UNIVERSITÀ
POLITECNICA
DELLE MARCHE

Facoltà di Ingegneria
Corso di Laurea in Ingegneria Informatica e dell'Automazione

Predizione dell'interesse sui social network utilizzando informazioni multimodali

Multimodal approach for predicting social network engagement

Candidato:
Alessio Paolucci

Relatore:
Prof. Emanuele Frontoni

Correlatore:
Dott. Marco Mameli

Anno Accademico 2021-2022

UNIVERSITÀ
POLITECNICA
DELLE MARCHE

Facoltà di Ingegneria
Corso di Laurea in Ingegneria Informatica e dell'Automazione

Predizione dell'interesse sui social network utilizzando informazioni multimodali

Multimodal approach for predicting social network engagement

Candidato:
Alessio Paolucci

Relatore:
Prof. Emanuele Frontoni

Correlatore:
Dott. Marco Mameli

Anno Accademico 2021-2022

Università Politecnica delle Marche
Facoltà di Ingegneria
Corso di Laurea in Ingegneria Informatica e dell'Automazione
Via Brecce Bianche – 60131 Ancona (AN), Italy

A mia nonna Maria
che per pochi mesi non ha potuto assistere
alla mia proclamazione di laurea come avrebbe tanto voluto...

Sommario

Questa tesi è frutto di un progetto di tirocinio interno alla Facoltà di Ingegneria Informatica e dell'Automazione dell'Università Politecnica delle Marche. Il Dott. Marco Mameli, nonché correlatore, mi ha seguito in questo percorso molto formativo e perciò lo ringrazio.
L'argomento principale dell'elaborato è chiaro già dal titolo: i **social network**. Essi svolgono un ruolo centrale nella vita di tutti i giorni. Sempre più utenti infatti li utilizzano con costanza. E' quindi fondamentale per i creatori di contenuti (che essi siano brand oppure singole persone) essere competitivi cercando cioè di pubblicare post che possano raggiungere più account possibili. Più aumentano i contenuti, più le bacheche sono intasate, rischiando quindi che l'utente finale non riesca neanche a visualizzare il post del creatore. E' importante quindi pubblicare contenuti di qualità cosicché possano comparire (possibilmente per primi) nei feed dei propri seguaci, in modo da aumentare il loro *coinvolgimento*.
Le piattaforme social dispongono già di tool e metriche per analizzare i propri account. Esse però sono metriche a posteriori, si basano cioè su dati già pubblicati. Abbiamo avuto l'idea di cercare di prevedere l'interesse futuro di un post *a priori*, così da pubblicarlo solo se potenzialmente di successo. E' proprio questa una delle particolarità del progetto, ovvero la **predizione dell'engagement rate**.
In più, la stima dell'interesse avviene tramite un **approccio multimodale**. La maggior parte dei progetti di Machine (e Deep) Learning utilizzano encoder che trattano solo immagini, o solo testo/numeri. L'altra idea che ci è venuta in mente è di elaborare *unitamente* tutte le informazioni presenti. Questo è un altro aspetto interessante e caratteristico del progetto.
Gli argomenti trattati sono molto stimolanti, attuali ed innovativi. Essi riguardano il **Deep Learning** (apprendimento profondo): *regressori*, *reti neurali*, *analisi di immagini e di testo (VGG16 e BERT)*, *approccio multimodale*, *previsioni di dati* e molto altro...

Indice

Elenco delle figure

Elenco delle tabelle

Elenco del Codice

Capitolo 1

Introduzione

1.1 Contesto

I social network svolgono un ruolo centrale nella vita quotidiana. In particolare, 1,5 miliardi di persone usano Instagram ogni mese [5], il che lo rende uno dei social network più popolari al mondo [18]. Inoltre, più del 90% degli utenti segue un account business [7]. Naturalmente, le aziende riconoscono diverse opportunità derivanti da questo uso diffuso dei canali social e cercano di cambiare i loro piani aziendali concentrandosi sui potenziali clienti così come essi emergono dai social network stessi. D'altra parte, gli influencer che sono creatori di comunità legate tra loro da specifici temi, cercano di avere sempre più follower così da renderle più numerose e coese. Inoltre, più le loro comunità sono estese ed unite, più attirano le attenzioni di potenziali partner pubblicitari. E' quindi fondamentale sia per i brand che per gli influencer sfruttare al meglio i loro contenuti, ottimizzandoli, in modo da aumentare il coinvolgimento degli utenti. La parola coinvolgimento (`engagement` in inglese) infatti è una delle parole chiave di questo progetto.

1.2 Obiettivo

L'obiettivo principale è prevedere l'engagement di un post futuro, data l'immagine e la caption (descrizione). I dati di input sono quindi sia testuali/numerici che visivi (immagini). Questo consiste in un approccio detto `multimodale`, altra parola chiave del progetto.

Per raggiungere l'obiettivo è stata stabilita una metodologia che utilizza reti neurali convoluzionali ed elaborazione del linguaggio naturale, due aspetti comuni del Deep Learning. L'approccio e le tecnologie adottate verrano spiegati nei capitoli successivi.

Capitolo 2

Metodologia

2.1 Scopo del progetto

Come indicato in precedenza, lo scopo principale di questo progetto è cercare di prevedere (con un'ottima accuratezza) l'`engagement` di un futuro post di Instagram, in modo da conoscere in anticipo se avrà potenzialmente successo. Ma come si calcola l'engagement? *Later.com*, uno dei siti di social network management più famosi, propone una formula di `engagement rate (ER)` [10] che in questo progetto è stata semplificata escludendo il numero di commenti (non disponibile a priori per un post futuro) in:

$$\boldsymbol{EngagementRate} = \frac{likes}{followers} * 100 \tag{2.1}$$

2.2 Strategia

E' necessario, al fine di raggiungere l'obiettivo, utilizzare una corretta strategia di apprendimento automatico. Come indicato nell'articolo [1], che si basa sul corso di Andrew Ng "AI For Everyone" [12], ci sono tre fasi chiave (fig. 2.1) di un progetto generico di Deep Learning:

Raccolta dei dati La raccolta dei dati è una delle attività più importanti in un progetto di apprendimento automatico. In questo passaggio si raccolgono informazioni da una o più fonti diverse. La raccolta dei dati consentirà poi, nella terza fase, di analizzarli per trovare schemi ricorrenti e costruire modelli predittivi. I modelli predittivi sono soddisfacenti solo quanto lo sono i dati da cui vengono costruiti. Questo passaggio è quindi fondamentale per sviluppare modelli accurati.

Preparazione dei dati e feature selection La preparazione dei dati è il processo di elaborazione dei dati raccolti in una forma più appropriata per la modellazione. Include una pulizia dei dati, una selezione ed un'eventuale trasformazione (cambiando per esempio il tipo di variabili). La feature selection in particolare consiste nell'individuazione delle caratteristiche maggiormente significative rispetto alle altre, andando ad eliminare eventuali informazioni superflue (sezione 3.1.3).

Sviluppo modello predittivo Questa fase consiste nello sviluppare il modello di Deep Learning per apprendere l'input (variabili, testo e immagine) in modo da produrre l'output, ovvero la previsione. I dati vengono etichettati e si utilizzano per calcolare i pesi corretti. Si utilizza quindi un algoritmo di `apprendimento supervisionato`. Questi tipi di algoritmi costruiscono il modello esaminando un gran numero di dati etichettati e cercano di ridurre al minimo le perdite. Questi tipi di algoritmi vengono spiegati più in dettaglio nella sezione successiva 2.2.1.

Figura 2.1: Metodologia

2.2.1 Regressione o classificazione?

Nel Deep Learning esistono due grandi famiglie di algoritmi: *regressori* e *classificatori*. [13] Sia i problemi di regressione che quelli di classificazione rientrano nella categoria del `supervised learning` (apprendimento supervisionato). Tutti gli algoritmi di apprendimento supervisionato partono dal presupposto che, fornendo al sistema un numero adeguato di esempi, questo accumulerà un'esperienza E sufficiente da permettergli di creare una funzione h_a adeguata ad approssimare la funzione h_b (e quindi il comportamento desiderato da chi ha fornito gli esempi). Data la similitudine tra le funzioni h_a e h_b, quando vengono proposti al sistema dei dati in ingresso non presenti nella sua esperienza E, la funzione h_a dovrebbe essere in grado di approssimare in maniera sufficientemente precisa la funzione h_b e fornire delle risposte sufficientemente soddisfacenti. Per raggiungere questo obiettivo il sistema sfrutta spesso due principi, che sono quello della *distribuzione matematica* e quello della *funzione di verosimiglianza*. Una volta identificata la distribuzione matematica che lega il variare dei valori degli input ai valori degli output desiderati, il sistema

sceglie i parametri che massimizzano la probabilità dei dati ed esprime la funzione di verosimiglianza appropriata.

Nonostante la similarità nell'obiettivo generale, i problemi di classificazione e regressione sono diversi. Nei problemi di classificazione, l'algoritmo apprende una funzione per mappare gli input agli output in cui il valore di output è un'etichetta di classe discreta (ad es. gatto o cane, maligno o benigno, spam o non spam, ecc.). Al contrario, i problemi di regressione riguardano la mappatura degli input sugli output in cui l'output è un numero reale continuo (ad es. il prezzo di una casa). Si può facilmente intuire che questo progetto si tratta di un problema di `regressione` dovendo stimare l'engagement rate, che è un numero reale (eq.[2.1]).

2.2.2 Regressione tramite rete neurale artificiale

Le reti neurali artificiali sono un modello matematico/informatico di calcolo basato sulle reti neurali biologiche. Tale modello è costituito da un gruppo di interconnessioni di informazioni costituite da neuroni artificiali e processi che utilizzano un approccio di connessionismo di calcolo. Nella maggior parte dei casi una rete neurale artificiale è un sistema adattivo che cambia la propria struttura in base a informazioni esterne o interne che scorrono attraverso la rete stessa durante la fase di apprendimento. In termini pratici le reti neurali sono strutture non lineari di dati statistici organizzate come strumenti di modellazione. Esse possono essere utilizzate per simulare relazioni complesse tra ingressi e uscite che altre funzioni analitiche non riescono a rappresentare. Per questo motivo vengono spesso utilizzate nella regressione come sostitute della regressione lineare. Una rete neurale artificiale riceve segnali esterni su uno strato di nodi (unità di elaborazione) di ingresso, ciascuno dei quali è collegato con numerosi nodi interni, organizzati in più livelli. Ogni nodo elabora i segnali ricevuti e trasmette il risultato ai nodi successivi. Normalmente una rete è formata da tre strati (fig.2.2):

1 Nel primo abbiamo gli *ingressi* (I). Questo strato si preoccupa di trattare gli ingressi in modo da adeguarli alle richieste dei neuroni.

2 Il secondo strato è quello *nascosto* (H, hidden). Questo strato si occupa dell'elaborazione vera e propria e può essere composto anche da più colonne di neuroni.

3 Il terzo strato è quello di *uscita* (O) e si preoccupa di raccogliere i risultati e adattarli alle richieste del blocco successivo della rete neurale.

Le reti neurali artificiali nell'apprendimento supervisionato sono basate sull'algoritmo di retropropagazione dell'errore (*backpropagation*). Esso permette di modificare i pesi delle connessioni in modo tale che si minimizzi una certa *funzione errore E*.

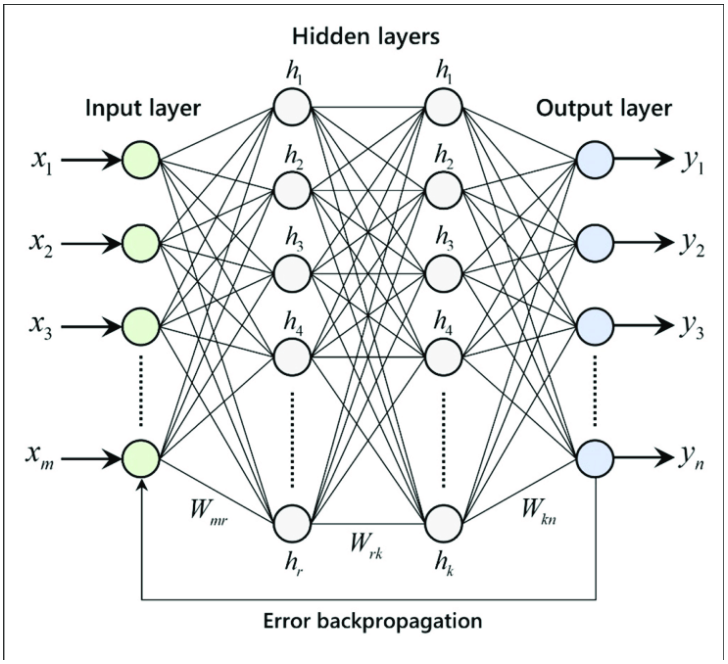


Figura 2.2: Rete neurale (fonte [14])

E(w) è una funzione dipendente dai pesi (che in generale variano nel tempo), per minimizzarla si può usare l'algoritmo della *discesa stocastica del gradiente (gradient descent)*, spiegato più in dettaglio nella sezione successiva 2.2.2.1.

L'algoritmo di backpropagation può essere diviso in due passi:

Forward pass: l'input dato alla rete è propagato al livello successivo e così via agli altri livelli (il flusso di informazioni si sposta in avanti, cioè forward). Si calcola dunque *E(w)*, l'errore commesso.

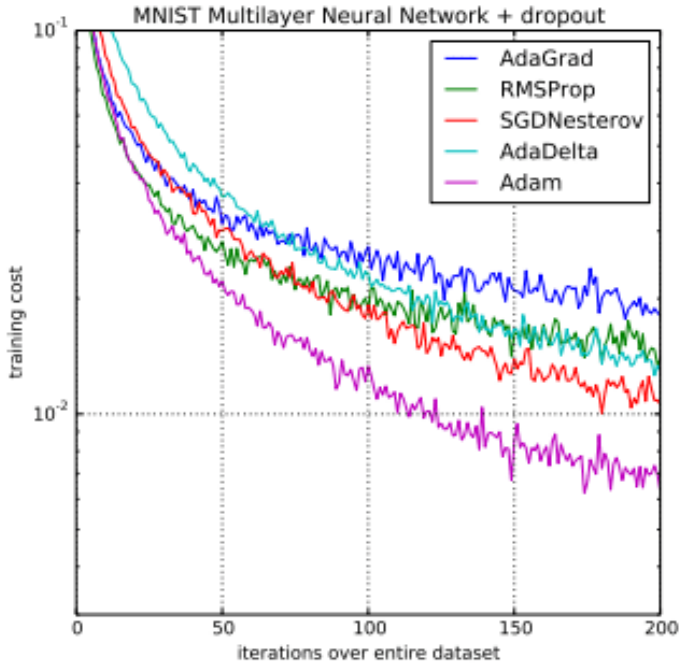
Backward pass: l'errore fatto dalla rete è propagato all'indietro (backward) e i pesi sono aggiornati in maniera appropriata. La backward pass richiede una `funzione di ottimizzazione` e una `funzione di loss (perdita)`, analizzate nelle sezioni successive 2.2.2.1 e 2.2.2.2

2.2.2.1 Ottimizzatore: Adam

Adam è una funzione di ottimizzazione, nota per essere una versione leggermente modificata dalla classica discesa stocastica del gradiente. La *discesa stocastica del gradiente* è un metodo iterativo per l'ottimizzazione di funzioni differenziabili, approssimazione stocastica del metodo di discesa del gradiente (GD). Specificamente, per una rete neurale, la stima stocastica è ottenuta dal gradiente della funzione di loss per un singolo punto (istanza) dei dati.
Gli autori [9] descrivono Adam come la combinazione dei vantaggi di altre due estensioni della discesa stocastica del gradiente. Nello specifico:

$$m_{t+1} = \beta_1 m_t + (1 - \beta_1)\nabla Q(w_t)$$
$$v_{t+1} = \beta_2 v_t + (1 - \beta_2)(\nabla Q(w_t))^2$$
$$\hat{m} = \frac{m_{t+1}}{1 - \beta_1^{t+1}}$$
$$\hat{v} = \frac{v_{t+1}}{1 - \beta_2^{t+1}}$$
$$w_{t+1} = w_t - \eta \frac{\hat{m}}{\sqrt{\hat{v}} + \epsilon}$$

(a) Equazioni Adam



(b) Risultati Adam (fonte [9])

Figura 2.3: Ottimizzatore Adam

- **Adaptive Gradient Algorithm (AdaGrad)** che mantiene un tasso di apprendimento per parametro che migliora le prestazioni su problemi con gradienti sparsi (ad esempio linguaggio naturale e problemi di visione artificiale).
- **Root Mean Square Propagation (RMSProp)** che mantiene a sua volta i tassi di apprendimento per parametro che vengono adattati in base alla media delle recenti grandezze dei gradienti per il peso (ad es. quanto velocemente sta cambiando). Ciò significa che l'algoritmo funziona bene su problemi online (che usano un solo elemento del dataset alla volta) e non stazionari (ad esempio

rumorosi).

Adam comprende i vantaggi sia di AdaGrad che di RMSProp. Invece di adattare i tassi di apprendimento dei parametri in base al primo momento medio (la media) come in RMSProp, Adam utilizza anche la media dei secondi momenti dei gradienti (la varianza non centrata). Nello specifico, l'algoritmo calcola una media mobile esponenziale del gradiente e del gradiente al quadrato e i parametri *beta1* e *beta2* controllano i tassi di decadimento di queste medie mobili. Il valore iniziale delle medie mobili e i valori *beta1* e *beta2* vicini a 1,0 (consigliato) comportano una distorsione delle stime momentanee verso lo zero. Questa distorsione viene superata calcolando le stime distorte prima di calcolare le stime corrette. (eq. 2.3a)

Nel documento originale [9], è stato dimostrato empiricamente che la convergenza soddisfa le aspettative dell'analisi teorica. Adam è stato applicato a vari algoritmi e su set di dati differenti risultando sempre particolarmente efficiente (fig. 2.3b). In particolare Adam risulta essere un'ottima scelta per ampi modelli e dataset (come nel caso di questo progetto).

2.2.2.2 Funzione di loss: MAE

La `funzione di loss` (perdita) è la misura della precisione con cui il modello è in grado di prevedere il risultato atteso. La funzione di loss prende due elementi come input: il valore di output del modello (la previsione dell'ER) e il valore reale (l'ER reale del post).

Un valore elevato della loss significa che il modello ha funzionato molto male, viceversa un valore basso della loss significa che il modello ha funzionato molto bene.

La selezione della funzione di loss è fondamentale per addestrare un modello correttamente. Esistono diverse funzioni, ognuna con delle proprie caratteristiche. In questo progetto è stata scelta la `MAE (Mean Absolute Error)`, ovvero l'errore medio assoluto.

MAE L'errore medio assoluto (fig. 2.4) è leggermente diverso nella definizione dall'MSE (*Mean Squared Error* ovvero errore quadratico medio), ma in modo interessante fornisce proprietà quasi esattamente opposte. Per calcolare il MAE, si prende la differenza tra le previsioni del modello e il dato reale, si applica il valore assoluto a tale differenza e quindi si calcola la media sull'intero set di dati. Il vantaggio e lo

$$\mathrm{MAE} = \frac{1}{n}\sum_{j=1}^{n} |y_j - \hat{y}_j|$$

Figura 2.4: Formula Mean Absolute Error (MAE)

svantaggio, a confronto con il MSE, sono:

Vantaggio il MAE copre direttamente lo svantaggio del MSE. Tutti gli errori infatti sono pesati sulla stessa scala lineare (dato l'utilizzo del valore assoluto). Pertanto, a differenza del MSE, non viene dato molto peso ai valori anomali e la funzione di loss fornisce una misura generica e uniforme delle prestazioni del modello.

Svantaggio se le previsioni anomale dovessero essere analizzate dal modello, allora il MAE non è altrettanto efficace. I grandi errori provenienti dai valori anomali finiscono per essere pesati esattamente come gli errori minori. Ciò potrebbe comportare che il modello sia ottimo per la maggior parte del tempo, ma ogni tanto faccia alcune previsioni molto scarse.

2.3 Approccio multimodale

Essendo un post di Instagram formato sia da testo (la caption) che da immagini, ho pensato di utilizzare un `approccio multimodale`. Per approccio multimodale si intende appunto l'utilizzo di dati di natura diversa. L'approccio multimodale è solitamente strutturato [11] [2] come rappresentato nel grafico 2.5

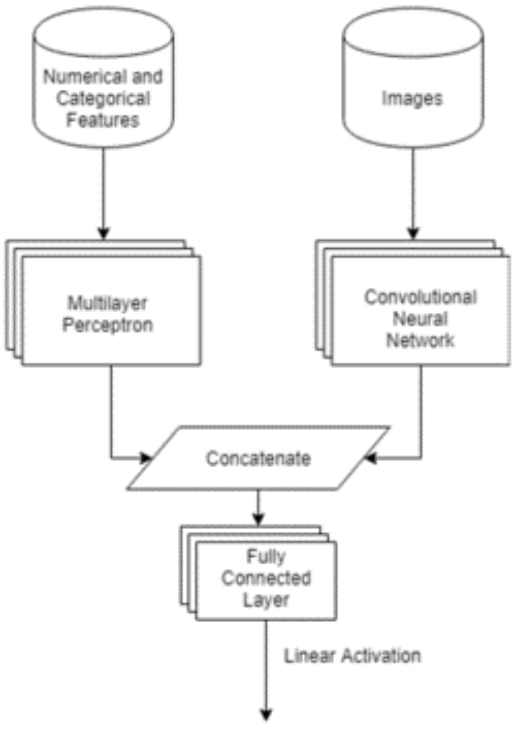


Figura 2.5: Approccio multimodale

Esso consiste nel concatenare in strati fully connected i dati elaborati con gli approcci monomodali, cioè con encoder di solo testo e solo immagine. In questo caso ho scelto `VGG16` per l'analisi visiva e `BERT` per l'analisi testuale.

2.3.1 Encoder di immagini: VGG16

L'encoder VGG, o VGGNet, che supporta 16 livelli (per questo motivo chiamato VGG16), è un modello di rete neurale convoluzionale proposto da A. Zisserman e K. Simonyan dell'Università di Oxford. Questi ricercatori hanno pubblicato il loro modello nel documento di ricerca intitolato "Very Deep Convolutional Networks for Large-Scale Image Recognition"[17].

Il modello VGG16 raggiunge quasi il 92,7% di precisione nei primi 5 test in ImageNet [3]. ImageNet è un set di dati composto da oltre 14 milioni di immagini appartenenti a quasi 1000 classi. Inoltre, è stato uno dei modelli più popolari presentati a ILSVRC-2014 [16]. Come accennato in precedenza, VGGNet-16 supporta 16 livelli e può classificare le immagini in 1000 categorie di oggetti, tra cui tastiera, animali, matita, mouse, ecc.

Architettura VGG16

La rete VGG è costruita con filtri convoluzionali molto piccoli. La VGG-16 in particolare è costituita da 13 strati convoluzionali e tre strati fully connected (fig.2.6).

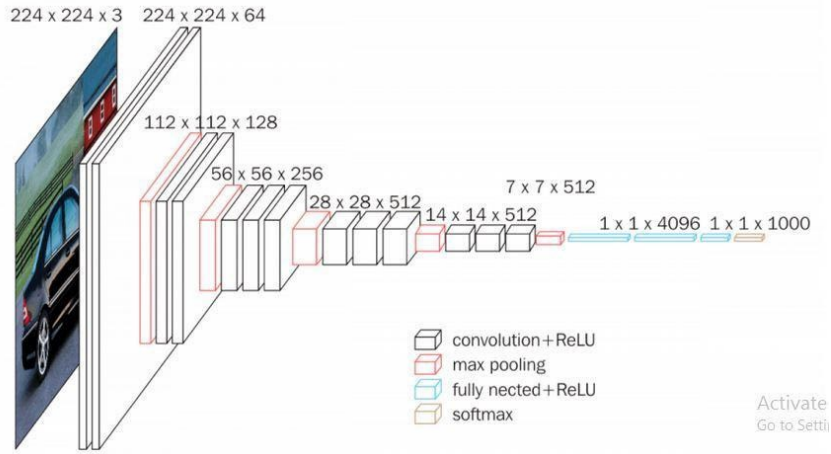


Figura 2.6: Struttura VGG16

L'architettura di VGG è così strutturata:

Input: VGGNet accetta una dimensione di input dell'immagine di 224×224. Per il concorso ImageNet, i creatori del modello hanno ritagliato la parte centrale in ciascuna immagine per mantenere coerenti le dimensioni di input (appunto, 224 × 224).

Strati convoluzionali : gli strati convoluzionali di VGG sfruttano un campo ricettivo minimo, ovvero 3 × 3, la dimensione più piccola possibile che cattura ancora su/giù e sinistra/destra. Inoltre, ci sono anche filtri di convoluzione 1×1 che fungono da trasformazione lineare dell'input. Questo è seguito da un'unità ReLU, che riduce i tempi di formazione. ReLU sta per funzione di attivazione

dell'unità lineare rettificata; è una funzione lineare a tratti che produrrà l'input se positivo; in caso contrario, l'output è zero. Il passo della convoluzione è fissato a 1 pixel per mantenere la risoluzione spaziale preservata dopo la convoluzione (il passo è il numero di spostamenti dei pixel sulla matrice di input).

Livelli nascosti : tutti i livelli nascosti nella rete VGG utilizzano ReLU. VGG di solito non sfrutta la normalizzazione della risposta locale (LRN) poiché aumenta il consumo di memoria e il tempo di addestramento. Inoltre, non apporta miglioramenti alla precisione complessiva.

Livelli fully connected : VGGNet ha tre livelli fully connected (completamente connessi). Dei tre livelli, i primi due hanno 4096 canali ciascuno e il terzo ha 1000 canali, 1 per ogni classe.

Risultati

VGG-16 è stata una delle architetture con le migliori prestazioni nella sfida ILSVRC 2014 [16]. È arrivata seconda nell'attività di classificazione con un errore del 7,32% (solo dietro a GoogLeNet con un errore di classificazione del 6,66%). È stata anche la vincitrice dell'attività di localizzazione con un errore del 25,32%.

2.3.2 Encoder di testo: BERT

Il mondo NLP (Natural Language Processing) è sempre più ricco e in via di sviluppo. Gli algoritmi di elaborazione del linguaggio naturale sono molti. In questo progetto ho deciso di utilizzare BERT. Le motivazioni sono molteplici: è un algoritmo piuttosto recente (2018), ottiene dei risultati ottimi anche con relativamente poche risorse, è bidirezionale e versatile.

BERT (Bidirectional Encoder Representations from Transformers) è stato di recente pubblicato dai ricercatori di Google AI Language. Ha suscitato scalpore nella comunità di Machine Learning presentando risultati all'avanguardia in un'ampia varietà di attività NLP, tra cui risposta alle domande (SQuAD v1.1), inferenza del linguaggio naturale (MNLI) e altre. L'innovazione tecnica chiave di BERT è l'applicazione della formazione bidirezionale di Transformer, un popolare modello di attention, alla modellazione del linguaggio. Ciò è in contrasto con gli sforzi precedenti che hanno esaminato una sequenza di testo da sinistra a destra o un addestramento combinato da sinistra a destra e da destra a sinistra. I risultati del documento [4] mostrano che un modello linguistico addestrato in modo bidirezionale può avere un senso più profondo del contesto e del flusso linguistico rispetto ai modelli linguistici unidirezionali. Nel documento [4], i ricercatori descrivono in dettaglio una nuova tecnica denominata Masked LM (MLM) che consente l'addestramento bidirezionale in modelli in cui era precedentemente impossibile.

Come funziona

BERT utilizza Transformer, un modello di attenzione che apprende le relazioni contestuali tra parole (o sotto-parole) in un testo. Nella sua forma originale, Transformer include due meccanismi separati: un codificatore che legge l'input di testo e un decodificatore che produce una previsione. Poiché l'obiettivo di BERT è generare un modello linguistico, è necessario solo il meccanismo del codificatore. Il funzionamento dettagliato di Transformer è descritto in un articolo di Google [20]. A differenza dei modelli direzionali, che leggono l'input di testo in sequenza (da sinistra a destra o da destra a sinistra), il codificatore Transformer legge l'intera sequenza di parole contemporaneamente. Pertanto è considerato bidirezionale, anche se sarebbe più corretto dire che non è direzionale. Questa caratteristica consente al modello di apprendere il contesto di una parola in base a tutto ciò che lo circonda (sinistra e destra della parola). L'input è una sequenza di token, che vengono prima incorporati nei vettori e quindi elaborati nella rete neurale. L'output è una sequenza di vettori di dimensione H, in cui ogni vettore corrisponde a un token di input con lo stesso indice. Quando si addestrano i modelli linguistici, è difficile definire un obiettivo di previsione. Molti modelli prevedono la parola successiva in una sequenza (ad esempio "Il bambino è tornato a casa da _"), un approccio direzionale che limita intrinsecamente l'apprendimento del contesto. Per superare questa sfida, BERT utilizza due strategie di formazione: *Masked LM* e *Next Sentence Prediction*. Quando si addestra BERT, Masked LM e Next Sentence Prediction vengono addestrati insieme, con l'obiettivo di minimizzare la funzione di loss combinata delle due strategie.

LM mascherato (MLM)

Prima di inserire sequenze di parole in BERT, il 15% delle parole in ciascuna sequenza viene sostituito con un token [MASK] (fig. 2.7). Questi token vengono pre-elaborati come segue: l'80% viene sostituito con un token "[MASK]", il 10% con una parola casuale e il 10% con la parola originale. Il modello tenta quindi di prevedere il valore originale delle parole mascherate, in base al contesto fornito dalle altre parole non mascherate nella sequenza. In termini tecnici, la previsione delle parole di output richiede:

- Aggiunta di un livello di classificazione sopra l'output del codificatore.
- Moltiplicazione dei vettori di output per la matrice di incorporamento, trasformandoli nella dimensione del vocabolario.
- Calcolo della probabilità di ogni parola nel vocabolario con softmax (funzione esponenziale normalizzata).

La funzione di loss di BERT prende in considerazione solo la previsione dei valori mascherati e ignora la previsione delle parole non mascherate. Di conseguenza, il modello converge più lentamente dei modelli direzionali, una caratteristica che è

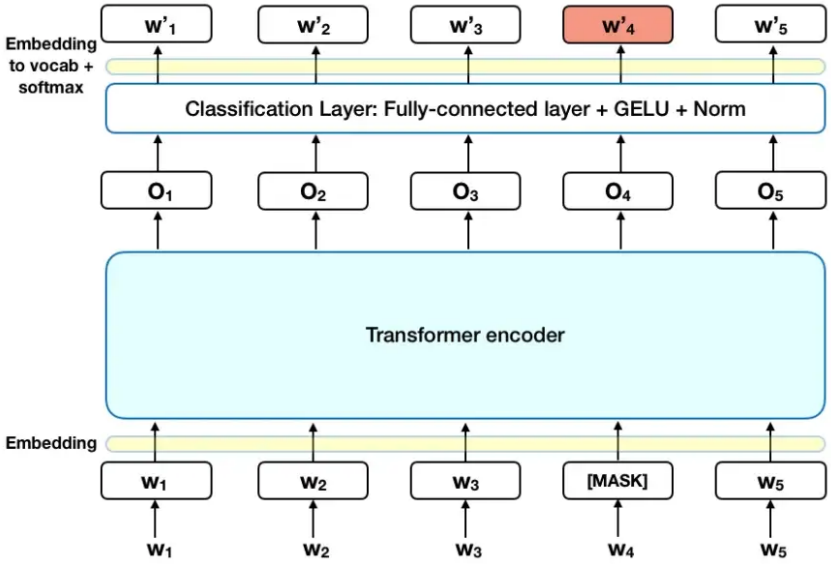


Figura 2.7: BERT: Masked LM (MLM) (fonte [4])

compensata dalla sua maggiore consapevolezza del contesto.

Next Sentence Prediction(NSP)

Nel processo di addestramento BERT riceve coppie di frasi come input e impara a prevedere se la seconda frase nella coppia è la frase successiva nel documento originale (fig. 2.8). Durante la formazione, il 50% degli input è una coppia in cui la seconda frase è la frase successiva nel documento originale, mentre nel restante 50% viene scelta come seconda frase una frase casuale. Si assume che la frase casuale sia disconnessa dalla prima frase. Per aiutare il modello a distinguere tra le due frasi durante l'addestramento, l'input viene elaborato nel modo seguente prima di entrare nel modello:

1 Un token [CLS] viene inserito all'inizio della prima frase e un token [SEP] viene inserito alla fine di ogni frase.

2 A ciascun token viene aggiunta una *sentence embedding* che indica la frase A o la frase B. La sentence embedding è analoga alla word embedding, ma viene fatta con le frasi intere. Per *word embedding* s'intende la costruzione di uno spazio vettoriale in cui i vettori delle parole sono più vicini se le parole occorrono negli stessi contesti linguistici, cioè se sono riconosciute come semanticamente più simili.

3 Una sentence embedding viene aggiunta a ciascun token per indicare la sua posizione nella sequenza.

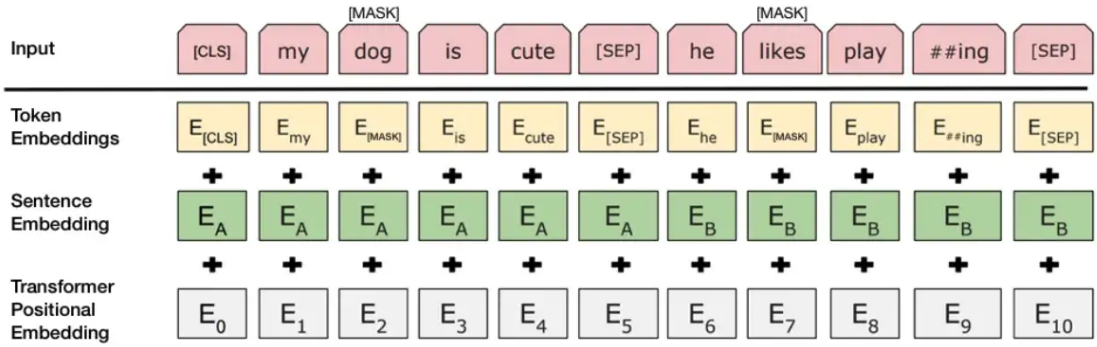


Figura 2.8: BERT: Next Sentence Prediction (NSP) (fonte [4])

Per prevedere se la seconda frase è effettivamente collegata alla prima, vengono eseguiti i seguenti passaggi:

1 L'intera sequenza di input passa attraverso il modello Transformer.

2 L'output del token [CLS] viene trasformato in un vettore a forma di 2×1, utilizzando un semplice livello di classificazione (matrici apprese di pesi e distorsioni).

3 Calcolo della probabilità di IsNextSequence con softmax (funzione esponenziale normalizzata).

Capitolo 3

Implementazione

La fase implementativa è stata realizzata con il linguaggio `Python`, sfruttando le librerie `PyTorch`. In aggiunta, per gestire con maggior comodità e telematicamente le varie fasi (training, testing e prediction) ho utilizzato la piattaforma `Weights & Biases` (`https://wandb.ai`). L'elaborazione del modello è stata eseguita nell'IDE `PyCharm`, collegandomi in remoto (tramite SSH) al server VRAI della Facoltà di Ingegneria dell'Informazione. Il server ha come caratteristiche hardware: Intel® Xeon® 4110 (2,10 GHz), 224 GB RAM, svariati TB di hard disk e 4 NVIDIA RTX 2080Ti.

3.1 Dati

3.1.1 Scraping o dataset esistente?

Il requisito primario per qualsiasi task di data analysis sono i dati (come indicato nella sezione 2.2). In questo caso i dati sono post di Instagram. Le strade da scegliere possono essere due: estrarre i dati direttamente da Instagram con l'utilizzo di API e scraper oppure utilizzare dei dataset già esistenti. Lo scopo di questo progetto è di stimare l'engagement di un profilo qualsiasi, non sono quindi necessari dei dati specifici di alcune categorie di utenti. Considerando poi i limiti piuttosto stringenti sull'utilizzo delle API di Instagram [6], ho deciso di usare un dataset preesistente.

3.1.2 Dataset utilizzato

Il dataset usato è reperibile a questo link: `https://sites.google.com/site/sbkimcv/dataset/instagram-influencer-dataset`.

Il dataset è stato originariamente raccolto con lo scopo indicato nell'articolo [8] ovvero classificare varie categorie di influencer utilizzando algoritmi di analisi immagini. Il dataset è composto da 10.180.500 post di Instagram prelevati da 33.935 account (300 post per ognuno). E' un dataset molto grande e ricco di informazioni, per questo ho deciso di utilizzarlo (anche se con dei limiti come indicato successivamente). Il dataset è suddiviso in due cartelle (IMG e JSON) che contengono rispettivamente le immagini (esempio 3.1) e i file JSON con tutte le informazioni dei vari post. Le informazioni specifiche di ogni influencer sono inserite nel file `influencers.txt` che

(a) Beauty (b) Family (c) Fashion (d) Fitness

Figura 3.1: Alcune immagini del dataset

è strutturato come in tabella 3.1. E' presente anche un file di mapping delle immagini

Tabella 3.1: Struttura `influencers.txt`

Username	Category	Followers	Followees	Posts
makeupbynvs	beauty	1432	1089	363
jaquelinevandoski	beauty	137600	548	569
...	...	...	...	...

con la struttura 3.2.

Tabella 3.2: Struttura `JSON-Image_files_mapping.txt`

influencer_name	JSON_PostMetadata_file_name	Image_file_name
00_rocketgirl	1188140434601337485.info	['1188140434601337485.jpg']
00_rocketgirl	1195378513372308151.info	['1195378513372308151.jpg']
...	...	...

3.1.3 Elaborazione dataset e feature selection

La formula per il calcolo dell'engagement di un post Instagram utilizzata è stata già mostrata (eq. 2.1). Nel dataset ci sono quindi delle informazioni sui post superflue. Per elaborare al meglio i dati con le librerie Python utilizzate (PyTorch), conviene convertire il tutto in un file CSV unico. Ho ideato a proposito questo script 5.1 che filtra anche le informazioni non necessarie. Il file CSV di output è strutturato come in tabella 3.3, il separatore tra ogni item è il carattere ",".

Tabella 3.3: Struttura file CSV

post_id	**owner_id**	**owner_username**	**owner_followers**	**owner_followees**	**like_count**	**comment_count**	**caption**	**file_names**
1932799488079063873	503542812	teresa_bass	184992.0	465.0	6375	76	Hoy todo al beige @marcosplazaretamosa	['1932799488079063873.jpg']
1563132729406836539	207174894	lalicler	211197.0	683.0	7242	59	No filters needed for natural beauty...	['1563132729406836539.jpg']
1660955935617541727	1384142525	cteatsout	68337.0	4163.0	1575	27	These vanilla scones by @wavehillbreads are calling our name...	['1660955935617541727.jpg']
...	...	...	...	...	...	...	...	...

Il dataset è stato fortemente ridimensionato, nonostante le ottime prestazioni del server su cui gli algoritmi sono stati eseguiti. Per estrapolare i dati ed organizzarli nel file CSV (eseguendo quindi lo script 5.1) occorrevano svariati giorni e perciò nella fase di training sarebbe risultato ancora più critico l'utilizzo del dataset completo. Il numero di post è stato ridotto a 200.000.

Per un ottimale processo di addestramento e valutazione del modello, è necessario utilizzare nelle fasi di `training`, `validation` e `testing` insiemi disgiunti di pattern di esempio.

- **Training set**: esempi da fornire al modello per l'ottimizzazione dei parametri per il calcolo della funzione obiettivo. Avevo inizialmente deciso di generare il training set direttamente dal dataset senza ulteriori modifiche. Facendo però delle previsioni (3.2.3) ho notato che il numero di follower (in particolare numeri molto grandi) influenza pesantemente l'engagement rate. Ho deciso quindi di generare un altro training set filtrando però il numero di follower dei corrispettivi account scartando post di utenti con molti seguaci.
- **Validation set**: esempi da fornire al modello per valutare la bontà dei parametri appresi. Anche in questo caso, per lo stesso motivo spiegato nel punto precedente ho deciso di generare due validation set diversi.
- **Test set**: esempi da fornire al modello per valutare le prestazioni finali del sistema e il grado di generalizzazione raggiunto. Anche i test set sono due, per la giustificazione già indicata.

I tre set di pattern dovrebbero essere disgiunti tra loro in modo da evitare sovrastime delle prestazioni e limitare l'*overfitting*. Questo perché se il validation o il test set sono sottoinsiemi del training set, significa che gli esempi forniti in fase di valutazione sono già stati visti ed "imparati" dal modello che quindi fornirà dei buoni risultati,

ma non sarà in grado di gestire in modo ottimale dati estranei nella fase di prediction. Se i set di valutazione contengono invece esempi mai visti prima dal modello, una buona risposta dello stesso denoterà un buon livello di generalizzazione raggiunto in fase di addestramento.

Nel mio caso specifico, il dataset è stato diviso in 3 set con le seguenti percentuali: 70% training set, 10% validation set e 20% testing set, cosicché ognuna delle parti non comprendesse post delle altre.

3.1.4 Tokenizzazione e data cleaning per BERT

Come indicato nella sezione 2.3.2, BERT richiede la sostituzione di alcune parole con dei token. La parte di codice 5.2 si occupa di ciò. In più, la caption di ogni post è stata pulita eliminando caratteri speciali o anomali, com'è buona norma fare in tutti gli algoritmi di NLP, così da migliorare l'apprendimento (codice 5.3).

3.2 Modello predittivo

Questa è la terza parte della metodologia indicata in fig. 2.1. Nel capitolo 2 sono già state illustrate e spiegate le tecnologie e gli algoritmi. Riassumendo velocemente, il modello utilizza gli encoder VGG16 e BERT, concatenandone gli output su più livelli, sfruttando l'approccio in fig. 2.5. Sono stati utilizzati VGG16 e BERT "default", quindi con i vari parametri (numero livelli, dimensione di input e output, ecc.) lasciati invariati, come descritti nelle sezioni 2.3.2 e 2.3.1. E' importante indicare che per ogni post viene analizzata solo la prima immagine. Se un post quindi contiene più immagini, non vengono considerate. Per far ciò andrebbe implementato anche un *approccio multi-view* (4.2) oltre a quello multimodale. Il modello completo, con tutti i livelli, è consultabile nel codice 5.4.

La parte più importante del modello sono gli ultimi 8 livelli. Essi si occupano della `concatenazione`, sono cioè il fulcro dell'approccio multimodale, nonché la particolarità di tutto il progetto. In quel punto le informazioni derivanti dai singoli encoder vengono unite ed analizzate come un'unica grande informazione, contenente naturalmente più dettagli rispetto alle singole. Da notare che l'output finale è di dimensione 1 (*out_features*) poiché si tratta di un task di regressione. Nel caso della classificazione la dimensione sarebbe stata n come il numero delle classi.

3.2.1 Training

Durante la fase di addestramento, vengono forniti i pattern contenuti nel training set, in modo che il modello possa adattare i propri parametri alla ricerca della funzione obiettivo ottimale. Non è però sufficiente un unico passaggio per ottenere un risultato eccellente, ma è necessario eseguire più cicli di elaborazione del training set ed adattamento dei parametri. Tra l'altro, per i limiti imposti dalla memoria e dalla potenza di calcolo disponibile per queste operazioni, non è solitamente possibile

elaborare l'intero training set in un colpo solo. Quindi ci sono dei parametri che entrano in gioco nel definire come deve svolgersi l'elaborazione del training set durante la fase di addestramento.

3.2.1.1 Batch size e epoch

Quando l'intero training set è stato sottoposto al modello, allora si ha una `epoch`. La fase di addestramento solitamente si esaurisce dopo diverse epoche (possono servirne anche decine di migliaia). In questo progetto la funzione di loss già dopo alcune decine di epoche risultava essere buona, quindi dopo aver effettuato diverse prove, il numero di epoche per ogni fase di addestramento che ho scelto è 50.

Come detto sopra, il training set potrebbe essere troppo grande per essere elaborato tutto in una sola volta. Quindi si può suddividere il training set in sottogruppi uniformi, chiamati `batch`. Il numero di esempi contenuti in ogni batch è chiamata `batch size`. Da cui deriva la definizione di *iterazione* che corrisponde al numero di batch necessari a completare una epoch.

Nel caso specifico di questo progetto il training set formato da 140.000 post (il 70% del dataset, come spiegato in 3.1.3) e quindi troppo grande per essere elaborato in una volta sola, è stato suddiviso in batch con 32 esempi ciascuno. 1 epoch è formata da 4375 iterazioni.

Il numero di epoche e la batch size influiscono sulla velocità di addestramento di un modello, ma anche sul suo modo di perfezionarsi. Come capita spesso nell'ambito dell'intelligenza artificiale e del Deep Learning, non ci sono regole scolpite nella roccia ma regole empiriche che consigliano di non usare batch size troppo grandi o troppo piccoli.

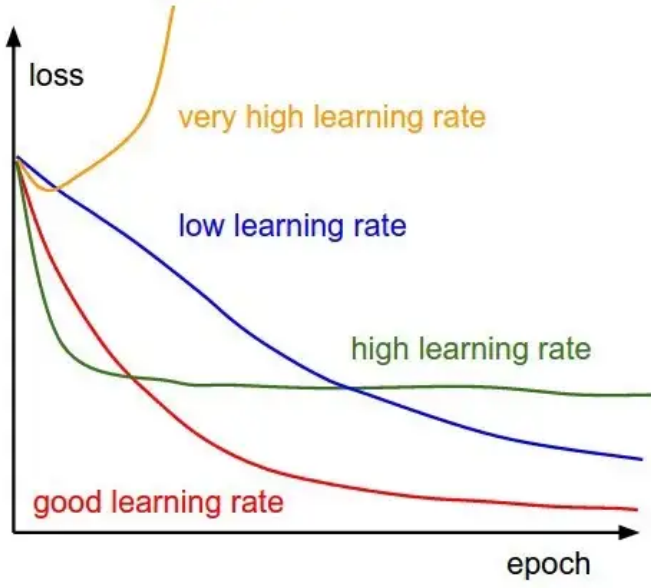
Per esempio, se la batch size è troppo grande si potrebbe avere un problema di esaurimento della memoria o una tendenza all'overfitting più accentuata.

Batch size tipici sono 32, 64 o 128 (solitamente potenze di 2 per motivi di allocazione della memoria). In effetti, come già detto poco sopra, la batch size che ho scelto è 32; un numero maggiore avrebbe dato errore di allocazione memoria della scheda video ed un numero minore non avrebbe sfruttato al massimo le potenzialità hardware e della suddivisione in batch.

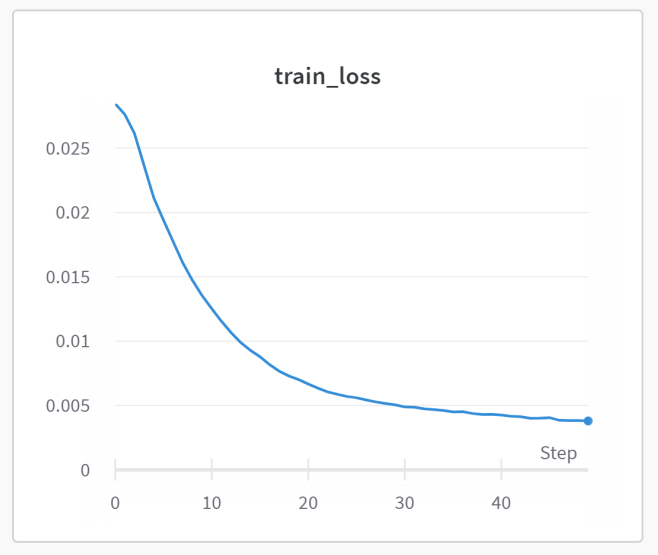
Learning rate dell'ottimizzatore (Adam)

La scelta del learning rate di Adam è stata effettuata seguendo le indicazioni di questo articolo [15], confrontando poi il mio grafico di training (fig. 3.2b) con quelli di riferimento (fig. 3.2a).

La fig. 3.2b si avvicina molto alla forma del grafico con un buon learning rate indicato in fig. 3.2, quindi la scelta fatta (ovvero 0.0001) risulta essere piuttosto corretta.



(a) Esempi di loss variando Learning Rate



(b) Loss del mio modello

Figura 3.2: Scelta del Learning Rate di Adam

3.2.1.2 Validation

Nel training il modello ha imparato le relazioni tra input e output. In questa fase il modello fornirà una y (l'engagement predetto) basata sui dati che gli vengono passati (caption e immagine).

A questo punto si dispone di una *y_predetta* (l'engagement predetto) e una *y_reale* già conosciuta (l'engagement reale). La `validation` consiste proprio nel confrontare le *y_predette* con le *y_reali* per vedere quanto il modello riesce a prevedere con una buona approssimazione la variabile di output. Si chiama validation set perché

si occupa di validare i risultati ottenuti nel training set. Se le performance fossero scarse, bisognerebbe modificare i parametri del modello e ripartire con il training, finché il risultato sulla validation non sarà soddisfacente.

In questo progetto la fase di validation viene eseguita al termine di ogni epoch, durante la fase di training, ma potrebbe anche essere eseguita successivamente.

3.2.1.3 Due modelli diversi

La fase di training è fortemente influenzata dal training set. Come già spiegato in 3.1.3, analizzando alcuni risultati ottenuti nelle predizioni, ho deciso di utilizzare due training set. Il primo è semplicemente una parte del dataset, senza ulteriori modifiche. Il secondo invece è formato da post pubblicati da account con più di 10 follower (escludendo quindi potenziali fake) e meno di 50.000 follower. La struttura interna del modello rimane la stessa (5.4), però essendo il training set diverso, si ottengono pesi diversi e quindi si avranno previsioni diverse, ottenendo di fatto due modelli diversi. Le previsioni effettuate su account con pochi follower sono in questo modo di gran lunga migliorate (come indicato in 3.2.3).

3.2.2 Testing

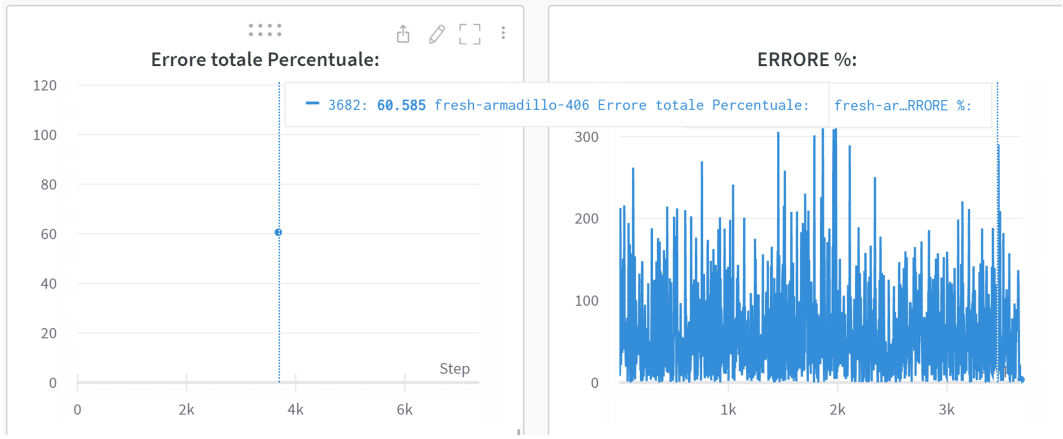
Questa fase è un'ulteriore verifica sulla precisione del modello. Infatti il modello viene testato su altre osservazioni che non ha mai visto (testing set). Anche in questo step si conosce il valore reale del dato da prevedere (l'engagement), che viene appunto confrontato con la previsione per verificarne l'accuratezza.

Nella fase di testing ho deciso di utilizzare una funzione "artigianale" molto semplice, senza usare delle funzioni preimpostate.

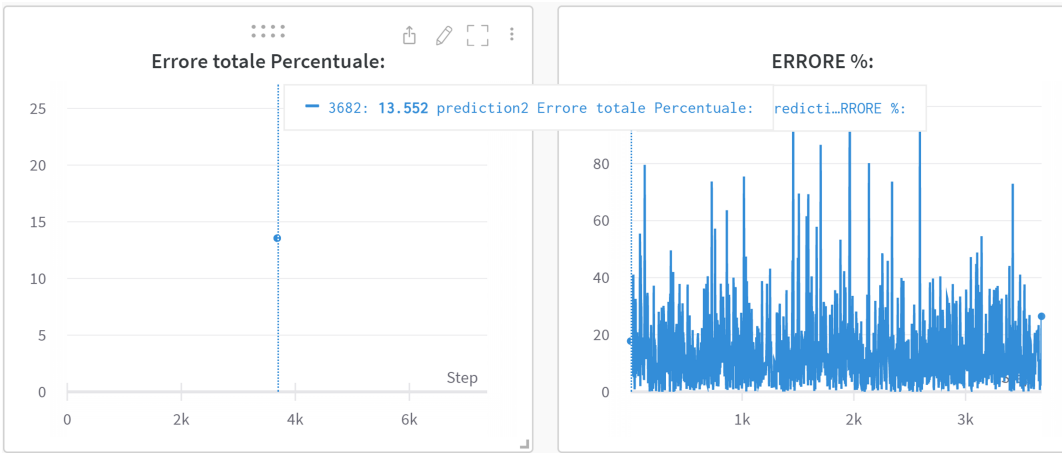
$$\boldsymbol{Errore}\% = \frac{|stima - valore\_reale|}{valore\_reale} * 100 \tag{3.1}$$

Ho voluto riportare due esempi significativi di testing (fig. 3.3), che mi hanno condotto nella scelta di usare due training set (e quindi ottenere due modelli diversi, come già spiegato in più punti precedenti della tesi). Ho voluto verificare se la mia intuizione fosse giusta, ovvero che gli account con un numero elevato di follower hanno un engagement rate molto diverso e diversificato rispetto agli utenti "comuni".

Nel primo grafico (fig. 3.3a) ho testato il modello trainato su un set di dati generico con un test set di dati filtrati per numero di follower (da 10 a 50.000). Nel secondo (fig. 3.3b) invece ho testato il modello trainato su un set di dati filtrati per numero di follower con un test set di dati anch'essi filtrati per lo steso numero di follower (da 10 a 50.000). Come si può notare nell'immagine il primo grafico (fig. 3.3a) presenta una percentuale media di errore molto alta rispetto alla seconda immagine (fig. 3.3b). Inoltre, le predizioni sono molto diverse tra loro, alcune ottime, altre decisamente errate (con errori fino al 300%). Ciò dimostra che effettivamente il numero di follower influisce sull'engagement rate, come intuito. La scelta quindi



(a) Testing con errore % alto.



(b) Testing con errore % accettabile.

Figura 3.3: Esempi di testing

di utilizzare due training set diversi è risultata corretta. Anche la sezione 3.2.3 lo dimostra. Il range di follower stabilito (tra 10 e 50.000) è stato arbitrario. Sarebbe curioso generare altri training set con diversi range in modo da individuare fasce più specifiche di utenti in base ai loro follower, così da compiere previsioni ancora più accurate.

3.2.3 Prediction

Questa è la fase finale, dove fornendo la caption e l'immagine completamente estranei ad ogni set di dato già usato, si riceve la stima dell'engagement. Va notato con attenzione che questa fase è fortemente diversa da quella di validation e testing (anche se possono sembrare simili). Infatti nella fase di testing e di validation il valore reale si conosce e l'obiettivo è confrontarlo con quello stimato. In questa fase **non** si conosce a priori il valore reale. Nelle mie predizioni ho comunque deciso di utilizzare post di cui già conoscevo l'engagement rate per verificarne la correttezza. Le predizioni sono state fatte utilizzando post del mio profilo Instagram (*@alex-paul_official*). Come indicato in precedenza, le prime previsioni effettuate davano valori con un alto tasso di errore. Poi però l'utilizzo del modello trainato su set con un numero di follower limitato ha fornito previsioni con accuratezze decisamente più accettabili.

```
Engagement stimato: [tensor(0.0540)] Quello vero: [tensor(0.1358)]ERORRE PERECENTUALE: 60.2165563378891
Engagement stimato: [tensor(0.1305)] Quello vero: [tensor(0.1358)]ERORRE PERECENTUALE: 3.905550960046867
```

Figura 3.4: Esempio di predizione

Nel caso specifico in fig. 3.4 utilizzando il modello trainato sul dataset generico la previsione ha un'errore % molto alto (60%) e quindi un'accuratezza di circa il 40%. Nel secondo caso invece l'accuratezza della previsione supera il 90%.

Capitolo 4

Conclusioni

4.1 Risultati

L'obiettivo di prevedere l'engagement con una buona precisione è stato raggiunto (accuratezza del 96% circa, fig. 3.2.3). Non solo, tramite questo progetto si dimostra anche come il numero di follower influisca fortemente sull'engagement rate. Gli account con più follower tendono ad avere un engagement rate in genere più basso, ma soprattutto i criteri che incidono sullo stesso sono diversi rispetto ai profili con meno seguaci.

4.2 Sviluppi futuri

Seppure l'obiettivo sia stato raggiunto, ottenendo anche delle conclusioni interessanti non previste a priori (la rilevanza del numero di follower), si potrebbero effettuare molte più analisi ed estensioni del progetto. Ad esempio:

- **Analizzare quali elementi di una foto influiscono sull'engagement**. In questo progetto infatti viene solamente stimato l'engagement rate, senza tener conto di quali parametri della foto influiscano sullo stesso (filtri, luminosità, soggetti raffigurati, ecc.).
- **Analizzare quali parole/argomenti nei post creano maggiore interazione**. L'analogo del punto precedente applicato alla caption.
- **Utilizzare più informazioni dei post (data e ora di pubblicazione, presenza di filtri, location, ecc.)**. Ogni post di Instagram è caratterizzato da moltissime informazioni che potrebbero influenzare l'engagement che però in questo caso non sono state considerate.
- **Utilizzare un approccio *multi-view* per i post con più immagini**. Come già esposto in 3.2, il modello elaborato non tiene conto delle altre immagini di un post, che però anch'esse potrebbero influire sul coinvolgimento di un post. Nell'articolo [19] sono presenti alcune informazioni interessanti su come potrebbe essere implementato questo punto.
- ...

Capitolo 5

Codice

```
# Le librerie usate
import json
import pandas as pd
import os

# Innanzitutto importo il file degli influencers
fileFollowers = 'Redownloaded/influencers.txt'
df = pd.read_csv(fileFollowers, sep="\t")

# Poi importo il file di mapping
fileImageMapping = 'Redownloaded/JSON-Image_files_mapping.txt'
print('Inizio lettura Image Mapping')
dfImageMapping = pd.read_csv(fileImageMapping, sep="\t")
print('Fine lettura Image Mapping')

# Variabile temporanea per la creazione DataFrame
rows = []

# Conta il numero di file totali (serve solo per scrivere a che punto si trova lo script)
numeroFile = 0
for filename in os.scandir('Redownloaded/JSON/info'):
if filename.is_file():
    numeroFile += 1

print ('Ho trovato ' + str(numeroFile) + ' files')

# Varibili che contano il progresso
i = 0
saltatiPath = 0
saltatiInfoFollow = 0
saltatiCaption = 0
saltatiJSON = 0

# Apre tutti i file info
for filename in os.scandir('Redownloaded/JSON/info'):
    # E' stato limitato a 200.000 post per le motivazioni spiegate
    if i == 200000:
        break
    print("\r", 'File ' + str(i) + ' su ' + str(numeroFile) + ' (' + str(saltatiPath) + ' pathSaltati, ' + str(saltatiJSON) + ' saltatiJSON, ' + str(saltatiInfoFollow) + ' saltatiInfoFollow, ' + str(saltatiCaption) + ' senzaCaption)', end="")
    if filename.is_file():
        try:
            with open(filename.path) as f:
                d = json.load(f)
            infoFile = pd.json_normalize(d)
        except json.decoder.JSONDecodeError:
            saltatiJSON += 1
            continue

# Vengono estratte le info necessarie. Se alcune mancano, il post viene scartato
    if (df[df['Username'] == infoFile['owner.username'].values[0]]['#Followers'].values.size > 0):
        followers = df[df['Username'] == infoFile['owner.username'].values[0]]['#Followers'].values[0]
    else:
        saltatiInfoFollow += 0.5
        followers = -1
    if (df[df['Username'] == infoFile['owner.username'].values[0]]['#Followees'].values.size > 0):
        followees = df[df['Username'] == infoFile['owner.username'].values[0]]['#Followees'].values[0]
```

```
    else:
        saltatiInfoFollow += 0.5
        followees = -1
    if (dfImageMapping[dfImageMapping['JSON_PostMetadata_file_name'] == infoFile['id'].
    values[0] + '.info']['Image_file_name'].values.size > 0):
        imageFileName = dfImageMapping[dfImageMapping['JSON_PostMetadata_file_name'] ==
    infoFile['id'].values[0] + '.info']['Image_file_name'].values[0]
    else:
        continue
        saltatiPath += 1
    # Questo controllo viene fatto per il conteggio dei commenti
    # In alcuni post (dei repost) il numero dei commenti si trova in una chiave diversa
    del file JSON
    if 'edge_media_to_parent_comment.count' in infoFile.keys():
        commentCount = infoFile['edge_media_to_parent_comment.count'].values[0]
    elif 'edge_media_to_comment.count' in infoFile.keys():
        commentCount = infoFile['edge_media_to_comment.count'].values[0]
    else:
        print("Manca Comment ")
        print(filename.path)
        continue
    # Innanzitutto viene controllato se la caption esiste
    if (len(infoFile['edge_media_to_caption.edges'].values[0]) > 0):

        # Ho provato a parsarla usando il JSON ma dava problemi in caso di ""
        # L'unico metodo e' troncare la stringa
        caption = str(infoFile['edge_media_to_caption.edges'].values[0])[20:-4]
    else:
        saltatiCaption += 1
        caption = 'no_text'
    data = [infoFile['id'].values[0], infoFile['owner.id'].values[0],
            infoFile['owner.username'].values[0],
            followers,
            followees,
            infoFile['edge_media_preview_like.count'].values[0],
            commentCount,
            caption,
            imageFileName]
    rows.append(data)
    f.close()
    i += 1

with open('RESULTS.txt', 'w') as f:
    f.write('File processati: ' + str(numeroFile) + ' (' + str(saltatiPath) + '
    pathStimati, ' + str(saltatiInfoFollow) + ' saltatiInfoFollow, ' + str(saltatiCaption
    ) + ' senzaCaption)')

print('\n')

# Viene creato il DataFrame alla fine e poi viene scritto in un file CSV unico
finalDataFrame = pd.DataFrame(rows, columns=['post_id', 'owner_id', 'owner_username', '
    owner_followers', 'owner_followees', 'like_count', 'comment_count', 'caption', '
    file_names'])
csvData = finalDataFrame.to_csv('dataset.csv', index=False)
print('File CSV creato con successo!')
```

Codice 5.1: Creazione file CSV

```
tokens = self.tokenizer.tokenize(preprocess(sentence), max_length=512, truncation=True)
tokens = ['[CLS]'] + tokens + ['[SEP]']

if len(tokens) < self.maxlen:
    tokens = tokens + ['[PAD]' for _ in range(self.maxlen - len(tokens))]
else:
    tokens = tokens[:self.maxlen - 1] + ['[SEP]']

input_ids = self.tokenizer.convert_tokens_to_ids(tokens)
input_ids = torch.tensor(input_ids)
```

Codice 5.2: Tokenizzazione caption per BERT

```
# Rimuove lo spazio tra numeri
def remove_space_between_numbers(text):
    text = re.sub(r'(\d)\s+(\d)', r'\1\2', text)
    return text

# Filtra le mail
def filter_emails(text):
    pattern = r'(?:(?!.*?[.]{2})[a-zA-Z0-9](?:[a-zA-Z0-9.+!%-]{1,64}|)|\"[a-zA-Z0-9.+!%
    -]{1,64}\")@[a-zA-Z0-9][a-zA-Z0-9.-]+(.[a-z]{2,}|.[0-9]{1,})'
    text = re.sub(pattern, '', text)
    return text

# Filtra siti web
def filter_websites(text):
    pattern = r'(http\:\/\/|https\:\/\/)?([a-z0-9][a-z0-9\-]*\.)+[a-z][a-z\-]*'
    text = re.sub(pattern, '', text)
    return text

# Filtra numeri di telefono
def filter_phone_numbers(text):
    pattern = r'(?:(?:\+|00)33[\s.-]{0,3}(?:\(0\)[\s.-]{0,3})?|0)[1-9](?:(?:[\s.-]?\d{2})
    {4}|\d{2}(?:[\s.-]?\d{3}){2})|(\d{2}[ ]\d{2}[ ]\d{3}[ ]\d{3})'
    text = re.sub(pattern, '', text)
    return text

# Funzione di pulizia che unisce tutte le precedenti
def clean_text(text):
    text = text.lower()
    text = text.replace(u'\xa0', u' ')
    text = filter_phone_numbers(text)
    text = filter_emails(text)
    text = filter_websites(text)
    text = remove_space_between_numbers(text)
    return text

# Funzione main, con ulteriori pulizie
def preprocess(sentence):
    sentence=str(sentence)
    sentence = sentence.lower()
    sentence=sentence.replace('{html}',"")
    cleanr = re.compile('<.*?>')
    cleantext = re.sub(cleanr, '', sentence)
    rem_url=re.sub(r'http\S+', '',cleantext)
    rem_num = re.sub('[0-9]+', '', rem_url)
    tokenizer = RegexpTokenizer(r'\w+')
    tokens = tokenizer.tokenize(rem_num)
    filtered_words = [w for w in tokens if len(w) > 2 if not w in stopwords.words('english
    ')]
    stem_words=[stemmer.stem(w) for w in filtered_words]
    lemma_words=[lemmatizer.lemmatize(w) for w in stem_words]
    return " ".join(lemma_words)
```

Codice 5.3: Pulizia testo caption

```
MixModel(
# BERT con L=12, H=768, A=12
 (bert): BertForSentimentClassification(
   (bert): BertModel(
     (embeddings): BertEmbeddings(
       (word_embeddings): Embedding(30522, 768, padding_idx=0)
       (position_embeddings): Embedding(512, 768)
       (token_type_embeddings): Embedding(2, 768)
       (LayerNorm): LayerNorm((768,), eps=1e-12, elementwise_affine=True)
       (dropout): Dropout(p=0.1, inplace=False)
     )
     (encoder): BertEncoder(
       (layer): ModuleList(
         (0): BertLayer(
           (attention): BertAttention(
             (self): BertSelfAttention(
               (query): Linear(in_features=768, out_features=768, bias=True)
               (key): Linear(in_features=768, out_features=768, bias=True)
               (value): Linear(in_features=768, out_features=768, bias=True)
               (dropout): Dropout(p=0.1, inplace=False)
             )
             (output): BertSelfOutput(
               (dense): Linear(in_features=768, out_features=768, bias=True)
               (LayerNorm): LayerNorm((768,), eps=1e-12, elementwise_affine=True)
               (dropout): Dropout(p=0.1, inplace=False)
```

```
            )
          )
          (intermediate): BertIntermediate(
            (dense): Linear(in_features=768, out_features=3072, bias=True)
            (intermediate_act_fn): GELUActivation()
          )
          (output): BertOutput(
            (dense): Linear(in_features=3072, out_features=768, bias=True)
            (LayerNorm): LayerNorm((768,), eps=1e-12, elementwise_affine=True)
            (dropout): Dropout(p=0.1, inplace=False)
          )
        )
    # ...
    # Sono stati omessi gli altri 11 livelli uguali...
    (pooler): BertPooler(
      (dense): Linear(in_features=768, out_features=768, bias=True)
      (activation): Tanh()
    )
  )
  (cls_layer): Linear(in_features=768, out_features=1, bias=True)
 )

# VGG16
 (vgg): VGG16FT(
  (vgg): VGG(
    (features): Sequential(
      (0): Conv2d(3, 64, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
      (1): ReLU(inplace=True)
      (2): Conv2d(64, 64, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
      (3): ReLU(inplace=True)
      (4): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False)
      (5): Conv2d(64, 128, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
      (6): ReLU(inplace=True)
      (7): Conv2d(128, 128, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
      (8): ReLU(inplace=True)
      (9): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False)
      (10): Conv2d(128, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
      (11): ReLU(inplace=True)
      (12): Conv2d(256, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
      (13): ReLU(inplace=True)
      (14): Conv2d(256, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
      (15): ReLU(inplace=True)
      (16): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False)
      (17): Conv2d(256, 512, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
      (18): ReLU(inplace=True)
      (19): Conv2d(512, 512, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
      (20): ReLU(inplace=True)
      (21): Conv2d(512, 512, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
      (22): ReLU(inplace=True)
      (23): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False)
      (24): Conv2d(512, 512, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
      (25): ReLU(inplace=True)
      (26): Conv2d(512, 512, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
      (27): ReLU(inplace=True)
      (28): Conv2d(512, 512, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
      (29): ReLU(inplace=True)
      (30): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False)
    )
    (avgpool): AdaptiveAvgPool2d(output_size=(7, 7))
    (classifier): Sequential(
      (0): Linear(in_features=25088, out_features=4096, bias=True)
      (1): ReLU(inplace=True)
      (2): Dropout(p=0.5, inplace=False)
      (3): Linear(in_features=4096, out_features=4096, bias=True)
      (4): ReLU(inplace=True)
      (5): Dropout(p=0.5, inplace=False)
      (6): Linear(in_features=4096, out_features=1000, bias=True)
    )
  )
 )

# La parte importante dell'approccio multimodale: la concatenazione
 (regression): Regression(
  (regression): Sequential(
    (0): Linear(in_features=19200, out_features=9600, bias=True)
    (1): ReLU()
    (2): Linear(in_features=9600, out_features=4800, bias=True)
    (3): ReLU()
    (4): Linear(in_features=4800, out_features=2400, bias=True)
    (5): ReLU()
```

```
106       (6): Linear(in_features=2400, out_features=1024, bias=True)
107       (7): ReLU()
108       (8): Linear(in_features=1024, out_features=512, bias=True)
109       (9): ReLU()
110       (10): Linear(in_features=512, out_features=256, bias=True)
111       (11): ReLU()
112       (12): Linear(in_features=256, out_features=64, bias=True)
113       (13): ReLU()
114       (14): Linear(in_features=64, out_features=1, bias=True)
115     )
116   )
117 )
```

Codice 5.4: Modello predittivo completo

Bibliografia

[1] J. Cantero Priego. *Predicting the number of likes on Instagram with TensorFlow*. PhD thesis, Universitat Politècnica de Catalunya, Oct. 2020.

[2] W. Chen, W. Wang, L. Liu, and M. S. Lew. New Ideas and Trends in Deep Multimodal Content Understanding: A Review. *Neurocomputing*, 426:195–215, Feb. 2021.

[3] J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li, and L. Fei-Fei. ImageNet: A Large-Scale Hierarchical Image Database. In *CVPR09*, 2009.

[4] J. Devlin, M.-W. Chang, K. Lee, and K. Toutanova. BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding, Oct. 2018. arXiv:1810.04805 [cs] version: 1 type: article.

[5] Instagram. About the Instagram Company.

[6] Instagram. Rate limit - API Graph - Documentazione.

[7] Instragram. Instagram for business.

[8] S. Kim, J.-Y. Jiang, M. Nakada, J. Han, and W. Wang. Multimodal Post Attentive Profiling for Influencer Marketing, Apr. 2020.

[9] D. P. Kingma and J. Ba. Adam: A Method for Stochastic Optimization. Technical report, arXiv, Jan. 2017. arXiv:1412.6980 [cs] type: article.

[10] Later.com. How to Calculate Your Engagement Rate on Instagram | Later.

[11] S. Miller, J. Howard, P. Adams, M. Schwan, and R. Slater. Multi-Modal Classification Using Images and Text. *SMU Data Science Review*, 3(3), Jan. 2021.

[12] A. Ng. AI For Everyone.

[13] K. Pykes. The difference between classification and regression in machine learning | by kurtis pykes | towards data science. *Towards Data Science*, 2022.

[14] S. R. Regression Analysis Using Artificial Neural Networks, Aug. 2021.

[15] A. Rakhecha. Understanding Learning Rate, July 2019.

[16] O. Russakovsky, J. Deng, H. Su, J. Krause, S. Satheesh, S. Ma, Z. Huang, A. Karpathy, A. Khosla, M. Bernstein, A. C. Berg, and L. Fei-Fei. ImageNet Large Scale Visual Recognition Challenge. *International Journal of Computer Vision*, 115(3):211–252, Dec. 2015.

[17] K. Simonyan and A. Zisserman. Very Deep Convolutional Networks for Large-Scale Image Recognition. Technical report, ICLR 2015, Apr. 2015. arXiv:1409.1556 [cs] version: 6 type: article.

[18] Statista. Biggest social media platforms 2022.

[19] S. TAFASCA. Multi-View Image Classification, July 2021.

[20] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, L. Kaiser, and I. Polosukhin. Attention Is All You Need. Technical report, Google, Dec. 2017. arXiv:1706.03762 [cs] type: article.