VQA) dataset.	3
1.2	Co-attentional transformer block.	7
1.3	Four categories of Vision Language Pre-training (VLP) models. . .	10
1.4	Comparing Transformer and ViT.	10
2.1	Example of instance of Natural Language for Visual Reasoning (NLVR) dataset.	16
2.2	Example of instance of Natural Language for Visual Reasoning for Real (NLVR2) dataset.	17
2.3	Example of correct predictions.	19
2.4	Example of wrong predictions with specific details description. . .	20
2.5	Example of wrong prediction with complex description.	20
2.6	Example of wrong predictions in counting descriptions.	21
3.1	Confusion Matrix of evaluation on Visual Entailment (VE) dev dataset.	27
3.2	VE error example.	28
3.3	Example of instance of VE dataset.	29
3.4	ViLT VE counting catastrophic forgetting confusion matrix.	31
3.5	Example of augmented VE counting instance. The sentence is deliberately not in fluent English; refer to the explanation below for the motivation behind this choice.	32
3.6	Counting probe segmentation example.	35
4.1	Example of “many” label prediction.	43
4.2	VQAv2 counting 10 confusion matrix.	45
4.3	10 epochs fine-tuning.	47

LIST OF FIGURES

4.4	20 epochs fine-tuning.	48
5.1	Example of instance of counting probe LM dataset.	52
5.2	Example of technically not wrong errors in ViLT Language Mod- eling (LM) not Fine-tuned.	53
5.3	Example of errors in ViLT LM fine-tuned.	54
5.4	LLaVA architecture.	55
5.5	LLaVA Example.	57
6.1	Visualization of LoRA’s functioning.	61
6.2	ViLT VE++ catastrophic forgetting confusion matrix.	66

List of Tables

2.1	NLVR2 Comparison Table.	18
3.1	Comparison between different models over the VE dataset.	27
3.2	ViLT checkpoints evaluated on the dev set of VE counting.	30
3.3	Evaluation of ViLT VE counting balanced.	31
3.4	Evaluation of ViLT VE counting augmented.	33
3.5	ViLT evaluated on VE counting dev set with different sum rules. <i>N</i> and <i>S</i> stands, respectively, for Normal and Segmented logits.	36
3.6	ViLT VE counting balanced evaluated on VE counting balanced dev set with different sum rules.	37
4.1	Evaluation of ViLT weights on counting probe VQA dev set.	41
4.2	Number of prediction for each label.	42
4.3	Evaluation of ViLT weights on VQAv2 counting on two different threshold.	44
4.4	Evaluation of ViLT-CNN-v1 with different parameters.	47
4.5	Evaluation of ViLT with different CNNs and upsampling.	49
5.1	Evaluation of ViLT counting probe LM dev set.	53
6.1	Comparison of fine-tuning and LoRA on the E2E NLG, a dataset for natural language generation, on GPT-2.	61
6.2	Comparison of results achieved on ViLT through fine-tuning alone (denoted by -) and results obtained with LoRA (denoted by +).	63

LIST OF TABLES

6.3	Comparison of evaluation metrics obtained on ViLT through various combinations of training methods. The - symbol denotes standard fine-tuning, while the + symbol indicates the use of LoRA. In strings of the form ViLT VExy with $x = -,+$ and $y = -,+$, the symbol x represents how ViLT was trained on VE, and the symbol y represents how ViLT was trained on VE counting.	64
6.4	Comparison of right and wrong classification obtained on ViLT through various combinations of training methods.	65
6.5	Catastrophic forgetting test on ViLT through various training methods.	65

List of Code Snippets

A.1	Necessary libraries to create the datasets.	71
A.2	Standard association.	72
A.3	VQAv2 association.	72
A.4	NLVR2 dataset	73
A.5	VE dataset.	74
A.6	LM dataset.	75
A.7	VQAv2 standard dataset.	76
A.8	VQAv2 dataset with ResNetV2 features.	77
A.9	VQAv2 dataset with SqueezeNet features.	78
A.10	DataLoader to train ViLT in batches.	79
B.1	SAM use example.	81
C.1	Vilt-CNN v1.	83
C.2	Vilt-CNN v2.	84
C.3	Vilt-CNN v3.	86
D.1	LoRA use example.	89
D.2	Printing name of modules.	90

List of Acronyms

AUC-ROC Area Under ROC Curve

BERT Bidirectional Encoder Representations from Transformers

BiT Big Transfer

BPE Byte Pair Encoding

CNN Convolutional Neural Network

CV Computer Vision

FC Fully Connected

FF Feed Forward

FN False Negative

FP False Positive

FPR False Positive Rate

GCC Google Conceptual Captions

IPOT Inexact Proximal Point for Optimal Transport

IT Instruction Tuning

ITM Image Text Matching

LM Language Modeling

LN Layer Normalization

LLM Large Language Model

LIST OF CODE SNIPPETS

- LLaVA** Large Language and Vision Assistant
- LSTM** Long Short-Term Memory
- LoRA** Low Rank Adaptation
- MLM** Masked Language Modeling
- MLP** Multilayer Perceptron
- MVM** Masked Vision Modeling
- MSCOCO** Microsoft COCO
- MSA** Multiheaded Self Attention
- MRP** Masked Region Prediction
- NLP** Natural Language Processing
- NLVR** Natural Language for Visual Reasoning
- NLVR2** Natural Language for Visual Reasoning for Real
- PEFT** Parameters Efficient Fine Tuning
- QA** Question Answering
- ROC** Receiver Operating Characteristic
- RoI** Region of Interest
- SAM** Segment Anything Model
- SNLI** Stanford Natural Language Inference
- SBU** SBU Caption
- SOTA** State Of The Art
- TE** Textual Entailment
- TP** True Positive
- TPR** True Positive Rate

TN True Negative

UNITER UNiversal Image-TExt Representation Learning

VE Visual Entailment

ViLBERT Vision and Language BERT

ViLT Vision and Language Transformer

VLM Visual Language Model

VLP Vision Language Pre-training

VL Vision Language

ViT Vision Transformer

VQA Vision Question Answering

VG Visual Genome



Introduction

1.1 MOTIVATIONS

In the dynamic landscape of modern AI, transformative technologies continue to reshape the way people interact with technology, especially with its novel generative aspect. Technologies such as Large Language Models (LLMs) for text generation or diffusion models for image generation play a central role in this ongoing revolution. As we move towards an era where artificial intelligence seamlessly integrates into our daily lives, a significant transformation is underway. This transformation is driving human-machine interaction towards a harmonious symbiosis that promises to redefine our technological landscape.

This paradigm shift towards increased reliance on machines is already evident in various aspects of work and daily activities. The rapid evolution of AI technologies, such as the remarkable progress of medical diagnostics through image recognition and the nuanced understanding of human language, is fueling a growing dependence on digital tools. As society adapts to this technological surge, exploring new avenues for the continued evolution of AI becomes imperative.

The frontier of artificial intelligence has witnessed a wave of attention and enthusiasm for multimodality on a global scale. Multimodal models, capable of navigating and comprehending various types of information, represent a significant advancement in the field. Among the various types of multimodality, the combination of text and image stands out prominently, immediately raises thoughts of a hypothetical futuristic AI.

1.2. HISTORY OF VISION LANGUAGE

NLP and CV have consistently been focused on grasping the broader context. NLP specializes in comprehending human language, parsing words, sentences, and documents. On the other hand, CV concentrates on interpreting visual information by analyzing pixels in images or videos. Both fields play pivotal roles in extracting valuable insights from their unique modalities. The synergy of these two fields enhances the AI's understanding, enabling individuals to reason within a broader framework. The impressive capabilities of LLMs, which provide access to the AI's "voice", are now also equipped with the perceptual ability of "sight".

This thesis focuses on investigating the collaboration between images and text in multimodal models and its potential applications. The research aims to contribute to the evolving landscape of multimodal AI systems by exploring the dynamic interplay between textual and visual information. The goal is to uncover the practical implications of image-text collaboration, paving the way for machines to easily understand the complicated process of human communication.

1.2 HISTORY OF VISION LANGUAGE

Drawing from the insights of Li et al. [21], this introduction contains the path of evolution of the VL field. Spanning from its origins to contemporary developments, key advancements are outlined, offering a concise overview of VL's transformative journey.

1.2.1 FIRST STEPS

The period between 2014 and 2018 saw the beginning of the collaboration between Vision and Language. During this phase, early methods were developed to address specific tasks by computing global features for both modalities. These methods utilized datasets that emerged for the first time. Some of these datasets, like the VQA dataset [3], persist in their utility for testing and training current models. Notably, this dataset, introduced in 2015, underwent a version update in 2017 [10], with its development team presenting challenges until 2021.

With this background established, a brief introduction and an illustrative example of an early solution to the task are presented.

VISUAL QUESTION ANSWERING

The VQA task consists of an image paired with a question in natural language. Typically, most models approach VQA as a classification task. The original VQA dataset paper [3] introduced an approach employing an Long Short-Term Memory (LSTM) for text component embeddings and the final layer of the VGG-Net as a visual embedder. This method involved aligning the image embedding size with the text counterparts via a Fully Connected (FC) Layer, followed by fusion through a straightforward element-wise multiplication of the two modalities. Ultimately, classification was executed using an Multilayer Perceptron (MLP) with two hidden layers.



Figure 1.1: Example of instances from VQA dataset.

1.2.2 VISION LANGUAGE JOINT REPRESENTATION

Inspired by the good performances of both vision and language models, in 2018 the pre-training and fine-tuning paradigm landed in the VL field, officially starting the Vision Language Pre-training (VLP) era.

One of the most groundbreaking innovations in recent years is the transformer, as developed by Vaswani et al. [40]. With its simple yet highly effective design, this technology has played a pivotal role in creating more powerful language models, thereby completely transforming the field of NLP.

The key feature of the transformer lies in its self-attention mechanism, allowing the model to consider every part of the input sequence. This mechanism assigns individual weights to each word during predictions. By stacking multiple transformers on top of each other, the model establishes meaningful

1.2. HISTORY OF VISION LANGUAGE

connections between tokens, extracting increasingly deeper information as it progresses through the layers. Each layer's output serves as a representation of the original tokens, and the depth of the layer corresponds to the level of abstraction or complexity in the features captured. In essence, the deeper the layer, the higher level are the features captured in the representation of the input sequence.

Models based on transformers leverage pre-training on large corpora¹ to learn general language representations that can be fine-tuned for specific tasks. This contributes to their adaptability and high performance across various NLP applications.

One of the earliest applications of the transformer architecture is Bidirectional Encoder Representations from Transformers (BERT), introduced by Devlin et al. [8]. BERT innovatively introduced a novel task enabling bidirectional communication between each token and others in the sequence. This task, known as Masked Language Modeling (MLM), involves replacing certain tokens with a special [MASK] placeholder, which is then predicted using only the information from the surrounding non-masked tokens. This non-supervised training approach is a key strength of BERT, making it one of the first foundation models².

Inspired by language models, VLP models adopt similar training, with the additional of having two kind of information to distinguish and to make communicate. In this regard, the methods for achieving this goal will be explored in depth, analyzing the main approaches that VLP models have used.

MODALITY EMBEDDING

Text As in NLP field, the analysis of text often involves the essential steps of tokenization and embedding. Tokenization aims to break down text into manageable units for analysis, and in this process, simplicity is maintained by adopting subword tokenization instead of treating each word as an individual token.

This approach, as proposed by Sennrich et al. [33], leverages Byte Pair Encoding (BPE), resulting in a more contained dictionary. This proves advantageous, particularly in handling words that may not frequently appear in the dataset.

¹Organized collection of texts or spoken language data

²Pre-trained neural network models that is used as starting point for many machine learning downstream tasks

After obtaining the dictionary of tokens an embedding is assigned to each of them. Most of the times embeddings comes from pre-trained BERT [8].

Visual Following the trend in text processing, images also require tokenization to assign them embeddings. However, this process is complex, particularly given the typical three-color two-dimensional channel structure. Below, we will describe the primary solution that has been adopted over the years.

1. **CNN features:**

CNNs have been and continue to be instrumental in extracting visual features. Two primary methods can be identified. **Grid features** involve applying a convolutional feature extractor over an image to obtain its characteristics.

On the other hand, **Region features** are extracted using a pre-trained object detector, utilizing detections to identify both the position and “class” of the identified area.

2. **Patch features:**

Simpler than their CNN-derived counterparts, Patch features involve dividing images into patches, followed by a straightforward linear projection. This method, initially introduced by Dosovitskiy et al. in Vision Transformer (ViT) [9], offers a significant efficiency advantage. By bypassing the need for a CNN, computation time is markedly improved.

Building upon this process, additional positional and modality embeddings are added. Positional embeddings are essential for comprehending the spatial interactions between words or visual tokens, while modality embeddings serve to distinguish the type of data being processed.

MODALITY FUSION

In this section of the model, a distinctive departure from the early Visual Language Model (VLM) approaches becomes evident. Here, the model establishes the logical connections between words and visual features, creating a bridge between the two domains. Two main different type of interaction can be distinguished.

1. **Dual stream:**

The two modalities of data are managed separately, with two dedicated encoders used to learn the embeddings for later feeding into the top layers. One advantage of this fusion type is that modality management can be adaptive. The final result can be improved by allowing interaction between the two encoders.

1.2. HISTORY OF VISION LANGUAGE

2. Single stream:

Most VLP models use this modality interaction to learn a joint representation of the input. The sequence of tokens is concatenated and fed into a series of transformers to compute the representation, learning single-modality and cross-modality representations without complicating the architecture with bridges between two encoders.

TRAINING

As happened in NLP, some tasks became the standard choice for the pre-training of VLMs. More specifically, we can identify three losses that better achieve the goal of achieving a joint representation between textual and visual data.

Image Text Matching (ITM) The goal of the ITM task is to determine whether an image and a text pair are matched. One way to frame ITM involves treating it as a binary categorization task. Past studies have employed a sigmoid function on the [CLS] token output to predict if the input image and text are a match.

$$\mathcal{L}_{\text{ITM}} = -E_{(W,V) \sim \mathcal{D}} \log \Pr(y|W, V) \quad (1.1)$$

with W and V representing a sequence of text tokens and a sequence of visual tokens, y can be 0 or 1 depending if the pair is matched or not.

Masked Language Modeling (MLM) The model is encouraged to learn the implicit relationship between linguistic tokens and visual content by using MLM. The objective is to use the visual contents and known language tokens to reconstruct the masked language tokens.

$$\mathcal{L}_{\text{MLM}} = -E_{(W,V) \sim \mathcal{D}} \log \Pr(w_i|W_{\setminus i}, V) \quad (1.2)$$

with $W_{\setminus i}$ representing the sentence without the i -th word. Note that subword tokenisation could mask individual subword tokens, which is conceptually wrong: a subword can be reconstructed from the other subwords, which together form a complete word. Masked tokens represent a whole word, achieving the goal of creating connection between modalities.

Masked Vision Modeling (MVM) Following along the lines of MLM, acMVM aims to reconstruct masked visual regions. There are several way to perform MVM but the most straightforward is Masked Region Prediction (MRP), whose goal is to predict the features of a region extracted by an object detector.

$$\mathcal{L}_{\text{MRP}} = \sum_{i=1}^M \left\| h_{\theta}^i - r(v_m^i) \right\|_2^2 \quad (1.3)$$

where \mathbf{v}_m represents the masked regions and h_{θ}^i is the prediction of the i -th region after a Region of Interest (RoI) pooling. Reconstructing a missing word demands a rich understanding, while reconstructing a visual region requires only knowledge of its visual neighbors, without the necessity for cross-modality.

Although methods for MVM exist and are well-established, research indicates that the MVM task yields marginal additional improvements to VLP models. Other studies confirmed that the main focus falls on the textual part instead of exploiting cross-modality.

VISION AND LANGUAGE BERT (ViLBERT)

The ViLBERT model, proposed by Lu et al. [26], much like many other VLP models, draws heavy inspiration from BERT [8]. Positioned within the dual-stream category, its primary structure comprises two BERT-like transformer streams designed to handle text and image. The authors introduce some modifications to facilitate modality interaction.

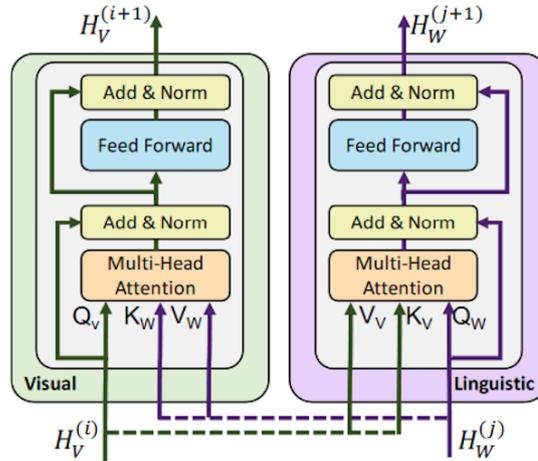


Figure 1.2: Co-attentional transformer block.

1.2. HISTORY OF VISION LANGUAGE

The Co-attentional Transformer, depicted in Figure 1.2, is introduced, the idea behind it is to pass the Key and the Value matrices of each modality to the other one, creating image-conditioned text features and text-conditioned visual features.

To obtain the embeddings that will later feed the co-attentional streams text and image undergo an initial phase of information extraction. Image features are obtained using an pre-trained object detector, positional embedding is then added exploiting the bounding box and the segmentation. A special [IMG] token marks the start of the visual region sequence.

Text is treated as in a standard BERT, but before entering the co-attentional streams it is first passed through a series of classic transformers to match the amount of information that the visual features already gathered.

The pre-training process uses Masked Multimodal Modelling, i.e. MLM and MVM applied simultaneously, and ITM task.

UNIVERSAL IMAGE-TEXT REPRESENTATION LEARNING (UNITER)

UNITER, by Chen et al. [5], followed the route of single stream models. The architecture is straightforward. After computing the embedding for the image and text, they are concatenated and fed into a series of standard transformers.

CNN extracts visual features in the form of pooled RoI for each region. Spatial information is also encoded in a vector. These two are then passed through their own fully connected layers. After summing the two outputs, they are passed into a Layer Normalization (LN) layer to obtain the final visual embedding. For the text embedding, BERT is followed, the sentence is tokenized and then summed with the position embedding to be passed into a LN layer.

For the pre-training MLM, MVM, ITM and Word-Region Alignment are used. In the latter approach, the model compares each visual token with each textual token to identify the most suitable pairing

1.2.3 FUTURE OF FIELD

In contrast to the field of LLM, which leverages extensive data from diverse sources such as the internet, books, and articles, VLP models rely on well-labeled data to establish a robust cross-modality alignment. The effectiveness of an LLM is closely tied to the quantity of data encountered during the pre-training phase. Conceptual Captions is the largest dataset used for VLM pre-

training, comprising three million image-text pairs. However, datasets of this size are still small compared to LLM pre-training datasets. To achieve enhanced generalization, it is necessary to take inspiration from these insights.

CLIP

CLIP, as introduced by Radford et al. [31], marked a breakthrough by successfully leveraging weakly-labeled data gathered from the internet, 400 million instances in total. In addition to the classic transformer embedder for the text, CLIP utilised a ViT for its visual component.

Its key innovation and strength, though, lie in the application of a contrastive objective: when presented with a batch of N (text, image) pairs, CLIP trains to predict the actual positive matches from the $N \times N$ possibilities. This is achieved by developing a multi-modal embedding space, wherein the image and text encoders are trained to maximize the cosine similarity for the N true pairings and minimize it for the remaining $N^2 - N$ pairings.

After pre-training, CLIP demonstrates optimal capacity in ITM. However, the model is only specialised in this specific task.

1.3 ViLT

Among the multitude of models, Vision and Language Transformer (ViLT), developed by Kim et al. [17], is a single stream VLP model that achieves competitive results with light computational resources. Its simplicity is found in the architecture, ViLT completely removes the CNN from its pipeline, achieving a computational time of one tenth with respect to a model of its size.

1.3.1 ViLT ARCHITECTURE

As illustrated in Figure 1.3, the categorization of VLP models by the author of ViLT involves four classes. The first category represents the early methods used to solve the first problems in the field. CLIP [31] can be placed in the second category, using two robust transformer embedders, but the interaction phase remains relatively shallow. Both ViLBERT [26] and UNITER [5] fall into the third category, employing substantial visual embedders along with a substantial interaction block.

1.3. ViLT

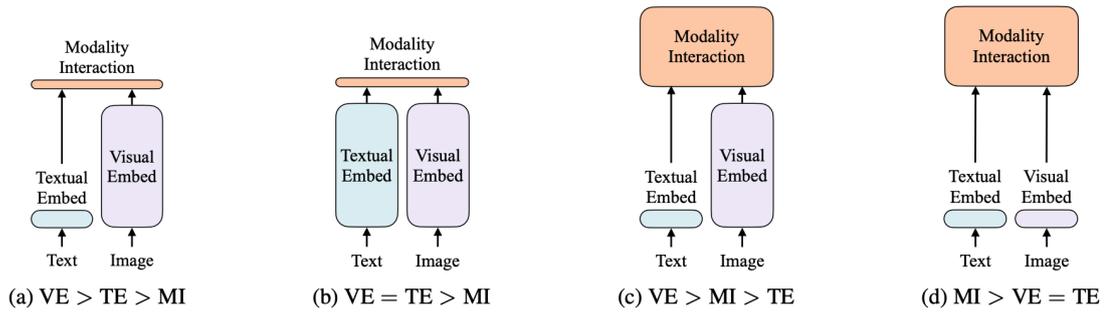


Figure 1.3: Four categories of VLP models.

Contrastingly, ViLT is distinct as the first model to focus entirely on the modality interaction phase, belonging to the fourth category, simplifying modality embeddings as much as possible.

In the textual domain, the model follows the BERT [8] approach, where sentences are tokenized, and each sub-token receives its embedding from a lookup table. On the visual side, embeddings are derived through patch embeddings, utilizing a linear projection that simplifies the computation of embeddings to the text level.

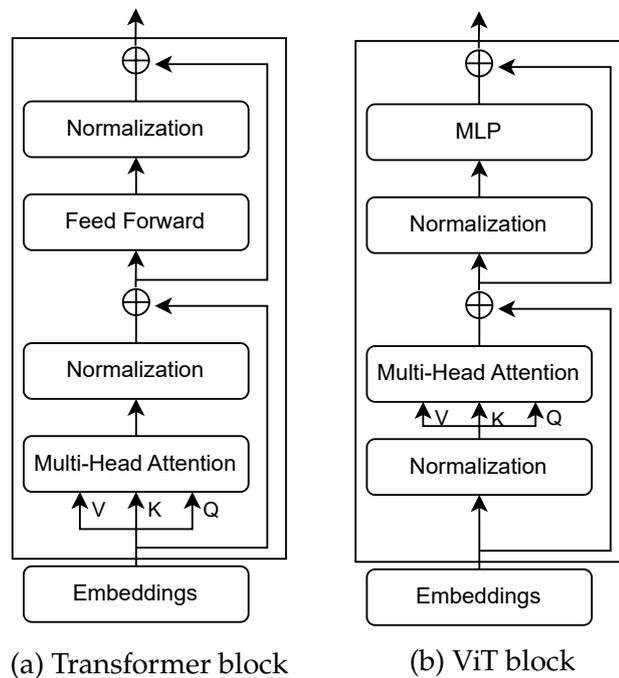


Figure 1.4: Comparing Transformer and ViT.

A main difference from classical VLP is the use of pre-trained ViT [9], patch size of 32, instead of BERT for the interaction module. Below, model equations

are reported.

$$\bar{t} = [t_{\text{class}}; t_1 T; \dots; t_L T] + T^{\text{pos}} \quad (1.4)$$

$$\bar{v} = [v_{\text{class}}; v_1 V; \dots; v_N V] + V^{\text{pos}} \quad (1.5)$$

$$z^0 = [\bar{t} + t^{\text{type}}, \bar{v} + v^{\text{type}}] \quad (1.6)$$

$$\hat{z}^d = \text{MSA} \left(\text{LN} \left(z^{d-1} \right) \right) + z^{d-1}, \quad d = 1, \dots, D \quad (1.7)$$

$$z^d = \text{MLP} \left(\text{LN} \left(\hat{z}^d \right) \right) + \hat{z}^d, \quad d = 1, \dots, D \quad (1.8)$$

$$p = \tanh \left(z_0^D W_{\text{pool}} \right) \quad (1.9)$$

As can be seen in figure 1.4, a main difference of ViT from standard transformer is the position of the normalization, that comes before the Multiheaded Self Attention (MSA) and the Feed Forward (FF)/Multilayer Perceptron (MLP).

1.3.2 PRE-TRAINING

ViLT, like many other models, undergoes pre-training to establish the link between textual and visual features. The two tasks chosen are the ITM, which, as will be seen in a moment, comes with a twist, and the MLM. The dataset used are Microsoft COCO (MSCOCO) [23], Visual Genome (VG) [20], SBU Caption (SBU) [28] and Google Conceptual Captions (GCC) [35].

IMAGE TEXT MATCHING

For each text-image pair, the associated image is replaced with a random image from the dataset with a probability of 50%. A simple single linear layer is used to compute the binary logits³ of the instance. The negative log likelihood is computed as the ITM loss.

Inspired by UNITER [5], word region alignment is taken into consideration. The Wasserstein distance is computed using Inexact Proximal Point for Optimal Transport (IPOT) [43] and later multiplied by 0.1 before being added to the ITM loss.

³Vector of raw (unnormalized) predictions

1.3. ViLT

MASKED LANGUAGE MODELING

The objective is to predict masked text tokens t_{masked} from vector $z_{\text{masked}}^D|t$. To avoid predicting a token based on its neighbours alone, whole word masking is employed along with visual information. Words are masked with a probability of 15%.

Based on $z_{\text{masked}}^D|t$, the two-layer MLP is used to obtain logits over the vocabulary for each token. The loss function for MLM is calculated by taking the negative log-likelihood of the masked tokens.

1.3.3 IMPLEMENTATION

The author of ViLT has made available the original model in their Github repository⁴. Although the instructions to repeat the different experiments are clear, for this thesis the implementation of ViLT made by Hugging Face is chosen⁵.

The decision was made with available resources in mind; not having a dedicated machine, but using a shared cluster⁶, it was easier and simpler to rely on Hugging Face’s Transformers library. In addition, as part of the Hugging Face ecosystem, the Transformers library makes it easy to interact with other models or methods and to find help.

When working with batches of data, models require inputs of the same size. It is important to note that sentences and images may not always have equal lengths, particularly after being tokenized. In the NLP field, to overcome this, [PAD] tokens are used. They add length to the token to match the maximum token sequence length in the batch (or a fixed size). An attention mask is used to determine which tokens to consider. Following this approach, a similar technique is applied to images by padding them to the maximum size in the batch instead of resizing them to achieve uniformity. They are then associated with an attention pixel mask to indicate which part of the image needs to be considered.

Each experiment followed the original setup. The AdamW optimizer [25] was used with a base learning rate of 10^{-4} and a weight decay of 10^{-2} . To

⁴<https://github.com/dandelin/ViLT>

⁵https://huggingface.co/docs/transformers/model_doc/vilt

⁶Blade computing cluster: <https://www.dei.unipd.it/bladecluster>

avoid early overfitting, the learning rate underwent a warming-up phase during the initial 10% of total training steps, followed by a linear decay to zero for the remaining training duration. For all downstream tasks tested, the training duration was set to 10 epochs, with some experiments varying this.

The original implementation employs RandAug, a preprocessing technique introduced by Cubuk et al. [7], in both training stages (pre-training and fine-tuning). According to the author, image preprocessing slightly enhances generalization during inference. However, in this thesis, image preprocessing has been omitted from training to simplify computation and maintain a straightforward association between text and image. Another deviation from the original work lies in the batch sizes used, as limited computing resources force the use of smaller dimensions than those given in the paper.

2

The Counting Problem

Using the particular NLVR2 dataset, the first experiment on ViLT is performed in this chapter. The feasibility of using the model is defined by fine-tuning it on the same task as in the original work, with similar results. Additionally, the task provides an important route to follow, as error analysis reveals a common trend of errors, particularly in the counting task. This recurring challenge throughout the work provides a useful opportunity to test various techniques aimed at improving the model’s performance.

2.1 NLVR2

Before embarking on ViLT experiments, the initial step was to assess the feasibility of conducting tests within a different environment. In this dissertation, the ViLT architecture was used, maintaining consistency with the original work, but implemented in a distinct library. While the theoretical expectation is for similar functionality, the assumption remains that the correctness of the test may not be verified in this different environment. To test this hypothesis, it is necessary to replicate the experiments.

The initial experiment selected focused on the Natural Language for Visual Reasoning for Real (NLVR2) dataset, chosen for its particular characteristics, as described in subsequent sections. The task involves evaluating the correctness of a statements associated with a pair of images. Reasoning over a pair of related images introduces an additional layer of complexity to the task.

2.1. NLVR2

2.1.1 ORIGIN OF DATASET

The VL field has witnessed the establishment of foundational datasets crucial for evaluating models and algorithms. Among the initial datasets proposed, the NLVR dataset, curated by Suhr et al. [37], stands out. While most other datasets follow a standard format comprising text, image, and correct output, NLVR stands out due to its unique setting.

Each instance within the NLVR dataset features a single image divided into three distinct boxes, accompanied by an associated sentence. Notably, the images in NLVR are synthetically generated, constructed through the introduction of geometric figures with various colors and positions, a process facilitated by computer algorithms. The accompanying sentences, crucial for task comprehension, are provided through human annotators, creating a dynamic interplay between automated image generation and human linguistic input.

The primary objective of the task set by NLVR is to predict the truth or falsity of a given statement based on the corresponding images. The inherent division into boxes necessitates models to engage in intricate reasoning, both within individual boxes and across them. This distinctive structure deviates from the conventional task involving a single image and the associated text, which increase the complexity of the model's cognitive processes.

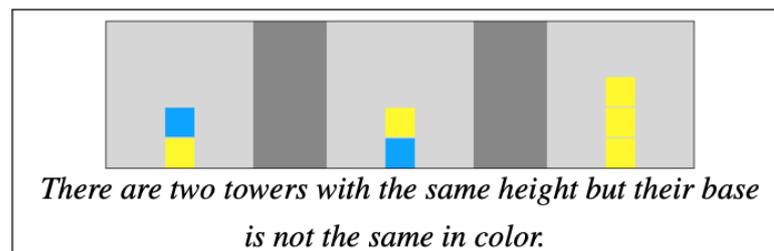


Figure 2.1: Example of instance of NLVR dataset.

While the NLVR dataset's synthetic images serve as a valuable resource, their inherent limitation lies in constraining the language used to describe them. Using a limited dictionary in the related sentences, the strict necessity to describe synthetic images may inadvertently lead to a degradation in models' language capability. This problem, combined with the growing need for systems capable of intricate reasoning in complex environments, has driven advances in dataset design.

In response to these considerations, the NLVR2 dataset, as introduced by

Suhr et al. [36], represents a significant upgrade. One improvement lies in the use of real-world images, transforming the task while maintaining its core objective. Unlike NLVR, each statement in NLVR2 is now associated with two separate real images, introducing an increased level of complexity.

The use of real-world images in NLVR2 not only enhances reasoning capabilities but also substantially enriches the textual vocabulary. Compared to NLVR, NLVR2 exhibits a significant increase in vocabulary, featuring 7457 word types as opposed to 262. These improvements highlight the dataset’s progress in presenting a more diverse and sophisticated set of visual stimuli while providing a broader linguistic context, contributing to a more comprehensive evaluation of models.

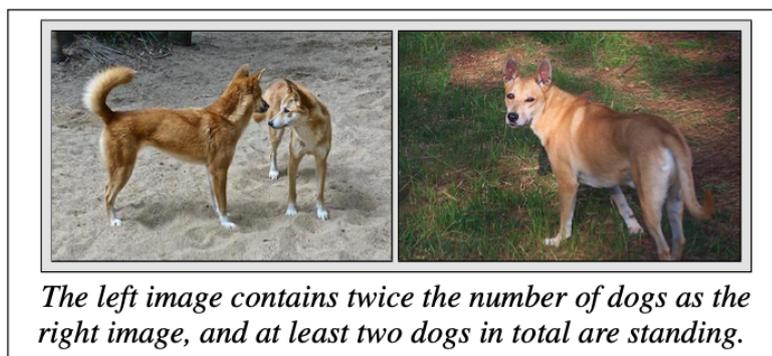


Figure 2.2: Example of instance of NLVR2 dataset.

2.1.2 EXPERIMENTS

As stated in the introduction, the NLVR2 dataset was specifically chosen for its unique task, requiring models to utilize their full reasoning capabilities. The task of understanding two images together is more challenging than understanding a single image alone. Some models may struggle with this task due to their design limitations.

To address this, ViLT introduces a reformulation of the problem by creating two pairs: (question, image1) and (question, image2). Each pair is processed by the model, and the pooled output of both pairs is concatenated and subsequently passed to the prediction head.

The NLVR2 dataset was used to fine-tune the model and evaluate the replicability of the results. As detailed in Section 1.3.3, the implementation in this thesis deviates from the original work. Beyond the reported differences, for

2.1. NLVR2

this task, all images are resized to a fixed dimension due to an issue with the padding system. For instance, future task will not suffer from this problem. The cross-entropy loss is used.

With all of these considerations in mind, evaluation of the original weights and thesis weights are evaluated on different metrics

- Accuracy: measures the overall correctness of the model by calculating the ratio of correctly predicted instances to the total instances.
- True Positive Rate (TPR): also known as sensitivity, measures the proportion of correctly identified actual positive instances by the model.
- False Positive Rate (FPR): this metric measures the proportion of actual negative instances that are incorrectly classified as positive by the model.
- Area Under ROC Curve (AUC-ROC): evaluates a model’s ability to distinguish between classes by measuring the area under the Receiver Operating Characteristic (ROC) curve. The ROC curve plots the trade-off between TPR and FPR at various thresholds.

Evaluations are reported below:

Evaluation Metric	Original Work	Thesis
Accuracy (%)	74.81	73.25
True Positive Rate (%)	74.93	71.38
False Positive Rate (%)	25.32	24.83
AUC-ROC	0.83	0.81

Table 2.1: NLVR2 Comparison Table.

While the fine-tuning process in this thesis deviates somewhat from the methodology outlined in the paper, the results, as illustrated in Table 2.1, demonstrate comparable performance. Notably, the AUC-ROC value remains almost unchanged. An AUC-ROC above 0.8 is generally considered indicative of a good classifier, with 1 representing a perfect classifier and 0.5 suggesting randomness. This consistency validates the viability of the adapted setup.

It is important to clarify that the primary objective of this thesis is not to push the boundaries of the current State Of The Art (SOTA). Instead, the focus is on improving the obtained results in terms of absolute values, aiming for a meaningful improvement within the established framework.

2.1.3 ERRORS

ViLT demonstrates a solid performance on the NLVR2 task, showcasing that its lightweight structure can be leveraged with minimal drawbacks. Post fine-tuning, the model can handle diverse scenarios, ranging from instances involving multiple subjects, as illustrated in Figure 2.3a, to situations requiring nuanced reasoning about two images either jointly or separately, as shown in Figure 2.3b. ViLT performs consistently and effectively across a range of non-trivial challenges.



Figure 2.3: Example of correct predictions.

While the achieved results by this model are commendable given its compact size, it is important to acknowledge its imperfections. Apart from cases where the model struggles with unfamiliar words, there are significant errors that offer valuable insights. Upon analysis, recurring patterns in these errors become apparent and will be discussed in the following sections.

OBJECT'S ATTRIBUTES

ViLT exhibits challenges when confronted with intricate and detailed object descriptions, particularly when referring to small details in the photo. This difficulty is illustrated in Figure 2.4, where the model struggles to comprehend the specified description. This issue may be related to the model's pre-training phase, which does not ensure the necessary text-visual connections, or to the ViT architecture, which may not extract appropriate features.

2.1. NLVR2



Associated Text: The bananas do not have a produce sticker on them.
Predicted: False

(a) Confusion.



Associated Text: A lock has a circle with in a circle on the front.
Predicted: False

(b) Not understanding.

Figure 2.4: Example of wrong predictions with specific details description.

COMPLEX DESCRIPTION

Errors may arise in images with lengthy descriptions, particularly when exceeding the average length in the dataset. The attention mechanism in the constrained architecture may have difficulty processing extensive details. Another source of potential confusion could arise from the use of uncommon terms or compound words that have been rarely encountered during the training phases.



Associated Text: The left image features an older chimp with two younger pink-faced chimps, and the right image features two similar-sized apes posed sitting close together.
Predicted: False

Figure 2.5: Example of wrong prediction with complex description.

COUNTING ERROR

An issue that will be consistently addressed throughout the thesis is inaccurate counting. When faced with a sentence involving a numeric description, the model does not behave properly and gets it wrong. This issue can occur with both simple and complex images, as seen in Figure 2.6.



Figure 2.6: Example of wrong predictions in counting descriptions.

2.2 COUNTING IN VLM

As documented in the literature, VLMs, including ViLT, often encounter difficulties in compositional reasoning between objects, as emphasized by Thrush et al. [38]. The aforementioned errors are frequently linked to this challenge. Specifically, when tasked with providing detailed descriptions or discussing specific aspects of a subject, ViLT may struggle to establish the necessary relationships. The counting task, in particular, is significantly affected by this issue, requiring robust connections between identified subjects and the numbers described in the sentence.

Paiss et al. [29] addressed the counting problem with CLIP, offering insights into potential reasons for this issue. While the methods discussed in their work are not directly applicable to this thesis, as CLIP employs a distinct training approach, the authors' analysis provides valuable context for understanding the existence of the counting problem.

Several challenges contribute to the counting problem, primarily due to the nature of the training datasets. Instances that explicitly specify the precise number of occurrences are rare, with descriptors like "many" or "a few" commonly used when referring to quantities. Furthermore, correlating the number of objects in the image with the number mentioned in the text may not be relevant for the model. In essence, alternative features may offer more informative results to better accomplish the specific counting task.

The fact that these challenges continue to exist in CLIP, a model that was trained on a significantly larger dataset, highlights the seriousness of the problem. Even with access to an extensive training corpus, the intrinsic difficulties

2.2. COUNTING IN VLM

in the counting task persist. Furthermore, the absence of instances with explicit counts in the well-labelled datasets used for training ViLT amplify the significance of the challenge.

2.2.1 OTHER APPROACHES

Counting objects in images is not a new concept in CV. Starting with classic CV techniques such as edge detection and morphological operations, and transitioning to deep learning, CNNs became the preferred method. Trained on datasets with object-wise labels, CNNs can learn to extract features associated with the object and effectively count it.

Recently, transformer-based architectures for counting have emerged. These models cannot be defined as VLMs due to their training; they are trained only for that specific task, skipping the creation of a broader “real world” knowledge. However, they do incorporate components from previously encountered models. The topology that is most similar, but not necessarily the most efficient, is CounTX, by Amini-Naieni et al. [2].

The CounTX architecture leverages the visual and textual embeddings of CLIP to create modalities representation. Similar to the co-attentional block of ViLBERT [26], this approach employs two transformer decoder layers. The visual part serves as the Query, while the textual part functions as the Key and Value. By utilizing the cross-attention mechanism, the model identifies similarities between patches and tokens, creating a density map of the addressed object through a decoder. This density map is then used to predict the number of objects.

While the architecture shares similarities with VLMs, as previously noted, it cannot be strictly classified as one. Despite the training approach establishing connections, it does not prioritize achieving a broader sense of general knowledge.

2.2.2 COUNTING PROBE DATASET

To evaluate the counting capabilities of VLP models, Parcalabescu et al. [30] introduced the Counting Probe dataset. The dataset comprises images and corresponding answers sourced from Visual7W, a Visual Question Answering (VQA) dataset that incorporates instances of counting. The data is presented in two forms:

1. Standard form: Reflecting the original Visual7W task, each image is associated with a question, a correct answer, and foil options.
2. Declarative form: Instances are transformed into sentences describing the image, rather than being presented as a question with associated answers.

The authors conducted several experiments using various VLP models to assess the effectiveness of pre-training and acquired knowledge, both in a zero-shot and fine-tuned manner. Building on this work, ViLT will be subjected to testing across various tasks to identify potential challenges and discrepancies arising from different training methods.

3

Visual Entailment

While NLVR2 served as a valuable dataset for unraveling the mechanisms of ViLT and gaining insights into the dynamics of VL tasks, its limitations lie not in complexity but in the number of instances. Recognizing the potential for a denser dataset to facilitate stronger knowledge and between words and images, the Visual Entailment (VE) task, introduced by Xie et al. [42], emerged as a valid option.

Distinguished by its substantial size compared to NLVR2 and novel to ViLT, as it was not included in the original work, VE offers an opportune opportunity for additional exploration and experimentation. This chapter discusses the integration of VE, the creation of a new task, and the exploration of potential improvements through specific techniques, from adding more training samples to trying to extract more information from images.

3.1 DATASET AND EXPERIMENTS

The VE dataset was conceived out of the necessity for a well-calibrated tool in the VL field, featuring a correct distribution of answers. As outlined in the dataset’s paper, the referenced VQA dataset exhibits distribution biases caused by common responses to questions with a particular format. For example, questions beginning with “Do you see a...?” frequently result in a “Yes” as a correct response, constituting 87% of correct answers during training. These inherent limitations across various datasets prompted the creation of VE.

3.1.1 DESCRIPTION

The VE task demands a high level of reasoning in real-world settings. Attention to detail is crucial, even for the smallest aspects. Like many VL tasks it draws inspiration from its NLP counterpart, Textual Entailment (TE). In the VE task, a visual premise, denoted as P_{image} , and a hypothesis in natural language form, denoted as H_{text} , are provided. The objective is to predict whether the text can be logically inferred from the accompanying image. The output can be assigned one of three labels:

- *Entailment*: there is evidence that H_{text} can be inferred as true by examining P_{image} .
- *Neutral*: insufficient information is available to draw a conclusion about H_{text} from P_{image} .
- *Contradiction*: there is evidence that H_{text} can be inferred as false by examining P_{image} .

The addition of the “neutral” label adds a higher level of complexity to the task. In this way, it differs from a standard binary classification task, such as the previous NLVR2.

The VE dataset is a combination of two influential sources: the Stanford Natural Language Inference (SNLI) dataset, presented by Bowman et al. [4], which is a widely-used TE dataset based on Flickr30k, and the Flickr30k dataset, proposed by Young et al. [44], which is frequently used for image captioning. Coming from the same root, this union is natural and provides a wide range of real-world images. Each image is paired with different hypotheses, generating different samples with different labels.

A noteworthy consideration arises from the fact that ViLT’s entire pre-training process involved concatenating the textual part first and the visual part second. This sequence oppose to the order specified by the VE task, where the hypothesis is provided visually first, followed by the text. Efforts to modify the model to respect the format requested by the VE task were unsuccessful, as ViLT’s weights are now tuned specifically for the original concatenation sequence.

While the application of the VE task in this context may be “philosophically” incongruent with ViLT’s pre-training, it remains technically feasible. The task, in essence, could be described as a form of “Visual Inference” though for the sake of simplicity, this definition will remain the same.

3.1.2 RESULTS

After fine-tuning ViLT using Cross Entropy Loss, it did not reach the SOTA level in the VE task, aligning with expectations, it is crucial to emphasize that the primary focus of this dissertation does not revolve around achieving the highest possible results in VE. The obtained ViLT checkpoint in the VE task serves as a starting point for subsequent tasks and future comparisons, as will be discussed later in the thesis.

Model	Accuracy (%)
EVE	70.81
ViLT	74.77
OFA	91.00

Table 3.1: Comparison between different models over the VE dataset.

As presented in Table 3.1, ViLT is compared with EVE and OFA. EVE, the initial model proposed for the VE task, originates from the original dataset paper [42] and is explicitly designed for this task. On the other hand, OFA, proposed by Wang et al. [41], currently the SOTA for numerous VL tasks, surpasses ViLT in both size and pre-training data volume. In addition to using cross-modal datasets, OFA also leverages unimodal datasets to gain rich knowledge and achieve robust performance.

Confusion Matrix

True Label	Contradiction	4647	880	412
	Neutral	672	4342	946
	Entailment	398	1197	4364
		Contradiction	Neutral Predicted Label	Entailment

Figure 3.1: Confusion Matrix of evaluation on VE dev dataset.

3.1. DATASET AND EXPERIMENTS

In the context of VE as a multi-label classification task, the use of the confusion matrix proves to be a useful tool for evaluation. Illustrated in Figure 3.1, ViLT demonstrates consistent performance on the VE Dev dataset following the standard fine-tuning phase. The diagonal cells with numerically higher values indicate correct classifications, which aligns with expectations for a non-random classifier. However, there are additional cells with suboptimal values, particularly those directly adjacent to the correct ones, suggesting potential for improvement.

3.1.3 ERRORS

ViLT demonstrates an aptitude for reasoning in less straightforward situations, where predicting the answer goes beyond mere recognition of individual elements. It involves understanding the scene and connecting the acquired information. While errors are inevitably present, some are attributed to the complexity of the scene, as illustrated in Figure 3.2. However, errors with a common underlying cause could not be identified.



Associated Text: The woman is looking up at an airplane.

Correct label: Neutral

Predicted: Contradiction

Figure 3.2: VE error example.

As VE is not designed as a true or false dataset, instances involving counting are not present in a format similar to that found in NLVR2. The main focus of VE is to generate descriptions based on visual content. While instances involving counting could theoretically exist, they were obviously not the primary focus of the dataset author.

3.2 COUNTING PROBE VE

The absence of counting instances in the VE dataset is likely due to the nature of the task, where the “neutral” label may not be as meaningful for counting scenarios. The number of objects is often definitive and less open to debate. However, the reasoning capabilities acquired by ViLT during the training phase on this dataset enable it to handle more complex associations. This capability could certainly be further exploited.

To leverage counting probe, a counting dataset in the style of VE is created. This is achieved by utilizing its declarative sentences. As recommended previously, the “neutral” label is excluded from consideration, as being “neutral” is not applicable in a counting context. The current dataset content designates foil sentences as “contradiction”, while correct ones are labeled as “entailment”. Numbers inside the phrases are switched from digits to words to keep a more consistent flow with the pre-training datasets.



Associated Text: Nine people can be seen.
Correct label: Entailment

Figure 3.3: Example of instance of VE dataset.

This novel dataset is used in two ways:

- To test how the starting weights trained on VE behave on the counting dataset. This serves to understand the implicit logic acquired during the first fine-tune, resembling a zero-shot¹ setting.
- To fine-tune ViLT on the counting dataset. This assesses the adaptability

¹In Deep Learning, “zero-shot” refers to a model’s capacity to handle tasks or inputs not seen during training, highlighting its adaptability and generalization prowess.

3.2. COUNTING PROBE VE

of the model, considering the constraint on the “neutral” label, and tests how much information is lost during this second round of training.

3.2.1 RESULTS

The following table displays all the achieved results. The VE checkpoint, specifically ViLT not fine-tuned on the counting probe VE, will be referred to as ViLT VE. The fine-tuned checkpoint will be denoted as ViLT VE counting.

Checkpoint	Accuracy (%)	TPR (%)	FPR (%)	AUC-ROC
ViLT VE	37.65	71.87	62.10	0.58
ViLT VE Counting	77.50	45.51	11.76	0.76

Table 3.2: ViLT checkpoints evaluated on the dev set of VE counting.

As observed from Table 3.2, without a proper fine-tuning, It seems that ViLT is not making full use of its prior training. Many errors arise due to a lack of consideration for the exact number of objects, which is not the desired outcome. Conversely, after fine-tuning, notably improved results are achieved.

The AUC-ROC score is reasonable, but signs of issues arise from TPR, which is notably low. This is probably due to imbalances in the distribution of the dataset. Specifically, for each “entailment” instance, there are three “contradiction” instances. Given the well-known challenges that ViLT, and VLMs in general, face in counting, the model may tend to choose the “contradiction” answer as an easier route to achieve high accuracy.

One of the challenges encountered when fine-tuning a neural network is known as “catastrophic forgetting”. This phenomenon describes the model’s tendency to unintentionally discard previously acquired knowledge when exposed to new data. As the model adapts itself to new tasks, there exists a risk of overwriting existing information, resulting in a degradation of performance on tasks previously mastered.

When testing ViLT VE counting on the development set of VE, an analysis of the results indicates the occurrence of catastrophic forgetting. The model achieves an accuracy of 64.78%, providing clear evidence of its occurrence. This outcome was expected, primarily due to the challenge posed by the “neutral” label. The absence of this label in the counting dataset resulted in confusion

Confusion Matrix

	Contradiction	Neutral	Entailment
True Label Contradiction	5132	200	607
True Label Neutral	2086	2179	1695
True Label Entailment	1411	290	4258
	Contradiction	Neutral Predicted Label	Entailment

Figure 3.4: ViLT VE counting catastrophic forgetting confusion matrix.

within the model regarding its interpretation and application, as evident in the middle row of Table 3.4.

To address the class imbalance in the dataset, modifications have been implemented to achieve a more equitable distribution of classes across different categories.

3.2.2 BALANCED DATASET

A straightforward approach involves retaining only a single foil instance for each image. This adjustment aims to achieve a more balanced distribution between “contradiction” and “entailment”, resulting in a model that is evenly exposed to both classes. However, it’s important to note that this will nearly halve the number of instances during the training phase.

Following the training of the model on this balanced dataset, evaluations were conducted on both a balanced development set and the standard, unbalanced counterpart.

Dataset	Accuracy (%)	TPR (%)	FPR (%)	ROC-AUC
Balanced dev set	63.24	61.75	35.26	0.69
Standard dev set	67.27	61.75	30.87	0.72

Table 3.3: Evaluation of ViLT VE counting balanced.

3.2. COUNTING PROBE VE

Table 3.3, shows a decrease in Accuracy and AUC-ROC metrics compared to previous results. However, this is due to trade-offs involving TPR and FPR, which now have better values. This indicates a more accurate differentiation between the two labels learned.

With the data now correctly balanced, these results may accurately reflect ViLT’s capabilities on the task. The metrics now accurately represent how well the model reasons on the counting task without the influence of unbalanced data, as distribution biases have been eliminated.

3.2.3 AUGMENTED DATASET

An alternative strategy to achieve a balanced dataset involves adding positive instances to the existing data. This method is intricate and can be implemented through various means. The selected method involves using a prompt-like approach: taking the correct sentence and concatenating it with a foil one, formulated as “Not *foil* but *correct*”. This method generates new instances of “entailment”, achieving the goal of a more balanced representation in the dataset.



Associated Text: Not there are two people but there are three people.

Correct label: Entailment

Figure 3.5: Example of augmented VE counting instance. The sentence is deliberately not in fluent English; refer to the explanation below for the motivation behind this choice.

As noticeable from Figure 3.5, these newly introduced instances may not strictly follow the conventional norms of communication. As previously mentioned, the approach was centered around utilizing a basic prompt format, not aiming at sentences with fluid and natural language. This experimental effort

primarily served as a test to assess the viability of such an augmentation. The subsequent results are detailed below.

Dataset	Accuracy (%)	TPR (%)	FPR (%)	ROC- AUC
Augmented dev set	81.87	71.60	11.23	0.87
More augmented dev set	69.78	46.38	12.59	0.76

Table 3.4: Evaluation of ViLT VE counting augmented.

ViLT was fine-tuning on two variations of the augmented dataset. The first version involved the addition of a single positive instance to each set following the previously mentioned method. While this approach showed promising results, improving every computed statistic, a detailed analysis of the outputs revealed that the model had essentially learned to classify the new “long” sentence as consistently positive without developing any form of reasoning. In essence, this dataset proved ineffective, as it ended up further perplexing the model instead of improving its capabilities.

In the final attempt to take advantage of the augmented dataset, a more expansive augmentation strategy was employed, involving the addition of more sentences. Based on the available foils per image, up to three new positive instances and up to two new negative instances were created. The negative instances were generated by combining two foils, ensuring that the “long” sentences did not exclusively classify under the “entailment” class. Despite the effort, as indicated in Table 3.4, all metrics exhibited a deterioration compared to the previous state. Both augmentation methods were considered inappropriate and rejected.

3.3 SEGMENTATION

Evidently, expanding the dataset did not yield the desired results. It appears that addressing the counting problem in ViLT requires more than just augmenting the dataset. As will be discussed in subsequent analyses, the impact of providing additional data is likely more pronounced in the pre-training phase, where the foundational connections between textual and visual features are established. Motivated by this consideration, the focus shifted towards providing more information rather than simply increasing the amount of data.

3.3. SEGMENTATION

A noteworthy advantage of VLMs employing region visual features is the involvement of a CNN trained for object detection. Its extracted features are employed for the positional embedding of each region, which definitely improves the model’s capability to extract specific features in contrast to the global features extracted by a ViT. Taking inspiration from this, exploring ways to make it easier to individuate objects in an image could potentially help with the counting task.

3.3.1 IMPROVING WITH ZERO-SHOT SEGMENTATORS

In deep learning, semantic segmentation is the process of classifying each pixel in an image into predetermined categories, providing a comprehensive understanding of the scene. Unlike object detection, which outlines bounding boxes around objects, semantic segmentation assigns a unique label to each pixel, enabling more detailed and precise image analysis. In the work presented by Nanni et al. [27], the efficiency of well established semantic segmentator is improved with the use of a zero-shot one. The paper introduces two different zero-shot models for experimentation, and in this thesis, Segment Anything Model (SAM), proposed by Kirillov et al. [18], is selected as the chosen model.

Skipping the architectural details, SAM is categorized within the class of promptable segmentator models. Drawing inspiration from well-established practices in NLP, where prompting has proven effective after intensive pre-training on general knowledge, a similar approach is proposed for promptable segmentation tasks. The objective is to generate a valid segmentation mask based on a provided specification, analogous to a prompt in NLP.

While the central theme of the paper involves the use of the segmentation mask generated by a mainstream model as a prompt for zero-shot models, an additional significant contribution involves the fusion of logits from both models. This aspect provides an interesting opportunity to investigate its potential application to the counting task. To be more specific, using ViLT with both a standard image and its segmented counterpart produced by SAM, and then summing their output logits, could potentially improve the overall classification performance. When processing segmented images, ViLT can focus on different aspects of the image, which may help obtain a better output.

Positive outcomes from this examination could suggest the feasibility of a dual-branch model. In deep learning, a dual-branch model utilizes two parallel

neural network pathways that operate independently on input data. This design enables the model to acquire diverse and complementary representations, resulting in a more complete understanding of the input and often leading to improved overall performance.

3.3.2 COUNTING WITH SEGMENTED IMAGES

SAM comes in different sizes. Given that this method is applied only at inference time, the initial configuration involved segmenting the image only when required. The choice of SAM Base was motivated on the need for quick computation times. However, it became clear that this configuration would lead to longer inference times, making testing and debugging impractical. While acknowledging that an architecture like this could be technically viable, for the purposes of this thesis, the decision was made to pre-segment the images to ensure more feasible and efficient experimentation.

With time constraints no longer a concern, the largest version of SAM, referred to as SAM Huge, was used. The model and its weights were obtained directly from the original GitHub repository². The segmented image is then generated from the raw masks using the Supervision library (see Appendix B for details).

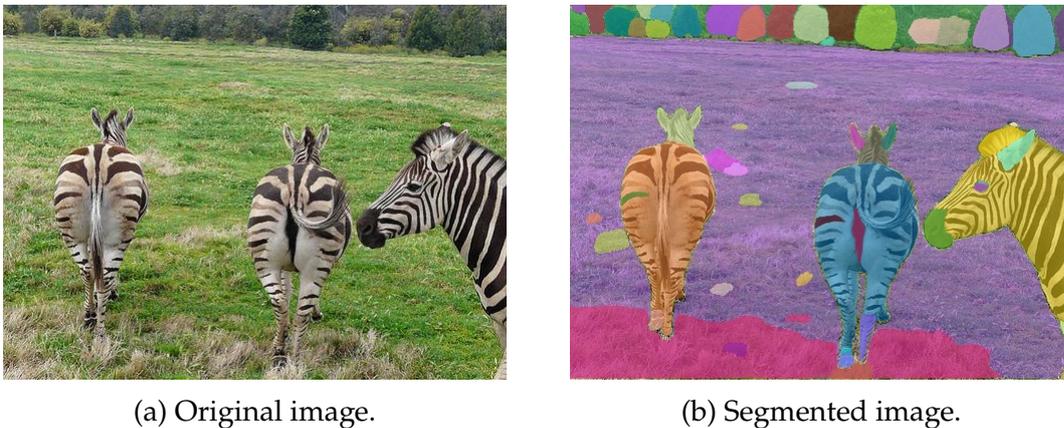


Figure 3.6: Counting probe segmentation example.

In Figure 3.6, an example of a segmented instance is provided. The fundamental concept behind this approach is that ViLT could potentially concentrate

²<https://github.com/facebookresearch/segment-anything>

3.3. SEGMENTATION

more effectively on the requested subjects when the entire image is segmented. If successful, this could represent a significant enhancement, particularly in visually complex images.

Logits fusion was performed with different sum rules in an attempt to adjust the weights of the original and segmented parts. In the first scenario, both were given equal weight, while in the second scenario, the standard image was given twice the weight of the segmented one. Results on the standard VE counting dataset are presented below, omitting TPR, FPR and AUC-ROC but instead providing the number of correct and wrong predictions for each class. The explanation for this decision will be provided shortly.

Checkpoint	Sum Rule	Accuracy (%)	TP	TN	FP	FN
ViLT VE	No	37.65	1451	2209	3619	568
ViLT VE Counting	No	77.50	1110	6425	856	1329
ViLT VE	$1 * N + 1 * S$	44.21	1385	2913	3327	737
ViLT VE Counting	$1 * N + 1 * S$	78.28	761	6849	433	1678
ViLT VE	$2 * N + 1 * S$	40.94	1420	2560	3497	653
ViLT VE Counting	$2 * N + 1 * S$	78.43	982	6643	639	1457

Table 3.5: ViLT evaluated on VE counting dev set with different sum rules. N and S stands, respectively, for Normal and Segmented logits.

Table 3.5 provides the True Positive (TP), True Negative (TN), False Positive (FP), and False Negative (FN) predictions for each checkpoint under different sum rules. At first glance, it may appear that ViLT performs better with segmented features. However, the accuracy improvement is primarily attributed to an increase in “contradiction” predictions. Given the highly unbalanced nature of the VE counting dataset, particularly biased towards negative instances, predicting more of them naturally leads to apparent improvements in results. It appears that ViLT may struggle to reason effectively with segmented images, encountering confusion. This is supported by the observations under the second sum rule. By assigning less significance to the segmented logits, positive predictions revert to their initial values, with an increase in TP and a decrease in TN.

To further validate the observed issue, segmented images are also evaluated on ViLT VE counting balanced using the VE counting balanced dataset. As indicated in Table 3.6, the tendency to predict more “contradiction” is main-

Sum Rule	Accuracy (%)	TP	TN	FP	FN
No	63.24	1506	1577	859	933
$1 * N + 1 * S$	62.40	1225	1817	619	1214
$2 * N + 1 * S$	63.69	1389	1716	720	1050

Table 3.6: ViLT VE counting balanced evaluated on VE counting balanced dev set with different sum rules.

tained, and assigning less weight to the segmented images continues to alleviate this confusion. Although the results showed some improvement by a certain percentage, it becomes apparent that this methodology is inherently flawed.

3.3.3 ERRORS

In summary, it appears that ViLT does not extract useful information from segmented images. This behavior is likely rooted in how the model learned to perceive visual data during its pre-training phase, being accustomed to “normal color objects”. Using the example in Figure 3.6, the fact that the zebras in the segmented images have different colors compared to their real counterparts could indeed induce confusion. As mentioned in the previous section, ViLT appears to struggle with image comprehension, resulting in confusion and a higher than normal tendency to make negative predictions.

The idea of a dual-branch architecture, subjected to a fine-tuning phase where one of the ViLTs learns from segmented instances, could potentially address the problem. However, the current indications do not strongly suggest this approach. The colored mask overlay on standard images represents a high level of abstraction, and expecting a VLM of ViLT’s magnitude, designed to possess a broad knowledge base, to interpret it accurately might be overly ambitious. Even with fine-tuning on segmented instances, the challenges of proper interpretation and learning may persist.

4

Visual Question Answering

After an extensive exploration of the VE task, valuable insights into ViLT’s reasoning capabilities were gained. While the exploration proved satisfactory, the limitation of classifying between only three classes may still be perceived as too restrictive. To overcome this limitation, the VQAv2 dataset is used. This dataset allows to directly answer questions related to the given image without the previous constraint of three classes.

To improve results, a different approach has been taken compared to previous attempts. In Chapter 3, the main focus was on the data, trying to extract more information from it. This chapter focuses on modifying the structure of the model itself in order to extract better features using a CNN. Various versions of the ViLT-CNN hybrid were tested to explore their potential.

4.1 EXPERIMENTS

To further deepen into experiments, the VQA task was used. Choosing from more than two or three classes could reveal other aspects of ViLT’s reasoning capabilities. Another motivation for undertaking experiments with this task is to have a more practical interaction with the model. This task is important in applications where it is necessary to comprehend visual content along with natural language queries.

The VQA task is tested for the same reasons that led to the use of the VE dataset. VQA requires specific answers for each sample, unlike general classifications such as NLVR2 and VE, which it tries to deviate from. This adds another

4.1. EXPERIMENTS

layer of complexity to ViLT’s comprehension and response capabilities.

4.1.1 VQAv2

VQA draws inspiration from the Question Answering (QA) task in NLP. The goal of QA is to create systems that can understand the context of a given question and extract or generate the most appropriate answer from a given set of documents or knowledge sources. In VQA, differently, the model is provided with an image, and users can ask questions related to the content of the image. The challenge for models is to understand the visual context, recognize objects and relationships within the image, and interpret the semantics of the question to respond in a meaningful way.

In the original paper [17], ViLT is fine-tuned using the VQAv2 dataset proposed by Goyal et al. [10]. This dataset extends the previously cited VQA dataset by Antol et al. [3], nearly doubling the number of real-world scenario samples. The creation of the VQAv2 dataset is motivated by addressing a fundamental challenge in VE, specifically the imbalance in its answer distribution.

As previously discussed in Section 1.2.1, despite the original annotations being in free-form natural language, most VLMs approach the VQA task as a classification task. ViLT maintains this strategy by approaching the task with a classifier head of 3129 answers, typically representing the most common answers. This is a significant deviation from the two or three classes seen in previously tested tasks.

In contrast to providing a single correct output for each instance, the VQAv2 dataset presents ten human-collected answers for each question. This approach allows models to undergo training in a manner distinct from a simple multi-class classification. The scores for each provided response, called soft scores, are calculated according to the guidelines outlined on the original dataset site¹ and are elaborated in Equation 4.1. Consequently, this setup handles situations where more than one answer may be correct due to ambiguity. The loss function employed during the training phase is a Binary Cross Entropy multiplied by the number of classes of the classifier. This approach enables optimization for

¹<https://visualqa.org/evaluation.html>

multiple answers by capturing the uncertainty in ground truth annotations.

$$S(Ans) = \min\left(\frac{\#\text{human that said } Ans}{3}, 1\right) \quad (4.1)$$

The author of ViLT fine-tuned the model using both the training and development sets of VQA_{v2}, achieving an accuracy of 71.26% on the test set. Similar to other results obtained, ViLT distinguishes itself with commendable scores and notably short inference times. Unfortunately, the test set in question does not publicly provide its answers, making it impractical to replicate the experiment. While using the training set for training and the development set for testing with the obtained checkpoint was a potential alternative, an unexpected error occurred. It appears that ViLT fails to learn effectively from the dataset. Consequently, the decision was made to abandon this approach.

4.1.2 COUNTING PROBE VQA

To avoid abstaining from experiments in the VQA task, the well-known counting probe dataset was used. As previously elucidated, the VQA task constitutes one of the two original representations of the dataset, making it straightforward to test pre-existing weights for ViLT on VQA. In alignment with the strategy adopted for the VE task, a fine-tuning process was conducted, commencing with ViLT VQA weights² on the counting probe training set. The obtained results are presented below.

Checkpoint	Accuracy (%)
ViLT VQA	55.64
ViLT VQA FT on counting probe	50.35

Table 4.1: Evaluation of ViLT weights on counting probe VQA dev set.

The concept of fine-tuning from a more broadly oriented task, such as VQA, to a more specific one, like counting, is a standard procedure. As depicted in Table 4.1, it is evident that in this case, the fine-tuning process failed to achieve its intended outcome. This failure is likely attributed to the manner in which the original training was conducted, as counting probes do not utilize soft scores

²Weights provided by original author

4.1. EXPERIMENTS

like VQAv2, potentially leading to confusion. Despite the negative outcome, not all efforts were in vain. Using ViLT’s VQA weights to analyze how the model reasons in this scenario can offer valuable insights.

ERRORS

Considering only the non-fine-tuned model, the number of predictions for each label, whether correct or incorrect, was carefully noted. Monitoring these counts has enabled the extraction of valuable information. Up to this point, the experiments have focused on determining whether the model can count, but an exploration of how the model performs counting has not been conducted. The relevant data is presented below.

Label Predicted	Correct Predictions	Wrong Predictions
0	154	77
1	474	203
2	483	261
3	128	186
4	64	134
5	14	60
6	20	54
7	1	11
8	7	25
9	2	7
10	4	24
11	-	1
12	5	7
13	-	1
15	1	10
20	-	12
30	-	1
50	-	1
100	-	2
many	-	5

Table 4.2: Number of prediction for each label.

From Table 4.2, it is easy to see how quickly the counting abilities deteriorate. Only the first three labels have positive Correct/Wrong rates. It is also interesting how the number of predictions drops significantly after the number six.

One possible explanation for the model’s inaccuracies is its difficulty in accurately counting a large number of objects. A more comprehensive exploration of this issue is presented in the literature by Kajic et al. [15]. Their study, among other insightful investigations into how different modalities handle counting tasks, reveals that VLMs trained on VQAv2, similar to the case with ViLT, encounter difficulties when dealing with large numbers (greater than 4). This aligns with the observations made for ViLT. In the VQAv2 dataset, small numbers (1-4) constitute 83.6% of the numeric instances, indicating a significant imbalance that likely contributes to these subpar results.



Associated Text: How many boats do you see?
Correct label: 14
Predicted label: many

Figure 4.1: Example of “many” label prediction.

One noteworthy aspect is the predictions associated with the *many* label, example shown in Figure 4.1. While these predictions may not be technically incorrect, they do not align with the intended objective. ViLT relies on its pre-defined dictionary, which includes numerous possible answers to accommodate various situations and types of questions. This extensive dictionary enables the model to generate labels that may seem plausible but do not necessarily fulfill the specific requirements of the task at hand.

4.1.3 VQAv2 COUNTING

To conduct more in-depth experimentation on the VQA task, a practical solution was formulated, given the inability to use the full VQAv2 dataset. Drawing inspiration from the counting probe methodology and leveraging the well-labeled nature of VQAv2, a subsection of the dataset was carefully selected.

4.1. EXPERIMENTS

VQAv2 includes questions and answers categorized by their types. By restricting the dataset to instances where the Question type is “how many” and the Answer type is “number”, the creation of VQAv2 counting became feasible.

The primary distinction between utilizing the entire dataset and the subset lies in the scale of the number of classes. For counting, this number is significantly reduced. Predicting precise counts for very large numbers of objects is nearly impossible for a VLM and is also not useful. To mitigate the use of impractical labels, numbers exceeding a specified threshold are constrained. For this purpose, labels larger than a certain number n are relabeled as “> n ”. Smaller numbers remain unchanged. Two different training scenarios were tested, setting n to 20 and 10, respectively.

By adopting this approach, the number of classes is reduced from over three thousand to either 22 (numbers from zero to 20 plus the “> 20” class) or 12 (numbers from zero to 10 plus the “> 10” class). This adjustment appears to influence the outcome of fine-tuning, as ViLT demonstrates a more confident prediction of classes.

However, the results on the development set align with the outcomes observed in the counting probe, showing suboptimal performance. Challenges in achieving satisfactory results in this specific task seem to persist. The detailed results are presented below.

Checkpoint	Accuracy (%)
ViLT VQAv2 counting 20	46.46
ViLT VQAv2 counting 10	48.65

Table 4.3: Evaluation of ViLT weights on VQAv2 counting on two different threshold.

ERRORS

Table 4.3 highlights the expected suboptimal results arising from an unbalanced dataset. Examining the confusion matrix, represented in Figure 4.2, of predictions reveals a concentration in the 0-1-2 range, consistent with the findings from the VQA counting probe. For a clearer interpretation, the confusion matrix for the $n = 10$ threshold is presented, which reflects similar outcomes obtained with the $n = 20$ threshold.

Confusion Matrix

o	561	362	194	50	22	11	3	0	3	0	0	42
r	89	1786	940	140	49	26	8	3	2	0	4	13
n	52	372	1769	419	153	26	15	3	7	0	1	19
m	23	78	426	568	206	34	12	3	1	1	0	17
v	23	39	153	250	263	95	31	8	4	0	1	22
l	11	13	44	97	167	56	31	6	2	0	1	24
e	5	12	26	41	83	50	60	13	13	1	1	32
r	2	4	6	23	31	26	20	11	16	3	2	12
o	4	6	6	10	26	29	36	11	20	5	3	33
o	2	2	3	5	6	12	16	10	6	4	1	15
h	3	2	5	9	14	11	13	5	13	6	3	39
u	26	19	35	29	39	32	40	18	39	19	25	546
	0	1	2	3	4	5	6	7	8	9	10	>10

True Label

Predicted Label

Figure 4.2: VQAv2 counting 10 confusion matrix.

Apart from the recognized issue of an imbalanced distribution, as discussed in Chapter 3, it appears that the model is not effectively focusing on the targets. To improve the finetuning for the counting task, this aspect should be addressed similarly to Chapter 3. This chapter does not focus on the data, so the dataset remains unchanged.

4.2 CNNs

Rather than simply drawing inspiration from them, future experiments will fully exploit the capabilities of CNNs. Instead of using images in their raw form, they will be processed through a pre-trained CNN using various methods. The CNN-ViT hybrid is well-established in the CV literature. As outlined in the work of Khan et al. [16], these architectures can be intricate, but the fundamental concept involves utilizing the image representation obtained from specific inner layers of a CNN to feed into the ViT. This approach ensures that the features contain more carefully selected and richer information than the simple linear projection used by ViT.

Since ViLT is based on ViT, it is technically reasonable to apply a CNN encoder to its images. The following section presents various experiments aimed at incorporating a CNN encoder into ViLT, involving different parameter settings and different architectures. The counting 20 task is used for testing, with the aim

4.2. CNNS

of utilizing the more complex features extracted by the CNN in the challenging task of counting.

4.2.1 LINEAR PROJECTION

The initial version of CNN-ViT tested was kept simple. In the original ViT paper by Dosovitskiy et al. [9], a CNN hybrid is mentioned and tested. The key difference from the standard architecture lies in applying the patch embedding projection to patches extracted from a CNN feature map instead of directly on the linearized image. Hugging Face’s Transformers library provides implementations for both ViT³ and ViT-hybrid⁴. Inspiration for the implementation was drawn from the source code of the latter.

The implementation by Hugging Face specifically utilizes Big Transfer (BiT), as proposed by Kolesnikov et al. [19], which provides a straightforward approach to facilitate transfer learning for CNNs. In this context, the chosen CNN is ResNetV2, by He et al. [11]. ResNetV2, short for Residual Networks Version 2, is a deep neural network architecture known as one of the first deep neural networks to eliminate the problem of vanishing gradient descent, as Li et al. point out in [22]. It has become a cornerstone in CV applications, consistently delivering optimal results in various image classification challenges thanks to its new method of extracting richer features.

As previously stated, a particular internal layer is selected to extract feature maps for each image. The use of the Hugging Face environment simplifies this process by providing convenient access to hidden layers (see the Appendix C for more information).

In this instance, the penultimate layer is selected, providing feature maps of size [1024, 28, 28] for each image. Similar to BERT, ViLT uses an embedding size of 768 for each token. To handle the linear projection, a convolutional layer is employed to transform the number of channels from 1024 to 768. The kernel size and stride of this convolutional layer define the patch size. Although ViLT typically uses a patch size of 32×32 for standard images, the size of each channel in the CNN’s feature map is 28×28 , meaning patches of that size cannot be directly utilized. However, this should not pose an issue, as these patches do

³https://huggingface.co/docs/transformers/model_doc/vit

⁴https://huggingface.co/docs/transformers/main/en/model_doc/vit_hybrid

not represent actual parts of images but serve as informative features.

The initial test involves using a convolutional layer with a kernel size and stride of 16. Employing this approach on a feature map of [1024, 28, 28] results in a single token, which is too limited to convey useful information to the model. Nonetheless, an evaluation was conducted to comprehend the dynamics. Subsequent attempts involved experimenting with different kernel sizes and stride values to increase the number of features. The results are presented below.

Kernel size	Stride	Number of tokens	Accuracy (%)
16	16	1	30.94
4	4	49	33.17
4	2	169	34.77

Table 4.4: Evaluation of ViLT-CNN-v1 with different parameters.

In the latest test, the adherence to the concept of patches is no longer maintained. With a stride smaller than the kernel size, the convolution operation uses certain elements more than once to compute the final result. This was attempted to further increase the number of tokens, it can be viewed as an exchange of information between them. The results in Table 4.4 are generally suboptimal, but a noticeable trend can be observed. As the number of tokens increases, the accuracy tends to improve.

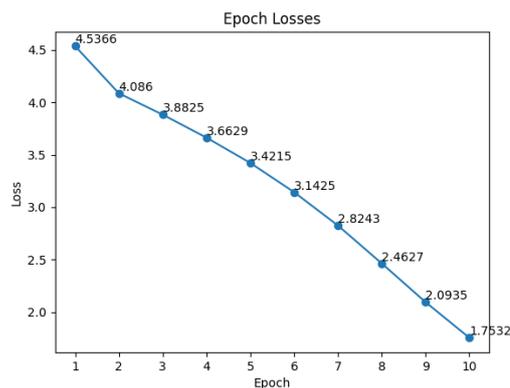


Figure 4.3: 10 epochs fine-tuning.

Observing the loss plots in Figure 4.3, it appears that the loss did not reach a plateau in the standard 10 epochs. Increasing the number of epochs was a logical next step to try to obtain improvements. After 20 epochs, the loss began

4.2. CNNs

to stabilize, as shown in Figure 4.4. However, extending the training time did not improve the results. The longer trained model achieved an unsatisfactory accuracy of 33.94%. Similar tests were conducted for subsequent versions, but better performance was not reached. One hypothesis is that more epochs may result in overfitting⁵ on the data, leading to worse results.

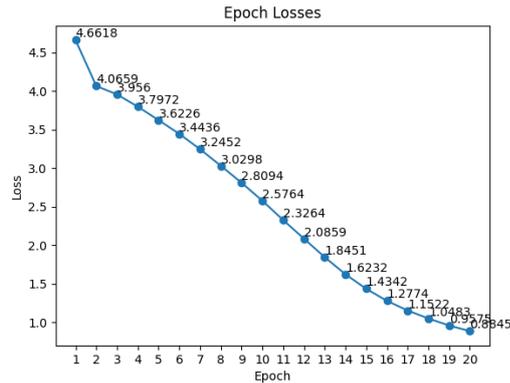


Figure 4.4: 20 epochs fine-tuning.

4.2.2 RESHAPING

As per ViLT’s paper [17], the maximum number of tokens created for each image is set at 200. With a kernel size of 4 and a stride of 2, 169 tokens are achieved, a substantial amount in terms of information conveyed. However, the method to obtain these tokens is somewhat vague. As mentioned earlier, the patch mechanism characterizing ViT is absent, employing different sizes from the 32×32 used to train the model. Certainly, ViLT demonstrates its ability to interpret the provided information, showcasing improved accuracy as the number of features increases. Nevertheless, employing a more structured method might offer the potential for better feature extraction.

A novel method, distinct from a simple linear projection, has been devised. Incorporating custom features into ViLT requires a complex process that involves changing the way the model generates the intended features. Currently, features are crafted through a form of downsampling to produce tokens of the required hidden size. Instead of compressing the information, an alternative approach

⁵In machine learning refers to a model learning the training data too precisely, leading to poor performance on new, unseen data.

involves the opposite process. The new method incorporates an upsampling component to restore features to the size of the original image. This way, ViLT autonomously operates, creating the features required in the correct format.

To accomplish this upsampling, a series of transposed convolutional layers is employed. These layers contribute to enhancing the spatial resolution of the input data. This approach could facilitate the network’s understanding of counting feature extraction, a task that requires refined and detailed features. Using a sequence of transposed convolutional layers, the network could learn a hierarchical representation and be used as a decoder for tasks such as segmentation [32].

This method is tested with two different CNNs. The first is still ResNetv2 implemented in the Hugging Faces library. The second one is SqueezeNet, proposed by Iandola et al. [14], from the PyTorch library. The decision to test a different network from a different library is made to validate the obtained results and avoid the risk of having inaccurate evaluations due to potential implementation discrepancies. The results outlined in Table 4.5 indicate that even with this method, meaningful results are not achieved.

CNN	Accuracy (%)
ResNetv2	33.50
SqueezeNet	33.60

Table 4.5: Evaluation of ViLT with different CNNs and upsampling.

4.2.3 ERRORS

Both the fields of CV and NLP demonstrate that models can be fine-tuned on tasks different from their original pre-training objectives. In CV, a model pre-trained on a classification task can be fine-tuned for an object detection task [34]. Similarly, in NLP, a model initially pre-trained for language modeling can be fine-tuned for sentiment analysis [12]. Notably, both of these examples may involve inputs different from their original scope.

It’s common practice in Machine Learning to modify inputs for different tasks. However, introducing a drastic change like the integration of CNN features could potentially overload the model. The input data could deviate significantly from the distribution learned during the pre-training phase, making

4.2. CNNS

it difficult for fine-tuning to produce significant improvements.

Another potential factor contributing to the model's inability to effectively utilize the newly provided features is the alteration in how the image is presented to the model. This modification could disrupt the current relationships between vision and text, leading to unpredictable or random behavior in the model.

The main cause of the suboptimal results should be investigated by examining these hypotheses.



Generative VLMs

Generative AI served as a primary motivation for this thesis’s development. However, ViLT, due to its limited size and the different objectives pursued in its original development, does not provide a suitable framework to properly test this aspect.

The Language Modeling (LM) head in ViLT represents the closest mechanism for generating responses. Using the counting probe dataset once again, it is possible to test this aspect of ViLT. The original paper [30] proposes experiments of this type by masking the number in declarative sentences. After finding this exploration of generation unsatisfactory, further investigation into VLMs better suited for this task was conducted. Large Language and Vision Assistant (LLaVA), a model much larger and with ad hoc training, has been chosen to delve into this field.

5.1 COUNTING VIA MLM

In another experiment conducted in the counting probe paper [30], the LM head of models was tested. As previously mentioned, MLM is one of the main tasks in VL, indispensable for creating associations between text and image. The creative component of LM is particularly intriguing as it represents a step towards the realm of generative AI. Additionally, the fact that the answers are not limited to a dictionary, as in classification tasks, is particularly interesting and could provide an assessment of counting skills at a different level of reasoning.

5.1. COUNTING VIA MLM

5.1.1 DATASET AND EXPERIMENTS

Declarative sentences from the counting probe dataset are utilized. Typically, in the MLM task, words are masked with a certain probability; however, in this case, only the number defining the quantity is masked. The masking of numbers in sentences is straightforward, as they are represented as digits, and a simple script is employed. Once detected, the numbers can be substituted with a [MASK] token and with the number in words. The generation of both the ground truth annotations and input sentences can be accomplished through this approach.



Associated Text: There is [MASK] yellow trucks.
Correct label: There is one yellow trucks.

Figure 5.1: Example of instance of counting probe LM dataset.

The instances follow a format similar to that of other tasks, consisting of an image paired with a descriptive sentence in both the masked and regular versions. An illustration of this format is presented in Figure 5.1.

ViLT undergoes two experiments: one evaluating its LM head in its original form, and the other assessing it after fine-tuning on the training dataset. This shift is interesting as the task transforms from a classification-oriented one to a more generative nature. While not a true generative AI task, it represents a reasonable compromise to explore this domain. The LM head is one of the two primary pre-training tasks. It is expected to have extensive general knowledge that can be leveraged.

As evident from Table 5.1, a significant improvement is observed through fine-tuning. It is logical to presume that the non-fine-tuned model has a diverse set of logits in its dictionary to handle a broad range of cases. On the other hand, the fine-tuned model prioritizes numbers, resulting in noticeably better

Model	Accuracy (%)
Not fine-tuned	29.64
Fine-tuned	61.17

Table 5.1: Evaluation of ViLT counting probe LM dev set.

performance.

5.1.2 ERRORS

The most frequent mistakes arise from the final observation. The non-fine-tuned model replaces the masked token with words that are technically correct but deviate, or fail to comprehend, the actual goal of the task. As illustrated in Figure 5.2, the model generates tokens that form a grammatically correct phrase but fail to capture the essence of the task. On the other hand, almost all correct predictions have a number of objects falling in the range $[1, 3]$, with the majority being two. This is likely influenced by the distribution of numbers encountered during the pre-training phase.

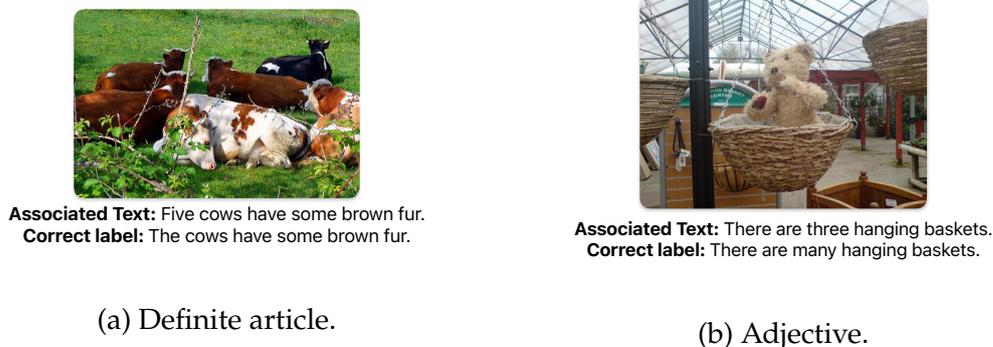


Figure 5.2: Example of technically not wrong errors in ViLT LM not Fine-tuned.

Figure 5.3 illustrates some errors for the fine-tuned model. A recurring error pattern emerges with a high number of objects, which is a well-known challenge for VLMS. Errors with a low number almost always occur in close proximity to the actual number.

The LM task is set aside, as it seems inappropriate to constrain its vocabulary trained for a general context in this manner. The focus has shifted towards exploring ways to achieve a more authentic generative output. However, due

5.2. VISUAL INSTRUCTION TUNING



Associated Text: There are fourteen bike riders.
Correct label: There are six bike riders.

(a) High number error.



Associated Text: There are two people.
Correct label: There are three people.

(b) Low number error.

Figure 5.3: Example of errors in ViLT LM fine-tuned.

to the computational limitations outlined earlier, the study is confined to the documentation of such methods.

5.2 VISUAL INSTRUCTION TUNING

Generative tasks for VLMs can span various applications, ranging from generating image descriptions to VQA. However, one of the most intricate generative tasks is conversation. The development of a chat-bot system that not only understands textual inputs but also incorporates visual understanding, as discussed in Chapter 1, is an example that is often cited, particularly for those who are not experts.

Instruction Tuning (IT) is a technique designed to guide a LLM towards generating correct responses. As outlined in the paper by Zhang et al. [45], the primary challenge lies in the mismatch between the model’s training objective, which aims to minimize contextual word prediction errors, and the user’s objective, which is to obtain an answer that adheres to their instructions.

By fine-tuning a pre-trained model on a dataset with a format of (I, O) , where I denotes human instructions for the model and O denotes the desired output aligning with the instruction, this technique helps the model quickly adapt to producing the required answers. It even allows for domain-specific knowledge restriction, paving the way for the development of an effective virtual assistant.

One of the primary challenges of the IT technique lies in the creation of high-quality data. Since it is a supervised task, leveraging vast unannotated corpora, as done in the pre-training of LLMs, is not feasible. This issue has also

been encountered in the pre-training of VLMs. To streamline the process and ease the data collection burden, outputs from other Language Models, such as GPT-4, are employed instead of manually gathering them.

5.2.1 LLaVA

In a straightforward understanding, IT enhances the zero-shot capabilities of models. To build an effective virtual assistant, these capabilities are essential, but this idea is less explored in the multimodal field. In the work presented by Liu et al. [24], the first visual instruction tuning dataset is introduced. With the help of this dataset, the training of LLaVA, an end-to-end large multimodal model for general visual and language understanding, is successfully accomplished.

Datasets such as GCC [35] and MSCOCO [23], where an image is associated with text, are well-known in the multimodal field and are typically used for conventional pre-training tasks. However, by utilizing models from the GPT family, multimodal instruction datasets can be created using these established datasets, leveraging the annotation capabilities of these models. For an image X_v and its associated caption X_c , a set of questions X_q can be generated to train the virtual assistant in describing the image. Through various techniques to address issues such as lack of diversity and limitations in in-depth reasoning (refer to the original paper [24] for detailed insights), The authors were able to produce a dataset of 158K unique speech-image instruction-following samples.

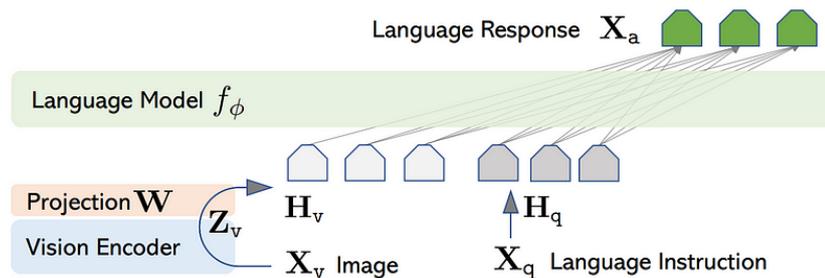


Figure 5.4: LLaVA architecture.

As shown in Figure 5.4, the architecture of LLaVA utilizes the power of a pre-trained LLM by utilizing a visual model to “translate” images to be understood by it. The LLM used in LLaVA is Vicuna, an open-source chat-bot achieved by fine-tuning LLaMA [39], which currently performs exceptionally well in the SOTA [6]. Visual features are obtained by leveraging the visual encoder of CLIP [31] ViT-L/14, known for its robustness due to extensive training data.

5.2. VISUAL INSTRUCTION TUNING

These grid features are then converted into the word embedding space using a simple linear layer, specifically a trainable projection matrix.

For each image X_v , a multi-turn conversation data $(X_q^1, X_a^1, \dots, X_q^T, X_a^T)$ is generated, where T is the total number of turns. To perform IT, the original auto-regressive objective for token predictions is selected:

$$\Pr(X_a | X_v, X_{\text{instruct}}) = \prod_{i=1}^L \Pr_{\theta}(x_i | X_v, X_{\text{instruct}, < i}, X_{a, < i}) \quad (5.1)$$

Here, θ represents the set of trainable parameters, and $X_{\text{instruct}, < i}$ and $X_{a, < i}$ denote the instruction and answer tokens in all turns before the current prediction token x_i . Following this established paradigm, LLaVA undergoes a two-stage IT phase.

Stage 1: Features alignment A filtered version of the CC3M dataset, containing images with associated descriptions, is utilized to create the instruction-following data. Language instructions requesting the description of the image are utilized as questions, while the description of the image itself serves as the prediction ground-truth. During training, all components are frozen except for the projection matrix. This approach ensures that visual features are aligned with the pre-trained LLM, effectively resembling the training of a compatible visual tokenizer.

Stage 2: Fine-tune The visual encoder weights remain frozen, while the weights updated during training include those of the LLM and the projection matrix. A chatbot is developed using the instruction-following data described earlier, incorporating both single-turn and multi-turn conversation samples. Additionally, the pre-trained model is trained on ScienceQA, which is the first large-scale multimodal science question dataset that annotates answers with detailed lectures and explanations.

Thanks to these training settings, LLaVA achieves significant reasoning capabilities. As illustrated in Figure 5.5, the model demonstrates optimal instruction-following ability, satisfactorily and correctly respond to every aspect of user requests. The model’s understanding of what is unusual in the photo highlights its strong reasoning abilities, outperforming GPT-4 and other concurrent models (See the paper for more details). LLaVA exhibits extensive knowledge



User What is unusual about this image?

LLaVA The unusual aspect of this image is a man ironing clothes on the back of a minivan or van. This is not a typical place to perform this activity, as one would usually iron clothes in a more stationary and safe location, such as a home, using a regular ironing board. The scene depicted in the image is peculiar as it involves a makeshift ironing setup on a vehicle, which can be both unsafe and unconventional. Additionally, it is not clear how the man is able to maintain balance and stability while ironing clothes in such an unstable environment.

Figure 5.5: LLaVA Example.

coverage and multilingual understanding capabilities, and it is capable of processing high-resolution images, enabling it to reason about individual elements of images as well as their overall context.

Comparisons with ViLT are not meaningful, as LLaVA leverages the power of two pre-trained models (Vicuna and the CLIP visual encoder) and employs a completely different training method. Furthermore, differences extend to the number of parameters, with ViLT consisting of a modest 111 million parameters compared to the substantial 175 billion parameters of LLaVA. This discrepancy is analogous to comparing models such as BERT with GPT-4. As previously stated in Chapter 1, it was not feasible for this dissertation to use a model of LLaVA's size, but it is important to recognize its potential for future research.



Low-Rank Adaptation

Following numerous tasks and attempts to improve results, where satisfactory outputs were not achieved, a decision was made to shift focus towards optimizing the training process rather than concentrating solely on the model or the data. This decision was partly influenced by the limited computational resources available. One method that combines both observations is Low Rank Adaptation (LoRA), a training technique capable of yielding comparable or superior results while reducing the required computational resources. LoRA operates by initiating constrained adaptations from pre-trained model weights, effectively achieving the desired outcomes without diverging significantly from the initial state. When applied to the VE task, LoRA yielded intriguing results.

6.1 WHAT IS LoRA

In the realm of NLP, there is a growing emphasis on developing techniques for efficient model training. These approaches aim to maximize performance while minimizing computational resources. Parameter-efficient training methods in NLP focus on optimizing model architecture, parameter utilization, and training algorithms to achieve high accuracy with reduced computational complexity. These methods are particularly crucial for deploying NLP models in resource-constrained environments, as is the case in this thesis.

While these methods are not novel in the machine learning domain, their application has been particularly encouraged in the NLP field due to the compu-

6.1. WHAT IS LORA

tational complexity inherent in its tasks. Recent advancements have predominantly centered around transformer-based architectures. Given that most VLMs are based on transformers, including ViLT, the application of these techniques to them is straightforward.

Most of the techniques proposed, while achieving their intended goals, come with some drawbacks. One such example is Adapters, which are modular neural network components inserted between layers of a pre-trained model. Adapters enable efficient fine-tuning for specific tasks by adding task-specific parameters while keeping the majority of the pre-trained model parameters frozen. However, a notable increase in latency can be observed during inference time, posing a challenge in real-time applications.

LoRA, proposed by Hu et al. [13], introduces a method that achieves comparable, if not superior, performance to classical fine-tuning while addressing the main issues associated with the aforementioned methods. One of the primary benefits of LoRA, aside from its simple design, is its applicability to any dense layer of a model, although it is common practice to apply it to specific weights of the transformer. LoRA, thanks to the way weights are updated, is also capable of reducing catastrophic forgetting during fine-tuning.

6.1.1 METHOD

A neural network is composed of multiple layers that perform matrix multiplication, where the weights of these layers are typically represented by full-rank matrices. It is known that pre-trained language models exhibit low “intrinsic dimension” [1], allowing them to efficiently learn even after random projection to a smaller subspace. The authors hypothesize that weights also possess low “intrinsic rank” during fine-tuning.

For a pre-trained weight matrix $W_0 \in \mathbb{R}^{d \times k}$, the authors constrain its update by representing it with a low-rank decomposition $W_0 + \Delta W = W_0 + BA$, where $B \in \mathbb{R}^{d \times r}$ and $A \in \mathbb{R}^{r \times k}$, with rank $r \ll \min(d, k)$. During training, only A and B are unfrozen, serving as the parameters capable of changing.

Figure 6.1 visualizes the modified forward pass. For input x , the output is obtained as $h = W_0x + \Delta Wx = W_0x + BAx$. A is initialized following a Gaussian distribution, and B is set to zero. The final step involves rescaling ΔW by a factor of $\frac{\alpha}{r}$, where α is a constant factor in r . α is the variable that scales the contribution of LoRA to the original weights during the training phase.

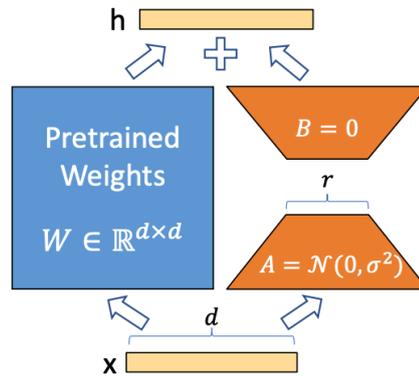


Figure 6.1: Visualization of LoRA's functioning.

During the inference phase, the final weight matrix can be explicitly computed as $W = W_0 + BA$, where both W_0 and BA are of the same size. Thanks to this, the inference phase does not take longer to compute than the standard model, as there is no extra layer introduced in the model.

6.1.2 RESULTS

In the Transformer architecture, the self-attention module consists of four weight matrices: Query, Value, Key, and Output, while the MLP module includes two additional weight matrices. The original paper on the application of LoRA in Transformers restricts its use to the attention weights, specifically the Query and Value matrices. This targeted application significantly reduces VRAM usage by 2/3 and results in a smaller checkpoint size by a factor of 10,000. Consequently, this optimization enables training on fewer GPUs. Additionally, the training process experiences a notable speedup compared to full fine-tuning, as it eliminates the need to compute gradients for the majority of parameters.

Model and Method	# Trainable Parameters	BLEU
GPT-2 M (FT)	354.92M	68.2
GPT-2 M (LoRA)	0.35M	70.4±0.1
GPT-2 L (FT)	774.03M	68.5
GPT-2 L (LoRA)	0.77M	70.4±0.1

Table 6.1: Comparison of fine-tuning and LoRA on the E2E NLG, a dataset for natural language generation, on GPT-2.

Referring to Table 6.1, the comparison using GPT-2 as an example clearly

6.2. EXPERIMENTS

demonstrates that despite a significant reduction in the number of parameters, the results achieved with LoRA surpass those of full standard fine-tuning. For further comparisons and detailed results, please refer to the original paper [13].

6.2 EXPERIMENTS

Addressing the counting problem raises concerns about ViLT’s potential lack of essential foundational information required for task handling. Its subpar performance may suggest that the model did not acquire the requisite capabilities during pre-training to accurately distinguish and enumerate selected entities. Another possibility is that important knowledge is lost during fine-tuning due to the significant adjustment of the weight matrices. It’s evident that the task is inherently challenging, yet focusing on improving either of these aspects could potentially yield better results.

Not being able to re-perform the entire pre-training process, given the low resources available, the decision on which approach to undertake becomes logical. To mitigate catastrophic forgetting, LoRA is employed. As introduced before, the authors hypothesize the low “intrinsic rank” of pre-trained language model weights. Given that ViLT is essentially a ViT, it is reasonable to make the same hypothesis for it as well. If true, LoRA could help mitigate the catastrophic forgetting of ViLT, potentially improving its results.

LoRA was initially designed for models much larger than ViLT, primarily to address computational costs. However, its other qualities could be exploited without any issues. Given the scarcity of literature on using LoRA with multimodal models, testing its impact on ViLT in terms of training time, resource usage, and performance improvements could provide insights into how larger VLMs could benefit from it.

To apply LoRA on ViLT, the Parameters Efficient Fine Tuning (PEFT) library from Hugging Face was chosen. This library offers various techniques to train models efficiently while conserving resources. Since ViLT is a model available in the Hugging Face Transformers library, the application of LoRA is straightforward. More details about the application can be found in the appendix D.

In all experiments, the parameters were set to rank $r = 16$ and $\alpha = 16$. These values proved to be effective, and no adjustments were made during the testing phase. However, there is potential for further improvements by tuning these parameters. The learning rate schedule and AdamW settings remained

unchanged to maintain consistency and closely resemble the full fine-tuning setup.

6.2.1 VISUAL ENTAILMENT

To thoroughly evaluate the improvements brought by LoRA, a suitable task must be selected. This task should consider not only classical evaluation metrics and computational resources but also a consistent training time to accurately identify any differences. Choosing the VE task, introduced in Chapter 3, for this evaluation is appropriate, as it offers a dataset with more than 400K samples, making it optimal for these tests. Below, an evaluation that takes into account these aspects is provided.

Model	# Trainable Parameters	Training Time	Accuracy (%)
ViLT VE-	114,562,566	98.2 hours	74.77
ViLT VE+	1,778,691	60.4 hours	75.51

Table 6.2: Comparison of results achieved on ViLT through fine-tuning alone (denoted by -) and results obtained with LoRA (denoted by +).

Applying LoRA to the Query and Value weight matrices, as in the original paper, significantly reduced the number of trainable parameters from over 114M to less than 1.8M, as demonstrated in Table 6.2. This represents only 1.55% of the total parameters, resulting in a substantial reduction in the number of parameters adjusted during training. Consequently, ViLT required significantly less memory for storage, allowing the batch size used in ViLT VE+ to increase from 64 to 128, doubling it compared to ViLT VE-.

The experimental results demonstrate a significant reduction in training time. ViLT VE+ completed training in 60.4 hours (approximately 2.5 days), a 38% speedup compared to full fine-tuning, which took 98.2 hours for ViLT VE- (equivalent to over 4 days). The doubling of batch size could be the cause of this difference. Additionally, ViLT VE+ achieved a higher accuracy percentage than ViLT VE-. These findings confirm the validity of the hypothesis.

Indeed, the modest improvement in accuracy is encouraging. Upon examining the predicted answer percentages by ViLT VE+, it becomes apparent that they exhibit less certainty compared to the full fine-tuning approach. While full fine-tuning often yields almost 100% certainty in its top prediction, LoRA results

6.2. EXPERIMENTS

in smoother probabilities across the classes. This could be attributed to the mitigated effects of catastrophic forgetting enabled by LoRA. Further investigation is needed to delve deeper into this aspect.

6.2.2 VISUAL ENTAILMENT COUNTING

The logical progression of testing LoRA on VE involves further fine-tuning the model on the VE counting task, as extensively described in Section 3.2. Given the already established and persistent challenges in the counting task, exploring whether LoRA can enhance the learning process is of considerable interest. For each experiment detailed in the tables, a batch size of 128 is utilized.

Model	Accuracy (%)	TPR (%)	FPR (%)	AUC-ROC
ViLT VE--	77.50	45.51	11.76	0.76
ViLT VE++	81.55	45.96	6.54	0.84
ViLT VE+-	77.92	51.90	13.38	0.80
ViLT VE-+	76.31	20.59	5.02	0.74

Table 6.3: Comparison of evaluation metrics obtained on ViLT through various combinations of training methods. The - symbol denotes standard fine-tuning, while the + symbol indicates the use of LoRA. In strings of the form ViLT VE_{xy} with x = -,+ and y = -,+, the symbol x represents how ViLT was trained on VE, and the symbol y represents how ViLT was trained on VE counting.

Table 6.3 clearly illustrates the significant improvement provided by LoRA in enhancing the model’s performance. Beyond accuracy, TPR, and FPR, the most crucial metric is the AUC-ROC, which determines the effectiveness of a classifier. Setting aside ViLT VE-+, which evidently does not leverage the benefits of LoRA, applying LoRA in the initial training phase results in improvements or comparable performance across all metrics compared to ViLT VE--. Focusing on the AUC-ROC scores, both models show enhancement, indicating promising results. However, further in-depth analysis is necessary to confirm the observed improvements definitively.

Table 6.4 provides the values for TP, TN, FP, and FN for each model. It is evident that ViLT VE+- and ViLT VE++ maintain or reduce the number of misclassified answers, confirming that the improvement in the AUC-ROC score is a valid result. The table confirms that ViLT VE-+ yields poor results, as the model attempts to achieve its objective by maximizing the number of correct

Model	TP	TN	FP	FN
ViLT VE--	1110	6425	856	1329
ViLT VE-+	502	6917	365	1935
ViLT VE+-	1266	6309	974	1173
ViLT VE++	1121	6807	476	1318

Table 6.4: Comparison of right and wrong classification obtained on ViLT through various combinations of training methods.

classifications. Given the dataset’s imbalance, the quickest route to achieve this is by making numerous negative predictions, a behavior previously observed in the task.

To address this behavior, LoRA was also applied to the Key weight matrix on ViLT VE-+. However, this adjustment increased the number of parameters, necessitating a return to a batch size of 64. Consequently, there was a slight improvement in critical metrics: Accuracy and AUC-ROC are now comparable to ViLT VE--, while the TPR increased to 25.63% due to an increase in TP and a decrease in FN. However, this approach was not further explored to maintain a batch size of 128.

Model	Accuracy (%)
ViLT VE--	64.78
ViLT VE-+	67.80
ViLT VE+-	43.89
ViLT VE++	46.07

Table 6.5: Catastrophic forgetting test on ViLT through various training methods.

The last test, the results of which are reported in Table 6.5, focuses on catastrophic forgetting. After adapting on VE counting, models are tested on the VE dataset. Despite its poor results, ViLT VEE-+ is the model that performs the best, while ViLT VE+- and ViLT VE++ exhibit notably worse performance. LoRA, by not altering the pre-trained weights, is expected to mitigate catastrophic forgetting in models. However, the difference in performance may stem from how pre-trained weights are defined. Applying LoRA on both VE and VE counting maintains the base ViLT as the starting point, while applying LoRA solely on VE counting considers ViLT VE as the initial state.

LoRA makes slight adjustments from the pre-trained base, and applying it

6.2. EXPERIMENTS

Confusion Matrix

True Label Contradiction	5514	1	424
True Label Neutral	4534	103	1323
True Label Entailment	3347	1	2611
	Contradiction	Neutral Predicted Label	Entailment

Figure 6.2: ViLT VE++ catastrophic forgetting confusion matrix.

twice in a row could almost completely eliminate the effects of the first application. This hypothesis is supported by the confusion matrix of ViLT VE++, as shown in Figure 6.2, where it's evident that ViLT VE++ almost completely removes the possibility of the "neutral" class.



Conclusions and Future Works

In this thesis, an exploration of the Vision Language field is conducted, delving into the functioning of various models. The study involves an understanding of the evolution of Visual Language Models, spanning from their origins to their anticipated future developments. Among numerous options, the choice was made to focus on ViLT. Working in an environment with limited computational resources, the model's simple architecture and short computation times appeared to be the optimal choice for initiating exploration in this domain.

The initial experimentation involved reproducing a result obtained in the original work. Despite differences in the training setup, comparable results were achieved on the Natural Language for Visual Reasoning for Real (NLVR2) task in relation to the author's findings, confirming the viability of working with the model. Throughout the dissertation, a common challenge encountered in every experiment is identified: the counting task in VLMs. As extensively documented in the literature, the counting problem poses a significant challenge for this type of models. Various tasks and methods are explored in an attempt to improve results.

To fine-tune ViLT on a task not addressed in the reported paper, the Visual Entailment (VE) dataset was utilized. The results obtained were mediocre; however, the primary focus of this dissertation is not to achieve the SOTA, but rather to enhance the baseline performance of the experiments. The VE dataset lacks sufficient counting samples to thoroughly test this aspect of the fine-tuned model. To address this limitation, the counting probe dataset is employed to augment the VE dataset, specifically focusing on the counting task. Through

this approach, the subpar performance of the model on the counting task is confirmed. Various data-handling techniques were explored in an attempt to improve performance, addressing both the textual and visual components of the data.

Various manipulations were applied to the textual part in an attempt to balance the distribution, but this only resulted in more confusion and worse results. The techniques employed were quite rudimentary, suggesting that a more sophisticated approach to textual augmentation, such as using GPT to generate synthetic data, could potentially improve the output. As for the visual component, the Segment Anything Model (SAM) (a zero-shot segmenter) was used to try to increase the focus on the subjects, but ViLT did not seem to extract any additional information from these segmented images.

It is hypothesized that training a dual-branch ViLT model with both standard and segmented images could yield more promising results. Additionally, SAM can be utilized with an initial prompt to assist in restricting segmentation to the desired subject, extracted from the associated caption. This approach could render segmented images much more relevant in terms of the information provided. Both of these experiments are left for future investigations.

To establish a more direct interaction with the model, the task was shifted to utilize the VQA_{v2} dataset, which is known for its free-form answers. The dataset was adapted for the counting task and employed to fine-tune ViLT, reaffirming the previously identified challenges. While in the previous task, data was the primary focus for improvement, in the VQA context, attention was focused on the model itself.

Drawing inspiration from the Vision Transformer (ViT), various CNN-ViT hybrids were designed and tested. The underlying concept was that pre-trained CNNs could extract more “rich” features to aid ViLT. Despite initially poor yet encouraging results in the first version, subsequent versions led to the abandonment of the experiment. One hypothesis for these subpar performances is that the model struggled to interpret the feature maps extracted by the CNN, which deviated from the input distribution. Another hypothesis proposes that modifying the visual characteristics disturbed the established connections between images and text, making them unidentifiable.

One possible solution may be to use the CNN during the pre-training phase. This would enable ViLT to learn how to effectively use this type of information. However, due to limited computational resources, this was not possible.

Generative AI was a central theme in the development of this thesis; however, ViLT lacks real text generation capabilities. To explore its generative potential, the Language Modeling head was tested by adapting the well-known counting probe dataset for this purpose. While MLM is not a true generative task, it provides insights into how ViLT might perform in such scenarios. As anticipated, the results aligned with those of previous tasks.

To address this issue, more advanced models and training methods have been investigated. In order to extend ViLT's text generation capabilities, the addition of a simple decoder head is considered a potential solution, although this route remains unexplored and is reserved for future experiments.

After exploring techniques to enhance results through data and model manipulation, a decision was made to investigate the training phase. Low Rank Adaptation (LoRA), a technique designed for efficient fine-tuning in LLMs, was selected for this purpose. It was applied to the tasks of VE and VE counting, facilitating standard evaluation and estimation of catastrophic forgetting.

Different combinations of standard training and LoRA were tested, resulting in improved (or at least comparable) outcomes. Although the evaluation of catastrophic forgetting did not yield the expected results, it provided valuable insights into its functioning.

Although LoRA is primarily developed to conserve resources in larger models, its application on ViLT demonstrates the potential for its utilization in VLMs with comparable results.

In conclusion, ViLT serves as an optimal starting point for delving into the vast field of VLMs, providing valuable insights and knowledge. However, it also reveals a significant problem of the models: their weak pre-training phase. LLMs are trained on large amounts of raw data, which gives them significant reasoning power. In contrast, VLMs require well-labeled datasets and do not achieve the same number of samples as LLMs, resulting in various biases, as seen with the counting task. In addition, the size of ViLT is a limitation; while it provides sufficient reasoning for basic tasks, its small size limits its effectiveness for more complex and demanding tasks.

Models such as LLaVA, explored in Section 5, with its extensive pre-training and large number of parameters, represent the future of the field. These models have the promise of addressing the weaknesses observed in ViLT and paving the way for advances in VLM capabilities.



Datasets management

In this appendix, the management of datasets in PyTorch is discussed. When initializing a dataset several arguments are passed:

- **annotations:** This argument consists of a list of dictionaries containing information like image ID and ground truth, necessary for creating dataset instances.
- **processor:** This refers to the method utilized for preparing image and text inputs for ViLT.
- **id2path:** This is a dictionary where image IDs serve as keys, mapping to their respective paths where the images are stored.
- **config:** This stores all the requisite details for instantiating a ViLT model.

It's important to note that each element's fields are squeezed to remove the batch dimension, as these elements will later be grouped into batches using a DataLoader.

```
1 import torch
2 from PIL import Image
```

Code A.1: Necessary libraries to create the datasets.

A.1 IMAGE PATHS

To obtain the dictionary associating image IDs to image paths, you can use the following simple nested *for* loop. Here, *root* is a string representing where the images are stored, and the ID of each image is simply the name of the file without the file type extension.

A.1. IMAGE PATHS

```
1 import os
2
3 def image_path(root):
4     id2path = {}
5     for images_root, dirs, files in os.walk(root):
6         for file in files:
7             file_path = os.path.join(images_root, file)
8             id2path[file[:-4]] = file_path
9
10    #path2id = {v: k for k, v in id2path.items()}
11    return id2path
12
13 images_path = "image/folder/path"
14 id2path = image_path(images_path)
```

Code A.2: Standard association.

The provided code snippet can be applied to process datasets, except for VQAv2, which requires a different approach due to its unique image naming format. To handle this, regular expressions are employed, a method sourced from the Allen Institute GitHub repository¹.

```
1 import re
2 import sys
3
4 from typing import Optional
5 from os import listdir
6 from os.path import isfile, join
7
8 filename_re = re.compile(r".*(\d{12})\.((jpg)|(png))")
9
10 def id_from_filename(filename: str) -> Optional[int]:
11     match = filename_re.fullmatch(filename)
12     if match is None:
13         return None
14     return int(match.group(1))
15
16 images_path = "image/folder/path"
17 file_names = [f for f in listdir(images_path) if isfile(join(
18     images_path, f))]
```

¹https://github.com/allenai/allennlp-models/blob/main/allennlp_models/vision/dataset_readers/vqav2.py

```

19 path2id = {images_path + "/" + file: id_from_filename(file) for file
    in file_names}
20 id2path = {v: k for k, v in path2id.items()}

```

Code A.3: VQAv2 association.

Following both scenarios, the dictionary *id2path* encompasses all the ID-path associations crucial for the proper operation of the datasets described subsequently.

A.2 NLVR2

```

1 class NLVR2Dataset(torch.utils.data.Dataset):
2     def __init__(self, annotations, processor, id2path):
3         self.annotations = annotations
4         self.processor = processor
5         self.id2path = id2path
6
7     def __len__(self):
8         return len(self.annotations)
9
10    def __getitem__(self, idx):
11        annotation = self.annotations[idx]
12
13        image_id = annotation['identifier']
14
15        sentence = annotation['sentence']
16
17        label = torch.tensor(annotation['label'])
18        left_image = Image.open(self.id2path[image_id]+'-img0.png')
19        right_image = Image.open(self.id2path[image_id]+'-img1.png')
20
21        if left_image.mode != 'RGB':
22            left_image = left_image.convert('RGB')
23        if right_image.mode != 'RGB':
24            right_image = right_image.convert('RGB')
25
26        left_image = left_image.resize((400, 400))
27        right_image = right_image.resize((400, 400))
28
29        encoding = self.processor([left_image, right_image], sentence
, padding="max_length", truncation=True, return_tensors="pt")

```

A.3. VE

```
30
31     # simple binary tensor
32     encoding['labels'] = label
33
34     for k, v in encoding.items():
35         encoding[k] = v.squeeze()
36
37     return encoding
```

Code A.4: NLVR2 dataset

A.3 VE

```
1 class VEDataset(torch.utils.data.Dataset):
2     def __init__(self, annotations, processor, id2path):
3         self.annotations = annotations
4         self.processor = processor
5         self.id2path = id2path
6
7     def __len__(self):
8         return len(self.annotations)
9
10    def __getitem__(self, idx):
11        annotation = self.annotations[idx]
12
13        image_id = annotation['image_id']
14        print(image_id)
15
16        sentence = annotation['sentence']
17
18        label = torch.zeros(3)
19        index = annotation['label']
20        label[index] = 1
21
22        image = Image.open(self.id2path[image_id])
23
24        if image.mode != 'RGB':
25            image = image.convert('RGB')
26
27        encoding = self.processor(image, sentence, padding="
max_length", truncation=True, return_tensors="pt")
28
```

```

29     # one-hot encoding tensor
30     encoding['labels'] = label
31
32     for k, v in encoding.items():
33         encoding[k] = v.squeeze()
34
35     return encoding

```

Code A.5: VE dataset.

A.4 LM

```

1 class LMDataset(torch.utils.data.Dataset):
2     def __init__(self, annotations, processor, id2path, training=
  False):
3         self.training = training
4         self.annotations = annotations
5         self.processor = processor
6         self.id2path = id2path
7
8     def __len__(self):
9         return len(self.annotations)
10
11    def __getitem__(self, idx):
12        annotation = self.annotations[idx]
13
14        image_id = annotation['image_id']
15
16        print(image_id)
17
18        sentence = annotation['masked_sentence']
19
20        label = self.processor.tokenizer(annotation['sentence'],
  padding="max_length", truncation=True, return_tensors="pt")
21
22        # to ignore special tokens different from [MASK] during loss
  calculation
23        if self.training:
24            label['input_ids'] = torch.where(label['input_ids'] == 0,
  torch.tensor(-100), label['input_ids'])
25            label['input_ids'] = torch.where(label['input_ids'] ==
  101, torch.tensor(-100), label['input_ids'])

```

A.5. VQAV2

```
26         label['input_ids'] = torch.where(label['input_ids'] ==
27         102, torch.tensor(-100), label['input_ids'])
28
29         image = Image.open(self.id2path[image_id])
30
31         if image.mode != 'RGB':
32             image = image.convert('RGB')
33
34         encoding = self.processor(image, sentence, padding="
35         max_length", truncation=True, return_tensors="pt")
36
37         # non-masked sequence of tokens
38         encoding['labels'] = label.input_ids.clone()
39
40         for k, v in encoding.items():
41             encoding[k] = v.squeeze()
42
43     return encoding
```

Code A.6: LM dataset.

A.5 VQAv2

A.5.1 STANDARD

```
1 class VQAv2Dataset(torch.utils.data.Dataset):
2     def __init__(self, annotations, processor, id2path, config=None):
3         self.annotations = annotations
4         self.processor = processor
5         self.id2path = id2path
6         self.config = config
7
8     def __len__(self):
9         return len(self.annotations)
10
11     def __getitem__(self, idx):
12         annotation = self.annotations[idx]
13
14         image_id = annotation['image_id']
15
16         sentence = annotation['question']
17
```

```

18     labels = torch.zeros(len(self.config.id2label))
19     for label, score in zip(annotation["labels"], annotation["
scores"]):
20         labels[label] = score
21
22     image = Image.open(self.id2path[image_id])
23
24     if image.mode != 'RGB':
25         image = image.convert('RGB')
26
27     encoding = self.processor(image, sentence, padding="
max_length", truncation=True, return_tensors="pt")
28
29     # one-hot encoding tensor
30     encoding['labels'] = labels
31
32     for k, v in encoding.items():
33         encoding[k] = v.squeeze()
34
35     return encoding

```

Code A.7: VQAv2 standard dataset.

A.5.2 RESNETV2 FEATURES

```

1 class VQAv2DatasetCNN(torch.utils.data.Dataset):
2     def __init__(self, annotations, vilt_processor, cnn_processor,
id2path, config=None):
3         self.annotations = annotations
4         self.vilt_processor = vilt_processor
5         self.cnn_processor = cnn_processor
6         self.id2path = id2path
7         self.config = config
8
9     def __len__(self):
10        return len(self.annotations)
11
12    def __getitem__(self, idx):
13        annotation = self.annotations[idx]
14
15        image_id = annotation['image_id']
16
17        sentence = annotation['question']

```

A.5. VQAV2

```
18
19     labels = torch.zeros(len(self.config.id2label))
20     for label, score in zip(annotation["labels"], annotation["
21 scores"]):
22         labels[label] = score
23
24     image = Image.open(self.id2path[image_id])
25
26     if image.mode != 'RGB':
27         image = image.convert('RGB')
28
29     encoding = self.vilt_processor.tokenizer(sentence, padding="
30 max_length", truncation=True, return_tensors="pt")
31
32     encoding['pixel_values'] = self.cnn_processor(image,
33 return_tensors="pt")['pixel_values']
34
35     encoding['labels'] = labels
36
37     for k, v in encoding.items():
38         encoding[k] = v.squeeze()
39
40     return encoding
```

Code A.8: VQAv2 dataset with ResNetV2 features.

A.5.3 SQUEEZE_{NET} FEATURES

```
1 class VQAv2DatasetCNNV3(torch.utils.data.Dataset):
2     def __init__(self, annotations, vilt_processor, cnn_processor,
3 id2path, config=None):
4         self.annotations = annotations
5         self.vilt_processor = vilt_processor
6         self.cnn_processor = cnn_processor
7         self.id2path = id2path
8         self.config = config
9
10    def __len__(self):
11        return len(self.annotations)
12
13    def __getitem__(self, idx):
14        annotation = self.annotations[idx]
```

```

15     image_id = annotation['image_id']
16
17     sentence = annotation['question']
18
19     labels = torch.zeros(len(self.config.id2label))
20     for label, score in zip(annotation["labels"], annotation["
scores"]):
21         labels[label] = score
22
23     image = Image.open(self.id2path[image_id])
24
25     if image.mode != 'RGB':
26         image = image.convert('RGB')
27
28     encoding = self.vilt_processor.tokenizer(sentence, padding="
max_length", truncation=True, return_tensors="pt")
29
30     encoding['pixel_values'] = self.cnn_processor(image)
31
32     encoding['labels'] = labels
33
34     for k, v in encoding.items():
35         encoding[k] = v.squeeze()
36
37     return encoding

```

Code A.9: VQAv2 dataset with SqueezeNet features.

A.6 DATALOADER

```

1 from torch.utils.data import DataLoader
2
3 def collate_fn(batch, processor):
4     input_ids = [item['input_ids'] for item in batch]
5     pixel_values = [item['pixel_values'] for item in batch]
6     attention_mask = [item['attention_mask'] for item in batch]
7     token_type_ids = [item['token_type_ids'] for item in batch]
8     labels = [item['labels'] for item in batch]
9
10    # create padded pixel values and corresponding pixel mask
11    encoding = processor.image_processor.pad(pixel_values,
return_tensors="pt")

```

A.6. DATALOADER

```
12
13     # create new batch
14     batch = {'input_ids': torch.stack(input_ids), 'attention_mask':
15             torch.stack(attention_mask),
16             'token_type_ids': torch.stack(token_type_ids), '
17             pixel_values': encoding['pixel_values'],
18             'pixel_mask': encoding['pixel_mask'], 'labels': torch.
19             stack(labels)}
20
21     return batch
22
23 # example of use
24 dataloader = DataLoader(train_dataset, collate_fn=lambda batch:
25                         collate_fn(batch, processor), batch_size=batch_size, shuffle=True)
```

Code A.10: DataLoader to train ViLT in batches.



Use of SAM

In Chapter 3, SAM were utilized for image segmentation. Detailed installation instructions for SAM can be found on the original GitHub repository¹.

To leverage the power of SAM without suffering from long computational times, segmented images were precomputed. These precomputed segmented images served as inputs for subsequent processes.

The Supervision library was employed to facilitate the creation of instances. More comprehensive information regarding the usage of this library can be found in the original documentation². Utilizing this library, the raw outputs of SAM were gathered and consolidated to generate the final segmented image.

Below is an example demonstrating the segmentation of a single image:

```
1 import torch
2 import cv2
3 from PIL import Image
4 import supervision as sv
5 from segment_anything import sam_model_registry
6 from segment_anything import SamAutomaticMaskGenerator
7
8 # path to sam weights
9 weights = "sam/weights/path"
10 # path to image to segment
11 image_path = "original/image/path"
12
```

¹<https://github.com/facebookresearch/segment-anything>

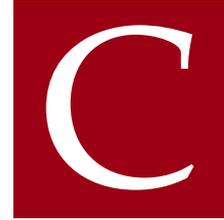
²<https://supervision.roboflow.com/latest/detection/core>

```

13 device = torch.device('cuda:0' if torch.cuda.is_available() else '
    cpu')
14 # size of SAM used, have to be the same as the weights
15 MODEL_TYPE = "vit_h"
16
17 # loading the model
18 sam = sam_model_registry[MODEL_TYPE](checkpoint=weights)
19 sam.to(device)
20
21 mask_generator = SamAutomaticMaskGenerator(sam)
22
23 image_bgr = cv2.imread(image_path)
24 image_rgb = cv2.cvtColor(image_bgr, cv2.COLOR_BGR2RGB)
25 result = mask_generator.generate(image_rgb)
26
27 mask_annotator = sv.MaskAnnotator(color_lookup = sv.ColorLookup.
    INDEX)
28
29 detections = sv.Detections.from_sam(result)
30
31 annotated_image = mask_annotator.annotate(scene=image_bgr.copy(),
    detections=detections)
32
33 with sv.ImageSink(target_dir_path=segmented/image/path) as sink:
34     sink.save_image(image=annotated_image, image_name=
    segmented_image_name)

```

Code B.1: SAM use example.



ViLT-CNN hybrid models

In this appendix, code snippets for the ViLT-CNN hybrid models are provided. These models are designed to work specifically with the architectures outlined but can serve as a baseline if different components are desired.

C.1 LINEAR PROJECTION

Using linear projection custom features are created. To pass them directly to ViLT *image_embeds* are used instead of *pixel_values*.

```
1 import torch
2 import torch.nn as nn
3 from transformers import ViltConfig, ViltProcessor,
4   ViltForQuestionAnswering
5
6 from transformers import AutoImageProcessor, BitModel
7
8 class ViLTwithCNNfeatures(nn.Module):
9     def __init__(self, cnn, vilt):
10         super(ViLTwithCNNfeatures, self).__init__()
11
12         self.cnn = cnn
13
14         self.vilt = vilt
15
16         # change kernel_size and stride to change the number of
17         # features obtained
```

C.2. RESHAPING

```
15     self.projection = nn.Conv2d(1024, 768, kernel_size=16, stride
16     =16)
17     def forward(self, input_ids, token_type_ids, attention_mask,
18     pixel_values, labels):
19         # to obtain the internal layer's output
20         cnn_outputs = self.cnn(pixel_values, output_hidden_states=
21     True)
22         cnn_features = cnn_outputs.hidden_states[3]
23
24         image_embeds = self.projection(cnn_features).flatten(2).
25     transpose(1, 2)
26         # to use custom features a pixel_mask have to be created
27     manually
28         pixel_mask = torch.ones(image_embeds.size()[:2]).to(
29     image_embeds.device)
30
31         output = self.vilt(input_ids = input_ids, token_type_ids=
32     token_type_ids, attention_mask=attention_mask,
33     image_embeds = image_embeds, pixel_mask =
34     pixel_mask, labels=labels)
35
36     return output
37
38 vilt = ViltForQuestionAnswering.from_pretrained("dandelin/vilt-b32-
39     mlm", config=config)
40 cnn = BitModel.from_pretrained("google/bit-50")
41
42 model = ViLTwithCNNfeatures(vilt=vilt, cnn=cnn)
```

Code C.1: Vilt-CNN v1.

C.2 RESHAPING

The internal layer feature maps of the CNN are extracted and reshaped using an upsampling block, resulting in an input size equivalent to an image, which can then be passed to ViLT as *pixel_values*.

C.2.1 RESNETV2

```
1 import torch
```

```

2 import torch.nn as nn
3 from transformers import ViltConfig, ViltProcessor,
  ViltForQuestionAnswering
4 from transformers import AutoImageProcessor, BitModel
5
6 class ViLTwithCNNfeaturesV2(nn.Module):
7     def __init__(self, cnn, vilt):
8         super(ViLTwithCNNfeaturesV2, self).__init__()
9
10        self.cnn = cnn
11
12        self.vilt = vilt
13
14        # upsampling block
15        self.up = nn.Sequential(
16            nn.ConvTranspose2d(in_channels=1024, out_channels=512,
kernel_size=4, stride=2, padding=1),
17            nn.ReLU(),
18            nn.ConvTranspose2d(in_channels=512, out_channels=256,
kernel_size=4, stride=2, padding=1),
19            nn.ReLU(),
20            nn.ConvTranspose2d(in_channels=256, out_channels=128,
kernel_size=4, stride=2, padding=1),
21            nn.ReLU(),
22            nn.ConvTranspose2d(in_channels=128, out_channels=3,
kernel_size=4, stride=2, padding=1),
23            nn.ReLU(),
24        )
25
26        def forward(self, input_ids, token_type_ids, attention_mask,
pixel_values, labels):
27            # to obtain the internal layer's output
28            cnn_outputs = self.cnn(pixel_values, output_hidden_states=
True)
29            cnn_features = cnn_outputs.hidden_states[3]
30
31            image_embeds = self.up(cnn_features)
32
33            output = self.vilt(input_ids=input_ids, token_type_ids=
token_type_ids, attention_mask=attention_mask,
34                               pixel_values=image_embeds, labels=labels)
35
36            return output

```

C.2. RESHAPING

```
37
38 vilt = ViltForQuestionAnswering.from_pretrained("dandelin/vilt-b32-
      mlm", config=config)
39 cnn = BitModel.from_pretrained("google/bit-50")
40
41 model = ViLTwithCNNfeaturesV2(vilt=vilt, cnn=cnn)
```

Code C.2: Vilt-CNN v2.

C.2.2 SQUEEZE NET

```
1 import torch
2 import torch.nn as nn
3 from transformers import ViltConfig, ViltProcessor,
      ViltForQuestionAnswering
4
5 class ViLTwithCNNfeaturesV3(nn.Module):
6     def __init__(self, cnn, vilt):
7         super(ViLTwithCNNfeaturesV3, self).__init__()
8
9         # to obtain the internal layer's output
10        self.cnn_features = nn.Sequential(
11            *list(cnn.features.children())[:-2]
12        )
13
14        # upsampling block
15        self.up = nn.Sequential(
16            nn.ConvTranspose2d(in_channels=512, out_channels=256,
17                               kernel_size=4, stride=2, padding=1),
18            nn.ReLU(),
19            nn.ConvTranspose2d(in_channels=256, out_channels=128,
20                               kernel_size=4, stride=2, padding=1),
21            nn.ReLU(),
22            nn.ConvTranspose2d(in_channels=128, out_channels=64,
23                               kernel_size=4, stride=2, padding=1),
24            nn.ReLU(),
25            nn.ConvTranspose2d(in_channels=64, out_channels=3,
26                               kernel_size=4, stride=2, padding=1),
27            nn.ReLU(),
28        )
29
30        self.vilt = vilt
```

```
28     def forward(self, input_ids, token_type_ids, attention_mask,
29 pixel_values, labels):
30         cnn_outputs = self.cnn_features(pixel_values)
31
32         image_embeds = self.up(cnn_outputs)
33
34         output = self.vilt(input_ids=input_ids, token_type_ids=
35 token_type_ids, attention_mask=attention_mask,
36                             pixel_values=image_embeds, labels=labels)
37
38         return output
39
40 vilt = ViltForQuestionAnswering.from_pretrained("dandelin/vilt-b32-
41 mlm", config=config)
42 cnn = torch.hub.load('pytorch/vision:v0.10.0', 'squeezenet1_0',
43                     weights=True)
44 model = ViLTwithCNNfeaturesV3(vilt=vilt, cnn=cnn)
```

Code C.3: Vilt-CNN v3.



Use of LoRA

In Chapter 6, the implementation of LoRA is explored using Hugging Face's PEFT library¹, which simplifies its application. This library offers various methods for efficient training and includes the LoRA implementation² for ease of use. An example of its usage is provided along with a list of its main parameters:

- *r* and *lora_alpha*: specify the values of the two parameters.
- *target_modules*: a list of module names to which LoRA will be applied.
- *bias*: determines whether model biases can be updated during training.
- *modules_to_save*: specifies which modules without LoRA are allowed to update during training.

```
1 import torch
2 from transformers import ViltConfig, ViltProcessor,
  ViltForQuestionAnswering
3 from peft import LoraConfig, get_peft_model
4
5 # To obtain ViLT with pretrained weights
6 model = ViltForQuestionAnswering.from_pretrained(
  pretrained_model_name_or_path="dandelin/vilt-b32-mlm")
7
8 config = LoraConfig(
9     r=16,
10    lora_alpha=16,
```

¹<https://huggingface.co/docs/peft/index>

²https://huggingface.co/docs/peft/package_reference/lora

```

11     target_modules=["query", "value"],
12     lora_dropout=0.1,
13     bias="none",
14     modules_to_save=["classifier"],
15 )
16
17 lora_model = get_peft_model(model, config)

```

Code D.1: LoRA use example.

With the PEFT library, applying LoRA is straightforward. It's worth noting that Hugging Face's models are well-suited for this library's application, but custom models can also be used following the guide provided³. To determine the correct module name for applying LoRA, one can refer to models similar to the one being used, as demonstrated in this thesis with ViT, or print a list of parameter names using the method outlined below.

```

1 print([(n, type(m)) for n, m in MLP().named_modules()])

```

Code D.2: Printing name of modules.

³https://huggingface.co/docs/peft/developer_guides/custom_models

References

- [1] Armen Aghajanyan, Luke Zettlemoyer, and Sonal Gupta. *Intrinsic Dimensionality Explains the Effectiveness of Language Model Fine-Tuning*. 2020. arXiv: 2012.13255 [cs.LG].
- [2] Niki Amini-Naieni et al. *Open-world Text-specified Object Counting*. 2023. arXiv: 2306.01851 [cs.CV].
- [3] Stanislaw Antol et al. "VQA: Visual Question Answering". In: *International Conference on Computer Vision (ICCV)*. 2015.
- [4] Samuel R. Bowman et al. *A large annotated corpus for learning natural language inference*. 2015. arXiv: 1508.05326 [cs.CL].
- [5] Yen-Chun Chen et al. *UNITER: UNiversal Image-Text Representation Learning*. 2020. arXiv: 1909.11740 [cs.CV].
- [6] Wei-Lin Chiang et al. *Vicuna: An Open-Source Chatbot Impressing GPT-4 with 90%* ChatGPT Quality*. 2023. URL: <https://lmsys.org/blog/2023-03-30-vicuna/>.
- [7] Ekin D. Cubuk et al. *RandAugment: Practical automated data augmentation with a reduced search space*. 2019. arXiv: 1909.13719 [cs.CV].
- [8] Jacob Devlin et al. *BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding*. 2019. arXiv: 1810.04805 [cs.CL].
- [9] Alexey Dosovitskiy et al. *An Image is Worth 16x16 Words: Transformers for Image Recognition at Scale*. 2021. arXiv: 2010.11929 [cs.CV].
- [10] Yash Goyal et al. "Making the V in VQA Matter: Elevating the Role of Image Understanding in Visual Question Answering". In: *Conference on Computer Vision and Pattern Recognition (CVPR)*. 2017.
- [11] Kaiming He et al. *Identity Mappings in Deep Residual Networks*. 2016. arXiv: 1603.05027 [cs.CV].

REFERENCES

- [12] Jeremy Howard and Sebastian Ruder. *Universal Language Model Fine-tuning for Text Classification*. 2018. arXiv: 1801.06146 [cs.CL].
- [13] Edward J. Hu et al. *LoRA: Low-Rank Adaptation of Large Language Models*. 2021. arXiv: 2106.09685 [cs.CL].
- [14] Forrest N. Iandola et al. *SqueezeNet: AlexNet-level accuracy with 50x fewer parameters and <0.5MB model size*. 2016. arXiv: 1602.07360 [cs.CV].
- [15] Ivana Kajić and Aida Nematzadeh. “Probing Representations of Numbers in Vision and Language Models”. In: *SVRHM 2022 Workshop @ NeurIPS*. 2022. URL: <https://openreview.net/forum?id=01hQQ16Lc9M>.
- [16] Asifullah Khan et al. “A survey of the vision transformers and their CNN-transformer based variants”. In: *Artificial Intelligence Review* 56.S3 (Oct. 2023), pp. 2917–2970. ISSN: 1573-7462. DOI: 10.1007/s10462-023-10595-0. URL: <http://dx.doi.org/10.1007/s10462-023-10595-0>.
- [17] Wonjae Kim, Bokyoung Son, and Ildoo Kim. *ViLT: Vision-and-Language Transformer Without Convolution or Region Supervision*. 2021. arXiv: 2102.03334 [stat.ML].
- [18] Alexander Kirillov et al. *Segment Anything*. 2023. arXiv: 2304.02643 [cs.CV].
- [19] Alexander Kolesnikov et al. *Big Transfer (BiT): General Visual Representation Learning*. 2020. arXiv: 1912.11370 [cs.CV].
- [20] Ranjay Krishna et al. *Visual Genome: Connecting Language and Vision Using Crowdsourced Dense Image Annotations*. 2016. arXiv: 1602.07332 [cs.CV].
- [21] Feng Li et al. *Vision-Language Intelligence: Tasks, Representation Learning, and Large Models*. 2022. arXiv: 2203.01922 [cs.CV].
- [22] Zewen Li et al. *A Survey of Convolutional Neural Networks: Analysis, Applications, and Prospects*. 2020. arXiv: 2004.02806 [cs.CV].
- [23] Tsung-Yi Lin et al. *Microsoft COCO: Common Objects in Context*. 2015. arXiv: 1405.0312 [cs.CV].
- [24] Haotian Liu et al. *Visual Instruction Tuning*. 2023. arXiv: 2304.08485 [cs.CV].
- [25] Ilya Loshchilov and Frank Hutter. *Decoupled Weight Decay Regularization*. 2019. arXiv: 1711.05101 [cs.LG].
- [26] Jiasen Lu et al. *ViLBERT: Pretraining Task-Agnostic Visiolinguistic Representations for Vision-and-Language Tasks*. 2019. arXiv: 1908.02265 [cs.CV].

- [27] Loris Nanni et al. “Improving Existing Segmentators Performance with Zero-Shot Segmentators”. In: *Entropy* 25.11 (2023). ISSN: 1099-4300. DOI: 10.3390/e25111502. URL: <https://www.mdpi.com/1099-4300/25/11/1502>.
- [28] Vicente Ordonez, Girish Kulkarni, and Tamara L. Berg. “Im2Text: Describing Images Using 1 Million Captioned Photographs”. In: *Neural Information Processing Systems (NIPS)*. 2011.
- [29] Roni Paiss et al. *Teaching CLIP to Count to Ten*. 2023. arXiv: 2302.12066 [cs.CV].
- [30] Letitia Parcalabescu et al. *Seeing past words: Testing the cross-modal capabilities of pretrained V&L models on counting tasks*. 2021. arXiv: 2012.12352 [cs.CV].
- [31] Alec Radford et al. *Learning Transferable Visual Models From Natural Language Supervision*. 2021. arXiv: 2103.00020 [cs.CV].
- [32] Olaf Ronneberger, Philipp Fischer, and Thomas Brox. *U-Net: Convolutional Networks for Biomedical Image Segmentation*. 2015. arXiv: 1505.04597 [cs.CV].
- [33] Rico Sennrich, Barry Haddow, and Alexandra Birch. *Neural Machine Translation of Rare Words with Subword Units*. 2016. arXiv: 1508.07909 [cs.CL].
- [34] Pierre Sermanet et al. *OverFeat: Integrated Recognition, Localization and Detection using Convolutional Networks*. 2014. arXiv: 1312.6229 [cs.CV].
- [35] Piyush Sharma et al. “Conceptual Captions: A Cleaned, Hypernymed, Image Alt-text Dataset For Automatic Image Captioning”. In: *Proceedings of ACL*. 2018.
- [36] Alane Suhr et al. *A Corpus for Reasoning About Natural Language Grounded in Photographs*. 2019. arXiv: 1811.00491 [cs.CL].
- [37] Alane Suhr et al. “A Corpus of Natural Language for Visual Reasoning”. In: *Annual Meeting of the Association for Computational Linguistics*. 2017. URL: <https://api.semanticscholar.org/CorpusID:19435386>.
- [38] Tristan Thrush et al. *Winoground: Probing Vision and Language Models for Visio-Linguistic Compositionality*. 2022. arXiv: 2204.03162 [cs.CV].
- [39] Hugo Touvron et al. *LLaMA: Open and Efficient Foundation Language Models*. 2023. arXiv: 2302.13971 [cs.CL].

REFERENCES

- [40] Ashish Vaswani et al. *Attention Is All You Need*. 2023. arXiv: 1706.03762 [cs.CL].
- [41] Peng Wang et al. *OFA: Unifying Architectures, Tasks, and Modalities Through a Simple Sequence-to-Sequence Learning Framework*. 2022. arXiv: 2202.03052 [cs.CV].
- [42] Ning Xie et al. *Visual Entailment: A Novel Task for Fine-Grained Image Understanding*. 2019. arXiv: 1901.06706 [cs.CV].
- [43] Yujia Xie et al. *A Fast Proximal Point Method for Computing Exact Wasserstein Distance*. 2019. arXiv: 1802.04307 [stat.ML].
- [44] Peter Young et al. "From image descriptions to visual denotations: New similarity metrics for semantic inference over event descriptions". In: *Transactions of the Association for Computational Linguistics 2* (2014). Ed. by Dekang Lin, Michael Collins, and Lillian Lee, pp. 67–78. DOI: 10.1162/tacl_a_00166. URL: <https://aclanthology.org/Q14-1006>.
- [45] Shengyu Zhang et al. *Instruction Tuning for Large Language Models: A Survey*. 2023. arXiv: 2308.10792 [cs.CL].

Acknowledgments

I would like to express my gratitude to my supervisors. Professor Giorgio Satta for his extraordinary guidance and expertise thought this dissertation. I am equally grateful to Professor Loris Nanni for his valuable contributions and suggestions for the various experiments of this work. Their insights was fundamental for the realization of this thesis.

Thanks to Marina for her unwavering support. Her critical thinking have been invaluable in helping me regain reason during challenging moments. I am grateful to my family for providing me with the opportunity of studying, even during the most challenging times.

Thanks to my fellow students Pietro, Marco and Paolo, who have accompanied me through these years of exams, projects and studies. they made my time in Padova more enjoyable. I extend my gratitude to all my friends, even if they were unaware of it, their presence helped me alleviate problems that troubled me at the time.