Politecnico di Torino

Collegio di Ingegneria Informatica, del Cinema e Meccatronica

Corso di Laurea Magistrale

in Ingegneria Informatica

Tesi di Laurea Magistrale

Riconoscimento di fake news in lingua italiana mediante analisi di contenuti multimodali

Relatore

Prof. Luca Cagliero

Correlatori

Dott. Lorenzo Vaiani

Dott. Davide Napolitano

Candidato

Lorenzo D'Amico

Luglio, 2023

Indice

Elenco delle figure

Elenco delle tabelle

Capitolo 1

Introduzione

Nell'era dell'informazione digitale, l'emergenza delle fake news rappresenta una sfida significativa per la società. Le fake news sono informazioni intenzionalmente distorte o inventate che hanno l'obiettivo di influenzare l'opinione pubblica e di manipolare le percezioni con conseguenze in ambito politico, sociale ed economico. La loro diffusione è facilitata dalla natura virale dei social media e dalle tecnologie di comunicazione sempre più avanzate, che consentono una rapida e incontrollata propagazione delle informazioni. Inoltre, la semplicità con cui chiunque può creare e condividere contenuti ha amplificato la portata e la velocità di diffusione delle fake news, rendendole negli ultimi anni un fenomeno onnipresente e pervasivo nella società contemporanea. Le fake news si diffondono fino a 100 volte di più delle notizie vere [2], con conseguenze significative. Sono in grado di influenzare gli esiti delle elezioni, alimentare conflitti sociali, danneggiare reputazioni personali e aziendali, e persino mettere a rischio la salute e la sicurezza pubblica. La dimostrazione di ciò si è verificata nel mese di maggio 2023, quando un tweet divenuto virale da parte di un account falso, presentatosi ingannevolmente come l'affermata agenzia di stampa "Bloomberg", riportava l'avvenimento di un'esplosione vicino al Pentagono, con annessa un'immagine generata da un'intelligenza artificiale (Figura 1.1). Il tweet in questione ha suscitato un clima di panico diffuso e ha innescato una brusca caduta del mercato azionario[1]. Pertanto, la ricerca di metodi e strategie per individuare e contrastare le fake news è una priorità improcrastinabile per garantire l'integrità e l'affidabilità dell'informazione.
A tal proposito, l'utilizzo dell'Intelligenza Artificiale come strumento per l'individuazione di notizie false mostra un potenziale significativo per due motivi principali:

- negli ultimi anni, le scoperte nel campo dell'analisi del linguaggio naturale da parte di reti neurali ha permesso di sviluppare metodi avanzati per l'analisi e la comprensione del testo. Tra questi troviamo i modelli denominati Transformer, che consentono di comprendere il significato delle parole in base al contesto in

[1] https://edition.cnn.com/2023/05/22/tech/twitter-fake-image-pentagon-explosion/index.html

Figura 1.1: Immagine falsa condivisa su Twitter dell'esplosione avvenuta vicino al Pentagono. Fonte: [1].

cui sono utilizzate. Questi si adattano particolarmente alla sfida posta dal riconoscimento delle fake news, consentendo un'analisi efficiente ed efficace della componente testuale di una notizia al fine di determinare la sua veridicità.

- L'utilizzo di immagini permette ad un post di avere più interazioni di uno che presenta esclusivamente testo [3]. Inoltre, l'osservazione di un evento tende a generare nelle persone una sensazione di autenticità [4]. Per queste ragioni le notizie sono spesso accoppiate con delle immagini. Le reti neurali convoluzionali consentono di estrarre le caratteristiche rilevanti dalle immagini e identificare eventuali manipolazioni o alterazioni visuali, offrendo un contesto visivo che supporta o confuta le informazioni presentate nel testo e fornire indizi preziosi sulla veridicità delle notizie.

Un modello di intelligenza artificiale per l'individuazione di fake news ha un'ampia gamma di possibili applicazioni, tra queste possiamo elencare:

- **Riduzione della diffamazione degli individui** Per nascondere l'entità di chi sparge notizie fasulle, spesso vengono usati account rubati di individui ignari, con una conseguente diffamazione di questi ultimi. Inoltre, l'utilizzo di notizie false può anche avere l'obiettivo diretto di diffamare persone o gruppi di persone per scopi di propaganda politica. L'impiego di tecnologie di rilevamento delle fake

news può contribuire significativamente a mitigare questo genere di diffamazione [5].

- **Difesa della democrazia** La disinformazione ha il potenziale di promuovere paura, polarizzare l'opinione pubblica e influenzare i risultati di votazioni politiche [6]. L'implementazione di un modello per l'individuazione delle notizie non vere potrebbe costituire un valido strumento per contenere la diffusione di informazioni false, preservando così i processi democratici.

- **Social Network** La maggior parte delle notizie fasulle vengono diffuse sui Social Network. Per i motivi sopracitati, è nell'interesse dei social di bloccare notizie false al momento della pubblicazione, poiché una revisione manuale di tutti i post pubblicati è un'impresa irrealizzabile. Pertanto, è indispensabile sviluppare un sistema che, in maniera immediata e automatica, sia in grado di svolgere tale compito.

Una delle problematiche nell'analisi del testo delle notizie è la presenza di articoli estremamente lunghi, che spesso vengono trascurati dai modelli di analisi testuale attuali preferendo prendere solamente la parte iniziale del testo. Questo rappresenta una limitazione poiché informazioni importanti per l'identificazione potrebbero essere contenute in qualsiasi parte del testo. Pertanto, è essenziale sviluppare approcci di rilevamento delle fake news che siano in grado di gestire efficacemente una maggiore lunghezza dei testi, garantendo così una migliore precisione nella classificazione. Inoltre, le strategie di rilevamento spesso non considerano la parte visuale della notizia oppure, se lo fanno, non considerano adeguatamente l'interazione tra le due diverse modalità (testuale e visuale). Tuttavia, come affermato in precedenza, sia il testo che l'immagine possono fornire indizi importanti per identificare la veridicità delle notizie. L'utilizzo di entrambe le modalità in modo combinato può portare a una migliore comprensione del contesto e fornire informazioni complementari per una classificazione accurata. Pertanto, è fondamentale sviluppare metodi che integrino efficacemente sia l'analisi testuale che l'elaborazione delle immagini per una valutazione completa e precisa delle notizie.
A tal proposito, l'obiettivo di questo studio è stato quello di sviluppare un modello di intelligenza artificiale che, data una notizia composta da testo e delle immagini associate, è in grado di valutarne l'autenticità, sfruttando tutto il testo disponibile in una notizia e la connessione esistente tra le immagini e il testo stesso. In particolare, ha trattato l'utilizzo combinato di transformer e reti neurali convoluzionali per la classificazione di una notizia in una di queste quattro categorie: Certamente Falsa, Probabilmente Falsa, Probabilmente Vera, Certamente Vera. Il progetto è stato avviato con una fase iniziale di studio approfondito del dataset, per acquisire una comprensione completa dello stesso e individuare eventuali inconsistenze al fine di ottenere informazioni fondamentali per l'implementazione del modello. Questo processo preliminare ha permesso di stabilire le

fondamenta su cui basare il lavoro successivo, evidenziando un disequilibrio nelle classi presenti nel dataset. Tale disequilibrio, se non affrontato, potrebbe condurre il modello a favorire in modo sproporzionato le classi che sono più numerose. Lo studio è continuato con la realizzazione di una serie di modelli utilizzati come baseline, seguito da una fase di test che ha coinvolto diverse modifiche e combinazioni di tali modelli di riferimento. Attraverso l'utilizzo di modelli preesistenti per l'analisi del testo, è stato implementato un approccio in grado di estendere la capacità di comprendere e analizzare testi più lunghi, superando così le limitazioni presenti. Inoltre, è stata proposta una metodologia che amplia le capacità di un modello preesistente basato sulla relazione tra immagine e testo. Dopo averlo ri-implementato, ed adattato per poter processare testi in lingua italiana, sono state sviluppate estensioni specifiche che consentono a quest'ultimo di catturare in modo più efficace le informazioni contenute sia nell'immagine che nel testo e ne arricchiscono la sua capacità discriminativa. Queste estensioni si concentrano sui seguenti obiettivi:

- **L'analisi del sentimento** che le notizie evocano, che tende ad essere più negativo per le notizie false, mentre risulta più neutrale per le notizie vere;
- **La concentrazione sulla semantica del testo** per generare nuovi campioni e ottenere un bilanciamento tra le classi attraverso la traduzione dei testi delle notizie in inglese seguita dalla ri-traduzione in italiano;
- **L'analisi delle immagini nel dominio della frequenza**, per un riconoscimento migliore di immagini false;
- **L'unione efficiente delle diverse rappresentazioni** per ciascuna modalità (visiva e testuale) ottenute da modelli diversi per permettere automaticamente al modello di pesare la loro importanza;
- **Un miglioramento nell'aggregazione delle informazioni** delle diverse componenti della notizia;
- **La focalizzazione nelle immagini su regioni di interesse** contenenti oggetti e/o soggetti.

Sono state analizzate le varie estensioni singolarmente e successivamente sono stati presi in considerazione modelli che le combinano sinergicamente. L'utilizzo delle estensioni implementate ha permesso di ottenere risultati superiori a quelli delle baseline prese in considerazione. In particolare, le performance migliori sono state ottenute utilizzando un modello ensemble, ovvero un modello che unisce le predizioni di alcuni dei singoli modelli. Il presente studio si è sviluppato in parallelo alla partecipazione alla competizione MULTI-Fake-DetectiVE (MULTImodal Fake News Detection and VErification)

organizzata da EVALITA (2023) nel contesto dell'etica computazionale, riuscendo ad ottenere il primo posto. Il modello si è inoltre dimostrato valido nella soluzione di una seconda task proposta dalla competizione, che consisteva nell'identificare se il testo e l'immagine fossero intenzionalmente accoppiati per essere fuorvianti, non fuorvianti o totalmente scorrelati. Per quanto riguarda eventuali sviluppi futuri, vi è la possibilità di migliorare la ricerca di correlazioni tra testo e immagine, data l'importanza che hanno nell'identificazione delle fake news. Inoltre, per ampliare ulteriormente le possibilità del modello, sarebbe interessante sperimentare diverse combinazioni delle estensioni o di cercare un'architettura ad hoc per la risoluzione della task secondaria. Il lavoro svolto in questo studio è così strutturato:

- Illustrazione degli elementi teorici e concetti teorici fondamentali alla base delle reti neurali e del deep learning (Capitolo 2);
- Panoramica delle architetture di base utilizzate nella fase di sperimentazione, dove si descrivono modelli esclusivamente testuali, esclusivamente visuali, e infine multimodali.(Capitolo 3);
- Definizione del problema e descrizione dei dati utilizzati per l'addestramento del modello (Capitolo 4);
- Metodologia e progettazione sperimentale (Capitolo 5);
- Presentazione dei risultati ottenuti (Capitolo 6)
- Conclusione e sviluppi futuri del lavoro (Capitolo 7)

Capitolo 2

Elementi teorici e aspetti fondamentali

2.1 Reti Neurali

Le reti neurali sono una tecnologia centrale nello sviluppo dell'IA. Già pensate negli anni '40, la loro nascita è dovuta al tentativo di emulare il cervello umano mediante un gran numero di neuroni interconnessi che simulano il funzionamento della loro controparte umana. In seguito si riportano i concetti fondamentali delle reti neurali dove verranno analizzati il funzionamento e le applicazioni.

2.1.1 Percettrone

Il primo tentativo di rete neurale nacque nel 1958 da Frank Rosenblatt, che propose il Percettrone [7]. Il percettrone è un semplice processo lineare, che prende ispirazione dal funzionamento dei neuroni umani. È un classificatore binario della forma:

$$y(x) = f(w^T x) \tag{2.1}$$

Dove $x \in R^d$ è il vettore di dimensione d che rappresenta un campione, che di seguito sarà chiamato, dall'inglese, *feature vector*. $w \in R^d$ è un vettore di pesi che verranno scelti in maniera adeguata per permettere la corretta classificazione di x. La funzione f è invece una funzione a gradino, tale che:

$$y = \begin{cases} +1 & se \;\; w^T x \geq 0 \\ -1 & se \;\; w^T x < 0 \end{cases} \tag{2.2}$$

Una rappresentazione grafica del percettrone si può osservare in figura 2.1. Questo può essere utilizzato per risolvere un problema di classificazione binario. L'algoritmo inizia

con $w = 0$, e per ogni campione x presente si calcola $y(x)$. Supponendo di star classificando un campione con etichetta $y = +1$ e $y(x)$ risulta essere $+1$, allora il campione è stato classificato correttamente, e non verrà effettuata nessuna modifica ai pesi w. Se invece $y(x)$ è -1, i pesi vengono aggiustati seguendo la formula $w = w + y^*x$ dove y^* è la vera label del campione x. Il processo è lo stesso nel caso in cui $y = -1$. Nonostante

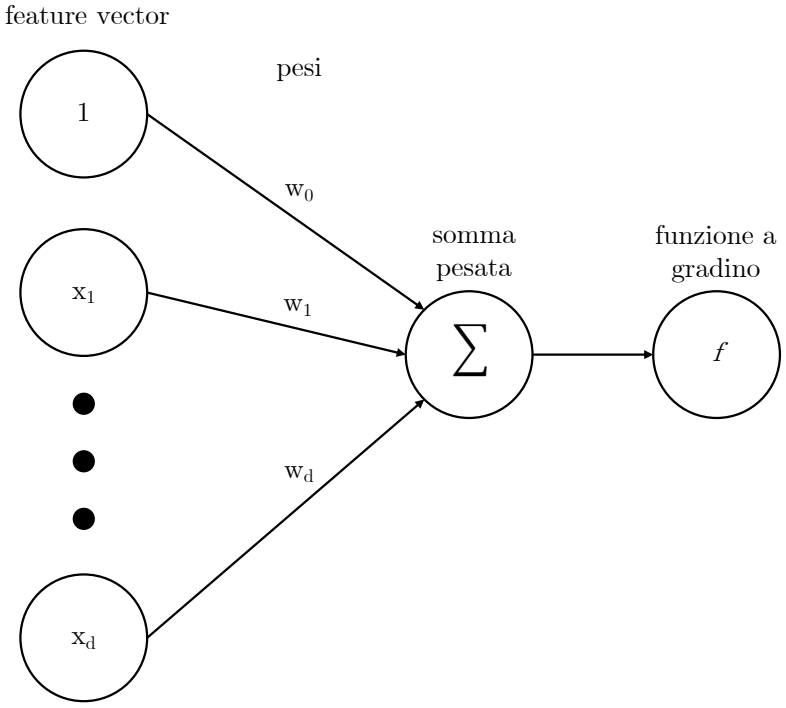


Figura 2.1: Rappresentazione grafica del percettrone.

fosse un classificatore binario molto semplice nel suo funzionamento, fornisce una forte garanzia formale, ovvero che se il dataset è **linearmente separabile**, il percettrone troverà un un iperpiano che separa le due classi in un numero finito di passi. Inoltre, sotto alcune condizioni, riesce anche a dare un tetto massimo al numero di questi passi. Il problema del percettrone sta nel fatto che se i dati non sono linearmente separabili, non convergerà mai, facendo oscillare per sempre i pesi. In aggiunta non dà nessuna garanzia sulla bontà del piano di separazione, ed è molto soggetto al fenomeno dell'overfitting. Il percettrone, così com'è stato descritto, non presenta un'interpretazione probabilistica delle predizioni. Per ottenere questo obiettivo, si può pensare di cambiare la funzione a gradino con un'altra funzione non lineare, la funzione Sigmoide, la cui comparazione con la precedente è osservabile in figura 2.2 e espressa dalla formula:

$$\sigma(x) = \frac{1}{1 + e^{-x}} \tag{2.3}$$

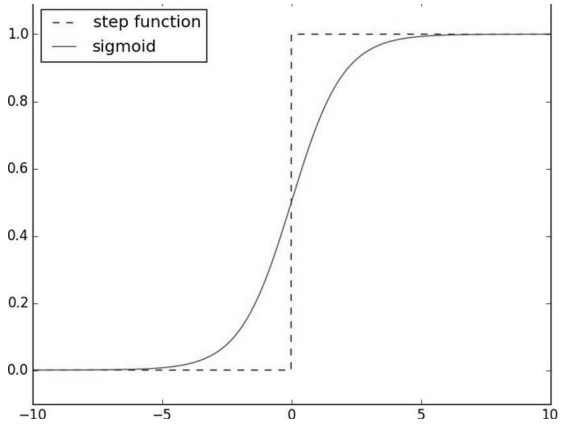


Figura 2.2: Confronto tra funzione a gradino e funzione sigmoide. Da notare che la funzione gradino qui rappresentata può assumere i valori 0/1, e non -1/1. Questo per semplificazione di confronto con la sigmoide. Fonte: [8].

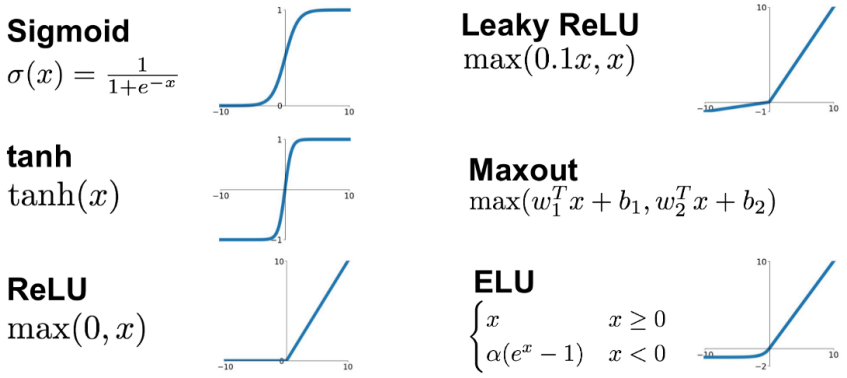


Figura 2.3: Una serie di possibili funzioni di attivazione utilizzabili. Fonte: [12].

2.1.2 Deep learning

Nonostante il percettrone riesca solamente a risolvere problemi che presentano dati perfettamente linearmente separabili, è il fondamento delle moderne reti neurali, che si basano sui **Teoremi di approssimazione universale** [9] [10] [11]. Questi teoremi affermano che una rete neurale a strati con un numero sufficientemente grande di neuroni è in grado di approssimare qualsiasi funzione continua su un insieme compatto. Di conseguenza, per raggiungere tale obiettivo, possiamo definire il **Multi Layer Perceptron (MLP)**, chiamato anche **Deep feed forward neural network**. Un MLP è composto da più strati di neuroni, e in particolare si può dividere in uno strato d'ingresso, diversi strati nascosti, e uno strato di uscita. L'output del layer l può essere espresso nella formula:

$$x_{l+1} = \sigma_l(W_l x_l) \tag{2.4}$$

La rappresentazione grafica di un layer del MLP è raffigurata in 2.4. La funzione σ

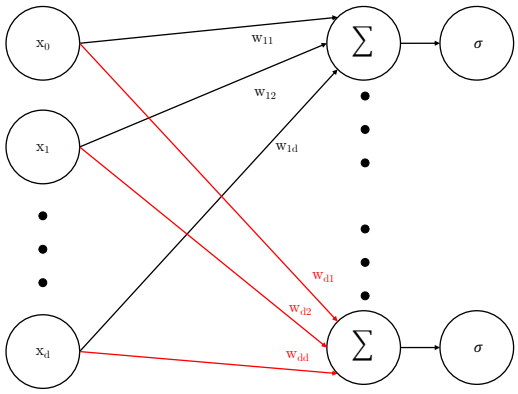


Figura 2.4: Un layer del MLP. I pesi sono raffigurati in colore diverso per sottolineare che sono relativi a diversi neuroni.

è una funzione non lineare che può essere di diverso tipo, può anche cambiare da layer a layer e viene chiamata **funzione di attivazione**. Questa è importante per introdurre non linearità nella rete, dal momento che i fenomeni reali che si vogliono approssimare sono di natura non lineare. Ci sono diverse funzioni di attivazione che possono essere utilizzate, è possibile osservare le più famose in figura 2.3. Una di quelle che hanno riscosso più successo è la **unità lineare rettificata** (ReLU dall'inglese Rectified Linear Unit), utilizzata per la prima volta in [13]. La ReLU è definita come:

$$ReLU(x) = max(0, x) \tag{2.5}$$

2.1.3 Training

Dato quanto visto finora, possiamo rappresentare un MLP come una funzione $g_W(x)$, dove W sono i pesi della rete, mentre x è l'input. Considerato un dataset etichettato $D : \{(x_i, y_i)\}_{i=1}^{n}$ con $(x_i, y_i) \in (X \times Y)$, dove X rappresenta lo spazio dei campioni, e Y lo spazio delle loro label, che possono assumere valori nell'insieme $(C_0, ..., C_n)$.

Funzioni di Loss

Possiamo definire la funzione di Loss (in seguito chiamata semplicemente Loss) la funzione che misura l'errore della predizione del classificatore $g_W(X)$ rispetto alle vere label Y. Quindi la Loss è una funzione $L_W : Y \times Y \rightarrow R$ e può essere vista come la penalità del classificare y^* quando la vera label è y. Per questo studio saranno importanti in particolare due tipi di loss, ovvero la **Cross entropy Loss** e la **Focal Loss**, che saranno ora discusse. La cross entropy loss è la più comune funzione di loss usata per il

problema di classificazione ed è definita come:

$$CE = -\frac{1}{N}\sum_{i=1}^{n}\sum_{j=1}^{c} y_{i,j}\log(g_{W,j}(x_i)) \tag{2.6}$$

dove $y_{i,j}$ è il j-esimo elemento di un vettore con una codifica "one-hot", ovvero un vettore di zeri con un 1 nella posizione della vera classe del campione x_i. Definendo $g_{W,j}(x_i)$ come la probabilità data in output di g_W della classe j-esima, si può osservare che il suo contributo è tanto più basso quanto più ci si avvicina a 1 nel caso in cui invece j lo sia. In generale quindi la cross entropy misura quanto la predizione del classificatore si avvicina alla label reale e si allontana al tempo stesso da quelle sbagliate. Tuttavia, la cross-entropy loss ha due problemi principali: il primo è che non tiene in considerazione un eventuale sbilanciamento tra classi, andando a dare implicitamente più importanza alla classe maggioritaria, il secondo è che non distingue campioni facili e difficili da classificare. I campioni difficili da classificare sono quei campioni per il quale il modello fa ripetutamente errori. Per mirare a risolvere questi errori, si può utilizzare una variante chiamata focal loss [14]. Abbreviando la predizione del modello con p_t per semplicità di notazione, possiamo definire la focal loss come:

$$FL = -\frac{1}{N}\sum_{i=1}^{n}\sum_{j=1}^{c} y_{i,j}(1-p_t)^{\gamma}\log(p_t) \tag{2.7}$$

Dove γ è chiamato **parametro di concentramento**, ed è un iper-parametro. Come si può osservare, se il modello classifica bene, il termine $(1-p_t)$ è piccolo, riconducendosi alla funzione 2.6. Nel caso in cui invece il classificatore sbagli, il termine $(1-p_t)$ cresce proporzionalmente all'errore commesso. Errori più grandi sono generati da campioni difficili da classificare. Questo errore viene scalato dal parametro di concentramento. Questa funzione di loss risolve il problema dello sbilanciamento delle classi perché i campioni della classe maggioritaria, essendo visti più spesso, sono più semplici da classificare, mentre quelli che si vedono di meno sono più difficili.

Algoritmi di ottimizzazione

L'idea di una rete neurale è quella di aggiornare i pesi W in modo da minimizzare la funzione di Loss, ovvero avvicinare il più possibile le predizioni alle vere label. La funzione di loss in genere non è una funzione convessa, di conseguenza per trovare il minimo di quest'ultima si può ricorrere al metodo di **Discesa del gradiente** (GD dall'inglese Gradient Descent), che come specificato in [15], è un modo per minimizzare una funzione di Loss $J(\theta)$ parametrizzata dai parametri $\theta \in R^d$ di un modello andando ad aggiornarli nella direzione opposta del gradiente della funzione $\nabla_\theta J(\theta)$ calcolato rispetto ai parametri stessi. In altre parole, si segue la direzione della discesa della superficie formata dalla funzione di Loss in modo da cercare un minimo come osservabile in figura

2.5. Quindi, formalmente, l'algoritmo è definibile come:

$$\theta^{(t+1)} = \theta^{(t)} - \alpha \nabla_W J(\theta^{(t)}) \tag{2.8}$$

Dove α è chiamato *Learning Rate* (LR) e indica la lunghezza dello step che si esegue

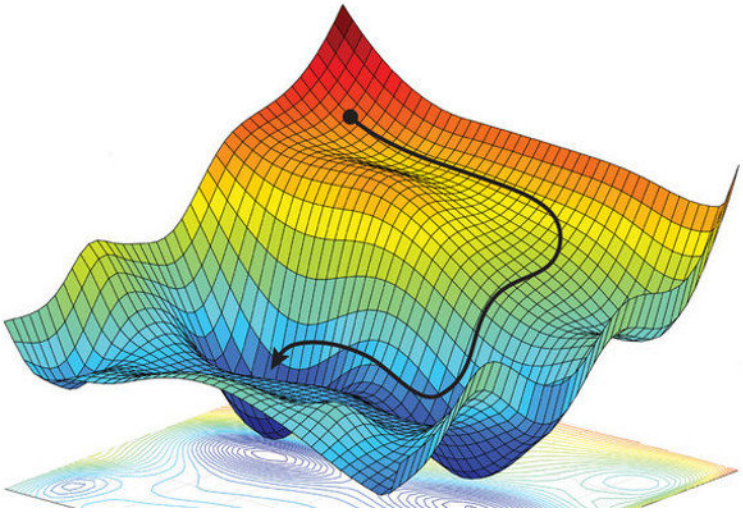


Figura 2.5: Rappresentazione grafica della discesa del gradiente. Fonte: [16].

per raggiungere il minimo. Un LR troppo grande rischia di far oscillare il valore attorno al minimo, mentre un valore troppo piccolo potrebbe risultare in una convergenza lenta dell'algoritmo. Un metodo che si è dimostrato efficace è quello di attuare strategie di **Learning Rate Decay**, ovvero di iniziare con un LR grande e usare, col tempo, valori più piccoli, seguendo un andamento predefinito come, ad esempio, un andamento a step, lineare o cosinusoidale. Inizializzare l'algoritmo con un LR alto permette al modello di "fuggire" da eventuali minimi locali, inoltre in [17] gli autori sostengono che questo può anche sopprimere la memorizzazione di dati rumorosi, e il suo successivo abbassamento nel tempo migliora l'apprendimento di pattern complessi. Il GD non garantisce che si otterrà il minimo globale della funzione, ma nel caso del Deep Learning questo può essere un vantaggio. Infatti la distribuzione dei dati di training non è uguale a quella dei dati sul quale il modello sarà utilizzato. Se si trovasse il minimo globale avremmo un problema di **overfitting**, per il quale il modello si è talmente tanto adattato ai dati usati in fase di training da non riuscire più a generalizzare. Come detto in precedenza, l'obiettivo è quello di minimizzare la Loss. Questa deve essere calcolata su tutti i campioni del dataset D, che sono $\{x_i, y_i\} \in X, Y$ con $i = 1, ..., n$ dove n è la cardinalità del dataset (cioè il numero totale di campioni). Questo significa che la Loss da minimizzare è una combinazione della loss calcolata per ogni campione, se ne può

prendere quindi la media:

$$L_W(X) = \frac{1}{n}\sum_{i=1}^{n} l_W(\{x_i, y_i\}) \tag{2.9}$$

Dove l_W è la particolare funzione di loss utilizzata. Da quest'ultima, si può osservare che per trovare il gradiente, bisogna calcolare il gradiente di ogni elemento della sommatoria, il che rende il calcolo computazionalmente dispendioso per due motivi: Il primo è il numero di parametri W per il quale non si può trovare una soluzione semplice. Il secondo invece è il numero di campioni n, che rende il calcolo del gradiente praticamente irrealizzabile per n molto grande. La soluzione è l'utilizzo del metodo della **Discesa del Gradiente Stocastico** (SGD dall'inglese Stochastic Gradient Descent), per il quale si considera un sottoinsieme rappresentativo di campioni di dimensione $m \ll n$ chiamato **batch**. Per questo sottoinsieme si ha che:

$$\frac{1}{m}\sum_{i=1}^{m} l_W(\{x_i, y_i\}) \approx \frac{1}{n}\sum_{i=1}^{n} l_W(\{x_i, y_i\}) \tag{2.10}$$

Questo significa che il calcolo del gradiente può avvenire solo su un batch alla volta, andando a usare ogni volta batch con dati diversi fino a coprire l'intero dataset di partenza. Quando questo accade, si dice che il modello è stato allenato per un'**epoca**. Il modello viene allenato per diverse epoche. SGD non solo diminuisce di molto il costo per iterazione, ma migliora anche la generalizzazione riducendo l'overfitting, dal momento che i campioni di un batch sono scelti casualmente ad ogni epoca e quindi il modello non ottimizza mai sugli stessi dati e, comunque, mai tutti insieme. Altri algoritmi di ottimizzazione, che sono varianti dello SGD, sono stati proposti. Tra questi troviamo il famoso algoritmo di ottimizzazione **Adam** [18]. L'idea di base di Adam è di calcolare dei learning rate adattivi per ciascun parametro sulla base di stime del primo e del secondo momento dei gradienti. Queste stime vengono calcolate utilizzando medie mobili esponenziali decadenti del gradiente e del gradiente al quadrato, rispettivamente. Oltre ai tassi di apprendimento adattivi, Adam include anche un termine di *momentum* che accumula il gradiente nel tempo e aiuta a velocizzare la convergenza, permettendo anche di uscire in maniera efficace da punti stazionari locali. Il termine di momentum viene anche adattato sulla base delle stime del primo e del secondo momento dei gradienti. Adam è efficiente dal punto di vista computazionale e richiede relativamente poca memoria. Per evitare overfitting, di solito si aggiunge un termine di penalità alla loss chiamato **weight decay**, in modo da favorire pesi con un valore non troppo grande. Gli autori di [19] hanno dimostrato che il weight decay in Adam può essere dannoso alle prestazioni. Hanno perciò introdotto un nuovo metodo chiamato **AdamW** (Adaptive Weight Decay with Adam). L'idea di AdamW è quella di disaccoppiare il weight decay

dal passo di aggiornamento del gradiente, applicandolo dopo quest'ultimo, invece di includerlo nell'aggiornamento stesso. Questo aiuta a prevenire l'annullamento del weight decay da parte del passo di ottimizzazione.

Backpropagation

Trovato il valore della Loss, bisogna aggiornare i pesi della rete, e per quanto abbiamo detto prima, bisogna trovare il $\nabla L_W(B)$ dove B è un batch del dataset. Per calcolare in maniera efficiente il gradiente, si utilizza un metodo chiamato Backpropagation [20]. Il calcolo del gradiente avviene trovando le derivate parziali di L_W rispetto a tutti i pesi della rete. Senza andare troppo nel dettaglio, questo avviene in due fasi:

1. *Forward Pass*, che altro non è il passaggio nel quale l'input viene propagato nella rete per ottenere il valore di output, ottenendo un valore per ogni neurone seguendo l'equazione 2.1

2. *Backward pass*, nel quale si calcolano le derivate parziali della loss rispetto ai nodi della rete a partire dall'ultimo layer. Per calcolare le derivate si applica la regola della catena ma si utilizza direttamente il valore della derivata calcolata al passo precedente.

Quindi, la back propagation è la differenziazione automatica con accumulazione all'indietro applicata alle reti neurali.

Fine-tuning

Per allenare una rete neurale, pertanto, serve una considerevole quantità di dati. Tuttavia, negli ultimi anni si è dimostrato che questo è vero a meno che non si voglia allenare una rete da zero. Infatti, pur non avendo a disposizione una grande quantità di dati, si può utilizzare la tecnica chiamata del **trasferimento dell'apprendimento** (*tranfer learning* in inglese). Il transfer learning è una tecnica di apprendimento che permette di utilizzare conoscenze acquisite da un problema di apprendimento per risolvere un altro problema simile. L'idea alla base del transfer learning è che molte caratteristiche apprese da un modello in una determinata area possono essere riutilizzate in altre aree. In questo modo, il transfer learning consente di ridurre i tempi e i costi di sviluppo dei modelli, migliorare le prestazioni e la generalizzazione del modello su nuovi dati. Ad esempio, nell'analisi di un testo, le informazioni a basso livello intrinseche della lingua sono sempre le stesse, a prescindere dal particolare testo scelto. Ci sono molte tecniche per sfruttare modelli già allenati, ma quella più utilizzata soprattutto nei problemi di classificazione, è la tecnica del **fine-tuning**. Il fine-tuning consiste nell'adattare un modello pre-addestrato su una grande quantità di dati ad un nuovo task specifico. In pratica, si rimuovono l'ultimo strato o gli ultimi strati del modello e si aggiunge un nuovo strato di output personalizzato. Quindi, si specializza l'intero modello su un insieme di dati di addestramento per il nuovo task specifico, utilizzando il modello pre-addestrato

come punto di partenza. Il fine-tuning è una tecnica molto efficace per ottenere buoni risultati con un numero relativamente ridotto di dati di addestramento per il nuovo task specifico. Il fine-tuning richiede un modello pre-addestrato che sia generalista, cioè in grado di estrarre feature di alto livello che siano utili per il nuovo task specifico. Ricapitolando, fino ad ora è stata descritta la struttura base di una rete neurale artificiale, come viene calcolato l'errore di predizione e come questo viene usato per aggiornare i pesi della rete. È stato inoltre osservato come allenare in maniera efficace una rete senza aver bisogno di una notevole quantità di dati.

2.1.4 Tecniche per migliorare le prestazioni

Ora andremo ad osservare diverse tecniche relative alle reti neurali utilizzate per migliorare le performance.

Normalizzazione del batch (batch normalization in inglese): Nell' allenamento di una rete profonda, la distribuzione degli input di ogni layer può variare molto, causando un'instabilità della rete rendendo difficile la sua ottimizzazione. Il processo di batch normalization mira a risolvere questo problema normalizzando gli input di ogni layer (l'output del layer precedente) in modo che abbiano media nulla e varianza unitaria. La normalizzazione è fatta su ogni batch. In particolare, per ogni dimensione si calcola media e la deviazione standard, si sottrae poi al vettore la media, e si divide per la deviazione standard. Inoltre, si possono utilizzare due parametri apprendibili direttamente dalla rete γ e β, che permettono alla rete di capire quanto è rilevante la normalizzazione del batch in quel layer. In termini formali quindi la normalizzazione è data da:

$$\hat{x} = \gamma \frac{x - E[x]}{\sqrt{Var[x]}} + \beta \tag{2.11}$$

Dove x è il batch dei vettori delle caratteristiche. Si può notare che se la rete si rende conto che la procedura di Batch normalization non è utile, può impostare $\gamma = \sqrt{Var[x]}$ e $\beta = E[x]$, risultando in $\hat{x} = x$. Quindi, la batch normalization migliora il flusso del gradiente attraverso la rete, aiutando a risolvere i problemi del gradiente che svanisce (vanishing gradient) e del gradiente che esplode (exploding gradient). Riduce inoltre la dipendenza dall'inizializzazione dei parametri, permette l'uso di un passo di apprendimento più alto e agisce anche come forma di regolarizzazione visto che è calcolato su un batch i cui campioni sono scelti casualmente.

Inizializzazione dei pesi (weight initialization in inglese): L'inizializzazione dei pesi è il processo con il quale vengono assegnati i valori iniziali ai pesi della rete neurale. La scelta iniziale dei pesi può avere un impatto significativo sulla convergenza e sull'accuratezza del modello. Di fatto, un approccio banale potrebbe essere quello di impostare inizialmente i pesi a 0. Tuttavia questa non è una buona idea perché quando tutti i pesi

sono uguali si crea il **problema della simmetria**, ovvero il gradiente sarà lo stesso per tutti i pesi e quindi saranno aggiornati della stessa quantità durante il processo di backpropagation, andando a ridurre la capacità del modello di apprendere pattern complessi. Quindi un principio importante è quello di rompere la simmetria nell'inizializzazione dei pesi. Un'idea per reti poco profonde è quella di inizializzare i pesi con piccoli valori casuali, ad esempio prendendoli da una distribuzione gaussiana con media zero e deviazione standard 10^{-2}, tuttavia per reti profonde questo metodo non funziona perché sopraggiunge il problema del gradiente che svanisce, quindi il gradiente tende a diventare nullo dopo diversi layer, non permettendo alla rete di imparare. Una possibile soluzione al problema è l'**inizializzazione di Xavier** [21], dove i pesi iniziali vengono campionati da una distribuzione gaussiana con media 0 e deviazione standard pari a $\sqrt{\frac{1}{D_{in}}}$, dove D_{in} è la dimensione in ingresso. Questo metodo si basa sul fatto che la varianza dell'output è la stessa di quella in input, infatti se questo non accade si possono verificare i problemi di convergenza sopracitati. Tuttavia questa inizializzazione è stata pensata per la funzione di attivazione *tanh*. Con la ReLU, per mantenere la varianza in ingresso uguale a quella in uscita bisogna usare come varianza $\sqrt{\frac{2}{D_{in}}}$, dove il 2 compensa per la mancanza della parte negativa della funzione.

Dropout: Il Dropout [22] è una tecnica usata per migliorare la generalizzazione in una rete neurale. Nel dropout una proporzione di neuroni selezionati casualmente sono temporaneamente rimossi durante il passo in avanti della fase di allenamento, ovvero il loro output è impostato a 0. L'idea del dropout è quella di prevenire il co-adattamento delle caratteristiche, ovvero evitare che una rete impari ad associare esclusivamente alcune caratteristiche ad altre. Per esempio, un gatto ha orecchie, pelo e zampe, quindi una rete potrebbe osservare queste tre cose e classificare un'immagine come "gatto". Tuttavia, queste caratteristiche non sono sempre presenti un un'immagine (ad esempio perché le zampe sono nascoste), e per questo motivo la rete potrebbe classificare male un'immagine a test time che non le possiede tutte. Mediante l'utilizzo di dropout si aiuta la rete a capire che non è necessario che siano sempre tutte presenti e ad evitare overfitting. Il dropout può anche essere visto come un grande insieme di modelli diversi che condividono gli stessi parametri. Andando ad annullare le connessioni dei neuroni, si abbassano anche il numero di parametri della rete, per questo motivo il dropout viene usato principalmente nei layer completamente connessi con molti neuroni. In fase di test, tuttavia, bisogna fare attenzione all'utilizzo del dropout, questo perché in questo caso tutti i neuroni sono attivi, ma in fase di allenamento questo non è mai stato fatto. Quello che si vuole ottenere è una "media" di tutti i modelli ottenuti usando dropout. Per affrontare questo problema, l'output di ogni neurone viene moltiplicato con la probabilità di dropout al momento del test, con l'effetto di simulare il valore atteso dell'output di quando il dropout viene applicato durante l'addestramento. Questo ridimensionamento garantisce che la rete rimanga ben calibrata e produca previsioni

accurate anche quando il dropout non viene applicato durante il test. Per ottenere lo stesso effetto ma performance più veloci, invece di moltiplicare l'output in fase di test, si può ugualmente dividere l'output del layer al momento di allenamento.

Insieme di modelli (model ensemble in inglese): il model ensemble è una tecnica di deep learning che coinvolge la combinazione di molteplici modelli per ottenere una prestazione predittiva migliore di ogni modello individuale. Questa tecnica è basata sul principio che combinando le predizioni di più modelli, si possono sfruttare le capacità predittive di ognuno e di mitigare un eventuale overfitting. Un modo diverso per fare un insieme di modelli è di prendere più stati dello stesso modello ottenuti in fase di allenamento. L'idea alla base di questo metodo è che il modello visita diversi minimi della funzione di loss, trovando diverse soluzioni che potrebbero dare risultati migliori se se ne calcola la media.

2.2 Applicazioni

2.2.1 Computer Vision

La computer vision (vision artificiale in italiano) è un ramo dell'IA che consente ai computer e ai sistemi di elaborare informazioni da immagini digitali, video e altri input visivi, e di utilizzare tali informazioni per compiere delle azioni. In passato, la computer vision viene definita come la tecnologia che si occupa di estrarre informazioni su una scena attraverso l'analisi di immagini di quella scena [23]. Tuttavia con il passare del tempo, ha trovato applicazione in molti altri campi, come nel riconoscimento di impronte digitali, modellazione 3D e stabilizzazione video [24]. Recentemente passi avanti sono stati fatti soprattutto nella generazione automatica di immagini [25] [26] e ulteriori ottimi risultati sono stati ottenuti grazie all'introduzione dei *Diffusion model* [27]. Le classiche reti neurali sono limitate nell'elaborazione delle immagini, perché riescono difficilmente a carpire le relazioni tra pixel vicini e sfruttare le informazioni apriori della realtà rappresentate nelle immagini. Le immagini presentano spesso delle strutture visuali definite che possono essere sfruttate nella loro elaborazione. Le principali sono:

Auto Similarità L'Auto similarità si riferisce alla proprietà di un'immagine di presentare una struttura ripetitiva a diverse scale di osservazione. Ciò significa che le parti dell'immagine sono simili tra loro e i pixel tendono a continuare quelli presenti nelle vicinanze.

Invarianza alla traslazione Le immagini presentano una invarianza alla traslazione, ovvero un oggetto rappresentato nell'immagine mantiene le sue caratteristiche, indipendentemente dalla sua posizione all'interno dell'immagine.

Gerarchia Le informazioni visive sono di solito strutturate "a strati", cioè le informazioni di basso livello come bordi e texture formano quelle di alto livello, che sono a

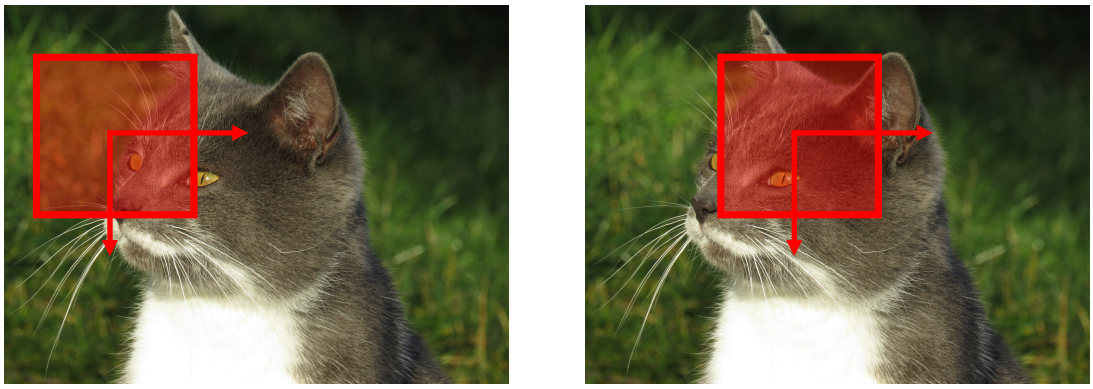


Figura 2.6: Rappresentazione grafica della convoluzione discreta su una fotografia.

loro volta parte di altre informazioni. Prendendo ad esempio un volto umano, bordi e texture sono presenti occhi, naso, bocca etc.. che a loro volta sono parte dell'intero volto.

Composizionalità La composizionalità si riferisce alla capacità di combinare queste caratteristiche per formare oggetti complessi, che di solito sono formati sempre allo stesso modo dalle loro componenti più a basso livello.

Questo significa che i dati visivi sono composti da schemi gerarchici, locali e invarianti alle traslazioni. Queste informazioni intrinseche alle immagini possono essere sfruttate per creare delle reti neurali che le utilizzano per un'elaborazione efficiente delle immagini. La prima rete di questo tipo è la Rete Neurale Convoluzionale (CNN dall'inglese Convolutional Neural Network), usata per la prima volta in [28]. Il loro nome nasce dal fatto che sfruttano l'operazione di *convoluzione*. Date due funzioni $f(t), g(t) : [a, b] \in R \to R$, si definisce convoluzione di f e g la funzione:

$$(f * g)(x) = \int_a^b f(t)g(t - x)dt \tag{2.12}$$

La convoluzione è una funzione lineare e commutativa, ma la proprietà più importante è che è l'equivarianza alla traslazione, cioè:

$$f(x - x_0) * g(x) = (f * g)(x - x_0) \tag{2.13}$$

il che si ricollega con la proprietà di Invarianza alla traslazione vista precedentemente. Le immagini sono in due dimensioni e formata da un numero finito di pixel, per questo possiamo definire **la convoluzione discreta a due dimensioni**. Sia $f : R^2 \to R^3$ un'immagine RGB (possiede tre canali), allora per ogni canale possiamo definire:

$$(f * g)[m, n] = \sum_k \sum_l f[k, l]g[m - k, n - l] \tag{2.14}$$

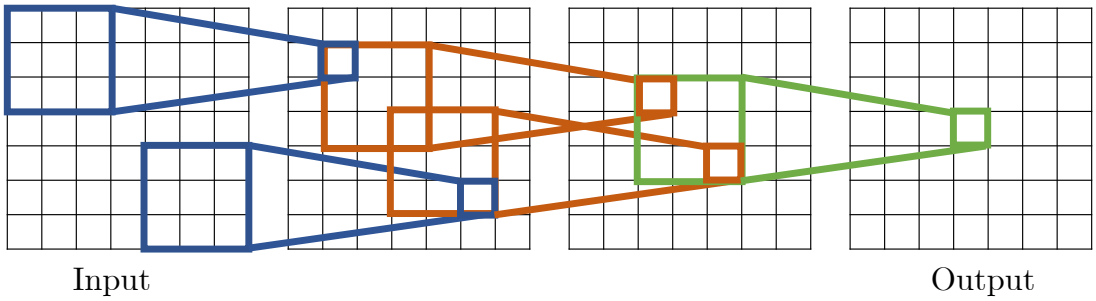


Figura 2.7: Visualizzazione dell'applicazione dei filtri.

La convoluzione dei vari canali viene combinata linearmente per avere un output con un solo canale, per questo ad un input con n canali verrà applicato un filtro con altrettanti canali. Inoltre un filtro ha una dimensione k, che sta per il numero di pixel in larghezza e in lunghezza. Quindi un filtro ha dimensioni $k \times k \times n$. Graficamente, equivale a definire una finestra mobile g che scorre sull'immagine, moltiplicando ogni suo valore per il valore del canale del pixel considerato. Una rappresentazione grafica è visibile in figura 2.6. Il filtro g prende il nome di **kernel**, mentre l'output dell'operazione di convoluzione su un'immagine viene chiamato **feature map**. Una feature map rappresenta la presenza o l'assenza di un tratto ad ogni posizione dell'immagine o della feature map del layer precedente, usato come input. L'applicazione di un filtro riduce la dimensione dell'output rispetto a quella dell'input. In particolare, preso un'input di dimensione $N \times N \times c$, applicando un kernel di dimensione $k \times k \times c$ si otterrà in output una feature map di $M \times M \times 1$, dove M è definita come $\frac{(N-k)}{passo} + 1$ dove per *passo* si intende di quanti pixel si muove il filtro. Anche considerando un passo pari a 1, la feature map risulterà più piccola dell'input. Per evitare tale problema si può applicare del padding attorno l'input. Per avere più canali in output, inoltre, si vanno ad applicare più filtri. Quindi applicando C filtri, in output si ottiene una dimensione pari a $M \times M \times C$. Ogni elemento dell'output di un kernel di dimensione k, dipende da una zona dell'input di dimensione $k \times k$, come si può osservare in figura 2.7. Questo permette di avere filtri che osservano parti dell'immagine sempre più ad alto livello man mano che si va in profondità nella rete 2.8.
Per rendere le reti più gestibili e controllare le dimensioni delle feature map, inoltre, si utilizza la tecnica chiamata **Pooling**. Il pooling può essere di diversi tipi, tra cui il max pooling e il average pooling. Il pooling consiste nell'aggregare le informazioni di una regione di input in un unico valore rappresentativo. Nel max pooling, ad esempio, si seleziona il valore massimo di una regione dell'input e lo si utilizza come valore rappresentativo per quella regione. Questo permette di mantenere solo le informazioni più rilevanti dell'input, ignorando le altre informazioni meno significative. Anche per il pooling è definibile una dimensione e un passo. In conclusione le CNN sono in gra-

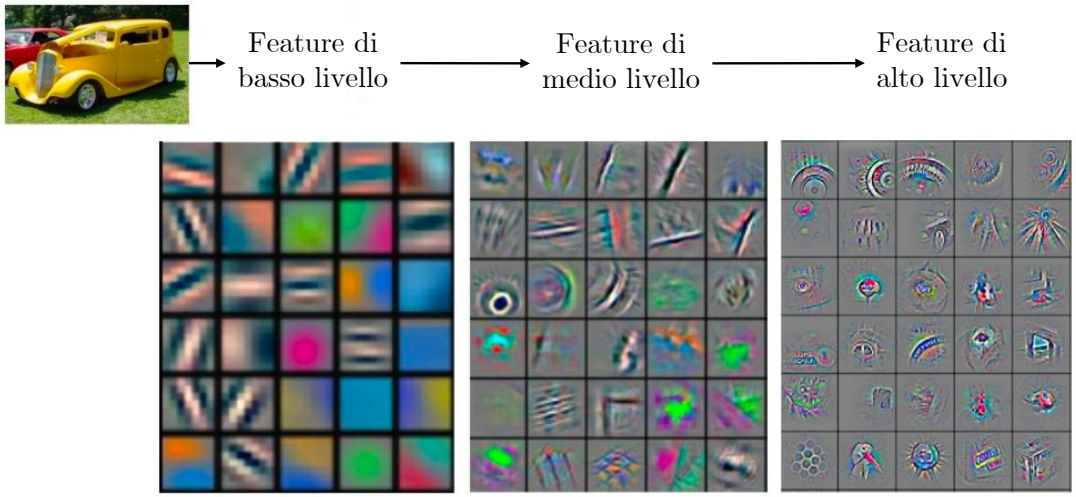


Figura 2.8: Rappresentazione visuale dei filtri che rappresentano le feature a diverso livello di dettaglio. Fonte: [29].

do di rilevare automaticamente feature complesse nelle immagini, aprendo la strada a una vasta gamma di applicazioni, tra cui riconoscimento di oggetti, classificazione di immagini e segmentazione di immagini.

2.2.2 Natural Language Processing

Mentre la Computer Vision si occupa dell'elaborazione delle immagini, il campo noto come Natural Language Processing (NLP) è quel ramo dell'IA che si occupa di sviluppare sistemi in grado di comprendere il linguaggio naturale degli esseri umani. L'obiettivo è quello di creare algoritmi e modelli che permettano ai computer di analizzare, interpretare e generare testo in modo simile a quello di un essere umano. Come dichiarato in [30], il NLP può essere classificato in due parti: il *Natural Language Understanding* (NLU) e il *Natural Language Generation* (NLG). Il NLU permette alle macchine di comprendere il linguaggio naturale e di analizzarlo mediante l'estrazione di concetti, entità, emozioni, parole chiave, e così via. D'altra parte il NLG è un processo informatico che consente ai computer di generare automaticamente testo in linguaggio naturale a partire da dati strutturati o informazioni fornite dall'esterno. Le applicazioni del NLP sono molteplici, tra queste troviamo:

- La traduzione automatica di testi da una lingua all'altra. Usare l'IA per la traduzione permette di tradurre intere frasi mantenendone il significato.
- La classificazione di testi, che permette di separare testi in base al loro contenuto. Quest'applicazione è anche alla base dei filtri anti spam.

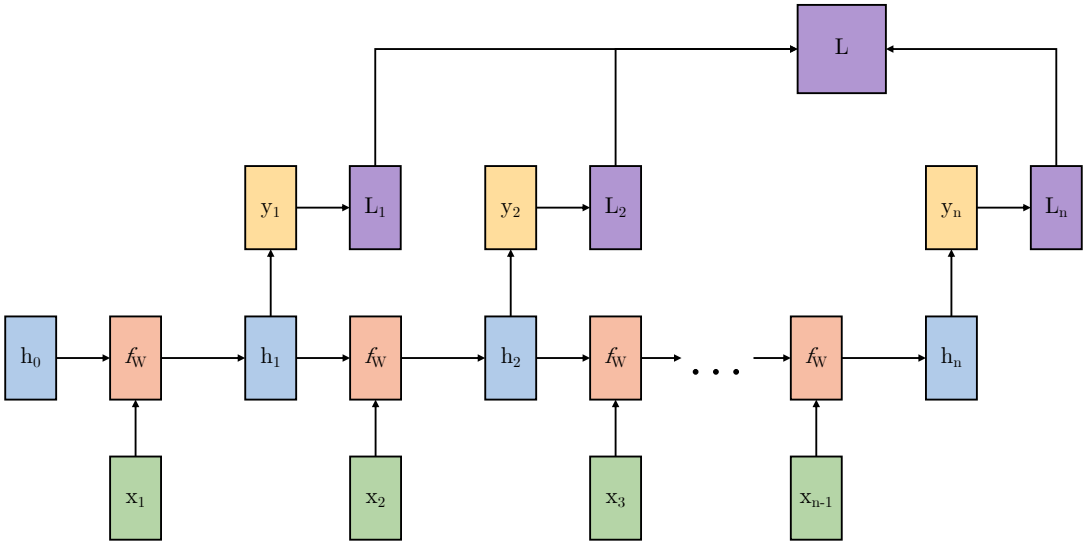


Figura 2.9: Rappresentazione grafica di una RNN con n input e n output.

- Estrazione di informazioni, con la quale si estraggono informazioni dal testo. Un esempio più specifico è il Riconoscimento di entità nominate (NER dall'inglese *Named Entity Recognition*) , che si concentra sull'individuazione e l'etichettatura di entità specifiche all'interno del testo, come persone, luoghi, organizzazioni, date, valute, quantità e così via.

Una tecnologia fondamentale per lo sviluppo dell'NLP sono le Reti Neurali Ricorrenti (RNN dall'inglese *Recurrent Neural Network*). Come specificato in [31], una RNN è una rete neurale con delle connessioni di feedback (a ciclo chiuso). Quindi, in una RNN, l'output di una determinata unità neurale viene utilizzato come input per la stessa unità neurale durante il passaggio successivo. Ciò consente alle RNN di modellare sequenze di dati, in cui l'output in una determinata posizione dipende non solo dall'input in quella posizione, ma anche da tutti gli input precedenti. Le RNN trovano spazio nell'analisi delle sequenze, in quanto possono catturare le dipendenze a lungo termine nei dati di sequenza, per questo sono particolarmente adatte all'elaborazione di testi, che sono sequenze di parole sintatticamente collegate tra di loro. La forma più semplice di RNN, anche chiamata "vanilla", può essere vista come una funzione che produce uno stato h_t, prendendo in input lo stato precedente h_{t-1} e l'input x_t. In formule:

$$h_t = f_W(h_{t-1}, x_t) \tag{2.15}$$

Dove W sono i pesi della rete. In particolare, per quanto detto nel paragrafo 2.1, la funzione è una funzione lineare dove è applicata non-linearità mediante σ:

$$h_t = \sigma(W_h h_{t-1} + W_x x_t) \tag{2.16}$$

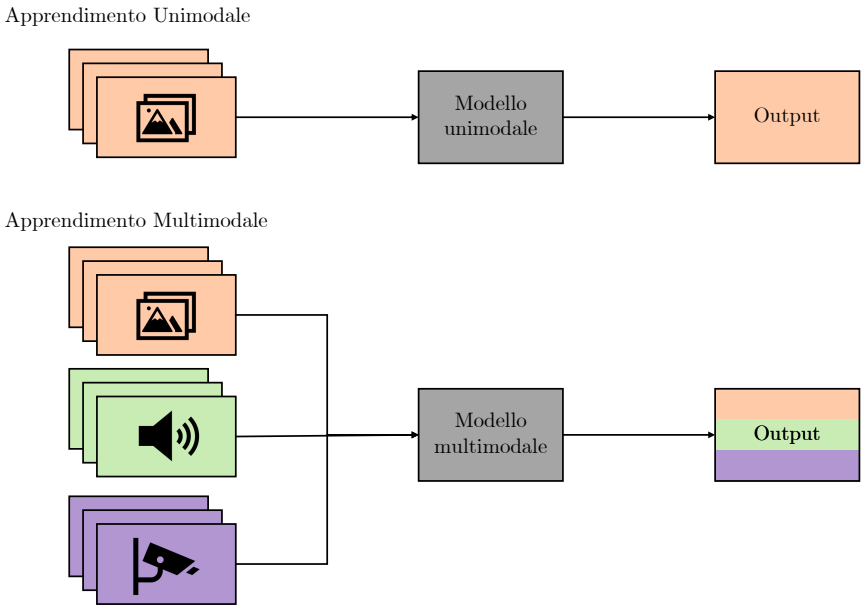


Figura 2.10: Rappresentazione visuale della differenza tra modelli unimodali e multimodali.

Di conseguenza, l'output y_t è infine dato da:

$$y_t = W_y h_t \tag{2.17}$$

La loss L_i generata da ogni output y_i viene combinata per ottenere una loss finale L. Una rappresentazione grafica di una RNN che genera n output a partire da n output (RNN *many-to-many*) è osservabile in figura 2.9. Le RNN hanno aperto nuove possibilità per il NLP e rimangono un ottimo punto di partenza, soprattutto per quanto riguarda la generazione di testi, vista la loro possibilità di usare lo stato precedente per generarne uno nuovo. Tuttavia, soffrono di vari difetti, il primo tra i quali è quello del **vanishing gradient**. Questo problema è causato dalle dipendenze a lungo termine nei dati sequenziali. Per questo motivo RNN vanilla non viene usata, preferendo modelli più complessi che presentano meno problemi, quale ad esempio il **Long Short Term Memory** (LSTM) [32] o il **Gated Recurrent Unit** [33], che non verranno qui approfondite dal momento che in questo studio è stato utilizzato un altro approccio più recente e avanzato basato sui modelli Transformer, che sarà trattato in modo più dettagliato in seguito.

2.2.3 Apprendimento Multimodale

Fino ad ora si è parlato esclusivamente di immagini e testo, tuttavia, il tipo di dati in input ad una rete neurale sono molteplici e diversi. Tra questi troviamo, ad esempio, audio, video, dati provenienti da sensori, dati biometrici ecc... Allenare una rete su

una modalità di dati potrebbe tuttavia non considerare completamente il contesto nel quale i dati sono presenti. Di fatto, nella realtà, le informazioni si trovano spesso sotto forma di diverse modalità che da sole non darebbero un chiaro quadro della situazione. Basti pensare a un filmato, che è composto da video e audio, oppure un libro, che si compone di testo e immagini. Per questo motivo, negli ultimi tempi, si è diffusa l'adozione di dati multimodali, ovvero dati di diversa natura che si riferiscono ad uno stesso fenomeno. La differenza tra modelli unimodali e multimodali è osservabile in figura 2.10. Come descritto in [34] il motivo per il quale usare dati multimodali sta nel fatto che informazioni complementari possono essere stratte da ognuna delle modalità considerate per la funzione di apprendimento, portando a una rappresentazione più ricca che potrebbe produrre prestazioni più elevate rispetto a quelle che si otterrebbero se si usassero le modalità singolarmente. Il processo di apprendimento delle reti neurali che utilizza dati multimodali viene chiamato apprendimento multimodale. Le applicazioni dell'apprendimento multimodale sono molteplici, ma si concentrano soprattutto su dati testuali uniti a dati visivi come immagini e video. Questo è dovuto al fatto che sono dati più usati, quindi di più interesse, ma soprattutto più facilmente reperibili in grande quantità. Esempi di applicazioni sono:

- **Risposta a domande visive** con il quale preso in input un'immagine e una domanda, il modello risponde a quest'ultima basandosi sul contenuto visivo.
- **Analisi mediche basate su immagini e testo** dove un modello multimodale può analizzare referti medici e immagini diagnostiche per identificare segni di malattie e fornire diagnosi più accurate.
- **Generazione di highlight in eventi sportivi** I frame video e l'audio dei commentatori possono essere utilizzati dal modello per identificare i momenti significativi come gol o falli.

L'utilizzo di dati multimodali, nonostante promettente, porta a diverse sfide. Le principali sono la disponibilità di dati, e l'allineamento delle modalità. La prima sfida riguarda il problema della collezione di una grande quantità di dati etichettati che siano consistenti tra di loro, il che potrebbe limitare l'efficacia dei modelli. La seconda consiste invece nel problema intrinseco dei dati di diverse modalità di avere diversi livelli di rumore e variabilità, che potrebbe portare a risultati subottimali o inaccurati. Parte importante dell'apprendimento multimodale è il modo nel quale i dati di diverse modalità vengono combinati tra di loro, e dipendentemente dall'obiettivo della rete, i vari metodi possono essere più o meno performanti. Questo processo prende il nome di "fusione" (fusion in inglese). I vari tipi di fusion sono classificabili approssimativamente in tre categorie, la cui descrizione è stata presa da [35] e qui di seguito riassunta:

Early Fusion La metodologia *Early Fusion* consiste nell'integrare i dati grezzi delle

varie modalità in una rappresentazione unica prima di procedere all'operazione di apprendimento. Per poter essere utilizzata, i dati devono essere efficacemente combinabili. Non è tuttavia una tecnica largamente utilizzata data la sua limitata applicabilità.

Intermediate Fusion La tecnica di *Intermediate Fusion*, conosciuta anche come *Feature-level fusion*,, si riferisce alla trasformazione dei dati grezzi in una rappresentazione più ad alto livello, mappandoli mediante l'utilizzo di una rete neurali. Dopo aver unito le varie rappresentazioni, si ottengono delle *feature map* multimodali.

Late Fusion L'approccio *Late Fusion* consiste nell'estrarre le decisioni delle singole modalità, per poi essere combinate e ottenere la decisione finale.

Concludendo, l'apprendimento multimodale è un'area di ricerca in rapida crescita, e ha il potenziale per consentire sistemi IA più potenti e flessibili in grado di apprendere da una gamma più ampia di dati. Pertanto, è probabile che sarà un'area di ricerca attiva per molti anni a venire.

2.3 Metriche di prestazione

Le metriche di prestazione sono strumenti fondamentali per valutare l'efficacia di un modello. In particolare, in un problema di classificazione, vogliamo misurare quanto accuratamente il modello riesce ad assegnare le etichette corrette ai campioni. Le metriche ci forniscono una valutazione quantitativa di questa capacità.
Una delle metriche di base per la valutazione di un modello di classificazione è l'**accuratezza**, definita come il rapporto tra le predizioni corrette e quelle totali. Supponendo un problema di classificazione binario con etichette "Positivo" e "Negativo", possiamo definire:

- *True Positive (TP)* il numero di campioni correttamente classificati come positivi, ovvero il numero di campioni effettivamente positivi e classificati come tali.
- *False Positive (FP)* il numero di campioni erroneamente classificati come positivi, ovvero il numero di campioni negativi classificati però come positivi.
- *True Negative (TN)* il numero di campioni correttamente classificati come negativi, ovvero il numero di campioni effettivamente negativi e classificati come tali.
- *False Negative (FN)* il numero di campioni erroneamente classificati come negativi, ovvero il numero di campioni positivi classificati però come negativi.

L'accuratezza è definita come:

$$accuratezza = \frac{(TP + TN)}{(TP + TN + FP + FN)} \tag{2.18}$$

L'accuratezza tuttavia è una metrica non affidabile quando la distribuzione delle label è sbilanciata. Infatti un modello che predice sempre "Negativo" potrebbe avere un'accuratezza molto alta solo perché il numero di campioni negativi è in maggioranza, nonostante non stia davvero classificando. Per misurare in maniera migliore le prestazioni di un classificatore si possono analizzare la performance su classi specifiche mediante altre due metriche, ovvero la **precisione** e il **richiamo**. La precisione misura la percentuale dei campioni correttamente classificati nella classe c rispetto al numero di campioni predetti come classe c e, per la classe positiva, è definita come:

$$\text{precisione} = \frac{TP}{(TP + FP)} \tag{2.19}$$

Il richiamo invece misura la percentuale dei campioni correttamente classificati nella classe c rispetto al numero totale di campioni della classe c e, per la classe positiva, è definita come:

$$\text{richiamo} = \frac{TP}{(TP + FN)} \tag{2.20}$$

È verificabile che se il modello predicesse sempre label "Negativo", per la classe Positiva si avrebbe una precisione del 100% e un richiamo dello 0%. Quindi queste metriche sono l'una l'antitesi dell'altra. Per avere un'unica metrica è stata introdotta un'ulteriore metrica chiamata **F1-score**, definita come:

$$\text{F1-score} = 2 * \frac{precisione * richiamo}{precisione + richiamo} \tag{2.21}$$

L'F1-score può essere calcolato solo per una classe, ma si possono usare dei metodi di aggregazione generalizzabili anche al caso non binario. Come specificato in [36], questi metodi consistono nel calcolare l'F1-score per ogni classe, per poi calcolarne la media seguendo un particolare criterio.
In questa ricerca sono stati utilizzati due metodi di aggregazione degli F1-score. Il primo viene chiamato **Macro F1-score** e consiste in una semplice media dei valori ottenuti per ogni classe. Verrà abbreviato da ora in poi in F1. Formalmente viene definito come:

$$\text{F1} = \frac{1}{C} \sum_{i=1}^{C} \text{F1-score}_i \tag{2.22}$$

Dove C è il numero di classi, e F1-score_i è il valore dell'F1-score ottenuto per la classe i.
La seconda metodologia permette invece di ottenere l'**F1-weighted**, che consiste nel tenere in considerazione la distribuzione delle label per poter calcolare una media pesata sul numero di campioni che corrispondono alla classe stessa. Formalmente può essere

definita come:

$$\text{F1-weighted} = \frac{1}{N_T} \sum_{i=1}^{C} \text{F1-score}_i * N_i \tag{2.23}$$

Dove N_T è il numero di campioni totali, mentre N_i è il numero di campioni che compongono la classe i. Questa metrica si dimostra particolarmente adatta per dataset caratterizzati da un disequilibrio nelle classi.

Capitolo 3

Architetture di base

3.1 Modello testuale

3.1.1 BERT

BERT [37] sta per Bidirectional Encoder Representations from Transformers, ed è un modello di linguaggio che ha rivoluzionato il campo del NLP. È stato introdotto da Google nel 2018 e usa un approccio bidirezionale per acquisire il contesto di una parola in un testo, considerando le parole che si trovano sia alla sua destra che alla sua sinistra. BERT è riuscito ad ottenere prestazioni allo stato dell'arte su vari benchmark di NLP. Alla base di BERT c'è un'architettura chiamata **Transformer**.

Transformer e il meccanismo di Self-attention
Il Transformer [38] è un'architettura introdotta nel 2017 da un team di ricercatori di Google. Il modello è progettato per usare il meccanismo dell'attention, e in particolare la **self-attention**, cioè un meccanismo che permette di ottenere una rappresentazione di una sequenza che tenga in considerazione anche le relazioni tra le diverse componenti di quest'ultima. Infatti, a differenza delle RNN i Transformer non processano sequenzialmente le parti di un testo, ma cercano le relazioni tra le parole considerandole tutte simultaneamente. Il Transformer è diventato le fondamenta di molti modelli allo stato dell'arte, oltre a BERT, infatti, anche la serie GPT (Generative Pre-trained Transformer) [39] di OpenAI è basata su di loro. In aggiunta, il principio di self-attention ha trovato applicazione in diverse aree di ricerca, tra cui la Computer Vision e la Speech Recognition. Gli autori dell'articolo sostengono che la natura sequenziale di RNN e LSTM preclude la parallelizzazione all'interno dei campioni nella fase di allenamento, la quale è critica per quanto concerne l'utilizzo della memoria per sequenze lunghe. Il meccanismo di Attention invece ha permesso di modellare le dipendenze incrementando il grado di parallelizzazione.

Il Transformer è un'architettura che permette la trasduzione di sequenze, ovvero un

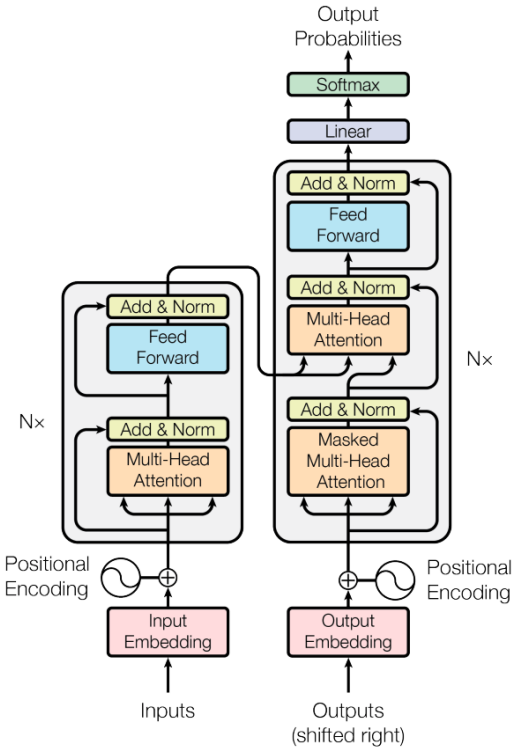


Figura 3.1: Architettura del Transformer. Fonte: [38].

modello che permette di ottenere una sequenza in output da una in input. L'architettura, visibile in figura 3.1, è basata sul modello **encoder-decoder** e verrà di seguito descritta. L'encoder prende in input la sequenza simbolica $x = (x_1, ..., x_n)$ e produce una sequenza di output $z = (z_1, ..., z_n)$. Da ora in poi ogni simbolo x_i verrà chiamato token. L'encoder è composto da una pila di sei layer identici, ognuno composto da due sotto-layer, ovvero un **meccanismo di self-attention multi-headed** e una **rete position-wise feed-forward completamente connessa**. Ogni sotto-layer produce un output di dimensione $d_{model} = 512$ ed è collegato mediante una connessione residua seguita da un'operazione di **Layer normalization**. Il decoder prende in input l'output generato in precedenza per generare l'output finale $y = (y_1, ..., y_m)$. Anch'esso è composto da sei layer identici composti da tre sotto-layer, due dei quali hanno la stessa struttura di quelli del decoder. Il terzo è sempre un meccanismo di self-attention multi-headed, che però prende come input l'output z dell'encoder. Anche qui è presente la connessione residua seguita dall'operazione di layer normalization per ogni sotto-layer. Tuttavia il primo sotto-layer di attention è stato leggermente modificato, questo perché durante l'operazione di self-attention ogni token partecipa al calcolo delle dipendenze con tutti gli altri token. Per fare in modo che i punteggi di attention per un token non vengano calcolati per il token stesso e quelli successivi a quest'ultimo, si spostano i token a destra di una posizione e si utilizza una tecnica chiamata **masking** che vedremo

successivamente.

Ora passiamo a una descrizione di come opera il modello dall'inizio alla fine. In input è presente una sequenza, ovvero un testo. Il testo viene diviso in token (che possono essere parole, oppure sotto parti di queste) e attraverso un dizionario ogni token viene mappato in un vettore di valori continui. A questi vettori viene aggiunta un'informazione sulla loro posizione nella sequenza, questo passaggio è necessario perché il Transformer non possiede uno stato come le RNN. Non è possibile pensare di utilizzare un singolo numero come indice per rappresentare la posizione del vettore, questo perché per sequenze lunghe l'indice può diventare molto grande, e se normalizzato tra 0 e 1, può causare problemi per sequenze di lunghezze diverse, dal momento che verrebbero normalizzate diversamente. Gli autori hanno perciò optato per usare delle **codifiche posizionali** (positional encoding in inglese) di questo tipo:

$$\begin{aligned} PE_{(k,2i)} &= sin(\frac{k}{10000^{\frac{2i}{d_{model}}}}) \\ PE_{(k,2i+1)} &= cos(\frac{k}{10000^{\frac{2i}{d_{model}}}}) \end{aligned} \tag{3.1}$$

Dove k è la posizione del token nella sequenza, i è la dimensione del vettore. Quindi, per ogni posizione dispari si crea un vettore usando la funzione coseno, mentre per quelle pari si usa una funzione seno. Questi vettori si sommano poi al vettore corrispondente nella sequenza, andando a dare alla rete informazioni sulla posizione. Si può osservare che fissato i, ad ogni k corrisponderà una sinusoide diversa con lunghezza d'onda $\lambda = 2\pi n^{\frac{2i}{d_{model}}}$. Quindi, le lunghezze d'onda formano una successione geometrica che varia da 2π a $2\pi n$. Pertanto utilizzare la codifica posizionale porta ai seguenti vantaggi:

1. Normalizzazione tra $[-1, 1]$ grazie all'utilizzo di funzioni sinusoidali
2. Si ottiene un indice unico per ogni posizione
3. Fornisce un modo per misurare la similarità tra diverse posizioni

Passiamo ora a vedere in dettaglio layer di Multi-headed attention, osservabile in figura 3.2. Il modulo viene definito multi-headed perché l'input viene diviso in N parti (dove N è il numero dei singoli moduli) prima di entrare nel modulo di attention, che viene appunto chiamato *testa* (head in inglese). Questo viene fatto perché ogni testa impara una rappresentazione diversa della sequenza di input e, combinando le uscite di più teste, il modello può catturare pattern più complessi. L'input viene innanzitutto passato per tre layer lineari completamente connessi, in modo da ottenere i vettori denominati **Query** (Q), **Key** (K) e **Value** (V). Tra Q e K viene eseguito un prodotto scalare per ottenere una matrice di punteggi, che viene scalata per $\sqrt{d_k}$, dove d_k è la dimensione di Q e K. Questo è fatto per evitare eventuali problemi numerici dovuti

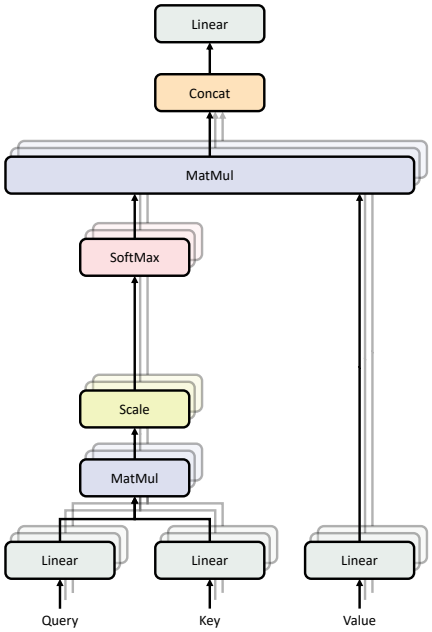


Figura 3.2: Layer di Multi-headed attention.

alla moltiplicazione. Successivamente si applica una funzione di Softmax per ottenere una matrice di probabilità, ovvero la matrice di pesi di attention, che viene moltiplicata mediante prodotto scalare per V per ottenere un output che manterrà solamente i token più importanti secondo la matrice dei pesi. Riassumendo il processo può essere visto come:

$$Attention(Q, K, V) = softmax(\frac{QK^T}{\sqrt{d_k}})V \tag{3.2}$$

L'output di ogni testa viene poi concatenato e passato ad un layer lineare. Alla fine si ottiene quindi un output che contiene informazioni su come ogni token è correlato a ogni altro token della sequenza.

Come anticipato il precedenza, il decoder si occupa della generazione di testo, ed è **auto-regressivo**, ovvero prende l'output come input, oltre che alle informazioni di attention della sequenza di input fornite dall'encoder. L'input dell'encoder inizia con il token <start>, per poi accrescere con i token da esso stesso generati. Questo input passa per un layer di multi-headed attention ma per i motivi citati in precedenza viene applicato il **masking** : alla matrice dei punteggi viene applicata una maschera $M(d_{i,j})$ con:

$$d_{i,j} = \begin{cases} -inf & se \;\; j > i \\ 0 & se \;\; j \leq i \end{cases} \tag{3.3}$$

Cioè le viene sommata una matrice con la stessa dimensione che possiede esclusivamente i valori 0 (nel triangolo inferiore della matrice e sulla diagonale), e $-inf$ nel triangolo

superiore. In questo modo, all'applicazione della softmax, la matrice dei pesi avrà 0 nelle posizioni relative ai token successivi a quello della riga considerata, così da preservare la natura auto-regressiva del transformer.
Al secondo layer di Multi-headed attention Q e K sono l'output dell'encoder, mentre V è l'output del primo layer di attention del decoder, questo processo permette al decoder di comprendere quale input dell'encoder è importante in relazione all'input del decoder. Dopo un altro layer lineare completamente connesso e la successiva normalizzazione, è presente un classificatore lineare seguito da una funzione softmax, questo ha un numero di output pari alla dimensione del dizionario dei token considerati, andando a prendere il token con la probabilità maggiore di essere selezionato. Questo viene aggiunto alla sequenza di input del decoder e il processo continua finchè non viene predetto il token <end>. Sia l'encoder che il decoder possono essere ripetuti andando a formare una struttura a livelli, con l'output finale dell'encoder che entra in ingresso a ogni livello del decoder.
In definitiva, il Transformer ha rivoluzionato il campo del NLP grazie alla sua capacità di gestire grandi quantità di dati e di modellare relazioni complesse tra le parole.

Architettura e strategia di allenamento

Come affermato in precedenza, BERT è un modello di NLP basato sull'architettura del transformer. Alla base di BERT c'è la sua natura bidirezionale, che gli fornisce l'abilità di comprendere il contesto di una parola in una sequenza prendendo in considerazione sia le parole che vengono prima, che quelle che vengono dopo nella sequenza stessa.
Gli autori hanno utilizzato due passi per l'allenamento di BERT: *pre-training* e *fine-tuning*. Durante la fase di pre-training il modello è allenato su dati non etichettati su diversi problemi. Nella seconda fase invece il modello è inizializzato con i parametri ottenuti dalla fase precedente e passa attraverso un processo di fine-tuning sul particolare problema che si sta considerando con dati etichettati. Come si può osservare, quindi, per diversi problemi l'architettura di BERT è la stessa, presentando solamente minime differenze. Nell'articolo originale sono presentate due versioni di BERT, ovvero $BERT_{BASE}$ che presenta 12 blocchi di transformer (L), una dimensione del vettore nascosto (H) pari a 768, e il numero di teste di self-attention (A) pari a 3, con un totale di 110M di parametri. $BERT_{LARGE}$ invece presenta L=24, H=1024, A=16 e un numero totale di parametri che risulta essere più del triplo, ovvero 330M. Dal momento che BERT deve essere adatto a risolvere problemi distinti usando la stessa strategia di pre-training, è stata sviluppata una strategia che permette di rappresentare in input sia una singola frase, che una coppia (Domanda, Risposta). Di fatto, il primo token di ogni sequenza è sempre il token speciale **[CLS]**. Questo token è molto importante perché la sua rappresentazione ottenuta dall'ultimo layer nascosto riassume tutta la sequenza, e quindi le dipendenze in essa contenute. Le coppie di sequenze, invece, vengono raggruppate in un'unica sequenza e separate dal token **[SEP]**. Per indicare se un token

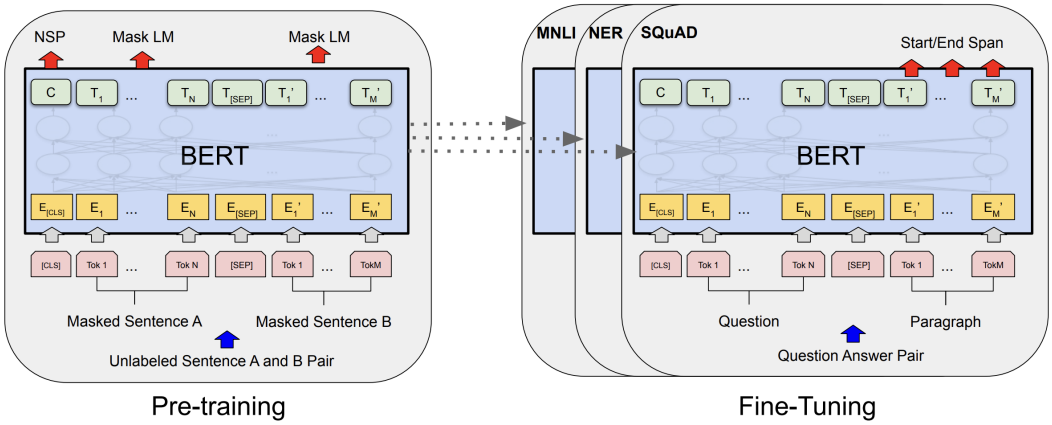


Figura 3.3: Processo di pre-training e fine-tuning di BERT. Fonte: [37].

appartiene alla prima sequenza o alla seconda, si aggiunge ai token un embedding E apprendibile tramite addestramento . In definitiva quindi dalla sequenza in input si ricavano i token, cui viene aggiunto l'embedding E e gli embedding posizionali. Una rappresentazione del processo di pre-training e il successivo fine-tuning si può osservare in figura 3.3

Fasi di allenamento

Come anticipato, BERT viene pre-allenato su due problemi con campioni non etichettati, ovvero non supervisionati. I modelli precedenti non sono bidirezionali, dal momento che se un questo potesse vedere sia i token precedenti che quelli successivi a un token, un token vedrebbe anche se stesso e il modello riuscirebbe a predire la parola successiva in maniera scontata. Gli autori sono riusciti quindi a trovare un problema per il pre-training chiamato **Masked LM** (MLM), nel quale si maschera una certa percentuale di token casuali in input usando il token speciale **[MASK]**, chiedendo al modello di predire i token mascherati. Al modello viene però richiesto di predire esclusivamente i token [MASK], invece che tutta la sequenza (che come abbiamo detto sarebbe banale da fare). Tuttavia questo creerebbe una mancata corrispondenza tra pre-training e fine-tuning, perché in quest'ultimo il token [MASK] non appare mai. Quindi la strategia attuata è stata quella di selezionare il 15% dei token, di questi selezionati, con una probabilità dell'80% viene sostituito con il token [MASK], con il 10% di probabilità viene sostituito con un token casuale, mentre il resto delle volte non viene sostituito affatto.
Il secondo problema usato per il pretraining prende il nome "Predizione della prossima frase" (Next Sentence Prediction (NSP) in inglese), scelto in modo che il modello riesca ad apprendere la relazione tra due testi, che è una capacità molto importante nei problemi di:

- Risposta alle domande (Question Answering (QA) in inglese), che è un problema per il quale il modello deve rispondere ad una domanda in input posta in linguaggio naturale

- Inferenza su linguaggio naturale (Natural Language Inference (NLI) in inglese), dove al modello vengono fornite due frasi che sono una premessa e un'ipotesi. Il modello deve determinare, basandosi sulla premessa, se l'ipotesi è vera, falsa, oppure indeterminata.

NSP consiste nell'allenare un classificatore binario dove le sequenze A e B, B il 50% delle volte è la vera sequenza che segue effettivamente A (etichettata come `isNext`), l'altro 50% delle volte è invece una sequenza casuale (etichettata come `notNext`). Per quanto riguarda la fase di fine-tuning, l'idea di BERT di inserire le due sequenze in input si è rivelata un'ottima scelta per ricavare mediante il meccanismo di self-attention la relazione tra le due sequenze, mentre in modelli precedenti era comune codificare indipendentemente i due pezzi di testo prima di applicare un meccanismo di attenzione incrociata. Per i problemi di classificazione in particolare, la rappresentazione del token [CLS] è usata come input per un classificatore.

Concludendo, BERT è un potente mezzo per l'NLP grazie al suo approccio innovativo per ottenere una rappresentazione delle sequenze che tiene in considerazione le relazioni tra le parole. BERT è riuscito a raggiungere lo stato dell'arte in vari problemi di NLP ed è chiaro che il suo impatto continuerà ad ispirare nuove ricerche nel campo dell'IA.

3.2 Modelli visuali

3.2.1 VGG Net

La rete VGG (Visual Geometry Group) [40] è stata proposta nel 2014, rivoluzionando il modo di costruire le architetture delle reti neurali convoluzionali. L'idea alla base dello studio di VGG è stata quella di mettere in discussione le convoluzioni usate fino a quel momento, e modificare la struttura delle reti convoluzionali per renderle più efficienti e performanti.

Innovazione del design dei blocchi di convoluzione Le reti VGG proposte sono due: **VGG16** e **VGG19**, le quali architetture sono osservabili in figura 3.4. Il numero nel nome indica il numero di layer della rete principale. La VGG16 è composta da cinque fasi convoluzionali composti da due o tre operazioni di convoluzione e uno di pooling, mentre la VGG19 ha quattro operazioni di convoluzione nelle ultime due fasi. La rete segue delle regole di design ben definite:

- Tutte le convoluzioni usano filtri 3x3 con passo 1 e padding 1

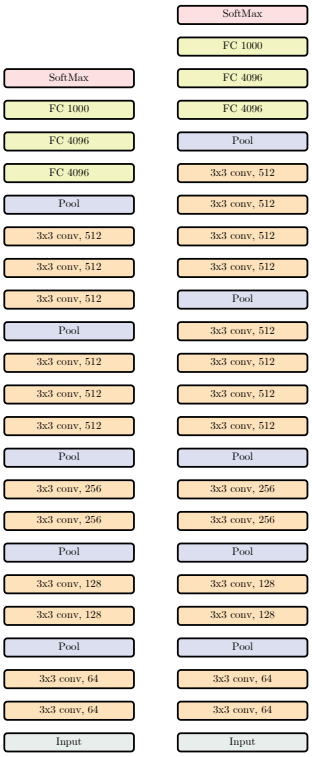


Figura 3.4: Architetture di VGG16 (a sinistra) e VGG19 (a destra).

- Tutte le operazioni di max pooling sono 2x2 con passo 2, e dopo queste si duplica il numero dei canali

Per quanto riguarda la prima regola, essa si basa sul fatto che usare due layer convoluzionali 3x3 porta ad avere lo stesso receptive field che si avrebbe utilizzando un unico layer convoluzionale 5x5, ma con meno parametri e costo computazionale. Di fatto, considerando una feature map di dimensioni $H \times W \times C$, per un layer convoluzionale 5x5 si hanno $25C^2$ parametri e $25C^2HW$ FLOP (Floating Point Operation), mentre con due layer 3x3 si passa a $18C^2$ parametri e $18C^2HW$ FLOP. Un altro vantaggio sta nel fatto che si hanno due convoluzioni e quindi due funzioni non lineari applicate dopo di queste, permettendo l'apprendimento di pattern più complessi. Anche la seconda regola permette di rendere più efficiente la rete, infatti mantenere le informazioni spaziali fisse ma duplicando il numero di canali si aumentano i parametri da $9C^2$ a $36C^2$ ma si riduce l'utilizzo di memoria da $4HWC$ a $2HWC$, rispetto al caso in cui si duplicano invece le dimensioni spaziali.

In conclusione, la VGG Net ha indubbiamente avuto un impatto significante nel campo della Computer Vision. La sua semplicità ed efficacia hanno messo in luce l'importanza della profondità di una rete e portato ispirato molti avanzamenti nel campo del deep learning, ponendo solide basi sullo sviluppo di reti più sofisticate.

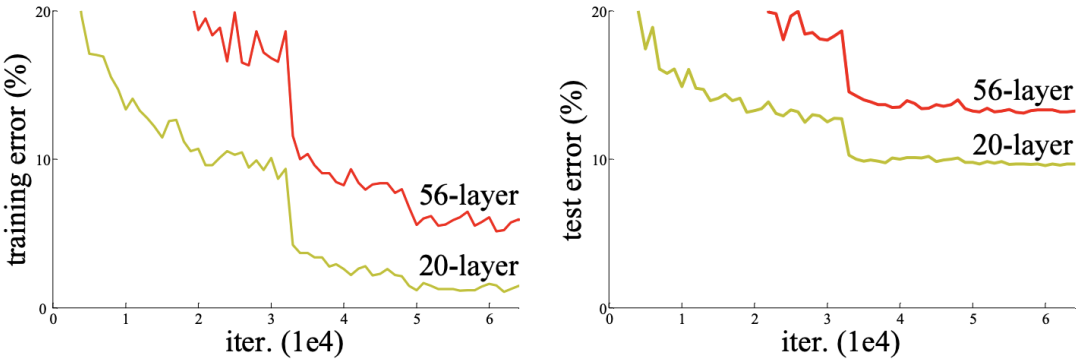


Figura 3.5: Errore in fase di train (a sinistra) e in fase di test (a destra) su CIFAR-10 [41] per una rete non residuale con 56 e 20 layer. Fonte: [42].

3.2.2 ResNet

Le Reti Residuali (Residual Network in inglese) anche chiamate ResNet sono un tipo di rete neurale che hanno rivoluzionato il campo della computer vision. Sono state introdotte per la prima volta nel 2015 in [42] e da quel momento sono diventate una delle reti più importanti per l'elaborazione delle immagini, soprattutto per il problema di classificazione.

Il blocco residuale

Le ResNet sono state rivoluzionarie perché hanno permesso di aumentare la profondità della rete, ovvero il numero di layer, senza perdere performance. Di fatto, si può notare dalla figura 3.5 che incrementando il numero di layer, anche se le performance dovrebbero aumentare, queste in realtà si abbassano, e modelli più profondi performano peggio di quelli superficiali, anche sul training set, andando a scartare l'ipotesi di overfitting. Il problema non dovrebbe sussistere, dal momento che reti più profonde dovrebbero essere capaci di imitare reti meno profonde andando semplicemente a impostare i layer extra come identità. Il problema è quindi di ottimizzazione, ovvero i modelli con più layer fanno fatica ad approssimare funzioni identità attraverso multipli layer non lineari. Inoltre, modelli troppo profondi presentano anche il problema del vanishing gradient. La soluzione trovata dagli autori è stata quella di cambiare la rete in modo che apprendere funzioni identità risulti semplice per il modello. Considerando quindi un blocco composto da due convoluzioni separate da una ReLU, si può aggiungere una "scorciatoia" additiva che bypassa completamente le operazioni di convoluzione e si somma all'output finale. Questo viene chiamato **blocco residuale**, visibile in figura 3.6. Questo blocco facilita la rete ad apprendere funzioni identità se necessario, permettendo più profondità dal momento che, capendo quali sono i layer in eccesso, può "saltarli" imitando una rete con meno layer. Questo rende l'ottimizzazione anche più veloce e migliore perché il processo di backpropagation non prende in considerazione questi layer facilitando il

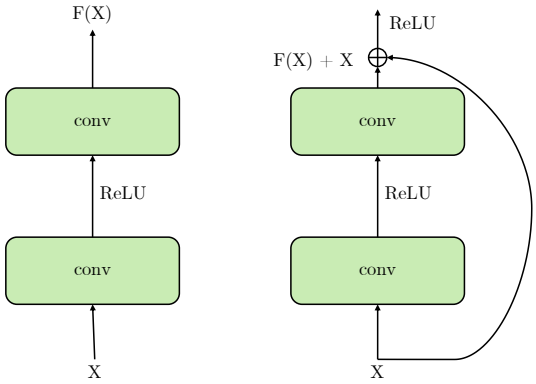


Figura 3.6: Classico blocco in una rete neurale (a sinistra) e blocco residuale (a destra).

flusso del gradiente. Gli autori hanno presentato per la prima volta una rete di 152 layer (la più profonda fino a quel momento) e, mediante un ensemble, hanno vinto la competizione di classificazione di ImageNet Large Scale Visual Recognition Challenge (ILSVRC) del 2015. In termini formali, si può definire l'output y di un blocco residuale come:

$$y = F(x, \{W_i\}) + x \tag{3.4}$$

Dove W_i sono i pesi del blocco e x è l'input. Come si può notare, questo cambiamento non introduca nessun parametro e non incrementa la complessità computazionale. Tuttavia si richiede che x e $F(x, \{W_i\})$ abbiano la stessa dimensione. Per risolvere tale problema, gli autori propongono una proiezione lineare W_s su x, da usare solamente quando le dimensioni devono combaciare. In termini formali quindi:

$$y = F(x, \{W_i\}) + W_s x \tag{3.5}$$

Un'altra soluzione che non introduce nuovi parametri sarebbe quella di lasciare che la scorciatoia senza proiezione lineare, andando tuttavia ad introdurre del padding di 0 per uguagliare le dimensioni.

Architettura

L'architettura della ResNet può essere vista come una serie di stadi, dove ogni stadio è composto da blocchi di due layer convoluzionali. Le convoluzioni dei blocchi seguono la filosofia di design della rete VGG: i layer convoluzionali hanno filtri 3x3 dove per la stessa dimensione di feature map di output, ogni layer ha lo stesso numero di filtri e se la dimensione della feature map viene dimezzata, il numero di filtri viene raddoppiato. L'operazione di *downsampling*, ovvero la riduzione della dimensione della feature map, avviene mediante layer convoluzionali con passo pari a 2. La rete termina con un'operazione di average pooling e un layer completamente connesso con 1000 neuroni in output, seguito da una funzione di Softmax per ottenere le probabilità per classe. Ci

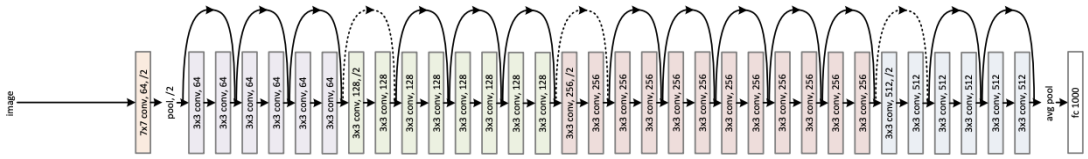


Figura 3.7: Architettura della ResNet-34. Fonte: [42].

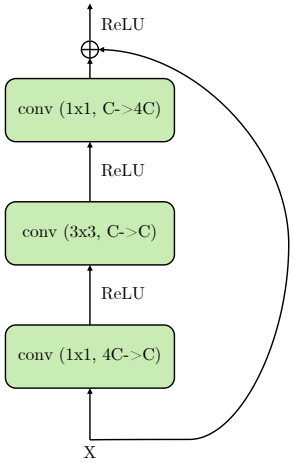


Figura 3.8: Blocco di bottleneck.

sono diverse ResNet, che differiscono per il numero di layer. Quelle con meno layer sono la ResNet-18 e la ResNet-34, l'architettura di quest'ultima è osservabile in figura 3.7. Il numero nel nome indica il numero di layer. Per le reti più profonde, quali ResNet-50, ResNet-101 e ResNet-152, il blocco di base è stato modificato usando il design chiamato **bottleneck**, osservabile in figura 3.8. Per allenare reti più profonde infatti, serve chiaramente più tempo, il blocco di bottleneck riduce il numero di FLOP da $18HWC^2$ a $17HWC^2$ dove H, W e C sono le dimensioni della feature map in ingresso al blocco. Il blocco passa ad avere tre layer di convoluzione al posto dei due presenti nel blocco residuale di base, dove il primo e l'ultimo sono convoluzioni 1x1 che si occupano di ridurre e dopo ricostruire il numero di canali. Il layer convoluzionale in mezzo è un layer con filtri 3x3 che ha una feature map con dimensioni minori in ingresso e in uscita rispetto a prima.

Concludendo, la ResNet, anche nella sua versione più grande (152) ha un livello di complessità minore delle reti VGG, con performance migliori, e risolvendo al tempo stesso il problema della profondità nelle reti. In generale, ResNet ha avuto un impatto significativo nel campo del deep learning e rimane uno strumento prezioso per la ricerca e le applicazioni di computer vision.

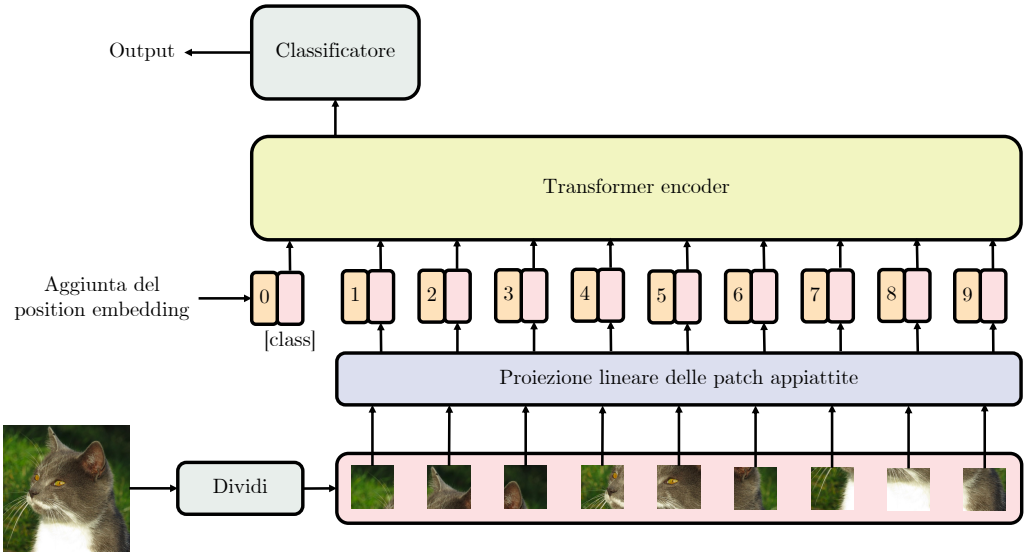


Figura 3.9: Architettura di ViT.

3.2.3 Vision Transformer

Visto il successo dei transformer nel NLP, si è pensato di applicare il meccanismo di self-attention anche al campo dell'elaborazione delle immagini, fino a quel momento dominato dalle CNN. Il risultato di questa idea è il **Vision Transformer** (ViT) [43].

Self-attention applicato alle immagini

L'obiettivo degli autori è stato quello di creare un'architettura (osservabile in figura 3.9) il più simile possibile a BERT, in modo da replicarne l'alta scalabilità e la capacità di fungere come modello pre-trainato per essere "fine-tunato" su un dataset più piccolo. Applicare alla lettera il meccanismo di self-attention vorrebbe dire mettere in relazione ogni pixel dell'immagine con ogni altro pixel, con un costo computazionale quadratico al numero di pixel nell'immagine. Per trovare un compromesso, si divide l'immagine in parti (*patch*) quadrate di dimensione (P, P). L'immagine $x \in R^{H \times W \times C}$ viene separata in una sequenza di patch appiattite $x_p \in R^{N \times (P^2 \cdot C)}$ dove H e W sono rispettivamente altezza e lunghezza dell'immagine in pixel, mentre C è il numero di canali. N è il numero risultante di patch, che risulta essere $\frac{H \cdot W}{P^2}$. Un layer di proiezione lineare porta la dimensione delle patch a D, dimensione utilizzata all'interno del Transformer. Alle rappresentazioni risultanti viene preposto un embedding apprendibile dalla rete chiamato **[class]** che ha la stessa funzione del token [CLS] in BERT, ovvero riassume l'immagine e le relazioni tra le varie patch. Vengono poi anche aggiunti dei position embedding monodimensionali per dare informazioni sulla posizione di ogni patch nell'immagine. Dopo il passaggio in un transformer encoder, il token [class] viene portato in input ad

un classificatore.
Si può osservare come la natura del modello sia molto diversa da quella delle CNN, che traggono profitto principalmente dalle strutture intrinseche a tutte immagini, discusse in 2.2.1. Sono infatti informazioni generali e localizzate a specifiche zone nell'immagine, che la convoluzione riesce a sfruttare. ViT d'altro canto permette ad ogni parte dell'immagine di mettersi in relazione con tutte le altre, non essendo confinate solamente ai pixel vicini. Le relazioni spaziali tra le patch, inoltre, sono apprese dalla rete durante il training, e non sono date a priori dai posiion embedding. Gli autori sostengono che la riduzione di questo bias può essere benefico quando si parla di immagini che non seguono i pattern spaziali assunti dalle CNN.

ViT è riuscito ad ottenere prestazioni allo stato dell'arte su diversi dataset di classificazione delle immagini, aprendo nuove possibilità per l'applicazione della tecnica rivoluzionaria di self-attention alla computer vision. La loro abilità di catturare un contesto globale gli permette di avere buone prestazioni su problemi dove la relazione tra le varie regioni delle immagini è importante. Tuttavia, ulteriori ricerche sono necessarie, per portare ViT a superare le CNN in altri problemi della computer vision.

3.3 Modello Multimodale

3.3.1 CLIP

CLIP (Contrastive Language-Image Pre-training) [44] è un modello di machine learning sviluppato da OpenAI capace di intendere la relazione tra immagini e linguaggio naturale. L'idea principale alla base di CLIP, è quella di utilizzare il testo scritto in linguaggio naturale per arricchire la capacità del modello di elaborare le immagini. Questo permette una delle caratteristiche di CLIP è la sua abilità nell'apprendimento **zero-shot**, ovvero a momento di test il modello deve poter classificare classi non osservate al momento dell'apprendimento. Per il pre-training di CLIP, è stata usata una tecnica altamente efficiente: dato un batch di N campioni composti da coppie (immagini, testo), CLIP è stato allenato per predire quale delle possibili N x N coppie composte da tutte le immagini e testo nel batch corrisponde. Il modo in cui l'allenamento di CLIP funziona è quello di andare a generare uno spazio comune multimodale nel quale proiettare linearmente gli *embedding* dei campioni ottenuti mediante due encoder: uno per le immagini e uno per il testo. CLIP cerca di massimizzare la cosine similarity dei due embedding dello stesso campione, minimizzando a sua volta quella degli embedding di campioni diversi. Lo pseudo-codice di questo processo è stato preso da [44] e qui riportato:

```
# image_encoder                ▷ ResNet or Vision Transformer
# text_encoder                 ▷ CBOW or Text Transformer
```

```
# I[n, h, w, c]                          ▷ minibatch of aligned images
# T[n, 1]                                ▷ minibatch of aligned texts
# W_i[d_i, d_e]                          ▷ learned proj of image to embed
# W_t[d_t, d_e]                          ▷ learned proj of text to embed
# t                                      ▷ learned temperature parameter

# extract feature representations of each modality
I_f = image_encoder(I)                   ▷ [n, d_i]
T_f = text_encoder(T)                    ▷ [n, d_t]

# joint multimodal embedding [n, d_e]
I_e = l2_normalize(np.dot(I_f, W_i), axis = 1)
T_e = l2_normalize(np.dot(T_f, W_t), axis = 1)

# scaled pairwise cosine similarities[n, n]
logits = np.dot(I_e, T_e^T) * np.exp(t)

# symmetric loss function
labels = np.arange(n)
loss_i = cross_entropy_loss(logits, labels, axis=0)
loss_t = cross_entropy_loss(logits, labels, axis=1)
loss = (loss_i + loss_t)/2
```

L'ottimizzazione avviene mediante una cross-entropy loss sui punteggi di similarità ottenuti. Come encoder per le immagini, CLIP è testato allenato sia con ResNet che con ViT, con delle leggere modifiche. Mentre per l'encoder di testo è stato utilizzato un Transformer.

Apprendimento zero-shot

Come già anticipato, la caratteristica chiave di CLIP è quella di poter effettuare apprendimento zero-shot. Tuttavia, come specificato dagli autori, la loro interpretazione dell'argomento è più ampia perché la considerano come la capacità del modello di generalizzarsi su dataset non visti a tempo di training, in modo da valutare quanto bene il modello possa adattarsi a risolvere diversi compiti, diversi da quello di accoppiamento eseguito in fase di allenamento. Questo tipo di trasferimento ad altri problemi viene quindi chiamato **trasferimento zero-shot**. Per la classificazione di immagini zero-shot, CLIP opera nel seguente modo: Per ogni dataset considerato, vengono utilizzati i nomi di tutte le classi presenti nel dataset come insieme di possibili testi e si prevede la coppia (immagine, testo) più probabile secondo CLIP. Per fare ciò, si calcola prima la

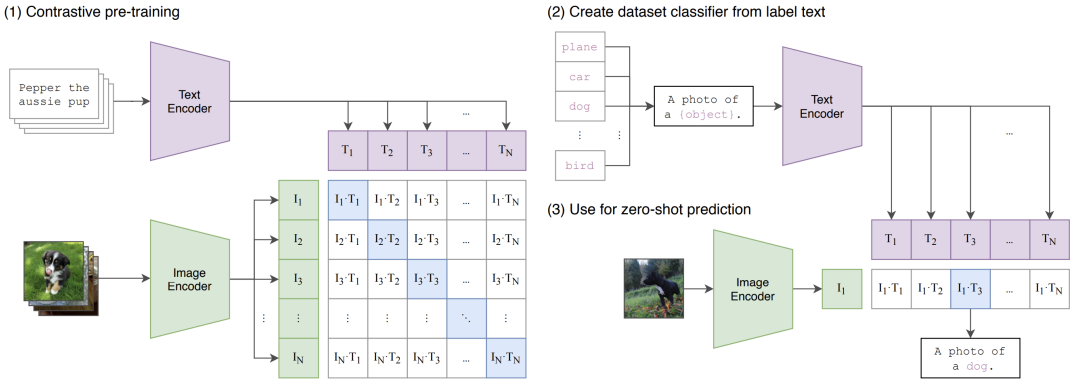


Figura 3.10: Classificazione zero-shot eseguita da CLIP. Fonte: [44].

rappresentazione vettoriale dell'immagine e quella del testo mediante rispettivi encoder. Successivamente, si calcola la similarità coseno tra queste rappresentazioni, si scala il risultato con un parametro di temperatura τ e si normalizza tramite una funzione di softmax per ottenere una distribuzione di probabilità. Una rappresentazione visuale di questo processo è osservabile in figura 3.10. Come si può osservare, quindi, CLIP è capace di adattarsi al problema di classificazione delle immagini nonostante questo non sia stato preso in considerazione in fase di allenamento. Un problema da risolvere con CLIP è che a volte i dataset di classificazione non forniscono il nome della classe, ma semplicemente un indice numerico che la indica, rendendo in questo caso il trasferimento zero-shot. Anche se il dataset indicasse la classe però, spesso questa è una singola parola, e inoltre nasce il problema della polisemia, che in linguistica è la capacità di una parola di esprimere diversi significati. Un esempio si trova nel dataset Oxford-IIIT [45], ovvero un dataset per la classificazione di razze di cani. In questo dataset la classe "boxer" si riferisce chiaramente alla razza del cane, ma senza contesto potrebbe anche essere riferito alla professione atletica. Questo ha portato gli autori di CLIP a sottolineare l'importanza del **prompt engineering**, dove invece di prendere semplicemente il nome della classe, si aggiunge del testo per dare contesto alla frase, oltre che allungare la frase e avvicinarsi al metodo di allenamento di CLIP per ottenere risultati migliori. Per questo, invece della classe "boxer", questa diventa "A photo of a boxer, a type of pet", che in italiano è: "La foto di un boxer, un tipo di animale domestico". Il trasferimento zero-shot permette a CLIP di essere anche più robusto al **domain shift**. Il domain shift è un fenomeno che si verifica quando la distribuzione dei dati di training e quella di testing differiscono. In altre parole, è un cambio del processo sottostante di generazione tra il dominio della sorgente (dove il modello è stato allenato) e il dominio di target (dove il modello è testato). I modelli per i quali non sono state pensate soluzioni, di solito, presentano un calo di prestazioni notevole a fronte di un cambio di dominio, questo perché il modello si è adattato a pattern e caratteristiche

specifiche dei dati di training, tuttavia la maggior parte di quest'ultime potrebbero non definire i dati del dominio di target. Essendo CLIP allenato con l'intento di essere un modello con trasferimento zero-shot, è intrinseco che sia anche robusto al domain-shift. Infatti, CLIP riesce a migliorare del 74.4%, rispetto a ResNet-101, la differenza tra l'accuratezza su ImageNet e quella su ImageNet-A (ImageNet Adversarial), che è un dataset creato appositamente per valutare la robustezza dei modelli di deep learning alle immagini perturbate.

Limiti

Nonostante CLIP ottenga ottimi risultati se confrontati con la baseline considerata nell'articolo originale, ovvero ResNet-50 con un semplice classificatore lineare, le sue performance sono comunque al di sotto di modelli più avanzati. Gli autori stimano che perché CLIP raggiunga performance alla pari con quelle allo stato dell'arte, bisogni avere una potenza computazionale che sia mille volte superiore a quella attuale, rendendo il problema intrattabile con l'hardware attuale. Un altro limite si riscontra proprio nel trasferimento zero-shot, infatti nonostante abbia buoni risultati su questo campo, le performance del modello sono comunque più basse di modelli specifici per il problema considerato. Inoltre CLIP presenta problemi nel differenziare modelli di macchine. specie di fiori e varianti di aerei, presentando anche difficoltà nell'affrontare problemi più astratti, ad esempio quello di contare il numero di oggetti in un'immagine. Inoltre, per task più complesse, come misurare la distanza in una foto, le sue performance sono quasi casuali. Come detto in precedenza, CLIP riesce a generalizzare bene immagini che sono fuori dalla distribuzione di training, ma nonostante ciò non riesce a farlo per dati che sono completamente fuori distribuzione. Un esempio si ritrova nel problema di OCR (Optical Character Recognition), ovvero il problema del riconoscere e convertire testo presente nelle immagini in testo digitale. CLIP ottiene buoni risultati dal momento che il dataset su cui è stato allenato contiene del testo nelle immagini. Tuttavia, il modello ottiene solamente l'88% di accuratezza sulle cifre del dataset MNIST [46], ottenendo risultati peggiori della baseline, una semplice regressione logistica sui pixel grezzi. La causa sta nel fatto che CLIP è stato allenato con l'assunzione che usando un vasto dataset di training, qualsiasi dato di test si trovi nella distribuzione che CLIP può gestire. Tra le altre limitazioni presenti, CLIP condivide con molti altri modelli il problema del **bias sociale**, infatti essendo allenato su un vasto dataset di coppie immagine-testo prese da internet e, pertanto, non filtrate. Questo porta il modello ad assorbire pregiudizi o discriminazioni presenti nei dati, per esempio andando ad associare alcune caratteristiche, azioni o occupazioni basandosi su genere, etnia o età, anche se non vi è alcuna correlazione reale tra di loro.

Concludendo, CLIP è un modello multimodale con un grande potenziale, ma che possiede ancora molti limiti, in parte risolvibili da un allenamento con dataset più vasti.

La sua capacità di associare immagini e testo è stata fondamentale nel nostro studio, essendo la correlazione tra testo e immagini una parte importante nel riconoscimento della veridicità di una notizia.

Capitolo 4

Definizione del problema e descrizione dei dati

4.1 Formulazione del problema

L'obiettivo di questo studio consiste nell'esplorare una metodologia per il riconoscimento automatico di notizie false in un ambiente multimodale comprendente sia testo che immagini. Più nello specifico, l'obiettivo del problema è quello di valutare la veridicità di un contenuto che combina elementi visivi e testuali (come ad esempio un post su un social media o un articolo di giornale), determinando la probabilità che la notizia sia vera o falsa. Il tema predominante delle notizie in questione è la situazione di conflitto tra la Russia e l'Ucraina. Il problema è stato posto per la competizione **MULTI-Fake-DetectiVE (MULTImodal Fake News Detection and VErification)** presentata da EVALITA nel contesto dell'etica computazionale.
Ci si riferirà alla task appena descritta come "task primaria", data la presenza di una task secondaria, non affrontata direttamente in questo studio, ma sulla quale il modello finale sarà testato per valutare la sua robustezza e capacità di adattamento a diversi contesti. L'obiettivo della task secondaria è quello di valutare la relazione tra modalità visiva e testuale per identificare come queste possano portare a interpretazioni sbagliate. Ad esempio, una notizia ingannevole potrebbe contenere un'immagine che presenta dei rifiuti accompagnata da un testo che riporta informazioni su delle proteste per il cambiamento climatico, suggerendo che sia colpa dei manifestanti.

4.2 Dataset

4.2.1 Dataset task primaria

Il dataset utilizzato, è costituito da post presi da Twitter e articoli di notizie di giornale che riguardano uno o più eventi reali noti per essere stati oggetto di notizie false. Nello specifico, sono concentrati sulla guerra tra Ucraina e Russia del 2022. Il dataset è composto da una parte per il training e una per il testing, che da ora in poi saranno denominati rispettivamente con training set e test set. Nella competizione la classifica è redatta basandosi sulla metrica F1-weighted ottenuta sul test set. Per il training set sono stati forniti 1057 campioni disponibili per il download. Tuttavia, come specificato dagli organizzatori dell'evento, la dimensione del dataset varia dal momento nel quale i partecipanti scaricano i dati. Questo perché i dati, essendo presi direttamente dal web, possono essere eliminati dagli autori o dalle piattaforme su cui questi sono stati pubblicati. Di fatto, il training set su cui sono stati eseguiti gli esperimenti su questo studio è composto da 1052 campioni etichettati, ma dopo aver rimosso i duplicati, questo numero è sceso a 908. D'altra parte il test set è composto da 199 campioni non etichettati. Il dataset è composto dai seguenti campi:

- **ID** Una stringa numerica che ha la funzione di identificatore unico per il campione
- **URL** URL del campione dal quale sono stati estratti testo e immagini
- **Type** Tipo di campione, che può essere esclusivamente di due tipi: *tweet* e *article*. Il tipo tweet sta ad indicare un campione prelevato da twitter.com, mentre il tipo article indica un vero e proprio articolo di giornale. Gli articoli sono prelevati da diverse testate giornalistiche.
- **Label** Etichetta del campione, disponibile solamente nei dati di training, e indica se la notizia è "Certamente Falsa" (CF), "Probabilmente Falsa" (PF), "Probabilmente Vera" (PV), "Certamente Vera" (CV). È un dato numerico che assume valori da 0 a 3, dove la corrispondenza valore-etichetta è espressa nella tabella 4.1.

Etichetta	Valore numerico	Numero campioni
Certamente Falsa	0	149
Probabilmente Falsa	1	200
Probabilmente Vera	2	403
Certamente Vera	3	156

Tabella 4.1: Corrispondenza etichetta - valore numerico nel dataset.

Il dataset presenta uno sbilanciamento significativo delle etichette, con l'etichetta 2 che supera di gran lunga le altre in numero. In particolare, l'etichetta 2 compare

più del doppio delle volte rispetto a ciascuna delle altre etichette, come è osservabile in figura 4.1. Inoltre, la distribuzione sbilanciata delle etichette persiste anche

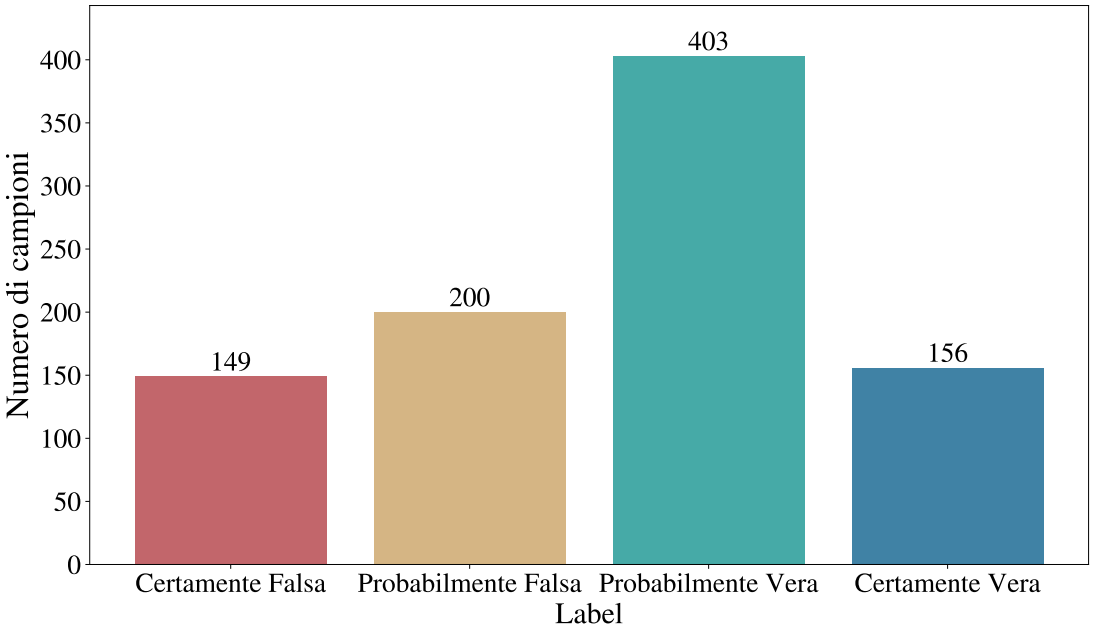
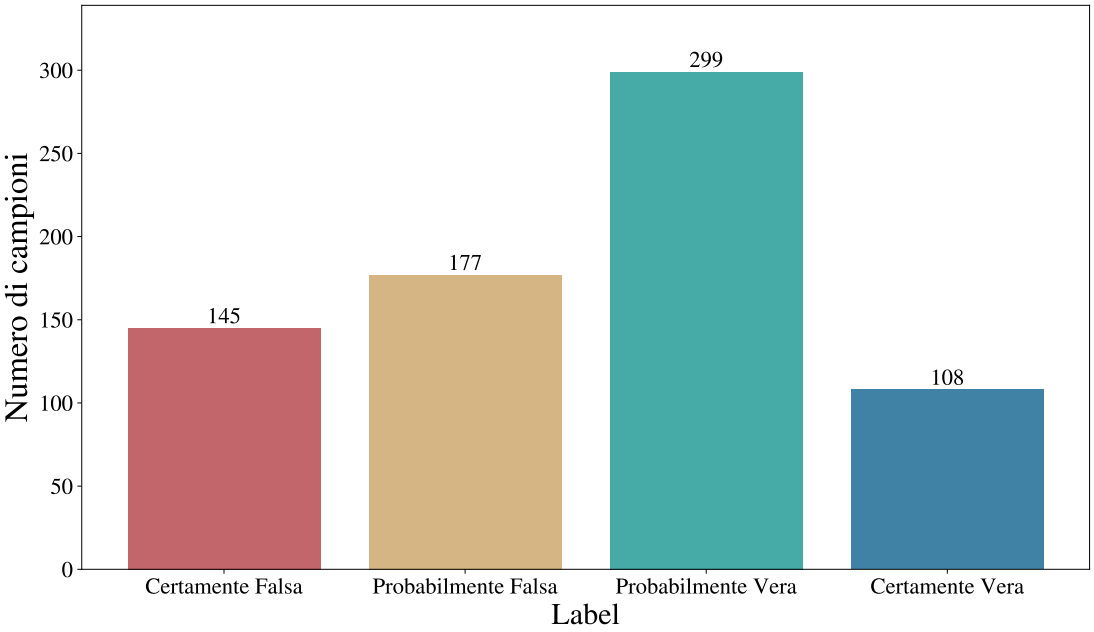


Figura 4.1: Distribuzione delle etichette nel dataset.

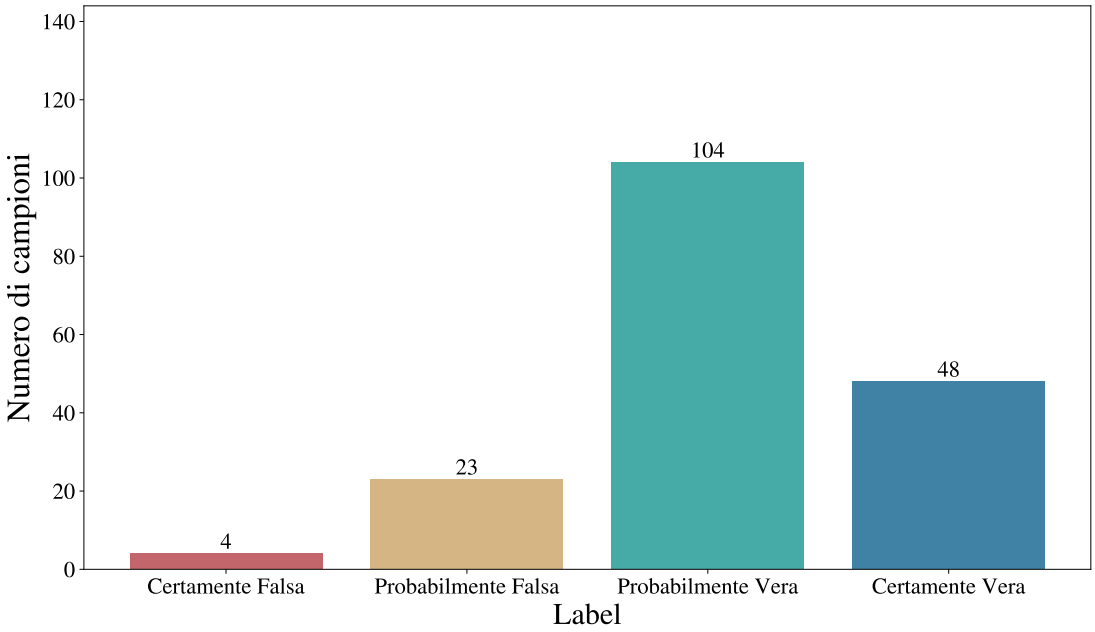
quando si considerano separatamente i tweet e gli articoli del dataset. Come illustrato nella figura 4.2, si può osservare che la label 0 è altamente sottocampionata per gli articoli, con soli 4 campioni rispetto ai 104 della label 2. In generale, includendo nella categoria falsi i campioni con label CF e PF, mentre con veri quelli con label PV e CV, possiamo notare che per gli articoli sono presenti molti meno campioni falsi (27) rispetto a quelli veri (152).

- **Date** Data di creazione del campione. Non tutti gli articoli presentano una data di creazione, perché non forniti di questa informazione. Tra tutti i campioni con data presenti nel dataset, che sono in totale 860, il $99,7\%$, ovvero 858, sono del 2022, mentre le restanti 2 sono del 2014 e del 2021. Non tenendo in considerazione queste ultime, la distribuzione per mese dei campioni del 2022 è osservabile nella figura 4.3

- **Text** Testo completo del campione. Nel caso in cui si tratti di un tweet, è seguito nuovamente dall'URL dal quale è stato prelevato. Il dataset presenta uno sbilanciamento significativo della lunghezza dei testi, con la lunghezza degli articoli che supera notevolmente quella dei tweet. Questo fenomeno è evidenziato nelle statistiche di base riportate nella tabella 4.2. Per analizzare come la lunghezza dei testi sia correlata alle etichette di classificazione, un grafico di distribuzione delle etichette dei sample con le rispettive lunghezze è stato riportato nella figu-



(a) Distribuzione dei tweet per label.



(b) Distribuzione degli articoli per label.

Figura 4.2: Distribuzione delle etichette per tipo.

ra 4.4. Un'ulteriore analisi che è stata eseguita è quella relativa alla presenza di lettere maiuscole all'interno dei testi, la figura 4.5 mostra come per gli articoli, la percentuale di lettere maiuscole nei testi mostra una certa stabilità per ogni etichetta, come indicato dalla media e dalla deviazione standard relativamente costanti. D'altro canto, per i tweet, la classe CF presenta un valore medio mol-

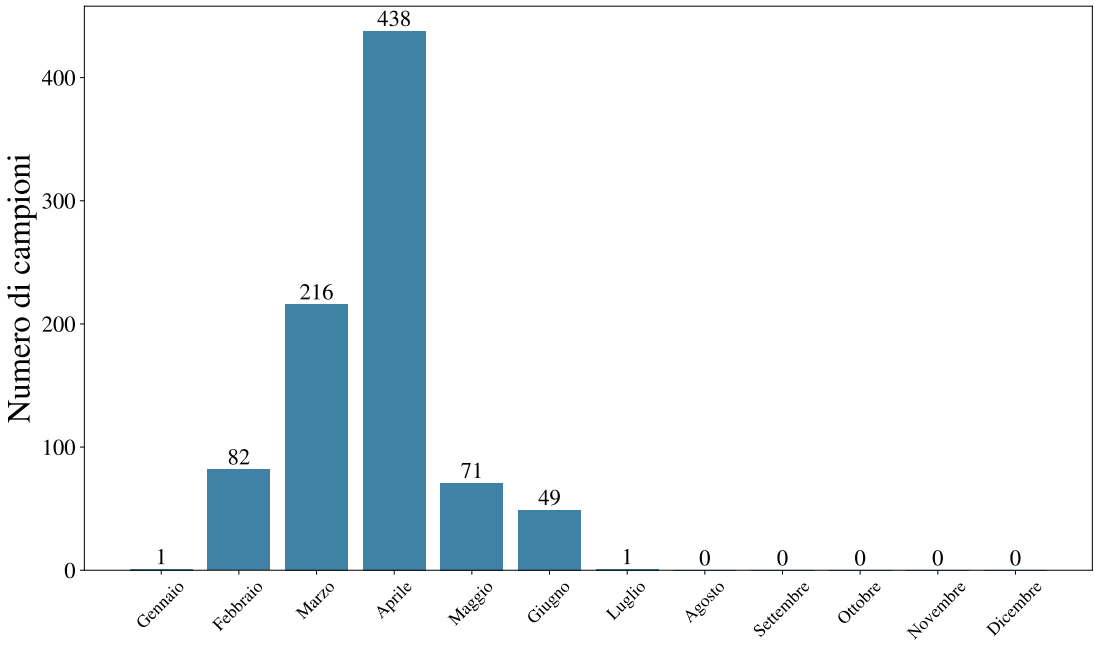
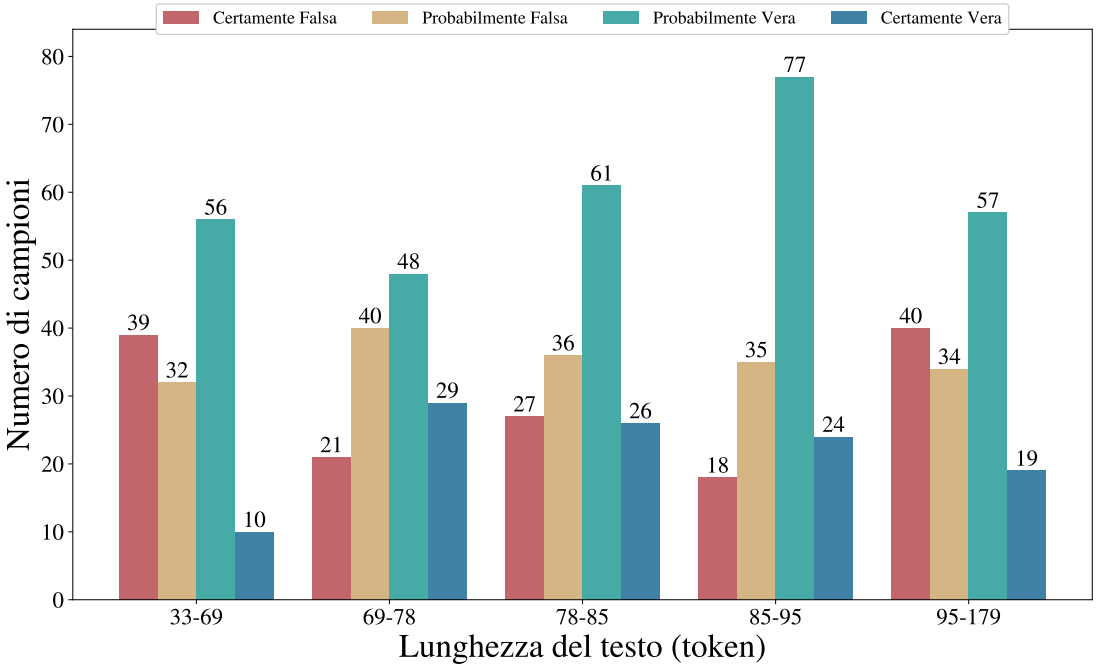


Figura 4.3: Distribuzione dei campioni per mese del 2022.

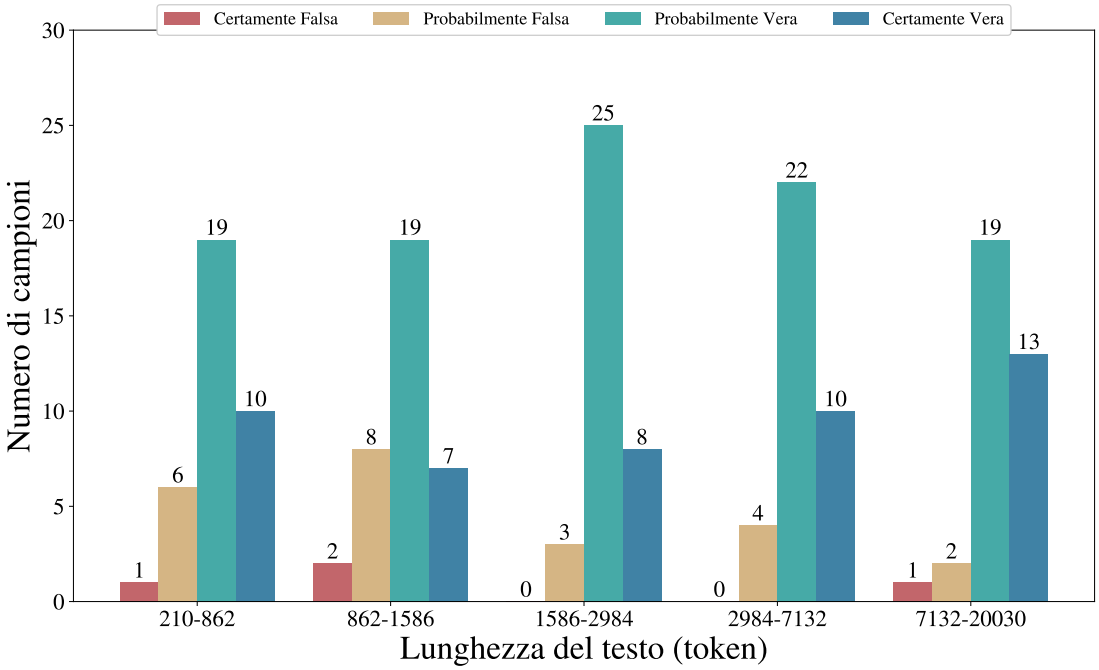
	Tutte	Cert. Falsa	Prob. Falsa	Prob. Vera	Cert. Vera
Tweet (caratteri)					
Max	414	414	357	329	304
Min	85	85	100	104	132
Media	259	252	263	260	259
Tweet (token)					
Max	179	179	133	146	120
Min	33	33	39	36	37
Media	82	85	82	82	82
Articoli (caratteri)					
Max	92195	38210	38521	92195	49486
Min	968	1913	1883	968	1271
Media	16512	13118	11146	16486	19424
Articoli (token)					
Max	20030	8030	9078	20030	11015
Min	210	401	382	210	262
Media	3608	2766	2450	3579	4295

Tabella 4.2: Lunghezza massima, minima e media dei testi dei campioni del dataset divisi per tweet e articoli. La lunghezza è stata riportata sia in caratteri che in token, estratti con il tokenizer fornito da Italian BERT cased.

to più elevato rispetto alle altre classi (21.7%) e una deviazione standard molto elevata (28.1%). Questi risultati suggeriscono che la presenza o l'assenza di testo in maiuscolo possa essere un indicatore utile per la classificazione dei tweet, in particolare per la classe CF. Inoltre, si osserva un abbassamento della media e



(a) Distribuzione etichette per tweet divise per lunghezza

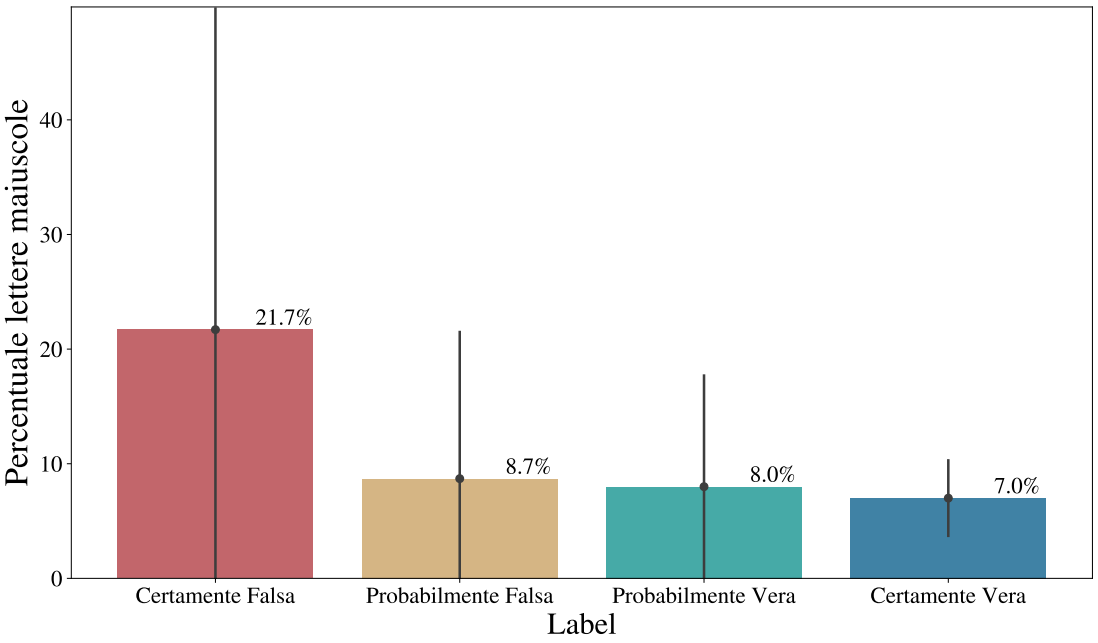


(b) Distribuzione etichette per articoli divise per lunghezza

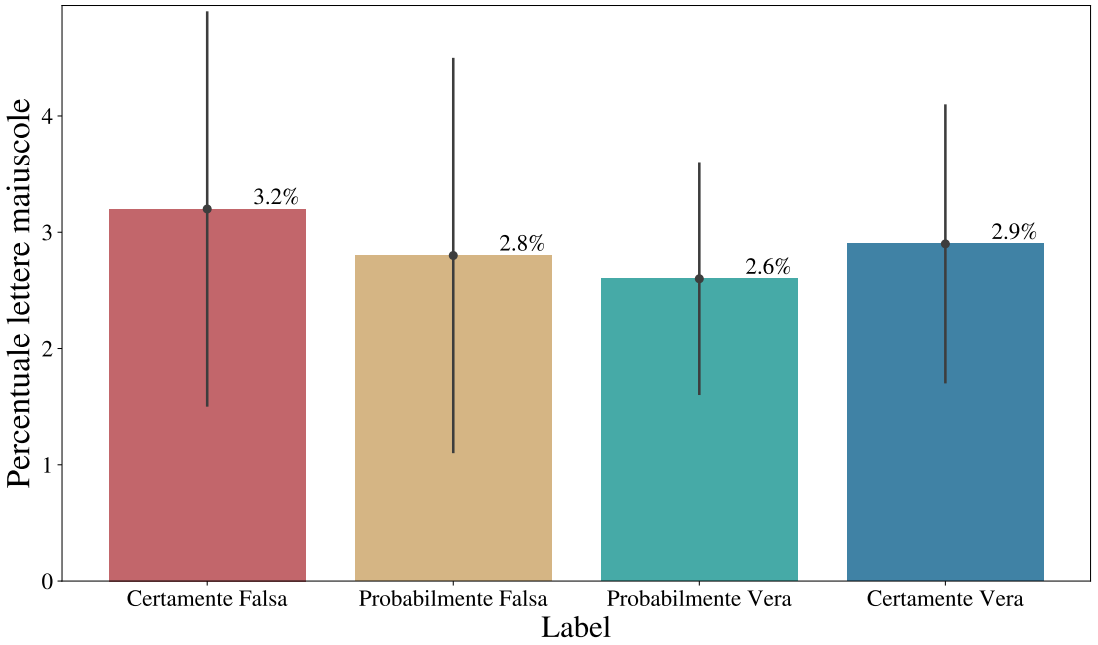
Figura 4.4: Distribuzione etichette rispetto alle lunghezze dei testi divise 5 bin con uguale frequenza.

della deviazione standard con l'aumentare della veridicità delle notizie.

- **Media** Lista di stringhe che rappresentano i nomi delle immagini che compongono



(a) Tweet



(b) Articoli

Figura 4.5: Media (grafico a barre) e deviazione standard (linee verticali), della percentuale di lettere maiuscole nei testi per ogni label.

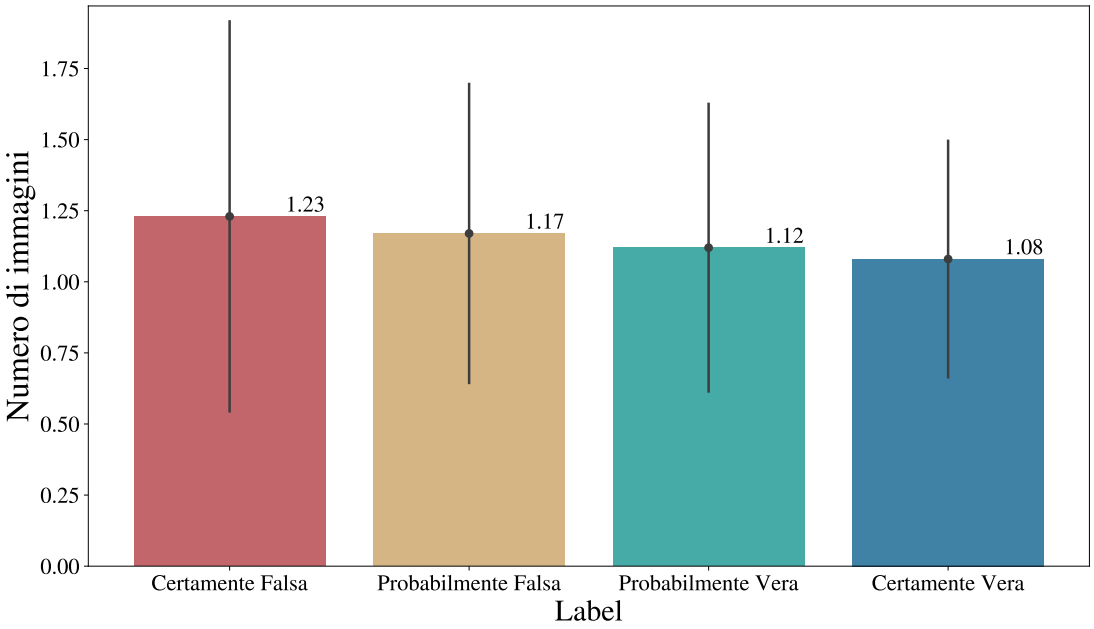


Figura 4.6: Media (grafico a barre) e deviazione standard (linee verticali), del numero di immagini nei campioni per ogni label.

Statistica	Training set	Test set
n° campioni	908	193
n° tweet	729	142
n° articoli	179	51
n° medio immagini	1.14	1.26
n° medio immagini [CF,PF,PV,CV]	[1.23, 1.17, 1.12, 1.08]	[1.69, 1.27, 1.20, 1.24]
n° massimo immagini	4	4

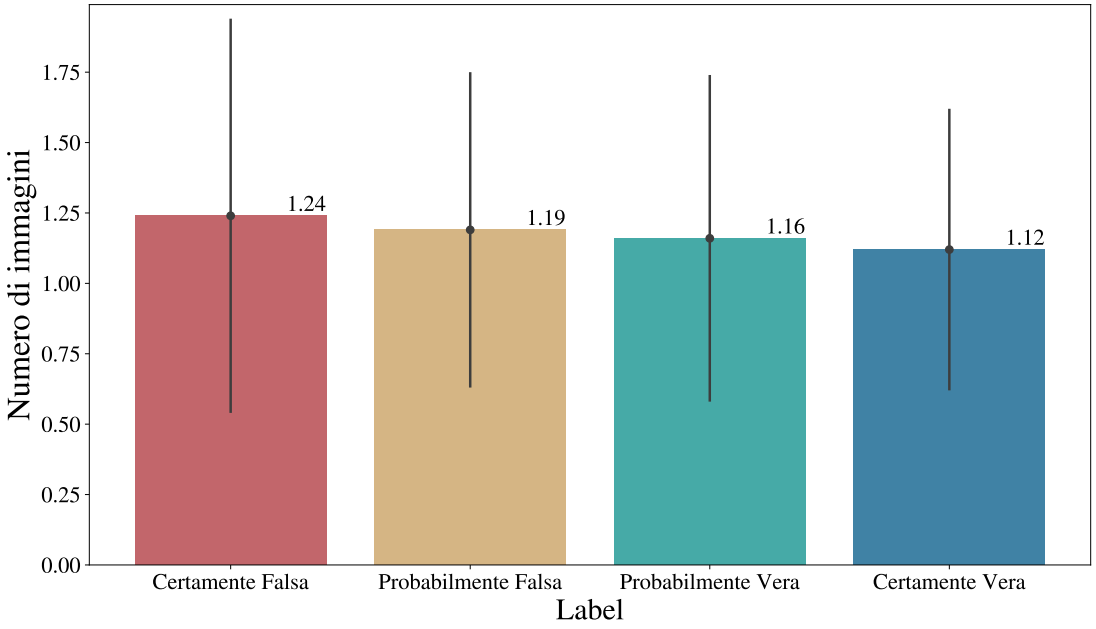
Tabella 4.3: Statistiche di base per train e test set (task principale).

il campione. Un campione ha sempre un solo testo mentre le immagini possono essere molteplici, ma ne è sempre presente almeno una. Media e deviazione standard per etichetta del numero di immagini sono riportate in figura 4.6, mentre è possibile osservare le stesse statistiche divise per tweet e articoli in figura 4.7. Si può notare come per gli articoli sia sempre presente una e una sola immagine. Per i tweet invece si osserva che la media e la deviazione standard sono leggermente più alti per la classe CF, ma non di un valore significativo.

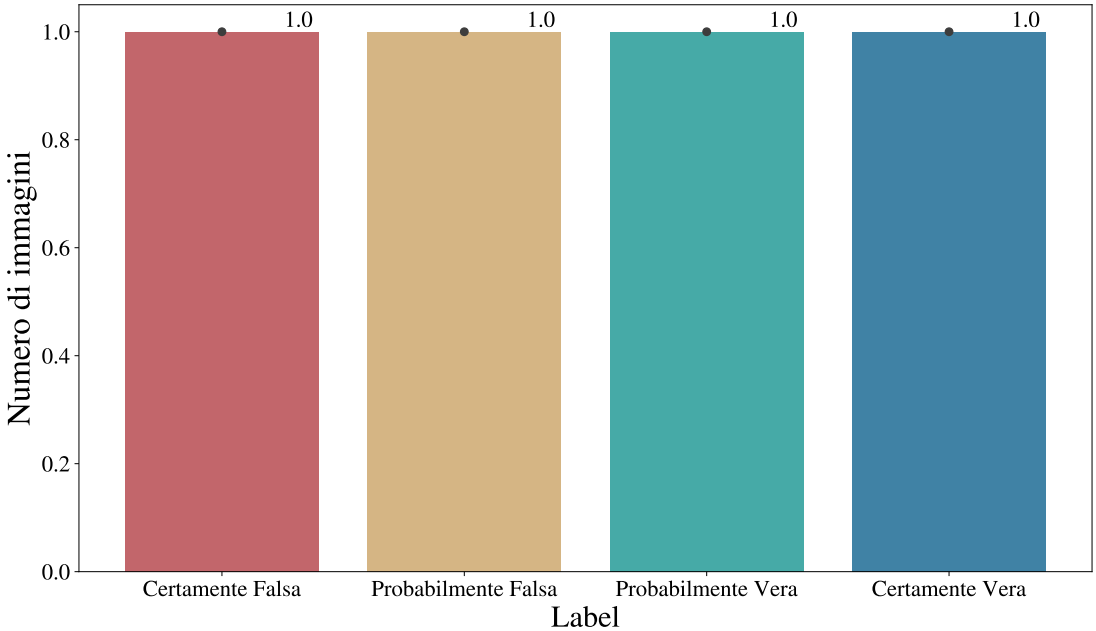
Le statistiche di base del test set sono riportate in tabella 4.3 confrontate con le statistiche del training set.

4.2.2 Dataset task secondaria

Per quanto riguarda la task secondaria, anche qui sono presenti un training set e un test set. Gli attributi sono gli stessi della task primaria, con l'eccezione delle etichette



(a) Tweet



(b) Articoli

Figura 4.7: Media (grafico a barre) e deviazione standard (linee verticali), del numero di immagini nei campioni per ogni label.

Statistica	Training set	Test set
n° campioni	1309	219
n° tweets	866	166
n° articoli	443	53
n° medio immagini	1.12	1.25
n° medio immagini [I,NC,NI]	[1.22, 1.04, 1.14]	[1.33, 1.04, 1.36]
n° massimo immagini	4	4

Tabella 4.4: Statistiche di base per train e test set (task secondaria).

che passano da quattro a tre. Le etichette sono le seguenti:

- **Ingannevole (I)**: Immagini e testo sono combinate in modo da rendere la notizia ingannevole per il lettore
- **Non correlati (NC)**: Immagini e testo sono completamente scorrelati
- **Non ingannevole (NI)**: Immagini e testo non sono combinate in modo da rendere la notizia ingannevole

Si riportano in tabella 4.4 le statistiche riassuntive di questo dataset.

Capitolo 5

Metodologia e Progettazione sperimentale

In questo capitolo forniremo una panoramica sulle opere correlate alla task di riconoscimento delle fake news, una descrizione dettagliata delle operazioni eseguite sul dataset, delle metodologie adottate per gli esperimenti, dei criteri che hanno guidato le scelte effettuate e dei dettagli relativi all'addestramento del modello. Considerato un campione x, gli embedding di dimensione d di una modalità generica k saranno indicate con $f^{(k)} = [f_0^{(k)}, ..., f_d^{(k)}]$. Con la concatenazione di feature di due modalità k_1 e k_2 si intende un vettore $[f_0^{(k_1)}, ..., f_{d_1}^{(k_1)}, f_0^{(k_2)}, ..., f_{d_2}^{(k_2)}]$.

5.1 Opere correlate

Gli approcci iniziali per l'identificazione di notizie false si sono basati principalmente sull'analisi del testo. In particolare alcune si sono basati sulle classiche tecniche di Machine Learning utilizzando statistiche [47] [48]. Altri invece hanno utilizzato modelli di Deep Learning, come FakeBERT [49] dove si utilizzano BERT e una CNN in parallelo per estrarre informazioni sul testo.
In questo studio ci si è concentrati sull'utilizzo di deep learning, utilizzando anche in questo caso modelli allo stato dell'arte come BERT. Tuttavia, il testo è stato integrato con elementi visivi al fine di migliorare la classificazione, creando così un problema di natura multimodale. Sono già stati proposti modelli multimodali per l'identificazione di fake news. Questi metodi utilizzano per la maggior parte BERT per l'estrazione di feature testuali, ma variano molto per l'ottenimento di quelle visuali. Tra questi troviamo SpotFake [50] che utilizza VGG per le immagini, CB-Fake [51] che invece utilizza CapsNet, e lo studio presentato in [52], dove le informazioni tra le diverse modalità vengono fuse in maniera progressiva nella rete. In questo studio si è continuato ad utilizzare BERT come modello testuale, ma quest'ultimo è stato accoppiato a ResNet

per gestire le immagini. Inoltre, non solo si è cercato di estrarre le rappresentazioni di base, ma si è pensato di metterle direttamente in relazione l'una con l'altra per poi arricchirle mediante rappresentazioni più complesse, come quelle del sentimento per il testo, o basate sulla DFT per le immagini. L'analisi del sentimento per la classificazione di fake news si è già rivelata di successo, come evidenziato in [53].
Infine, tutti i metodi sopra citati si limitano a classificare le notizie come "Vere" o "False", tuttavia questa distinzione netta è difficilmente osservabile. Un approccio più conservativo nella classificazione è già stato seguito nel dataset proposto in [54], dove le label sono: "Quasi certamente vera", "Probabilmente Falsa", "Quasi certamente Falsa", e "Non riesco a decidere". Possiamo tuttavia notare che queste label mancano di simmetria. Per questo studio si sono utilizzate label che classificano livelli intermedi di veridicità delle notizie in maniera simmetrica, definendo le label "Certamente Falsa", "Probabilmente Falsa", "Probabilmente Vera", "Certamente Vera".

5.2 Suddivisione del dataset

All'inizio della sperimentazione è stato fornito esclusivamente il training set da 1052 campioni, perciò il primo passo è stato quello di pre-processarlo andando ad eliminare i duplicati. Dopo l'eliminazione dei duplicati, i 908 campioni restanti sono stati mescolati casualmente suddivisi in training set e validation set, seguendo una proporzione rispettivamente dell'80% e 20%. La suddivisione è stata effettuata in modo stratificato, considerando sia le etichette che il tipo di campione (tweet o articoli), come osservabile in figura 5.1. Questo approccio è stato adottato per avere nel validation set una proporzione di label e tipo all'incirca uguale a quella del training set e ottenere risultati imparziali ed equilibrati.

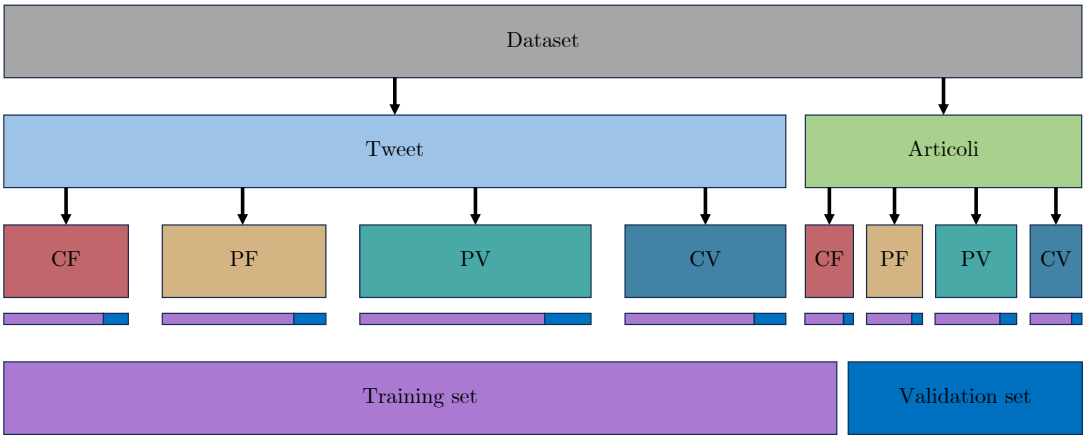


Figura 5.1: Processo di suddivisione stratificata per label e tipo. Nel dataset si sono individuati i tweet e gli articoli con etichette CF, PF, PV, CV, si è poi preso l'80% di ogni categoria per formare il training set. Il restante 20% compone il validation set.

5.3 Baseline

Sono state condotte un'ampia serie di sperimentazioni utilizzando diverse architetture e modelli al fine di acquisire conoscenze approfondite e orientare le azioni future, di seguito si riportano tutti i test effettuati e nella prossima sezione saranno ne saranno riportati i risultati.
I primi test sono stati eseguiti per ottenere delle baseline, ovvero dei punti di riferimento che saranno utilizzati per il confronto con i modelli successivi. Le baseline selezionate consistono in modelli che prendono in considerazione solo il testo, solo le immagini o entrambi, al fine di confrontare come ciascuna modalità contribuisca alla classificazione. Per il fine-tuning di tali modelli sono stati presi gli embedding delle backbone ed è stato piazzato un classificatore con dei layer completamente connessi, e una softmax ai 4 output per ottenere la probabilità per ogni classe.
Baseline testuali Per il testo sono stati testati dei modelli allo stato dell'arte del NLP pretrainati su dati italiani, in particolare: Italian BERT[1], versione italiana di BERT, BART-IT [55], versione italiana di BART [56], GilBERTo[2], modello ottenuto mediante pretrain su dati in italiano usando l'architettura di RoBERTa [57] l'approccio di tokenizzazione di CamemBERT [58].

Baseline visuali Per quanto riguarda le immagini invece sono state considerate ResNet e ViT.

Baseline multimodali Come modelli multimodali, infine, è stato considerato un modello che concatena insieme gli embedding finali di BERT e di ResNet, ma anche quelli testuali e visuali della versione italiana di CLIP[3].

Successivamente alla fase di sperimentazione delle baselines, sono state condotte ulteriori analisi su tali baselines al fine di valutare le diverse versioni di ResNet, comprese ResNet-18, ResNet-50, ResNet-101 e ResNet-152.

5.4 Strategie di gestione del testo

Per quanto riguarda la parte testuale, come osservato nella sezione 4.2, ci sono due aspetti da considerare, ovvero la capitalizzazione dei caratteri nel testo e la lunghezza di quest'ultimo. Per quanto riguarda il primo problema, è stata sperimentata la versione **uncased** di Italian BERT[4], ovvero una versione del modello che tratta le lettere maiuscole e minuscole allo stesso modo, ignorando la loro differenza e riducendo il dizionario. D'altro canto, la versione **cased** della baseline mantiene la differenziazione tra caratteri

[1] https://huggingface.co/dbmdz/bert-base-italian-xxl-cased
[2] idb-ita/gilberto-uncased-from-camembert
[3] https://huggingface.co/clip-italian/clip-italian
[4] https://huggingface.co/dbmdz/bert-base-italian-xxl-uncased

maiuscoli e minuscoli, aspetto che potrebbe risultare utile considerando la specificità del nostro problema. Per quanto riguarda la seconda questione, il problema emerge dal momento che la versione di BERT utilizzata supporta una lunghezza massima di token pari a t_{max} (compresi il token di inizio e di fine inseriti direttamente da BERT). Tale limitazione non costituisce un ostacolo per i tweet, ma risulta invece problematica per gli articoli, che presentano una lunghezza media in token pari a 3608, come osservabile in tabella 4.2. Sono state perciò analizzate quattro strategie

- La strategia **head** equivale comportamento di default di BERT, ovvero considera esclusivamente i primi t_{max} token, scartando completamente il resto del testo. Preso un testo t, ne vengono estratti m token $T = [T_1, T_2, ..., T_m]$, supponendo $m > t_{max}$, la strategia head considererà solamente $\hat{T} = [T_1, ..., T_{t_{max}}]$.

- Un'altra strategia, denominata **tail** è stata invece quella di prendere solamente la parte finale del testo, cioè gli ultimi t_{max} token. La strategia tail considererà quindi i token $\hat{T} = [T_1, T_{m-t_{max}+2}, ..., T_m]$. Ricordiamo che bisogna aggiungere il token di [CLS], per questo anche T_1 è presente in $\hat{T}$

- La terza, la **head-tail**, è invece la via di mezzo tra le prime due. Sono stati presi i $t_{max}/2$ token iniziali e i $t_{max}/2$ token finali. La strategia head-tail considererà quindi i token $\hat{T} = [T_1, ..., T_{t_{max}/2}, ..., T_{m-t_{max}/2+1}, ..., T_m]$

- Infine l'ultima è la più pesante computazionalmente, ma fiduciosamente anche la più performante, ed è stata chiamata **Long** che applicata a BERT possiamo denominare Long BERT (LBERT per abbreviare). Questa strategia per gestire lunghe sequenze di testo prende ispirazione da [59]. Il testo viene convertito in token, vengono poi presi sequenze di token da t_{max} e aggiunto del padding all'ultima sequenza nel caso in cui i token iniziali non sono divisibili per t_{max}. Ciascuna di queste sequenze viene elaborata da BERT, e il risultato dell'elemento [CLS] di ogni sequenza viene poi sottoposto a un layer completamente connesso. Successivamente, viene calcolata la media di tutti i vettori ottenuti.
  Definendola formalmente, a T vengono rimossi il primo e l'ultimo token perché sono quelli aggiunti da BERT per riconoscere l'inizio e la fine. Il vettore rimanente $T = [T_2, ..., T_{m-1}]$ viene diviso in p parti di lunghezza $t_{max} - 2$. Ad ogni parte $\hat{T}^{(i)} = [T_1^{(i)}, ..., T_{t_{max}-2}^{(i)}]$ con $i = 1, ..., p$ viene aggiunto il token iniziale e finale per ottenere una lista di lunghezza finale t_{max}. Ognuna di queste liste viene processata da BERT ottenendo per ogni token gli embedding $e_i = [e_{i,1}, ..., e_{i,t_{max}}]$. Ogni elemento del vettore e è a sua volta un vettore di dimensione $d = 768$. L'embedding finale $f^{(t)}$ utilizzato si ottiene calcolando $f^{(t)} = media(e_{1,1}, e_{2,1}, ..., e_{p,1})$. Il processo è osservabile in figura 5.2.

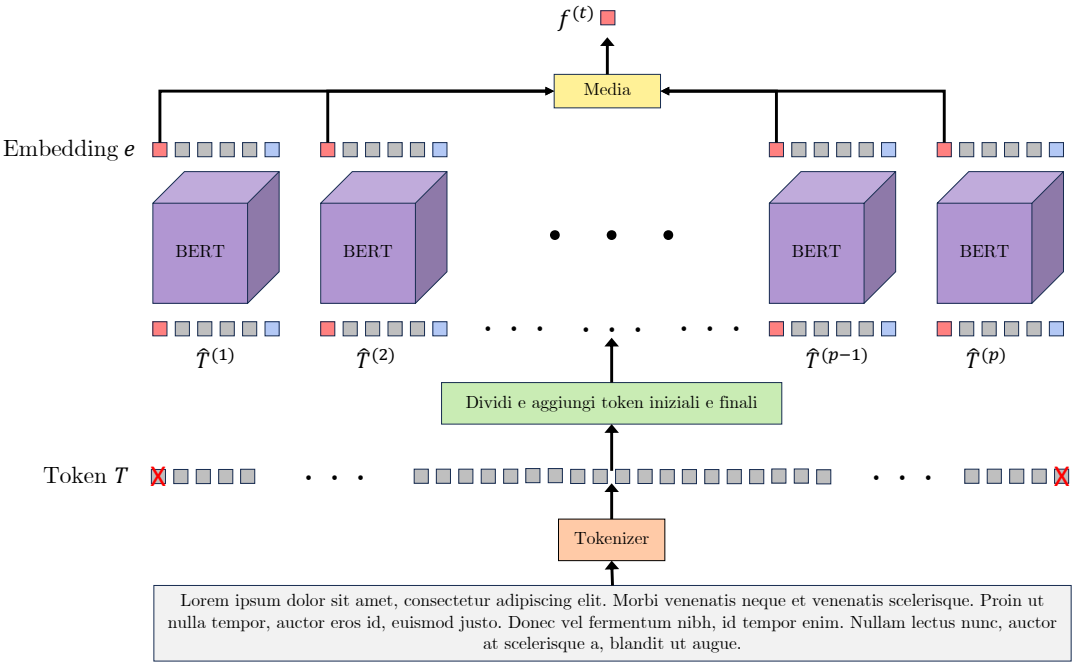


Figura 5.2: Strategia LBERT per sequenze di token più lunghe della lunghezza massima con cui lavora BERT.

5.5 FND-CLIP

Dato il fallimento dei precedenti approcci nel migliorare le prestazioni, si è passato ad utilizzare il framework definito in [60], chiamato FND-CLIP, progettato appositamente per il task di riconoscimento delle notizie false. L'architettura di FND-CLIP si può osservare in figura 5.3. L'idea alla base è quella di utilizzare BERT, CLIP e ResNet in maniera congiunta, ottenendo gli embedding testuali da BERT $f^{(bt)} \in R^{d_{bt}}$ e dall'encoder testuale di CLIP $f^{(ct)} \in R^{d_c}$, e concatenarli per ottenere un vettore di feature testuali $f^{(t)} \in R^{d_c+d_{bt}}$ che sfrutta la capacità di estrazione di entrambi. La stessa cosa viene fatta anche per le immagini utilizzando l'encoder per le immagini di CLIP $f^{(ci)} \in R^{d_c}$ e ResNet $f^{(ri)} \in R^{d_{ri}}$ ottenendo una rappresentazione visuale $f^{(i)} \in R^{d_c+d_{ri}}$. Un'ulteriore rappresentazione $f^{(f)} \in R^{2d_c}$ (denominata fusione) è ottenuta andando a concatenare le feature visive con quelle testuali ottenute da CLIP. Le feature così ottenute passano per una **Testa di proiezione**, ovvero due layer completamente connessi separati da Batch Normalization e ReLU che hanno lo scopo di permettere al modello di apprendere pattern più complessi ma anche quello di portare tutte le feature ad avere la stessa dimensione $d_{fnd} = 1024$. La rappresentazione ottenuta dalla testa di proiezione di fusione, viene moltiplicata per similarità tra le immagini e il testo, normalizzata sul valore medio corrente e sulla deviazione standard corrente e passata in input ad una sigmoide per mapparla su valori tra 0 e 1. In questo modo si inserisce all'interno

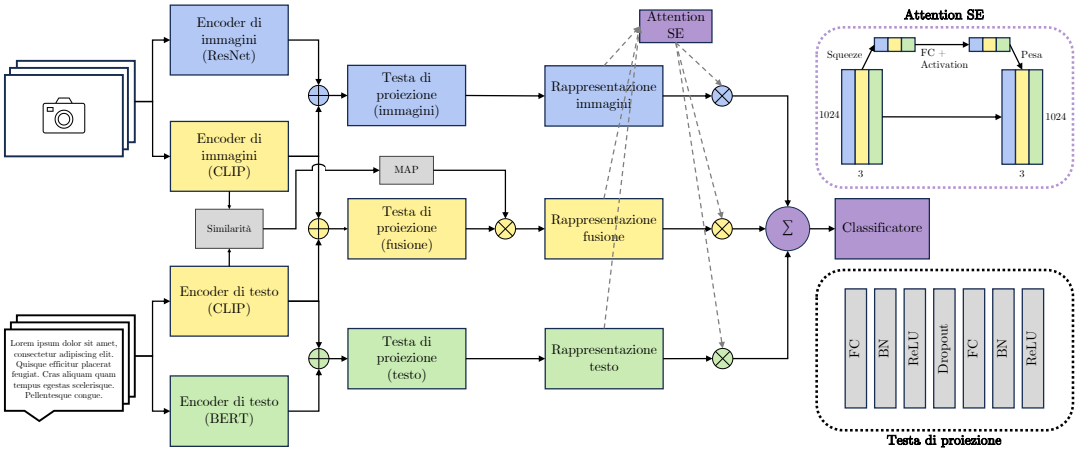


Figura 5.3: Framework FND-CLIP.

della rete l'informazione sulla similarità di immagini e testo, calcolabile con CLIP dal momento che testo e immagine vengono proiettati nello stesso spazio degli embedding. Questo approccio è efficace in questo specifico caso come si può osservare in figura 5.4, dove si può osservare come la similarità tra testo e immagini aumenti all'aumentare della veridicità della notizia.

Il processo appena descritto che porta l'input a un vettore delle feature di ogni modalità verrà da ora in poi chiamato **flusso**. Le feature risultanti da ogni flusso vengono poi concatenate per ottenere una matrice di dimensioni $d_{fnd}, 3$, la quale viene mandata in input ad uno **Squeeze and excitation** layer [61], che riassume in un unico valore tutte le feature calcolandone la media, passando per due layer completamente connessi con funzione di attivazione GELU [62]. I valori ottenuti vengono successivamente moltiplicati per le corrispondenti feature in ingresso. L'obiettivo dello squeeze and excitation layer è quello di agire come un modulo di Attention per le 3 diverse modalità presenti, in modo da ottenere dei pesi che riescono a dare più o meno importanza alle diverse feature. I vettori ottenuti per ogni modalità vengono infine sommati tra di loro, ottenendo un vettore finale $f \in R^{d_{fnd}}$ che viene portato in ingresso al classificatore.

5.5.1 Estensioni di FND-CLIP

Partendo da questa struttura, si è deciso inizialmente di testare andando ad allenare anche la parte di CLIP. Come risultato di questa scelta viene a mancare la parte del calcolo delle similarità dal momento che CLIP, come espresso nella sezione 3.3.1, è allenato con l'obiettivo di accoppiare un'immagine con il testo corrispondente. Cambiando completamente task con una di classificazione, i pesi vengono aggiornati per ottenere embedding di testo e immagine che permettono la divisione in classi, andando a far perdere di significato la similarità. Viene anche rimossa la parte di ResNet, dal momento

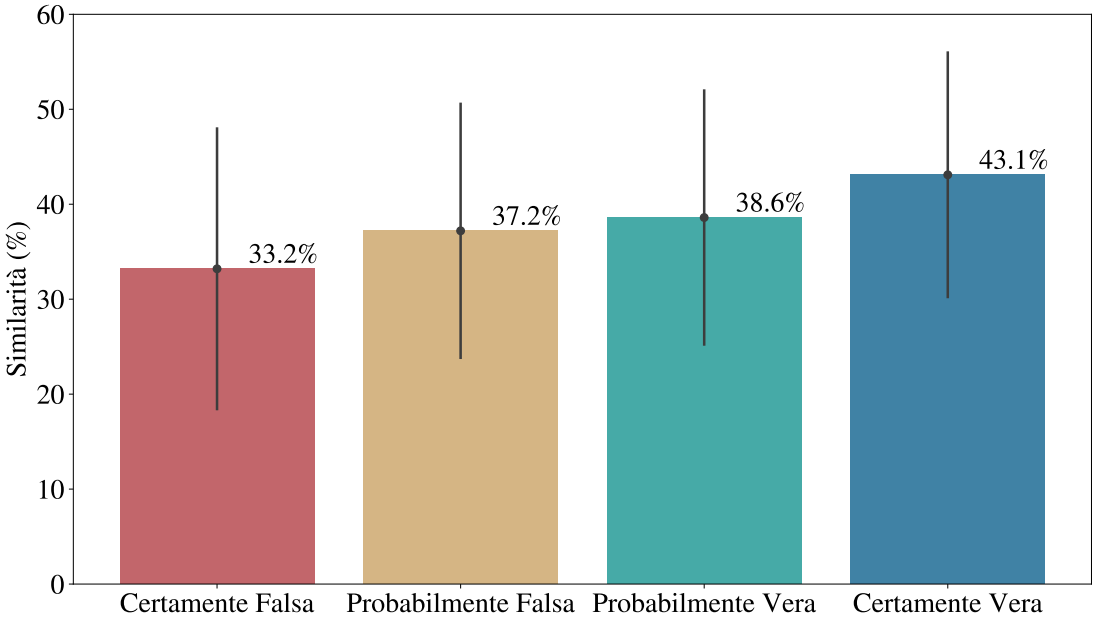


Figura 5.4: Media e deviazione standard della similarità coseno tra immagini e testi calcolata da CLIP.

che non ha fornito alcun beneficio, probabilmente a causa del fatto che CLIP e ResNet, vedendo le stesse immagini e combinando insieme le feature da loro estratte, non riescono a generalizzare causando overfitting. Un'ulteriore aggiunta che è stata fatta è stata quella di inserire un'operazione di reshaping delle feature finali da un vettore di dimensione 1024 a una matrice 32x32, per poter utilizzare una MobileNetV3 [63], per cercare di ottenere informazioni che riuscissero a comprendere le 3 diverse modalità in maniera non convenzionale. Nonostante i risultati siano stati leggermente migliori, non sono stati quelli sperati, per questo si è tornati ad utilizzare la versione standard di FND-CLIP andando questa volta effettivamente a congelare CLIP sfruttando a pieno il suo pretraining, in questo modo ResNet può essere reinserita e non rischiare overfitting dato che CLIP non è allenato sui dati di training. Le estensioni principali pensate per FND-CLIP sono sei, le quali saranno identificate da una lettera per facilitarne la lettura:

A. **Rappresentazione testuale basata sul sentimento** Questa estensione propone l'arricchimento degli embedding testuali già presenti con una terza rappresentazione estratta con BERT addestrato per la Sentiment Analysis[5]. Il modello è pretrainato per classificare un testo in base al sentimento che evoca, che può essere "Positivo", "Negativo" o "Neutrale". L'idea alla base è che l'obiettivo di una notizia falsa è quella di generare una reazione in chi la legge, di solito negativa. Le notizie vere, d'altro canto, dovrebbero essere il più neutrali possibili per riportare la notizia in maniera professionale e imparziale. Questa teoria è confermata

[5]https://huggingface.co/neuraly/bert-base-italian-cased-sentiment

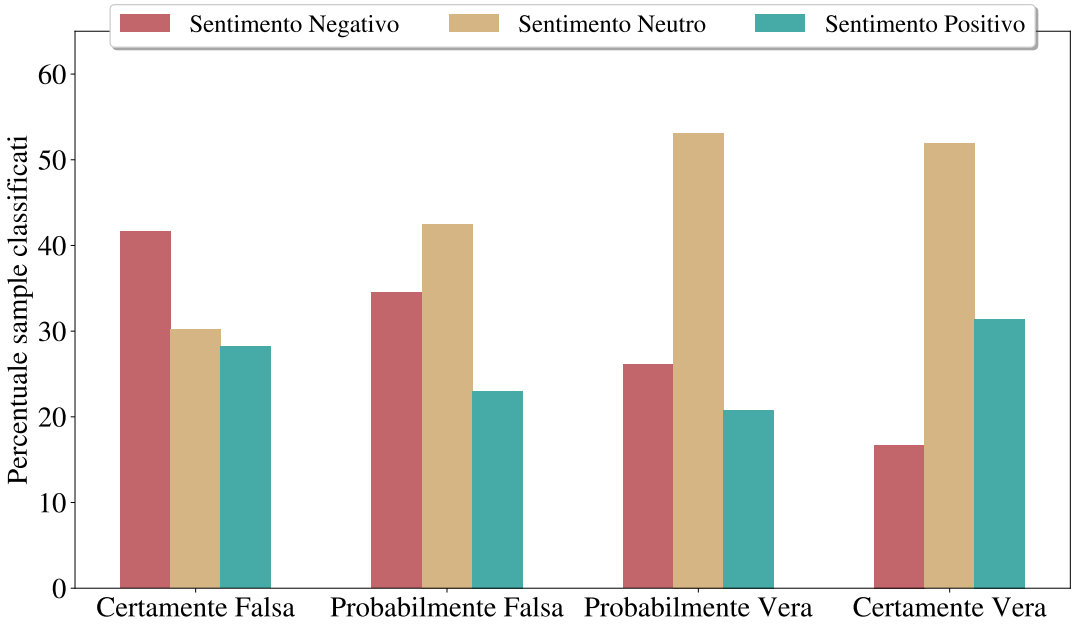


Figura 5.5: Analisi del sentimento che evocano i testi delle notizie divisi per label.

dall'analisi del sentimento dei testi della notizia, osservabile nel grafico in figura 5.5. Sono state testate due varianti di questo approccio:

A_1. Concatenazione con il flusso testuale In questa variante gli embedding di sentimento sono stati concatenati agli embedding testuali di BERT e di CLIP prima della testa di proiezione.

A_2. Flusso a parte Nella seconda variante, invece, è stato sviluppato un flusso a parte per gli embedding di sentimento, presentando quindi una testa di proiezione dedicata.

Le due varianti sono state sottoposte a test per valutare se l'arricchimento dell'embedding testuale con i sentimenti apporti una maggiore discriminazione rispetto agli embedding di sentimenti considerati singolarmente.

B. Flusso basato sulla DFT Passare dal dominio dello spazio a quello della frequenza permette il riconoscimento di immagini false generate mediante modelli generativi [64]. Di fatto, tali immagini, sono indistinguibili da quelle reali nel dominio spaziale, ma presentano delle distorsioni visive chiaramente riconoscibili quando analizzate nel dominio delle frequenze. Questo approccio è stato analizzato nel campo nel riconoscimento delle fake news in [52]. Prendendo spunto da quest'ultimo, le immagini sono state convertite al dominio della frequenza mediante la Trasformata Discreta di Fourier (DFT) per poi essere immesse in un flusso dedicato.

Nel metodo proposto dagli autori, viene utilizzata la DFT sull'immagine per ottenere una combinazione della parte reale e immaginaria, che vengono poi concatenate e utilizzate come input per VGG19 per ottenere una rappresentazione $f^{(freq)} = \text{VGG19}(concat(IF_{imag}, IF_{real}))$. In questo studio l'approccio utilizzato è stato leggermente diverso, infatti sono stati utilizzati due rami di VGG19, dove sono state inserite in input separatamente la parte reale e la parte immaginaria. La rappresentazione ottenuta si può riassumere quindi come $f^{(freq)} = concat(\text{VGG19}(IF_{imag}), \text{VGG19}(IF_{real}))$. Il motivo di questo cambiamento sta nell'idea che essendo VGG una rete convoluzionale, cerca di sfruttare le informazioni di località delle immagini mediante filtri convoluzionali, che possono essere carpite in maniera più efficace analizzando parte reale e immaginaria separatamente, concatenando gli embedding alla fine. Inoltre parte reale e immaginaria presentano ordini di grandezza molto diversi tra di loro, rendendo difficile l'aggiornamento dei pesi della rete.

C. **Concatenazione degli embedding** Mentre FND-CLIP somma gli embedding risultanti da ogni flusso prima del classificatore, si è provato invece a concatenarli. È già stato dimostrato come l'utilizzo della concatenazione sia un modo efficace di combinare informazioni provenienti da diverse modalità [65]. Infatti, quando si sommano gli embedding, si perdono inevitabilmente informazioni che sono cruciali in un contesto multimodale per comprendere la relazione tra le diverse modalità. Inoltre, mediante la concatenazione il modello può decidere più facilmente quanto è importante ai fini della classificazione.

D. **Riequilibrio delle classi mediante data augmentation** Come già accennato in precedenza nella sezione 4.2, il dataset presenta uno sbilanciamento non trascurabile delle classi. È stato quindi deciso di generare nuovi campioni per le classi sottocampionate mediante la tecnica di back-translation (BT da ora in poi), prendendo ispirazione da [66].
La tecnica di back-translation consiste nel tradurre un testo da una lingua all'altra e dopo tradurla nuovamente nella lingua originale, in questo modo la semantica del testo rimane intatta, andando però a cambiare le parole che formano la frase. Formalmente, preso un testo t e due traduttori T_1 e T_2, si ottiene un testo $t_{bt} = T_2(T_1(t))$. Il processo è osservabile in figura 5.6. In particolare, il testo di ogni campione che non fosse appartenente alla classe PV (che ricordiamo essere la classe più popolosa) è stato tradotto e ritradotto in fase di training con una probabilità del 50%, al quale viene associata la stessa immagine associata al testo t, ottenendo quindi un campione $(t_{bt}, i_1, ..., i_m)$ a partire da $(t, i_1, ..., i_m)$.
La lingua ausiliaria utilizzata è stata l'inglese e i modelli per la traduzione sono

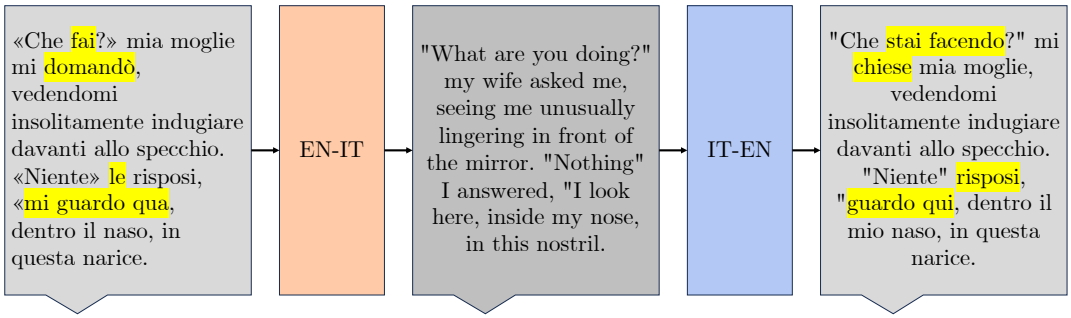


Figura 5.6: Rappresentazione del processo di BT da italiano a inglese e viceversa. Passaggio tratto da [67].

stati presi da Helsinki-NLP[6].

E. **Squeeze and excitation separatamente per flusso** Prendendo l'idea dell'utilizzo dei layer di squeeze and excitation di FND-CLIP per le modalità di ogni flusso, sono stati usati anche all'inizio dei flussi delle modalità testuali e visuali. Questo è stato fatto per permettere alla rete di apprendere autonomamente quali fossero gli embedding più o meno importanti per la classificazione tra quelli ottenuti dai vari encoder prima della loro concatenazione. In altre parole, l'utilizzo dei layer di squeeze and excitation consente al modello di pesare il contributo di ogni embedding, riducendo l'importanza delle informazioni presenti in alcuni di questi rispetto agli altri.

F. **Utilizzo di patch embedding** Inoltre, è stato condotto un test sull'uso degli embedding delle patch ottenute utilizzando Detectron2 [68], un modello in grado di rilevare e localizzare oggetti specifici in un'immagine. Sono stati estratti gli embedding relativi a tali oggetti (patch embedding) con l'obiettivo di fornire al modello informazioni visive localizzate e discriminanti per la classificazione. Questi embedding sono stati combinati con gli embedding visivi esistenti per mantenere il contesto generale dell'immagine. Sono state testate due versioni di Detectron2, una pretrainata su COCO [69] e l'altra su LVIS [70].

Le estensioni appena descritte sono state testate individualmente e successivamente combinate tra loro. A causa del numero eccessivo di possibili combinazioni, non è stato tuttavia possibile testarle tutte. Tra tutte le combinazioni provate, la migliore è quella che presenta le estensioni A_1, C e D (che da ora in poi chiameremo "best performer"), la cui architettura è osservabile in figura 5.7.

[6]https://huggingface.co/Helsinki-NLP

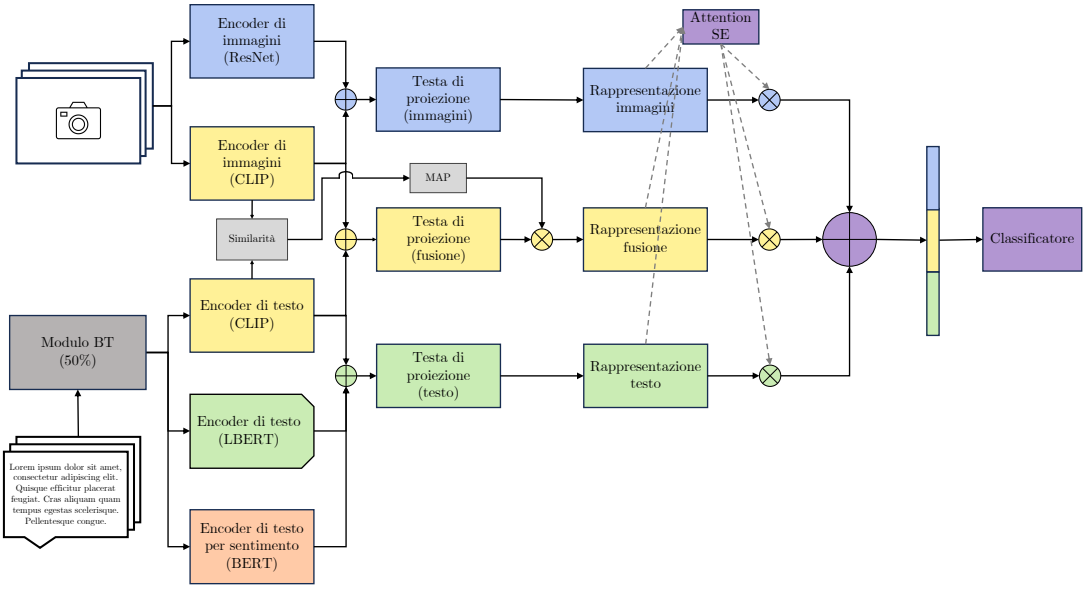


Figura 5.7: Architettura del modello best performer. Primo modello usato per l'ensemble.

5.5.2 Ulteriori test

Come già anticipato, e come vedremo successivamente nel capitolo 6, il miglior modello si è ottenuto con l'utilizzo combinato delle estensioni **A_1, C e D**. A partire da questa combinazione, per cercare di ottenere risultati migliori sono stati eseguiti ulteriori esperimenti la cui metodologia sarà ora descritta:

- **Doppia testa** Utilizzo di una seconda testa di classificazione, in parallelo a quella classica, utilizzata per categorizzare due classi: **"Falsa"** dove sono state raggruppate le notizie con label CF e PF, e **"Vera"** dove sono state aggregate invece le notizie PV e CV. L'idea è quella di aiutare il modello a distinguere meglio notizie false e vere, a prescindere dal grado intermedio di veridicità.

- **Pseudo classe** Con questo cambiamento si è provato ad aggregare invece le notizie PF e PV in un'unica classe "Probabile". Nel caso in cui questa classe fosse scelta, allora verrebbe assegnato PF o PV rispettivamente seguendo le probabilità delle classi CF e CV.

- **Attention** Con questa strategia si è provato invece ad utilizzare dei layer di multi-headed attention prima della classificazione. L'idea è che alla fine si avrà una concatenazione di rappresentazioni provenienti da diverse modalità, il meccanismo di attention dovrebbe essere quindi in grado di capire come le diverse parti sono correlate tra di loro e dare in output una rappresentazione più informativa.

- **Gerarchia** Per questa strategia è stato preso spunto da [71], che propone un modo per eseguire la task di semantic segmentation in maniera gerarchica. Si è provato

a riportare la stessa idea applicata al task di fake news detection considerando le label in maniera gerarchica con una struttura ad albero. In particolare CF e PF sono sottocategorie di "Falso", mentre PV e CV sono sottocategorie di "Vero". In fase di training lo score di ogni classe è stato post-processato per essere coerente con una struttura gerarchica in modo che lo score di ogni nodo sia il massimo dei suoi nodi figli. Successivamente, lo score di ogni nodo sia il minimo dei suoi antenati. A tempo di test si sommano gli score di tutti i possibili rami e si prende quello con lo score maggiore.

- **Collage** Invece di scegliere a caso un'immagine nel training, si è provato a fare un "collage" delle immagini affiancandole spazialmente, in modo da fornire in un'unica immagine tutto il contesto della notizia.

- **Embracenet** Una metodologia più raffinata è stata la tecnica tratta da Embracenet [72], per la quale le feature delle diverse modalità sono state processate dai cosidetti "Docking layer" che le hanno portate ad avere tutte la stessa dimensione. Successivamente, le feature risultati vengono portate in input all'*Embracement layer* con il quale si ottiene in output un vettore i quali elementi sono presi singolarmente dalle modalità sfruttando una distribuzione multinomiale.

- **Emozioni** In linea con la strategia utilizzata nell'analisi dei sentimenti, si è ipotizzato che le emozioni evocate dalle notizie potessero svolgere un ruolo importante. Pertanto, è stato esplorata l'unione degli embedding per la classificazione delle emozioni alla modalità testuale. Questi sono stati estratti utilizzando il modello FEEL-IT [73], che classifica le emozioni "gioia", "tristezza", "paura" e "rabbia".

- **Image-to-text** Durante l'elaborazione delle immagini, spesso viene rilevato del testo presente all'interno di esse. Questo testo viene estratto e successivamente processato utilizzando il modello Italian BERT. Le informazioni testuali estratte dalle immagini sono state integrate con il testo originale o con le informazioni visive, allo scopo di arricchire la comprensione complessiva.

- **Riassunto articoli** Invece di utilizzare LBERT, si è provato invece a riassumere gli articoli utilizzando IT5-Large [74]. In questo modo BERT-IT dovrebbe essere in grado di sfruttare a pieno il suo potenziale essendo che gli articoli riassunti di una lunghezza che può gestire.

- **Pretrain task 2** Svolgere un pre-training sulla task secondaria, che coinvolge dati simili a quelli utilizzati per addestrare il modello, può risultare vantaggioso per acquisire in modo più efficace le caratteristiche rilevanti anche per la task principale.

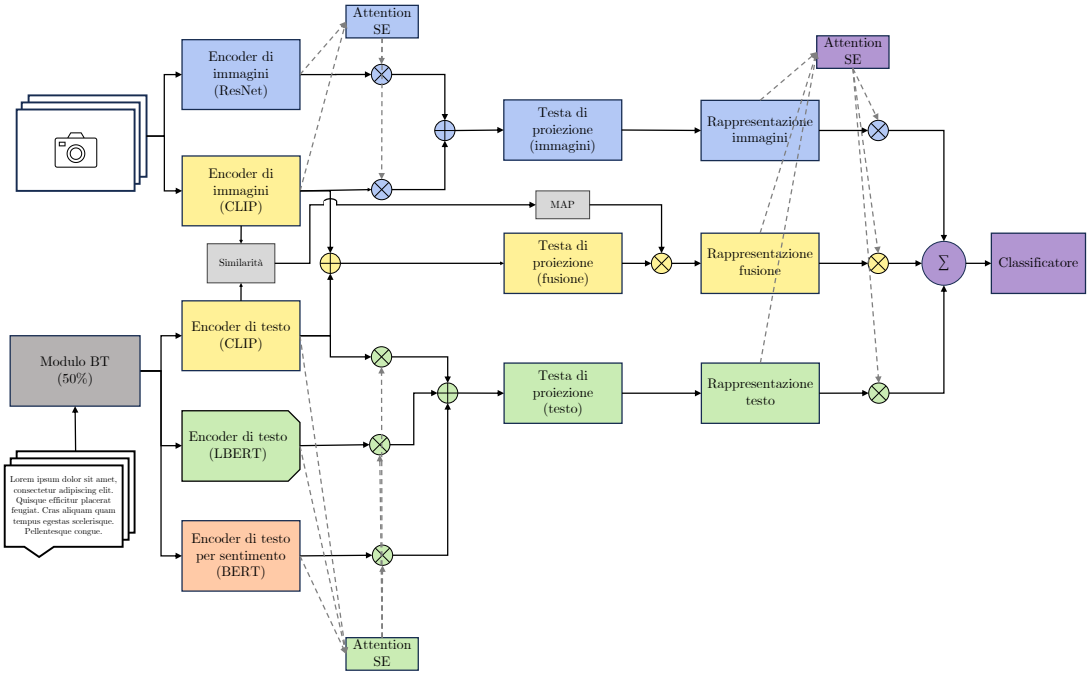


Figura 5.8: Architettura del secondo modello usato per l'ensemble.

5.6 Modello ensemble

Per concludere, è stata condotta un'approfondita analisi su una serie di ensemble. Gli ensemble sono una strategia per migliorare le prestazioni di classificazione, che consiste nel combinare le previsioni di diversi modelli per ottenere una previsione finale più accurata e robusta. L'impiego di un ensemble ha diversi vantaggi:

- Consente di sfruttare le potenzialità dei vari modelli, i quali, grazie alla loro eterogeneità, apprendono rappresentazioni diverse dei medesimi dati.

- Permette di mitigare l'effetto dei singoli modelli che potrebbero avere prestazioni inferiori o essere influenzati da rumore nei dati.

- Affronta problemi di bias o overfitting, poiché la combinazione di modelli diversi tende a ridurre la variabilità e la tendenza a specializzarsi su particolari caratteristiche o casi.

Gli ensemble sono stati creati combinando i modelli derivati dalle diverse estensioni menzionate in precedenza. In particolare, è stata calcolata la media dei logits estratti da ogni modello singolo e le predizioni sono state fatte a partire da quest'ultimi. Successivamente, sono stati eseguiti dei test sui modelli ensemble più performanti andando

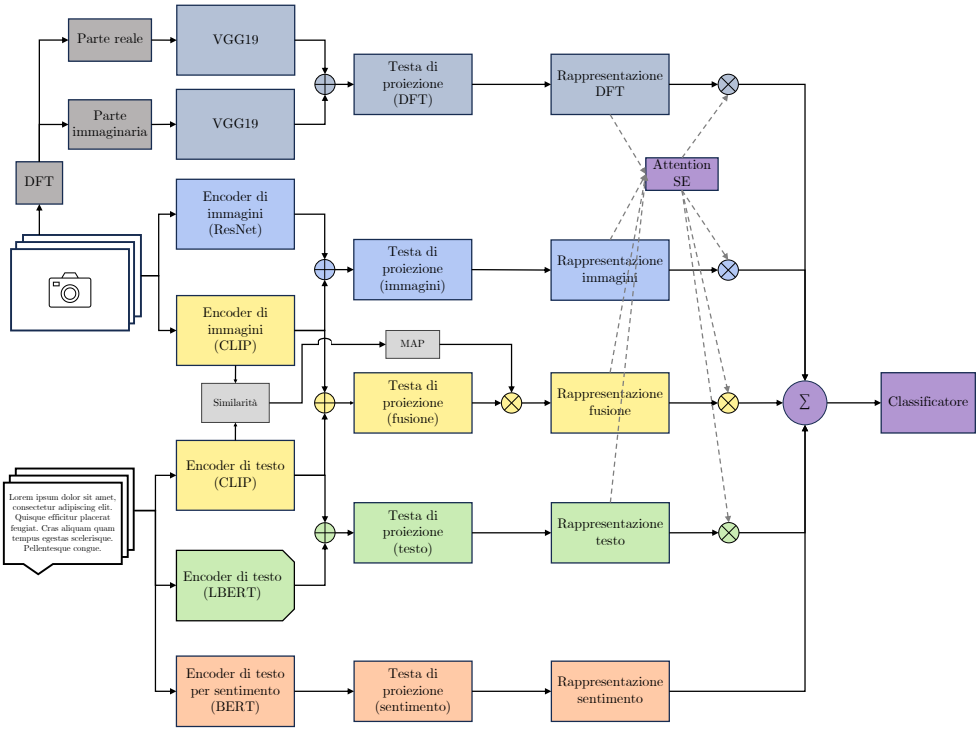


Figura 5.9: Architettura del terzo modello usato per l'ensemble.

a considerare una media pesata dei logits. I risultati migliori sono stati ottenuti combinando il best performer $\text{FND}_{\text{A}_1,C,D}$ con altri due modelli: $\text{FND}_{\text{A}_1,D,E}$ (figura 5.8) e $\text{FND}_{\text{A}_2,B}$ (figura 5.9). È importante specificare che questi ultimi due non sono tra i migliori per quanto riguarda la performance, tuttavia uniti al primo sono quelli che forniscono il miglior risultato. Questo è dovuto probabilmente al fatto che essendo molto diversi tra di loro, imparano a discriminare le notizie in maniera differente, migliorando la classificazione unendo i loro punti di forza.

5.6.1 Dettagli di training

Per quanto riguarda il setup di training, i modelli sono stati allenati per un massimo di 80 epoche e con un batch di dimensione 16, anche se sono stati testati anche altri valori. L'ottimizzatore utilizzato è stato AdamW e uno scheduler lineare per il learning rate. Per quanto riguarda quest'ultimo e per il weight decay, sono stati testati diversi valori ma quello che ha dato performance migliori sul modello finale è stato per entrambi 10^{-3}.

È cruciale evidenziare che, nel contesto della metodologia LBERT, il modello di base BERT è stato inizialmente addestrato e successivamente congelato, dopo aver rimosso la componente di classificazione. L'addestramento simultaneo di numerosi modelli BERT sarebbe stato eccessivamente oneroso in termini di calcolo e memoria, rendendo

il processo impraticabile.
Inizialmente si è utilizzata la cross-entropy loss, ma dal momento che il dataset è altamente sbilanciato, si è optato per un l'utilizzo della focal loss.
Infine, come accennato in precedenza, ogni campione x è composto da un testo e almeno un'immagine, cioè il campione x si può definire come $(t, i_1, ..., i_m)$ con m numero di immagini di x, t il testo e i_j l'immagine j-esima. Per gestire la presenza di immagini multiple per campione è stato attuato un procedimento per il quale in fase di training ne viene scelta una a caso tra quelle disponibili $i_r = rand(i_1, ...i_m)$, e la classificazine avviene usando esclusivamente la coppia (t, i_r). In fase di test, invece, il testo di un campione viene duplicato per ogni immagine presente, ottenendo per ogni campione m coppie $(t, i_1), (t, i_2), ..., (t, i_m)$ considerate singolarmente dal modello. Dai logits $l = (l_1, l_2, ..., l_m)$ ottenuti viene calcolata la media e infine estratta la probabilità per ogni classe, ovvero $softmax(mean(l))$. Questa procedura, oltre che a un metodo per gestire più immagini in un campione, può anche essere vista come un tipo di data augmentation che permette al modello di generalizzare meglio.

Capitolo 6

Presentazione dei risultati

Nel presente capitolo, saranno esposti e analizzati i risultati conseguiti utilizzando i metodi menzionati nel capitolo precedente. Gli esperimenti presentati offrono una panoramica dello studio, che include numerosi test supplementari, qui non inclusi per garantire una maggiore chiarezza nella lettura. Per il confronto tra i modelli sono state riportate le misure di Accuracy (Acc.), Precision (Prec.), Richiamo (Ric.), Macro F1-score (F1) e F1-score weighted (F1-w). Tuttavia, per il confronto tra i diversi modelli, si farà riferimento principalmente a quest'ultima. Si riportano inoltre anche i valori di F1 per ciascuna classe nell'ordine seguente: CF, PF, PV e CV, racchiusi tra parentesi quadre nelle tabelle.

6.1 Risultati

6.1.1 Test delle baseline

I primi risultati riportati sono quelli riguardanti le varie baseline, e sono visibili in tabella 6.1. Come si può notare, i modelli di linguaggio ottengono performance superiori a quelli

Modello	Acc.	Prec.	Ric.	F1	F1-w	F1 per classe
BERT	0.554	0.558	0.470	0.491	0.531	[0.449, 0.353, 0.663, 0.500]
GilBERTo	0.522	0.510	0.462	0.473	0.514	[0.391, 0.422, 0.632, 0.448]
BART	0.495	0.447	0.428	0.429	0.472	[0.542, 0.290, 0.620, 0.264]
ResNet18	0.451	0.381	0.335	0.319	0.399	[0.286, 0.250, 0.631, 0.108]
ViT	0.402	0.336	0.322	0.324	0.388	[0.264, 0.310, 0.564, 0.159]
BERT+ResNet18	0.489	0.497	0.408	0.415	0.465	[0.421, 0.329, 0.618, 0.293]
BERT+ResNet152	0.516	0.516	0.423	0.433	0.479	[0.538, 0.324, 0.631, 0.238]
CLIP	0.538	0.523	0.478	0.484	0.520	[0.557, 0.400, 0.638, 0.340]

Tabella 6.1: Modelli di baseline, BERT e GilBERTo come baseline che considerano solo il testo, ResNet-18 e ViT come modelli che considerano solo le immagini, BERT + ResNet-18 e CLIP come modelli multimodali.

Modello	Acc.	Prec.	Ric.	F1	F1-w	F1 per classe
ResNet18	0.451	0.381	0.335	0.319	0.399	[0.286, 0.250, 0.631, 0.108]
ResNet50	0.435	0.370	0.332	0.323	0.398	[0.208, 0.333, 0.600, 0.150]
ResNet101	0.413	0.365	0.356	0.358	0.404	[0.333, 0.247, 0.552, 0.299]
ResNet152	0.451	0.406	0.373	0.380	0.436	[0.298, 0.265, 0.608, 0.351]

Tabella 6.2: Confronto tra le diverse ResNet.

visuali. In particolare, tra questi, BERT ottiene i risultati migliori, per questo motivo è stato l'encoder di testo utilizzato nei test successivi. Per quanto riguarda i modelli visuali, la loro debolezza era prevista, dal momento che in una notizia le immagini da sole mancano del contesto necessario a raccontare una storia. Tra il modello ResNet e ViT osserviamo che il primo performa leggermente meglio. Per quanto concerne i modelli multimodali, l'impiego dell'encoder testuale e visuale di CLIP come estrattori di feature si dimostra superiore a BERT + ResNet, in particolare per quanto riguarda la metrica di F1. Tale risultato può essere attribuito al pretraining congiunto di testo e immagine di CLIP (che utilizza il contrastive learning), il quale si adatta in modo ottimale alla sfida del riconoscimento di notizie false, nella quale la correlazione tra testo e immagine riveste un ruolo cruciale. Il fatto che un modello di linguaggio performi meglio di uno multimodale ottenuto dalla concatenazione di feature testuali e visuali estratte separatamente mostra come sia importante avere un metodo per esplicitare in maniera migliore la relazione tra testo e immagine.

6.1.2 Test delle ResNet

Per quanto riguarda i test sulle ResNet, osservabili in tabella 6.2, si può notare come modelli più profondi ottengano risultati migliori, a costo di un costo computazionale più alto e tempi di training più lunghi. Questo comportamento è dovuto al fatto che, aumentando la dimensione del modello, si aggiungono ulteriori blocchi residuali che permettono di apprendere rappresentazioni sempre più complesse dei dati. Questa maggiore complessità consente di catturare dettagli e pattern più complessi nei dati di input, migliorando le capacità di generalizzazione e l'accuratezza delle previsioni. La ResNet-152 supera la ResNet-18 sia su F1 che su F1-weighted, rendendola la scelta per i modelli finali.

6.1.3 L'impatto di LBERT

In tabella 6.3 si possono osservare i confronti tra le varie strategie head (h), tail (t), head-tail (ht) e LBERT applicate a BERT. La strategia head è migliore delle altre due, dimostrando come la parte iniziale del testo porti informazioni migliori per classificare una notizia falsa. Il risultato non è sorprendente se si considera che le prime righe di

Modello	Acc.	Prec.	Ric.	F1	F1-w	F1 per classe
$BERT_h$ (C)	0.554	0.558	0.470	0.491	0.531	[0.449, 0.353, 0.663, 0.500]
$BERT_h$ (U)	0.538	0.532	0.440	0.446	0.499	[0.491, 0.338, 0.667, 0.286]
$BERT_t$ (C)	0.527	0.518	0.472	0.479	0.515	[0.528, 0.473, 0.617, 0.298]
$BERT_{ht}$ (C)	0.533	0.524	0.440	0.454	0.500	[0.500, 0.369, 0.647, 0.298]
LBERT	0.543	0.530	0.483	0.499	0.533	[0.491, 0.385, 0.648, 0.473]
$CLIP_h$	0.530	0.523	0.478	0.484	0.520	[0.557, 0.400, 0.638, 0.340]
$CLIP_t$	0.495	0.480	0.466	0.471	0.493	[0.526, 0.424, 0.563, 0.373]
$CLIP_{ht}$	0.533	0.524	0.400	0.454	0.500	[0.500, 0.369, 0.647, 0.298]
LCLIP	0.554	0.550	0.479	0.496	0.543	[0.440, 0.404, 0.689, 0.449]
CLIP+LBERT	0.576	0.603	0.490	0.516	0.555	[0.489, 0.471, 0.670, 0.436]

Tabella 6.3: Confronto tra le varie strategie di gestione di testi lunghi.

un articolo sono importanti per ottenere l'interesse del lettore. Gli stessi risultati si ottengono anche utilizzando la stessa strategia con il modello multimodale CLIP, che ottiene risultati migliori per la strategia head. Tail invece presenta risultati leggermente peggiori ma comunque migliori di head-tail. Questo risultato era atteso dal momento che prendendo metà parte iniziale e metà parte finale si crea un testo scollegato semanticamente.
Un altro risultato aspettato è quello che mostra come la versione cased di BERT (indicata con (C) in tabella) ottenga risultati migliori della versione uncased ((U) in tabella), questo è dovuto alla presenza predominante di caratteri maiuscoli all'interno dei tweet falsi.
Performance migliori si ottengono però con la strategia LBERT, che non solo permette di avere risultati migliori quando viene applicata su BERT, ma anche quando viene utilizzata con l'embedding testuale di CLIP (LCLIP)
Il problema di CLIP (e quindi anche di LCLIP) sta nel fatto che utilizza una versione uncased di BERT, che nel nostro caso inficia sulle prestazioni. Per questo si è provato a unire l'embedding visuale di CLIP con quello testuale LBERT, ottenendo le migliori prestazioni fino a questo momento.

6.1.4 Esperimenti con CLIP

Essendo CLIP e BERT i modelli che hanno dato risultati migliori fino a questo momento, ci si è concentrati su test che riguardano questi ultimi, cercando sempre di sfruttare la multimodalità dei dati. I risultati sono osservabili in figura 6.4. Questa porzione di studio non ha prodotto risultati notevoli o sorprendenti, tuttavia è stata utile per verificare l'inefficacia di alcuni metodi. Si può infatti osservare come l'utilizzo della concatenazione degli embedding di diverse modalità abbia un leggero vantaggio rispetto alla somma di quest'ultime, perché permette di modellare relazioni più complesse tra testo e immagini. Inoltre, si può osservare che l'utilizzo di 3 layer completamente

Modello	Acc.	Prec.	Ric.	F1	F1-w	F1 per classe
LCLIP						
sum	0.543	0.540	0.478	0.492	0.538	[0.426, 0.417, 0.674, 0.453]
concat	0.554	0.550	0.479	0.496	0.543	[0.440, 0.404, 0.689, 0.449]
concat (3 layers)	0.549	0.530	0.483	0.497	0.541	[0.462, 0.391, 0.682, 0.453]
CLIP+LBERT						
CE	0.576	0.603	0.490	0.516	0.555	[0.489, 0.471, 0.670, 0.436]
Focal	0.565	0.567	0.547	0.554	0.567	[0.593, 0.489, 0.617, 0.516]
Focal weighted	0.543	0.533	0.489	0.499	0.537	[0.417, 0.473, 0.643, 0.464]
Focal mix. fusion	0.587	0.582	0.510	0.530	0.570	[0.468, 0.427, 0.698, 0.526]

Tabella 6.4: Studio sulle prestazioni di LCLIP e CLIP+LBERT.

Modello	Acc.	Prec.	Ric.	F1	F1-w	F1 per classe
CLIP+LBERT+Detectron2						
COCO CE	0.560	0.568	0.504	0.522	0.554	[0.522, 0.458, 0.655, 0.452]
COCO Focal	0.571	0.565	0.493	0.512	0.550	[0.538, 0.388, 0.677, 0.444]
LVIS CE	0.549	0.544	0.477	0.496	0.534	[0.480, 0.395, 0.656, 0.453]
LVIS Focal	0.565	0.557	0.485	0.502	0.546	[0.435, 0.423, 0.677, 0.475]

Tabella 6.5: Confronto performance aggiungendo le patch embedding di detectron2. COCO e LVIS sono i dataset su cui è stato pre-trainato Detectron2.

connessi nella testa di classificazione invece dei 2 utilizzati non migliori le prestazioni. Tuttavia, un risultato di particolare interesse emerge dall'utilizzo della focal loss (indicata con Focal in tabella), che permette di ottenere il modello più performante tra quelli testati fino ad ora, mostrando come quest'ultima sia benefica dato l'elevato squilibrio tra le classi presente nel dataset. Infatti, i risultati sono migliori rispetto alla Cross-entropy loss (CE in tabella). Gli ultimi due risultati in tabella sono relativi al pesare le classi con pesi inversi alla propria frequenza in modo da bilanciare meglio la classificazione e all'utilizzo della tecnica di mixed fusion, che processa mediante layer completamente connessi gli embedding delle diverse modalità prima di concatenarli. Pesare le classi non è benefico al modello, questo accade perché probabilmente porta ad una situazione di overfitting, dove il modello si concentra eccessivamente sulla classe minoritaria. L'utilizzo della mixed fusion non apporta vantaggi significativi, ma allo stesso tempo non compromette le prestazioni del modello.

6.1.5 Test con Detectron2

Nella tabella 6.5 si può osservare che, unendo al modello CLIP + LBERT, le feature ottenute dalle patch di detectron2, le performance non migliorino, andando leggermente a peggiorare. Ciò è dovuto all'eccessiva informazione visuale fornita al modello (ViT di CLIP + Detectron2), che porta all'overfitting dello stesso. Si può tuttavia osservare come il pretraining su COCO performi meglio di quello su LVIS, dimostrando come le

Modello	Acc.	Prec.	Ric.	F1	F1-w	F1 per classe
Senza MobileNet, Cross Entropy loss						
FND-NF	0.533	0.504	0.452	0.463	0.511	[0.491, 0.416, 0.653, 0.292]
FND-NF$_{NR}$	0.538	0.512	0.456	0.465	0.518	[0.462, 0.419, 0.674, 0.304]
Con MobileNet, Cross Entropy loss						
FND-NF$_{NR}$	0.543	0.537	0.491	0.501	0.533	[0.582, 0.455, 0.633, 0.333]
FND-NF$_{NR,LL}$	0.571	0.577	0.492	0.512	0.559	[0.444, 0.381, 0.710, 0.511]
Con MobileNet, Focal loss						
FND-NF$_{NR,LL}$	0.587	0.584	0.521	0.538	0.579	[0.528, 0.437, 0.708, 0.480]

Tabella 6.6: Confronto tra le variazioni di FND-NF.

patch raffiguranti persone siano più utili alla classificazione. Rimuovere l'encoder visuale per lasciare esclusivamente detectron2 non è stata una scelta presa in considerazione per due motivi. In primo luogo a volte detectron2 non riesce a catturare nessuna patch significativa, andando a perdere molta informazione. In secondo luogo, anche quando riesce a trovare una o più patch significative, il resto del contesto viene tagliato fuori, andando a rimuovere informazioni potenzialmente importanti. Tuttavia, anche qui possiamo apprezzare il miglioramento dovuto all'utilizzo della focal loss rispetto alla cross-entropy.

6.1.6 Test su FND-CLIP con fine-tuning di CLIP

In tabella 6.6 sono riportati i risultati riguardanti il modello FND-CLIP (abbreviato come FND in tabella) ma senza che i parametri del modello di base CLIP fossero congelati (FND-NF). Si può notare che le performance non sono buone per il modello FND-CLIP di base, ma migliorano di gran lunga nel momento in cui viene rimossa la ResNet (NR). Questo, come già ribadito precedentemente, è dovuto all'overfitting. Infatti, combinando diverse modalità in un modello multimodale, incrementa il rischio di overfitting, specialmente se i dati di una modalità sono più informativi di quelli dell'altra. Unendo i dati ottenuti da due encoder visuali allenati sugli stessi dati risulta nel "duplicare" le stesse informazioni, danneggiando la capacità discriminativa delle feature multimodali ottenute alla fine. Lo stesso discorso si può fare con le feature testuali. Inoltre, l'utilizzo della MobileNetV3 si è rivelato efficace nel catturare le relazioni delle diverse modalità. Infine, è stato valutato l'utilizzo della strategia Long sia per CLIP che per BERT (LL). L'impiego di questa strategia ha avuto un impatto positivo in questo specifico caso.

6.1.7 Esperimenti su FND-CLIP senza fine-tuning di CLIP

Appurata dagli esperimenti precedenti l'importanza della focal loss, tutti i risultati che verranno presentati da ora in poi saranno riferiti a modelli allenati con quest'ultima. Si è deciso di utilizzare FND-CLIP andando a mantenere congelati i parametri di CLIP,

Modello	Acc.	Prec.	Ric.	F1	F1-w	F1 per classe
FND	0.538	0.537	0.490	0.507	0.530	[0.519, 0.395, 0.615, 0.500]
FND_{A_1}	0.565	0.558	0.525	0.530	0.552	[0.625, 0.406, 0.637, 0.453]
FND_{A_2}	0.560	0.541	0.521	0.526	0.550	[0.536, 0.377, 0.644, 0.545]
FND_{B}	0.587	0.587	0.506	0.525	0.565	[0.519, 0.406, 0.697, 0.480]
FND_{C}	0.576	0.578	0.521	0.539	0.568	[0.500, 0.500, 0.656, 0.500]
FND_{D}	0.565	0.540	0.517	0.524	0.555	[0.576, 0.423, 0.663, 0.433]
FND_{E}	0.576	0.563	0.558	0.560	0.574	[0.633, 0.430, 0.639, 0.540]
FND_{F}	0.549	0.548	0.469	0.481	0.524	[0.545, 0.400, 0.660, 0.318]
$\text{FND}_{\text{A}_2,\text{B}}$	0.587	0.578	0.548	0.560	0.581	[0.545, 0.453, 0.659, 0.581]
$\text{FND}_{\text{A}_1,\text{D}}$	0.587	0.587	0.560	0.570	0.583	[0.519, 0.395, 0.615, 0.500]
$\text{FND}_{\text{D},\text{C}}$	0.592	0.573	0.520	0.525	0.565	[0.623, 0.387, 0.705, 0.385]
$\text{FND}_{\text{D},\text{E}}$	0.582	0.555	0.537	0.543	0.574	[0.526, 0.444, 0.678, 0.523]
$\text{FND}_{\text{C},\text{F}}$	0.554	0.530	0.480	0.483	0.526	[0.548, 0.344, 0.674, 0.367]
$\text{FND}_{\text{A}_1,\text{C},\text{D}}$	0.609	0.595	0.593	0.593	0.606	[0.536, 0.377, 0.644, 0.545]
$\text{FND}_{\text{A}_2,\text{C},\text{D}}$	0.576	0.561	0.524	0.535	0.564	[0.542, 0.412, 0.667, 0.517]
$\text{FND}_{\text{A}_1,\text{E},\text{D}}$	0.598	0.603	0.569	0.582	0.596	[0.625, 0.406, 0.637, 0.453]
$\text{FND}_{\text{C},\text{E},\text{F}}$	0.554	0.520	0.505	0.505	0.541	[0.562, 0.382, 0.663, 0.414]
$\text{FND}_{\text{C},\text{D},\text{E}}$	0.571	0.559	0.508	0.524	0.558	[0.519, 0.406, 0.697, 0.480]
$\text{FND}_{\text{A}_2,\text{B},\text{C},\text{D}}$	0.614	0.613	0.569	0.582	0.604	[0.667, 0.486, 0.685, 0.491]
$\text{FND}_{\text{A}_2,\text{B},\text{C},\text{E}}$	0.592	0.601	0.549	0.568	0.588	[0.654, 0.557, 0.648, 0.414]
$\text{FND}_{\text{A}_1,\text{B},\text{C},\text{D},\text{E}}$	0.576	0.568	0.529	0.542	0.572	[0.520, 0.532, 0.655, 0.462]

Tabella 6.7: Confronto tra le estensioni di FND-CLIP e le loro combinazioni.

attuando fine-tuning solamente sull'encoder testuale BERT e di quello visuale ResNet-152 per due motivi:

1. Lasciare CLIP congelato permette al modello di sfruttare a pieno il suo pretraining e di ottenere feature generiche che non portano le stesse informazioni di quelle ottenute dagli altri due encoder dove viene fatto fine-tuning. Questo aiuta il modello ad ottenere feature specifiche e unirle a feature di natura più generica.

2. Evitando il fine-tuning su CLIP permette di trarre il massimo vantaggio dallo spazio comune degli embedding visuali e testuali, su cui si può calcolare la similarità.

In tabella 6.7 sono riportati i risultati ottenuti dalle estensioni citate nel capitolo precedente. Ricordiamo qui il significato di ogni lettera:

A. Rappresentazione basata sul sentimento, con:

  $\mathbf{A}_1$ Concatenazione degli embedding di sentimento con il flusso testuale

  $\mathbf{A}_2$ Embedding di sentimento con flusso a parte

B. Flusso basato sulla DFT applicata alle immagini

Modello	Acc.	Prec.	Ric.	F1	F1-w	F1 per classe
			FND_D			
tweet e articoli	0.576	0.565	0.497	0.502	0.544	[0.581, 0.361, 0.690, 0.375]
tweet 4 lingue	0.560	0.543	0.488	0.501	0.540	[0.536, 0.400, 0.667, 0.400]
tweet inglese	0.565	0.540	0.517	0.524	0.555	[0.576, 0.423, 0.663, 0.433]

Tabella 6.8: Confronto tra diverse tecniche di back-translation.

C. Concatenazione finale degli embedding delle diverse modalità

D. Riequilibrio delle classi mediante data augmentation con back-translation

E. Squeeze and excitation per ogni flusso

F. Utilizzo di patch embedding

Si può osservare come ogni estensione abbia portato beneficio al modello di base, tranne l'estensione **F**. Questo, come detto in precedenza nel sotto-paragrafo 6.1.5, è dovuto al problema di overfitting risultante dall'utilizzo di multiple modalità che trasportano la stessa informazione. L'estensione che sembra portare risultati migliori è la **E**, che permette al modello di riconoscere quale tra le diverse rappresentazioni è più importante riducendo il problema di overfitting.
Si può notare come, unire gli embedding di sentimento al flusso testuale ($\mathbf{A}_1$), produca risultati leggermente migliori dell'utilizzo di un flusso a parte di sentimento ($\mathbf{A}_2$), mostrando come una rappresentazione testuale arricchita con altre informazioni porti a una discriminazione maggiore.
Come previsto, l'utilizzo delle tecniche **B** e **C** presentano risultati superiori rispetto alla baseline, confermando le aspettative iniziali.
L'estensione **D** porta un guadagno non indifferente nelle prestazioni, poiché contribuisce a produrre un modello che generalizza in maniera migliore e permette, in parte, di riequilibrare le classi. Riguardo quest'estensione è stata usata solamente la lingua inglese come lingua ausiliaria per tradurre esclusivamente i tweet che non appartengono alla classe "Probabilmente Vera", cioè quella più popolosa. Questa scelta è stata basata sui risultati di altri test condotti visibili in tabella 6.8 dove si è osservato che:

- eseguire la back-translation sia di tweet che articoli ha generato prestazioni inferiori. Questo è dovuto al fatto che tradurre testi troppo lunghi, soprattutto se tradotti a tratti, può portare a dati rumorosi.

- Utilizzare la back-translation dei tweet usando come lingue ausiliare 4 lingue (inglese, spagnolo, tedesco, lituano) può generare testi troppo rumorosi a causa della maggiore accuratezza della traduzione in inglese (essendo una lingua molto diffusa), mentre le altre tre lingue presentano una minore precisione nella traduzione.

Modello	Acc.	Prec.	Ric.	F1	F1-w	F1 per classe
			$FND_{A_1,C,D}$			
Doppia testa	0.603	0.592	0.557	0.564	0.592	[0.576, 0.462, 0.689, 0.531]
Pseudo classe	0.598	0.588	0.533	0.543	0.578	[0.623, 0.493, 0.688, 0.367]
Attention (1)	0.565	0.559	0.559	0.553	0.563	[0.645, 0.464, 0.610, 0.493]
Attention (2)	0.582	0.586	0.520	0.536	0.565	[0.621, 0.464, 0.660, 0.400]
Attention (4)	0.565	0.569	0.487	0.505	0.549	[0.409, 0.452, 0.677, 0.481]
Gerarchia	0.587	0.572	0.536	0.549	0.578	[0.582, 0.451, 0.678, 0.484]
Collage	0.571	0.546	0.495	0.506	0.549	[0.491, 0.369, 0.691, 0.475]
Embracenet	0.598	0.585	0.531	0.545	0.580	[0.596, 0.441, 0.698, 0.444]
Emozioni	0.582	0.586	0.537	0.554	0.577	[0.560, 0.463, 0.659, 0.533]
Image2Text (T)	0.587	0.586	0.533	0.549	0.575	[0.556, 0.448, 0.667, 0.525]
Image2Text (V)	0.565	0.557	0.521	0.535	0.561	[0.566, 0.494, 0.640, 0.441]
Riassunto articoli	0.598	0.590	0.539	0.543	0.574	[0.656, 0.373, 0.695, 0.448]
Pretrain task 2	0.565	0.556	0.502	0.520	0.554	[0.510, 0.442, 0.663, 0.464]

Tabella 6.9: Confronto tra le estensioni aggiuntive al best performer. (V) indica che gli embedding sono stati uniti al flusso visuale, (T) a quello testuale.

Per quanto riguarda le combinazioni delle varie estensioni, è evidente che in genere producano risultati migliori, mettendo in luce i benefici distinti che apportano alla classificazione. In particolare l'estensione **D** si dimostra sempre efficace nel migliorare le prestazioni quando usata in congiunzione con le altre. L'efficacia di **E** si può notare dal test **C,E,F** che, rispetto a **C,F** conduce a risultati superiori, dimostrando come lo squeeze and excitation layer per le diverse rappresentazioni riesca a dare meno importanza a detectron e a portare ad una migliore generalizzazione. I risultati migliori si sono ottenuti con la combinazione $\mathbf{A_1}$**, C, D**, ottenendo uno score F1-weighted pari a 0.606, divenendo il miglior modello utilizzato singolarmente.
Confrontando **C,D,E** e **D,E**, inoltre, si può notare come l'utilizzo della concatenazione sia dannoso nel caso in cui si utilizzino i layer di squeeze and excitation. Questo può essere attribuito al fatto che questi ultimi sono già capaci di pesare le informazioni.
È interessante notare come unire tutte le rappresentazioni (a meno di **F**), non porti ai risultati migliori. Questo è probabilmente a causa dell'unione di troppe informazioni che rendono il modello troppo complesso e, quindi, poco generalizzante a dati nuovi.

6.1.8 Ulteriori estensioni

Come anticipato nel capitolo precedente, a partire dal best performer $FND_{A_1,C,D}$, si sono testate delle ulteriori estensioni. I risultati di quest'ultime sono osservabili in tabella 6.9 ed ora verranno commentate quelle più rilevanti dalle quali ci si sarebbe atteso un miglioramento delle performance che, tuttavia, non c'è stato.

- **Attention** In tabella il numero tra parentesi sta ad indicare il numero di layer

Modello	Acc.	Prec.	Ric.	F1	F1-w	F1 per classe
Senza pesi	0.641	0.638	0.601	0.615	0.635	[0.621, 0.521, 0.708, 0.610]
Con pesi	0.658	0.661	0.621	0.637	0.653	[0.643, 0.560, 0.712, 0.633]

Tabella 6.10: Risultati modello ensemble, con e senza media pesata.

di multi-headed attention. Ci si aspettava che mediante dei layer di attention il modello sarebbe stato capace di meglio interpretare le relazioni tra le modalità, tuttavia cosi non è stato. Un motivo potrebbe essere che, data la complessità del problema, i dati di training non siano abbastanza per stimare correttamente i pesi di attention. Una conferma di questo è data dal fatto che l'utilizzo di un solo layer di attention performa meglio di multipli layer.

- **EmbraceNet** Per quanto riguarda la strategia di EmbraceNet, nonostante sia una strategia sofisticata di fusione, ha performance peggiori della semplice concatenazione. Questo perché, come specificato dai creatori, la loro strategia si concentra sull'affrontare la mancanza di modalità nei dati. Questa situazione non avviene mai nel nostro caso di studio dal momento che ogni campione ha sempre un testo e almeno un'immagine.

- **Emozioni** Un altro risultato contro intuitivo è quello dell'utilizzo degli embedding per le emozioni, che segue lo stesso ragionamento dei sentimenti. Tuttavia, il fatto che le performance scendano molto potrebbe essere dato dal fatto che le informazioni delle emozioni si sovrappongono a quelle del sentimento, duplicando le rappresentazioni e causando overfitting.

- **Pretrain su task 2** In questo caso, pretrainare su dati simili dovrebbe aver dato al modello la capacità di apprendere meglio la struttura dei dati. Tuttavia, avendo pre-trainato su un task diverso, i parametri trovati si trovano probabilmente su un minimo della funzione di loss lontano da quello vero, fornendo una cattiva inizializzazione.

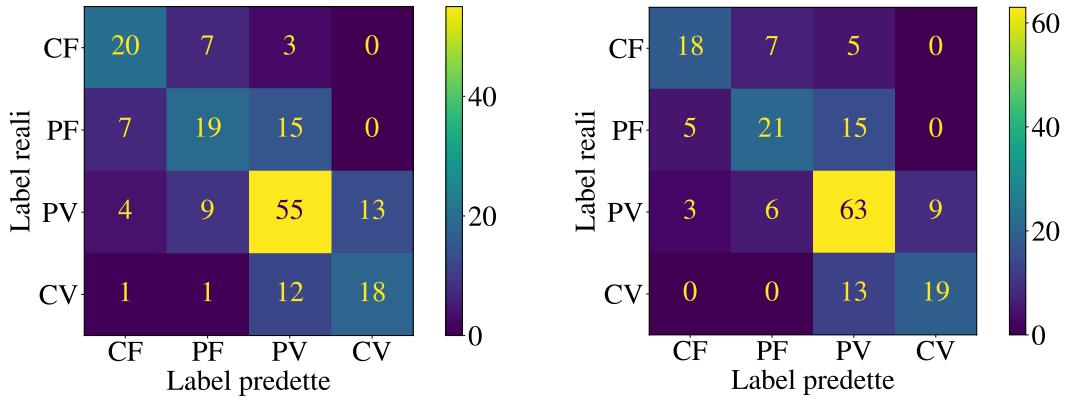
Il resto dei risultati sono stati riportati in tabella per completezza. Tuttavia, non verranno commentati per semplicità di lettura e perché non sono particolarmente interessanti.

6.1.9 Modelli ensemble

In tabella 6.10 sono esposti i risultati per i modelli ensemble utilizzando, come anticipato, i modelli:

- $\text{FND}_{\text{A}_1\text{C,D}}$, cioè il best performer, che chiameremo **modello 1**

- $\text{FND}_{\text{A}_1\text{,E,D}}$ che chiameremo **modello 2**



(a) Best performer **(b)** Modello ensemble

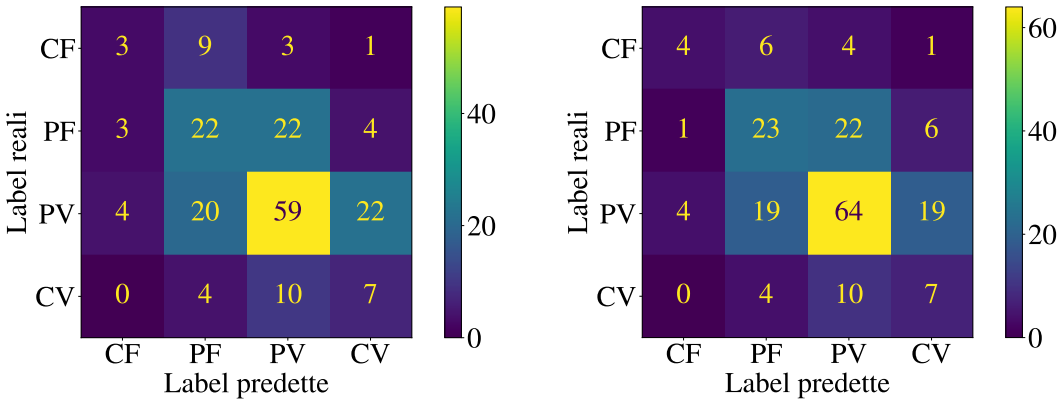
Figura 6.1: Matrici di confusione sul validation set.

Modello	Acc.	Prec.	Ric.	F1	F1-w	F1 per classe
Best performer	0.472	0.383	0.379	0.373	0.479	[0.231, 0.415, 0.593, 0.255]
Ensemble	0.505	0.435	0.412	0.414,	0.512	[0.333, 0.442, 0.621, 0.259]

Tabella 6.11: Risultati sul test set.

- $\text{FND}_{\text{A}_2,\text{B}}$ che chiameremo **modello 3**

Sono state testate molte combinazioni tra i modelli presenti, ma questa è quella che ha portato a risultati migliori. In particolare si può vedere come si ottenga un F1-weighted pari a 0.635 andando a fare la media dei logits dei singoli modelli per poi eseguire le predizioni.
Come si può vedere, i modelli utilizzati, presi globalmente, utilizzano tutte le estensioni proposte (a meno della F che, come abbiamo visto, non è efficace). Questo conferma ulteriormente che le singole estensioni sono utili alla classificazione. Inoltre, il fatto che i modelli migliori per l'ensemble siano molto diversi tra di loro, per quanto concerne le estensioni usate, è un comportamento che si allinea con l'idea di ensemble, che sfrutta le competenze individuali dei diversi modelli che hanno appreso informazioni in maniera diversa. Utilizzando una media pesata invece, si ottiene un F1-weighted di 0.653. Sono state analizzate diverse combinazioni di pesi, tuttavia quella migliore è stata 1.0, 0.8 e 0.7, rispettivamente per i modelli 1, 2 e 3, coerentemente con i risultati individuali ottenuti. In figura 6.1 possiamo osservare le matrici di confusione del best performer e dell'ensemble a confronto. Come possiamo vedere l'ensemble è più capace a riconoscere le notizie PV, e non classifica mai una notizia vera come falsa. Tuttavia peggiora leggermente per la classificazione su CF, che passano da 20 del best performer a 18.



(a) Best performer **(b)** Modello ensemble

Figura 6.2: Matrici di confusione sul test set.

Modello	F1-w
Ensemble	**0.512**
Partecipante 1	0.507
Partecipante 2	0.488
Best performer	0.479
Miglior baseline testo	0.479
Miglior baseline multimodale	0.463
Miglior baseline visuale	0.402
Partecipante 3	0.393

Tabella 6.12: Risultati della competizione.

6.1.10 Risultati sul test set

In tabella 6.11 si possono osservare i risultati sul test set per il best performer e per l'ensemble. I risultati, nonostante notevolmente inferiori a quelli del set di validazione, sono comunque coerenti con quest'ultimi, con l'ensemble che supera il miglior modello singolo ottenendo performance superiori su tutte le classi. In figura 6.2 si possono osservare le matrici di confusione sul test set. Com'è facile vedere, l'ensemble è di nuovo migliore nel classificare la classe PV. Tuttavia, in questo caso, riesce leggermente meglio a classificare le notizie false. In tabella 6.12 si può osservare che il modello proposto supera, sul test set, le performance degli altri partecipanti alla competizione, oltre che le baseline proposte dagli organizzatori.

6.1.11 Risultati task secondaria

Le architetture principali presentate per la task primaria sono state testate anche per quella secondaria, per analizzare la robustezza del modello al cambio di task. Come si può vedere nella tabella 6.13, il modello che performa meglio questa volta è quello con la

Modello	Acc.	Prec.	Ric.	F1	F1-w	F1 per classe
$FND_{A_1,C,D}$	0.591	0.606	0.585	0.592	0.592	[0.621, 0.602, 0.553]
$FND_{A_2,B}$	0.598	0.610	0.592	0.598	0.599	[0.652, 0.631, 0.512]
$FND_{A_1,E,D}$	0.621	0.627	0.611	0.616	0.619	[0.652, 0.661, 0.535]
Ensemble	0.636	0.652	0.630	0.637	0.637	[0.687, 0.655, 0.570]

Tabella 6.13: Risultati per la task secondaria sul validation set.

combinazione $\mathbf{A_1}$**, E, D**, mentre l'ensemble, come ci si aspetta, continua anche in questo caso a performare meglio dei singoli modelli. In 6.14 si può osservare il risultato della competizione per la task secondaria. Nonostante sia stata meno partecipata dell'altra competizione, si può vedere come il modello presentato superi le baseline, anche se di poco. Questo dimostra che indubbiamente il modello ha la capacità di essere robusto al cambio di task e di essere efficace al riconoscimento di relazione tra testo e immagini. Tuttavia, non essendo stato fatto ad hoc per la task secondaria, le performance sono di poco superiori al modello solo testuale.

Modello	F1-w
Ensemble	**0.517**
Miglior baseline testo	0.506
Miglior baseline multimodale	0.461
Miglior baseline visuale	0.436
Partecipante 1	0.421

Tabella 6.14: Risultati della competizione per la task secondaria.

Capitolo 7

Conclusioni e sviluppi futuri

L'obiettivo di questo lavoro di tesi è stato quello di sviluppare un modello di intelligenza artificiale multimodale per il riconoscimento di fake news. I campi di applicazione dell'intelligenza artificiale a questo campo sono molteplici, dal momento che le notizie false hanno la capacità di influenzare l'esito delle elezioni, alimentare conflitti sociali, danneggiare reputazioni personali e aziendali, e mettere a rischio la salute e la sicurezza pubblica. Presa una notizia composta da testo e immagini, il modello sviluppato ha lo scopo di classificarla in una di queste quattro categorie: "Certamente Falsa", "Probabilmente Falsa", "Probabilmente Vera", "Certamente Vera". Il problema principale dei modelli multimodali è quello di trovare un modo per combinare adeguatamente le diverse modalità. A tal proposito, lo scopo di questa ricerca è stato lo sviluppo di un approccio per la gestione di testi più lunghi di quelli gestibili dai classici modelli e un metodo per mettere in relazione in maniera efficiente testo e immagini che compongono una notizia. In particolare, il contributo principale è stato ottenuto progettando delle estensioni per un modello allo stato dell'arte che si concentrano sui seguenti obiettivi:

- Analisi del sentimento che evoca il testo di una notizia
- Ottenimento di un bilanciamento delle classi tramite tecniche di data augmentation
- Analisi delle immagini nel dominio della frequenza
- Unione efficiente delle rappresentazioni ottenute da modelli diversi per ogni modalità
- Aggregazione migliore delle varie modalità
- Focalizzazione su regioni di interesse delle immagini

Le estensioni adottate hanno permesso di migliorare il modello utilizzato come base e, le loro combinazioni, hanno ottenuto performance nettamente superiori alle configurazioni

usate come riferimento. Inoltre, un ulteriore progresso è stato raggiunto tramite l'uso di un ensemble di tali modelli.
L'architettura finale proposta si è dimostrata efficace, ottenendo il primo posto alla competizione MULTI-Fake-DetectiVE (MULTImodal Fake News Detection and VErification) organizzata da EVALITA (2023) nel contesto dell'etica computazionale.
Il modello si è inoltre dimostrato robusto al cambio di task, dal momento che ha ottenuto buoni risultati anche ad una seconda task che consisteva nell'identificare se il testo e l'immagine fossero intenzionalmente accoppiati per essere fuorvianti, non fuorvianti o totalmente scorrelati.
Per quanto riguarda eventuali sviluppi futuri, vi è la possibilità di cercare un modo migliore per trovare la relazione tra testo e immagine, dal momento che il modello usato come base utilizza la similarità calcolata sulle rappresentazioni ottenute dal modello CLIP che, tuttavia, è allenato su frasi molto più corte di quanto possa esserlo una notizia, andando a inficiare sul computo della similarità. Inoltre, si potrebbe pensare di creare delle estensioni o un'architettura ad hoc per la task secondaria, in modo da migliorare le prestazioni su quest'ultima. Infine, si potrebbero effettuare ulteriori test per la ricerca di combinazioni delle estensioni e/o iperparametri migliori. In conclusione, il modello sviluppato in questo studio ha dimostrato di essere competitivo, ma vi è ancora spazio per ulteriori miglioramenti.
Spero che questo studio possa costituire una fonte di ispirazione per future ricerche nel campo del riconoscimento delle fake news.

Acronimi

BART Bidirectional Autoregressive Transformer

BERT Bidirectional Encoder Representations from Transformers

BT Back Translation

CF Certamente Falsa

CLIP Contrastive Language-Image Pre-training

CNN Convolutional Neural Network

CV Certamente Vera

DFT Discrete Fourier Transform

FLOP Floating Point Operation

FN False Negative

FND-CLIP Fake News Detection - CLIP

FP False Positive

GD Gradient Descent

GELU Gaussian Error Linear Unit

GPT Generative Pre-trained Transformer

IA Intelligenza artificiale

LBERT Long BERT

LR Learning Rate

LSTM Long Short Term Memory

MLM Masked LM

MLP Multi Layer Perceptron

NER Named Entity Recognition

NLG Natural Language Generation

NLI Natural Language Inference

NLP Natural Language Processing

NLU Natural Language Understanding

NSP Next Sentence Prediction

OCR Optical Character Recognition

PF Probabilmente Falsa

PV Probabilmente Vera

QA Question Answering

ReLU Rectified Linear Unit

RNN Recurrent Neural Network

RoBERTa Robustly Optimized BERT Pretraining Approach

SGD Stochastic Gradient Descent

TN True Negative

TP True Positive

VGG Visual Geometry Group

ViT Vision Transformer

Bibliografia

[1] Stacy Liberatore. Fake image showing an explosion at the pentagon goes viral on twitter - sending markets plummeting, 2023. URL https://www.dailymail.co.uk/sciencetech/article-12112413/Fake-image-showing-explosion-Pentagon-goes-viral-Twitter-sending-markets-plummeting.html. Accessed: 2023-06-15.

[2] Soroush Vosoughi, Deb Roy, and Sinan Aral. The spread of true and false news online. *Science*, 359(6380):1146–1151, 2018. doi: 10.1126/science.aap9559. URL https://www.science.org/doi/abs/10.1126/science.aap9559.

[3] Bhuvanesh Singh and Dilip Kumar Sharma. Predicting image credibility in fake news over social media using multi-modal approach. *Neural Computing and Applications*, 34(24):21503–21517, 2022.

[4] Lisa Fazio. Curbing fake news: Here's why visuals are the most potent form of misinformation, Feb 2020. URL https://scroll.in/article/953395/curbing-fake-news-heres-why-visuals-are-the-most-potent-form-of-misinformation.

[5] Marco Anders. Fake news detection. URL https://edps.europa.eu/press-publications/publications/techsonar/fake-news-detection_en. Accessed: 2023-06-13.

[6] Andreas Kaplan. Artificial intelligence, social media, and fake news: Is this the end of democracy? *IN MEDIA & SOCIETY*, page 149, 2020.

[7] Frank Rosenblatt. The perceptron: a probabilistic model for information storage and organization in the brain. *Psychological review*, 65 6:386–408, 1958.

[8] Matthew Joseph Adiletta and Brian Flanagan. Optimization techniques to improve inference performance of a forward propagating neural network on an fpga, 2020.

[9] Kurt Hornik, Maxwell Stinchcombe, and Halbert White. Universal approximation of an unknown mapping and its derivatives using multilayer feedforward

networks. *Neural Networks*, 3(5):551–560, 1990. ISSN 0893-6080. doi: https://doi.org/10.1016/0893-6080(90)90005-6. URL https://www.sciencedirect.com/science/article/pii/0893608090900056.

[10] George V. Cybenko. Approximation by superpositions of a sigmoidal function. *Mathematics of Control, Signals and Systems*, 2:303–314, 1989.

[11] Kurt Hornik. Approximation capabilities of multilayer feedforward networks. *Neural Networks*, 4(2):251–257, 1991. ISSN 0893-6080. doi: https://doi.org/10.1016/0893-6080(91)90009-T. URL https://www.sciencedirect.com/science/article/pii/089360809190009T.

[12] Shruti Jadon. Introduction to different activation functions for deep learning. URL https://medium.com/@shrutijadon/survey-on-activation-functions-for-deep-learning-9689331ba092. Accessed: 2023-06-23.

[13] Kunihiko Fukushima. Cognitron: A self-organizing multilayered neural network. *Biological Cybernetics*, 20:121–136, 1975.

[14] Tsung-Yi Lin, Priya Goyal, Ross Girshick, Kaiming He, and Piotr Dollár. Focal loss for dense object detection, 2018.

[15] Sebastian Ruder. An overview of gradient descent optimization algorithms, 2017.

[16] Alexander Amini, Ava Soleimany, Sertac Karaman, and Daniela Rus. Spatial uncertainty sampling for end-to-end control, 2019.

[17] Kaichao You, Mingsheng Long, Jianmin Wang, and Michael I. Jordan. How does learning rate decay help modern neural networks?, 2019.

[18] Diederik P. Kingma and Jimmy Ba. Adam: A method for stochastic optimization, 2017.

[19] Ilya Loshchilov and Frank Hutter. Decoupled weight decay regularization, 2019.

[20] David E. Rumelhart, Geoffrey E. Hinton, and Ronald J. Williams. Learning representations by back-propagating errors. *Nature*, 323:533–536, 1986.

[21] Xavier Glorot and Y. Bengio. Understanding the difficulty of training deep feedforward neural networks. *Journal of Machine Learning Research - Proceedings Track*, 9:249–256, 01 2010.

[22] Nitish Srivastava, Geoffrey Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. Dropout: A simple way to prevent neural networks from overfitting. *Journal of Machine Learning Research*, 15(56):1929–1958, 2014. URL http://jmlr.org/papers/v15/srivastava14a.html.

[23] A. Rosenfeld. Computer vision: basic principles. *Proceedings of the IEEE*, 76(8): 863–868, 1988. doi: 10.1109/5.5961.

[24] Richard Szeliski. *Computer vision: algorithms and applications*. Springer Nature, 2022.

[25] Tero Karras, Samuli Laine, and Timo Aila. A style-based generator architecture for generative adversarial networks, 2019.

[26] Xinyu Gong, Shiyu Chang, Yifan Jiang, and Zhangyang Wang. Autogan: Neural architecture search for generative adversarial networks, 2019.

[27] Robin Rombach, Andreas Blattmann, Dominik Lorenz, Patrick Esser, and Björn Ommer. High-resolution image synthesis with latent diffusion models, 2022.

[28] Y. Lecun, L. Bottou, Y. Bengio, and P. Haffner. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324, 1998. doi: 10.1109/5.726791.

[29] Matthew D. Zeiler and Rob Fergus. Visualizing and understanding convolutional networks. In David Fleet, Tomas Pajdla, Bernt Schiele, and Tinne Tuytelaars, editors, *Computer Vision – ECCV 2014*, pages 818–833, Cham, 2014. Springer International Publishing. ISBN 978-3-319-10590-1.

[30] Diksha Khurana, Aditya Koli, Kiran Khatter, and Sukhdev Singh. Natural language processing: State of the art, current trends and challenges. *Multimedia tools and applications*, 82(3):3713–3744, 2023.

[31] Larry R Medsker and LC Jain. Recurrent neural networks. *Design and Applications*, 5:64–67, 2001.

[32] Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. *Neural computation*, 9(8):1735–1780, 1997.

[33] Kyunghyun Cho, Bart van Merrienboer, Dzmitry Bahdanau, and Yoshua Bengio. On the properties of neural machine translation: Encoder-decoder approaches, 2014.

[34] Dhanesh Ramachandram and Graham W. Taylor. Deep multimodal learning: A survey on recent advances and trends. *IEEE Signal Processing Magazine*, 34(6): 96–108, 2017. doi: 10.1109/MSP.2017.2738401.

[35] Said Boulahia, Abdenour Amamra, Mohamed Madi, and Said Daikh. Early, intermediate and late fusion strategies for robust deep learning-based multimodal action recognition. *Machine Vision and Applications*, 32, 11 2021. doi: 10.1007/s00138-021-01249-8.

[36] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay. Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research*, 12:2825–2830, 2011.

[37] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. Bert: Pre-training of deep bidirectional transformers for language understanding, 2019.

[38] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser, and Illia Polosukhin. Attention is all you need, 2017.

[39] Alec Radford and Karthik Narasimhan. Improving language understanding by generative pre-training. 2018.

[40] Karen Simonyan and Andrew Zisserman. Very deep convolutional networks for large-scale image recognition, 2015.

[41] Alex Krizhevsky, Vinod Nair, and Geoffrey Hinton. Cifar-10 (canadian institute for advanced research). URL `http://www.cs.toronto.edu/~kriz/cifar.html`.

[42] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition, 2015.

[43] 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, 2021.

[44] 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, 2021.

[45] Omkar M Parkhi, Andrea Vedaldi, Andrew Zisserman, and C. V. Jawahar. The oxford-iiit pet dataset. URL `https://www.robots.ox.ac.uk/~vgg/data/pets/`.

[46] Yann LeCun, Corinna Cortes, and CJ Burges. Mnist handwritten digit database. *ATT Labs [Online]. Available: http://yann.lecun.com/exdb/mnist*, 2, 2010.

[47] Julio C. S. Reis, André Correia, Fabrício Murai, Adriano Veloso, and Fabrício Benevenuto. Supervised learning for fake news detection. *IEEE Intelligent Systems*, 34(2):76–81, 2019. doi: 10.1109/MIS.2019.2899143.

[48] Z Khanam, B N Alwasel, H Sirafi, and M Rashid. Fake news detection using machine learning approaches. *IOP conference series. Materials Science and Engineering*, 1099(1):12040–, 2021. ISSN 1757-8981.

[49] Rohit Kumar Kaliyar, Anurag Goswami, and Pratik Narang. Fakebert: Fake news detection in social media with a bert-based deep learning approach. *Multimedia Tools Appl.*, 80(8):11765–11788, mar 2021. ISSN 1380-7501. doi: 10.1007/s11042-020-10183-2. URL https://doi.org/10.1007/s11042-020-10183-2.

[50] Shivangi Singhal, Rajiv Ratn Shah, Tanmoy Chakraborty, Ponnurangam Kumaraguru, and Shin'ichi Satoh. Spotfake: A multi-modal framework for fake news detection. In *2019 IEEE Fifth International Conference on Multimedia Big Data (BigMM)*, pages 39–47, 2019. doi: 10.1109/BigMM.2019.00-44.

[51] Balasubramanian Palani, Sivasankar Elango, and Vignesh Viswanathan K. Cb-fake: A multimodal deep learning framework for automatic fake news detection using capsule neural network and bert. *Multimedia Tools Appl.*, 81(4):5587–5620, feb 2022. ISSN 1380-7501. doi: 10.1007/s11042-021-11782-3. URL https://doi.org/10.1007/s11042-021-11782-3.

[52] Jing Jing, Hongchen Wu, Jie Sun, Xiaochang Fang, and Huaxiang Zhang. Multimodal fake news detection via progressive fusion networks. *Information Processing & Management*, 60(1):103120, 2023. ISSN 0306-4573. doi: https://doi.org/10.1016/j.ipm.2022.103120. URL https://www.sciencedirect.com/science/article/pii/S0306457322002217.

[53] Miguel A. Alonso, David Vilares, Carlos Gómez-Rodríguez, and Jesús Vilares. Sentiment analysis for fake news detection. *Electronics*, 10(11), 2021. ISSN 2079-9292. doi: 10.3390/electronics10111348. URL https://www.mdpi.com/2079-9292/10/11/1348.

[54] Carlos Castillo, Marcelo Mendoza, and Barbara Poblete. Information credibility on twitter. In *Proceedings of the 20th International Conference on World Wide Web*, WWW '11, page 675–684, New York, NY, USA, 2011. Association for Computing Machinery. ISBN 9781450306324. doi: 10.1145/1963405.1963500. URL https://doi.org/10.1145/1963405.1963500.

[55] Moreno La Quatra and Luca Cagliero. Bart-it: An efficient sequence-to-sequence model for italian text summarization. *Future Internet*, 15(1), 2023. ISSN 1999-5903. doi: 10.3390/fi15010015. URL https://www.mdpi.com/1999-5903/15/1/15.

[56] Mike Lewis, Yinhan Liu, Naman Goyal, Marjan Ghazvininejad, Abdelrahman Mohamed, Omer Levy, Ves Stoyanov, and Luke Zettlemoyer. Bart: Denoising

sequence-to-sequence pre-training for natural language generation, translation, and comprehension, 2019.

[57] Yinhan Liu, Myle Ott, Naman Goyal, Jingfei Du, Mandar Joshi, Danqi Chen, Omer Levy, Mike Lewis, Luke Zettlemoyer, and Veselin Stoyanov. Roberta: A robustly optimized bert pretraining approach, 2019.

[58] Louis Martin, Benjamin Muller, Pedro Javier Ortiz Suá rez, Yoann Dupont, Laurent Romary, Éric de la Clergerie, Djamé Seddah, and Benoît Sagot. CamemBERT: a tasty french language model. In *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*. Association for Computational Linguistics, 2020. doi: 10.18653/v1/2020.acl-main.645. URL `https://doi.org/10.18653%2Fv1%2F2020.acl-main.645`.

[59] Andriy Mulyar, Elliot Schumacher, Masoud Rouhizadeh, and Mark Dredze. Phenotyping of clinical notes with improved document classification models using contextualized neural language models, 2020.

[60] Yangming Zhou, Qichao Ying, Zhenxing Qian, Sheng Li, and Xinpeng Zhang. Multimodal fake news detection via clip-guided learning, 2022.

[61] Jie Hu, Li Shen, Samuel Albanie, Gang Sun, and Enhua Wu. Squeeze-and-excitation networks, 2019.

[62] Dan Hendrycks and Kevin Gimpel. Gaussian error linear units (gelus), 2020.

[63] Andrew Howard, Mark Sandler, Grace Chu, Liang-Chieh Chen, Bo Chen, Mingxing Tan, Weijun Wang, Yukun Zhu, Ruoming Pang, Vijay Vasudevan, Quoc V. Le, and Hartwig Adam. Searching for mobilenetv3, 2019.

[64] Joel Frank, Thorsten Eisenhofer, Lea Schönherr, Asja Fischer, Dorothea Kolossa, and Thorsten Holz. Leveraging frequency analysis for deep fake image recognition, 2020.

[65] Lorenzo Vaiani, Moreno La Quatra, Luca Cagliero, and Paolo Garza. Leveraging multimodal content for podcast summarization. In *Proceedings of the 37th ACM/SIGAPP Symposium on Applied Computing*, pages 863–870, 2022.

[66] Jiaheng Hua, Xiaodong Cui, Xianghua Li, Keke Tang, and Peican Zhu. Multimodal fake news detection through data augmentation-based contrastive learning. *Applied Soft Computing*, 136:110125, 2023. ISSN 1568-4946. doi: https://doi.org/10.1016/j.asoc.2023.110125. URL `https://www.sciencedirect.com/science/article/pii/S1568494623001436`.

[67] L. Pirandello. *Uno, nessuno e centomila*. Feltrinelli Editore, 2010. ISBN 9788807947964. URL https://books.google.it/books?id=0uRAboip_VIC.

[68] Yuxin Wu, Alexander Kirillov, Francisco Massa, Wan-Yen Lo, and Ross Girshick. Detectron2. https://github.com/facebookresearch/detectron2, 2019.

[69] Tsung-Yi Lin, Michael Maire, Serge J. Belongie, Lubomir D. Bourdev, Ross B. Girshick, James Hays, Pietro Perona, Deva Ramanan, Piotr Doll'a r, and C. Lawrence Zitnick. Microsoft COCO: common objects in context. *CoRR*, abs/1405.0312, 2014. URL http://arxiv.org/abs/1405.0312.

[70] Agrim Gupta, Piotr Dollár, and Ross Girshick. Lvis: A dataset for large vocabulary instance segmentation, 2019.

[71] Liulei Li, Tianfei Zhou, Wenguan Wang, Jianwu Li, and Yi Yang. Deep hierarchical semantic segmentation, 2022.

[72] Jun-Ho Choi and Jong-Seok Lee. Embracenet: A robust deep learning architecture for multimodal classification. *Information Fusion*, 51:259–270, 2019.

[73] Federico Bianchi, Debora Nozza, and Dirk Hovy. "FEEL-IT: Emotion and Sentiment Classification for the Italian Language". In *Proceedings of the 11th Workshop on Computational Approaches to Subjectivity, Sentiment and Social Media Analysis*. Association for Computational Linguistics, 2021.

[74] Gabriele Sarti and Malvina Nissim. IT5: Large-scale text-to-text pretraining for italian language understanding and generation. *ArXiv preprint 2203.03759*, mar 2022. URL https://arxiv.org/abs/2203.03759.