ALMA MATER STUDIORUM – UNIVERSITÀ DI BOLOGNA
CAMPUS DI CESENA

Scuola di Scienze
Corso di Laurea in Ingegneria e Scienze Informatiche

PREVISIONE DEL SUCCESSO DI PRODOTTI DI MODA PRIMA DELLA COMMERCIALIZZAZIONE: UN NUOVO DATASET E MODELLO DI VISION-LANGUAGE TRANSFORMER

Elaborato in
Text Mining

Relatore
Prof. Gianluca Moro

Co-relatore
Dott. Stefano Salvatori

Presentata da
Matteo Andruccioli

Seconda Sessione di Laurea
Anno Accademico 2020 – 2021

PAROLE CHIAVE

Cross-modal Deep Neural Networks

Vision-and-Language Transformer

Transformer Based Models

Market Success Prediction

Fashion Industry

A chiunque mi sia stato vicino,
e mi abbia aiutato a raggiungere questo traguardo.

Indice

Elenco delle figure

Introduzione

Il mondo del commercio online è in rapida espansione da oltre dieci anni, la pandemia che ha colpito tutto il globo negli ultimi due anni ha rafforzato questa tendenza. Stando a dati OCSE [2] il numero di acquisti online al dettaglio effettuati in Europa è incrementato del 30% in 12 mesi (Aprile 2019 - Aprile 2020), aumenti importanti sono stati riscontrati anche nel mercato asiatico e in quello americano. Con l'aumento del numero di acquisti si è verificato anche un aumento dei commercianti che si lanciano sul mercato online. Una delle esigenze di chi vende prodotti, sopratutto in settori come quello della moda, è quella di prevedere quale prodotto avrà successo: questo è di grande aiuto al fine di massimizzare le possibilità di guadagno sui prodotti con le migliori prospettive di vendita, ad esempio attraverso una adeguata campagna pubblicitaria o preparando quantità di prodotto sufficienti a soddisfare la richiesta. Al contrario, sapendo con il giusto anticipo che un prodotto non avrà successo è possibile ridurre le risorse che si intende investire su di esso e quindi minimizzare le spese. Fino a pochi anni fa valutazioni di questo tipo potevano essere effettuate solo facendo provare il prodotto ad un gruppo di tester oppure dopo aver messo il prodotto sul mercato, basandosi sull'andamento delle vendite riscontrato nei primi giorni dalla pubblicazione. Queste tecniche non sono però esenti da lati negativi: effettuare focus group e avere a disposizione un numero significativo di persone che fornisca feedback su prodotti può essere costoso e i feedback possono facilmente essere viziati. Allo stesso tempo effettuare una stima del successo ottenuto dal prodotto a posteriori di un primo periodo di vendita può essere utile per eventuali investimenti futuri ma non supporta il venditore nel primo investimento, necessariamente effettuato prima che il prodotto sia disponibile al pubblico. Sarebbe quindi utile per i commercianti avere a disposizione un software in grado, date le informazioni relative ad un prodotto, di determinare se questo avrà successo o meno. Le moderne tecniche di deep learning possono essere utilizzate per costruire programmi in grado di effettuare previsioni di questo tipo con un certo grado di confidenza. Per farlo sono tuttavia necessarie grandi moli di informazioni relative a prodotti di cui è noto il successo di vendita. Questi dati possono essere usati per addestrare un modello neurale ad eseguire la previsione. Una volta addestrato, il modello

sarà in grado di effettuare previsioni sul successo che verrà riscosso da prodotti non ancora pubblicati.
In questa tesi si affronta il problema di previsione del successo di vendita di un prodotto di Moda utilizzando modelli cross-modali basati su transformer. Tali modelli effettuano previsioni basandosi su informazioni note al venditore prima che l'articolo venga messo in commercio; le informazioni provengono da testi ed immagini. La tesi è divisa in due parti: nella prima parte vengono riportate le nozioni di teoria apprese durante lo studio di modelli sviluppati per l'analisi di dati non strutturati come testo ed immagini, con particolare attenzione al meccanismo di attention e ai transformer; nella seconda parte vengono utilizzate queste tecnologie per sviluppare un modello in grado di predire il successo di vendita di prodotti del settore moda.

Per prima cosa è stato affrontato lo studio di modelli Sequence to Sequence (seq2seq), ovvero quei modelli che prendono in ingresso e restituiscono dati sequenziali (stringhe di testo). Questo tipo di modelli sono stati sviluppati per risolvere problemi legati alla manipolazione del linguaggio come Machine Translation, Question Answering, Chat-bot, ecc.; più recentemente queste tecnologie sono state applicate anche a tipi di dato diversi come le immagini. Studiando questi modelli si può entrare in confidenza con l'architettura basata su Encoder-Decoder e con il meccanismo di Attention. Il meccanismo di attention ha consentito di sviluppare modelli come i Transformer, che non fanno uso della ricorsione e consentono quindi di parallelizzare operazioni eseguite durante i task e massimizzare l'efficienza, senza per questo rinunciare alle prestazioni. I Transformer introducono il meccanismo di self-attention e costituiscono la base dei più moderni modelli di deep-learning. Il primo modello studiato basato su Transformer è BERT: si tratta di un modello bidirezionale che esegue task con input e output testuali. BERT è un modello fortemente flessibile: a partire dal modello preaddestrato dagli autori è possibile ottenere un modello specializzato per uno specifico downstream task con pochi cicli di training (fine-tuning). I risultati ottenuti dai modelli addestrati su downstream tasks (come question answering, classificazione, scelta multipla, traduzione, ecc.) sono paragonabili a quelli ottenuti da modelli SOTA. Proprio per questa sua flessibilità, e anche per le buone performance dimostrate, BERT è stato preso come punto di riferimento per molti modelli successivi. A partire da BERT è stato sviluppato per esempio Vision Transformer (ViT), un modello che elabora immagini. ViT ha una struttura simile a quella di BERT ma, a differenza dei suoi predecessori, non utilizza reti convoluzionali pre-addestrate: le funzioni svolte da queste vengono assolte in ViT da visual embedding e meccanismo di attention. In questo modo, si ottiene un aumento dell'efficienza sia in termini di velocità, sia in termini di costo computazionale richiesto per l'addestramento; il tutto senza che si

verifichino degradi di performance. L'ultimo modello studiato, che costituisce la base del lavoro svolto in fase progettuale, è stato ViLT: Vision-and-Language Transformer. Si tratta di un modello cross-modale, che fa parte della famiglia dei modelli Vision-and-Language Pretraining (VLP). Questo modello prende in input una coppia testo-immagine ed è in grado di eseguire task che richiedono di estrarre e combinare informazioni provenienti dalle due modalità di input.

Per quanto riguarda la parte progettuale il primo passo è stato quello di reperire i dati necessari ad addestrare il modello che si vuole realizzare. Si è quindi scelto di utilizzare dati di prodotti venduti su Amazon relativi al settore della moda. I dati sono stati recuperati da un dataset chiamato *Amazon review dataset* messo a disposizione da Jianmo Ni. I dati sono quindi andati incontro a una fase di pulizia e preparazione che ha portato alla creazione di un dataset "pulito" ed etichettato. Si è quindi passati allo sviluppo del modello; sono state realizzate più implementazioni basate su ViLT. É stato necessario modificare quelle parti del modello che gestiscono il caricamento dati e l'esecuzione di downstream task. Il modello ViLT base è stato inoltre esteso affinchè fosse in grado di elaborare una quantità di input testuale più consistente, in questo modo è stato possibile fornire al modello maggiori informazioni. Sono stati sviluppati tre task: uno per il pretraining del modello e due downstream task. Il task per il pretraining consente al modello di apprendere a quale categoria (t-shirt, pantaloni, accessori, ecc.) appartenga un prodotto in base alle informazioni passate in input. Questo task è stato pensato per essere utilizzato in combinazione con task tipicamente utilizzati durante il preaddestramento di modelli cross-modali, come MLM e ITM. I due task downstream mirano invece a determinare il successo di vendita che verrà riscosso da un prodotto a partire dalle altre informazioni note su di esso; uno dei due task effettua classificazione multiclass single-label, l'altro è un task di regressione. É stato necessario effettuare molteplici addestramenti per determinare quali parametri utilizzare per l'addestramento dei modelli e quali informazioni fornire in input. In chiusura alla tesi viene riportato un commento dei risultati ottenuti.

Capitolo 1

Stato dell'arte

1.1 Modelli Seq2Seq e Attention

Nella prima parte della tesi verranno presi in considerazione gli aspetti teorici alla base del lavoro svolto. L'idea è quella di fornire al lettore una visione generale delle conoscenze di cui è necessario disporre per comprendere e sviluppare il progetto. Riporterò in questo e nei prossimi capitoli le nozioni di teoria che ho appreso durante lo svolgimento della tesi. Assumerò siano note al lettore alcune conoscenze elementari relative al Deep Learning, come: il funzionamento di neurone, reti neurali feed forward(FNN), reti neurali ricorrenti(RNN) e le varianti architetturali Long Short-Term Memory(LSTM) e Gated Recurrent Units (GRU).

1.1.1 Modelli Sequence to Sequence

I modelli sequence to sequence, spesso abbreviati nella formula seq2seq, sono una classe di architetture basate su RNN solitamente utilizzati nella risoluzione di problemi legati alla manipolazione del linguaggio come *Machine Translation*, *Question Answering*, creazione di *Chat-bot*, *Text Summarization*, ecc. Sono chiamati sequence to sequence in quanto in ciascuno di questi problemi il modello deve prendere in ingresso, elaborare e restituire dei dati sequenziali. La lunghezza di sequenza in uscita non deve necessariamente essere uguale a quella della sequenza in ingresso, il più delle volte non lo è.

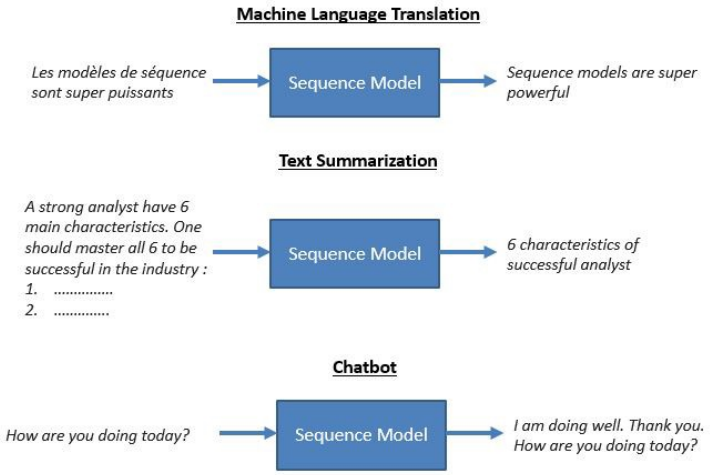


Figura 1.1: Esempio task che utilizzano modelli Sequence to Sequence

Di seguito riporto la descrizione ad alto livello di come può essere affrontato un problema di Machine Translation, ovvero traduzione di un testo da un linguaggio ad un altro, in particolare, in questo, caso dall'inglese all'italiano. In questo modo sarà possibile da un lato descrivere come funzionano i modelli Sequence to Sequence e in secondo luogo introdurre le architetture basate su encoder-decoder e il concetto di attention, che saranno due aspetti fondamentali nelle tecnologie studiate durante lo sviluppo del progetto.

1.1.2 Encoder-Decoder Architecture

L'architettura più comunemente utilizzata per costruite modelli Sequence to Sequence è la cosidetta *Encoder-Decoder Architecture*.

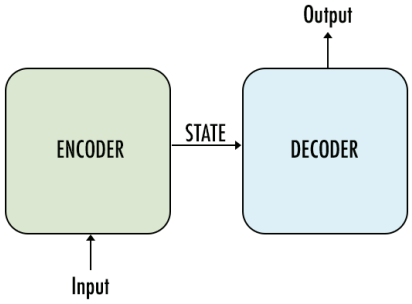


Figura 1.2: Encoder-Decoder Architecture

Di seguito alcuni aspetti caratterizzanti dell'architettura:

- sia encoder che decoder sono solitamente modelli LSTM o GRU

- l'**encoder** ha il compito di leggere la sequenza in input e riassumere l'informazione all'interno di un un vettore, che quindi contiene lo "stato interno" del modello. Tale informazione riassuntiva dell'input è anche detta *hidden state* e il vettore può essere chiamato *state vector* o *thought vector*. Lo stato iniziale dell'encoder è inizializzato ad un valore random o a 0. L'output dell'encoder viene scartato in quanto l'informazione a cui siamo interessati è contenuta nell'internal state, che al contrario viene preservato e dato in pasto al decoder

- il **decoder** viene inizializzato impostando come stato iniziale lo stato finale dell'encoder. A partire da questo, il decoder comincia a generare una sequenza di output. Nel caso del decoder siamo interessati alla sequenza prodotta come risultato e non allo stato finale. L'output del decoder verrà quindi conservato mentre il suo stato interno finale verrà scartato

- il decoder si comporta in modo leggermente differente durante l'addestramento e la fase di inferenza. Durante l'addestramento viene utilizzata una tecnica chiamata **Teacher Forcing** che aiuta a diminuire il tempo richiesto per l'addestramento. Il **Teacher Forcing** consiste nel fornire in input al decoder ad ogni istante temporale la parola che avrebbe dovuto predire nell'istante precedente, indipendentemente dal fatto che la predizione abbia prodotto la parola corretta o meno. Durante la fase di inferenza, ad ogni istante temporale t viene dato in input al decoder il risultato predetto all'istante t-1

- come intuizione si può pensare quindi che l'encoder abbia il compito di riassumere la sequenza in input nello *state vector* che viene poi dato in pasto al decoder, il quale a partire da questa conoscenza acquisita produce il risultato

Di seguito riporto un esempio illustrato del funzionamento di questa architettura in cui mostro come avviene la traduzione della frase inglese "the cat is on the table" nella corrispondente frase italiana "il gatto è sul tavolo".

Encoder

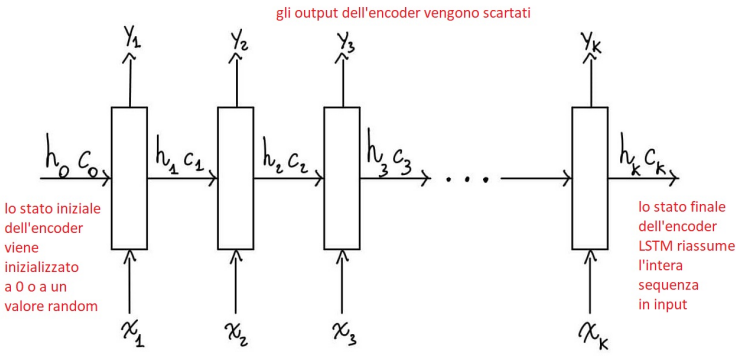


Figura 1.3: Struttura di un Encoder

La rete LSTM che implementa l'encoder legge i dati di una sequenza uno per volta; per cui se l'input è una sequenza lunga k, l'encoder completerà la lettura in k istanti temporali. Ad ogni istante temporale i abbiamo quindi:

- x_i: input al tempo i. Gli input sono costituiti da word embedding costituiti a partire dalla parola che deve essere data in pasto all'encoder. I word embedding hanno tutti la stessa lunghezza prestabilita

- h_i e c_i: LSTM mantiene due stati: h_i è l'hidden state, c_i è detto cell state. Combinati costituiscono lo stato interno della rete al tempo i. Nel caso avessimo usato GRU avremmo avuto solo h_i, ma il concetto sarebbe stato lo stesso. Lo stato interno serve a ricordare tutto ciò che è stato letto: quindi (h_1, c_1) ricorderà la sola prima parola, (h_k, c_k) ricorderà l'intera sequenza

- y_i: output al tempo i. Nel caso di word level language model y_i indica una distribuzione di probabilità calcolata sull'intero vocabolario utilizzando come funzione di attivazione softmax. Quindi ogni y_i è un vettore, con lunghezza pari al numero di parole nel vocabolario, i cui valori rappresentano una distribuzione di probabilità

Decoder

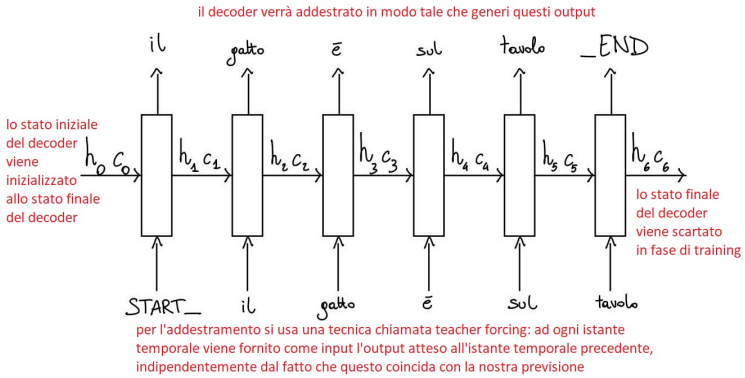


Figura 1.4: Decoder addestrato con la frase di esempio

Come parzialmente già spiegato, il decoder segue due comportamenti leggermente diversi in fase di Training e di Inference.
Di seguito alcuni aspetti che caratterizzano il decoder:

- come l'encoder, passa in rassegna alla sequenza in input parola per parola e allo stesso modo genera una sequenza in output aggiungendo un termine per volta

- l'aspetto fondamentale è che lo stato iniziale del decoder (h_0, c_0) venga inizializzato con i valori dello stato finale del dell'encoder. Intuitivamente questa operazione fornisce al decoder le informazioni di cui ha bisogno per generare l'output, nel nostro esempio effettuare la traduzione

- per quanto riguarda l'input fornito al decoder questo dipende dal fatto che sia in **Training Mode** o **Inference Mode**, ma in entrambi i casi avremo uno START_ token iniziale ed un _END token la cui generazione va a validare la stop condition che determina la fine del task di traduzione

- in **Training Mode** viene calcolata la loss function sugli output predetti ad ogni time step e gli errori vengono propagati indietro nel tempo in modo da aggiornare i parametri della rete

Inference Algorithm

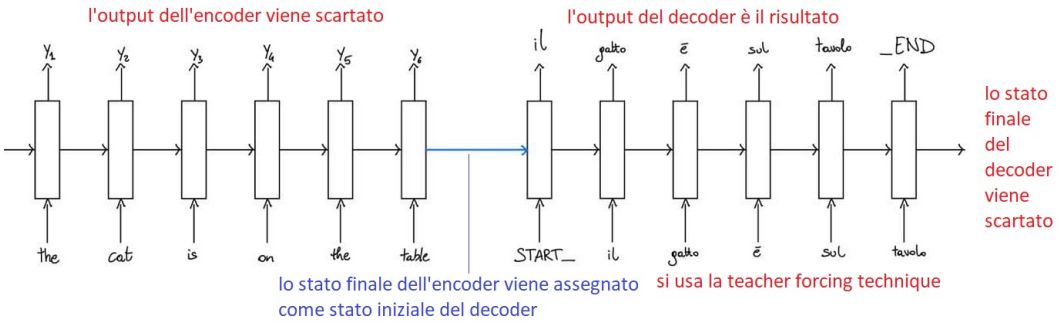


Figura 1.5: Architettura Encoder-Decoder con passaggio dell'hidden state

Durante la fase di inferenza viene generata una parola per volta, per cui il decoder viene chiamato in loop fino alla generazione dell' _END token e ad ogni iterazione viene generata una sola parola. Come mostrato in figura dalla freccia blu centrale, lo stato di partenza del decoder è inizializzato con i valori dello stato finale dell'encoder. Indipendentemente dalla frase che dovrà esser generata, il primo token passato in input al decoder sarà START_. Ad ogni istante temporale lo stato interno del decoder viene preservato e fornito in input al decoder all'istante temporale successivo, stessa cosa avviene per quanto riguarda la parola predetta.

1.1.3 Attention nei modelli Seq2Seq

Una delle idee che maggiormente hanno influenzato i modelli più recenti sviluppati in ambito Deep Learning è quella di **Attention** [3]. Il meccanismo di attention nasce nell'ambito della Neural Machine Translation per affiancare e migliorare le performance di soluzioni basate su modelli Sequence to Sequence, ma viene poi ripreso e migliorato in soluzioni successive.
In questa sezione riprendo in mano il problema di Neural Machine Translation illustrato nel paragrafo precedente per presentare i modelli Seq2Seq e mostro come quel modello possa essere esteso e migliorato innestando il meccanismo di attention. Questo mi permetterà di spiegare l'idea che sta dietro ad attention e che sarà alla base di strumenti di cui parlerò diffusamente nei prossimi capitoli come i transformer e Bert.

I modelli Seq2Seq necessitano di Attention

I modelli Seq2Seq sono solitamente caratterizzati da una architettura basata su Encoder-Decoder, dove il decoder elabora la sequenza in input codificando,

comprimendo e riassumendo l'informazione all'interno di un context vector (detto anche *thought vector*) di lunghezza predeterminata. Questa rappresentazione dovrebbe essere un buona sintesi della sequenza in input. Lo stato interno del decoder viene quindi inizializzato con questo context vector, che dovrebbe permettere di produrre l'output atteso.
Sebbene in linea teorica quanto descritto sia valido, si può dimostrare empiricamente che all'aumentare della lunghezza delle sequenze il sistema incontri una sempre maggiore difficoltà nel "ricordare" le informazioni apprese in fase di elaborazione dell'input. In particolare le maggiori complicazioni sono legate alla parte iniziale della sequenza, che viene più facilmente dimenticata. Queste difficoltà sono legate alla lunghezza fissa del context vector. Il meccanismo di attention nasce proprio per risolvere questo problema.

Idea alla base del meccanismo di Attention

Nell'immagine 1.5 abbiamo visto come nei modelli Seq2Seq tradizionali tutti gli stati intermedi dell'encoder vengano scartati e venga utilizzato solo lo stato finale per inizializzare il decoder. Tale tecnica porta a buoni risultati per le sequenze più corte, ma all'aumentare della lunghezza, fare affidamento su un unico context vector per riassumente un'intera sequenza diventa un limite. L'idea per superare questo problema è quindi quella di utilizzare tutti gli stati intermedi dell'encoder per costruire il context vector richiesto dal decoder per generare la sequenza di output.

Significato del nome Attention

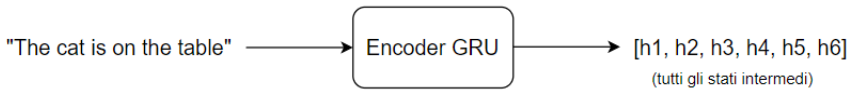


Figura 1.6: Encoding della frase di esempio [1]

Nell'esempio la frase "the cat is on the table" deve essere tradotta nella versione italiana "il gatto è sul tavolo". Le figure utilizzate in questa sessione faranno riferimento a encoder e decoder implementati con delle reti ricorrenti GRU.

Quando il decoder predice la parola "gatto" è ovvio che stiamo ottenendo il risultato della traduzione del termine "cat" della frase in input, in altre parole possiamo dire che stiamo dando *attenzione* alla parola "cat"; allo stesso modo quando viene generata la parola "tavolo" si starà prestando *attenzione* alla parte finale della sequenza in input. Il meccanismo di Attention mira proprio a

far si che il modello impari a prestare attenzione alla giusta parte dell'input in fase di inferenza.

Funzionamento del meccanismo di Attention

Come abbiamo già detto il decoder viene chiamato ripetutamente per generare la traduzione della frase in input. Lo schema riportato in seguito mostra le operazioni che devono essere eseguite ad ogni iterazione. L'esecuzione di una iterazione viene decomposta in 5 step, a seguito dell'immagine viene riportato un breve commento per ogni step.

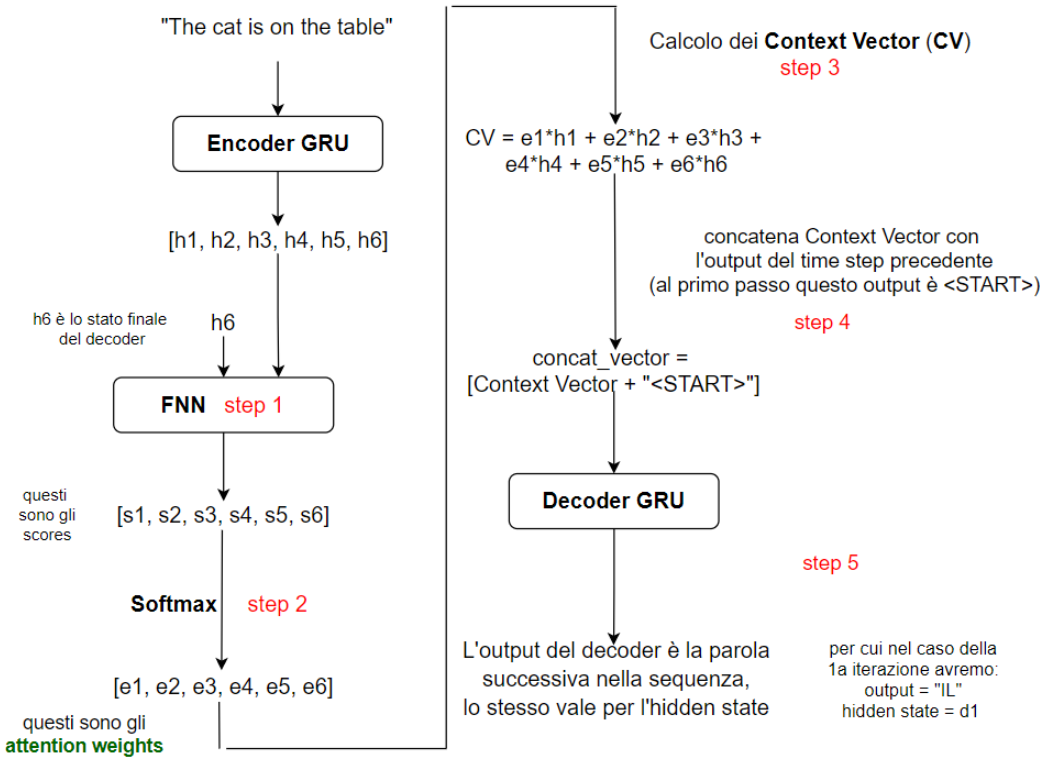


Figura 1.7: processo di NMT con uso del meccanismo di Attention [1]

Prima di iniziare la fase di decoding, è necessario effettuare l'encoding della sequenza in input; questo ci permetterà di ottenere il set degli stati interni dell'encoder (nel nostro caso $[h_1, h_2, h_3, h_4, h_5, h_6]$).
A questo punto l'idea è che ad ogni step la parola che deve essere predetta dipende contemporaneamente dallo stato corrente del decoder e dall'hidden state corrispondente dell'encoder.

Step 1 - calcolare un punteggio (score) per ogni stato dell'encoder

Dal momento che deve produrre la prima parola, il decoder non ha alcuno stato interno. Per questa ragione viene passato come stato interno l'ultimo dell'encoder, ovvero h_6. A questo punto, il decoder deve determinare a quali stati intermedi dell'encoder è necessario prestare più attenzione. Ad esempio al primo passo dovremo tradurre l'articolo "the" e ottenere come traduzione "il"; il decoder dovrebbe quindi capire di concentrarsi sugli stati dell'encoder [h_1, h_2] per effettuare questa traduzione. Questa comprensione di quali siano gli stati interni del decoder a cui deve essere prestata più attenzione viene fornita da una semplice rete neurale feed-forward (FNN). Durante la fase di addestramento, la FNN apprende come identificare gli stati più rilevanti dell'encoder. FNN restituisce questa informazione nella forma di punteggi (score) assegnati agli stati intermedi dell'encoder, abbiamo quindi [s_1, s_2, s_3, s_4, s_5, s_6]; i pesi saranno tanto maggiori quanto più si deve prestare attenzione all'hidden state corrispondente, ci aspettiamo quindi che per la prima predizione [s_1, s_2] abbiano valori più elevati di tutti gli altri elementi.

Step 2 - calcolare gli attention weights

Una volta generati gli score, viene applicata a questi valori una funzione softmax che ci restituisce i cosidetti *attention weights*. Si applica la softmax per ottenere valori e_i compresi nell'intervallo [0,1] e con somma = 1. Tali valori consentono una buona interpretazione probabilistica relativa all'attention.

Step 3 - calcolare context vector

Una volta determinati gli *attention weights*, viene calcolato il context vector (thought vector) che verrà usato per predire la parola successiva della sequenza.

$$contextvector = e_1 * h_1 + e_2 * h_2 + e_3 * h_3 + e_4 * h_4 + e_5 * h_5 + e_6 * h_6$$

In questo modo se i valori e_1, e_2 sono molto più alti degli altri, per la predizione verrà tenuto conto sopratutto dell'informazione veicolata degli stati h_1, h_2.

Step 4 - concatenare context vector e l'output dell'istante precedente

A questo punto il decoder utilizza context vector e la parola in output generata all'istante precedente per prevedere la parola successiva nella sequenza. Questi due vettori vengono semplicemente concatenati e dati in pasto al decoder. Nel caso specifico mostrato in figura, poichè viene riportata la prima iterazione del ciclo di decoding non abbiamo una parola predetta all'istante precedente, per questo viene usato il token speciale <START>.

Step 5 - output del decoder

Il decoder poi genera la parola successiva nella sequenza in output, nonché lo stato interno, che viene poi passato come input all'iterazione successiva. Quando il decoder genera il token <END> il processo di generazione di nuove parole per la sequenza in output viene interrotto.

Vantaggi e svantaggi di Attention

Bisogna notare che, a differenza del context vector utilizzato per ogni time step del decoder nel caso di modelli Seq2Seq tradizionali, nel caso del modello appena descritto si calcola un context vector separato per ogni time step dal momento che ad ogni iterazione vengono ricalcolati gli *attention weights*.
Utilizzando questo meccanismo il modello Seq2Seq è in grado di trovare correlazioni interessanti tra diverse parti della sequenza in input e rifletterle nella sequenza di output.
Anche in questo caso in fase di training viene solitamente usato il Teacher Forcing, come spiegato nella sezione precedente.
É comunque necessario notare che il meccanismo di Attention non comporta solo vantaggi: il maggior svantaggio è il fatto che sia fortemente costoso in termini di tempo e difficilmente parallelizzabile. Per risolvere questo problema Vaswani et. al. hanno proposto un nuovo modello chiamato "Transformer" che utilizza la sola Attention [4] e si sbarazza di ricorsione e layer convoluzionale, realizzando così un modello altamente parallelizzabile e computazionalmente efficiente.

1.2 Transformers

I transformer sono un modello basato sul meccanismo di attention precedentemente descritto e proposto da Google attraverso il paper "Attention is all you need" [4] nell'anno 2017. Come riportato nell'abstract del paper appena citato, i transformers fanno maggiore affidamento, rispetto ai modelli preesistenti, sul meccanismo di attention. L'attention diventa l'elemento fondamentale dell'intero modello consentendo di fare a meno di ricorrenza e convoluzione e permettendo di conseguenza un utilizzo più consistete della parallelizzazione ed un abbattimento significativo dei tempi di training. Per di più, le implementazioni di questo modello hanno consentito di superare i risultati ottenuti dai modelli preesistenti in molti task di natural language processing e sono diventati la base di numerosi modelli che sono ora lo stato dell'arte in questo campo. In questa sezione illustrerò l'architettura interna ai trasformers basandomi prevalentemente sulle informazioni ricavate dal paper "Attention is all you need" e da articoli ed approfondimenti scritti a riguardo su siti che si occupano di Deep Learning come "Toward Data Sciente". Le immagini che utilizzerò per descrivere il modello e il percorso logico che seguirò verranno invece ripresi dall'articolo "The Illustrated Transformer" [5] postato da Jay Alammar sul suo blog personale.

1.2.1 Architettura transformer ad alto livello

Di seguito viene descritto il modello dei transformer utilizzando come esempio un task di neural machine translation, in particolare vedremo la traduzione della frase "Je suis étudiant" nella corrispondente versione inglese "I am a student". Si seguirà un approccio dal generale al particolare: ad alto livello il transformer può essere descritto come una "macchina" che trasforma una frase in un altra, nell'esempio preso in considerazione questa trasformazione è una traduzione. Il transformer al suo interno è composto da 2 componenti principali: un blocco di encoding, che elabora la frase in ingresso e restituisce un risultato successivamente dato in pasto al blocco di decoding, il quale restituisce la frase in output. All'interno di questi due blocchi gli encoders e i decoders sono disposti in due stack; viene utilizzato lo stesso numero di encoders e decoders. L'immagine riportata in seguito mostra attraverso uno schema quanto è stato descritto.

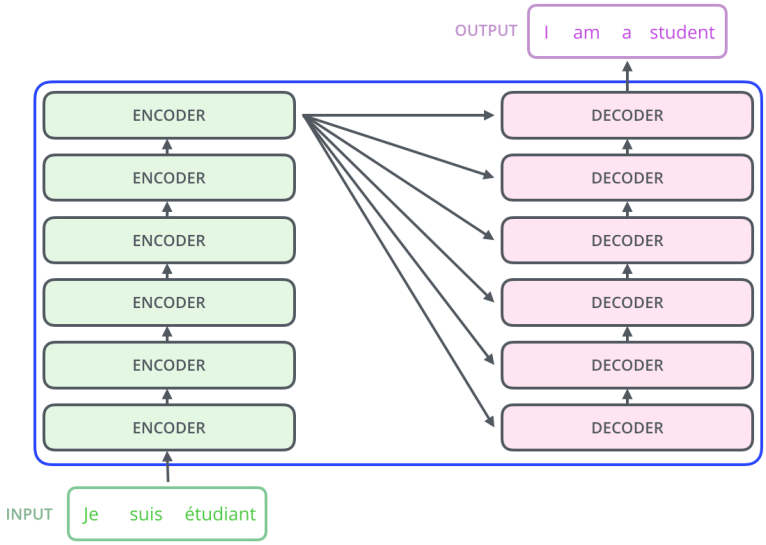


Figura 1.8: Transformers costituito da 2 stack: il 1° di encoders, il 2° decoders

Gli encoders hanno tutti la stessa struttura, nonostante non condividano i pesi. Ogni encoder è formato da 2 layer principali:

- **Self-Attention layer:** è un layer che consente all'encoder di prestare attenzione alle altre parole presenti nella frase in input durante l'*encoding* di un termine

- **Feed Forward Neural Network:** applica una trasformazione al risultato dell'encoding. Si può applicare la stessa rete neurale parallelamente a tutte le posizioni

L'input di un encoder passa prima attraverso il Self-Attention layer e solo successivamente viene elaborato da Feed Forward Neural Network.

Ogni decoder a sua volta ha la stessa struttura, che contiene entrambi questi layer separati però da un terzo later: l'**Encoder-Decoder Attention layer**. Come dice il nome, si tratta di un ulteriore layer di attention che aiuta il decoder a concentrarsi sulla parte rilevante della frase in input in modo analogo a quello che abbiamo visto essere il ruolo del meccanismo di attention nei modelli Seq2Seq.

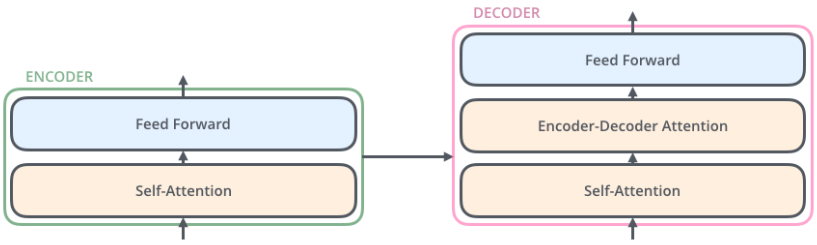


Figura 1.9: struttura dei layer interni a encoder e decoder

1.2.2 Encoder

Ora che è stata delineata la struttura di encoder e decoder, può essere utile prendere in considerazione come l'input venga memorizzato in vettori e tensori e come venga trasformato nella transazione attraverso i componenti del transformer.

Chiamiamo "frase" l'input testuale dato in pasto al transformer. La frase deve essere passata al transformer nella forma di una lista di word embedding. Si può immaginare (anche se è solo una semplificazione) che ad ogni parola di una frase corrisponda un word embedding. Quindi in una fase detta di "**Embedding**", che precede il passaggio della frase al transformer, vengono determinati gli embedding per tutte le parole che compongono la frase, dopo di che questi embedding vengono memorizzati in un vettore che può essere passato in input al transformer.
Ogni embedding è un vettore numerico di lunghezza prestabilita (uguale per tutti gli embedding es.: 512) che indica la posizione di un termine in uno spazio multidimensionale. Questa rappresentazione delle parole è utile in quanto veicola alcune informazioni significative relative ai termini: ad esempio, in questo spazio multidimensionale più i termini sono vicini tra loro più sono semanticamente simili.
Oltre alla dimensione dei vettori word embedding, anche la dimensione dei vettori frase è fissata a priori. Frasi più lunghe della dimensione prestabilita vengono troncate, mentre a frasi più corte della dimensione prestabilita vengono aggiunti degli elementi "segnaposto" che vengono riconosciuti e non presi in considerazione dal transformer nell'esecuzione del task nlp; tale processo viene definito *padding*.
La fase di embedding viene eseguita solo prima che la frase venga passata al primo encoder, dopo di che il risultato del primo encoder viene dato direttamente in pasto al secondo encoder e così via fino all'ultimo encoder.

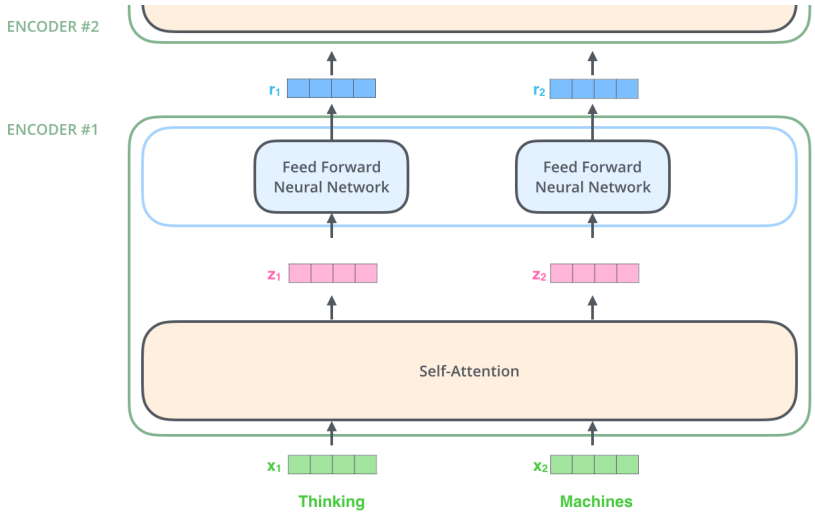


Figura 1.10: Flusso informazioni all'interno di un encoder

In figura 1.10 viene mostrata una delle proprietà chiave del transformer: la *parola* (da quì in poi intesa come *word embedding*) in ogni posizione segue un proprio percorso all'interno dell'encoder. All'interno del self-attention layer ci sono dipendenze tra questi percorsi, mentre nel feed-forward layer le dipendenze sono assenti per cui in questa seconda fase i calcoli da effettuare sulle varie parole possono essere eseguiti in parallelo.
Quindi complessivamente quello che accade ogni volta che una nuova frase deve essere elaborata è che, una volta che la frase è passata attraverso il processo di embedding, questa viene data in pasto all'encoder come lista di word embeddings. L'encoder elabora la lista facendo passare i word embedding attraverso un Self-Attention layer e poi attraverso una FNN, dopo di che il risultato viene inviato come input all'encoder successivo.

Self-Attention

Si è parlato di Self-Attention layer, ma fin quì non è mai stato detto esplicitamente cosa sia la **self-attention**. Il concetto di self-attention può essere spiegato adeguatamente immaginando di dover tradurre una frase dall'inglese: prendiamo come esempio la frase "The animal didn't cross the street because it was too tired". Per effettuare correttamente la traduzione verso una qualsiasi lingua è necessario avere ben chiaro, tra le altre cose, a che cosa si riferisca il pronome "it". Per un uomo è facile comprendere che fa riferimento al termine "animal", ma per una macchina, che si basa semplicemente sulle regole grammaticali, potrebbe essere anche riferito a "street" o all'intera frase. La self-attention è quel meccanismo che mette in relazione un termine con tutti

gli altri termini contenuti della frase in cui compare, per comprendere quali siano le parole ad esso maggiormente correlate, ovvero quelle a cui è necessario prestare attenzione nella fase di traduzione.

Self attention con vettori come input

Di seguito riporto più nel dettaglio come avviene il calcolo della self attention utilizzando un esempio grafico che mostra le diverse parole della frase come vettori.

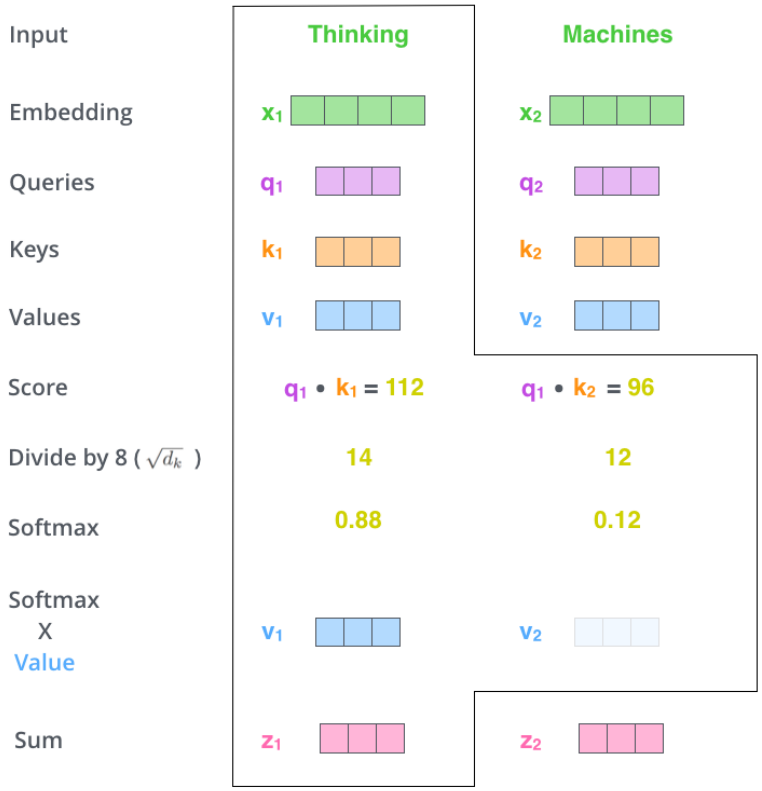


Figura 1.11: Calcolo della Self-Attention

É quindi possibile decomporre il calcolo della self attention in sei passi:

- **1° passo**: per prima cosa, da ogni word embedding contenuto nel vettore in input si ricavano tre vettori: **Query** vector, **Key** vector, **Value** vector. Questi vettori sono il risultato del prodotto tra un vettore word embedding e 3 matrici i cui valori sono parametri appresi durante il processo di training. Nella figura esemplificativa 1.12, i vettori risultato q, k, v hanno una lunghezza inferiore a quella del word embedding x da

cui vengono generati. Questa è una scelta architetturale compiuta per l'esempio mostrato, ma in un contesto reale non è necessariamente valida. Nell'esempio possiamo immaginare che gli embedding siano lunghi 512, i vettori q, k, v siano lunghi 64 e ogni matrice W abbia una forma 512x64.

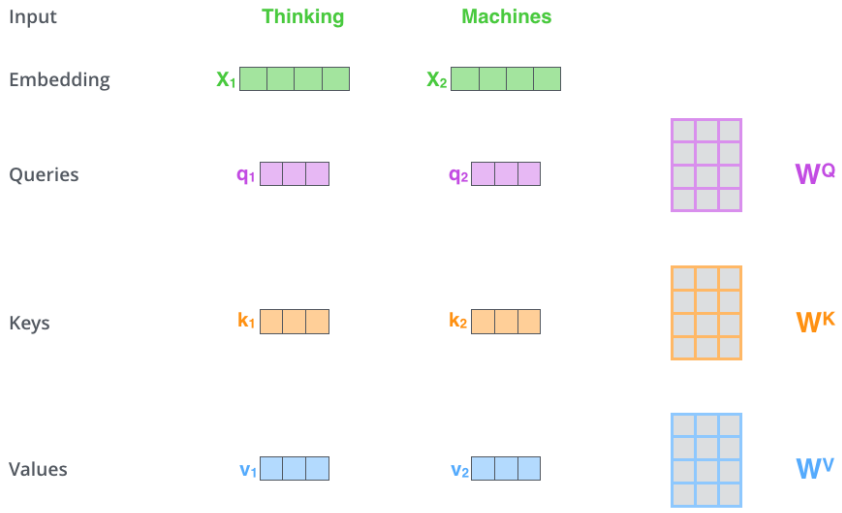


Figura 1.12: Calcolo dei vettori q, k, v

I tre vettori q, k, v sono astrazioni utili per ragionare sul meccanismo di attention e quindi effettuare i calcoli

- **2° passo**: è quindi necessario calcolare lo **score**. Se immaginiamo di calcolare la self-attention per la prima parola dell'esempio: "Thinking". Dobbiamo assegnare un punteggio ad ogni parola contenuta nella frase in input, calcolato rispetto al termine "Thinking". Tale punteggio determina quanta attenzione porre sulle varie parti della frase in input mentre si esegue l'encoding della parola presa in considerazione. Lo score è il risultato del prodotto scalare tra il vettore q relativo alla parola su cui si sta eseguendo l'embedding e il vettore k relativo alla parola di cui si sta calcolando lo score. Quindi durante l'encoding di x_1 dovrò calcolarmi:
  - $score_1 = q_1 \cdot k_1$, score di x_1 calcolato rispetto a x_1
  - $score_2 = q_1 \cdot k_2$, score di x_2 calcolato rispetto a x_1
  - $score_3 = q_1 \cdot k_3$, score di x_3 calcolato rispetto a x_1
  - ...
- **3° passo**: Per stabilizzare il gradiente, si dividono gli score ottenuti per la radice quadrata di d_k; dove d_k è dimensione (lunghezza) dei vettori key

usati. Nel nostro esempio tale lunghezza era 64 quindi ogni score verrà diviso per 8

- **4° passo**: Si procede quindi alla normalizzazione degli score applicando una operazione softmax, al fine di ottenere tutti score positivi, la cui somma è 1. Nell'esempio abbiamo preso in considerazione solo 2 score ed otteniamo come risultati: 0.88 e 0.12, che rispettano queste due condizioni. Si ottengono in questo modo dei pesi (0.88 e 0.12 nell'esempio) che determinano quanto ogni termine è influente nel calcolo dell'embedding. Naturalmente, dal momento che stiamo prendendo in considerazione il termine "Thinking" otterremo lo score massimo dal confronto con se stesso, tuttavia su frasi lunghe anche altre parole spesso vengono tenute in grande considerazione, ed è proprio questo l'aspetto importante

- **5° passo**: si procede quindi moltiplicando gli score ottenuti con i vettori v: in questo modo i termini a cui sono associati score più elevati saranno quelli più influenti, mentre termini a cui è assegnato un valore vicino a 0 avranno una influenza irrilevante nell'encoding del termine preso in considerazione

- **6° passo**: viene quindi effettuata la somma dei vettori ottenuti al passo precedente; in questo modo ogni termine porta un suo contributo più o meno grande nel calcolo di embedding. Il vettore risultante può quindi essere inviato alla rete neurale feedforward

Self attention con matrici come input

Nelle implementazioni reali, questi calcoli vengono svolti in forma matriciale per velocizzare il tempo di calcolo. Di seguito viene riportata brevemente una rappresentazione grafica che mostra come viene trattato l'input in forma matriciale.

Il primo passo prevede il calcolo delle matrici *Query*, *Key* e *Value*. Tali matrici vengono calcolate raggruppando gli embedding in una matrice X, che viene poi moltiplicata per le matrici di pesi W^Q, W^K, W^V che sono state addestrate in fase di training.

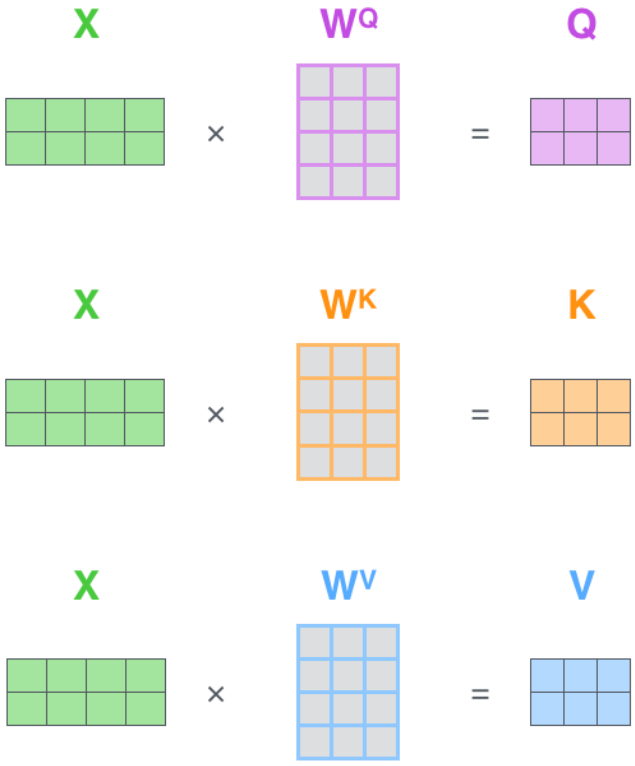


Figura 1.13: Calcolo delle matrici Q, K, V Ogni riga della matrice X corrisponde ad una frase in input. La figura mette in risalto la differenza tra la lunghezza degli embedding(4 quadrati) e quella dei vettori (3 quadrati)

La prossima figura condensa invece quanto avviene nei passi 2-6 illustrati in precedenza: Z è la matrice risultato ottenuta dall'applicazione dell'algoritmo di self-attention.

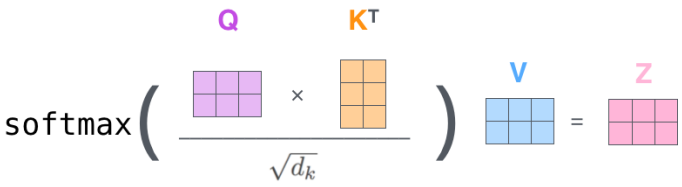


Figura 1.14: Calcolo self-attention con matrici

Multi-Head Attention

Nel paper "Attention is all you need" [4], gli autori presentano un meccanismo che va oltre quello della self-attention e viene chiamato "multi-head-attention". Questo meccanismo migliora le prestazioni del layer di attention in due modi:

- espande la capacità del modello di concentrarsi su diverse posizioni. Nell'esempio illustrato nelle figure precedenti, z_1 deriva dalla combinazione degli encoding delle parole in ogni posizione, tuttavia il contributo maggiore viene apportato dalla parola stessa. Ma se prendessimo il caso della traduzione di una frase, ad esempio "The animal didn't cross the street beacuse it was too tired", sarebbe molto utile capire a chi si riferisca il termine "it", ovvero vorremmo concentrarci maggiormente sui contributi delle parole diverse dal termine preso in considerazione

- dà al layer di attention un sub-space di rappresentazione maggiore. Con la multi-head attention abbiamo molti insiemi di matrici Query/Key/Value contenenti pesi: Transformer usa 8 attention heads ("teste"), per cui finiamo ad avere 8 insiemi per ogni encoder/decoder. I pesi di ognuno di questi insiemi viene inizializzato con un valore randomico, che viene poi corretto in fase di training. Dopo l'addestramento, ogni set viene usato per proiettare l'input in un diverso sub-space

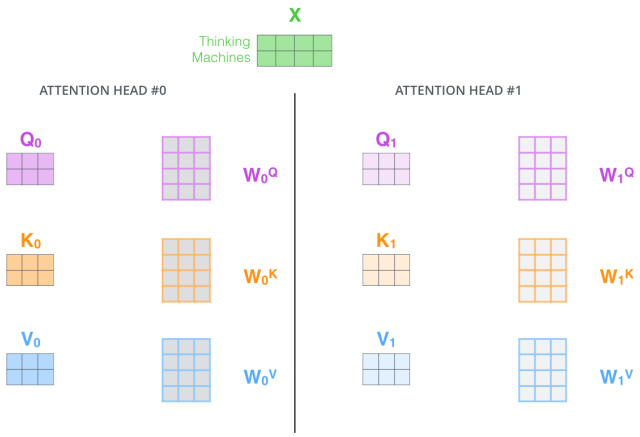


Figura 1.15: Con la multi-head attention si hanno matrici separate di pesi Q/K/V per ogni head

Se effettuiamo 8 volte, con 8 set di matrici diverse, il calcolo per la self-attention, come mostrato precedentemente, otterremo 8 matrici Z diverse:

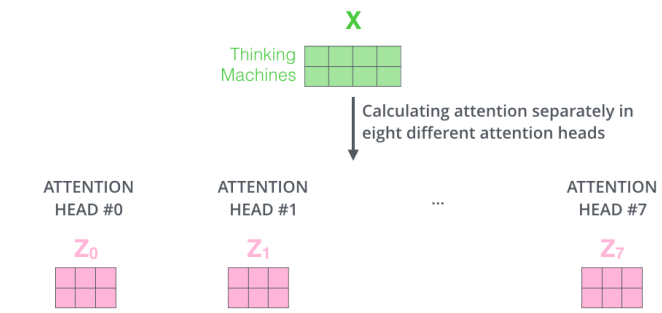


Figura 1.16: risultato applicazione multi-head self attention

Il layer feedforward tuttavia non si aspetta di ricevere 8 matrici diverse: attende una singola matrice: ovvero un vettore per ogni termine. É quindi necessario individuare un modo per condensare in un unica matrice queste 8 matrici risultato. Il modo scelto per effettuare questa operazione si basa sul prodotto matriciale per una matrice di pesi W^O. Di seguito una figura che mostra graficamente quanto avviene:

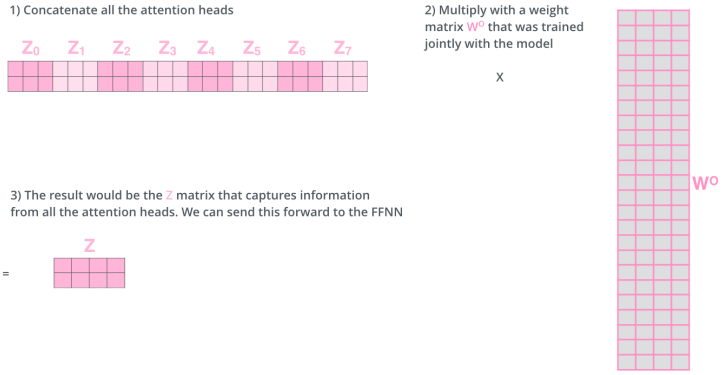


Figura 1.17: multi-head self attention, il risultato delle teste viene condensato in un unica matrice attraverso un prodotto matriciale per W^O

Di seguito riporto una figura che mostra in un unica immagine tutti i passaggi illustrati fino a questo momento.

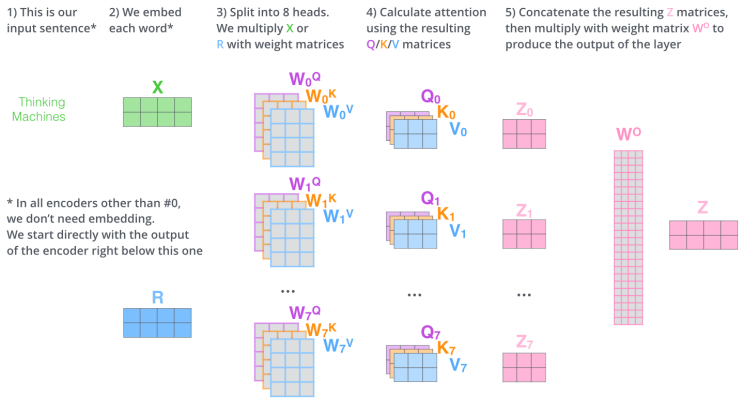


Figura 1.18: multi-head self attention

Ora che è stato descritto come il risultato delle diverse "teste" viene combinato, è possibile capire meglio l'esempio preso in considerazione all'inizio di questa sezione. Se pensiamo alla frase "The animal didn't cross the street because it was too tired" e ci concentriamo sul termine "it" teste diverse possono porre un peso maggiore a diverse parti della frase. Nella prossima figura viene mostrato un esempio:

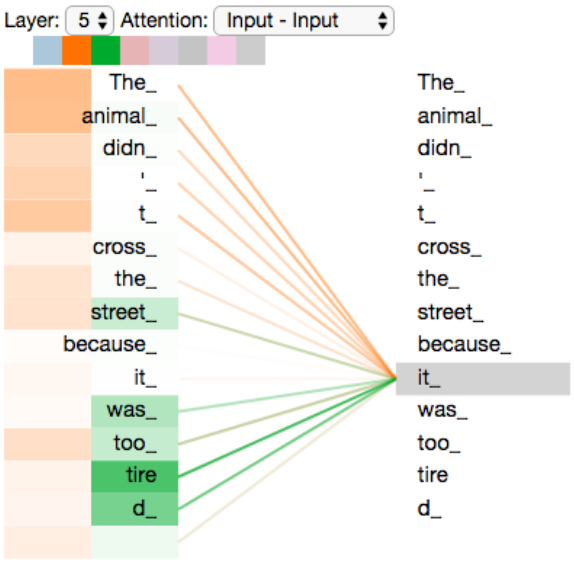


Figura 1.19: focus di 2 teste in multi-head self attention

Nella figura 1.19 vengono messe in evidenza le parti di frase su cui si concentrano 2 teste durante il calcolo della multi-head self attention. La prima testa, quella in arancione, si concentra sulla connessione tra "it" e i termini

"the" e "animal"; la seconda testa, in verde, si concentra sul legame tra "it" e "tired". Si noti che entrambi gli aspetti sono in effetti collegati a "it" e l'importanza dell'utilizzo di più teste sta proprio nel fornire al meccanismo di attention la capacità di rappresentare un termine con legami verso più parti della stessa frase, questo consente di catturare una maggiore quantità di informazione e più sfumature di significato.

1.2.3 Embedding degli input

Un aspetto ancora non citato del modello descritto riguarda la posizione delle parole nella frase data in pasto al modello. I Transformers gestiscono questa informazione aggiungendo dei vettori (in figura chiamati "positional encoding") agli input embedding. Questi vettori seguono un pattern specifico, che viene appreso dal modello e che consente di determinare la posizione di ogni parola o la distanza tra diverse parole in una sequenza. L'idea alla base di questa tecnica è che, aggiungendo questi valori ai vettori di embedding, il modello possa raggiungere una buona comprensione delle distanze tra i termini che compongono frase in input durante il calcolo di attention. Si riporta a seguito una figura a scopo esemplificativo:

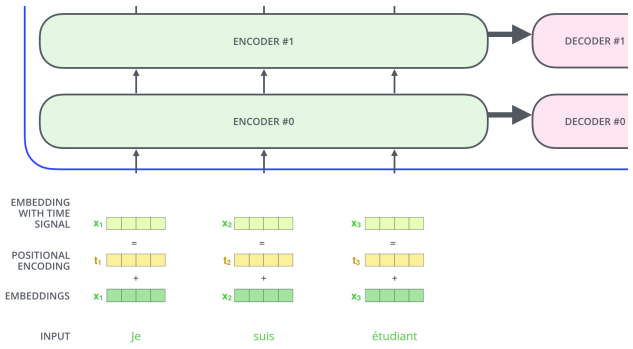


Figura 1.20: Per fornire al modello una comprensione dell'ordine delle parole, vengono aggiunti agli embedding dei vettori detti *POSITIONAL ENCODING*

1.2.4 Residuals

Dal punto di vista implementativo, è possibile terminare la descrizione dell'encoder aggiungendo un dettaglio a livello architetturale. Ogni sub-layer di un encoder (come spiegato in precedenza ne esistono due: self-attention-layer e FFNN) è "circondato" da una *residual connection* e seguito da un layer di normalizzazione. Questa accortezza mira sempre ad evitare i problemi di vanishing/exploding gradient.

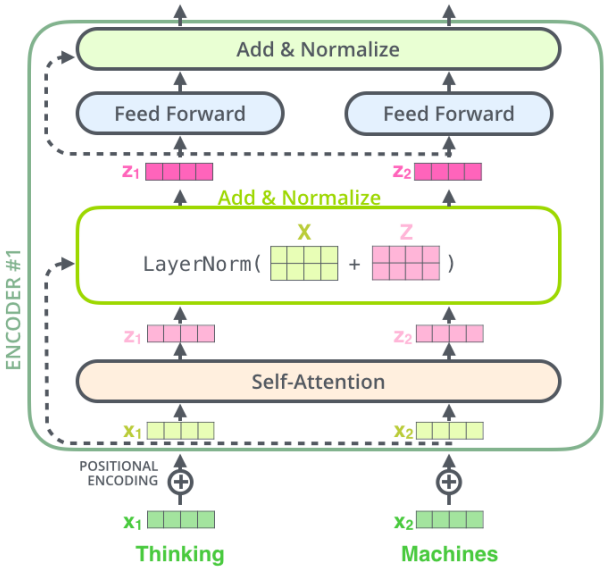


Figura 1.21: Residual connections e normalization-layer nell'architettura di un encoder

Le residual connection e i layer di normalizzazione circondano anche ogni sub-layer di ogni decoder. Quindi se volessimo rappresentare l'architettura globale di un Transformer costituito da 2 encoder e decoder impilati, otterremmo l'architettura mostrata nella seguente figura:

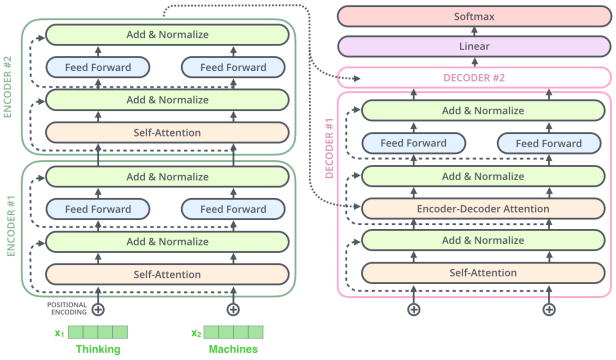


Figura 1.22: Residual connections e normalization-layer nell'architettura di un Transformer

1.2.5 Decoder

Una volta compreso il funzionamento dell'encoder, è facile comprendere quello del decoder, le cui componenti principali sono simili a quelle dell'encoder.

Per comprendere il funzionamento del decoder è utile sapere come encoder-decoder operano a livello di sistema. La figura 1.23 mostra lo stadio finale di un task di traduzione portato a termine da un transformer.

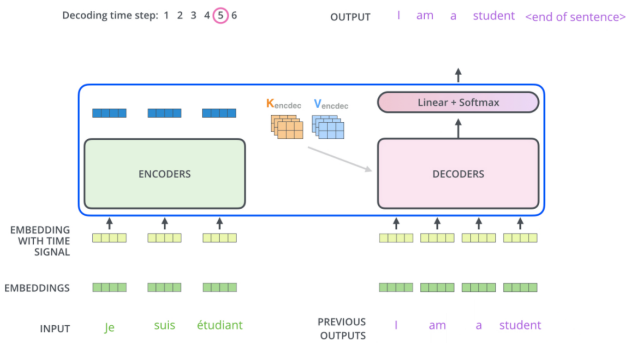


Figura 1.23: Traduzione di una frase eseguita da un transformer

L'encoder inizia elaborando la sequenza in input. L'output dell'ultimo encoder (quello in cima allo stack) viene trasformato in un set di attention vector K e V. Questi devono essere usati da ogni decoder all'interno del layer "encoder-decoder attention" per comprendere su quali parti della sequenza in input ci si deve concentrare nell'esecuzione del task. Ad ogni passo della fase di decoding, che inizia una volta terminata quella di encoding, il transformer restituisce un elemento in output. I passi della fase di decoding vengono ripetuti finchè non viene generato un simbolo speciale che indica l'avvenuto completamento del processo di generazione del risultato. L'output di ogni passo di decoding viene dato in pasto al primo decoder (quello più in basso nello stack). I risultati intermedi attraversano lo stack di decoders esattamente nello stesso modo in cui avviene per lo stack di encoders. Inoltre anche agli embedding dei termini forniti in input ai decoder vengono aggiunti i vettori *positional encoding* per far si che i decoder conoscano le posizioni delle parole nella sequenza in output.

I layer di self-attention all'interno del decoder operano in modo leggermente diverso da quanto accade per l'encoder. Nel decoder, il layer di self attention può solo prestare attenzione alle parole che precedono quella da determinare nella sequeza finale. Questo meccanismo viene implementato mascherando le posizioni successive (ovvero settandole a *-inf*) prima del passo di soft-max durante il calcolo della self-attention.

Il layer di "Encoder-Decoder Attention" funziona in maniera analoga al layer self-attention, tranne che per un aspetto: la matrice Q (queries) viene creata dal layer sottostante e prende le matrici K (keys) e V (values) dall'output dello stack di encoder.

1.2.6 Layers finali: Linear e Softmax

Lo stack di decoder emette in output un vettore di floats, non un termine del dizionario. É necessario trasformare tale vettore in una parola e questo è il compito assolto da **Final-layer** e **Softmax-layer**. Il Final-layer è una semplice rete neurale totalmente connessa che proietta il vettore risultato prodotto dallo stack di decoder in un vettore molto più grande detto **logits vector**. Assumiamo che il modello conosca 10000 termini unici che costituiscono il vocabolario di output del nostro modello; tale vocabolario viene appreso a partire dal training dataset. Il final-layer dovrà quindi produrre un logits vector lungo 10000 celle, dove ogni cella corrisponde ad un punteggio assegnato ad uno dei termini del dizionario. Il softmax layer traduce quindi tali punteggi ("scores") in probabilità: avremo 10000 valori positivi che sommano a 1.00. Il termine che corrisponde alla cella con la più alta probabilità viene scelto come output del time step. Questo è il modo in cui viene interpretato l'output prodotto dai decoder. Di seguito una rappresentazione grafica:

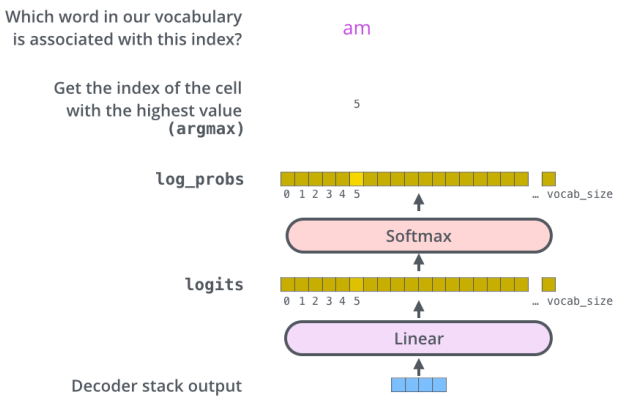


Figura 1.24: Esempio di funzionamento dei layer finali di un Transformer

1.2.7 Training

In questo paragrafo viene riportata una suggestione di come viene affrontato il training di un modello basato su transformer. Questa informazione è rilevante ai fini del progetto perchè i modelli utilizzati nella fase di sviluppo si basano su transformer, di conseguenza anche il training di tali modelli riprende le tecniche di addestramento utilizzate per tali modelli.

In fase di training, un modello non addestrato esegue il forward-pass come indicato nelle sezioni precedenti. Tuttavia, dal momento che il modello viene addestrato su un dataset etichettato, è possibile confrontare gli output ottenuti con quelli attesi. Per visualizzare meglio la situazione si può pensare ad un

esempio pratico: supponiamo che il vocabolario del modello contanga solo 6 termini: "a", "am", "i", "thanks", "student", "<eos>"; dove "<eos>" sta per "end of sentence" ed è il termine speciale che indica il termine della frase.

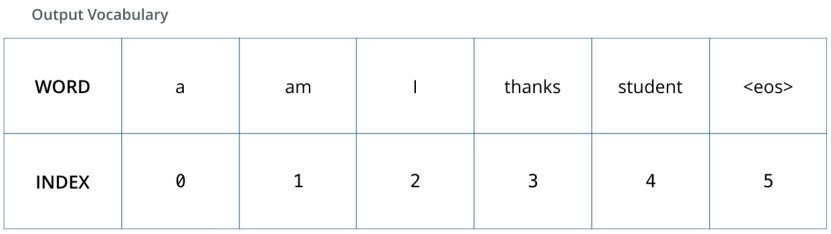


Figura 1.25: Prima dell'inizio dell'addestramento viene creato un vocabolario con i termini che possono comporre la sequenza di output del modello

Una volta costruito il vocabolario di output, è possibile usare un vettore della stessa lunghezza del dizionario per indicare una parola appartenente al vocabolario secondo la codifica "one-hot-encoding".

Supponiamo quindi di voler tradurre dall'italiano all'inglese ed essere nella fase di training. Come primo passo in fase di training potremmo voler tradurre "grazie" in "thanks". Questo vuol dire che voglio ottenere come output una distribuzione di probabilità che porta il modello a restituire la parola "thanks" in quanto individuata come la più probabile traduzione corretta di "grazie". Ma dal momento che il modello non è ancora stato addestrato è improbabile ottenere tale risultato.

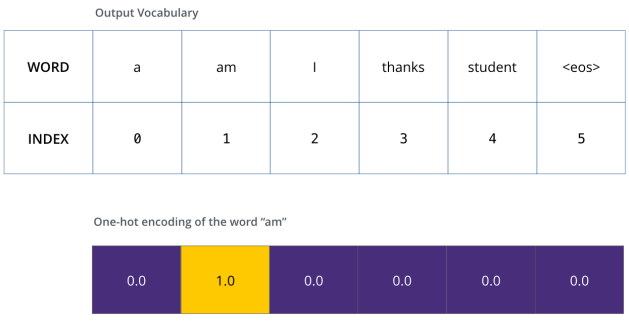


Figura 1.26: Dal momento che i pesi del modello sono tutti inizializzati randomicamente, il modello produce una distribuzione di probabilità random durante i primi cicli di addestramento. É tuttavia possibile confrontare tali risultati ottenuti con l'output atteso e quindi aggiustare tutti i pesi usando backpropagation per portare il modello a produrre un risultato più simile all'output desiderato

L'esempio portato è estremamente semplificato. Per l'addestramento vengono usate intere frasi. Ad esempio, supponendo di avere come input la frase "Io sono uno studente" si vuole ottenere la frase "I am a student". Questo vuol dire volere un modello che restituisca in output una sequenza di distribuzioni di probabilità in cui:

- ogni distribuzione di probabilità è rappresentata da un vettore che ha lunghezza pari alla grandezza del vocabolario (6 nell'esempio)
- la prima distribuzione di probabilità deve avere la più alta probabilità in corrispondenza della cella associata al termine "i"
- la seconda distribuzione di probabilità deve avere la più alta probabilità in corrispondenza della cella associata al termine "am"
- e così via fino alla quinta distribuzione di probabilità che deve indicare "<eos>" come simbolo

Dopo aver addestrato il modello per una quantità di tempo sufficiente su un grande dataset, si spera di ottenere una distribuzione di probabilità di questo tipo:

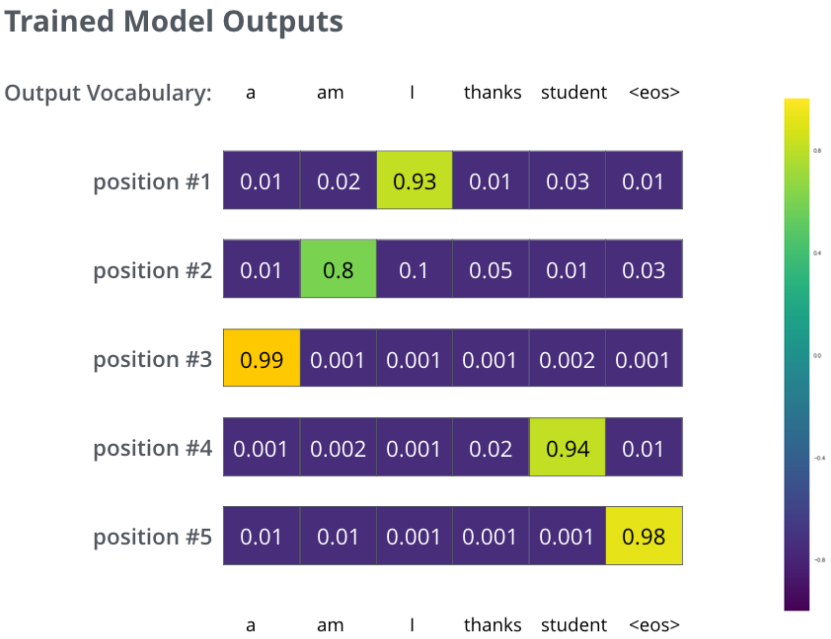


Figura 1.27: Esempio sequenza distribuzione di probabilità desiderata dopo il training

Dal momento che il modello produce un output alla volta, si potrebbe dar per scontato che il modello selezioni il termine a cui è associata la più alta probabilità nella distribuzione e rigetti il resto. Questo è uno dei modi possibili per fare training. Un altro modo sarebbe quello di tenersi da parte le parole a cui sono associati i valori di probabilità più elevati, ad esempio le prime due parole; dopo di che, allo step di training successivo, eseguire il modello 2 volte: una volta supponendo di aver selezionato il primo termine e una seconda volta supponendo di aver selezionato il secondo termine. Questo metodo viene chiamato "beam search".

1.3 Bert

Con il paper "BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding" [6] è stato introdotto un nuovo language representation model chiamato "BERT", nome che sta per "**B**idirectional **E**ncoder **R**epresentations from **T**ransformers".

BERT è nato dall'idea di sviluppare un modello flessibile, che fosse in grado di adattarsi facilmente a diversi task di text mining. Il modello è composto da più layer, ognuno dei quali sfrutta rappresentazioni bidirezionali del linguaggio pre-addestrate su testi non etichettati. Le rappresentazioni dell'input testuale sono dette bidirezionali in quanto vengono inferite tenendo conto sia della parte di contesto che precede che di quella che segue il testo preso in considerazione.

A partire dal modello pre-addestrato è facile ottenere modelli con performance vicine allo stato dell'arte per task come *question answering* e *language inference*, senza dover apportare modifiche sostanziali all'architettura del modello. Per ottenere i nuovi modelli è infatti sufficiente aggiungere un output layer specifico per l'attività da svolgere ed effettuare pochi cicli di addestramento (che in questo caso viene detto **fine-tuning**) su una quantità di dati notevolmente inferiore rispetto a quella necessaria per addestrare (**pre-training**) l'interno modello ex novo. L'obiettivo dei fine-tuning è infatti quello di addestrare solo i layer aggiunti al modello preaddestrato, in quanto il resto del modello è pronto per essere utilizzato. Nei paragrafi seguenti illustrerò più in dettaglio BERT, basandomi sulle informazioni contenute nel paper in cui è stato presentato.

1.3.1 Fine-Tuning: Language Model bidirezionali, MLM, NSP

Il pre-addestramento di language model è stato dimostrato essere una tecnica molto efficace per quanto riguarda la risoluzione di task nel campo del natural language processing.

Esistono due strategie per applicare delle rappresentazioni del linguaggio pre-addestrate ai task: **feature-based** e **fine-tuning**.

Gli approcci feature-based usano architetture diverse per ogni specifico task, tali architetture includono le rappresentazioni pre-addestrate come feature aggiuntive.

L'approccio fine-tuning, al contario, introduce una quantità minima di parametri specifici per ogni task; il modello apprende come risolvere lo specifico task a cui lo si vuole applicare attraversando una fase di fine-tuning, si parte però da pesi ottenuti in fase di pre-training.

Fino all'avvento di BERT entrambi gli approcci condividevano la stessa funzione obiettivo nella fase di pre-trainig, in cui venivano utilizzati language

model unidirezionali per apprendere una rappresentazione del linguaggio generale, ovvero adatta ad essere il punto di partenza per ogni task. Gli ideatori di BERT erano però convinti che le tecniche preesistenti limitassero il potenziale delle rappresentazioni pre-trained, sopratutto quando accompagnate da un approccio basato su fine-tuning. La più grande limitazione, a parer loro, è legata all'utilizzo di language model unidirezionali, che riduce il numero delle possibili architetture utilizzabili in fase di pre-training. Utilizzando language model unidirezionali, in ogni self-attention layer di un transformer ogni token può prestare attenzione ai token che lo precedono, ma questo è fortemenete limitante quando dobbiamo applicare il modello a task a livello di token come question answering, in cui è di fondamentale importanza incorporare nella risposta l'intero contesto. BERT riduce i limiti imposti dai language model unidirezionali utilizzando *Masked Language Model* (**MLM**) durante il pre-addestramento. Un task di MLM maschera alcuni token dell'input scelti casualmente e richiede al modello di predire il valore originale della parola mascherata; il modello è in grado di ricavare tale informazione prendendo in considerazione l'intero contesto del termine da individuare, ovvero l'insieme dei token che precedono e seguono la parola mascherata. In questo modo viene pre-addestrato un deep bidirectional Transformer. Durante il pre-training viene affiancato a MLM un task che effettua *Next Sentence Prediction* (**NLP**), questo consente al modello di apprendere relazioni tra le coppie di frasi.

1.3.2 Implementazione del modello

Di seguito illustrerò come è stato implementato il modello BERT. Come già accennato ci sono due aspetti fondamentali nel framework messo a disposizione dagli autori: *pre-training* e *fine-tuning*. Durante la fase di *pre-training*, il modello viene addestrato su più task utilizzando *dati non etichettati*. Durante il *fine-tuning*, il modello BERT viene prima inizializzato con i parametri pre-trained, dopo di che viene applicata una operazione di fine-tuning su tutti i parametri utilizzando *dati etichettati* legati allo specifico task, detto **downstream task**. Ogni downstream task ha dei modelli fine-tuned propri, sebbene tutti vengano inizializzati con gli stessi parametri pre-addestrati. Di seguito utilizzerò a scopo esemplificativo *question-answering* (QA) come downstream task. Un tratto distintivo di BERT è la sua architettura facilmente adattabile ai diversi downstream task: la differenza tra l'architettura utilizzata in fase di pre-addestramento e quella usata sui task downsteam è minima, dal momento che il corpo centrale del modello rimane lo stesso.

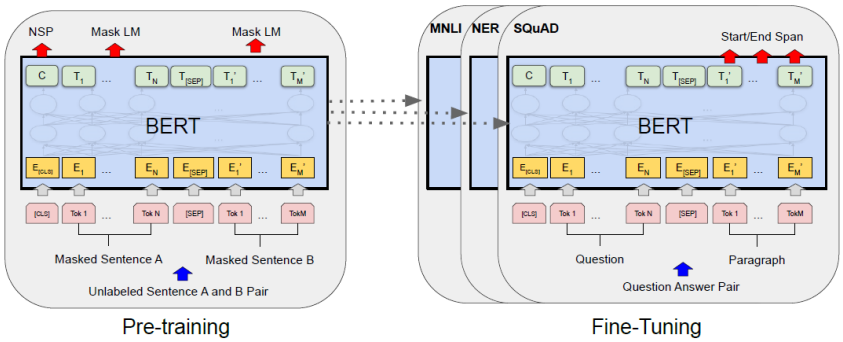


Figura 1.28: La figura mostra una visione di insieme delle procedure di pre-training e fine-tuning in BERT. Come si evince confrontando i due schemi, fatti salvi gli output layer, l'architettura usata per pre-training e fine-tuning è la stessa. Inoltre downstream task diversi tra loro utilizzano i pesi derivati dallo stesso modello pre-addestrato. Durante la fase di fine-tuning, tutti i parametri vengono addestrati nuovamente per ottimizzare i valori rispetto al task da eseguire. Tra i token risaltano: [CLS] è un simbolo speciale che viene anteposto ad ogni input; [SEP] è un token speciale che permette di separare token relativi a diverse parti dell'input: in QA viene usato per separare la parte relativa alla domanda e quella relativa alla risposta

Architettura del Modello

L'architettura del modello BERT è basata su quella dell'encoder del Transformer multi-layer e bidirezionale descritta nella sezione precedente. Di seguito adotterò la nomeclatura e le notazioni usate dagli autori del paper "BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding", che chiamano con L il numero di layer (i Transformer block); con H si fa riferimento alla hidden size; con A al numero di self-attention heads. All'interno del paper, gli autori riportano i risultati ottenuti con due modelli poi messi a disposizione:

- **BERT$_{\mathbf{BASE}}$**: L = 12, H = 768,A = 12, Parametri totali = 110M
- **BERT$_{\mathbf{LARGE}}$**: L = 24, H = 1024,A = 16, Parametri totali = 340M

ho riportato queste due configurazioni in quanto costituiscono spesso il punto di partenza per lo sviluppo di modelli successivi a BERT.

Rappresentazioni di input/output

Al fine di consentire a BERT di gestire molteplici task, gli sviluppatori hanno studiato per questo modello un meccanismo di rappresentazione dell'input che

permettesse di dare in pasto al modello in maniera non ambigua una o due frasi nella forma di un unica sequenza di token. Per esempio, per quanto riguarda Question Answering, il modello deve essere in grado di poter prendere in input due frasi distinte: la domanda e la risposta. Grazie a questo meccanismo, una "frase" può essere una stringa di testo di lunghezza arbitraria piuttosto che una vera e propria frase in senso stretto. Quindi quando parliamo di sequenza in input al modello, ci riferiamo alla lista di token data in input a BERT, che può essere costituita da una o due frasi concatenate assieme. Gli sviluppatori hanno utilizzato WordPiece embedding con un vocabolario di 30 mila token.

Come mostrato in figura 1.28, il primo token di ogni sequenza è sempre un simbolo speciale usato per la classificazione: [CLS]. La sequenza di token che costituisce l'hidden state finale correlato a questo token viene usata come una rappresentazione aggregata dell'input nei task di classificazione. Quando si hanno 2 frasi in input, queste vengono impacchettate in un unica sequenza al cui interno le frasi vengono distinte in 2 modi:

- si pone tra la prima e la seconda frase uno speciale token [SEP]
- ad ogni token viene aggiunto un embedding che consente di distinguere token appartenenti alla prima da quelli appartenenti alla seconda frase

Come mostrato in figura 1.28, gli autori del paper denotano:

- con E l'input embedding
- con $C \in R^H$ l'hidden vector finale relativo al token speciale [CLS]
- con $T_i \in R^H$ l'hidden vector finale corrispondente all'i-esimo token di input

Dato un token, la rappresentazione dell'input è costruita sommando i corrispondenti token embedding, segment embedding, position embedding, come mostrato nella seguente figura.

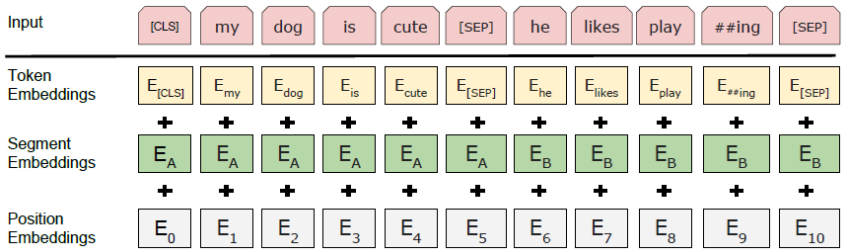


Figura 1.29: La figura riporta una rappresentazione di come viene costruita la rappresentazione dell'input data in pasto al modello BERT

1.3.3 Pre-training BERT

A differenza di altri modelli tradizionali, il pre-training di BERT non prevede l'utilizzo di language model left-to-right o right-to-left; vengono invece utilizzati due task non supervisionati descritti di seguito. Si può guardare alla parte sinistra di 1.28 per avere un riferimento grafico di quanto descritto in questa sezione.

MLM: Masked Language Modeling

Intuitivamente, è ragionevole pensare che un modello bidirezionale profondo (deep) sia nettamente più "potente" di un modello left-to-right o di una concatenazione superficiale di un modello right-to-left al modello left-to-right. Purtroppo però i conditional language model standard possono essere addestrati solo su modelli right-to-left o left-to-right, dal momento che il bidirectional condictioning consentirebbe ad ogni parola di vedere se stessa indirettamente, e il modello sarebbe in grado in maniera molto banale di predire la parola obiettivo in un contesto multi-layer. Per addestrare una rappresentazione bidirezionale profonda, viene mascherata una percentuale random di input token, che devono poi essere predetti. Tale procedimento prende il nome di "Masked Language Modeling". Nel caso riportato dal paper [6], gli hidden vector finali che corrispondono ai token mascherati vengono dati in pasto ad una softmax sull'intero vocabolario, come avviene in un language model standard. Nel paper viene riportato che vengono mascherati il 15% di tutti i WordPiece token scelti randomicamente in ogni sequenza. Il modello si occupa solo di predire i token mascherati piuttosto che ricostruire l'intera sequenza. Sebbene questo consenta di ottenere un modello bidirezionale pre-addestrato, ci sono anche dei lati negativi. Per esempio in questo modo si viene a creare una incompatibilità tra la procedura di pre-training e quella di fine-tuning dal momento che il token [MASK] non compare mai in fase di fine-tuning. Per mitigare questo problema, si può evitare di rimpiazzare tutte le volte la parola da mascherare con un token [MASK]. Immaginando che l'i-esimo token venga scelto per essere mascherato, questo viene:

- rimpiazzato con il token [MASK] l'80% delle volte
- rimpiazzato con un token scelto randomicamente il 10% delle volte
- lasciato invariato il 10% delle volte

dopo di che T_i verrà usato per predite il token originale con cross entropy loss.

NSP: Next Sentence Prediction

Molti downstream task come Question Answering (QA) e Natural Language Inference (NLI) sono basati sulla comprensione della relazione tra 2 frasi, che non viene catturata automaticamente durante il language modeling. Per addestrare un modello in grado di comprendere la relazione tra le frasi, si può effettuare un pre-addestramento sul task binario di *Next Sentence Prediction*, i cui input possono essere facilmente generati a partire da un qualsiasi corpus di documenti monolingue. In particolare, si va a costruire un insieme di esempi da sottoporre al modello in fase di pre-training tale per cui:

- ogni esempio è composto da una coppia di frasi: frase A e frase B
- il 50% delle volte si sceglie come frase B quella frase che nel corpus segue la frase A; a tali esempi viene assegnata l'etichetta *IsNext*
- l'altro 50% delle volte si sceglie come frase B una frase a caso del corpus; a tali esempi viene assegnata l'etichetta *NotNext*

1.3.4 Fine-Tuning BERT

L'operazione di fine-tuning risulta molto semplice grazie al meccanismo di self-attention derivato da Transformer: questo consente a BERT di modellare molti downstream task diversi tra loro.

In altri modelli, per le applicazioni che richiedono l'utilizzo di coppie di frasi in input, spesso si adotta come strategia quella di effettuare indipendentemente l'encoding alle coppie di testo prima di applicare cross-attention bidirezionale. BERT invece utilizza il meccanismo di self-attention per unire questi due passi, dal momento che effettuando l'encoding di una coppia di frasi concatenate con self attention permette di attuare in maniera efficace cross-attention bidirezionale tra le due frasi.

Quindi per adattare BERT ad un qualsiasi task è sufficiente aggiungere al modello pre-addestrato i layer di input ed output specifici per quel task ed effettuare il fine-tuning end-to-end di tutti i parametri del modello.

A confronto della fase di pre-training, quella di fine-tuning è relativamente poco costosa.

1.4 Vision Transformer

In questa sezione riporto le informazioni relative al modello chiamato "Vision Trasformer" o informalmente "ViT". ViT è stato presentato attraverso il paper "An image is worth 16x16 Words: Transformers for image recognition at scale" [7] (Dosovitskiy et al.,2021) ed è un modello che mira ad adottare i Transformers per risolvere task che prendono in input immagini. ViT si differenzia dai modelli precedentemente pubblicati per non limitarsi ad usare il meccanismo di attention a supporto di reti CNN o al posto di alcune parti di esse, ma rimpiazza l'intera struttura CNN con un modello basato su transformer. Gli autori del paper hanno dimostrato che ViT consente di ottenere risultati eccellenti se comparati a quelli ottenuti da reti CNN che rappresentavano lo stato dell'arte, e raggiunge tali risultati con un costo computazionale inferiore per la fase di training. Come descriverò nei capitoli successivi tale modello, assieme a BERT, ha un ruolo di primo piano all'interno del progetto da me sviluppato. Per questo motivo, nelle prossime sezioni, descriverò in dettaglio il modello ViT. La descrizione che farò sarà basata sulle informazioni contenute nel paper con cui è stato presentato [7].

1.4.1 Da input testuali a immagini

I modelli basati su Transformers e più in generale tutte le architetture basate sul meccanismo di self-attention costituiscono lo stato dell'arte per quanto riguarda task NLP. L'approccio adottato è quello di addestrare il modello scelto, ad esempio BERT, con una fase di pre-training condotta su un dataset di grandi dimensioni; dopo di che il modello può andare incontro ad una seconda fase di addestramento (fine-tuning) specifica per il task che il modello deve imparare ad eseguire e che viene condotta su un dataset di dimensioni inferiore. I modelli di questo tipo sono migliori di quelli tradizionali per quanto riguarda efficienza e rendono possibile addestrare modelli su molti più parametri e dataset più grandi, ottenendo migliori performance.

Le architetture basate su reti CNN (si vedano per esempio [8, 9, 10]) hanno continuato a dominare la scena nel campo della computer vision fino a tempi recenti. I successi riscossi dai modelli basati su self-attention e transformer sui task NLP hanno spinto i ricercatori a combinare la self-attention con le architetture basate su CNN: si vedano Wang et al.,2018 [11] e Carion et al., 2020 [12]. Altri studi hanno portato a rimpiazzare completamente la convoluzione: Ramachandran et al. [13] e Wang et al.,2020 [14]. Nonostante questi ultimi modelli risultino altamente efficienti, fanno uso di pattern di attention specifici che hanno inibito i tentativi di far scalare in modo efficiente i modelli sulle

GPU moderne. Per questo motivo le architetture ResNet (come quelle degli studi riportati nei paper [15, 16, 17]) risultano ancora lo stato dell'arte.

In questo contesto si inserisce il contributo di Dosovitskiy e degli altri autori del paper [7]. ViT è stato realizzato cercando di applicare il Transformer, nella sua forma standard, all'immagine applicando la minor quantità di trasformazioni possibile. Per far questo, l'immagine è stata suddivisa in frammenti (ognuno dei quali detto **patch**); per ogni patch è stato ricavato un embedding lineare poi dato in input al Transformer. Da questa breve descrizione del funzionamento di ViT è facile intuire il parallelismo con i task NLP: ogni immagine corrisponde ad un testo preso in input; ogni patch corrisponde ad una parola di cui si ricava l'embedding.

Gli autori di ViT hanno effettuato addestramenti supervisionati sul modello utilizzando un task di image classification. Sono stati effettuati addestramenti su dataset di diverse dimensioni e confronti dei risultati ottenuti con le performance di ResNets. I test su dataset di medie dimensioni hanno evidenziato per ViT performance poco al di sotto di quelle ottenute da ResNet. Con dataset di grandi dimansioni (14M-300M immagini) invece le performance di ViT superano quelle ottenute da modelli ResNet. Questo comportamento è da imputare al fatto che i Transformer mancano dei bias induttivi che sono intrinsechi nelle reti CNN come la *translation equivariance* e *locality*, per cui non riescono a generalizzare adeguatamente se non addestrati su un numero di dati sufficiente.

1.4.2 Preparazione dell'immagine

Come accennato, l'immagine deve essere preparata prima di essere data in pasto ai Transformer. Gli autori hanno elaborato il metodo delle patch dopo essere passati in rassegna alle tecniche utilizzate da altri modelli pre-esistenti.

Una applicazione naive del meccanismo di self-attention alle immagini implicherebbe far si che ogni pixel presti attenzione ad ogni altro pixel. Questo implica un costo quadratico rispetto al numero di pixel, il che impedisce l'applicazione a problemi reali: questo approccio non scala a sufficienza. Per questo motivo, nel corso del tempo, sono stati tentati diversi approcci con l'obiettivo di effettuare approssimazioni tali da rendere applicabili i modelli basati su transformer nel campo dell'elaborazione delle immagini. Di seguito riporto alcune delle tecniche adottate.

Come descritto in un paper [18] del 2018, Parmar et al. hanno applicato un meccanismo di self-attention locale in cui ogni pixel prestava attenzione ai soli pixel vicini. Tali blocchi di pixel a cui si applica self-attention locale multi-head possono rimpiazzare completamente la convoluzione: si vedano i lavori condotti da Hu et al. [19], Ramachandran et al. [13] e Zhao et al. [20]. Una strategia diversa è stata adottata con gli Sparse Transformer [21]

che sfruttano approssimazioni scalabili per rendere la tecnica di self-attention globale applicabile alle immagini. Un modo alternativo per far scalare l'attention è quella di applicarla a blocchi di dimensioni variabili, come illustrato nel paper [22] di Weissenborn et al.; un caso estremo di tale strategia comporta l'applicazione dell'attention solo lungo alcuni assi [23, 14]. Molte di queste architetture di attention hanno prodotto risultati incoraggianti su task di computer vision, tuttavia richiedono un complesso lavoro di ingegnerizzazione per poter essere adottati efficientemente sull'hardware.

Tra i modelli preesistenti, quello più simile a ViT è stato presentato da Cordonnier et al. [24]. Il modello estrae patches di dimensione 2x2 dall'immagine in input e applica full self-attention sulla sequenza di patch. Il modello è molto simile, ma i risultati ottenuti sono meno convincenti di quelli riportati da ViT. Uno dei problemi di tale modello sta nel fatto che la dimensione ridotta delle patch renda difficile la gestione di immagini di grandezza medio-grande.

Come precedentemente accennato, molti progetti hanno tentato di combinare reti convoluzionali con forme di self-attention: ad esempio aumentando le feature maps per il task di *image classification* [25] o usando il meccanismo di self-attention per elaborare il risultato di una rete CNN. Quest'ultima tecnica è stata applicata a diversi downstream task tra cui *object detection* [26], *video processing* [11, 27], *image classification* [28], *unsupervised object discovery* [29], ecc.

Tra i modelli sviluppati più di recente c'è poi iGPT [30] (image GPT), che applica i Transformers ai pixel delle immagini dopo aver ridotto la risoluzione dell'immagine e lo spazio dei colori. Il modello è addestrato in maniera non supervisionata, come modello generativo. La rappresentazione che ne risulta può andare incontro ad un processo di fine-tuning ad esempio per un task di classificazione.

Il lavoro condotto da Dosovitskiy et al. mira ad esplorare il campo dell'image recognition a un livello di scala maggiore di quello dei dataset solitamente usati con ImageNet. L'utilizzo di ulteriori sergenti di dati consente di raggiungere risultati SOTA sui banchmark standard (Mahajan et al., 2018; Touvron et al., 2019; Xie et al., 2020).

1.4.3 Spiegazione del modello

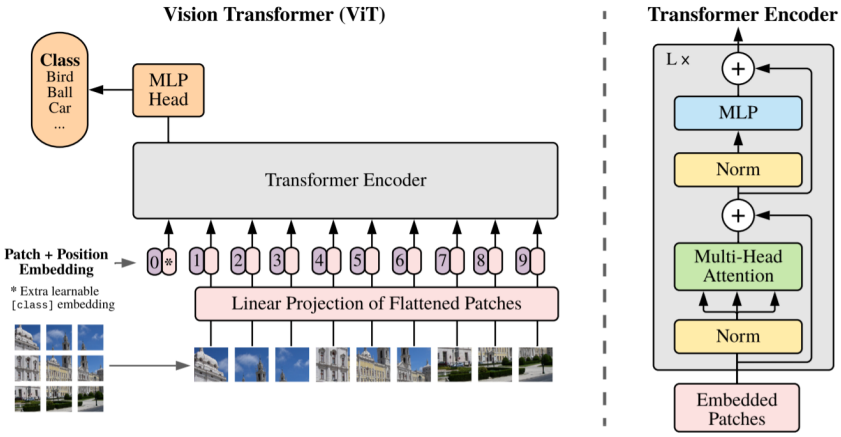


Figura 1.30: Panoramica del modello ViT. L'immagine viene suddivisa in un numero predefinito di patches, per ognuna delle quali vengono calcolati linear embedding e position embedding. Le due sequenze di embedding vengono quindi sommate tra loro; dopo di che, nel caso di task come quello di classificazione, viene anteposto alla sequenza risultante un "classification token". La sequenza di token così ottenuta viene data in pasto ad un Transformer Encoder standard

In fase di progettazione gli autori hanno adottato il principio KISS: l'idea è stata quella di ottenere un modello semplice, il più possibile versatile, scalabile e pronto all'uso. Per questo motivo il Transformer Encoder utilizzato segue la struttura proposta da Vaswani et al. nel loro paper [4].

La figura 1.30 descrive la struttura del modello ViT. Il Transformer riceve come input una sequenza monodimensionale di token embedding. Per far si che il Transformer sia in grado di gestire le immagini bidimensionali, ogni immagine $x \in \mathbb{R}^{H \times W \times C}$ viene ridimensionata in una sequenza monodimensionale di patch a due dimensioni: $x_p \in \mathbb{R}^{N \times (P^2 \cdot C)}$, dove (H, W) è la risoluzione dell'immagine originale, C è il numero di canali, (P, P) è la risoluzione di ogni patch dell'immagine, e $N = HW/P^2$ è il numero di patch ottenuto dalla frammentazione dell'immagine, nonchè la lunghezza della sequenza di patch passata in input al Transformer. Il Transformer usa un vettore latente di dimensione D costante attraverso tutti i propri layer interni, quindi ogni patch viene mappata su un vettore a D dimensioni attraverso una proiezione lineare addestrabile (si osservi Eq.1 nella prossima figura). Il risultato di questa proiezione sono i cosidetti *patch embeddings*.

In maniera analoga al token [CLS] di BERT, un learnable embedding ($z_0^0 = x_{class}$) viene anteposto alla sequenza dei patch embedding. Lo stato di

tale learnable embedding dopo l'elaborazione subita dall'encoder (z_L^0) viene utilizzato come rappresentazione dell'immagine y (Eq. 4). Sia durante il pre-training, che durante il fine-tuning z_L^0 viene dato in pasto ad una testa che si occupa di fare classificazione (*classification head*). La classification head è implementata utilizzando una MLP con un hidden layer in fase di pre-training ed un singolo layer in fase di fine-tuning.

Affinchè il Transformer possa tener conto della posizione delle patch, ai *patch embedding* vengono sommati dei *position embedding*. I position embedding vengono conservati in una sequenza monodimensionale ottenuta per apprendimento. Sono stati fatti anche test per valutare l'utilizzo di una notazione bidimensionale per la rappresentazione della posizione, ma tali test non hanno mostrato miglioramenti di performance, gli autori hanno quindi preferito anche in questo caso l'implementazione più semplice. La sequenza risultante viene data in pasto al Transformer encoder.

Il Transformer encoder è formato da una serie di layer diversi che si alternano tra loro; abbiamo:

- multiheaded self-attention layer (MSA)
- blocchi MLP (Eq. 2 e 3)
- un layer di normalizzazione (LN) viene applicato prima di ogni blocco (sia di quelli MSA che di quelli MLP)
- delle *residual connection* sono applicate dopo ogni blocco

$$\mathbf{z}_0 = [\mathbf{x}_{\text{class}}; \mathbf{x}_p^1\mathbf{E}; \mathbf{x}_p^2\mathbf{E}; \cdots; \mathbf{x}_p^N\mathbf{E}] + \mathbf{E}_{pos}, \qquad \mathbf{E} \in \mathbb{R}^{(P^2 \cdot C) \times D},\ \mathbf{E}_{pos} \in \mathbb{R}^{(N+1) \times D} \tag{1}$$

$$\mathbf{z}'_\ell = \text{MSA}(\text{LN}(\mathbf{z}_{\ell-1})) + \mathbf{z}_{\ell-1}, \qquad \ell = 1 \dots L \tag{2}$$

$$\mathbf{z}_\ell = \text{MLP}(\text{LN}(\mathbf{z}'_\ell)) + \mathbf{z}'_\ell, \qquad \ell = 1 \dots L \tag{3}$$

$$\mathbf{y} = \text{LN}(\mathbf{z}_L^0) \tag{4}$$

Un aspetto rilevante da notare è il fatto che ViT abbia molti meno *inductive bias* specifici per l'immagine di CNN. In CNN, *locality* (struttura bidimensionale usata per la *neighborhood*) e *translation equivariance* vengono combinati assieme attraverso ogni layer del modello. In ViT, *locality* e *translation equivariance* interessano solo i layer MLP, mentre i self-attention layer sono globali. Le strutture bidimensionali per la gestione della *neighborhood* sono usate con moderazione: nella prima parte del modello durante la frammentazione dell'immagine in patches, e durante il fine-tuning per regolare i position embedding per immagini con diversa risoluzione. Inoltre, a tempo di inizializzazione, i position embedding non portano informazioni relative alle posizioni bidimensionali delle patch e tutte le relazioni tra le patch legate al loro posizionamento nell'immagine devono essere apprese ex novo.

1.4.4 Comprensione del funzionamento del meccanismo di attention

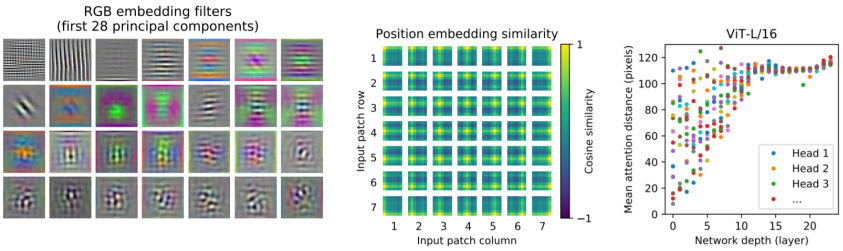


Figura 1.31: **A sinistra:** i filtri del linear embedding per i valori RGB di ViT-L/32. **Al centro:** similarità coseno in position embeddings del modello ViT-L/32. Le tessere mostrano la similarità coseno tra i position embedding della patch alla riga e colonna indicata con tutte le altre patches. **A destra:** dimensione dell'area attesa in base a head e profondità della rete. Ogni punto mostra la distranza media di attention tra le immagini per una delle 16 heads di un layer

Per cercare di comprendere come ViT elabori i dati relativi alle immagini, è utile analizzare le rappresentazioni interne al modello. Il primo layer di ViT proietta linearmente le patch in uno spazio a dimensionalità ridotta (Eq. 1). La figura 1.31 (a sinistra) mostra i componenti principali dei filtri di embedding appresi durante l'addestramento. I componenti assomigliano a funzioni base plausibili per una rappresentazione a bassa dimensionalità di una struttura fine all'interno di ogni patch.

Dopo esser stato sottoposto a proiezione, ogni position embedding viene sommato alla rappresentazione delle patch. La figura 1.31 (al centro) mostra che il modello apprende come codificare le distanze all'interno dell'immagine grazie alla similarità coseno dei position embeddings: da patch più vicine tra loro si tende a generare position embedding più simili. Si può inoltre notare una struttura riga-colonna: da patch che si trovano sulla stessa riga/colonna vengono generati embedding simili tra loro.

Il meccanismo di self-attention consente a ViT di integrare informazioni raccolte da ogni parte dell'immagine. Per comprendere fino a che punto la rete usa questa capacità, gli autori hanno calcolato la distanza media entro la quale una informazione risulta essere integrata nello spazio dell'immagine, questo viene fatto in base ai pesi dell'attention (figura 1.31 a destra). Questa misura, chiamata dagli autori "attention distance", è analoga a quella del "receptive field size" nelle reti convoluzionali (CNNs). Da tale analisi è emerso che alcune head prestano attenzione alla maggior parte dell'immagine già dai primi layer

del modello, dimostrando che ViT utilizza effettivamente la propria capacità di integrare informazioni raccolte in modo globale. Tuttavia altre attention head hanno valori di attention distance anche molto inferiori nei layer più bassi. L'attention distance aumenta all'aumentare della profondità della rete. É anche stato dimostrato che il modello presta maggiormente attenzione a quelle parti delle immagini che risultano essere più semanticamente rilevanti ai fini della classificazione 1.32.

Figura 1.32: Rappresentazione visiva delle regioni su cui il modello si concentra applicando il meccanismo di attention

1.5 Vision-and-Language Transformer

In questa sezione andrò a descrivere il modello ViLT, proposto da Kim et al. nel paper "ViLT: Vision-and-Language Transformer Without Convolution or Region Supervision" [31] pubblicato nell'anno 2021. La descrizione di questo modello è particolarmente importante ai fini del progetto in quanto è il modello scelto per elaborare i dati raccolti in fase di costruzione del dataset. Tale scelta verrà giustificata nei capitoli dedicati alla spiegazione del progetto condotto, in questa sezione mi limiterò alla descrizione del modello, facendo riferimento al paper pubblicato dagli autori. ViLT fa parte della famiglia di modelli VLP. VLP è un acronimo che sta per Vision-and-Language Pre-training e fa riferimento a quei modelli che eseguono task downstream di diverso tipo (classificazione, regressione, ...) prendendo in input ed elaborando dati testuali e immagini. Stando alle dichiarazioni degli autori [31], le motivazioni che li hanno portati a sviluppare ViLT derivano dalla volontà di superare i limiti che caratterizzano gli approcci precedentemente utilizzati nella gestione delle immagini nei modelli VLP. Questi erano basati prevalentemente su processi di estrazione delle *image feature*, molti dei quali implicano *region supervision* (ad esempio attraverso *object detection*) e l'uso di architetture convoluzionali come ResNet. Gli autori di ViLT trovano questi approcci problematici sotto due aspetti:

- *efficienza* e *velocità*: la semplice estrazione delle feature dagli input richiede un costo computazionale maggiore degli step di interazione

- *potere espressivo*: limitato dal potere espressivo del visual embedder e dal vocabolario predefinito utilizzato da quest'ultimo

Come verrà spiegato in questa sezione, ViLT ha un approccio molto semplificato all'elaborazione degli input visivi che vengono processati in maniera analoga a quelli testuali, rigettando la convoluzione. Questo diverso approccio rende il modello più snello e semplice e comporta un aumento di performance e velocità sui task downstream: gli autori riportano che dai test condotti ViLT risulta essere anche 10 volte più veloce dei suoi predecessori.

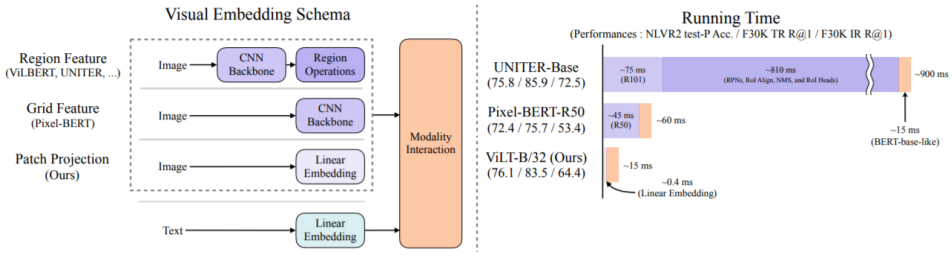


Figura 1.33: Confronto visivo tra alcune architetture VLP convenzionali e il modello ViLT. In ViLT è stata completamente rimossa la rete CNN dalla pipeline VLP, non si riscontra tuttavia un calo delle performance sui task downstream. ViLT è il primo modello VLP la cui componente modale (ovvero la parte del modello che elabora le immagini) richiede una computazione minore di quella richiesta dal transformer che si occupa dell'elaborazione delle interazioni tra i diversi tipi di input

1.5.1 Modelli VLP precedenti

Negli ultimi anni, i modelli basati su pre-training e fine-tuning sono stati applicati prima a problemi con input di natura testuale e poi, a quelli relativi a immagini. Il passo successivo è stato quello di applicare tali modelli a problemi che prendono entrambi gli input, si è iniziato così a parlare di modelli VLP.

Tra i modelli sviluppati e i paper pubblicati in relazione a tale argomento si possono citare: ViLBERT[32], UNITER[30], VL-BERT[33], VisualBERT[34], LXMERT[35], Unicoder-VL[36], "12-in-1: Multi-Task Vision and Language Representation Learning"[37], X-LXMERT[38], ImageBERT[39], "FiLM: Visual Reasoning with a General Conditioning Layer"[40], Pixel-BERT[41], Oscar[42], "Large-Scale Adversarial Training for Vision-and-Language Representation Learning"[43], ERNIE-ViL[44] e VinVL[45].

Questi modelli sono pre-addestrati utilizzando task di *Image Text Matching*, *Masked Language Modeling* che prendono come input immagini e le relative descrizioni. Sui modelli pre-addestrati in questo modo è poi possibile applicare una attività di fine-tuning che prepara il modello per uno specifico downstream task multi-modale. Parliamo di task e modelli cross-modali quando facciamo riferimento a modelli/task che utilizzano come input l'unione di dati di tipo diverso, nel caso di ViLT abbiamo dati testuali ed immagini.

Affinchè le informazioni provenienti dalle immagini possano esser date in pasto ai modelli VLP, è necessario derivare dai pixel delle immagini degli embedding densi, in maniera analoga a quanto accade per i token ricavati dalla componente testuale dell'input. Fin dalla pubblicazione del lavoro "ImageNet Classification with Deep Convolutional Neural Networks" [9](2012), si è reputato che le reti convoluzionali profonde (deep CNN) fossero necessarie per

questo step di visual embedding. La maggior parte dei modelli VLP utilizza un *object detector* preaddestrato sul dataset Visual Genome [46]. Pixel-BERT [41] costituisce un'eccezione a questa tendenza, dal momento che usa delle varianti di ResNet [10, 47] pre-addestrate su ImageNet [48] classification che eseguono l'embedding dei pixel al posto dei moduli di object detection.

Fino all'arrivo di ViLT la maggior parte degli studi focalizzati sul miglioramento delle performance puntavano ad aumentare le capacità dei visual embedders. Questo ha come effetto collaterale un appesantimento di tali moduli. Sebbene il calo di performance così prodotto possa esser tamponato per quanto riguarda gli studi accademici attraverso il caching delle region features in fase di training, non si può fare lo stesso su applicazioni reali dal momento che le query devono andar incontro a un lento processo di estrazione.

Volendo risolvere questo problema, gli autori di ViLT si sono focalizzati su alleggerire e velocizzare la generazione di embedding per la componente visuale dell'input. Lavori recenti [7, 49] hanno dimostrato che una semplice proiezione lineare di una patch è in grado di produrre embedding adatti ad essere dati in pasto ai Transformers. L'intuizione degli autori di ViLT è stata quella che i moduli Transformers, usati nei modelli VLP per elaborare le interazioni tra input di natura diversa, fossero anche in grado di gestire l'elaborazione di features provenienti da immagini, sostituendo in questa operazione i visual embedder e processando quindi i dati visivi allo stesso modo di quelli testuali.

1.5.2 Architettura dei modelli VLP

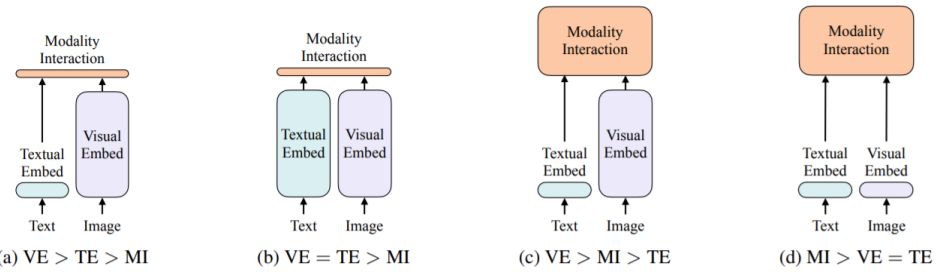


Figura 1.34: Quattro categorie di VLP models. L'altezza dei rettangoli denota il costo computazionale. VE sta per *Visual Embedder*, TE sta per *Textual Embedder*, MI sta per *Modality Interaction*

Tassonomia dei modelli VLP

Gli autori di ViLT hanno proposto una tassonomia per distinguere i diversi modelli VLP basandosi su due aspetti:

- si distinguono modelli in cui le due modalità di input (VE, TE) hanno o meno lo *stesso livello di espressività* in termini di parametri e/o computazione dedicati

- si distinguono modelli in cui le due modalità *interagiscono in una rete profonda* da quelli in cui l'interazione avviene in una rete bassa

le possibili combinazioni di questi due aspetti portano a definire 4 archetipi come mostra la figura 1.34.

Alla figura 1.34(a) appartengono modelli VSE (*visual semantic embedding*) come VSE++ [50] e SCAN [51]. I modelli appartenenti a questo archetipo usano degli embedders separati per immagine e testo; i visual embedders risultano molto più complessi e pesanti di quelli testuali. Dopo aver ricavato gli embedding di entrambi i tipi di input queste vengono combinate attraverso prodotto scalare o un layer di attention superficiale, non profondo.

Un secondo archetipo è quello illustrato nella figura 1.34(b) e a cui appartengono modelli come CLIP [52]. Questi modelli usano embedders separati per gli input di diverso tipo, ma la complessità computazionale è la stessa. L'interazione tra gli embedding di diverso tipo avviene sempre in una rete bassa o attraverso prodotto vettoriale. Modelli appartenenti a questa categoria possono ottenere buone performance su alcuni specifici downstream task, come mostrato da CLIP sul task image-to-text retrieval, tuttavia incontrano molta difficoltà ad apprendere il resto dei task, come dimostrato da Suhr et al. [53] usando il dataset NLVR2. Tali difficoltà derivano secondo gli autori dall'eccessiva semplicità del modulo MI: dai test effettuati sembra infatti che una fusione "semplice" dei risultati ottenuti dagli embedder unimodali, anche qualora questi abbiano buone performance, non produca risultati soddisfacenti su task complessi che prendono in input più tipi di dati. Ne deriva la necessità di individuare un più complesso e rigorosamente strutturato schema di interazione inter-modale.

Queste conclusioni hanno portato i ricercatori a sviluppare modelli VLP con un modulo Modality Interaction più complesso, come mostrato in figura 1.34(c). Modelli che adottano architetture di questo tipo solitamente usano dei Transformer per modellare le interazioni tra le informazioni ricavate dai diversi tipi di input, abbiamo quindi l'utilizzo di reti neurali profonde. Gli input del modulo che gestisce l'interazione sono sempre i risultati dell'elaborazione condotta da textual e visual embedders. Anche in questo caso, come in figura 1.34(a), i visual embedder risultano più complessi di textual embedder dal momento che sono il più delle volte implementati attraverso reti convoluzionali che estraggono e si occupano dell'embedding delle image features. Rientrano in questa famiglia anche modelli come quelli modulation-based e vision-and-

language models [40, 54] che usano una rete CNN come visual embedder, una rete RNN come textual embedder, e una rete CNN a moduli come MI.

Il modello ViLT risulta essere il primo modello a seguire l'archetipo riportato in figura 1.34(d), in cui i layer di embedding per i pixel risultano poco profondi, come quelli usati per i text token. All'interno di architetture di questo tipo la maggior parte della computazione si concentra nel modulo che gestisce l'interazione tra gli embedding di diverso tipo.

Schema per modality interaction

Al centro dei modelli VLP moderni vi è solitamente un Transformer. Questo prende in input gli embedding testuali e visivi, dopo di che impara a riconoscere le interazioni inter ed intra modali, e restituisce in output una sequenza di feature determinata dalla combinazione di informazioni apprese.

Secondo una classificazione proposta da Buglianello et al. [55] i modelli di interazione possono essere suddivisi in due gruppi:

- quelli che adottano un approccio *single-stream*. Fanno parte di questa categoria modelli come Visual-BERT [34] e UNITER [30]in cui i layer operano su una concatenazione di embedding derivati da testo ed immagine in input
- quelli che adottano un approccio *dual-stream*. Categoria a cui appartengono modelli come ViLBERT [32] e LXMERT [35], in cui le due modalità non vengono concatenate a livello di input

ViLT è stato realizzato cercando di minimizzare la complessità del modello, per questo motivo segue l'approccio single-stream, che richiede meno parametri di dual-stream.

Schema per Visual Embedding

Quasi tutti i modelli VLP antecedenti a ViLT condividono gli aspetti relativi all'elaborazione testuale riprendendo da BERT textual embedder, tokenizer, word embedding, position embedding. Tali modelli differiscono invece per i moduli relativi a visual embedder, che costituisce solitamente il collo di bottiglia per questi modelli. ViLT si differenzia da questi modelli introducendo la *patch projection* al posto di *region features* o *grid features* solitamente utilizzate e la cui estrazione richiede l'impiego di moduli particolarmente pesanti.

Region Feature

I modelli VLP usano per lo più region features, anche note come bottom-up features [56]. Queste vengono calcolate utilizzando un *object detector*,

solitamente scegliendone uno già noto in letteratura e quindi pronto all'uso, ad esempio Faster R-CNN [57].

Solitamente le region features vengono utilizzate in questo modo:

- per prima cosa, un *region proposal network* (RPN) determina le *region of interest* (RoI) in base alle grid features raccolte attraverso pooling dalla rete CNN usata come backbone

- a quel punto, usando la tecnica del non-maximum suppression (NMS), si riduce a poche migliaia il numero di RoI

- le RoI vengono quindi raggruppate usando operazioni come RoI Align [58] e vengono date in input a delle RoI heads, diventando poi region features

- Si applica quindi nuovamente NMS ad ogni classe per ridurre il numero di features a poche centinaia

Il processo descritto implica diversi fattori che possono incidere sulle performance e sull'esecuzione, ad esempio: la backbone, lo stile di NMS o le RoI heads utilizzate. Per quanto riguarda tali scelte:

- tra le **backbones** più utilizzate abbiamo ResNet-101 [32, 35, 33] e ResNet-152 [34, 45]

- solitamente **NMS** viene applicata in modo distinto tra le classi esistenti. L'applicazione di NMS ad ogni classe può costituire un collo di bottiglia con un grande numero di classi come dimostrato sul dataset VG [59] (1600 classi). Per cercare di ridurre questo problema è recentemente stato introdotto class-agnostic NMS [45]

- per quanto riguarda **RoI heads** invece, inizialmente venivano utilizzate C4 [56] heads. Successivamente sono state introdotte FPN-MLP heads [60]. Dal momento che le heads operano su ogni RoI esse costituiscono un limite importante a tempo di esecuzione.

Per quanto possano esser leggeri, object detectors sono raramente più veloci delle backbone o di una operazione di convoluzione svolta su un singolo layer. Effettuare in anticipo il freezing della visual backbone e il caching delle region features può aiutare in fase di training ma non allevia la complessità incontrata durante l'inferenza, senza considerare che queste operazioni possono portare ad una riduzione delle performance.

Grid Features

Tra gli altri utilizzi, il reticolo di feature (grid feature) restituito come output da reti neurali convoluzionali come ResNet può essere usato come insieme di feature visive da dare in pasto ad un modello VLP in fase di addestramento. L'uso diretto di grid features è stato per la prima volta proposto con i modelli VQA-specifici [59, 54], principalmente per evitare l'utilizzo delle operazioni estremamente lente di region selection. Gli autori di ViLT hanno deciso di non adottare grid features poichè le reti CNN profonde risultano tuttora talmente costose in termini computazionali da risultare uno dei moduli più costosi nei modelli che le utilizzano.

Patch Projection

Per minimizzare i costi dovuti a questa operazione, gli autori hanno scelto di adottare lo schema di visual embedding più semplice: la *proiezione lineare* applicata sulle patch ricavate dalla frammentazione delle immagini. Come illustrato nella sezione precedente, l'utilizzo di patch projection embedding è stato introdotto da ViT [7] per il task di classificazione delle immagini. La patch projection semplifica il passo di embedding dell'input visivo talmente tanto da renderne il costo paragonabile a quello di textual embedding, che a sua volta consiste in una semplice operazione di proiezione (ovvero di lookup). ViLT utilizza 32x32 patch projection, che richiede solo 2.4M di parametri. Questi costi determinano il netto miglioramento di performance rispetto ai risultati ottenuti dalle molto più pesanti backbone ResNet e dai detection components. Tali vantaggi vengono riscontrati anche in fase di esecuzione. Per una comparazione tra le performance dei vari modelli si può fari riferimento alla figura 1.33.

1.5.3 Il modello ViLT

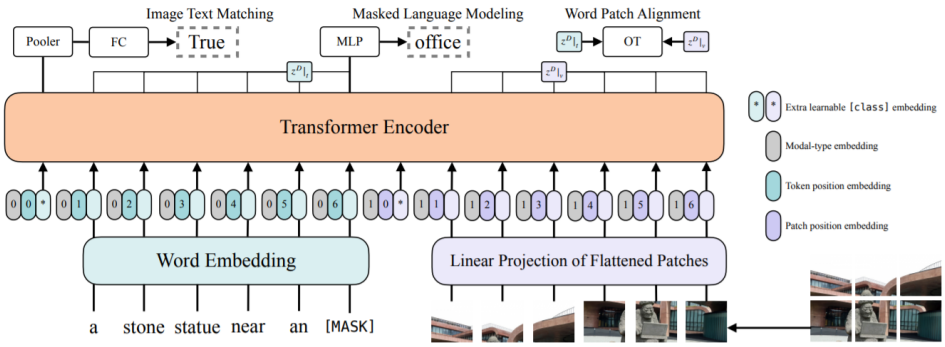


Figura 1.35: Panoramica del modello

ViLT è un modello VLP dalla struttura molto concisa, sopratutto per quanto riguarda la pipeline di elaborazione dei visual embedding e come già spiegato segue l'approccio *single stream*.

Per quanto riguarda la parte di modello che elabora le immagini, i pesi vengono inizializzati usando quelli del modello pre-addestrato ViT. Tale inizializzazione sfrutta il potere dei layer di interazione per elaborare le features visive senza la presenza di un visual embedder separato.

$$\bar{t} = [t_{\text{class}}; t_1 T; \cdots; t_L T] + T^{\text{pos}} \tag{1}$$

$$\bar{v} = [v_{\text{class}}; v_1 V; \cdots; v_N V] + V^{\text{pos}} \tag{2}$$

$$z^0 = [\bar{t} + t^{\text{type}}; \bar{v} + v^{\text{type}}] \tag{3}$$

$$\hat{z}^d = \text{MSA}(\text{LN}(z^{d-1})) + z^{d-1}, \qquad d = 1 \ldots D \tag{4}$$

$$z^d = \text{MLP}(\text{LN}(\hat{z}^d)) + \hat{z}^d, \qquad d = 1 \ldots D \tag{5}$$

$$p = \tanh(z_0^D W_{\text{pool}}) \tag{6}$$

Come visto nella sezione dedicata, ViT è formato da dei blocchi impilati l'uno sull'altro, tra questi abbiamo il layer multihead di self-attention (MSA) e un layer MLP. La posizione del layer di normalizzazione (LN) risulta l'unica differenza rispetto a BERT:

- in BERT LN viene posto dopo MSA e MLP, per questo motivo si parla di "post-norm"
- in ViT LN viene posto prima di MSA e MLP, per questo motivo si parla di "pre-norm"

A partire dal testo in input $t \in \mathbb{R}^{L \times |V|}$, viene calcolato l'embedding $\bar{t} \in \mathbb{R}^{L \times H}$ utilizzando una matrice per il word embedding $T \in \mathbb{R}^{|V| \times H}$ ed una matrice per il position embedding $T^{pos} \in \mathbb{R}^{(L+1) \times H}$.

L'immagine in input $I \in \mathbb{R}^{C \times H \times W}$ viene suddivisa in più patch e trasformata in forma flat (ovvero le patch vengono memorizzate in una sequenza); si ottiene così $v \in \mathbb{R}^{N \times (P^2 \cdot C)}$, dove (P, P) è la risoluzione delle patch e $N = HW/P^2$. Dopo di che viene applicata la proiezione lineare $V \in \mathbb{R}^{(P^2 \cdot C) \times H}$ e vengono calcolati i position embedding $V^{pos} \in \mathbb{R}^{(N+1) \times H}$, si calcola quindi l'embedding di v: $\overline{v} \in \mathbb{R}^{N \times H}$.

Gli embedding di testo e immagine vengono sommati con i corrispettivi vettori di embedding che indicano il tipo di input (testo/immagine) $t^{type}, v^{type} \in \mathbb{R}^H$; questi vengono detti modal-type embedding. Il risultato viene poi concatenato a formare un'unica sequenza chiamata z^0. Il vettore z viene aggiornato iterativamente mentre attraversa i layer di un transformer con profondità D. Il risultato di tali trasformazioni è un vettore z^D in cui sono state codificate informazioni testuali e visive tenendo conto del loro contesto. p è una rappresentazione pooled dell'intero input cross-modale e viene ottenuta applicando la proiezione lineare $W_{pool} \in \mathbb{R}^{H \times H}$ e la tangente iperbolica sul primo indice della sequenza z^D, ovvero il corrispettivo del token [CLS] in BERT.

Per tutti gli esperimenti condotti, gli autori hanno utilizzato i seguenti parametri: hidden size H = 768; profondità dei layer D = 12; patch size P = 32; MLP size = 3072; numero di attention head = 12. I pesi utilizzati sono quelli recuperati da ViT-B/32 re-addestrato su ImageNet, da quì il nome ViLT-B/32. ViT-B/32 è un codice che identifica il modello Vision Transformer nella versione "Base": 12 Layers, Hidden Features=768, MLP size=3072, 12 Heads, 86M parametri; il valore "32" indica che l'immagine viene decomposta in patch di dimensione 32×32.

Obiettivi usati per il pre-training

ViLT viene addestrato su 2 obiettivi comunemente utilizzati per il pre-training di modelli VLP: *image text matching* (ITM) e *masked language modeling* (MLM).

Image Text Matching

Questo task prevede che le immagini che devono essere passate assieme al testo corrispondente vengano sostituite da altre immagini con una probabilità del 50%. Una ITM head, costituita da un solo layer lineare, effettua una proiezione

delle feature in output p su logits calcolati su una classe binaria, viene quindi calcolata negative log-likelihood loss come funzione di loss legata alla ITM head.

Masked Language Modeling

L'obiettivo, in questo caso, è quello di predire una label originale sostituita da un text token mascherato t_{masked} basandosi sul vettore contestualizzato $z^D_{masked}|_t$. Seguendo l'euristica proposta da Devlin e collaboratori [6], ogni token t viene mascherato con una probabilità del 15%. Per eseguire tale task si utilizza una head formata da due layer MLP e MLM, che prende in input $z^D_{masked}|_t$ e restituisce in output dei logits calcolati sul vocabolario adottato, allo stesso modo di quanto avviene con l'obiettivo MLM di BERT. La loss MLM viene quindi calcolata come negative log-likelihood loss relativa ai token mascherati.

Whole Word Masking

É una tecnica di mascheramento che maschera tutti i token consecutivi che sono stati generati da parti della stessa parola. Questa versione di MLM si è rivelata efficace quando è stata applicata a BERT e a Chinese BERT [61].
La decisione di adottare questa accortezza è derivata dall'ipotesi che un mascheramento dell'intera parola fosse necessario per forzare il modello VLP a basarsi anche sull'altra modalità e non solo su quella testuale. Per esempio, utilizzando il tokenizer *bert-base-uncased*, la parola "giraffe" viene tokenizzata in 3 token, ognuno dei quali codifica una parte di parola: ["gi","##raf","##fe"]. Immaginando di mascherare solo il token centrale, si otterrebbe: ["gi","[MASK]","##fe"], a quel punto il modello potrebbe facilmente risalire al token mascherato basandosi solo sui token non mascherati che derivano dai restanti frammenti della parola "giraffe", ovvero ["gi","##fe"]; non utilizzerebbe quindi l'informazione proveniente dalla componente visiva per ricostuire la label mascherata e questo limiterebbe l'efficacia dell'attestramento nell'apprendere le connessioni inter-modali. In fase di addestramento, gli autori hanno scelto di mascherare interamente ogni parola con una probabilità del 15%.

Image Augmentation

Quella della Image Augmentation è una tecnica molto utile per aumentare la capacità di generalizzazione dei modelli che prendono in input immagini. L'idea è quella di dare in pasto al modello, in fase di training, delle immagini leggermente modificate attraverso varie tecniche quali rotazione, traslazione,modifica di contrasto o luminosità, mascheramento di alcuni pixel, ecc. Tali tecniche non devono modificare completamente l'immagine: il modello deve riuscire a

ricavare dall'immagine modificata la maggior parte delle informazioni principali che avrebbe ricavato dall'immagine originale. In questo modo è possibile aumentare il numero di immagini da sottoporre al modello e statisticamente si osserva un miglioramento nella capacità di generalizzazione del modello. I lavori condotti su modelli VLP precedenti a ViLT hanno raramente esplorato i benefici apportati da questa tecnica. Gli autori di ViLT hanno invece tratto benefici dell'image augmentation applicando RandAugment [62] in fase di fine-tuning, usando come iperparametri N=2 e M=9. Per l'applicazione su ViLT sono state escluse 2 delle tecniche solitamente utilizzate in RandAug:

- color inversion: poichè il testo può contenere anche informazioni sul colore, modificando il colore dei pixel si comprometterebbe la capacità del modello di apprendere la relazione tra le parole che indicano un colore e i colori che caratterizzano l'immagine
- cutout: per evitare la perdita di piccoli oggetti sparsi nell'immagine

1.6 Progetti correlati

In passato sono stati pubblicati alcuni lavori in cui venivano effettuate previsioni sul numero di esemplari venduti per ogni prodotto. Questi lavori hanno un obiettivo simile a quello di questa tesi: il numero di esemplari venduti per un determinato prodotto è l'aspetto principale tenuto in considerazione nel calcolo del successo del prodotto da me utilizzato. Le soluzioni proposte differiscono tra loro per le informazioni utilizzate per effettuare la previsione, per i modelli utilizzati e per le categorie di prodotti a cui vengono applicate. Tra il 2011 e il 2016 sono stati proposti alcuni lavori [63, 64, 65] che fondano le proprie previsioni sulle informazioni contenute nelle recensioni lasciate dagli acquirenti precedenti. Queste soluzioni non possono tuttavia essere utilizzate dai venditori prima di aver presentato il prodotto al pubblico, dal momento che si basano su dati generati da clienti. Nel 2017 Pryzant e alcuni collaboratori hanno pubblicato una soluzione [66] in cui la previsione veniva effettuata ad opera di una rete LSTM sulla base delle informazioni contenute nella sola descrizione fornita dal venditore. La soluzione di Pryzant et al. è, in questo senso, quella più simile al lavoro svolto in questa tesi perchè effettua la previsione basandosi su conoscenze note prima della commercializzazione del prodotto. Questo lavoro non tiene tuttavia conto delle informazioni che possono essere estratte da immagini che ritraggono il prodotto. In un e-commerce, queste informazioni possono essere anche molto rilevanti nel catturare l'attenzione del potenziale cliente come dimostrato nel lavoro pubblicato da Di et al. [67]. Tra i lavori che prendono in considerazione anche le immagini vi è la tesi prodotta da Michael Kranzlein nel 2018, intitolata "A Multiple Classifier System for Predicting Best-Selling Amazon Products" [68]. Di seguito riporto un'analisi dei lavori citati.

Deriving the Pricing Power of Product Features by Mining Consumer Reviews

In questo paper [63], Archak et al. utilizzano i dati contenuti nelle review di fotocamere e videocamere vendute su Amazon per predire in numero di vendite per ogni prodotto. Archak et al. evidenziano come le soluzioni precedenti a quella da loro pubblicata non utilizzino le informazioni contenute nel testo delle recensioni, ma solo alcuni metadati ricavati da queste: numero di recensioni, numero di recensioni positive/negative, numero di recensioni totali; questo porta ad una grande perdita di informazioni che potrebbero essere ottenute attraverso tecniche di machine learning. Archak et al. decidono di affrontare il problema estraendo dal testo le feature utilizzate dai clienti per descrivere il prodotto; una volta determinate quelle più importanti, utilizzano una tecnica di regressione

ispirata al metodo dei prezzi edonici per determinare quanto venderà il prodotto a partire dai valori associati alle feature, determinati a partire dalle recensioni. Per quanto riguarda l'estrazione delle feature fanno uso di due soluzioni: la prima è completamente automatizzata e basata sull'utilizzo di un POS (Part of Speech) tagger per determinare termini candidati a divetare feature; a supporto di questa viene utilizzato anche un approccio semi-automatizzato per la scoperta di feature basato su crowdsourcing: degli operatori umani individuano le feature contenute in alcune recensioni, per questa seconda soluzione Archak et al. hanno utilizzato i servizi messi a disposizione da Amazon Mechanical Turk. Durante l'attività di previsione, oltre al valore delle feature, hanno tenuto in considerazione anche statistiche fornite da Google relative al numero di ricerche effettuate per il prodotto preso in considerazione e il relativo brand; questi dati sono stati utilizzati per tenere in considerazione anche i fattori esterni alle recensioni in grado di influenzare la richiesta del prodotto. Per quanto riguarda le previsioni, sono stati sviluppati due task: uno di classificazione ed uno di regressione; entrambi si basano sui dati estratti dalle recensioni pubblicate nell'ultima settimana. Il task di classificazione determina se le vendite del prodotto aumenteranno o diminuiranno; vengono utilizzati 4 classificatori: logistic regression (si rivelerà essere quello che fornisce i risultati migliori), support vector machine, decision tree, random forest. Il task di regressione cerca di prevedere le vendite ottenure la settimana successiva. Confrontando risultati ottenuti con quelli raggiunti dalle soluzioni precedenti Archak et al. hanno dimostrato che le informazioni estrapolate dai testi delle recensioni sono più efficaci di metadati numerici (ranking medio del prodotto, numero di recensioni, prezzo, ecc.) per effettuare previsioni sulle vendite. La soluzione presentata consente anche di estrarre per ciascun prodotto le feature di maggiore interesse per il cliente.

Predicting online product sales via online reviews, sentiments, and promotion strategies

In questo paper [64], Chong et al. propongono delle soluzioni software per raccogliere dati relativi a prodotti venduti su Amazon ed effettuare previsioni sul numero di esemplari venduti per prodotto. Vengono raccolti dati di 12000 prodotti appartenenti alle seguenti categorie: fotocamere, videocamere, televisori, pc e altri prodotti di elettronica caratterizzati da una shelf life ridotta. Per ogni prodotto vengono raccolti metadati, recensioni, informazioni relative alle strategie promozionali utilizzate. Per far questo, Chong et al. hanno sviluppato una architettura big data basata su chiamate input/output asincrone in grado di fare scraping delle pagine dei prodotti amazon. I testi delle recensioni sono stati elaborati utilizzando modelli NLP che hanno consentito di estrarre feature

in grado di descrivere i prodotti. Le recensioni sono andate incontro anche ad una seconda elaborazione: sono state suddivise in gruppi (nello stesso gruppo vengono raccolte recensioni pubblicate a breve distanza di tempo) e su ciascun gruppo vengono applicati dei modelli di sentiment analysis per determinare il modo in cui un potenziale cliente può essere influenzato da queste recensioni. Tutte le informazioni così raccolte vengono utilizzate per prevedere il numero di esemplari venduti per ogni prodotto. Durante la previsione vengono effettuate analisi per valutare l'importanza delle diverse feature nel task di previsione. Di seguito riporto quanto emerge da questo studio:

- ogni feature utilizzata ha un potere predittivo molto limitato se presa in considerazione singolarmente, tuttavia la composizione di tutte le feature porta ad ottenere una buona capacità predittiva
- le informazioni derivate dalla sentiment analysis hanno un bassissimo potere predittivo se considerate singolarmente, tuttavia quando combinate all'informazioni relative a quantità e valenza delle recensioni costituiscono una nuova feature più complessa e rilevante
- tra le feature con influenza maggiore sulla previsione delle vendite abbiamo: quantità di recensioni, numero di ricerche online relative al prodotto, qualità del customer service offerto dal venditore, strategie di marketing adottate
- tra le feature con influenza minore sulla previsione delle vendite abbiamo: consegna a domicilio gratuita (viene considerata un servizio che deve essere offerto non un bonus, quindi ha al più una influenza negativa quando non viene garantita) e informazioni derivate dalla sentiment analysis

Estimating the Helpfulness and Economic Impact of Product Reviews: Mining Text and Reviewer Characteristics

In questo paper [65], Ghose et al. utilizzano i dati contenuti nelle review di prodotti venduti su Amazon per predire il numero di copie vendute per ogni prodotto e l'utilità delle nuove recensioni. Anche in questo caso le previsioni vengono effettuate principalmente a partire da feature testuali estrapolate dal testo delle recensioni, a queste si aggiungono poi alcune informazioni sul recensore. Vengono presi in considerazione prodotti appartenenti a 3 categorie: 144 lettori audio-video, 109 telecamere digitali, 158 DVD. I prodotti vengono scelti dall'elenco dei più venduti da Amazon tra Gennaio e Marzo 2015. Per ogni prodotto vengono conservate informazioni relative a: prezzo al dettaglio, data di commercializzazione, l'elenco di review e il sales rank. Il sales rank è un valore che indica la posizione del prodotto in una graduatoria che determina

quante copie sono state vendute: quando si parla di numero di prodotti venduti o domanda di mercato per un prodotto si fa riferimento a questo campo. Per ogni review vengono memorizzare informazioni su testo, utilità, recensore. L'utilità è un valore numerico che indica quante persone hanno trovato utile la recensione. Per ogni recensore vengono salvate eventuali informazioni personali rivelate (nome reale, hobby, residenza, lavoro, età, ecc.), l'elenco di recensioni effettuate, il rating medio delle sue recensioni in termini di utilità ed eventuale appartenenza alla categoria "top-reviewer". Tra gli aspetti maggiormente tenuti in considerazione durante l'analisi vi sono la leggibilità e la soggettività della recensione. La leggibilità riguarda la facilità di lettura del testo della recensione: vengono quindi tenute in considerazione lunghezza delle frasi (numero medio di termini e caratteri) e errori di ortografia. La soggettività invece riguarda quanto la recensione si concentra su caratteristiche oggettive o soggettive del prodotto. Ghose et al. hanno analizzato l'impatto di questi aspetti sui task implementati:

- previsione del sales rank relativo ad un prodotto: è un task di regressione
- previsione dell'utilità di una recensione: questo secondo task è stato modellato sia come regressione, considerando l'utilità un valore compreso nell'intervallo [0,1], sia come classificazione su due classi: utile/non utile. Sono stati utilizzati due modelli per la classificazione SVM e Random Forest; queste ultime si sono rivelate essere il modello più efficace ed efficiente in termine di risorse richieste dal training

Tra i risultati emersi nello studio dell'influenza sul prezzo abbiamo che:

- le recensioni che contengono un misto di valutazioni soggettive ed oggettive vengono associate negativamente al prezzo rispetto a recensioni che contengono solo valutazioni oggettive o solo valutazioni soggettive
- per prodotti appartenenti alla categoria search goods come le telecamere, recensioni con una maggiore leggibilità vengono associate a più alti sales rank
- le recensioni contenenti molti errori ortografici sono associate a più bassi valori di sales rank per prodotti appartenenti alla categoria degli experience goods (DVD). Il numero di errori ortografici risulta irrilevante per prodotti appartenenti alla categoria dei search goods (lettori audio-video e telecamere).
- recensioni che hanno un rating fortemente negativo possono essere associate a prodotti con un alto sales rank se sono molto dettagliate e veicolano bene l'informazione

Per quanto riguarda la previsione dell'utilità di una recensione è emerso che:

- le recensioni che contengono un misto di valutazioni soggettive ed oggettive vengono considerate più informative di recensioni che riportano solo valutazioni oggettive o soggettive
- le recensioni con una maggiore leggibilità ed una minore quantità di errori di ortografia risultano più utili nel caso di prodotti appartenenti alla categoria DVD.
- i classificatori Random Forest hanno più successo nella previsione dell'utilità di una recensione nel caso di search goods (lettori audio-video e telecamere) piuttosto che in experience goods (DVD)
- recensioni che valutano un prodotto in modo fortemente negativo possono ricevere una valutazione di utilità molto bassa dagli utenti non perchè la recensione non sia utile, ma perchè gli utenti si trovano in disaccordo con quanto espresso nella recensione
- recensioni relative a prodotti che ricevono valutazioni ad elevata variabilità spesso ricevono valutazioni di utilià ad elevatà variabilità

Un ultima valutazione fatta è quella sull'importaza relativa delle tre categorie di feature: "feature relative al recensore", "soggettività nella recensione", "leggibilità della recensione". É emerso che eseguendo lo stesso task utilizzando una sola di queste tre categorie o utilizzandole tutte e 3 il risultato non cambia; inoltre, a partire da una coppia composta da due categorie qualsiasi tra le tre, è possibile determinare la terza categoria. Questo vuol dire che le tre categorie sono intercambiabili.

Predicting Sales from the Language of Product Descriptions

Con questo paper [66], Pryzant et al. propongono un modello in grado di prevedere il numero di prodotti venduti a partire da features estratte dalla descrizione del prodotto fornita dal venditore. Il fatto che la previsione venga fatta a partire da informazioni messe a disposizione dal venditore rende questo modello interessante per i commercianti. I venditori possono infatti utilizzare il modello prima di mettere in commercio il prodotto per massimizzare le possibilità di successo di vendita. Pryzant et al. fanno notare l'importanza del modo in cui il venditore parla del prodotto all'interno della descrizione: spesso lo stesso prodotto è venduto da più commercianti con un successo di vendita diverso; una possibile spiegazione di questo fenomeno è che la narrazione del prodotto fatta dal commerciante di maggior successo sia quella più efficace.

Gli autori del paper vogliono quindi sfruttare le descrizioni dei prodotti per prevederne il successo di vendita e nel far questo determinare quali frasi e termini rendono una descrizione efficace. Il modello proposto da Pryzant et al. è in grado di raggiungere questo obiettivo escludendo dalla ricerca quelle frasi e termini che fanno riferimento ad alcuni fattori di confusione (prezzo, brand e specifico prodotto), in questo modo le espressioni risultati sono adatte ad essere utilizzate per qualsiasi prodotto. Queste espressioni che vengono ricercate sono le feature utilizzate durante la previsione.

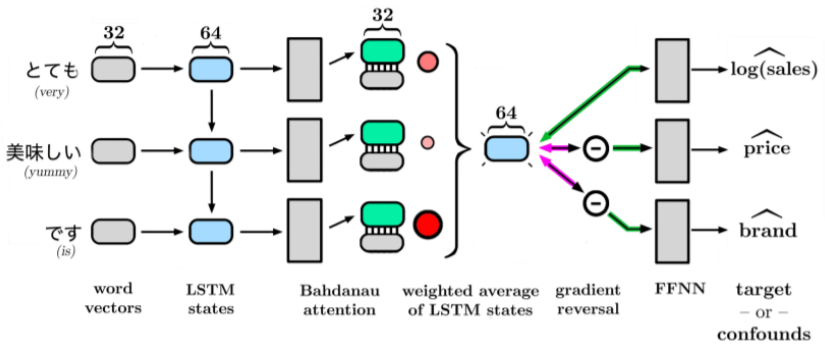


Figura 1.36: Architettura del modello RNN+GF proposto da Pryzant et al. In figura vengono mostrate esplicitamente tutte le operazioni e dimensionalità. I rettangoli con gli angoli smussati rappresentano vettori, le matrici risultati da una moltiplicazione tra vettori sono riportati come rettangoli semplici. Tutti i parametri che vengono addestrati sono contraddistinti dal colore grigio, mentre i valori calcolati dinamicamente sono di colore verde. I gradient reversal layers moltiplicano il gradiente per -1 durante la backpropagation. Il token candidato a generare una feature è quello a cui il modello assegna il più alto valore di attention al termine dell'addestramento

La figura 1.36 riporta uno schema del nuovo modello neurale sviluppato da Pryzant et al. per estrarre feature dalle descrizioni. Di seguito descrivo il funzionamento. Le descrizioni vengono segmentate in token, questa operazione può essere fatta utilizzando un tokenizer ed ottenendo quindi dei morfemi, oppure utilizzando Byte-Pair Encoding (BPE). I token vengono quindi dati in pasto al modello, che viene addestrato utilizzando tre task: previsione delle vendite, del prezzo e del brand. Bisogna tuttavia notare l'utilizzo peculiare degli ultimi due task: mentre vogliamo che il modello apprenda il task di previsione delle vendite, vogliamo che sia incapace di prevedere prezzo e brand, in questo modo si scoraggia l'utilizzo di feature correlate a questi due fattori di confusione. É anche importante notare che questo modello fa uso del meccanismo di attention Bahdanau [3]: durante l'addestramento ad ogni token

viene associato un punteggio di attention che aumenta in base alla rilevanza del token per l'esecuzione del task. Al termine dell'addestramento è quindi possibile recuperare i token con il punteggio di attention più elevato; da questi token vengono ricavate le feature che indicano gli aspetti più rilevanti contenuti in una descrizione.
Il modello è stato applicato a dati relativi a prodotti venduti su uno store online giapponese chiamato Rakuten nell'anno 2012. I prodotti appartengono a due categorie molto diverse tra loro: la prima categoria comprende prodotti al cioccolato, caratterizzati da una grande variabilità poichè spesso sono prodotti fatti a mano, di conseguenza anche le descrizioni risultano molto diverse tra loro; la seconda categoria riguarda prodotti di categoria Salute, spesso sono prodotti farmaceutici e la variabilità è molto ridotta.
Sono poi stati usati i modelli Odds Ratio (OR) e Mutual Information (MI) per valutare la rilevanza delle feature estratte dal modello di Pryzant et al. rispetto ad altre informazioni tra cui i fattori di confusione, e quindi informazioni relative al brand, al prezzo, ecc. Da questo confronto è emerso che tenendo in considerazione anche le feature più rilevati estratte dal modello RNN+GF si ottiene un notevole aumento della capacità predittiva, indipendentemente dal dataset e dal metodo di selezione delle feature adottato.
É stata valutata l'efficacia del modello anche per quanto riguarda l'esclusione dei fattori di confusione: è stata valutata la correlazione tra le feature estratte da RNN+GF e i fattori di confusione, il risultato è poi stato confrontato con la correlazione tra fattori di confusione e feature estratte usando altri metodi (L1, MI, OR), ne è risultato che i valori di correlazione più bassi sono quelli ottenuti a partire dalle feature estratte da RNN+GF. A riprova di ciò, le feature più rilevanti selezionate da RNN+GF non fanno riferimento a Brand, prezzo e prodotto.
Da una analisi delle feature più rilevanti selezionate da RNN+GF è emerso che queste fanno prevalentemente parte di quattro categorie: termini informativi ("formato famiglia", "souvenir", ecc.), termini che fanno riferimento ad autorità ("dottor", "staff", ecc.), termini che indicano stagionalità ("Natale", "Capodanno", ecc.), appellativi onorifici (è legato all'utilizzo di appellativi onorifici, usati come forma di rispetto nella lingua giapponese).

Is a Picture Really Worth a Thousand Words? - On the Role of Images in E-commerce

In questo paper [67], Di et al. studiano l'importanza dell'utilizzo di immagini nel contesto dei commerci online. Il lavoro parte dal presupposto che uno dei problemi principali incontrati da un cliente di uno store online sia quello di non poter toccare con mano e quindi analizzare il prodotto come farebbe in

un negozio fisico prima dell'acquisto. Questa mancanza viene compensata ricercando informazioni sul prodotto: parte di queste vengono solitamente fornite dal venditore, un'altra parte consistente di informazioni può essere ricavata dalle recensioni di chi ha già comprato il prodotto e rende pubblici i propri feedback. Le informazioni testuali sono spesso accompagnate da immagini, che possono essere immagini stock o fotografie scattate al prodotto venduto. Gli autori di questo paper indagano il contributo delle immagini nel commercio online. Vengono quindi costruiti due dataset a partire da dati raccolti tra il 2011 e il 2012 in un marketplace online peer to peer. Il primo dataset contiene i dati che descrivono i prodotti e i relativi venditori; questo dataset viene utilizzato per studiare conversion rate e tipi di venditori. Il secondo dataset invece raccoglie informazioni relative al comportamento degli acquirenti e viene utilizzato per studiare il comportamento relativo alla ricerca dei prodotti. Per quanto riguarda le immagini, gli autori si sono concentrati su tre caratteristiche chiave: formato, numero di immagini per prodotto e qualità delle immagini. Di seguito riporto i principali risultati emersi da questo studio. Maggiore è il numero, la qualità e le dimensioni delle immagini messe a disposizione maggiore risulta essere il numero di vendite previsto, ed anche il conversion rate; questo fenomeno viene spiegato con il fatto che le condizioni descritte mettono a disposizione del cliente una maggiore quantità di informazioni, per cui diminuisce l'apprensione del cliente nell'effettuare l'acquisto. Va evidenziato il fatto che le immagini stock, per quanto siano solitamente immagini grandi e di buona qualità, risultano avere una efficacia dimezzata rispetto a fotografie che ritraggono il prodotto venduto, anche questo può essere spiegato con il fatto che una foto stock permette all'utente di avere un'idea di come sia fatto il prodotto, ma non dà garanzie sullo stato dell'esemplare che si sta acquistando. Lo studio fa emergere come le immagini spingano il cliente a "seguire" il prodotto che si vuole acquistare, il che è indice di interesse da parte del cliente, tuttavia solo una parte molto ridotta di chi si interessa in questo modo ad un prodotto poi procede nell'acquisto, il tasso di conversione è nell'intorno di 1/3. Lo studio ha inoltre evidenziato come l'influenza delle immagini sia più forte nel caso di alcune categorie di prodotti, come quelli di moda, piuttosto che altri (es.: prodotti di elettronica).

A Multiple Classifier System for Predicting Best-Selling Amazon Products

L'ultimo lavoro citato è la tesi prodotta nel 2018 da Michael Kranzlein [68]. In questa tesi, Kranzlein utilizza una parte dei dati provenienti dal dataset *Amazon Review Data* per costruire un multiclassificatore in grado i riconoscere i "Best Selling Products", ovvero i prodotti che hanno un elevato successo di

vendita, dagli altri prodotti. Kranzlein seleziona dati appartenenti a 3 categorie: *Books*, *Home and Kitchen*, *Sports and Outdoors*; dopo di che li suddivide in 3 gruppi:

- BSP (Best Selling Products) = prodotti che ottengono un ottimo successo nella vendita
- NBSP (Not Best Selling Products) = prodotti che non sono BSP, sono in parte WSP e in parte prodotti che ottengono risultati intermedi
- WSP (Worst Selling Products) = prodotti che ottengono i peggiori risultati di vendita

Kranzlein sviluppa un multiclassificatore basato su 5 classificatori che prendono un input testuale (la descrizione): Naïve Bayes, Random Forest, Ridge Forest, Ridge Regression, SVM e un classificatore basato su una CNN che prende come input le immagini associate ai prodotti.
Kranzlein crea due dataset a partire dai 3 gruppi di dati: $BSP \cup NBSP$ e $BSP \cup WSP$, dopo di che ha utilizzato il multiclassificatore per effettuare classificazione binaria su entrambi i dataset cercando per ogni prodotto di predire l'appartenenza o meno alla classe BSP. Viene indicato che i risultati ottenuti su $BSP \cup NBSP$ risultano leggermente peggiori di quelli ottenuti su $BSP \cup WSP$ dal momento che nel primo caso le due famiglie di prodotti sono meno diverse e quindi più difficilmente distinguibili. Kranzlein evidenzia inoltre come l'unico classificatore che prende in input immagini produca risultati più scarsi di tutti i classificatori con input testuale. Tra i tratti che distinguono questo lavoro dai precedenti vi è l'elaborazione delle immagini. Va tuttavia considerato il fatto che un multiclassificatore di questo tipo non è in grado di cogliere legami tra dati contenuti in diversi tipi di input (testo e immagini), la quantità di informazioni che è in grado di estrarre è quindi inferiore a quella che può essere appresa da un modello Vision-Language Transformer.

Il lavoro da me svolto differisce da tutti quelli presentati per il fatto che prende in considerazione sia informazioni testuali che visive, utilizza inoltre un modello basato su Vision-Language Transformer in grado di catturare correlazioni tra le informazioni contenute in testo e immagine usati come input. I dati su cui viene applicata la soluzione sviluppata appartengono al settore della Moda. L'appartenenza dei prodotti a questa categoria rende ancor più rilevante tenere in considerazione l'immagine. L'importanza delle informazioni visive varia in base alla categoria del prodotto che l'utente sta consultando: nel caso di un indumento l'immagine può essere determinante nella decisione di acquisto, l'aspetto esteriore è sicuramente una caratteristica meno rilevante in un libro (categoria di prodotti presa in considerazione da Kranzlein).

Capitolo 2

Dataset

In questo capitolo verrà descritto il processo di creazione del dataset su cui sono stati utilizzati i modelli prodotti. Una delle prime attività svolte per lo sviluppo della tesi è stata una ricerca di un dataset idoneo ad addestrare i modelli che si volevano sviluppare. Di seguito le caratteristiche ricercate nel dataset:

- il dataset deve contenere informazioni relative a prodotti di Moda
- per ogni prodotto il dataset deve contenere dati testuali che riportano le informazioni conosciute dal venditore prima della commercializzazione del prodotto
- per ogni prodotto il dataset deve contenere almeno una immagine del prodotto condivisa dal venditore
- per ogni prodotto devono essere note informazioni che permettano di determinare il successo di vendita: ad esempio il numero di esemplari del prodotto venduti e livello di gradimento espresso da chi ha acquistato il prodotto

Purtroppo non sono stati trovati dataset con queste caratteristiche pronti per essere utilizzati. Tuttavia è stata individuata una raccolta di dati Amazon, contenente anche prodotti di ambito moda, a partire dalla quale è stato possibile creare un dataset in grado di soddisfare le mie esigenze.

2.1 Amazon Review Data

I dati utilizzati sono stati resi disponibili da Jianmo Ni et al.[69] in una raccolta, chiamata *Amazon Review Data*, che comprende informazioni relative a prodotti venduti su Amazon e le recensioni da questi ricevute; particolarità del

dataset, che lo rende adatto al progetto, è la presenza di informazioni relative al successo riscosso dai vari prodotti nella vendita. I dati raccolti da Jianmo Ni e dai collaboratori sono suddivisi in più slice che corrispondono a delle macro-categorie a cui i prodotti appartengono, tra queste abbiamo: "Books", "Electronics", "Clothing, Shoes and Jewelry", "Sports and Outdoors", "Cell Phones and Accessories", "Video Games", "Digital Music", ... Tra tutte queste categorie ho selezionato "Clothing, Shoes and Jewelry", che come suggerisce il nome contiene dati relativi a vestiti, scarpe e accessori di vario tipo (orologi, collane, braccialetti, ...); appartengono a questo slice dati su 2,685,059 prodotti e 32,292,099 recensioni relative ad essi. I dati contenuti in questo dataset sono stati raccolti a più riprese tra Maggio del 1996 e Ottobre del 2018. Questo dato fa già emergere alcuni problemi relativi al dataset:

- come è evidente i dati non sono stati raccolti di recente, ne risulta che lavoriamo su dei dati che potrebbero non rispecchiare totalmente quello che è lo stato attuale delle cose

- man mano che passano gli anni gli e-commerce hanno aumentato sempre più la quantità di informazioni relative ai prodotti messe a disposizione del pubblico. Dal momento che tali dati risalgono nel caso migliore a 3 anni fa e in quello peggiore a olte 25 anni fa, alcuni metadati recenti potenzialmente utili non saranno disponibili

- per quanto riguarda il successo di vendita riscosso da un prodotto, questo può essere ricavato a partire da un campo che riporta la posizione del prodotto in una classifica che vede al primo posto il prodotto piu venduto e all'ultimo posto quello meno venduto. Dal momento che il dataset contiene informazioni raccolte in istanti temporali diversi potremmo scoprire che la stessa posizione in classifica è occupata da più prodotti. Inoltre la posizione di 2 prodotti qualsiasi in classifica non fa necessariamente riferimento allo stesso istante temporale. Si è deciso di ignorare tale aspetto dal momento che non è un problema risolvibile e che non sono stati trovati altri dati utilizzabili. Per quanto riguarda la gestione della posizione in classifica nel calcolo della classe verrà spiegato in seguito come viene trattata questa informazione

Oltre alle criticità evidenziate, una prima analisi del dataset ha evidenziato il fatto che:

- non per tutti i prodotti sono presenti tutti i campi indicati da Jianmo Ni et al., e talvolta quando anche il campo è presente l'informazione può essere mancante o non consistente con quello che dovrebbe essere il contenuto del campo

- quando l'informazione è presente e consistente, può capitare che venga riportata in forme diverse: per esempio la descrizione di un prodotto potrebbe essere un oggetto stringa o un vettore di stringhe
- talvolta alcuni prodotti sono ripetuti, solitamente quando questo accade presentano anche lo stesso identificativo quindi sono facilmente riconoscibili
- pochi prodotti se analizzati risultano non appartenere alle categorie riportate (non sono nè vestiti nè scarpe nè accessori) e per questo vanno eliminati

viste tutte queste problematiche del dataset di partenza, non è stato possibile utilizzarlo senza aver prima effettuato una fase preparatoria di analisi, pulizia, normalizzazione e selezione dei prodotti in esso contenuti. Questa è stata condotta con un metodo iterativo dal momento che le problematiche legate ad ogni campo non erano note a priori, il dataset è quindi andato incontro a ripetuti cicli di normalizzazione del contenuto dei campi e selezione di prodotti adatti ad essere utilizzati nel progetto. Nelle prossime sezioni riporterò le operazioni effettuate sui campi di interesse.

Prima di passare alle prossime sezioni è necessario evidenziare che mentre i dati relativi ai prodotti sono conservati in un file chiamato *metadata*, quelli relativi alle recensioni sono conservati in un file chiamato *reviews*. Ogni prodotto ha un proprio id univoco chiamato "asin"; ogni recensione ha, tra le altre informazioni, un campo "asin" contenente il codice identificativo del prodotto a cui si riferisce. É stato quindi necessario determinare per ogni prodotto l'insieme di recensioni relativo e, a partire da queste, estrarre le informazioni relative al gradimento espresso da chi ha comprato l'articolo in questione.

2.2 Normalizzazione dei campi

Riporto in questo paragrafo le informazioni relative ai campi che descrivono i prodotti nel dataset *Amazon Review Data* (si fa quindi riferimento al file *metadata*). Per ogni campo riporto il significato dell'informazione contenuta, indico se tale informazione venga conservata o meno nel dataset finale, per le informazioni conservate descrivo eventuali operazioni di normalizzazione a cui sono state sottoposte:

- **asin**: è un campo stringa contenente un id univoco per il prodotto. In rari casi i prodotti non possiedono questo campo o hanno una stringa vuota o un valore None; in tutti questi casi il prodotto è stato scartato.

In casi rari capita che 2 prodotti distinti abbiamo lo stesso identificativo: in questi casi i prodotti differiscono per tutti i campi tranne asin, quindi è possibile riconoscerli ed eliminare entrambi i prodotti: non sarebbe possibile avere la certezza di quali recensioni si riferiscono all'uno o all'altro articolo. L'identificatore serve anche a ricollegare un prodotto alla relative recensioni, senza avere un valore univoco per ASIN non è possibile determinare il gradimento espresso dai clienti che hanno acquistato il prodotto

- **title**: è solitamente un campo stringa che contiene il titolo del prodotto, ovvero il nome assegnato dal venditore al prodotto. Talvolta il campo stringa è contenuto all'interno del primo indice di una list. Il campo title può raramente presentarsi vuoto o estremamente lungo. Effettuando controlli empirici su titoli con una lunghezza superiore a 300 caratteri si è notato che spesso in questi casi anomali è presente del codice o l'intero testo della descrizione all'interno di questo campo; per questo motivo si è deciso di sostituire il contenuto di tale campo con una stringa vuota nel caso in cui la stringa superasse i 300 caratteri. Si tratta comunque di eccezioni. Le informazioni contenute in tale campo sono state normalizzate e nel dataset finale il campo è sempre una stringa

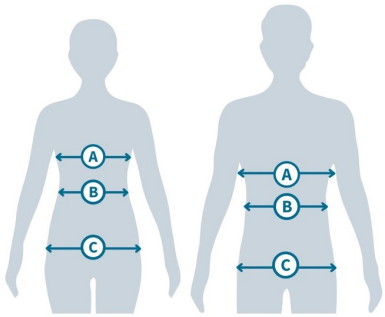
- **description**: questo campo contiene la descrizione testuale del prodotto. Le informazioni sono riportate all'interno di una stringa che talvolta è contenuta all'interno di una list, altre volte dentro una list di list, altre volte ancora è semplicamente una stringa. Le descrizioni possono essere corte (anche meno di una decina di parole) o molto lunghe, la maggior parte ha una lunghezza inferiore al centinaio di parole. Talvolta il contenuto di questo campo è None. Il campo è stato normalizzato, nel dataset finale la descrizione è sempre un campo stringa non vuota. I prodottila cui descrizione è un valore None o una stringa vuota vengono omessi

- **price** questo campo riporta il costo dell'articolo nella forma di una stringa. L'ammontare viene sempre espresso in dollari quando presente. Il campo non è sempre esistente e non di rado assume valori None o di stringa vuota. Quando il contenuto è significativo può essere espresso in due forme: come un range di prezzi "$prezzo_minimo - $prezzo_massimo" o come un prezzo esatto "$prezzo_esatto". Il valore numerico utilizza il punto come separatore tra le unità e la parte decimale, la virgola come separatore per le migliaia. Nel dataset finale tutti i prodotti hanno un campo *price*, a quelli il cui il prezzo è sconosciuto viene assegnato il valore "unknown". La decisione di lasciare il prezzo nella forma di stringa deriva

dal modo in cui si è deciso di darlo in pasto al modello: i modelli basati su transformer solitamente prendono in pasto dati testuali, al più immagini. Come verrà spiegato più nel dettaglio in seguito, sono stati addestrati più modelli che hanno elaborato l'informazione relativa al modello in maniera diversa: ad alcuni modelli il prezzo è stato dato in input nella forma di stringa, ad altri nella forma di una coppia di valori float. Nel secondo caso il software estrae l'informazione relativa al prodotto nella forma di stringa, dopo di che elabora l'informazione utilizzando delle funzioni RegEx; tale trasformazione tuttavia non rientra nella fase di preparazione del dataset

- **imageURLHighRes**: questo campo può contenere uno o più url (stringa) ciascuno relativo ad una immagine associata al prodotto. Il campo non è sempre presente; quando presente ed esiste solo un'immagine, il campo può essere semplicemente una stringa contenente l'url; quando abbiamo più immagini, il campo è solitamente nella forma di una lista di url o una lista di liste al cui interno sono contenuti gli url nella forma di stringhe. E' anche possibile trovare una lista vuota. Tra i campi scartati ne esiste uno molto simile a questo: *imageURL*; la differenza sta nella grandezza dell'immagine: nel caso di *imageURL* altezza e larghezza ammontano a qualche decina di pixel (solitamente meno di un centinaio), le immagini i cui url sono contenuti in *imageURLHighRes* hanno solitamente almeno una delle due dimensioni che supera i 200 pixel e la maggior parte di tali immagini variano tra 200x200 e 500x500, la dimensione è solitamente rettangolare. Non tutte le immagini i cui url vengono riportati sono ancora disponibili online, verificare la disponibilità dell'immagine e le dimensioni è stata parte integrante del processo di pulizia. Da un confronto tra tutti gli url contenuti nel dataset è emerso che talvolta prodotti diversi contengono lo stesso url. Da uno studio empirico su questi casi è emerso che nei casi in cui i prodotti coinvolti sono diversi (identificativi diversi), spesso le copie degli url erano contenuti in prodotti tali per cui uno è la variante dell'altro: è questo il caso di un venditore che vende più scarpe dello stesso modello ma con colore diverso e che pubblica un annuncio per ogni scarpa di diverso colore. L'immagine relativa alla suola della scarpa può essere riutilizzata in quanto la suola è effettivamente identica. Un altro caso in cui questo fenomeno si può verificare è quello in cui l'immagine non sia in realtà una foto del prodotto, ma contenga una tabella che illustra alcune informazioni, ad esempio come calcolare la taglia per un abito a partire da una misurazione fatta con metro da sarta. Per vestiti appartenenti alla stessa linea o allo stesso brand varrà lo stesso metodo di misurazione e probabilmente il venditore riporterà

l'immagine in più prodotti.



Taglie donna		XS 40	S 42	M 44	L 46	XL 48	XXL 50
A	CIRCONFERENZA DEL SENO	80/84	88/92	96/100	104/108	112/116	120/124
B	CIRCONFERENZA DELLA VITA	60/66	68/72	76/80	84/88	92/96	100/104
C	CIRCONFERENZA DEL BACINO	88/90	94/98	102/106	110/114	118/122	126/130

Taglie uomo		XS 40	S 42	M 44	L 46	XL 48	XXL 50
A	CIRCONFERENZA DEL TORACE	76/80	84/88	92/96	100/104	108/112	116/120
B	CIRCONFERENZA DELLA VITA	66/70	74/78	82/86	90/94	100/106	112/118
C	CIRCONFERENZA DEL BACINO	80/84	88/92	96/100	104/108	112/116	120/124

Figura 2.1: Esempio di una immagine il cui url potrebbe essere associato a più prodotti

In tutti questi casi si è proceduto eliminando l'url e anche il prodotto nel caso quella in questione fosse l'unica immagine disponibile per il prodotto. La decisione di tale rimozione è motivata dal fatto che tali immagini non sono significative per riconoscere il prodotto e quindi predirne il successo in termini di vendite. Alla fine della fase di normalizzazione si ottiene una lista contenente i diversi url e per ognuno di essi la grandezza delle due dimensioni. In fase di creazione del dataset finale viene scelta una sola immagine tra quelle disponibili, viene scaricata e ridimensionata affinchè le dimensioni diventino 512x512. L'immagine viene quindi salvata all'interno del campo *img* nel dataset finale nella forma di un vettore multidimensionale; nel dataset finale, all'interno del campo *url*, viene anche conservato l'indirizzo da cui è stata scaricata l'immagine

- **salesRank**: questo campo contiene informazioni relative alla categoria assegnata al prodotto e al posizionamento del prodotto in una classifica in base alla categoria di appartenenza. Nella classifica i primi posti sono riservati agli articoli più venduti e l'ultimo all'articolo meno venduto. I valori contenuti in questo campo sono di fondamentale importanza per determinare la classe del prodotto e la famiglia a cui appartiene. A partire da questo campo vengono determinati i seguenti campi del dataset finale: *category_1*, *category_2*, *category_3*, *category_4*, *position_1*, *index_inCategoryList_orderedBy_position_1*, *salesRank_coefficient*. I primi quattro campi riportati fanno riferimento alla categoria: ciascuno indica la categoria di appartenenza del prodotto ad un certo livello di astrazione. Le categorie sono infatti organizzate gerarchicamente in una struttura ad albero e la categoria di primo livello (*category_1*) rappresenta la radice dell'albero. Tutti i prodotti che prendiamo in considerazione

hanno la stessa categoria di primo livello, in quanto appartenenti al settore della moda; il nome di questa categoria è "Clothing, Shoes & Jewelry". Un grafico navigabile della gerarchia delle categorie dei prodotti Amazon può essere consultato al seguente indirizzo: `https://www.browsenodes.com/`. La gerarchia riportata sul sito BrowseNodes è parziale: vengono indicati solo i livelli più alti dell'albero, che dovrebbero essere condivisi da tutti i prodotti messi in vendita su Amazon. Per un venditore è infatti importante indicare correttamente le categorie di più alto livello perchè questo favorisce la possibilità che il prodotto venga mostrato ai clienti che effettuano ricerche relative ad articoli della stessa categoria del prodotto. Il venditore può tuttavia personalizzare ed estendere la gerarchia. Per quanto riguarda il progetto, ho preso in considerazione solo i primi 4 livelli della gerarchia, noti per la maggior parte dei prodotti contenuti nel dataset originale. Questi primi quattro livelli suddividono i prodotti in base ad alcune caratteristiche come: tipo di cliente (bambina/o, donna o uomo), tipo di prodotto (orologi, magliette, pantaloni, cappelli, scarpe, ...), contesto in cui vengono usati (indumenti per fare sport, da lavoro, ...). Per quanto riguarda i campi *position_1*, *index_inCategoryList_orderedBy_position_1*, *salesRank_coefficient*, questi vengono generati a partire dalla posizione del prodotto relativa alla categoria di primo livello (*Clothing, Shoes & Jewelry*). Tale posizione è riportata nel campo *position_1*. A partire dai valori di *position_1* dei prodotti appartenenti alla stessa categoria di quarto livello è possibile stabilire una classifica di prodotti appartenenti a quella categoria; la posizione di un prodotto in questa classifica è riportata in *index_inCategoryList_orderedBy_position_1*. Il valore della posizione di un prodotto in classifica sarà tanto più basso quanto migliore è stato il suo successo in termini di vendita: in altre parole la prima posizione, quella con indice 0, è assegnata al miglior prodotto. Quindi, data una categoria a cui appartengono N elementi, il valore di *index_inCategoryList_orderedBy_position_1* per questi prodotti sarà un intero appartenente all'intervallo $[0, N-1]$. A partire dalla posizione in questa classifica è poi possibile ricavare un valore, *salesRank_coefficient*, che indica quanto successo ha avuto il prodotto preso in considerazione paragonato agli altri articoli appartenenti alla stessa categoria di quarto livello. Il valore viene ricavato attraverso il seguente calcolo: $1 - (\frac{index\_inCategoryList\_orderedBy\_position\_1}{N})$.

É ora necessario giustificare due scelte: il numero di livelli di categorie da tenere in considerazione e l'utilizzo della posizione del prodotto nella graduatoria delle vendite amazon relativa alla categoria di livello 1 *Clothing, Shoes & Jewelry*.

Per quanto riguarda il numero di categorie tenuto in considerazione la scelta arbitraria di tenere in considerazione le prime 4 è stata fatta tenendo in considerazione i seguenti aspetti:

- per ogni prodotto abbiamo a disposizione un numero di livelli di categorie diverso, non tutte le categorie associate ai diversi prodotti raggiungono lo stesso livello di profondità, è stato quindi necessario individuare un livello di profonità tale che venisse raggiunto dalla maggior parte dei prodotti
- man mano che il numero di livelli aumenta la probabilità di trovare delle categorie custom aumenta drasticamente: questo è problematico perchè ogni venditore utilizza una propria tassonomia e nomenclatura; questo porta ad un'esplosione del numero di classi e alla frammentazione dell'insieme di prodotti in tanti piccoli sottoinsiemi, talvolta simili e poco rilevanti
- non si vuole frammentare eccessivamente l'insieme di prodotti: le categorie di prodotti devono essere significative
- si vuole comunque scendere nel dettaglio per quanto possibile: giacche e t-shirt fanno sempre parte della famiglia dei vestiti (che li distingue da scarpe o accessori) ma hanno costi diversi e ci si attende una diversa frequenza di vendita, quindi vanno considerate in classi separate

tenendo in considerazione questi ed altri aspetti si è reputato che il quarto livello costituisse un livello di dettaglio adeguato.

Per quanto riguarda la scelta di utilizzare il salesRank relativo alla categoria *Clothing, Shoes & Jewelry* piuttosto che quello relativo alla categoria di quarto livello, è necessario tenere in considerazione il seguente aspetto: per ogni prodotto non sono noti i SalesRank relativi ad ogni livello della gerarchia di categorie; in quasi tutti i prodotti è noto quello relativo alla categoria di primo livello e a quella di livello massimo, qualche volta viene riportato il salesRank relativo a un livello intermedio. Dal momento che solitamente dei prodotti tenuti in considerazioni sono noti più di 4 livelli di categoria, la conoscenza del salesRank relativa alla categoria di quarto livello non è frequente. Tuttavia si può tenere conto di un aspetto: ogni qual volta abbiamo due prodotti A e B tali per cui salesRank di livello 1 di A è minore a livello 1 di B, questo vuol dire che A vende più di più di B, di conseguenza anche il salesRank di livello 4 di A sarà inferiore a salesRank di livello 4 di B. Quindi è possibile basare la classifica del successo di vendita su *position_ 1*.

Come è evidente, i dati contenuti in questo campo sono di estrema importanza per il progetto in quanto necessari a determinare il grado di successo di un prodotto. L'estrazione di tali informazioni è tuttavia complessa per via della variabilità di forme in cui i dati vengono riportati in questo campo. Per prima cosa è necessario tener conto della possibilità che tale campo non esista o abbia come contenuto None. Qualora il contenuto invece risulti significativo, esso può essere riportato seguendo 3 strutture diverse come indicato in seguito. In fase di normalizzazione occorre quindi: riconoscere in quale forma si presentano i dati e in base a questa effettuare le operazioni necessarie a ricondurre i dati in forma normale. A quel punto è possibile estrarre i dati utili a caratterizzare il prodotto nel dataset finale. Altro aspetto da tenere in considerazione è il fatto che per ogni prodotto possono essere riportate più gerarchie di categorie, vengono tenute in considerazione solo quelle che hanno come primo livello *Clothing, Shoes & Jewelry* e, se necessario compiere una scelta, si sceglie la gerarchia che scende ad un livello di profondità maggiore. Di seguito una spiegazione dei diversi modi in cui le informazioni possono essere riportate.

Tipologia 1

```
>#436,337 in Toys & Games (See Top100 in Toys & Games)
>#3,501 in Clothing, Shoes & Jewelry > Costumes &
    Accessories > Kids & Baby > Boys > Costumes
>#4,243 in Toys & Games > Preschool > Pre-Kindergarten Toys
    > Pretend Play > Role PLay & Dress Up
>#5,902 in Toys & Games > Dressing Up & Costumes > Costumes
```

Listato 2.1: Esempio contenuto del campo salesRank di tipologia 1

La prima tipologia è caratterizzata dal fatto che i dati sono memorizzati in una stringa. All'interno della stringa possono essere contenuti più salesRank, come mostra il listato 2.1. Ogni indicazione di salesRank è preceduta da questa sequenza di caratteri: ">#". Il primo elemento riporta sempre il salesRank per una gerarchia di livello 1, in questo caso Toys & Games; inoltre termina con un contenuto tra parentesi. Per tutti gli altri salesRank la struttura è ">#"; seguito dalla posizione in classifica; seguito da "in"; seguito da una gerarchia di categorie, in cui la gerarchia di ogni livello è separata da quello di livello precedente da un carattere '>'. Notare che questo particolare prodotto, conoscendo solo queste informazioni, dovrebbe essere rigettato in quanto non è presente alcuna

indicazione circa la posizione (salesRank) relativo alla classe di livello 1 *Clothing, Shoes & Jewelry*. Per effettuare tale valutazione è necessario ricondurre le informazioni nella forma normalizzata come mostrato nella prossima figura.

```
"structured_rank": [
  {
     "position": 436337, "category": ["Toys & Games"]
  },
  {
     "position": 3501,
     "category": ["Clothing, Shoes & Jewelry", "Costumes &
         Accessories", "Kids & Baby", "Baby", "Costumes"]
  },
  {
     "position": 4243,
     "category": ["Toys & Games", "Preschool",
         "Pre-Kindergarten Toys", "Pretend Play", "Role PLay
         & Dress Up"]
  },
  {
     "position": 436337,
     "category": ["Toys & Games", "Dressing Up & Costumes",
         "Costumes"]
  },
]
```

Listato 2.2: SalesRank di tipologia 1 in forma strutturata standard

Tipologia 2

```
734,888inClothing,ShoesJewelry(
```

Listato 2.3: Esempio contenuto del campo salesRank di tipologia 2

Come mostrato nel listato 2.3, la seconda tipologia prevede nuovamente che le informazioni vengano riportate nella forma di una stringa secondo questo schema: posizione; seguita da "in"; seguita da una stringa riconducibile a una delle categorie di primo livello mostrate nella gerarchia disponibile al sito `https://www.browsenodes.com/`; seguita da un carattere "(". Lavorando sulla stringa ed utilizzando funzioni di mapping è possibile ricondursi alla forma normalizzata vista in precedenza. In questo

caso potremmo ottenere solo la posizione relativa alla categoria di livello 1 se ci basassimo solamente sul contenuto di questo campo. Esiste tuttavia un altro campo che riporta una gerarchia a cui appartiene il prodotto: **categories**. La prossima immagine mostra la forma normalizzata a cui viene ricondotta l'informazione riportata secondo la tipologia 2.

```
"structured_rank": [
  {
     "position": 734888,
     "category": ["Clothing, Shoes & Jewelry"]
  }
]
```

Listato 2.4: SalesRank di tipologia 2 in forma strutturata standard

Tipologia 3

```
>#in Clothing, Shoes & Jewelry > Luggage & Travel Gear >
   Messenger Bags
```

Listato 2.5: Esempio contenuto del campo salesRank di tipologia 3

Come mostrato nel listato 2.5 nel caso della terza tipologia le informazioni sono contenute in una lista, la lista può riportare più posizioni e categorie secondo una struttura simile a quella vista nella tipologia 1 per tutti i rank successivi al primo (il primo rank aveva una sola categoria di livello 1 e presentava del contenuto tra parentesi che andava rimosso). Nella prossima figura viene mostrato come anche in questo caso ci si possa ricondurre alla forma normale vista per le precedenti 2 tipologie

```
"structured_rank": [
  {
     "position": ,
     "category": ["Clothing, Shoes & Jewelry", "Luggage &
        Travel Gear", "Messenger Bags"]
  }
]
```

Listato 2.6: SalesRank di tipologia 3 in forma strutturata standard

- **categories**: questo campo riporta una gerarchia di categorie a cui il prodotto appartiene nella forma di una lista di stringhe. Tale campo può essere assente, la lista potrebbe essere vuota o riportare una

gerarchia di categorie che non rispetta quelle indicate dal sito `https://www.browsenodes.com/`. Può essere utilizzato in combinazione con il campo **salesRank** per determinare il valore dei campi *category_1*, *category_2*, *category_3*, *category_4* nel dataset finale.

- **feature**: campo contenente informazioni relative al prodotto che su Amazon vengono riportate nella forma di elenco puntato. Nel dataset originale questo campo non è presente per tutti i prodeotti e quando presente è una lista di stringa, in cui ogni elemento contiene il valore di un punto dell'elenco. Nel dataset finale il contenuto di questo campo è sempre una stringa ottenuta concatenando le stringhe relative ai singoli punti dell'elenco. Nel caso dei prodotti per cui questa informazione è sconosciuta viene utilizzata una stringa vuota.

Di seguito riporto un elenco dei campi scartati, di tali campi non è stata preservata l'informazione nel dataset finale:

- **imageURL**: come in **imageURLHighRes**, in questo campo vengono riportati uno o più url relativi a immagini del prodotto. Come già accennato, le immagini quì riportate hanno una dimensione mediamente inferiore, per questo motivo si è preferito utilizzare **imageURLHighRes**
- **related**: questo campo riporta un elenco di asin che identificano prodotti correlati a quello in questione
- **brand**: riporta il nome della marca del prodotto
- **tech1**: contiene informazioni relative a dettagli tecnici del prodotto
- **tech2** come **tech1**

2.3 Integrazione informazioni relative alle recensioni

Nel paragrafo precedente sono state prese in considerazione tutte le informazioni provenienti dal file "**metadata**", ovvero i dati che descrivono il prodotto. In questo paragrafo descrivo quelle salvate nel file "**reviews**" che, come dice il nome, contiene le recensioni relative ai prodotti salvati in **metadata**. Per ogni recensione abbiamo le seguenti informazioni:

- **reviewerID**: identificativo per la recensione

- **asin**: identificativo del prodotto a cui la recensione fa riferimento, viene utilizzato per associare prodotti e recensioni
- **reviewerName**: nome della persona che ha lasciato la recensione
- **vote**: numero di voti ricevuti dalla recensione come "recensione utile"
- **style**: contiene un dictionary con alcuni metadati relativi al prodotto
- **reviewText**: testo della recensione
- **overall**: valutazione espressa da chi ha effettuato le recensioni, i valori sono sempre interi compresi nell'intervallo $[1, 5]$ ma vengono riportati in formato floating point. Corrisponde al numero di stelle assegnate al prodotto dal recensore
- **summary**: riassunto della review
- **reviewTime**: è una marca temporale che indica quando è stata effettuata la recensione
- **image**: immagini pubblicate dal recensore e associate alla recensione

Di tutti questi campi gli unici che vengono presi in considerazione sono **asin** e **overall**. Il primo viene utilizzato per individuare il prodotto; il secondo consente di determinare per ogni prodotto il numero di recensioni con numero di stelle uguale a 1/2/3/4/5: questi valori vengono poi conservati nei campi **num_n_star_review_received** con $n \in [1, 5]$, del dataset finale. In base al numero di stelle ricevuto è poi possibile determinare i valori memorizzati nei campi:

- **average_star_score** è una media pesata calcolata sul numero di recensioni: $\frac{\sum_{n=1}^{5} num\_n\_star\_review\_received}{n\_recensioni\_totali}$
- **index_inCategoryList_orderedBy_averageStarScore**: è l'indice della posizione del prodotto in una lista ordinata secondo un ordine crescente di **average_star_score**. La lista contiene solo elementi appartenenti alla stessa categoria di 4° livello. Più alta è la media di stelle ricevute da un prodotto tanto maggiore sarà il valore dell'indice. Questo indice è utilizzato per il calcolo di *starScore_coefficient*. Il valore è quindi sempre un numero intero che varia nell'intervallo $[0, (|C| - 1)]$ dove C è la categoria presa in considerazione e $|C|$ è il numero di prodotti associati a tale categoria

- **starScore_coefficient**: è un indicatore di soddisfazione dei clienti che hanno acquistato l'oggetto e come tale viene usato nel calcolo del successo riscosso da un prodotto, in combinazione con **salesRank_coefficient**. Viene calcolato in questo modo: $\frac{index\_inCategoryList\_orderedBy\_averageStarScore}{|C|}$ dove C è la categoria presa in considerazione e $|C|$ è il numero di prodotti associati a tale categoria

Come indicato nella descrizione del campo *index_inCategoryList_orderedBy_averageStarScore*, non viene utilizzato direttamente *average_star_score* per valutare quanto il pubblico abbia apprezzato un prodotto, ma questo valore deve essere confrontato con quello degli altri prodotti appartenenti alla stessa categoria. Questo accade perchè, come mostra la figura 2.2, diverse categorie presentano una diversa distribuzione di *average_star_score*, per cui un certo valore di *average_star_score* potrebbe essere ottimo per prodotti appartenenti ad una certa categoria e pessimo per prodotti appartenenti ad un altra.

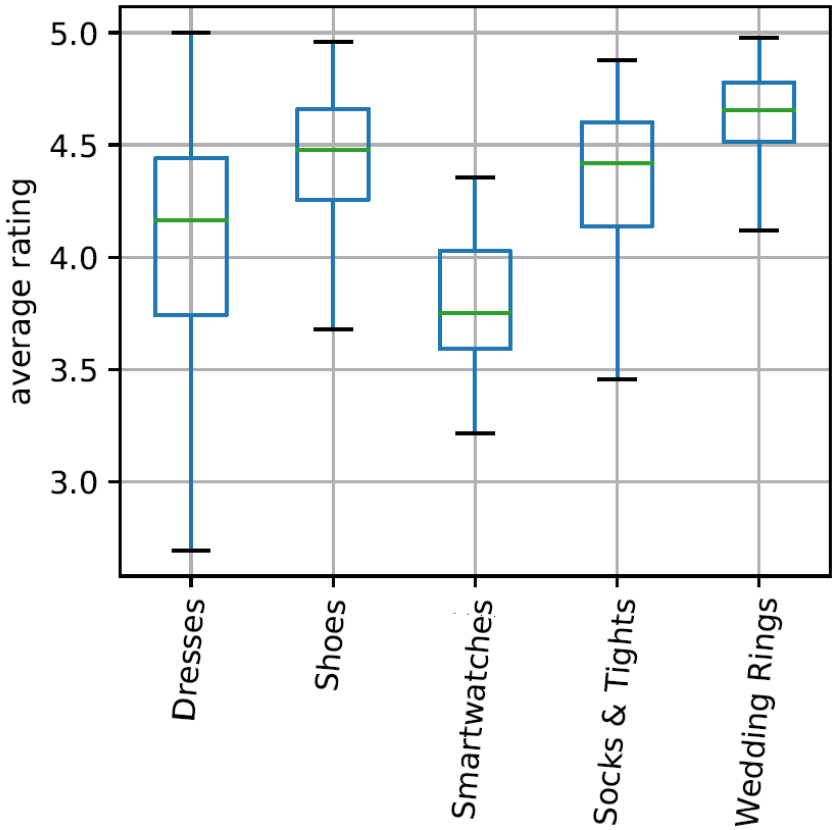


Figura 2.2: Box plot che riportata la media delle review ricevute per 5 categorie random tra quelle contenute nel dataset finale

Dall'analisi del dataset è emerso che non per tutti i prodotti esistono recensioni, per la maggior parte di prodotti inoltre abbiamo un numero di recensioni molto basso: su poco più di 2 milioni di prodotti, oltre 737 mila hanno meno di 5 recensioni; oltre 413 mila hanno tra 5 e 10 recensioni; oltre 293 mila hanno tra 10 e 15 recensioni e poco meno di 230 mila hanno tra 15 e 20 recensioni. Man mano che prendiamo in esame un numero di recensioni soglia superiore il numero di prodotti a cui sono associate una quantità di recensioni superiore alla soglia cala. Si è inoltre notato che tendenzialmente i prodotti con una media di stelle bassa hanno un numero limitato di recensioni, quelli con una media di gradimento elevata tendono ad avere parecchie recensioni. Questo può essere sensato: i prodotti di scarsa qualità potrebbero mediamente avere un successo di vendita inferiore e quindi essere recensiti da meno persone.

2.4 Determinazione della classe

La classe di appartenenza di un prodotto riflette il successo da esso ottenuto. Nel nostro caso abbiamo deciso di valutare il successo di un prodotto in base a due fattori: la quantità di esemplari venduti e la soddisfazione degli acquirenti. Questi due fattori sono rappresentati da due coefficienti numerici compresi nell'intervallo [0,1] che possono essere combinati tra loro per ottenere un nuovo coefficiente: il "success_score", il cui valore numerico è compreso nello stesso intervallo. Dal momento che "success_score" varia in [0,1] è facile partizionare questo intervallo in più sotto-intervalli e associare a ciascuno di essi una classe. Conoscendo il "success_score" di un prodotto e a quale intervallo numerico corrisponde ogni classe è poi possibile associare ad ogni prodotto una classe. Il fatto che la classe venga determinata a partire da un valore numerico apre anche alla possibilità di effettuare task di regressione per determinare questo valore a partire da altre informazioni relative al prodotto. Dal momento che il numero di classi in cui suddividere l'intervallo per il task di classificazione è relativamente arbitrario: i dati non impongono un numero di classi prestabilito, si è scelto di non riportare il valore dell'attributo classe all'interno del dataset finale. Nel dataset vengono riportati i due coefficienti numerici a partire dai quali è possibile calcolare l'attributo classe, in questo modo si delega questo compito all'applicazione che userà il dataset, garantendo una maggiore flessibilità di utilizzo dei dati. I due coefficienti numerici salvati nel dataset sono:

- **salesRank_coefficient**: è un indicatore del successo di vendita riscosso da un prodotto. Viene calcolato tenendo conto del numero di istanze vendute e della categoria di appartenenza del prodotto

- **starScore_coefficient**: è un indicatore della soddisfazione espressa da chi ha recensito il prodotto, anche in questo caso si tiene conto della categoria di appartenenza del prodotto

A partire da questi due valori è stato calcolato il valore dell'indicatore *success_score* utilizzando la formula:

$$success\_score = \alpha * salesRank\_coefficient + (1 - \alpha) * starScore\_coefficient$$

dove α è un iperparametro che può assumere valori nell'intervallo $[0, 1]$ dando maggiore importanza all'uno o all'altro coefficiente.

2.5 Selezione di prodotti

Durante la fase di analisi dei dataset e normalizzazione dei campi è stata condotta ciclicamente anche una attività di selezione dei prodotti. Sono stati eliminati tutti quei prodotti di cui non erano note le informazioni richieste dal task di classificazione: ovvero prodotti per cui erano mancanti o vuoti alcuni campi tra: *asin*, *description*, *imageURLHighRes* o *salesRank*. Sono stati rimossi anche tutti quei prodotti che non risultavano appartenere all'ambito della moda o di cui non era noto il posizionamento nella classifica stilata da amazon relativa ai prodotti di categoria *Clothing, Shoes & Jewelry*. Sono poi stati rimossi tutti i prodotti che sono risultati essere delle copie di altri prodotti o per cui non esistono almeno 10 recensioni. Sono stati eliminati anche tutti i prodotti con un numero di recensioni inferiore a 20 e una media di stelle ricevute superiore a 3.0: 20 è stato reputato un buon valore di soglia affinché potesse essere effettuata una media rilevante sul gradimento espresso dalle recensioni. Poiché il gradimento medio per prodotti che ricevono almeno 20 recensioni risulta molto elevato (quasi sempre superiore a 4 stelle), si è deciso di tenere in considerazione anche prodotti con un gradimento basso (al massimo una media di 3.0 stelle) ed un numero di recensioni compreso tra 10 e 20. In questo modo è stato possibile bilanciare il dataset. Per di più questa scelta non è del tutto incoerente in quanto 10 recensioni (soglia minima) possono essere sufficienti per dare una indicazione di gradimento, inoltre, il fatto stesso di avere poche recensioni può essere considerato un rafforzativo del basso livello di gradimento medio riscontrato da questo gruppo di prodotti.

2.6 Dataset Finale

Di seguito riporto brevemente le informazioni contenute nel dataset finale che è stato creato elaborando i dati contenuti in *Amazon Review Data*:

- **asin**: stringa utilizzata come identificativo univoco del prodotto
- **price**: stringa che riporta il prezzo a cui è venduto il prodotto; se il prezzo non è noto la stringa riporta il valore "unknown"; se il prezzo è noto può essere un prezzo esatto es.: "$15.99", oppure un range di prezzi "$10.99 - $15.99". I numeri utilizzati sono puramente esemplificativi ma la struttura della stringa è quella indicata e può essere elaborata attraverso delle funzioni regex dal software che elabora il dataset.
- **title**: è il nome del prodotto, può essere una stringa vuota
- **description**: è la descrizione del prodotto, è sempre una stringa non vuota
- **category_1**: stringa che indica la categoria di 1° livello a cui appartiene il prodotto, assume sempre il valore *Clothing, Shoes & Jewelry* per i prodotti selezionati
- **category_2**, **category_3**, **category_4**: stringhe che indicano la categoria di 2°/3°/4° livello a cui appartiene il prodotto
- **num_1_star_review_received**: numero di recensioni con valutazione 1 stella (intero)
- **num_2_star_review_received**: numero di recensioni con valutazione 2 stelle (intero)
- **num_3_star_review_received**: numero di recensioni con valutazione 3 stelle (intero)
- **num_4_star_review_received**: numero di recensioni con valutazione 4 stelle (intero)
- **num_5_star_review_received**: numero di recensioni con valutazione 5 stelle (intero)
- **position_1**: posizione del prodotto nella classifica amazon relativa alla categoria di primo livello (numero intero)
- **index_inCategoryList_orderedBy_position_1**: indice corrispondente alla posizione occupata dal prodotto in una lista ordinata per ordine crescente di *position_1* (numero intero)

- **salesRank_coefficient**: è un indicatore del successo di vendita riscosso da un prodotto. Viene calcolato tenendo conto del numero di istanze vendute e della categoria di appartenenza del prodotto (numero reale assume valori in [0.0,1.0])
- **average_star_score**: media di stelle ricevute (numero reale assume valori in [1.0,5.0])
- **index_inCategoryList_orderedBy_averageStarScore**: indice corrispondente alla posizione occupata dal prodotto in una lista ordinata per ordine crescente di *average_star_score*(numero intero)
- **starScore_coefficient**: è un indicatore della soddisfazione espressa da chi ha recensito il prodotto, anche in questo caso si tiene conto della categoria di appartenenza del prodotto (numero reale assume valori in [0.0,1.0])
- **url**: url (stringa) dell'immagine relativa al prodotto
- **img**: vettore multidimensionale (512x512x3) che contiene i valori dei pixel che compongono l'immagine
- **feature**: una stringa ottenuta concatenando le frasi contenute nell'omonimo elenco puntato del dataset originale

Di seguito riporto un esempio che mostra come vengono riportate le informazioni relative ad un oggetto contenuto nel dataset; il contenuto dei campi "description" e "feature" è stato troncato per ragioni di spazio.


```
{
  "asin": "6040972467",
  "price": "$22.99",
  "title": "Pistachio Women's Sun Flower Flowing Knee Length Summer
     Dress",
  "description": "GorgeoUS lightweight cotton dress in red, pink or
     purple. Made from 100% cotton, ...",
  "category_1": "Clothing, Shoes & Jewelry",
  "category_2": "Women",
  "category_3": "Clothing",
  "category_4": "Dresses",
  "url": "https://images-na.ssl-images-amazon.com
     /images/I/61SkcLp5T6L.jpg",
  "num_1_star_review_received": 4,
  "num_2_star_review_received": 1,
  "num_3_star_review_received": 7,
```

```
  "num_4_star_review_received": 8,
  "num_5_star_review_received": 22,
  "position_1": 1131061,
  "index_inCategoryList_orderedBy_position_1": 1754,
  "index_inCategoryList_orderedBy_averageStarScore": 2212,
  "salesRank_coefficient": 0.5218102335929871,
  "average_star_score": 4.023809432983398,
  "starScore_coefficient": 0.6030534505844116,
  "feature": "100% Cotton Women's Floral Pistachio Dress. Beautiful
     Floral And Spot Design To The Fabric ..."
}
```


I dati sono stati salvati utilizzando il formato **HDF5** (Hierarchical Data Format) che consente una memorizzazione efficiente di file di grandi dimensioni. Ogni campo viene conservato in un "dataset" all'interno del file HDF5, i dati relativi allo stesso prodotto vengono conservati all'interno dei diversi dataset allo stesso indice, questo rende facile l'accesso indicizzato a tutte le informazioni relative allo stesso articolo. Il dataset viene interrogato utilizzando la libreria **h5py** [70].

2.7 Dataset Prodotti

Finora si è parlato di dataset finale, lasciando intendere di aver prodotto un'unica collection di dati a partire da quella iniziale. In realtà sono state prodotte più versioni del dataset finale, tutte contenenti lo stesso insieme di campi per ogni prodotto; ogni campo ha le caratteristiche descritte nei paragrafi precedenti. Le diverse versioni di dataset finale hanno alcune caratteristiche in comune: sono suddivise in 3 slice chiamate rispettivamente *train*, *validation*, *test*; all'interno di ogni slice i prodotti sono divisi equamente tra le 3 classi: *Bad*, *Medium*, *Good*. Nell'ambito del progetto, salvo diversa indicazione, quando faremo riferimento a queste 3 classi si assumerà che la label da attribuire ad uno specifico prodotto venga calcolata in questo modo: per prima cosa viene ricavato lo score utilizzando la seguente espressione con il parametro $\alpha = 0.7$:

$$success = \alpha * salesRank\_coefficient + (1 - \alpha) * starScore\_coefficient$$

dopo di che la classe viene assegnata al prodotto applicando la seguente funzione definita a tratti:

$$c(p) = \begin{cases} bad & \text{if } s(p) < \frac{1}{3} \\ medium & \text{if } \frac{1}{3} \leq s(p) < \frac{2}{3} \\ good & \text{if } s(p) \geq \frac{2}{3} \end{cases}$$

Di seguito riporto nomi e caratteristiche delle diverse versioni del dataset finale.

AmazonProducts_base:

- è la prima versione realizzata di dataset finale; l'immagine associata ad ogni prodotto è stata scelta casualmente a partire da quelle disponibili
- nello slice *train* sono contenuti 52200 prodotti (17400 per ogni classe); nello slice *validation* sono contenuti 17400 prodotti (5800 per ogni classe); nello slice *test* sono contenuti 8700 prodotti (2900 per ogni classe)
- per quanto riguarda la distribuzione di prodotti con prezzo tra le diverse classi all'interno delle slice abbiamo:
  - in *train* l'informazione relativa al prezzo è nota per: il 29.10% dei prodotti appartenenti alla classe *bad*, il 55.22% dei prodotti appartenenti alla classe *medium*, l'88.93% dei prodotti appartenenti alla classe *good*
  - in *validation* l'informazione relativa al prezzo è nota per: il 28.55% dei prodotti appartenenti alla classe *bad*, il 57.12% dei prodotti appartenenti alla classe *medium*, l'89.26% dei prodotti appartenenti alla classe *good*
  - in *test* l'informazione relativa al prezzo è nota per: il 33.21% dei prodotti appartenenti alla classe *bad*, il 60.03% dei prodotti appartenenti alla classe *medium*, l'88.48% dei prodotti appartenenti alla classe *good*

AmazonProducts_first_image:

- dal momento che l'ordine degli url relativi alle immagini di un prodotto nel dataset di origine riflette presumibilmente l'ordine con cui le immagini vengono presentate sul sito di Amazon, si è deciso per ogni prodotto di prendere in considerazione la prima immagine disponibile per creare una nuova versione del dataset. L'idea nasce dall'ipotesi che le prime

immagini mostrate del prodotto siano più rilevanti in quanto servono a catturare l'attenzione del potenziale cliente. Queste immagini dovrebbero rappresentare meglio il prodotto: capita spesso che la prima immagine di un prodotto riporti una visione chiara dell'intero oggetto, mentre le immagini successive si concentrino su dettagli. Prendendo l'esempio di una scarpa, la prima fotografia solitamente mostra la parte superiore della scarpa vista frontalmente, mentre le immagini successive si concentrano su: la suola, i laccetti, le cuciture, ecc. Si è quindi pensato che potesse essere sensato valutare se il modello fosse in grado di trarre giovamento da una immagine così selezionata

- Come in ***AmazonProducts_base***, 52200 prodotti appartengono a *train*, 17400 a *validation* e 8700 a *test*; in tutti e 3 gli slice i prodotti sono distribuiti equamente tra le 3 classi *Bad*, *Medium*, *Good*

- La distribuzione di prodotti per cui è nota l'informazione relativa al prezzo tra le 3 classi all'interno dei 3 split è uguale a quella riportata per ***AmazonProducts_base***

AmazonProducts_fi_price_known:

- questa versione del dataset finale è molto ridotta rispetto alle due appena viste in termini di numero di prodotti contenuti negli slice. Questo dataset viene infatti creato a partire dal dataset ***AmazonProducts_first_image*** (per questo motivo nel nome compare "fi") e contiene solo prodotti il cui prezzo è conosciuto. Dato uno dei 3 slice di ***AmazonProducts_fi_price_known***, questo non contiene tutti i prodotti dal prezzo noto contenuti nel corrispondente slice di ***AmazonProducts_first_image*** perchè in ogni slice si è mantenuto il bilanciamento del numero di prodotti rispetto alle 3 classi

- Per quanto riguarda il numero di prodotti all'interno dei vari slice: in *train* sono contenuti 15150 prodotti (5050 per ogni classe); in *validation* sono contenuti 4920 prodotti (1640 per ogni classe); in *test* sono contenuti 2880 prodotti (960 per ogni classe)

AmazonProducts_fi_Bad_Good_Only:

- questa è una versione ridotta del dataset ***AmazonProducts_first_image*** che contiene per ogni slice solo quei prodotti che appartengono alla classe Bad o alla classe Good. Questa versione del dataset è stata realizzata per poter effettuare dei test di classificazione binaria, in modo da valutare la capacità del modello di distinguere i prodotti che raggiungono livelli massimi e minimi di successo di vendita

- nello slice *train* sono contenuti 34800 prodotti (17400 per classe); nello slice *validation* sono contenuti 11600 prodotti (5800 per ogni classe); nello slice *test* sono contenuti 5800 prodotti (2900 per ogni classe)

- per quanto riguarda la distribuzione di prodotti con prezzo tra le diverse classi all'interno degli slice, abbiamo una distribuzione uguale a quella rilevata in ***AmazonProducts_first_image*** e ***AmazonProducts_base*** per le classi *Bad* e *Good*

Capitolo 3

Sviluppo del modello

In questo capitolo descrivo le tecnologie utilizzate in fase di sviluppo, la struttura del modello e le configurazioni adottate per i principali modelli addestrati.

3.1 ViLT come punto di partenza

Come descritto precedentemente, si vuole realizzare un modello basato su Transformer in grado di prendere in input dati testuali ed immagini riferite ad un prodotto e stabilire il successo di vendita di un prodotto associandolo ad una classe. Nel caso base le classi sono tre: *Bad*, *Medium*, *Good*.

Una volta stabilito questo, è iniziata una fase di ricerca volta ad identificare le soluzioni esistenti in letteratura e valutare la possibilità di adottare un modello già esistente. Da tale ricerca è emerso ViLT, acronimo di Vision and Language Transformer, un modello VLP proposto da Kim et al. nel paper "ViLT: Vision-and-Language Transformer Without Convolution or Region Supervision" [31] pubblicato nell'anno 2021. Le caratteristiche di questo modello sono già state spiegate in maniera approfondita nella sezione dedicata, per questo riporterò di seguito solo le motivazioni che mi hanno spinto ad adottare tale modello.

Per prima cosa ViLT rispecchia il modello che si voleva implementare: è infatti un modello cross-modale (prende in input coppie di testo e immagine) ed è basato su trasformer.

ViLT è un modello interessante in quanto introduce delle novità rispetto ai modelli precedenti. É infatti il primo modello VLP ad elaborare le immagini sfruttando una strategia analoga a quella utilizzata per il testo: le immagini vengono suddivise in patch, dopo di che ad ogni patch viene applicata una proiezione lineare, secondo la tecnica di *patch projection* embedding già proposta dal modello ViT [7]. In questo modo viene completamente rigettata la convoluzione

ed il calcolo dei visual embedding diventa meno oneroso e paragonabile a quello dei text embedding in termini di costo computazionale. Allo stesso tempo viene dato spazio a quella parte di modello (*modality interaction*) dedicata all'individuazione di schemi di interazione inter-modale, che viene implementata utilizzando un transformer e non un semplice prodotto scalare o un layer di attention poco profondo.

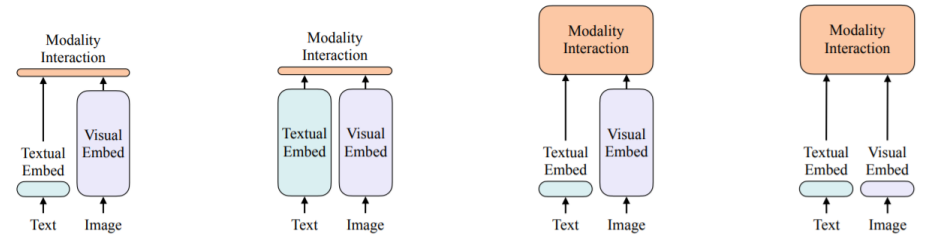


Figura 3.1: Quattro categorie di VLP models. L'altezza dei rettangoli denota il costo computazionale. VE sta per *Visual Embedder*, TE sta per *Textual Embedder*, MI sta per *Modality Interaction*

Nella figura 3.1 viene fatto un confronto in termini di costo computazionale di vari modelli VLP. ViLT è l'unico modello appartenente alla categoria (d) e, a differenza degli altri modelli, richiede un costo computazionale notevolmente inferiore per il calcolo di visual embedding e presenta una complessità maggiore nel componente di *modality interaction*, questo consente una maggiore comprensione delle relazioni intermodali.

Il superamento della convoluzione in favore della tecnica di *patch projection* risulta in un aumento dell'*efficienza* e della *velocità di elaborazione* dell'immagine; a questo si aggiunge un incremento del *potere espressivo*, non più limitato da quello del modello utilizzato per l'estrazione delle feature dalle immagini. Per quanto riguarda il caso specifico del progetto sviluppato, l'utilizzo di patch projection porta anche ad ulteriori vantaggi. Spesso i modelli che utilizzano Fast R-CNN lavorano su immagini in cui sono presenti più oggetti/forme/animali e queste reti consentono al modello di riconoscere quali elementi siano contenuti nell'immagine. I modelli che vogliamo sviluppare devono però lavorare ad un livello di dettaglio maggiore: non è sufficiente che il modello sia in grado di capire che nella figura è ritratta una borsa. Le immagini su cui lavoreranno i modelli sviluppati spesso conterranno un solo oggetto, quello venduto, e il modello dovrebbe essere in grado di determinare i dettagli che caratterizzano quello specifico oggetto (colore, motivi decorativi, tipo di finiture, ...) per poterlo distinguere da altri appartenenti alla stessa categoria. La tecnica di patch projection consente ai modelli di raggiungere una comprensione a questi

livelli di dettaglio, come dimostrato nel paper "FashionBERT: Text and Image Matching with Adaptive Loss for Cross-modal Retrieval" [71].

Per quanto riguarda lo schema di *modality interaction*, volendo adottare la classificazione proposta da Buglianello [55], possiamo dire che ViLT adotta un approccio *single-stream*. Gli embedding risultanti dall'elaborazione di dati testuali ed immagini vengono concatenati e poi dati in pasto al componente *modality interaction* nella forma di un unica stringa. Un modello di questo tipo richiede meno parametri di un modello *dual-stream* in cui i diversi tipi di input vengono passati al modello separatamente.

Ne risulta un modello efficiente sia in termini di risorse computazionali che tempi di elaborazione richiesti e complessivamente poco complesso, che sembra essere un ottimo punto di partenza per il progetto.

3.2 Tecnologie Utilizzate

La scelta delle tecnologie da impiegare è stata una naturale conseguenza dell'adozione di ViLT come punto di partenza per il progetto. ViLT è infatti stato implementato basandosi su *PyTorch Lightning*.

PyTorch

PyTorch [72] è una libreria open-source per il machine learning sviluppata da Facebook. É specializzata nel calcolo su tensori, differenziazione automatica e GPU acceleration. Per queste sue caratteristiche, per l'integrazione con il linguaggio python e per la facile estendibilità è una delle librerie per deep learning più popolari, sopratutto nella comunità di ricerca scientifica; ed è la principale concorrente di Keras e Tensorflow.

Volendo confrontare queste tre librerie si potrebbe dire che PyTorch mette a disposizione API di più basso livello rispetto a quelle offerte da Keras, consentendo così ai ricercatori la possibilità di una maggiore customizzazione nella produzione di nuovi modelli. Tensorflow mette a disposizione funzionalità analoghe a quelle fornite da PyTorch. Nel contesto di questo progetto è stato scelto PyTorch per compatibilità rispetto al codice esistente (ViLT).

PyTorch Lightning

PyTorch Lightning [73] è una versione più recente e migliorata di PyTorch. Tra gli aspetti chiave che caratterizzano PyTorch Lightning abbiamo:

- elevata scalabilità: possibilità di eseguire il software su hardware di vario tipo (CPU, GPU, TPU) senza la necessità di apportare modifiche al modello

- automatizzazione del training loop
- rimozione di codice boilerplate
- integrazione con framework di visualizzazione e logging come Tensorboard
- disponibilità di implementazioni pronte per l'uso delle metriche più comuni (accuracy, precision, recall, ...). Queste implementazioni ne facilitano l'utilizzo anche nel caso di calcolo distribuito su più GPU; si veda per questo il pacchetto TorchMetrics
- facile creazione di checkpoint: salvataggi automatici dello stato dell'ultima epoca di training per assicurare la possibilità di riprendere l'esecuzione in caso di interruzione
- EarlyStopping automatizzato: possibilità di automatizzare la terminazione anticipata di task di addestramento qualora si verifichino certe condizioni

Nel progetto è stato utilizzato *PyTorch Lightning* 1.4.5, *PyTorch1.9.0* e *Python* 3.9.7.

Tensorboard

Per quanto riguarda il monitoraggio dell'avanzamento degli addestramenti, si è fatto affidamento a Tensorboard, una suite di web application che consente di ispezionare lo stato di avanzamento di task di deep learning, mostrando come variano i valori delle metriche nel tempo. L'evoluzione di tali valori nel tempo viene mostrato anche attraverso piani cartesiani, è inoltre possibile confrontare i progressi relativi a più addestramenti.

3.3 Struttura del modello

Per quanto riguarda la struttura del modello rappresentato occorre tenere conto di un aspetto fondamentale. Sebbene fosse chiaro l'obiettivo finale: produrre un software in grado di distinguere prodotti che avranno un successo di vendita buono/medio/scarso, era impossibile sapere a priori quali dati avrebbero permesso al modello di raggiungere tale risultato. Per questo motivo sono stati sviluppati più modelli dall'architettura simile, ma che differiscono per quanto riguarda quei componenti che si occupano della ricezione e della preparazione dei dati in input. Tra le informazioni che sono state date in pasto alle varie implementazioni abbiamo: *titolo* e *descrizione* del prodotto, sono due input testuali; *prezzo*, utilizzato a volte come input testuale o altre

volte come input numerico; *immagine*, input visivo. Non tutti i modelli sono stati addestrati ricevendo in input tutte le informazioni, questo ha permesso di effettuare confronti e stabilire il contributo apportato dai dati di diversa natura. Alcuni modelli sono stati addestrati per effettuare task diversi da quello della classificazione: ad esempio un task di regressione, in questo caso le differenze riguardano anche la parte terminale dell'architettura (la testa) che varia a seconda del task da eseguire. In ogni caso l'architettura alla base di tutti i modelli è la stessa, salvo le variazioni indicate.

Nella seguente immagine riporto la rappresentazione dell'architettura utilizzata per addestrare un modello su un task di classificazione multiclass a 3 classi: Bad, Medium, Good; le classi fanno riferimento al successo di vendita e il modo in cui sono state calcolate è quello già spiegato in precedenza.

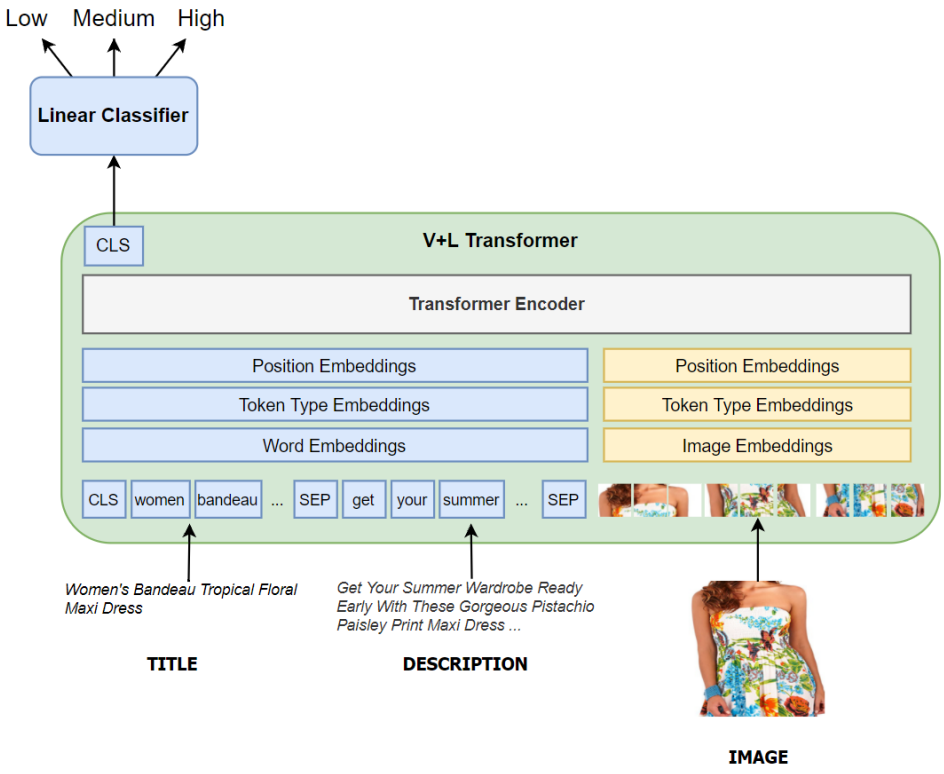


Figura 3.2: L'immagine riporta gli aspetti che caratterizzano l'implementazione di un modello in grado di effettuare classificazione multiclass prendendo come input testuali titolo e descrizione e come input di tipo visuale una immagine del prodotto

Come mostrato in figura 3.2 al modello vengono passati input testuali ed immagine. In un primo momento i 2 tipi di input vengono elaborati

separatamente: per quanto riguarda la componente testuale, l'input viene dato in pasto a un embedder di BERT ottenendo così i Word Embedding; l'immagine viene suddivisa in un numero prestabilito di patch, dopo di che a partire dalle patch vengono calcolati gli Image Embedding utilizzando una proiezione lineare. Position embedding, Token Type Embedding e Word/Image Embedding sono tutti tensori della stessa dimensione *hidden size*, 768 nel nostro caso. Token Type Embedding assumono due possibili valori: uno viene utilizzato per i token testuali, l'altro per quelli derivati dalla proiezione delle immagini; in questo modo il modello impara a riconoscere gli embedding dei due tipi di input. I Position Embedding vengono poi utilizzati per distinguere la posizione della parola o quella della patch da cui è stato derivato il token. Ogni Word/Image Embedding, viene quindi sommato al corrispettivo Token Type Embedding e Position Embedding. Una volta ottenuti gli embedding relativi a token testuali e immagine è possibile concatenarli tra loro ottenendo un'unica sequenza, che comprende anche alcuni embedding speciali: quelli corrispondenti al token [CLS] e [SEP]. L'embedding che corrisponde al token [CLS] viene anteposto all'intera sequenza. L'embedding corrispondente al token [SEP] viene utilizzato per separare diverse informazioni in input, viene posto ad esempio tra titolo-descrizione e tra descrizione-immagine. La sequenza di embedding viene quindi data in pasto al Transformer Encoder, che produce come output una sequenza delle stesse dimensioni. In testa alla sequenza di output è presente un tensore contenente le hidden feature associate al token [CLS]. Questo viene utilizzato come input per una piccola rete neurale in grado di eseguire il downstream task desiderato, in questo caso quello di classificazione. Nell'esempio questo componente, chiamato *testa*, in inglese *head*, è rappresentato dal *Linear Classifier*: un layer lineare che ha *hidden size* neuroni di input, ovvero un numero di valori pari al numero di hidden features relative al token [CLS], e 3 neuroni in output: 1 per ogni label a cui può essere associato il prodotto.

Nel caso di modelli che non contengono una parte di input, ad esempio sono privi di immagine (solo input testuale) o non prendono in input il titolo, il modello sarà privo di tutte quelle componenti necessarie ad elaborare quell'input. Nel caso manchino le immagini quindi, non saranno presenti i layer relativi a Image Embedding e tutta quella parte di modello ripresa da ViT, mentre nel caso sia il titolo a mancare, la componente testuale dell'input deriverà solamente dal testo contenuto nelle descrizioni. In quei casi in cui il prezzo è stato utilizzato in forma testuale, il valore del prezzo è stato anteposto al resto dell'input testuale ed è andato incontro alle stesse trasformazioni illustrate per titolo e descrizione.

Il Transformer Encoder è il cuore del modello e si occupa di comprendere

le relazioni inter/intra-modali tra gli embedding ottenuti dai diversi tipi di input. L'architettura utilizzata è la stessa descritta per ViLT, si rimanda quindi al capitolo dedicato, per una descrizione dettagliata. Riassumendo in breve quanto già spiegato in quel capitolo, il trasformer adottato da ViLT riprende l'architettura ed i pesi di ViT pre-trained. ViT è formato da un insieme di "block" impilati l'uno sull'altro che includono layer MSA (multiheaded self-attention) e layer MLP. Voldendolo esprimere con un formalismo matematico abbiamo:

$$\overline{t} = [t_{class}; t_1 T; ...; t_L T] + T^{pos} \tag{3.1}$$

$$\overline{v} = [v_1 V; ...; v_L V] + V^{pos} \tag{3.2}$$

$$z^0 = [\overline{t} + t^{type}; \overline{v} + v^{type}] \tag{3.3}$$

$$\hat{z}^d = MSA(LN(z^{d-1})) + z^{d-1} \qquad d = 1...D \tag{3.4}$$

$$z^d = MLP(LN(\hat{z}^d)) + \hat{z}^d \qquad d = 1...D \tag{3.5}$$

$$p = tanh(z_0^D W_{pool}) \tag{3.6}$$

dove: T è la matriche dei Word Embedding; T^{pos} è la matrice dei Position Embedding relativa all'input testuale; V è la matrice degli Image Embedding e V^{pos} è la matrice dei Position Embedding relativa all'input visivo. I tensori t^{type} e v^{type} sono i Token Type Embedding rispettivamente per input testuale ed immagine. z è il contesto che viene aggiornato ogni volta che viene attraversato uno dei D layer che compongono il transformer (nel nostro caso D=12). p è una rappresentazione pooled dell'intero input e viene calcolata applicando una proiezione lineare W_{pool} e una tangente iperbolica sull'elemento corrispondente all'indice 0 del tensore z^D, ovvero l'elemento corrispondente al token [CLS].

Per quanto riguarda le head che eseguono i downstream task, ad ogni task, sia che venga utilizzato per il pre-training che per il fine-tuning, è associata una testa dedicata. Queste teste sono costituite da reti neurali più o meno profonde, anche questo aspetto è stato oggetto di studio. Le teste possono inoltre differire per numero di neuroni presi in input e restituiti in output. Solitamente, nei task implementati, il numero di neuroni di input in una testa è pari a *hidden size*: ovvero alla lunghezza del tensore contenente le hidden feature associate a [CLS]; tuttavia, nei casi in cui il prezzo è stato passato come un valore numerico, questo viene dato direttamente in pasto alla testa, quindi il numero di neuroni in input cresce. Per quanto riguarda il numero di neuroni di output, avremo tanti neuroni quante sono le label nel caso di classificazione multiclass single-label, mentre avremo un solo neurone in output nel caso di un task di regressione. Più in generale il numero di neuroni in output dipende dal task che si vuole eseguire.

3.4 Implementazione

Di seguito descrivo come è stato implementato il modello illustrato nel paragrafo precedente. La descrizione si concentrerà sulle componenti principali e sul modo in cui sono stati sviluppati i seguenti aspetti: caricamento e preparazione dati, implementazione del modello e downstream tasks, calcolo delle metriche.

3.4.1 Caricamento e preparazione dati

Il caricamento e la preparazione dati sono funzionalità messe a disposizione da due classi: *AmazonProductsDataset* e *AmazonProductsDataModule*.

AmazonProductsDataset

AmazonProductsDataset è una classe che si occupa di prelevare i dati relativi ai singoli prodotti da un file hdf5 e di metterli a disposizione del modello. Estende la classe *torch.utils.data.Dataset* messa a disposizione dalla libreria PyTorch. PyTorch mette a disposizione più tipi di dataset, AmazonProductsDataset implementa un tipo di dataset chiamato *map-style*. I *map-style dataset* rispettano un protocollo basato sull'implementazione delle funzioni `_len_()` e `_get_item_()`, e rappresentano una mappa che associa univocamente un indice ad un elemento (prodotto in vendita). La funzione `_len_()` restituisce il numero di prodotti contenuti nel dataset. La funzione `_get_item_()` si occupa di recuperare tutte le informazioni relative al prodotto associato ad un certo indice nei dataset del file h5 e preparare i dati in modo tale che possano essere utilizzati dal modello. Per quanto riguarda la preparazione dei dati abbiamo:

- preparazione della componente testuale dell'input: le varie informazioni testuali (titolo, descrizione) da passare al modello vengono concatenate aggiungendo gli opportuni separatori; viene effettuato l'encoding che restituisce token_id ed attention_mask

- preparazione della componente visiva dell'input: l'immagine corrispondente al prodotto viene caricata, viene effettuato resize e applicato RandAugmentation [62]. Può essere caricata una ulteriore immagine non corrispondente al prodotto, utilizzata nel caso in cui si esegua anche un task Image Text Matching

- viene calcolata una label che identifica la categoria di appartenenza del prodotto, utilizzata nel caso venga eseguito un task di classificazione sulla categoria del prodotto (talvolta utilizzato in fase di pre-addestramento)

- per quanto riguarda i task relativi al successo di vendita, viene calcolata la label che il task di classificazione dovrà predire o il valore numerico che il task di regressione dovrà determinare durante l'addestramento. La logica secondo cui vengono calcolati questi valori cambia quindi in base al downstream task da eseguire e per questo, la funzione che effettua questo calcolo viene passata come parametro durante la creazione dell'oggetto AmazonProductsDataset

- viene caricata l'informazione relativa alla fascia di prezzo, questa è contenuta nel file in forma di stringa, viene quindi commutata in forma numerica secondo un formato standard

AmazonProductsDataModule

AmazonProductsDataModule estende la classe *LightningDataModule* di PyTorch Lightning e si occupa di supervisionare il caricamento e la preparazione dei dati che compongono i 3 slice *train*, *validation*, *test*. Per ogni slice viene istanziato un oggetto di classe AmazonProductsDataset, a cui viene assegnato un tokenizer e una serie di trasformazioni da operare sui dati nella fase di preparazione secondo quanto stabilito nel file di configurazione. Dopo aver creato i 3 oggetti AmazonProductsDataset, AmazonProductsDataModule li dà in pasto ai 3 corrispettivi *DataLoader* (*torch.utils.data.DataLoader*), che vengono poi utilizzati dal Trainer per caricare i dati in fase di addestramento e test. Il dataloader gestisce la creazione dei batch.

3.4.2 Implementazione modello e downstream tasks

L'implementazione del modello e dei downstream tasks devono necessariamente essere trattate assieme in quando in base al downstream task implementato varia quella componente del modello chiamata head, ovvero la parte terminale, che usa il risultato delle trasformazioni precedenti per eseguire un task specifico. L'implementazione della maggior parte del modello (tutta la struttura escluse le teste usate in fase di pre-training) è coservata all'interno della classe ViLTransformerSS; le implementazioni delle teste relative ai task di pre-training sono contenute nelle classi ITMHead, MLMHead; la logica su cui si basa il calcolo dei task viene codificata all'interno di funzioni dedicate, conservate in un file chiamato "objectives.py", le relative teste sono implementate nel all'interno della classe ViLTransformerSS.

ViLTransformerSS

ViLTransformerSS implementa il corpo principale del modello, estende quindi la classe LightningModule di PyTorch Lightning dedicata alla realizzazione di modelli per reti neurali. Tra i metodi implementati all'interno di questa classe abbiamo:

- `_init_()` prevede l'esecuzione delle seguenti operazioni:
  - creazione di text_embedding, token_type_embedding
  - creazione del modello (caricato da ViT), con pesi inizializzati random
  - creazione ed inizializzazione delle head di pre-training e di quelle relative ai downstream task.
  - vengono inizializzate le metriche utilizzate per monitorare le performance dei task
  - vengono caricati i pesi a partire da un eventuale modello preaddestrato;

- `infer()`: è un metodo che viene invocato dalle funzioni che implementano i downstream task. Prende un batch e lo da in pasto a tutta quella parte di modello che precede la testa. In particolare, per ogni prodotto in input vengono calcolati text embedding e image embedding, combinando i quali si ottiene l'input da dare in pasto alla sequenza di blocks che compongono l'Interaction Transformer. I dati vengono quindi elaborati dai 12 livelli di tale Transformer, dopo di che le hidden features corrispondenti al [CLS] (da quì chiamate *cls_feats*) token vanno incontro a una operazione di normalizzazione e poi date in pasto a una rete neurale che effettua pooling. Le *cls_feats* vengono quindi restituite assieme ad altre informazioni.

- `forward()`: come dice il nome implementa il forward pass, vengono invocate tutte le funzioni che implementano la logica dei task da eseguire

- `training_step()`, *validation_step()*, *test_step()*: vengono invocati in base alla fase che si sta eseguendo, eseguono rispettivamente training, validation o test loop. Ogni passo prevede l'esecuzione di forward pass e calcolo della loss function

- `training_epoch_end()`, `validation_epoch_end()`, `test_epoch_end()`: invocati alla fine di un epoca di training/validation/test, innescano il calcolo delle metriche di fine epoca

heads

Tra le teste utilizzate e recuperate dall'implementazione di ViLT abbiamo ITMHead, MLMHead, MPPHead; a queste si aggiunge Pooler la rete neurale utilizzata per il pooling.

Per quanto riguarda le teste implementate nella classe ViLTransformerSS abbiamo invece quelle per i task: *category_prediction*, *success_prediction*, *success_regression*. Le teste vengono implementate utilizzando una rete neurale composta da due layer lineari, sul risultato del primo layer lineare viene effettuata una normalizzazione e poi calcolata la funzione di attivazione GELU. La differenza tra le 3 teste sta nel numero di neuroni in output: le prime 2 teste vengono utilizzate per effettuare classificazione quindi hanno un numero di neuroni di output pari al numero di classi; nel caso della testa utilizzata per regressione abbiamo un solo un neurone di output.

3.4.3 Calcolo delle metriche

Le metriche utilizzate per monitorare lo stato di avanzamento degli addestramenti e le performance del modello sono state recuperate dalla libreria tochmetrics messa a disposizione da PyTorch. Tra le metriche prese in considerazione abbiamo: Accuracy, F1-measure, precision, recall, Kappa di Cohen, matrice di confusione. Nel caso base, quello della classificazione multiclass single label per determinare il successo di vendita del prodotto abbiamo che tutti i dataset utilizzati sono bilanciati, questo implica che f1-measure è uguale ad accuracy, tuttavia precision, recall e matrice di confusione sono utili anche in questo caso per capire quali tipi di errori vengono commessi dal modello.

Accuracy vs f1-measure

Nel caso di esperimenti con dataset sbilanciato la f1-measure risulta più significativa di accuracy. L'accuratezza è infatti focalizzata sul numero di elementi correttamente classificati e quindi sul numero di True-Positive riconosciuti per ogni classe; per cui, se il problema risulta fortemente sbilanciato, è possibile ottenere valori di accuratezza "buoni" anche nel caso in cui gli elementi appartenenti alla classe con meno elementi vengano riconosciuti raramente: può essere sufficiente che vengano riconosciuti correttamente gli elementi appartenenti alla classe più grande. Questo può portare ad ottenere valori di accuratezza notevoli anche con un classificatore dummy su problemi molto sbilanciati. La f1-measure, al contrario, viene calcolata a partire dai valori di precision e recall, quindi il suo valore è più fortemente condizionato dal numero di false-positive e false-negative riscontrati e restituisce una valutazione più veritiera della bontà del classificatore su un problema sbilanciato.

Metriche micro/macro-averaged

Quando abbiamo un problema di classificazione multiclass, le metriche accuracy, f1-measure, precision, recall possono essere calcolate in 2 modi: micro-averaged o macro-averaged. Di seguito spiegherò la differenza prendendo in considerazione l'accuratezza, ma la stessa logica si applica alle altre metriche.

Il calcolo dell'accuratezza micro-averaged è la media delle accuratezze calcolate sul singolo campione (ovvero sul singolo prodotto). Questo vuol dire che l'accuratezza viene calcolata come il rapporto tra la somma dei prodotti correttamente classificati e il numero di prodotti totali. L'impatto di ogni classe sul risultato è quindi proporzionale al numero di prodotti appartenenti a quella classe.

Il calcolo dell'accuratezza macro-averaged è la media delle accuratezze calcolate sulle classi esistenti. In altre parole per ogni classe viene calcolata l'accuratezza dopo di che si fa il rapporto tra la somma di questi valori e il numero di classi. In questo modo l'apporto di ogni classe sul calcolo dell'accuratezza finale ha lo stesso peso, ed è indipendente dal numero di elementi che appartengono a quella classe.

Nel caso di un problema di classificazione multiclass single-label con classi bilanciate (ogni classe ha la stessa quantità di elementi) i valori assunti dalle metriche calcolati secondo i due metodi sono gli stessi. Dato un problema sbilanciato invece, micro-average darà un peso maggiore al contributo delle classi con più elementi e macro-average darà un peso maggiore al contributo di quelle con meno elementi nel calcolo della metrica. La scelta del modo in cui effettuare il calcolo va quindi effettuata in base alla classe che ci interessa maggiormente analizzare.

Kappa di Cohen

La *Kappa di Cohen* è una metrica spesso utilizzata per valutare quanto due classificatori siano in accordo tra loro; può anche essere utilizzata per valutare quanta parte della capacità di un classificatore di classificare correttamente gli elementi non possa essere attribuita al caso.

Come altre metriche, viene calcolata a partire dagli indicatori contenuti nella matrice di confusione; tuttavia, a differenza di altre metriche come l'accuratezza, tiene conto dell'eventualità che le classi possano essere sbilanciate.

La *Kappa di Cohen* viene calcolata attraverso la seguente formula:

$$K = \frac{p_0 - p_e}{1 - p_e}$$

dove: p_0 è l'accuratezza complessiva; mentre p_e è una misura che indica quanto le previsioni effettuate dal modello sono simili a quelle che derivano da un classificatore casuale.

Prendendo come esempio un problema di classificazione multiclass single label con 2 classi: abbiamo che $p_e = p_{e1} + p_{e2}$, dove p_{e1} e p_{e2} sono rispettivamente la probabilità che la previsione sia corretta e preveda classe 1/ classe 2. Assumendo che i due classificatori (ad esempio quello preso in esame e quello casuale) siano indipendenti tra loro, le probabilità p_{e1} e p_{e2} vengono calcolate moltiplicando la percentuale di elementi per cui si predice la classe x per la percentuale di elementi appartenenti alla classe x. Si ottiene:

$$p_e = p_{e1} + p_{e2} = p_{e1,prevista} * p_{e1,reale} + p_{e2,prevista} * p_{e2,reale}$$

Da un punto di vista pratico quello che fa *Kappa di Cohen* è togliere all'accuratezza del classificatore quella componente positiva portata dal caso: ovvero la probabilità che il classificatore in accordo col classificatore casuale predica correttamente la classe di un elemento. Il valore risultante è quella componente dell'accuratezza che non può essere riconducibile al caso, ma solo alla bontà del classificatore.

In generale *Kappa di Cohen* può assumere valori nell'intervallo $[-1, 1]$ e si tende a valutare il risultato secondo questa scala [74]:

- se k assume valori inferiori a 0, allora non c'è concordanza
- se k assume valori compresi tra 0-0,4, allora la concordanza è scarsa
- se k assume valori compresi tra 0,4-0,6, allora la concordanza è discreta
- se k assume valori compresi tra 0,6-0,8, la concordanza è buona
- se k assume valori compresi tra 0,8-1, la concordanza è ottima

3.4.4 Entry point del programma

L'entry point del programma si occupa principalmente di caricare la configurazione desiderata dal file di configurazione, inizializzare i principali componenti e avviare il training/test. Scendendo maggiormente nel dettaglio si ha quanto segue: per prima cosa viene impostato un seed per rendere il più possibile riproducibili i risultati ottenuti; in base al downstream task da eseguire viene individuata la funzione da utilizzare per calcolare la label associata ad ogni prodotto; viene quindi istanziato un oggetto *AmazonProductsDataModule* indicando la funzione di calcolo delle label individuata e il modello *ViLTransformerSS*; vengono impostate tutte le callback tra cui quelle che gestiscono

il salvataggiodi checkpoint e il logging delle metriche su tensorboard; viene poi istanziato un oggetto di classe Trainer della libreria PyTorch Lightning ed avviata l'esecuzione dell'addestramento/test effettuata dal trainer sui dati recuperati da datamodule con il modello precedentemente istanziato.

Capitolo 4

Modelli addestrati

In questo capitolo vengono descritti tutti gli esperimenti che hanno portato all'addestramento di alcuni modelli per la risoluzione di vari task. Per ogni addestramento o test effettuato verranno indicati i principali parametri utilizzati nel lancio dell'applicazione. Non verranno mostrati i soli esperimenti finali, ma verranno riportati anche tutti i test che hanno permesso di comprendere quali parametri adottare e l'influenza dei diversi tipi di input sul risultato, questo servirà anche a giustificare le scelte compiute.

4.1 Task sviluppati

Di seguito riporto una breve descrizione dei tre principali task sviluppati.

success_prediction

L'obiettivo primario del progetto è quello di costruire un classificatore in grado di prendere in input dati testuali ed immagini relative ad un prodotto inerente al settore della moda (capi di abbigliamento, accessori, scarpe,...) e determinare il successo di vendita atteso per questo prodotto. Si tratta quindi di un problema multiclass single-label e per il caso base si è deciso di definire 3 classi corrispondenti a 3 gradi di successo: Bad (scarso successo di vendita), Medium (successo medio), Good (buon successo di vendita). Questo task è stato denominato *success_prediction*.

Per quanto riguarda le metriche di cui il modello è stato dotato per valutare le performance su questo task abbiamo: accuracy, loss, confusion matrix, kappa di Cohen, precision, recall, f1-measure. Nell'esposizione dei vari esperimenti non verranno riportati i valori di tutte le metriche ogni volta, ma solo di quelle particolarmente interessanti per valutare l'aspetto studiato attraverso quel particolare esperimento.

success_regression

Come indicato nel capitolo relativo alla costruzione del dataset, la classe effettiva di appartenenza dei prodotti viene calcolata in base ad un valore numerico, da me denominato *success_score*. E' quindi possibile realizzare un task di regressione che presi in input gli stessi dati usati per *success_prediction* riesca a determinare tale valore. Il task è per certi versi simile a quello di classificazione, in quando l'obiettivo è sempre quello di determinare il successo previsto per un prodotto, ma il fatto che la rete neurale debba produrre un risultato numerico influenza il risultato che si verrà ad ottenere. Sarà quindi interessante effettuare un confronto tra i risultati ottenuti da un modello addestrato (fine-tuning) su questo task e quelli ottenuti dal modello addestrato su *success_prediction*. Al task di regressione è stato assegnato il nome *success_regression*.

Per valutare le performance su questo task sono state adottate le stesse metriche usate per *success_prediction*; a partire dal valore individuato dalla regressione è infatti possibile ricondursi alla classe di successo corrispondente a quel valore e quindi possono essere applicate tutte le metriche adatte ad un task di classificazione.

category_prediction

I prodotti di moda contenuti nel dataset appartengono a categorie diverse (scarpe, maglie, ecc.), e questa categoria viene tenuta in considerazione nel momento in cui si calcola il successo di vendita di un prodotto. Nel momento in cui si vuole effettuare un preaddestramento sui dataset realizzati un task significativo risulta quello di classificazione multiclass single-label dei prodotti, in cui la classe da associare ad ogni prodotto è quella corrispondente alla categoria di appartenenza. In sostanza è un task che, date le informazioni testuali e una immagine del prodotto, è in grado di determinare di quale tipo di indumento si tratta. Nel dataset sono contenute in totale 204 categorie differenti, ogni prodotto è associato ad una sola categoria per livello di astrazione. Il task è utilizzato solo in fase di pre-training ed è denominato *category_prediction*.

Per valutare le performance su questo task sono state utilizzate le seguenti metriche: accuratezza, loss e f1-measure. Particolarmente utile è la loss function che viene combinata alle loss function relative agli altri task utilizzati in fase di pre-training per addestrare il modello.

4.2 Panoramica processo di creazione modelli

Il processo che porta all'addestramento di un modello neurale profondo prevede spesso un percorso basato su tentativi ed errori, in quanto le performance del modello addestrato variano in base a molti fattori: qualità dei dati, capacità del modello di estrarre informazioni utili dai dati in fase di addestramento e parametri che caratterizzano il processo di addestramento del modello. La scelta del valore da assegnare ad ogni parametro e la conoscenza di quanta informazione il modello è in grado di estrarre dai diversi tipi di dati non sono nozioni note a priori. Per addestrare adeguatamente un modello è quindi necessario passare attraverso un processo fatto di tentativi ed errori fino ad ottenere il risultato migliore. Per questo motivo è stato necessario eseguire più addestramenti e quindi ottenere più modelli, ciascuno dei quali ha permesso di acquisire maggiore conoscenza dell'influenza di parametri e dati sul risultato. Farò riferimento ad ogni tentativo con il nome di "esperimento". In questo paragrafo riporto i principali passi che hanno segnato questo percorso così che poi sia più facilmente comprensibile l'analisi dei risultati ottenuti dai modelli addestrati.

Per prima cosa è necessario definire il punto di partenza per l'addestramento del nostro modello. Pre-addestrare un modello è una operazione anche molto onerosa e richiede grandi moli di dati, il modello da me realizzato tuttavia è basato su ViLT, i cui autori hanno messo a disposizione alcuni checkpoint: dei file contenenti pesi di un modello ViLT preaddestrato. Per questo motivo in questa prima fase ho deciso di inizializzare il modello con i pesi contenuti in uno di questi checkpoint. Il nome del checkpoint utilizzato è "vilt_200k_mlm_itm.ckpt", il modello da cui deriva è stato addestrato per 200k step su due task di pretraining: Masked Language Modeling (MLM) e Image Text Matching (ITM). I dataset utilizzati per questo preaddestramento sono ben noti in letteratura: GCC [75], SBU [76], COCO [77], VG [46]; inoltre il modello si basa per quanto riguarda la componente visual (l'elaborazione dell'immagine) su ViT-B/32 preaddestrato su ImageNet. Nei miei primi esperimenti carico sul mio modello i pesi contenuti in questo checkpoint, dopo di che sottopongo il modello ad un addestramento (fine-tuning) per l'apprendimento del task *success_prediction* sui dati contenuti nel dataset da me preparato. La versione del dataset utilizzata nei primi test e quella in cui ad ogni prodotto è associata una immagine random tra quelle rese disponibili.

Il primo aspetto che ho deciso di indagare è stato l'impatto dei diversi potenziali input sul modello. Gli input presi in considerazione sono: titolo, descrizione, immagine, prezzo. L'obiettivo del progetto era quello di creare una Deep Neural Network (DNN) cross-modale, tuttavia è stato necessario verificare

l'effettivo impatto di descrizione e immagine sul modello. Un altro aspetto che non poteva essere certo a priori era l'importanza del titolo: questo riporta spesso informazioni contenute in maniera più esaustiva anche nella descrizione e non sempre risulta presente, quando assente è sostituito da una stringa vuota. Per quanto riguarda il prezzo poi le incertezze erano ancora maggiori: molti prodotti non contengono l'informazione relativa al prezzo e i prodotti che contengono tale informazione non sono distribuiti in modo equo tra le 3 classi (Bad, Medium, Good) prese in considerazione nella classificazione. Per quei prodotti che contengono il prezzo, questa informazione può essere espressa sia come prezzo esatto che come un range. A livello formale ci si è ricondotti a una forma unica "standardizzata", come spiegato nella sezione dedicata all'elaborazione dati, questa precauzione non offre tuttavia alcuna garanzia che il modello sia in grado di trarre valore da questo dato. Molti modelli basati su trasformer, ed in particolare quello utilizzato, non sono pensati per elaborare in maniera particolarmente efficace informazioni numeriche di questo tipo; quindi, prima di effettuare esperimenti a riguardo, era difficile prevedere l'impatto di questa informazione sul modello. Come verrà mostrato nei prossimi paragrafi, da un primo gruppo di esperimenti è emerso che l'informazione relativa al prezzo risulta difficile da interpretare correttamente, viene quindi scartata; il maggior contributo verrà apportato dalla componente testuale: il titolo ha un'influenza superiore a quella attesa; l'introduzione dell'immagine porta a seppur limitati miglioramenti del modello. Da questi primi esperimenti risulta quindi che l'utilizzo di un modello cross-modale porta comunque a vantaggi rispetto ai risultati ottenuti da modelli basati unicamente su un tipo di dati.

In secondo luogo ho voluto testare i livelli di performance raggiunti da modelli preaddestrati per un diverso numero di epoche e con un diverso valore di dropout. Allenare un modello per un numero di epoche maggiore può portare dei vantaggi nel caso in cui il modello non sia stato addestrato adeguatamente (ovvero se siamo in una situazione di underfitting), oppure può portare a dei peggioramenti se siamo in una situazione di overfitting tale per cui il modello cattura pattern esistenti nel training set, ma non validi sull'intero dominio, in altre parole quando il modello perde capacità di generalizzare. Dal momento che i primi esperimenti vengono fatti addestrando il modello per un numero di epoche (5) relativamente scarso, si prova ad aumentare il numero di epoche raddoppiandolo. Come detto, un numero di epoche elevato può portare ad overfitting; per scongiurare questa eventualità possono essere usate varie tecniche, tra cui quella del dropout. Il dropout è una tecnica che viene applicata in fase di addestramento; l'overfitting è collegato alla complessità della rete, un modo per ridurre la possibilità che insorga è quello di semplificare la rete utilizzata, ad esempio "spegnendo" alcuni dei neuroni

che ne fanno parte. In sostanza applicare la tecnica del dropout consiste nei seguenti passi: viene stabilita una percentuale di probabilità con cui un qualsiasi neurone della rete può essere spento; all'inizio di ogni epoca si valuta quali neuroni resteranno accesi (e quindi verranno utilizzati per l'apprendimento durante quell'epoca) e quali invece verranno spenti; viene quindi eseguito l'addestramento in quell'epoca; al termine di tutte le epoche i pesi relativi allo stesso neurone vengono combinati e si ottengono così i valori risultanti. Durante la fase di test viene utilizzata l'intera rete neurale. Nel mio caso alcuni esperimenti sono stati dedicati a verificare l'impatto dell'aumento del dropout sulle performance.

Dopo aver fatto valutazioni su questi parametri è stato indagato un altro aspetto relativo alle immagini utilizzate: il task di classificazione *success_prediction* è stato applicato alla versione del dataset in cui ad ogni prodotto è associata la prima immagine disponibile nella lista delle immagini caricata dal venditore.

Il passo successivo è stato quello di aumentare la complessità della testa utilizzata in classificazione ed effettuare dei preaddestramenti, alcuni dei quali utilizzano il task *category_prediction* in combinazione a MLM e ITM. I preaddestramenti sono sempre stati effettuati a partire da un modello su cui sono stati caricati i pesi del checkpoint ViLT "vilt_200k_mlm_itm.ckpt". In questa fase è stata introdotta anche un ulteriore tecnica di manipolazione delle immagini che consente un aumento della capacità di generalizzazione del modello: RandAugment [62].

4.3 Parametri di configurazione

I vari esperimenti effettuati si differenziano per i diversi tipi di input, per i dataset utilizzati e per i parametri con cui è avvenuto il test/l'addestramento. Nonostante ciò la maggior parte dei parametri risulta comune ai diversi task utilizzati. In questa sezione riporto i principali parametri salvati nel file di configurazione, indicandone la funzione e il valore di base utilizzato. Al lancio degli addestramenti e dei test è possibile indicare un valore per ciascuno di questi parametri.

Parametri per l'elaborazione delle immagini

- **image_size** (default = 384); prima di essere date in pasto al Transformer le immagini vengono ridimensionate, nel nostro caso a 384x384 pixel; la scelta di tale valore è stata dettata dal modello ViT scelto per l'elaborazione delle immagini ("vit_base_patch32_384")

- **patch_size** (default = 32): è un intero che determina la dimensione delle patch in cui ogni immagine è suddivisa; nel nostro caso saranno patch di 32x32 pixel. Anche in questo caso la scelta è stata dettata dal modello ViT scelto per l'elaborazione delle immagini ("vit_base_patch32_384"). Questo, unito al fatto che le immagini hanno una dimensione 384x384 pixel determina il fatto che a partire da ogni immagine vengono ricavate 144 patch, a partire dalle quali si otterrà un numero equivalente di embedding da dare in pasto al Transformer.

- **train_transform_keys**, **val_transform_keys**: sono due liste che riportano degli id corrispondenti a trasformazioni da applicare durante l'estrazione dei dati dal dataset. Tra i valori possibili abbiamo: "pixelbert" e "pixelbert_randaug", che applicano il resize dell'immagine e nel secondo caso anche RandAugment utilizzando i valori contenuti nei parametri *randaug_N* e *randaug_M*

- **randaug_N** (default = 2), **randaug_M** (default = 9): sono due parametri usati quando viene applicato RandAugment sulle immagini. *randaug_N* determina il numero di trasformazioni applicate ad ogni immagine, *randaug_M* indica l'intensità della distorsione apportata dalle trasformazioni ed è un numero che varia nell'intervallo [0,30], dove 0 indica nessuna distorsione e 30 distorsione massima. Tra le trasformazioni che possono essere applicate abbiamo traslazioni, rotazioni, modifiche a livello di contrasto, luminosità, nitidezza, equalizzazione, solarizzazione

Parametri per l'elaborazione del testo

- **max_text_len** (default = 128): Indica il massimo numero di token dati in pasto al transformer limitatamente alla componente testuale dell'input. Nei primi esperimenti il valore utilizzato è 40, ovvero lo stesso valore usato dagli sviluppatori di ViLT; successivamente sono stati condotti esperimenti in cui a questo campo sono stati assegnati i valori 128, 256, 512

- **tokenizer** (default = "bert-base-uncased"): indica il tokenizer pre-addestrato da utilizzare. Il modello da me sviluppato, esattamente come ViLT da cui deriva, si basa su BERT per quanto riguarda l'elaborazione dell'input testuale. Di conseguenza utilizza il tokenizer *bert-base-uncased*.

- **mlm_prob** (default = 0.15): probabilità con cui ogni token può essere mascherato nel caso in cui si applichi il task MLM. Viene usato durante il pre-training

Parametri di configurazione Transformer del modello

- **hidden_size** (default = 768): indica il numero di hidden features associate ad ogni token embedding
- **num_heads** (default = 12): numero di teste del Transformer
- **num_layers** (default = 12): numero di layer che compongono il transformer
- **drop_rate** (default = 0.1): dropout associato ai neuroni della rete
- **vit** (default = "vit_base_patch32_384"): nome della versione del modello ViT adottato

Parametri per la configurazione dell'optimizer

- **optim_type** (default = "adamw"): nome dell'optiomizer scelto per affiancare il Trainer nella fase di addestramento. Gli optimizer sono algoritmi o metodi usati per modificare alcune caratteristiche della rete neurale come pesi e learning rate durate l'addestramento, questo può essere utile per ottenere un modello in grado di fornire maggiori prestazioni
- **learning_rate** (default = 1e-5): come dice il nome è il valore del learning rate con cui il modello viene addestrato. Il learning rate è un parametro che controlla quanto il modello cambia ogni volta che i pesi devono essere aggiornati in risposta all'errore stimato. Un learning rate basso impedirà al modello di apprendere adeguatamente l'esecuzione di un task, un learning rate troppo alto può impedire al modello di giungere a convergenza
- **weight_decay** (default = 0.01): viene usato per la regolarizzazione dei pesi: serve a penalizzare pesi particolarmente grandi nella rete neurale
- **max_epoch**: numero massimo di epoche di addestramento. Per quanto riguarda il fine-tuning sono stati effettuati esperimenti con 5 e 10 epoche; per il pre-training sono state utilizzate 30 epoche. Si parla di numero massimo di epoche e non semplicemente numero di epoche perchè l'addestramento può essere interrotto anticipatamente se non ci sono miglioramenti nella metrica di riferimento per un numero di epoche consecutive definite dal parametro "patience"
- **warmup_steps** (default = 1000): Un'epoca passa nel momento in cui il modello ha preso in considerazione tutti i campioni del dataset

1 volta. I campioni vengono passati al modello in gruppi (chiamati batch) e durante uno step viene effettuato l'addestramento su un batch di campioni. Durante i primi step, se i dati passati al modello hanno particolari caratteristiche non valide sull'intero dominio, può capitare che si incorra in una situazione di overfitting. Il warm-up è una tecnica che mira a ridurre la possibilità che si verifichi tale overfitting. La tecnica consiste nel far partire l'addestramento con un valore di learning rate prossimo allo zero e aumentarlo ad ogni step i primi *warmup_steps* passi. I valori assegnati sono: 500 nel caso di fine-tuning; 1000 in caso di pre-training

- **lr_mult** (default = 1): è il coefficiente per cui viene moltiplicato il learning_rate di alcuni task (*success prediction*, *success regression*, *category prediction*). I valori assegnati sono in linea di massima 1 per la fase di pre-training e 20 per la fase di fine-tuning

Parametri per la configurazione del Trainer

- **patience** (default = 2): viene utilizzato in fase di addestramento per interrompere prematuramente il training. Se non ci sono miglioramenti nelle performance del modello per un numero di epoche consecutive *patience*, l'addestramento viene interrotto.

Parametri per la configurazione dell'ambiente di esecuzione

- **seed** (default = 0): seme a partire dal quale vengono generati tutti i valori randomici, impostandolo adeguatamente si ottiene la riproducibilità delle run eseguite

- **batch_size** (default = 4096): indica il numero di esempi che vengono dati in pasto alla rete durante i passi forward/backward. Più grande è il numero, maggiore è l'occupazione di memoria del programma

- **per_gpu_batchsize**: assegno valori diversi in base allo spazio disponibile, deve essere un intero dal valore inferiore a quello assegnato a *batch_size*

- **load_path**: è una stringa contenente il path al checkpoint contenente i pesi da caricare. I pesi possono essere sia il punto di partenza per un addestramento che quelli di un modello completamente addestrato sul quale eseguire dei test

Parametri per la configurazione dei downstream tasks

- **num_success_classes** (default = 3): numero di classi da utilizzare nel task di classificazione (success prediction). Il task di classificazione base prevede la presenza di 3 classi: "Bad", "Medium", "Good".

- **alpha** (default = 0.7): è un parametro utilizzato nel calcolo del *success_score*, ovvero il grado di successo di un prodotto. Questo valore viene infatti calcolato combinando il successo riscontrato analizzando le recensioni del prodotto e la posizione del prodotto in una classifica di prodotti che appartengono alla stessa categoria; *alpha* è il peso associato a quest'ultima componente

- **medium_grouping** (default = "ALONE"): mentre nel caso base del task di classificazione si prendono in considerazioni 3 classi separate ("Bad", "Medium", "Good"), sono stati condotti anche esperimenti in cui la classe "Medium" viene accorpata ad una delle altre due classi. Il valore di questo parametro dice quale tipo di raggruppamento deve essere applicato alla classe medium. I valori che possono essere assunti dal parametro sono: "ALONE", "WITH_GOOD" e "WITH_BAD"

- **average_type** (default = "micro"): alcune metriche (accuratezza, f1-measure, precision, recall, ecc.) applicate sui downstream tasks (classificazione e regressione) possono essere calcolate in vario modo: esistono versioni micro-averaged, macro-averaged, pesate. Questo parametro consente di indicare in che modo devono essere calcolate tali metriche: "micro", "macro", "weighted"

- **cat2catID_dict_path**, **catID2cat_dict_path**: due stringhe che indicano il path a due file contenenti dictionary che consentono data la categoria di un prodotto di ricavare il corrispondente category id e viceversa

4.4 Esperimenti

Il task di default su cui sono stati condotti esperimenti è quello di classificazione multiclass single-label (*success_prediction*), ad ogni prodotto viene assegnata una tra 3 possibili classi: *Bad*, *Medium*, *Good* in base al successo di vendita ottenuto. I primi gruppi di esperimenti, quelli che precedono il primo preaddestramento, vengono condotti sul dataset *AmazonProducts_base*.

4.4.1 G1 - Impatto dei diversi input sulle performance

Il primo aspetto che si è voluto verificare è stato l'impatto delle diverse informazioni in input sull'addestramento dei modelli per determinare quali tipi di dato sono rilevanti e quali non lo sono. Per effettuare queste valutazioni e quelle successive è stato necessario confrontare molti modelli, il che vuol dire effettuare molteplici esperimenti. Per limitare i tempi richiesti si è deciso di partire dai pesi del checkpoint messo a disposizione dagli sviluppatori di ViLT (*vilt_200k_mlm_itm.ckpt*) e, a partire da questo, effettuare solo una fase di fine-tuning di 5 epoche con un valore di dropout pari a 0.1. Nelle prossime due immagini riporto delle tabelle relative ai primi 8 esperimenti effettuati, ogni esperimento corrisponde ad un modello addestrato (fine-tuning) su un insieme di dati di input diverso. I possibili input sono prezzo, titolo, descrizione, immagine; nelle tabelle per ogni tipo di input è presente una colonna contenente valori booleani: se il modello prende in considerazione un certo tipo di dato, la colonna corrispondente avrà valore "True" altrimenti "False". I preaddestramenti vengono effettuati cercando di massimizzare la metrica "accuracy" (accuratezza), calcolata nella sua versione micro-averaged. Il nome dell'esperimento è contenuto nella colonna label e lo userò come identificatore per uno specifico modello. Per quanto riguarda il modo in cui i dati testuali e numerici vengono passati al modello abbiamo che: se oltre alla descrizione è presente il titolo, questo viene anteposto alla descrizione; se oltre alla descrizione è presente il prezzo, questo viene utilizzato nella forma di stringa e anteposto alla descrizione; qualora siano presenti tutti e tre gli input il prezzo viene riportato nella forma di stringa e i tre valori vengono concatenati secondo l'ordine prezzo-titolo-descrizione. Un altro aspetto da tenere in considerazione è il fatto che i modelli tengono conto solo dei primi 40 token estratti dalla componente testuale dell'input. I valori di accuratezza riportati di seguito sono stati calcolati sul test set; gli elementi contenuti nel dataset test sono del tutto nuovi per il modello in quanto non contenuti nei dataset validation/train.

label	prezzo	titolo	descrizione	immagine	accuracy
Esperimento 1	False	False	True	False	0.4093
Esperimento 2	True	False	True	False	0.5323
Esperimento 3	False	True	True	False	0.4287
Esperimento 4	True	True	True	False	0.5351

Tabella 4.1: Accuratezza di modelli che non prendono in input immagini, addestrati (fine-tuning) per 5 epoche con dropout=0.1 a partire dai pesi *vilt_200k_mlm_itm.ckpt*

label	prezzo	titolo	descrizione	immagine	accuracy
Esperimento 5	False	False	True	True	0.4192
Esperimento 6	True	False	True	True	0.5363
Esperimento 7	False	True	True	True	0.4462
Esperimento 8	True	True	True	True	0.5452

Tabella 4.2: Accuratezza di modelli che prendono in input immagini, addestrati (fine-tuning) per 5 epoche con dropout=0.1 a partire dai pesi *vilt_200k_mlm_itm.ckpt*

In questa fase sono stati effettuati anche 8 ulteriori test: per ciascuno dei modelli preaddestrati è stato fatto un test utilizzando lo stesso set di dati in input e i pesi del modello vilt *vilt_200k_mlm_itm.ckpt* senza effettuare fine-tuning. Questi esperimenti avevano l'obiettivo di verificare che tale configurazione con cui sono stati inizializzati i pesi dei modelli restituisse un risultato randomico e per assicurarsi che i risultati di un modello fine-tuned fossero migliori e si distinguessero da quelli di un modello non addestrato. Non riporto tali esperimenti in quanto i valori di accuratezza ottenuti sono quelli attesi per un modello non addestrato: i risultati ottenuti cadono nell'intervallo [0.3325, 0.3441]. Risulta quindi evidente che il preaddestramento consente al modello di apprendere un certo grado di conoscenza dal momento che il risultato peggiore riscontrato si attesta su 0.4093.

Confronto con e senza immagine

Mentre tutti i modelli prendono in input almeno la descrizione, non tutti i modelli addestrati sono effettivamente cross-modali: i primi 4 prendono solo la componente testuale dell'input. Volendo realizzare un modello cross modale è stato di fondamentale importanza assicurarsi che i modelli fossero in grado di trarre vantaggio dall'introduzione dell'immagine. Per effettuare questa valutazione è sufficiente confrontare ogni esperimento della prima immagine con quelli con l'esperimento corrispondente nella seconda immagine: è facile notare come i modelli che prendono in input anche l'immagine ottengano sempre risultati migliori in termini di accuratezza, anche se la variazione notata risulta tutto sommato piccola.

Confronto con e senza titolo

Il secondo confronto che si può affrontare è la comparazione tra i modelli che prendono in input il titolo e modelli che non lo prendono. Per far questo vengono confrontati tra loro le seguenti coppie di modelli: *Esperimento 1* ed

Esperimento 3; *Esperimento 2* ed *Esperimento 4*; *Esperimento 5* ed *Esperimento 7*; *Esperimento 6* ed *Esperimento 8*. In ciascuno di questi casi è possibile notare un miglioramento dell'accuratezza nei modelli in cui viene utilizzato anche il titolo. Il titolo risulta quindi una informazione utile da passare al modello per far si che questo apprenda il task di classificazione. Si può inoltre notare che i miglioramenti riscontrati nei casi in cui i modelli non gestiscono il prezzo sono più consistenti di quelli rilevati in modelli che prendono in input il prezzo. Allo stesso modo i miglioramenti sono più consistenti nei modelli che tengono conto dell'immagine rispetto a quelli che non tengono conto dell'immagine.

Confronto con e senza prezzo

Il terzo confronto che si può affrontare è la comparazione tra i modelli che prendono in input il prezzo e modelli che non lo prendono. Per far questo vengono confrontati tra loro le seguenti coppie di modelli: *Esperimento 1* ed *Esperimento 2*; *Esperimento 3* ed *Esperimento 4*; *Esperimento 5* ed *Esperimento 6*; *Esperimento 7* ed *Esperimento 8*. In tutti i casi in cui viene introdotto il prezzo si ha un netto miglioramento delle performance, soprattutto quando si prendono in considerazione coppie di modelli che non prendono in input il titolo. Questo dovrebbe suggerire l'importanza del prezzo come informazione nel task di classificazione, tuttavia ci sono degli elementi che devono necessariamente essere presi in considerazione.
Il primo aspetto da prendere in considerazione è che il sistema di embedding utilizzato sulle parole non funziona altrettanto bene con i prezzi, in quanto il modello ViLT da cui sono partito, così come gli altri modelli basati su transformers di cui sono a conoscenza, non sono addestrati per elaborare informazioni numeriche. Il modello è in grado di riconoscere solo singoli numeri interi nell'intervallo [0,9] ne deriva che una singola indicazione di prezzo non viene considerata nella sua interezza ma scomposta nelle cifre che la compongono durante la creazione degli embedding. Questo è un primo segnale che fa sorgere un dubbio sulla natura del miglioramento di performance ottenuto aggiungendo l'informazione prezzo. Il transformer dovrebbe infatti essere in grado, a partire dagli embedding delle singole cifre che compongono il prezzo, di ricostruire il valore del prezzo e quindi interpretarne adeguatamente il significato; potenzialmente possibile ma molto complesso.
Un secondo aspetto che deve necessariamente essere preso in considerazione è il fatto che il prezzo non è un dato disponibile per ogni prodotto ed i prodotti per cui è noto il prezzo non sono distribuiti equamente tra le classi nei tre split dei dataset *AmazonProducts_base* e *AmazonProducts_first_image*. In particolare, prendendo in considerazione lo split Train di *AmazonProducts_first_image* la percentuale di prodotti di cui è nota l'informazione prezzo è: 33% per i

prodotti appartenenti alla classe Bad, 60% per i prodotti appartenenti alla classe Medium, 88% per i prodotti appartenenti alla classe Good. La seguente figura mette in risalto la situazione descritta.

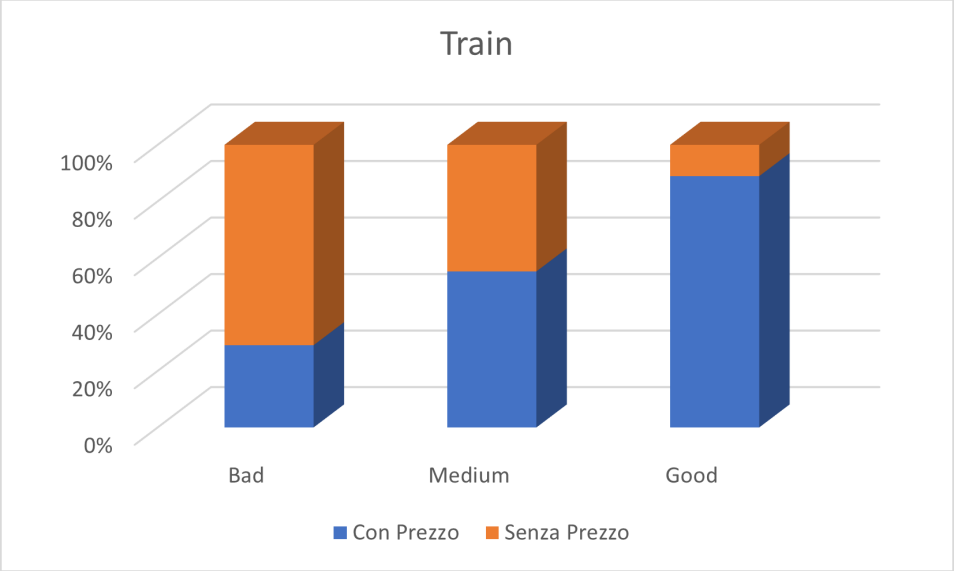


Figura 4.1: Distribuzione dei prodotti con e senza prezzo nelle tre classi dello slice Train del dataset *AmazonProducts_base*. Il numero di prodotti appartenenti a ciascuna classe è lo stesso

Una distribuzione di questo tipo, unita al drastico miglioramento di performance ottenuta introducendo il prezzo nell'insieme degli input e alla forte possibilità che il modello trovi difficoltà nel comprendere correttamente l'informazione veicolata dal prezzo lascia adito all'ipotesi che il modello classifichi i prodotti non tanto sfruttando la conoscenza del valore del prezzo quanto controllando che il prezzo sia presente o meno. Se questa ipotesi fosse vera il comportamento del modello dovrebbe essere il seguente: se viene riscontrata la presenza di valori numerici nella parte iniziale dell'input (il prezzo è presente) allora è più probabile che il prodotto venga assegnato alla classe Good, in caso contrario è più probabile che appartenga alla classe Bad. Se questa ipotesi fosse vera si dovrebbe riscontrare un calo di prodotti assegnati alla classe Medium ed un aumento di prodotti assegnati alle classi Good/Bad, dal momento che la classe Medium dovrebbe essere la più difficile da riconoscere dando grande importanza al prezzo.

Per validare questa ipotesi riporto quindi i valori di TP+FP associati ai vari esperimenti. TP sta per "true positive" ed indica i prodotti associati ad una certa classe ed effettivamente appartenenti a quella classe, mentre FP sta

per "false positive" ed indica il numero di prodotti associati ad una classe ma appartenenti ad un altra. TP+FP è quindi il numero di tutti i prodotti a cui è stata assegnata una certa classe.

label	prezzo	tp+fp Bad	tp+fp Medium	tp+fp Good	tp	fp
Esperimento 1	False	3015	1850	3835	3561	5139
Esperimento 2	True	3265	1146	4289	4631	4069
Esperimento 3	False	3435	2067	3198	3730	4970
Esperimento 4	True	2913	1776	4011	4655	4045

Tabella 4.3: Numero di prodotti assegnati alle 3 classi dai modelli che non prendono in input immagini. tp+fp Bad indica il valore di (TP+FP) ottenuto per la classe "Bad"; tp+fp Medium e tp+fp Good sono i valori corrispondenti rispettivamente alle classi "Medium" e "Good"

label	prezzo	tp_Bad	tp_Medium	tp_Good	tp
Esperimento 1	False	1339	638	1584	3561
Esperimento 2	True	1894	461	2276	4631
Esperimento 3	False	1547	760	1423	3730
Esperimento 4	True	1743	695	2217	4655

Tabella 4.4: Numero di prodotti assegnati *correttamente* alle 3 classi dai modelli che non prendono in input immagini

Come visto precedentemente, gli esperimenti vanno valutati in coppia, si devono considerare quindi: *Esperimento 1* ed *Esperimento 2*, *Esperimento 3* ed *Esperimento 4*. Osservando l'immagine 4.3 è possibile notare come l'introduzione del prezzo tra gli input porti sempre ad un calo del numero di prodotti assegnati alla classe Medium ed un aumento di quelli assegnati alla classe Good. Per quanto riguarda la classe Bad, la prima coppia si comporta come previsto mentre nella seconda coppia si verifica un calo di prodotti assegnati alla classe Bad. Quest'ultimo comportamento non va necessariamente in contraddizione con la teoria: osservando la variazione subita da *tp_Bad* in figura 4.4 si può notare che nonostante il calo netto del numero di prodotti assegnati a Bad abbiamo un aumento netto di prodotti **correttamente** assegnati a Bad. Nella stessa figura si può riscontrare un aumento nei valori di *tp_Bad* e *tp_Good* mentre si registra un calo *tp_Medium* coerente con l'ipotesi che la classe Medium risulti quella meno riconoscibile dai modelli che prendono in input il prezzo.

Si possono effettuare considerazioni analoghe sugli esperimenti relativi a modelli che prendoni in input anche immmagini:

label	prezzo	tp+fp Bad	tp+fp Medium	tp+fp Good	tp	fp
Esperimento 5	False	3055	1420	4225	3647	5053
Esperimento 6	True	3086	1555	4059	4666	4034
Esperimento 7	False	2798	2550	3352	3882	4818
Esperimento 8	True	2686	1705	4309	4743	3957

Tabella 4.5: Numero di prodotti assegnati alle 3 classi dai modelli che prendono in input immagini

label	prezzo	tp_Bad	tp_Medium	tp_Good	tp
Esperimento 5	False	1368	532	1747	3647
Esperimento 6	True	1826	626	2214	4666
Esperimento 7	False	1339	988	1555	3882
Esperimento 8	True	1684	714	2345	4743

Tabella 4.6: Numero di prodotti assegnati *correttamente* alle 3 classi dai modelli che prendono in input immagini

Anche in questo caso gli esperimenti vanno valutati in coppia, si devono considerare quindi: *Esperimento 5* ed *Esperimento 6*, *Esperimento 7* ed *Esperimento 8*. Nel caso della prima coppia, osservando la figura 4.5 si potrebbe pensare che la teoria proposta non sia valida in quanto, sebbene siano lievi, le variazioni nel numero complessivo dei prodotti assegnati alle classi è l'opposto di quello atteso: *tp+fp Bad* e *tp+fp Good* risultano inferiori in *Esperimento 6* rispetto a *Esperimento 5* e contemporaneamente *tp+fp Medium* risulta superiore. Tuttavia occorre notare che le differenze tra i valori non sono molto elevate rispetto a quelle riscontrate negli esperimenti precedenti, inoltre controllando le variazioni nel numero di prodotti correttamente assegnati alle 3 classi in figura 4.6 notiamo che sono tutte positive. Confrontando i valori ottenuti sugli indicatori dalla coppia *Esperimento 7* ed *Esperimento 8* si può riscontrare un comportamento analogo a quello visto precedentemente nella coppia *Esperimento 4* ed *Esperimento 5*.

Per cercare di dirimere ulteriormente la questione è stato condotto un ulteriore esperimento. In primo luogo è stato creato un nuovo dataset (insieme di tre split Train, Validation e Test) contenente solo prodotti il cui prezzo era

noto. Il nuovo dataset è stato chiamato *AmazonProducts_fi_price_known*. Non è stato utilizzato un dataset di questo tipo fin da subito in quanto il numero di prodotti con tali informazioni è drasticamente ridotto, di conseguenza le dimensioni del nuovo dataset sono risultate piuttosto ristrette. Avendo un dataset di questo tipo si evita che il risultato sia riconducibile alla sola presenza/assenza del prezzo. Dopo di che, per risolvere il problema legato all'incapacità del transformer di elaborare l'informazione relativa al prezzo, è stato modificato il modo in cui il modello elabora questo tipo di dato. Al posto di passare il prezzo nella forma di una stringa concatenando il valore a quello degli altri input testuali, si è scelto di passare l'informazione relativa al prezzo in forma numerica, direttamente alla testa del modello incaricata di eseguire il task di classificazione. Sono stati quindi addestrati più modelli, diversi tra loro per tipi di dato presi in input, in modo analogo a quanto visto finora e sono quindi stati calcolati i risultati di accuratezza. Dai confronti tra risultati è emerso che il risultato di modelli che differivano per la sola presenza/assenza del prezzo tra gli input risultava sostanzialmente invariato. Questo ha portato a scegliere di escludere il prezzo dai possibili input per i modelli addestrati successivamente.

4.4.2 G2 - numero epoche e dropout in fine-tuning

Il secondo blocco di esperimenti è stato dedicato allo studio dell'impatto sulle performance di due parametri: *max_epoch* e *drop_rate*. Come spiegato nella sezione dedicata all'illustrazione dei parametri, corrispondono rispettivamente al numero massimo di epoche per cui un modello viene addestrato e al valore di dropout associato ad ogni neurone del modello in fase di addestramento. In linea di massima il comportamento atteso è il seguente: se il numero di epoche per cui viene addestrato un modello è eccessivamente basso, il modello non ha modo di apprendere adeguatamente come elaborare i dati, di conseguenza alzando il numero di epoche si può osservare un miglioramento; se il numero di epoche diventa troppo elevato il modello può arrivare a riconoscere delle features tipiche del training set ma non valide sul test set, in altre parole perde capacità di generalizzare (overfitting). Il dropout è un parametro che consente in fase di addestramento di ridurre la complessità della rete che viene addestrata in una determinata epoca, riducendo di conseguenza anche la possibilità di incorrere in overfitting, dal momento che questa condizione è direttamente correlata alla complessità del modello. In sostanza nei seguenti addestramenti si tenta di aumentare il numero di epoche per ottenere risultati migliori, in questo modo tuttavia si rischia di incorrere in overfitting, per questo motivo vengono addestrati anche modelli in cui oltre al numero di epoche viene aumentato il

numero di dropout, per contrastare questa tendenza negativa. L'obiettivo è quello di determinare il giusto valore da assegnare ai due parametri.

Nelle seguenti tabelle riporto i risultati degli esperimenti effettuati. Sono stati addestrati (fine-tuning) modelli per ogni combinazione di dati in input, ovvero tutte le combinazioni di titolo, descrizione, immagine (la descrizione deve sempre essere presente). Per ciascuna combinazione di input sono stati addestrati 2 modelli: modello con *epoche*=5 e *dropout*=0.1; modello con *epoche*=10 e *dropout*=0.2. Il dataset su cui è stato effettuato il fine-tuning e i test di cui riporto i risultati è *AmazonProducts_base*. Il numero di token tenuti in considerazione per quanto riguarda componente testuale dell'input (*max_text_len*) è 40. Nella prima tabella vengono riportati gli esperimenti in cui il titolo non compare tra gli input del modello, nella seconda invece sono contenuti tutti i modelli che prendono in input il titolo. Il modello viene inoltre addestrato a partire dai pesi contenuti nel checkpoint *vilt_200k_mlm_itm.ckpt*.

label	epoche	dropout	titolo	descrizione	immagine	accuracy
Esperimento 1	5	0.1	False	True	False	0.4093
Esperimento 5	5	0.1	False	True	True	0.4192
Esperimento 11	10	0.2	False	True	False	0.4185
Esperimento 12	10	0.2	False	True	True	0.4161

Tabella 4.7: Risultati relativi a modelli che non prendono in input il titolo. I primi due modelli sono addestrati per 5 epoche con dropout=0.1; gli ultimi due modelli sono addestrati per 10 epoche con dropout=0.2

label	epoche	dropout	titolo	descrizione	immagine	accuracy
Esperimento 3	5	0.1	True	True	False	0.4287
Esperimento 7	5	0.1	True	True	True	0.4462
Esperimento 15	10	0.2	True	True	False	0.4339
Esperimento 16	10	0.2	True	True	True	0.4562

Tabella 4.8: Risultati relativi a modelli che prendono in input il titolo. I primi due modelli sono addestrati per 5 epoche con dropout=0.1; gli ultimi due modelli sono addestrati per 10 epoche con dropout=0.2

Confrontando le accuratezze ottenute applicando i diversi modelli al Test set sul task di default (success_prediction 3 classi single-label) è possibile evidenziare i seguenti aspetti:

- **importanza del titolo**: confrontando le coppie di esperimenti (1-3), (5-7), (11-15), (12-16) emerge come i modelli che prendono in input il titolo riescano ad ottenere livelli di accuratezza sensibilmente maggiori
- **utilità dell'immagine**, soprattutto quando utilizzata in combinazione al titolo. Nel caso delle coppie di esperimenti (1-5) e (11-12) il contributo dell'immagine è piccolo, in realtà nel caso di (11-12) l'introduzione dell'immagine da un contributo leggermente negativo nel calcolo dell'accuratezza. Tuttavia osservando le coppie di esperimenti (3-7) e (15-16) si può notare come l'introduzione dell'immagine nel caso sia presente anche il titolo apporti un significativo miglioramento dei risultati ottenuti, già migliorati dalla presenza del titolo. Questo potrebbe essere giustificato dall'apprendimento da parte del modello di conoscenza combinando le informazioni testuali e visive proprio grazie alla componente dedicata all'elaborazione dell'input cross-modale
- **impatto positivo dell'aumento del numero di epoche**: osservando le coppie di esperimenti (1-11), (3-15), (7-16) è possibile notare come l'aumento del numero di epoche e del dropout durante il fine-tuning abbia un'influenza positiva sulle performance. Nel caso della coppia di esperimenti (5-12) si nota un lieve peggioramento di performance, questo tuttavia evidenzia ulteriormente i benefici apportati dall'informazione contenuta nel titolo

label	epoche	dropout	titolo	descrizione	immagine	accuracy
Esperimento 16	10	0.2	True	True	True	0.4562
Esperimento 21	10	0.1	True	True	True	0.4484

Tabella 4.9: Risultati ottenuti da 2 modelli addestrati per 10 epoche. I modelli prendono entrambi in input titolo+descrizione+immagine e differiscono solo per il dropout

Nell'immagine 4.9 vengono messi a confronto due modelli che differiscono solo per il dropout. Il test eseguito mira a determinare se un innalzamento del valore di dropout possa portare ad un miglioramento significativo delle prestazioni. In effetti si riscontra un miglioramento seppur lieve del risultato. Il fatto che il miglioramento ottenuto non sia di grande entità fa pensare che questo aspetto possa esser valutato di caso in caso negli esperimenti futuri.

Da questo secondo gruppo di esperimenti vengono tratte le seguenti informazioni: viene confermato il fatto che l'insieme di informazioni da utilizzare in input ai modelli sia titolo, descrizione, immagine; viene inoltre evidenziato

un miglioramento nelle performance di modelli addestrati per un numero di epoche superiore a 5.

4.4.3 G3 - impatto delle immagini

Una volta studiati gli input da utilizzare e i valori per i parametri relativi a numero epoche e dropout si è voluto appurare l'influenza dell'immagine scelta dai prodotti sulle performance del modello. Per fare questo, è stata preparata un'apposita versione del dataset (*AmazonProducts_first_image*), che utilizza gli stessi prodotti di *AmazonProducts_base* ma si assicura di recuperare la prima immagine disponibile per ogni prodotto. Questo implica in linea di massima scegliere l'immagine che il venditore ha posto in prima posizione nella gallery dedicata al prodotto, e che dovrebbe essere quella maggiormente in grado di attrarre il cliente. Sono stati quindi addestrati 4 modelli sul task di default (*success_prediction* 3 classi single-label) che si differenziano tra loro per il tipo di input, esattamente come accade per gli esperimenti condotti sul secondo cruppo (G2). Anche in questo caso durante l'addestramento (fine-tuning) al parametro max_text_len è stato assegnato valore 40, i modelli sono inoltre stati addestrati per 10 epoche con un dropout pari a 0.2. Nella tabella di seguito riporto le performance ottenute dai modelli sul dataset test, la metrica utilizzata è sempre l'accuratezza.

label	epoche	dropout	titolo	descrizione	immagine	accuracy
Esperimento 17	10	0.2	False	True	False	0.4179
Esperimento 18	10	0.2	False	True	True	0.4246
Esperimento 19	10	0.2	True	True	False	0.4429
Esperimento 20	10	0.2	True	True	True	0.4494

Tabella 4.10: Accuratezza di modelli addestrati per 10 epoche con dropout=0.2 su dataset *AmazonProducts_first_image*

La seguente tabella contiene un riepilogo degli esperimenti realizzati per il gruppo G2 con cui vanno confrontati quelli appena mostrati:

label	epoche	dropout	titolo	descrizione	immagine	accuracy
Esperimento 11	10	0.2	False	True	False	0.4185
Esperimento 12	10	0.2	False	True	True	0.4161
Esperimento 15	10	0.2	True	True	False	0.4339
Esperimento 16	10	0.2	True	True	True	0.4562

Tabella 4.11: Accuratezza di modelli addestrati per 10 epoche con dropout=0.2 su dataset *AmazonProducts_ base*

Confrontando i risultati contenuti nella tabella in figura 4.10 con quelli contenuti in figura 4.11 è possibile notare che le differenze ottenute sono scarse e non spingono a pensare che un dataset consenta di ottenere risultati drasticamente migliori dell'altro. Si può notare infatti che nel caso di modelli che prendono in input solo destrizione e modelli che prendono in input tutti e tre i tipi di dato si ottengono risultati migliori sul dataset *AmazonProducts_ base*, al contrario, negli altri due casi si ottengono risultati migliori su *AmazonProducts_first_ image*. É quindi lecito pensare che le differenze tra le performance ottenute sui due dataset non spingano necessariamente a preferirne uno all'altro. Nei test successivi ho scelto di utilizzare *AmazonProducts_first_ image* per attenermi alla scelta del venditore e perchè la prima immagine è quella che deve essere in grado di catturare l'attenzione del cliente e quindi, dovendone scegliere una, risulta essere la più rilevante al fine di valutare il prodotto.

4.4.4 G4 - modelli pre-trained

Una volta stabiliti quali dati dare in pasto al modello e il numero di epoche da utilizzare in fase di fine-tuning, è stato effettuato il pre-training di due modelli. Il pre-training è un addestramento in cui il modello viene allenato per una quantità di tempo elevata (solitamente per un numero di epoche superiore a quello usato per il fine-tuning) e su task che mirano a consentire al modello di comprendere le caratteristiche del dominio. Come visto negli esperimenti sopra riportati, il dataset su cui il viene effettuato il pre-training non deve necessariamente essere lo stesso su cui viene effettuato fine-tuning; tuttavia l'utilizzo dello stesso dataset può consentire al modello di catturare una quantità maggiore di informazioni; questo uno dei motivi per cui sono stati pre-addestrati i due modelli sul dataset *AmazonProducts_first_ image*. I due modelli preaddestrati si differenziano per i task utilizzati durante l'addestramento: nel primo modello, che chiamerò "*pretrained_ 1*", sono stati utilizzati i task ITM e MLM come avviene nel modello ViLT; nel caso del secondo modello ("*pretrained_ 2*") a questi due task è stato aggiunto un ulteriore task chiamato "*category_ prediction*". Il task Image Text Matching (ITM) consiste nel sottoporre alla rete coppie

di informazioni testuali e immagini in cui l'immagine può, con una certa probabilità, non essere allineata al testo; la rete deve imparare a riconoscere quando le immagini corrispondono al testo. Il task Masked Language Modeling consiste nel mascherare alcuni text token e chiedere alla rete di determinare quale sia il token originale in base alla restante parte dell'input. Questi due task consentono al modello di apprendere correlazioni inter/intra-modali. Il task *category_prediction*, da me aggiunto, chiede alla rete di determinare la categoria di appartenenza del prodotto a partire dalle informazioni (titolo + descrizione + immagine) passate in input. Imparare a dedurre questa informazione potrebbe essere conveniente per il modello in task come quello di classificazione in base al successo di vendita. Per ciascuno dei task coinvolti viene calcolata una loss function (nel caso dei primi due viene calcolata una negative log-likelihood loss, come avviene in ViLT); le loss vengono quindi sommate e l'obiettivo dell'addestramento è quello di minimizzare il valore risultante. Durante l'elaborazione delle immagini che precede l'invio dell'input al modello, le immagini vengono trasformate utilizzando RandAugment: ad ogni immagine vengono applicate 2 trasformazioni con un livello di intensità pari a 9 su 30. RandAugment ha come effetto quello di aumentare la capacità di generalizzare del modello e quindi consente di ottenere prestazioni migliori su test e ridurre la possibilità di overfitting, Entrambi i modelli vengono preaddestrati per un massimo di 30 epoche (max_epoch=30), tuttavia l'addestramento termina prematuramente se per 12 epoche consecutive non c'è alcun miglioramento su accuracy (patience=12). In entrambi i casi viene utilizzato un learning rate pari a 2.0e-05 e l'optimizer "adamw" con decay_power "cosine".

A partire da ciascuno dei due modelli ottenuti (*pretrained_1* e *pretrained_2*) vengono addestrati (fine-tuning) due modelli: un modello viene addestrato sul task di default (*success_prediction* 3 classi single-label), l'altro sul task di regressione. I quattro fine-tuning condividono i restanti parametri: i modelli vengono addestrati al massimo per 10 epoche (max_epoch=10); l'addestramento si interrompe se non si riscontrano miglioramenti di performance per 2 epoche consecutive (patience=2); le immagini vengono pre-elaborate con RandAugment, ogni immmagine subisce 2 trasformazioni (randaug_N=2) con una intensità di 9 su 30 (randaug_M=9); il learning rate è impostato a 4e-4 (learning_rate=2.0e-05 e lr_mult=20); il dropout (drop_rate) a 0.1; per quanto riguarda l'optimizer viene utilizzato "adamw" con decay_power "cosine". Nella tabella contenuta nella seguente figura riporto i risultati ottenuti in fase di test su questi quattro modelli.

label	pretrained on	fine-tuning task
Esperimento 22	ITM+MLM	success_prediction
Esperimento 23	ITM+MLM	success_regression
Esperimento 24	ITM+MLM+category_prediction	success_prediction
Esperimento 25	ITM+MLM+category_prediction	success_regression

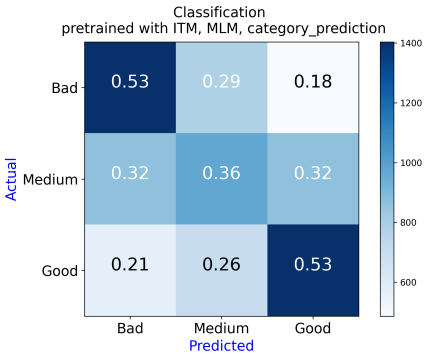
Tabella 4.12: In tabella 4.13 vengono riportate in dettaglio le performance ottenute da questi modelli

label	accuracy	f1_measure	precision	recall	loss	CohenKappa
Esperimento 22	0.4715	0.4715	0.4715	0.4715	1.0966	0.2079
Esperimento 23	0.4714	0.4714	0.4714	0.4714	0.0575	0.2058
Esperimento 24	0.4707	0.4707	0.4707	0.4707	1.1503	0.2063
Esperimento 25	0.4649	0.4649	0.4649	0.4649	0.0558	0.1964

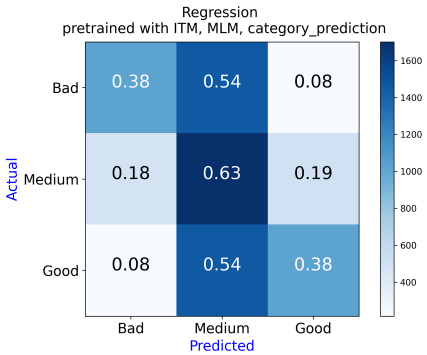
Tabella 4.13: In tabella 4.12 vengono riportati in dettaglio i task utilizzati per l'addestramento di questi modelli

Dal momento che si tratta di un problema di classificazione multiclass single-label su dataset completamente bilanciato, i valori delle metriche accuracy, f1-measure, precision, recall micro-averaged risultano identici. I risultati ottenuti non sono eccessivamente diversi, tuttavia risulta che i modelli addestrati a partire da *pretrained_1* ottengano risultati migliori. Va inoltre precisato che nel caso del task di regressione è possibile ricavare le metriche riportate calcolando la classe a partire dai dal valore di success_score calcolato dal task e confrontando la classe corrispondente a questo valore con quella effettiva.

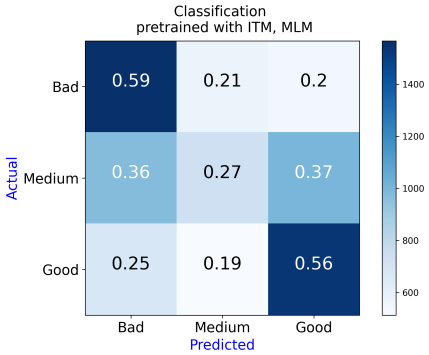
Di seguito riporto anche le matrici di confusione calcolate durante l'esecuzione del test sui modelli:



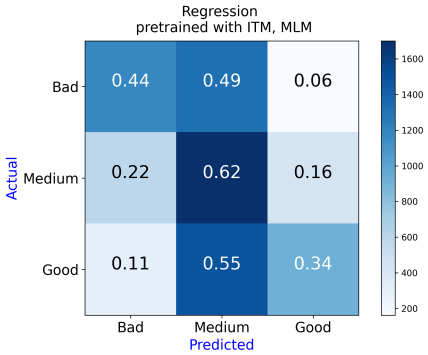
(a) Confusion matrix relativa a Esperimento 24: fine-tuning sul task success_prediction, a partire dal modello *pretrained_2* addestrato sui task MLM, ITM e category_prediction



(b) Confusion matrix relativa a Esperimento 25: fine-tuning sul task success_regression, a partire dal modello *pretrained_2* addestrato sui task MLM, ITM e category_prediction



(a) Confusion matrix relativa a Esperimento 22: fine-tuning sul task success_prediction, a partire dal modello *pretrained_1* addestrato sui task MLM e ITM



(b) Confusion matrix relativa a Esperimento 23: fine-tuning sul task success_regression, a partire dal modello *pretrained_1* addestrato sui task MLM e ITM

Confrontando modelli fine-tuned su task di regressione (success_regression, a sinistra) con quelli fine-tuned su task di classificazione (success_prediction, a destra) si può notare come il task di regressione tenda ad assegnare i prodotti alla classe Medium mentre il task di classificazione tende ad assegnarli alle classi Bad e Good. Questo è legato al modo in cui viene calcolata la classe durante l'esecuzione. Il task di regressione deve determinare un numero nell'intervallo [0,1] da associare al prodotto; tale numero nel nostro caso indica il successo riscontrato dal prodotto. Il task cerca di ridurre il proprio errore; il modo per commettere l'errore minore è quello di predire valori vicini al

centro dell'intervallo (ovvero vicini a 0.5) e i valori nell'intervallo [1/3, 2/3] corrispondono alla classe Medium, di conseguenza abbiamo molti prodotti per cui la classe prevista è Medium. Sulla classificazione invece è probabile pensare che i modelli tendano a predire le classi Good e Bad perchè sono quelle i cui prodotti sono maggiormente riconoscibili, mentre i prodotti appartenenti a Medium hanno presumibilmente caratteristiche intermedie e quindi possono essere più facilmente confusi come appartenenti ad una delle altre due classi. Confrontando i due task di classificazione (Esperimento 22 ed Esperimento 24) si notano i seguenti comportamenti: per quanto riguarda la classe Medium *Esperimento 22* produce un minore numero TP rispetto a *Esperimento 24*, sulle classi Bad e Good è vero il contrario. Il numero di FP è sempre maggiore in *Esperimento 24* mentre il numero di FN è sempre leggermente maggiore in *Esperimento 22*. Confrontando i due task di regressione (Esperimento 23 ed Esperimento 25) si nota che il valore di TP maggiore viene ottenuto sulla classe Medium in entrambi gli esperimenti ed è uguale; per quanto riguarda la classe Bad *Esperimento 23* ha un valore di TP leggermente più alto, sulla classe Good è vero il contrario. Per quanto riguarda FP *Esperimento 23* fa peggio su Bad ma meglio su Medium e Good. Per quanto riguarda FN *Esperimento 25* fa peggio su Bad mentre per le altre due classi si ottengono valori molto simili. Nel complesso, anche se di poco per quanto riguarda il task *success_prediction*, sono i modelli che derivano dal fine-tuning su *pretrained_1* ad avere sempre performance leggermente migliori.

4.4.5 G5 - classificazione su due classi

A partire dal checkpoint *pretrained_1* ottenuto dal preaddestramento sui task ITM e MLM, sono stati effettuati alcuni addestramenti (finetuning) sul task di classificazione *success_prediction* con due sole classi. Gli esperimenti si distinguono sopratutto per il dataset utilizzato e per la classe assegnata ai prodotti. Finora sono stati utilizzati i dataset *AmazonProducts_base* e *AmazonProducts_first_image*; all'interno di questi dataset i prodotti sono divisi equamente tra tre classi: Bad, Medium, Good. Volendo lavorare su due sole classi è stato necessario rivalutare questa assegnazione. Due degli esperimenti riportati in seguito lavorano su dataset diversi bilanciati: uno di questi dataset è *AmazonProducts_first_image*, in cui i prodotti precedentemente attribuiti alla classe Medium vengono ripartiti equamente tra Bad e Good; il secondo dataset prende il nome di *AmazonProducts_fi_Bad_Good_Only* e viene ottenuto da *AmazonProducts_first_image* rimuovendo tutti i prodotti recedentemente attribuiti alla classe Medium. Abbiamo poi alcuni esperimenti che operano su dataset sbilanciato: in questi casi è sempre stato utilizzato *AmazonProducts_first_image*, in cui i prodotti precedentemente attribuiti alla

classe Medium vengono assegnati alla classe Bad o alla classe Good. Mentre il dataset *AmazonProducts_fi_Bad_Good_Only* è stato creato fisicamente, la ripartizione dei prodotti precedentemente attribuiti alla classe Medium nelle altre due classi viene gestita a runtime e determinata da due iperparametri: il numero di classi *num_success_classes* e il parametro *medium_grouping*. Di seguito elenco alcuni aspetti che hanno caratterizzato gli esperimenti effettuati:

- sono stati addestrati (fine-tuning) cinque modelli a partire dal checkpoint *pretrained_1*, utilizzando il task *success_prediction*.

- il learning rate utilizzato in tutti i fine-tuning è 4e-4, infatti abbiamo: *learning_rate*=2e-5 e *lr_mult*=20. *lr_mult* è un coefficiente per cui viene moltiplicato il valore assegnato a *learning_rate* per determinare il valore effettivamente utilizzato

- il valore di dropout (*drop_rate*) utilizzato è sempre 0.1

- in tutti gli addestramenti riportati in questo paragrafo vengono dati in input ai modelli le informazioni relative a titolo, descrizione e immagine

- le performance dei modelli vengono calcolate con metriche micro-averaged nel caso venga utilizzato un dataset perfettamente bilanciato, macro-averaged in caso contrario

Di seguito riporto una tabella che riassume le performance dei modelli addestrati, seguita da un commento e dei confronti tra alcuni di questi modelli.

label	dataset	medium
Esperimento 26	AmazonProducts_fi_Bad_Good_Only	classe assente
Esperimento 27	AmazonProducts_first_image	metà in Bad/Good
Esperimento 28	AmazonProducts_first_image	con Bad
Esperimento 29	AmazonProducts_first_image	con Good
Esperimento 30	AmazonProducts_first_image	con Good

Tabella 4.14: In figura sono riportati informazioni sui dataset utilizzati nell'addestramento e test di modelli che effettuano classificazione su due classi. Le performance sono contenute nella tabella 4.15

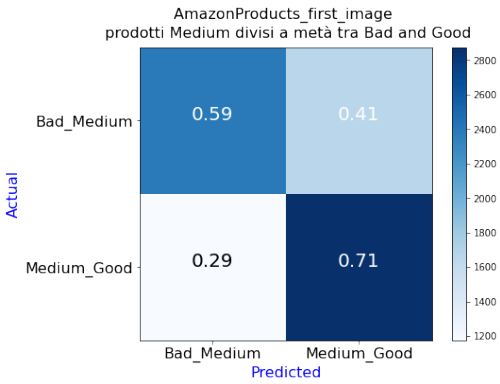
label	accuracy	f1_measure	precision	recall	loss	CohenKappa	average
Esperimento 26	0.6986	0.6986	0.6986	0.6986	0.6835	0.3972	micro
Esperimento 27	0.6462	0.6462	0.6462	0.6462	0.7105	0.2915	micro
Esperimento 28	0.6214	0.6977	0.6520	0.6214	0.6169	0.2632	macro
Esperimento 29	0.5577	0.6841	0.6407	0.5577	0.6196	0.1398	macro
Esperimento 30	0.6033	0.6871	0.6364	0.6033	0.6835	0.2270	macro

Tabella 4.15: In figura sono riportati i risultati relativi a modelli addestrati nella classificazione su due classi. La tabella 4.14 contiene indicazioni relative ai dataset utilizzati

Come si può osservare, i primi due modelli sono quelli addestrati su dataset bilanciati, per questo sono stati calcolati valori micro-averaged delle metriche e quindi i valori ottenuti per accuracy, f1-measure, precision e recall sono identici. In *Esperimento 28* i prodotti precedentemente attribuiti alla classe Medium vengono assegnati alla classe Bad; in *Esperimento 29* ed *Esperimento 30* vengono invece assegnati alla classe Good. *Esperimento 30* si differenzia da *Esperimento 29* per il fatto che durante l'addestramento sono indicati dei pesi per gli errori.

Modelli addestrati su dataset bilanciati

(a) Confusion matrix Esperimento 26

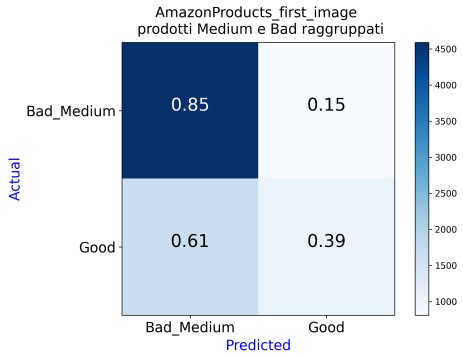


(b) Confusion matrix Esperimento 27

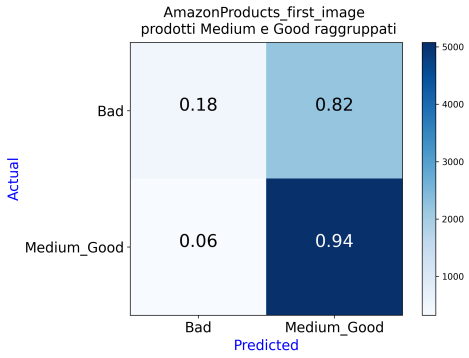
In questo paragrafo confronto i due modelli addestrati su dataset bilanciati. Esperimento 26 viene condotto su un dataset in cui i prodotti Medium sono stati rimossi, i prodotti che deve confrontare presentano quindi caratteristiche piuttosto diverse tra loro e quindi è più facile per il modello apprendere come classificare correttamente i dati. Nel caso di Esperimento 27 i prodotti Medium vengono conservati e suddivisi equamente tra le classi Bad e Good, ne consegue

che esistono prodotti che appartengono ad una delle due classi ma non presentano caratteristiche molto diverse da quelli dell'altra classe, il modello ha quindi più difficoltà nell'apprendere come effettuare correttamente la classificazione. Confrontando le due confusion matrix è facile notare come il modello *Esperimento 27* compia più errori su entrambe le classi rispetto a *Esperimento 26*. Le osservazioni fatte vengono supportate anche dalle performance riportate in figura 4.15: *Esperimento 26* è quello che produce un modello con le migliori performance confrontando i valori ottenuti da f1_measure e Cohen's Kappa.

Modelli addestrati su dataset non bilanciati



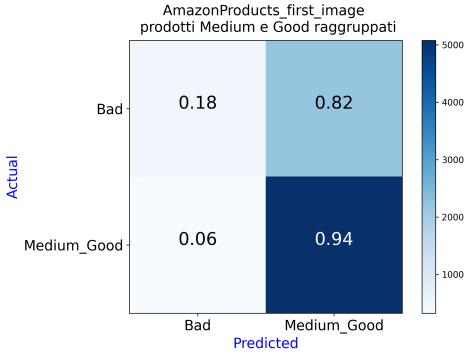
(a) Confusion matrix Esperimento 28



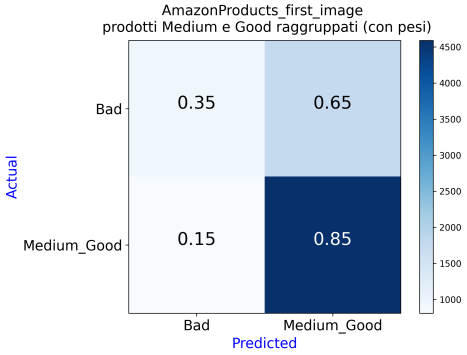
(b) Confusion matrix Esperimento 29

In figura vengono riportate le confusion matrix relative a *Esperimento 28* e *Esperimento 29*. Entrambi i modelli vengono addestrati sul dataset *AmazonProducts_first_image* ma nel caso di *Esperimento 28* i prodotti di classe Medium vengono raggruppati a quelli di classe Bad, nel caso di *Esperimento 29* a quelli di classe Good: abbiamo quindi dataset sbilanciati. Osservando le confusion matrix è facile notare come, a differenza di quanto avviene in *Esperimento 26* e *Esperimento 27*, sia in *Esperimento 28* che in *Esperimento 29* c'è la tendenza ad assegnare la maggior parte dei prodotti alla classe di dimensione maggiore. Questa tendenza è più forte in *Esperimento 29* in cui si ottiene un classificatore molto simile a quello dummy che assegna a tutti i prodotti la classe Good. Dal momento che le classi sono sbilanciate si ottengono risultati numericamente non troppo scarsi: osservando i valori delle f1-measure in 4.15 notiamo valori al di sopra di 0.68% tuttavia un classificatore come quello ottenuto da *Esperimento 29* risulta inutile nel distinguere i prodotti appartenenti alla classe Bad da tutti gli altri.

Influenza di pesi custom sugli errori



(a) Confusion matrix Esperimento 29



(b) Confusion matrix Esperimento 30

Come mostrato nell'ultimo confronto il classificatore ottenuto da *Esperimento 29* è molto simile ad un classificatore dummy. Si è voluta indagare la possibilità di ottenere un modello migliore specificando pesi diversi per diversi errori. É quindi stato condotto un ulteriore addestramento in cui gli errori su prodotti di classe Bad vengono cosiderati due volte più gravi di quelli su prodotti di classe Good. Questo ha portato ad un miglioramento del risultato ottenuto: il classificatore così realizzato ottiene risultati più simili a quelli ottenuti per *Esperimento 28*.

Questi esperimenti hanno portato a riconoscere le difficoltà dei classificatori così addestrati nel suddividere prodotti in classi diverse quando questi non molto simili tra loro, in particolare questo si vede confrontando i modelli addestrati su *AmazonProducts_first_image* con quelli addestrati su *AmazonProducts_fi_Bad_Good_only*.

4.4.6 G6 - introduzione di features tra gli input

Nei primi esperimenti effettuati, quelli in cui i modelli venivano addestrati con solo finetuning, è stato utilizzato un input testuale di lunghezza pari a 40 token. Negli esperimenti in cui sono stati effettuati i preaddestramenti e poi in tutti quelli fin quì riportati è stato utilizzato un input testuale di lunghezza pari a 128 token. Dopo aver cercato di massimizzare i risultati ottimizzando il modello, si è deciso di cercare di ottenere miglioramenti di performance dando in pasto al modello maggiori informazioni. Per questo motivo, sono stati effettuati ulteriori esperimenti in cui è stato passato al modello un input testuale contenente 256 e 512 token. Per quanto riguarda il contenuto dell'input testuale, a titolo e descrizione è stato aggiunto il contenuto del campo feature:

sono informazioni che descrivono il prodotto in maniera analoga a quanto fa la descrizione, ma vengono mostrati all'utente nella forma di elenco puntato. Di seguito elenco alcuni aspetti che hanno caratterizzato gli esperimenti condotti:

- sono stati addestrati (fine-tuning) cinque modelli a partire dal checkpoint *pretrained_1* utilizzando il task *success_prediction*; uno di questi, tuttavia, non prende in input feature ed è utilizzato solo come termine di paragone
- il learning rate utilizzato in tutti i fine-tuning è 4e-4, infatti abbiamo: *learning_rate*=2e-5 e *lr_mult*=20
- il valore di dropout (*drop_rate*) utilizzato è 0.1
- tutti i modelli vengono addestrati su un dataset bilanciato, per questo motivo le performance vengono calcolate con metriche micro-averaged, ne deriva che i valori di accuracy, f1-measure, recall, precision sono uguali tra loro. Per praticità viene riportato solo il valore di accuracy

La seguente figura riporta una tabella contenente le informazioni che caratterizzano gli esperimenti e ne descrivono le performance

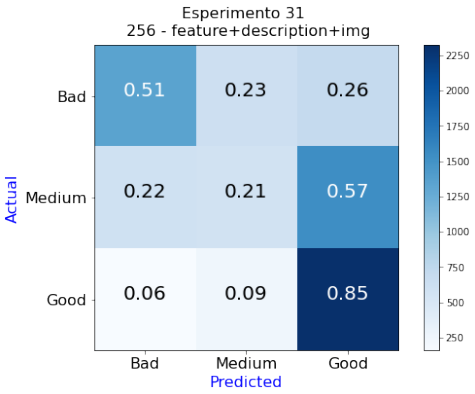
label	max_text_len	titolo	features	descrizione	immagine	accuracy	CohenKappa	loss
Esperimento 31	256	False	True	True	True	0.5211	0.2842	0.9670
Esperimento 32	256	False	False	True	True	0.4409	0.1616	1.1151
Esperimento 33	512	True	True	True	False	0.5371	0.3072	0.9237
Esperimento 34	512	True	True	True	True	0.5460	0.3198	0.9639
Esperimento 35	512	True	True	True	True	0.7795	0.5590	0.4953

Tabella 4.16: In figura sono riportati i risultati relativi a modelli addestrati studiando il contributo delle informazioni contenute nel campo feature sul task success_prediction. *n_class* il numero di classi diverse su cui viene effettuato il task di classificazione. *max_text_len* indica la lunghezza (numero di token) della componente testuale di input passata al modello. Negli esperimenti 31, 32, 33, 34 i modelli sono stati addestrati sul dataset AmazonProducts_first_image e la classificazione avviene su 3 classi; in esperimento 35 il modello è stato addestrato sul dataset AmazonProducts_fi_Bad_Good_Only e la classificazione avviene su 2 classi

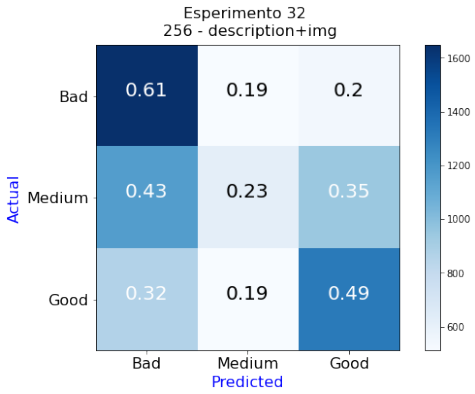
Come si può notare osservando la tabella in figura 4.16 i primi due esperimenti (*Esperimento 31* e *Esperimento 32*) sono caratterizzati dalla mancanza del titolo tra le informazioni passate in input al modello e da un input testuale lungo 256 token, mentre tutti gli altri hanno un input testuale lungo 512 token. *Esperimento 32* è l'unico caso in cui viene addestrato un modello

a cui non viene passato feature come input. *Esperimento 33* differisce da *Esperimento 34* per l'assenza di immagine in input; entrambi i modelli sono addestrati su un input testuale lungo 512 token. *Esperimento 34* corrisponde al modello che prende in input la quantità di informazioni più completa ed è quello che ottiene performance migliori nella classificazione su tre classi (Bad, Medium, Good). *Esperimento 35* è un modello simile a *Esperimento 34* per input e parametri usati per il fine-tuning, tuttavia il modello viene addestrato nella classificazione su due sole classi (Bad e Good), utilizzando il dataset *AmazonProducts_fi_Bad_Good_Only*. Di seguito metto in evidenza alcune informazioni che emergono confrontando i risultati ottenuti da questi modelli. Il primo confronto effettuato è stato quello tra *Esperimento 31* ed *Esperimento 32*, i due modelli differiscono solo per la presenza o assenza di feature tra gli input testuali. Paragonando i valori ottenuti su accuracy è evidente come l'introduzione di feature porti ad un drastico aumento delle prestazioni ottenute: si passa da 0.4409 di accuracy ottenuta da *Esperimento 32* a 0.5211 ottenuta da *Esperimento 31* tenendo in considerazione le feature. *Esperimento 34* si differenzia da *Esperimento 31* in quanto l'input testuale ha una lunghezza maggiore (512 token invece di 256) e contiene anche il titolo. Le performance calcolate da *Esperimento 34* risultano migliori di quelle ottenute da *Esperimento 31*, è possibile verificarlo confrontando tutte le metriche riportate. Il miglioramento risulta comunque molto inferiore a quello ottenuto introducendo feature. *Esperimento 33* è stato effettuato per valutare il contributo dell'immagine sulle performance ottenute: il modello è infatti stato addestrato in maniera simile a quello di *Esperimento 34*, l'unica differenza con quest'ultimo è la mancanza in *Esperimento 33* delle immagini nell'input. le prestazioni ottenute dal modello che prende in input anche le immagini sono migliori ma la variazione è così piccola da risultare inconsistente. Questo implica che il modello non riesce a trarre un vero giovamento dall'utilizzo dell'immagine in combinazione con il testo.

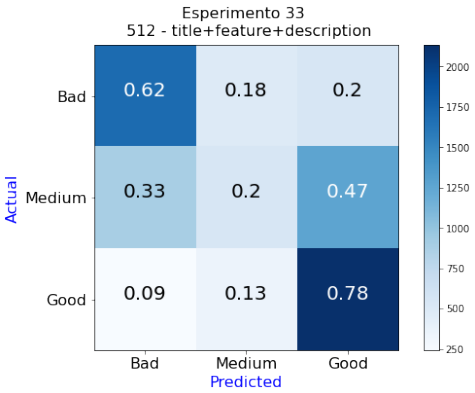
Di seguito riporto le matrici di confusione relative ai primi quattro esperimenti:



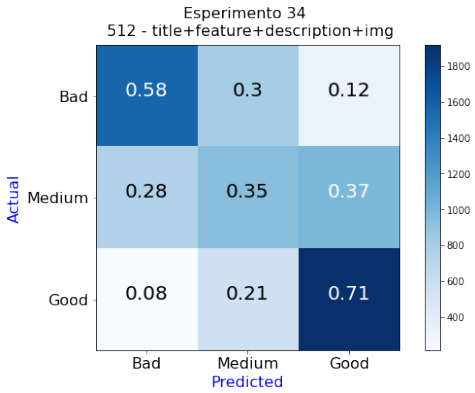
(a) Confusion matrix Esperimento 31



(b) Confusion matrix Esperimento 32



(a) Confusion matrix Esperimento 33



(b) Confusion matrix Esperimento 34

Come si può osservare confrontando le matrici di confusione relative ai quattro modelli, la classe maggiormente difficile da riconoscere è Medium: nel migliore dei casi (Esperimento 34) viene riconosciuta con una frequenza del 35%, che è leggermente meglio di quello che sarebbe in grado di fare un classificatore dummy, tuttavia il tasso di classificazioni corrette sulle classi Good e Bad risulta maggiore. Nel caso dei tre modelli che prendono in input feature si può notare una maggiore tendenza a classificare correttamente i prodotti appartenenti a classe Good rispetto a quelli appartenenti a classe Bad.

Visto il successo ottenuto da questi modelli, è stato effettuato un addestramento (*Esperimento 35*) anche sul task success_prediction su due classi (Bad, Good) utilizzando il dataset *AmazonProducts_fi_Bad_Good_only*. Come riportato in figura 4.16, il modello addestrato fornendo in input anche le informazioni

contenute in feature raggiunge un valore di accuratezza pari a 0.7795, contro il risultato ottenuto da *Esperimento 26*: 0.6986 (figura 4.15). Anche i valori ottenuti per il coefficiente statistico *Kappa di Cohen* indicano un miglioramento: *Esperimento 26* raggiungeva 0.3972 mentre *Esperimento 35* 0.5590.

(a) Confusion matrix Esperimento 26 (b) Confusion Matrix Esperimento 35

In figura viene effettuato un confronto tra la confusion matrix ottenuta da *Esperimento 26* e quella ottenuta da *Esperimento 35*. Si nota che l'introduzione di feature tra le informazioni passate in input ha portato all'addestramento di un modello in grado di riconoscere molto meglio i prodotti di classe Good: si passa da 76% a 94% di prodotti Good correttamente riconosciuti e ad un lieve calo della percentuale di prodotti Bad classificati correttamente: *Esperimento 26* riconosce correttamente il 64% dei prodotti mentre *Esperimento 35* solo il 61%.

Questo gruppo di esperimenti ha dato prova che l'introduzione di maggiori informazioni testuali e l'utilizzo di una più consistente quantità di dati in input ha un impatto fortemente positivo sulle performance ottenute dai modelli.

4.4.7 G7 - preaddestramento con feature

Visti i miglioramenti di performance ottenuti su quei modelli che prendono in input anche le informazioni contenute nel campo feature, si è deciso di tentare di migliorare ulteriormente i risultati effettuando un preaddestramento in cui vengono utilizzate come informazioni in input titolo, descrizione, feature e immagini. Il preaddestramento è durato 30 epoche, è stato utilizzato un learning rate pari a 2e-5 e dropout (drop_rate) pari a 0.1. Il modello è stato preaddestrato utilizzando i task Image Text Matching (ITM) e Masked Language Modeling (MLM); il modello è stato pre-addestrato sui dati contenuti

in AmazonProducts_first_image. Il modello ottenuto è stato poi sottoposto a due finetuning sul task di classificazione *success_prediction*, sono stati così ottenuti i modelli di *Esperimento 36* e *Esperimento 37*. In entrambi i casi:

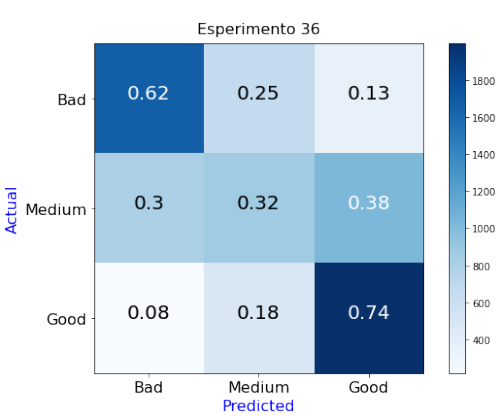
- il modello prende in input informazioni relative a titolo, feature, description, immagine
- è stato utilizzato un valore di dropout pari a 0.1
- è stato utilizzato un valore di learning rate pari a 4e-4; più nel dettaglio i valori passati al modello sono: *learning_rate*=2e-5 e *lr_mult*=20
- il numero di token tenuti in considerazione per la componente testuale dell'input è (max_text_len) 512

In *Esperimento 36*, il modello viene addestrato sul dataset AmazonProducts_first_image, il task di classificazione deve suddividere correttamente il prodotto su 3 classi; nel caso di *Esperimento 37*, il modello viene addestrato sul dataset AmazonProducts_fi_Bad_Good_Only e la classificazione avviene solo su 2 classi. In entrambi i casi il dataset è bilanciato, per questo motivo le performance vengono calcolate con metriche micro-averaged, ne deriva che i valori di accuracy, f1-measure, recall, precision sono uguali tra loro. Per praticità, nella tabella 4.17 viene riportato solo il valore di accuracy.

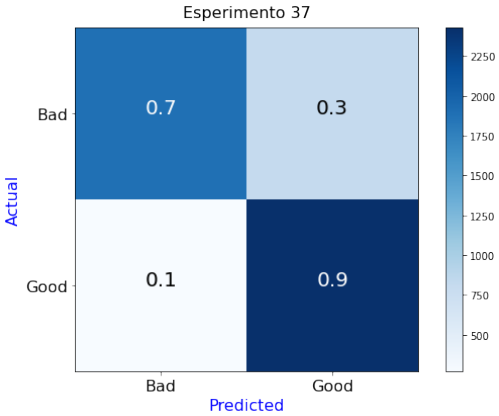
label	n_class	accuracy	CohenKappa	loss
Esperimento 36	3	0.5590	0.3394	0.9610
Esperimento 37	2	0.8031	0.6062	0.5970

Tabella 4.17: Risultati per i modelli ottenuti da preaddestramento e finetuning su informazioni provenienti dai campi: title, feature, description, image.

I risultati ottenuti da Esperimento 36 e Esperimento 37 risultano in entrambi i casi migliori di quelli ottenuti a partire dai modelli preaddestrati senza feature, rispettivamente Esperimento 34 ed Esperimento 35 . Di seguito riporto le confusion matrix relative ai due nuovi modelli.



(a) Confusion matrix Esperimento 36



(b) Confusion Matrix Esperimento 37

Capitolo 5

Conclusioni

Con questa tesi viene presentata per la prima volta una soluzione software per la previsione del successo di vendita di prodotti di Moda che fa uso di un modello cross-modale basato su Vision-Language Transformer. Il modello effettua le previsioni basandosi su dati noti al venditore prima della commercializzazione del prodotto. Il lavoro svolto include la creazione del dataset utilizzato nell'addestramento del modello.

La prima parte di questo progetto è stata dedicata alla creazione di un dataset. Sono stati utilizzati dati relativi a prodotti venduti su Amazon tra Maggio 1996 e Ottobre 2018. I dati provengono da una raccolta chiamata Amazon Review Dataset che contiene informazioni su prodotti appartenenti a diverse categorie; sono stati selezionati solo quelli legati al settore Moda. Il dataset realizzato contiene informazioni relative a: titolo (nome del prodotto), descrizione, feature (un elenco di caratteristiche), immagini, le valutazioni assegnate dai recensori (numero intero compreso tra 1 e 5), un indicatore della quantità di prodotto venduta, i nomi delle categorie a cui il prodotto appartiene. Amazon Review Dataset può contenere informazioni parziali, duplicate o memorizzate in modo non standard. É stato quindi necessario selezionare i prodotti, e poi effettuare pulizia e standardizzazione del contenuto dei campi. A partire dalle valutazioni dei consumatori e dalla quantità di prodotto venduto è stato ricavato il valore relativo al successo di vendita, in base a questo dato i prodotti sono stati suddivisi tra 3 classi: Bad (basso successo di vendita), Medium (successo di vendita medio), Good (ottimo successo di vendita). Il dataset creato contiene 78300 prodotti ed è bilanciato sulle tre classi. É stato necessario creare il dataset in quanto nessuno di quelli disponibili online contenente prodotti di Moda riporta, per ogni articolo, la quantità di prodotto venduta.

La seconda parte della tesi ha portato allo sviluppo di un modello cross-

modale basato su Vision-Language Transformer. Il modello prende in input una immagine e informazioni testuali che descrivono il prodotto ed effettua una previsione sul successo di vendita. La previsione avviene nella forma di una classificazione su 3 classi: Bad, Medium, Good. Sono stati effettuati molteplici addestramenti per ottimizzare le performance del modello e comprendere quali fattori abbiano la massima influenza sul risultato. Tutti i modelli prendono in input la descrizione a cui possono essere aggiunti titolo, feature e immagine. Tra questi, feature contiene le informazioni che hanno un maggiore impatto sull'accuratezza media: produce un miglioramento pari a 8%. Il secondo dato più rilevante è il titolo che può portare un aumento dell'accuratezza pari a 4%. Il contributo dell'immagine è minimo: includere l'immagine tra gli input porta ad un miglioramento di poco inferiore all'1%. L'importanza di feature probabilmente deriva dal fatto che questo campo contiene una quantità di testo corposa che descrive il prodotto in modo dettagliato e quindi può avere molta influenza sul numero di vendite. L'importanza del titolo può essere ricondotta alla sua funzione: quella di attirare l'attenzione dell'utente e contemporaneamente descrivere il prodotto, il fatto che risulti meno influente di feature può essere ricondotto alla brevità del testo. La scarsa influenza delle immagini è invece un elemento inatteso e può essere sintomo di un problema nei dati: le immagini raramente contengono il solo prodotto in primo piano, cosa che faciliterebbe l'estrazione di informazioni. Il risultato migliore è stato ottenuto con un modello che prende in input titolo, feature, descrizione e immagine; il livello di accuratezza raggiunto da tale modello è 55.90%.

Il lavoro di tesi si presta ad essere migliorato e arricchito con nuove funzionalità. Il dataset potrebbe essere esteso aggiungendo prodotti pubblicati recentemente. Il modello potrebbe poi essere modificato affinchè proponga nomi e descrizioni per prodotti ancora non disponibili sul mercato, così da aiutare il venditore a massimizzare la probabilità di successo. Il modello attuale compie la maggior parte degli errori sulla classe Medium, risolvere questo problema porterebbe ad un aumento significativo delle performance. L'applicazione di tecniche di bagging e boosting potrebbe migliorare ulteriormente i risultati. Si potrebbero condurre studi di explainability analizzando a quali embedding viene assegnato il massimo attention score, dato un certo input. Questa analisi potrebbe essere applicata sia al testo che alle immagini. Conoscendo quali elementi delle immagini catturano maggiormente l'attenzione del modello, si potrebbe trovare una spiegazione al basso impatto delle figure sulle performance e determinare un modo per migliorare il contributo della componente visiva dell'input. Lo studio degli attention score potrebbe anche permetterebbe di capire se il modello è affetto da bias, ad esempio legati a sesso o etnia di persone che compaiono nelle immagini dei prodotti.

Bibliografia

[1] Harshall Lamba. Intuitive understanding of attention mechanism in deep learning, 2019.

[2] OECD. E-commerce in the time of covid-19. 2020.

[3] Dzmitry Bahdanau, Kyunghyun Cho, and Yoshua Bengio. Neural machine translation by jointly learning to align and translate. In Yoshua Bengio and Yann LeCun, editors, *3rd International Conference on Learning Representations, ICLR 2015, San Diego, CA, USA, May 7-9, 2015, Conference Track Proceedings*, 2015.

[4] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser, and Illia Polosukhin. Attention is all you need. In Isabelle Guyon, Ulrike von Luxburg, Samy Bengio, Hanna M. Wallach, Rob Fergus, S. V. N. Vishwanathan, and Roman Garnett, editors, *Advances in Neural Information Processing Systems 30: Annual Conference on Neural Information Processing Systems 2017, December 4-9, 2017, Long Beach, CA, USA*, pages 5998–6008, 2017.

[5] Jay Alammar. The illustrated transformer, 2018.

[6] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. BERT: pre-training of deep bidirectional transformers for language understanding. In Jill Burstein, Christy Doran, and Thamar Solorio, editors, *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, NAACL-HLT 2019, Minneapolis, MN, USA, June 2-7, 2019, Volume 1 (Long and Short Papers)*, pages 4171–4186. Association for Computational Linguistics, 2019.

[7] Alexey Dosovitskiy, Lucas Beyer, Alexander Kolesnikov, Dirk Weissenborn, Xiaohua Zhai, Thomas Unterthiner, Mostafa Dehghani, Matthias Minderer, Georg Heigold, Sylvain Gelly, Jakob Uszkoreit, and Neil Houlsby. An image is worth 16x16 words: Transformers for image recognition at scale. In

9th International Conference on Learning Representations, ICLR 2021, Virtual Event, Austria, May 3-7, 2021. OpenReview.net, 2021.

[8] Y. LeCun, B. Boser, J. S. Denker, D. Henderson, R. E. Howard, W. Hubbard, and L. D. Jackel. Backpropagation Applied to Handwritten Zip Code Recognition. *Neural Computation*, 1(4):541–551, 12 1989.

[9] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E. Hinton. Imagenet classification with deep convolutional neural networks. In Peter L. Bartlett, Fernando C. N. Pereira, Christopher J. C. Burges, Léon Bottou, and Kilian Q. Weinberger, editors, *Advances in Neural Information Processing Systems 25: 26th Annual Conference on Neural Information Processing Systems 2012. Proceedings of a meeting held December 3-6, 2012, Lake Tahoe, Nevada, United States*, pages 1106–1114, 2012.

[10] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *2016 IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2016, Las Vegas, NV, USA, June 27-30, 2016*, pages 770–778. IEEE Computer Society, 2016.

[11] Xiaolong Wang, Ross B. Girshick, Abhinav Gupta, and Kaiming He. Non-local neural networks. In *2018 IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2018, Salt Lake City, UT, USA, June 18-22, 2018*, pages 7794–7803. Computer Vision Foundation / IEEE Computer Society, 2018.

[12] Nicolas Carion, Francisco Massa, Gabriel Synnaeve, Nicolas Usunier, Alexander Kirillov, and Sergey Zagoruyko. End-to-end object detection with transformers. In Andrea Vedaldi, Horst Bischof, Thomas Brox, and Jan-Michael Frahm, editors, *Computer Vision - ECCV 2020 - 16th European Conference, Glasgow, UK, August 23-28, 2020, Proceedings, Part I*, volume 12346 of *Lecture Notes in Computer Science*, pages 213–229. Springer, 2020.

[13] Niki Parmar, Prajit Ramachandran, Ashish Vaswani, Irwan Bello, Anselm Levskaya, and Jon Shlens. Stand-alone self-attention in vision models. In Hanna M. Wallach, Hugo Larochelle, Alina Beygelzimer, Florence d'Alché-Buc, Emily B. Fox, and Roman Garnett, editors, *Advances in Neural Information Processing Systems 32: Annual Conference on Neural Information Processing Systems 2019, NeurIPS 2019, December 8-14, 2019, Vancouver, BC, Canada*, pages 68–80, 2019.

[14] Huiyu Wang, Yukun Zhu, Bradley Green, Hartwig Adam, Alan L. Yuille, and Liang-Chieh Chen. Axial-deeplab: Stand-alone axial-attention for

panoptic segmentation. In Andrea Vedaldi, Horst Bischof, Thomas Brox, and Jan-Michael Frahm, editors, *Computer Vision - ECCV 2020 - 16th European Conference, Glasgow, UK, August 23-28, 2020, Proceedings, Part IV*, volume 12349 of *Lecture Notes in Computer Science*, pages 108–126. Springer, 2020.

[15] Dhruv Mahajan, Ross B. Girshick, Vignesh Ramanathan, Kaiming He, Manohar Paluri, Yixuan Li, Ashwin Bharambe, and Laurens van der Maaten. Exploring the limits of weakly supervised pretraining. In Vittorio Ferrari, Martial Hebert, Cristian Sminchisescu, and Yair Weiss, editors, *Computer Vision - ECCV 2018 - 15th European Conference, Munich, Germany, September 8-14, 2018, Proceedings, Part II*, volume 11206 of *Lecture Notes in Computer Science*, pages 185–201. Springer, 2018.

[16] Qizhe Xie, Minh-Thang Luong, Eduard H. Hovy, and Quoc V. Le. Self-training with noisy student improves imagenet classification. In *2020 IEEE/CVF Conference on Computer Vision and Pattern Recognition, CVPR 2020, Seattle, WA, USA, June 13-19, 2020*, pages 10684–10695. Computer Vision Foundation / IEEE, 2020.

[17] Alexander Kolesnikov, Lucas Beyer, Xiaohua Zhai, Joan Puigcerver, Jessica Yung, Sylvain Gelly, and Neil Houlsby. Big transfer (bit): General visual representation learning. In Andrea Vedaldi, Horst Bischof, Thomas Brox, and Jan-Michael Frahm, editors, *Computer Vision - ECCV 2020 - 16th European Conference, Glasgow, UK, August 23-28, 2020, Proceedings, Part V*, volume 12350 of *Lecture Notes in Computer Science*, pages 491–507. Springer, 2020.

[18] Niki Parmar, Ashish Vaswani, Jakob Uszkoreit, Lukasz Kaiser, Noam Shazeer, Alexander Ku, and Dustin Tran. Image transformer. In Jennifer G. Dy and Andreas Krause, editors, *Proceedings of the 35th International Conference on Machine Learning, ICML 2018, Stockholmsmässan, Stockholm, Sweden, July 10-15, 2018*, volume 80 of *Proceedings of Machine Learning Research*, pages 4052–4061. PMLR, 2018.

[19] Han Hu, Zheng Zhang, Zhenda Xie, and Stephen Lin. Local relation networks for image recognition. In *2019 IEEE/CVF International Conference on Computer Vision, ICCV 2019, Seoul, Korea (South), October 27 - November 2, 2019*, pages 3463–3472. IEEE, 2019.

[20] Hengshuang Zhao, Jiaya Jia, and Vladlen Koltun. Exploring self-attention for image recognition. In *2020 IEEE/CVF Conference on Computer Vision*

and Pattern Recognition, CVPR 2020, Seattle, WA, USA, June 13-19, 2020, pages 10073–10082. Computer Vision Foundation / IEEE, 2020.

[21] Rewon Child, Scott Gray, Alec Radford, and Ilya Sutskever. Generating long sequences with sparse transformers. *CoRR*, abs/1904.10509, 2019.

[22] Dirk Weissenborn, Oscar Täckström, and Jakob Uszkoreit. Scaling autoregressive video models. In *8th International Conference on Learning Representations, ICLR 2020, Addis Ababa, Ethiopia, April 26-30, 2020*. OpenReview.net, 2020.

[23] Jonathan Ho, Nal Kalchbrenner, Dirk Weissenborn, and Tim Salimans. Axial attention in multidimensional transformers. *CoRR*, abs/1912.12180, 2019.

[24] Jean-Baptiste Cordonnier, Andreas Loukas, and Martin Jaggi. On the relationship between self-attention and convolutional layers. In *8th International Conference on Learning Representations, ICLR 2020, Addis Ababa, Ethiopia, April 26-30, 2020*. OpenReview.net, 2020.

[25] Irwan Bello, Barret Zoph, Quoc Le, Ashish Vaswani, and Jonathon Shlens. Attention augmented convolutional networks. In *2019 IEEE/CVF International Conference on Computer Vision, ICCV 2019, Seoul, Korea (South), October 27 - November 2, 2019*, pages 3285–3294. IEEE, 2019.

[26] Han Hu, Jiayuan Gu, Zheng Zhang, Jifeng Dai, and Yichen Wei. Relation networks for object detection. In *2018 IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2018, Salt Lake City, UT, USA, June 18-22, 2018*, pages 3588–3597. Computer Vision Foundation / IEEE Computer Society, 2018.

[27] Chen Sun, Austin Myers, Carl Vondrick, Kevin Murphy, and Cordelia Schmid. Videobert: A joint model for video and language representation learning. In *2019 IEEE/CVF International Conference on Computer Vision, ICCV 2019, Seoul, Korea (South), October 27 - November 2, 2019*, pages 7463–7472. IEEE, 2019.

[28] Bichen Wu, Chenfeng Xu, Xiaoliang Dai, Alvin Wan, Peizhao Zhang, Masayoshi Tomizuka, Kurt Keutzer, and Peter Vajda. Visual transformers: Token-based image representation and processing for computer vision. *CoRR*, abs/2006.03677, 2020.

[29] Francesco Locatello, Dirk Weissenborn, Thomas Unterthiner, Aravindh Mahendran, Georg Heigold, Jakob Uszkoreit, Alexey Dosovitskiy, and Thomas Kipf. Object-centric learning with slot attention. In Hugo Larochelle,

Marc'Aurelio Ranzato, Raia Hadsell, Maria-Florina Balcan, and Hsuan-Tien Lin, editors, *Advances in Neural Information Processing Systems 33: Annual Conference on Neural Information Processing Systems 2020, NeurIPS 2020, December 6-12, 2020, virtual*, 2020.

[30] Yen-Chun Chen, Linjie Li, Licheng Yu, Ahmed El Kholy, Faisal Ahmed, Zhe Gan, Yu Cheng, and Jingjing Liu. UNITER: universal image-text representation learning. In Andrea Vedaldi, Horst Bischof, Thomas Brox, and Jan-Michael Frahm, editors, *Computer Vision - ECCV 2020 - 16th European Conference, Glasgow, UK, August 23-28, 2020, Proceedings, Part XXX*, volume 12375 of *Lecture Notes in Computer Science*, pages 104–120. Springer, 2020.

[31] Wonjae Kim, Bokyung Son, and Ildoo Kim. Vilt: Vision-and-language transformer without convolution or region supervision. In Marina Meila and Tong Zhang, editors, *Proceedings of the 38th International Conference on Machine Learning, ICML 2021, 18-24 July 2021, Virtual Event*, volume 139 of *Proceedings of Machine Learning Research*, pages 5583–5594. PMLR, 2021.

[32] Jiasen Lu, Dhruv Batra, Devi Parikh, and Stefan Lee. Vilbert: Pretraining task-agnostic visiolinguistic representations for vision-and-language tasks. In Hanna M. Wallach, Hugo Larochelle, Alina Beygelzimer, Florence d'Alché-Buc, Emily B. Fox, and Roman Garnett, editors, *Advances in Neural Information Processing Systems 32: Annual Conference on Neural Information Processing Systems 2019, NeurIPS 2019, December 8-14, 2019, Vancouver, BC, Canada*, pages 13–23, 2019.

[33] Weijie Su, Xizhou Zhu, Yue Cao, Bin Li, Lewei Lu, Furu Wei, and Jifeng Dai. VL-BERT: pre-training of generic visual-linguistic representations. In *8th International Conference on Learning Representations, ICLR 2020, Addis Ababa, Ethiopia, April 26-30, 2020*. OpenReview.net, 2020.

[34] Liunian Harold Li, Mark Yatskar, Da Yin, Cho-Jui Hsieh, and Kai-Wei Chang. Visualbert: A simple and performant baseline for vision and language. *CoRR*, abs/1908.03557, 2019.

[35] Hao Tan and Mohit Bansal. LXMERT: learning cross-modality encoder representations from transformers. In Kentaro Inui, Jing Jiang, Vincent Ng, and Xiaojun Wan, editors, *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing, EMNLP-IJCNLP 2019,*

Hong Kong, China, November 3-7, 2019, pages 5099–5110. Association for Computational Linguistics, 2019.

[36] Gen Li, Nan Duan, Yuejian Fang, Ming Gong, and Daxin Jiang. Unicoder-vl: A universal encoder for vision and language by cross-modal pre-training. In *The Thirty-Fourth AAAI Conference on Artificial Intelligence, AAAI 2020, The Thirty-Second Innovative Applications of Artificial Intelligence Conference, IAAI 2020, The Tenth AAAI Symposium on Educational Advances in Artificial Intelligence, EAAI 2020, New York, NY, USA, February 7-12, 2020*, pages 11336–11344. AAAI Press, 2020.

[37] Jiasen Lu, Vedanuj Goswami, Marcus Rohrbach, Devi Parikh, and Stefan Lee. 12-in-1: Multi-task vision and language representation learning. In *2020 IEEE/CVF Conference on Computer Vision and Pattern Recognition, CVPR 2020, Seattle, WA, USA, June 13-19, 2020*, pages 10434–10443. Computer Vision Foundation / IEEE, 2020.

[38] Jaemin Cho, Jiasen Lu, Dustin Schwenk, Hannaneh Hajishirzi, and Aniruddha Kembhavi. X-LXMERT: paint, caption and answer questions with multi-modal transformers. In Bonnie Webber, Trevor Cohn, Yulan He, and Yang Liu, editors, *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing, EMNLP 2020, Online, November 16-20, 2020*, pages 8785–8805. Association for Computational Linguistics, 2020.

[39] Di Qi, Lin Su, Jia Song, Edward Cui, Taroon Bharti, and Arun Sacheti. Imagebert: Cross-modal pre-training with large-scale weak-supervised image-text data. *CoRR*, abs/2001.07966, 2020.

[40] Ethan Perez, Florian Strub, Harm de Vries, Vincent Dumoulin, and Aaron C. Courville. Film: Visual reasoning with a general conditioning layer. In Sheila A. McIlraith and Kilian Q. Weinberger, editors, *Proceedings of the Thirty-Second AAAI Conference on Artificial Intelligence, (AAAI-18), the 30th innovative Applications of Artificial Intelligence (IAAI-18), and the 8th AAAI Symposium on Educational Advances in Artificial Intelligence (EAAI-18), New Orleans, Louisiana, USA, February 2-7, 2018*, pages 3942–3951. AAAI Press, 2018.

[41] Zhicheng Huang, Zhaoyang Zeng, Bei Liu, Dongmei Fu, and Jianlong Fu. Pixel-bert: Aligning image pixels with text by deep multi-modal transformers. *CoRR*, abs/2004.00849, 2020.

[42] Xiujun Li, Xi Yin, Chunyuan Li, Pengchuan Zhang, Xiaowei Hu, Lei Zhang, Lijuan Wang, Houdong Hu, Li Dong, Furu Wei, Yejin Choi, and Jianfeng

Gao. Oscar: Object-semantics aligned pre-training for vision-language tasks. In Andrea Vedaldi, Horst Bischof, Thomas Brox, and Jan-Michael Frahm, editors, *Computer Vision - ECCV 2020 - 16th European Conference, Glasgow, UK, August 23-28, 2020, Proceedings, Part XXX*, volume 12375 of *Lecture Notes in Computer Science*, pages 121–137. Springer, 2020.

[43] Zhe Gan, Yen-Chun Chen, Linjie Li, Chen Zhu, Yu Cheng, and Jingjing Liu. Large-scale adversarial training for vision-and-language representation learning. In Hugo Larochelle, Marc'Aurelio Ranzato, Raia Hadsell, Maria-Florina Balcan, and Hsuan-Tien Lin, editors, *Advances in Neural Information Processing Systems 33: Annual Conference on Neural Information Processing Systems 2020, NeurIPS 2020, December 6-12, 2020, virtual*, 2020.

[44] Fei Yu, Jiji Tang, Weichong Yin, Yu Sun, Hao Tian, Hua Wu, and Haifeng Wang. Ernie-vil: Knowledge enhanced vision-language representations through scene graphs. In *Thirty-Fifth AAAI Conference on Artificial Intelligence, AAAI 2021, Thirty-Third Conference on Innovative Applications of Artificial Intelligence, IAAI 2021, The Eleventh Symposium on Educational Advances in Artificial Intelligence, EAAI 2021, Virtual Event, February 2-9, 2021*, pages 3208–3216. AAAI Press, 2021.

[45] Pengchuan Zhang, Xiujun Li, Xiaowei Hu, Jianwei Yang, Lei Zhang, Lijuan Wang, Yejin Choi, and Jianfeng Gao. Vinvl: Revisiting visual representations in vision-language models. In *IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2021, virtual, June 19-25, 2021*, pages 5579–5588. Computer Vision Foundation / IEEE, 2021.

[46] Ranjay Krishna, Yuke Zhu, Oliver Groth, Justin Johnson, Kenji Hata, Joshua Kravitz, Stephanie Chen, Yannis Kalantidis, Li-Jia Li, David A. Shamma, Michael S. Bernstein, and Li Fei-Fei. Visual genome: Connecting language and vision using crowdsourced dense image annotations. *Int. J. Comput. Vis.*, 123(1):32–73, 2017.

[47] Yujia Xie, Xiangfeng Wang, Ruijia Wang, and Hongyuan Zha. A fast proximal point method for computing exact wasserstein distance. In Amir Globerson and Ricardo Silva, editors, *Proceedings of the Thirty-Fifth Conference on Uncertainty in Artificial Intelligence, UAI 2019, Tel Aviv, Israel, July 22-25, 2019*, volume 115 of *Proceedings of Machine Learning Research*, pages 433–453. AUAI Press, 2019.

[48] Olga Russakovsky, Jia Deng, Hao Su, Jonathan Krause, Sanjeev Satheesh, Sean Ma, Zhiheng Huang, Andrej Karpathy, Aditya Khosla, Michael S.

Bernstein, Alexander C. Berg, and Li Fei-Fei. Imagenet large scale visual recognition challenge. *Int. J. Comput. Vis.*, 115(3):211–252, 2015.

[49] Hugo Touvron, Matthieu Cord, Matthijs Douze, Francisco Massa, Alexandre Sablayrolles, and Hervé Jégou. Training data-efficient image transformers & distillation through attention. In Marina Meila and Tong Zhang, editors, *Proceedings of the 38th International Conference on Machine Learning, ICML 2021, 18-24 July 2021, Virtual Event*, volume 139 of *Proceedings of Machine Learning Research*, pages 10347–10357. PMLR, 2021.

[50] Fartash Faghri, David J. Fleet, Jamie Ryan Kiros, and Sanja Fidler. VSE++: improving visual-semantic embeddings with hard negatives. In *British Machine Vision Conference 2018, BMVC 2018, Newcastle, UK, September 3-6, 2018*, page 12. BMVA Press, 2018.

[51] Kuang-Huei Lee, Xi Chen, Gang Hua, Houdong Hu, and Xiaodong He. Stacked cross attention for image-text matching. In Vittorio Ferrari, Martial Hebert, Cristian Sminchisescu, and Yair Weiss, editors, *Computer Vision - ECCV 2018 - 15th European Conference, Munich, Germany, September 8-14, 2018, Proceedings, Part IV*, volume 11208 of *Lecture Notes in Computer Science*, pages 212–228. Springer, 2018.

[52] Alec Radford, Jong Wook Kim, Chris Hallacy, Aditya Ramesh, Gabriel Goh, Sandhini Agarwal, Girish Sastry, Amanda Askell, Pamela Mishkin, Jack Clark, Gretchen Krueger, and Ilya Sutskever. Learning transferable visual models from natural language supervision. In Marina Meila and Tong Zhang, editors, *Proceedings of the 38th International Conference on Machine Learning, ICML 2021, 18-24 July 2021, Virtual Event*, volume 139 of *Proceedings of Machine Learning Research*, pages 8748–8763. PMLR, 2021.

[53] Alane Suhr, Stephanie Zhou, Ally Zhang, Iris Zhang, Huajun Bai, and Yoav Artzi. A corpus for reasoning about natural language grounded in photographs. In Anna Korhonen, David R. Traum, and Lluís Màrquez, editors, *Proceedings of the 57th Conference of the Association for Computational Linguistics, ACL 2019, Florence, Italy, July 28- August 2, 2019, Volume 1: Long Papers*, pages 6418–6428. Association for Computational Linguistics, 2019.

[54] Duy-Kien Nguyen, Vedanuj Goswami, and Xinlei Chen. Movie: Revisiting modulated convolutions for visual counting and beyond. In *9th International Conference on Learning Representations, ICLR 2021, Virtual Event, Austria, May 3-7, 2021*. OpenReview.net, 2021.

[55] Emanuele Bugliarello, Ryan Cotterell, Naoaki Okazaki, and Desmond Elliott. Multimodal pretraining unmasked: Unifying the vision and language berts. *CoRR*, abs/2011.15124, 2020.

[56] Peter Anderson, Xiaodong He, Chris Buehler, Damien Teney, Mark Johnson, Stephen Gould, and Lei Zhang. Bottom-up and top-down attention for image captioning and visual question answering. In *2018 IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2018, Salt Lake City, UT, USA, June 18-22, 2018*, pages 6077–6086. Computer Vision Foundation / IEEE Computer Society, 2018.

[57] Shaoqing Ren, Kaiming He, Ross B. Girshick, and Jian Sun. Faster R-CNN: towards real-time object detection with region proposal networks. In Corinna Cortes, Neil D. Lawrence, Daniel D. Lee, Masashi Sugiyama, and Roman Garnett, editors, *Advances in Neural Information Processing Systems 28: Annual Conference on Neural Information Processing Systems 2015, December 7-12, 2015, Montreal, Quebec, Canada*, pages 91–99, 2015.

[58] Kaiming He, Georgia Gkioxari, Piotr Dollár, and Ross B. Girshick. Mask R-CNN. In *IEEE International Conference on Computer Vision, ICCV 2017, Venice, Italy, October 22-29, 2017*, pages 2980–2988. IEEE Computer Society, 2017.

[59] Huaizu Jiang, Ishan Misra, Marcus Rohrbach, Erik G. Learned-Miller, and Xinlei Chen. In defense of grid features for visual question answering. In *2020 IEEE/CVF Conference on Computer Vision and Pattern Recognition, CVPR 2020, Seattle, WA, USA, June 13-19, 2020*, pages 10264–10273. Computer Vision Foundation / IEEE, 2020.

[60] Yu Jiang, Vivek Natarajan, Xinlei Chen, Marcus Rohrbach, Dhruv Batra, and Devi Parikh. Pythia v0.1: the winning entry to the VQA challenge 2018. *CoRR*, abs/1807.09956, 2018.

[61] Yiming Cui, Wanxiang Che, Ting Liu, Bing Qin, Ziqing Yang, Shijin Wang, and Guoping Hu. Pre-training with whole word masking for chinese BERT. *CoRR*, abs/1906.08101, 2019.

[62] Ekin D. Cubuk, Barret Zoph, Jonathon Shlens, and Quoc V. Le. Randaugment: Practical automated data augmentation with a reduced search space. In *2020 IEEE/CVF Conference on Computer Vision and Pattern Recognition, CVPR Workshops 2020, Seattle, WA, USA, June 14-19, 2020*, pages 3008–3017. Computer Vision Foundation / IEEE, 2020.

[63] Nikolay Archak, Anindya Ghose, and Panagiotis G. Ipeirotis. Deriving the pricing power of product features by mining consumer reviews. *Manag. Sci.*, 57(8):1485–1509, 2011.

[64] Alain Yee Loong Chong, Boying Li, Eric W.T. Ngai, Eugene Ch'ng, and Filbert Lee. Predicting online product sales via online reviews, sentiments, and promotion strategies. *International Journal of Operations & Production Management*, 36, 2016. School:C-Bus1, C-HSS8,.

[65] Anindya Ghose and Panagiotis G. Ipeirotis. Estimating the helpfulness and economic impact of product reviews: Mining text and reviewer characteristics. *IEEE Trans. Knowl. Data Eng.*, 23(10):1498–1512, 2011.

[66] Reid Pryzant, Youngjoo Chung, and Dan Jurafsky. Predicting sales from the language of product descriptions. In Jon Degenhardt, Surya Kallumadi, Maarten de Rijke, Luo Si, Andrew Trotman, and Yinghui Xu, editors, *Proceedings of the SIGIR 2017 Workshop On eCommerce co-located with the 40th International ACM SIGIR Conference on Research and Development in Information Retrieval, eCOM@SIGIR 2017, Tokyo, Japan, August 11, 2017*, volume 2311 of *CEUR Workshop Proceedings*. CEUR-WS.org, 2017.

[67] Wei Di, Neel Sundaresan, Robinson Piramuthu, and Anurag Bhardwaj. Is a picture really worth a thousand words?: - on the role of images in e-commerce. In Ben Carterette, Fernando Diaz, Carlos Castillo, and Donald Metzler, editors, *Seventh ACM International Conference on Web Search and Data Mining, WSDM 2014, New York, NY, USA, February 24-28, 2014*, pages 633–642. ACM, 2014.

[68] Michael Kranzlein. A multiple classifier system for predicting best-selling amazon products. 2018.

[69] Jianmo Ni, Jiacheng Li, and Julian J. McAuley. Justifying recommendations using distantly-labeled reviews and fine-grained aspects. In Kentaro Inui, Jing Jiang, Vincent Ng, and Xiaojun Wan, editors, *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing, EMNLP-IJCNLP 2019, Hong Kong, China, November 3-7, 2019*, pages 188–197. Association for Computational Linguistics, 2019.

[70] Andrew Collette. *Python and HDF5*. O'Reilly, 2013.

[71] Dehong Gao, Linbo Jin, Ben Chen, Minghui Qiu, Peng Li, Yi Wei, Yi Hu, and Hao Wang. Fashionbert: Text and image matching with adaptive loss for cross-modal retrieval. In Jimmy Huang, Yi Chang, Xueqi Cheng,

Jaap Kamps, Vanessa Murdock, Ji-Rong Wen, and Yiqun Liu, editors, *Proceedings of the 43rd International ACM SIGIR conference on research and development in Information Retrieval, SIGIR 2020, Virtual Event, China, July 25-30, 2020*, pages 2251–2260. ACM, 2020.

[72] Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, Alban Desmaison, Andreas Köpf, Edward Z. Yang, Zachary DeVito, Martin Raison, Alykhan Tejani, Sasank Chilamkurthy, Benoit Steiner, Lu Fang, Junjie Bai, and Soumith Chintala. Pytorch: An imperative style, high-performance deep learning library. In Hanna M. Wallach, Hugo Larochelle, Alina Beygelzimer, Florence d'Alché-Buc, Emily B. Fox, and Roman Garnett, editors, *Advances in Neural Information Processing Systems 32: Annual Conference on Neural Information Processing Systems 2019, NeurIPS 2019, December 8-14, 2019, Vancouver, BC, Canada*, pages 8024–8035, 2019.

[73] William Falcon and The PyTorch Lightning team. PyTorch Lightning, 3 2019.

[74] Wikipedia. Kappa di cohen — wikipedia, l'enciclopedia libera, 2019. [Online; in data 19-ottobre-2021].

[75] Qi Wang, Junyu Gao, Wei Lin, and Yuan Yuan. Pixel-wise crowd understanding via synthetic data. *Int. J. Comput. Vis.*, 129(1):225–245, 2021.

[76] Vicente Ordonez, Girish Kulkarni, and Tamara L. Berg. Im2text: Describing images using 1 million captioned photographs. In John Shawe-Taylor, Richard S. Zemel, Peter L. Bartlett, Fernando C. N. Pereira, and Kilian Q. Weinberger, editors, *Advances in Neural Information Processing Systems 24: 25th Annual Conference on Neural Information Processing Systems 2011. Proceedings of a meeting held 12-14 December 2011, Granada, Spain*, pages 1143–1151, 2011.

[77] Tsung-Yi Lin, Michael Maire, Serge J. Belongie, James Hays, Pietro Perona, Deva Ramanan, Piotr Dollár, and C. Lawrence Zitnick. Microsoft COCO: common objects in context. In David J. Fleet, Tomás Pajdla, Bernt Schiele, and Tinne Tuytelaars, editors, *Computer Vision - ECCV 2014 - 13th European Conference, Zurich, Switzerland, September 6-12, 2014, Proceedings, Part V*, volume 8693 of *Lecture Notes in Computer Science*, pages 740–755. Springer, 2014.