UNIVERSITÀ DEGLI STUDI DI PALERMO

DIPARTIMENTO DI INGEGNERIA

LAUREA MAGISTRALE IN INGEGNERIA INFORMATICA - INTELLIGENZA ARTIFICIALE

TECNICHE DI AML PER LA GENERAZIONE DI ATTACCHI QUERY EFFICIENT

Tesi di Laurea di
Giuseppe Ganci

Relatore:
Prof. Marco Morana

Controrelatore:
Prof. ?

Correlatore:
?

Anno Accademico 2022/2023

Indice

Introduzione **3**

1 Stato dell'arte **4**

2 Formulazione matematica del problema **5**

2.1 Formulazione della ricerca 6

2.1.1 Branching 7

2.1.2 Bound 8

2.2 Formulazione delle Azioni di ricerca 9

2.3 Derivazione del criterio di importanza 11

2.4 Fisher Information Matrix ed Hessiano 13

3 Metodo adottato **16**

4 Esperimenti e risultati **17**

4.1 Test sul dataset CIFAR-100 17

4.1.1 Test con ViT-Small 19

5 Caso di studio: deploy su mobile **21**

Conclusioni **22**

Ringraziamenti **23**

Elenco delle figure **24**

Bibliografia **25**

Introduzione

Capitolo 1

Stato dell'arte

Capitolo 2

Formulazione matematica del problema

Un problema di Neural Architecture Search (NAS) consiste in una ricerca in uno spazio degli stati, a partire da una architettura di partenza, detta **Supernet**, per identificare, tra tutte le possibili **Subnet**, quella che massimizza una determinata funzione obiettivo. In questo caso, al fine di ottenere una compressione di modelli *Transformer*, la ricerca si configura come un problema di **ottimizzazione multi-obiettivo**, tramite il quale vengono ottimizzate sia la performance del modello (attraverso l'accuratezza) sia la dimensione (attraverso il numero di parametri).
Uno dei problemi principali nell'ambito NAS è proprio la dimensione dello spazio degli stati, che risulta elevatissima, poiché il numero di possibili configurazioni cresce in maniera combinatoria rispetto al numero di componenti del modello. Nei Vision Transformer, ogni blocco Transformer presenta molteplici gradi di libertà: è possibile variare il numero di teste di attenzione nell'unità di *Multi-Head Attention* (MHSA), la dimensione dei vettori *Query-Key* e *Value*, ma anche la dimensione del livello nascosto nel modulo *Multi-Layer Perceptron* (MLP), ecc... Anche considerando un numero limitato di opzioni per ogni strato, il numero totale di subnet candidate diventa rapidamente intrattabile per una ricerca esaustiva.
Proprio per rendere trattabile l'esplorazione dello spazio degli stati è stata scelta una formulazione conveniente del problema, attraverso un algoritmo di ricerca *Branch and Bound* e l'utilizzo di azioni di *pruning* predefinite. Di seguito la formulazione matematica nel dettaglio.

2.1 Formulazione della ricerca

Dato un modello ViT M con parametri θ, definiamo un processo iterativo di pruning. Per ogni iterazione viene eseguita una ricerca di tipo *Depth First* a profondità limitata, ottimizzata tramite algoritmo *Branch and Bound*. Durante la ricerca verranno selezionate delle azioni da eseguire sul modello, e successivamente, attraverso un secondo problema combinatorio vengono selezionate le specifiche dimensioni da eliminare. L'obiettivo è massimizzare la seguente funzione rispetto alla maschera binaria m, ai parametri θ e al dataset di validazione D:

$$\max_{m} \text{Obj}(m,\theta,D) = \log_2(\mathscr{A}(m,\theta,D)) - \lambda \cdot \log_2\left(\frac{\mathscr{P}(m,\theta)}{\mathscr{P}_{tot}(\theta)}\right) \tag{2.1}$$

Dove:

- $m \in \{0,1\}^N$ è la maschera di pruning globale, con N numero totale di parametri.
- La maschera è strutturata per blocchi: $m = \{m_{\text{emb}}, m^{(1)}, \ldots, m^{(i)}, \ldots, m^{(L)}\}$, con L numero di layer.
- Per ogni blocco i: $m^{(i)} = \{m_{\text{head}}, m_{\text{q\_k}}, m_{\text{v\_proj}}, m_{\text{mlp}}\}$.
- $\mathscr{A}(m,\theta,D)$ indica l'accuratezza del modello mascherato sul dataset D.
- $\mathscr{P}(m,\theta)$ rappresenta il numero di parametri attivi (non zero).
- $\mathscr{P}_{tot}(\theta)$ rappresenta il numero totale di parametri del modello originale.

Notare che tramite questa formulazione è possibile ricondurre il problema in esame ad un **Problema di Programmazione Non Lineare Binario** (PNLPB). In questo contesto, le variabili decisionali sono rappresentate dai componenti della maschera $m \in \{0,1\}^N$, mentre la non linearità è introdotta sia dalla natura della funzione di accuratezza $\mathscr{A}$ (legata ai pesi della rete neurale) sia dall'operatore logaritmico utilizzato nella funzione obiettivo. Nonostante esistano in letteratura delle tecniche di *Smoothing* delle variabili binarie per risolvere questo tipo di problemi combinatori, come quelle utilizzate da Murray et al. [4], i tempi di risoluzione, con un numero ridotto di variabili binarie rispetto al caso in esame, sono molto elevati. Proprio per tale motivo si è scelto di utilizzare un differente approccio algoritmico basato sul *Branch and Bound*.

2.1.1 Branching

L'esplorazione dello spazio di ricerca avviene tramite la costruzione di un apposito albero, seguendo un algoritmo di esplorazione *Depth First* a profondità limitata; di conseguenza, è possibile definire la strategia di esplorazione come **Depth Limited Depth First Branch and Bound** (DL-DFBnB).
La radice di questo albero è rappresentata dalla *Supernet*, ovvero il modello *Vision Transformer* di dimensione originale, specializzato su uno specifico dominio tramite *fine-tuning*. A partire dalla radice, viene eseguita la fase di **Branching** applicando le azioni di seguito elencate:

- **Pruning Multi-Layer Perceptron**
- **Pruning Query-Key**
- **Pruning Value-Projection**
- **Pruning Head**
- **Pruning Embedding**

Si noti che tutte le azioni, ad eccezione del *Pruning* dell'*Embedding*, sono locali a uno specifico blocco *Transformer*. La formalizzazione matematica delle singole azioni è rimandata alla sezione successiva.
Attraverso il *branching* viene costruito un **albero 5-ario** di ricerca, dove ogni nodo rappresenta una specifica *Subnet* ottenuta a partire dal nodo genitore. Ad ogni nodo viene associato un valore della funzione obiettivo, calcolato valutando l'accuratezza sul *Search Set* (utilizzando esclusivamente i parametri attivi) e il relativo numero di parametri residui. Questo valore verrà successivamente utilizzato per verificare i vincoli di *Bound* e, qualora risultasse più elevato del miglior valore individuato dalla ricerca, si aggiornerebbe quest'ultimo e si salverebbe la maschera di *Pruning* corrispondente.
Al fine di massimizzare l'efficacia della potatura, la strategia di esplorazione *Depth First* è stata scelta per favorire il raggiungimento dei nodi foglia dell'albero di ricerca. Questo permette all'algoritmo di individuare rapidamente configurazioni caratterizzate da un elevato numero di azioni di *pruning* e, di conseguenza, da una significativa riduzione dei parametri.
L'introduzione di un limite di profondità nell'albero di ricerca permette di mitigare l'impatto cumulativo del *pruning* sulle prestazioni del modello. Un'esplorazione eccessivamente profonda,

infatti, comporterebbe una rimozione massiva di parametri, rischiando di degradare l'accuratezza in modo irreversibile. In tali scenari, il danno strutturale all'architettura potrebbe risultare troppo elevato, rendendo difficoltoso il recupero delle performance originali anche attraverso l'impiego di tecniche avanzate come la *Knowledge Distillation*.

2.1.2 Bound

La dimensione dell'albero di ricerca risulta significativa; in particolare, detta $\mathscr{D}$ la massima profondità dell'albero, e considerato che si tratta di un albero 5-ario, il numero totale di nodi è pari a:

$$\mathscr{N} = \sum_{i=0}^{\mathscr{D}} 5^i \tag{2.2}$$

Ad esempio, se $\mathscr{D} = 6$, il numero totale di nodi è pari a 19.531. Considerando che per ogni nodo è necessario calcolare i gradienti per tutti i parametri attivi del modello al fine di stimare l'importanza, il tempo necessario per una ricerca esaustiva risulterebbe proibitivo. Tale criticità viene risolta attraverso la fase di **Bound**, che permette di effettuare un *pruning on-the-fly* sull'albero di ricerca, escludendo interi rami computazionalmente onerosi ma poco promettenti dal punto di vista della funzione obiettivo.
In particolare, per ogni nodo n esplorato, viene verificato il seguente vincolo di ammissibilità:

$$\text{Obj}(n) > \text{Obj}_{best} - \varepsilon \tag{2.3}$$

Qualora il vincolo venisse violato, il nodo non verrebbe sottoposto a *Branching*, troncando di fatto l'esplorazione di quel ramo. È importante sottolineare la presenza del **fattore di tolleranza** ε: esso consente alla ricerca di non arrestarsi prematuramente, permettendo l'esplorazione di subnet che, pur essendo leggermente inferiori al miglior valore individuato (Obj_{best}), si trovano in regioni dello spazio degli stati potenzialmente ottimali. Questo approccio garantisce una ricerca più robusta e completa, anche se maggiormente onerosa in termini computazionali.

2.2 Formulazione delle Azioni di ricerca

Per ogni nodo dell'albero di ricerca, l'applicazione delle azioni di *pruning* genera i cinque nodi figli corrispondenti. Ciascuna azione identifica un gruppo specifico di parametri $\mathscr{W}_g$ all'interno dell'architettura che può essere rimosso minimizzando il degrado delle prestazioni del modello.
Al fine di individuare le componenti meno rilevanti per le capacità predittive del modello, viene adottata una metrica di importanza basata sulla sensibilità della funzione di perdita rispetto ai parametri. Per un singolo parametro w, l'importanza è definita come:

$$\mathscr{I}(w) = w^2 \cdot \mathbb{E}_{x\sim\mathscr{D}}\left[\left(\frac{\partial \mathscr{L}(x;\theta^*)}{\partial w}\right)^2\right] \tag{2.4}$$

Dove $\mathscr{L}(x;\theta^*)$ è proprio la funzione di perdita, calcolata a partire dai parametri attivi θ^*, sul *Search Set* (x). Operando un *pruning strutturato*, l'importanza di un intero gruppo di parametri $\mathscr{W}_g$ (come un intero neurone) viene calcolata come la somma dei contributi individuali:

$$\mathscr{I}(\mathscr{W}_g) = \sum_{w\in\mathscr{W}_g} \mathscr{I}(w) \tag{2.5}$$

Le specifiche azioni di *pruning* possono essere definite come un problema di minimizzazione locale volto a identificare il gruppo di dimensioni (formate da più parametri) g la cui rimozione minimizza l'impatto sulla funzione di perdita $\mathscr{L}$. Per garantire la compatibilità tra il modello compresso e l'hardware pensato per l'accelerazione di questi modelli (es. Tensor Core), imponiamo un vincolo di cardinalità $|g| = K$ (con K multiplo di 8). Di seguito la formulazione matematica dettagliata per ogni singola azione:

- **Pruning MLP:** Sia $\mathscr{B}$ l'insieme dei blocchi *Transformer* e $MLP_{b_i} = \{g \subset \mathscr{W}_{MLP,b_i} : |g| = K\}$ l'insieme dei possibili raggruppamenti di cardinalità K di dimensioni del modulo MLP nel blocco i-esimo. L'azione identifica il gruppo g^*_{MLP} che minimizza:

$$g^*_{\text{MLP}} = \arg \min_{\substack{b_i\in\mathscr{B} \\ g_K\in MLP_{b_i}}} \sum_{W_g\in g_K} \mathscr{I}(W_g) \tag{2.6}$$

- **Pruning QK (Query-Key):** Sia $\mathscr{B}$ l'insieme dei blocchi *Transformer* e $QK_{b_i} = \{g \subset \mathscr{W}_{QK,b_i} : |g| = K\}$ l'insieme dei raggruppamenti di K dimensioni condivise tra i moduli Query e Key di tutte le teste di attenzione del blocco i-esimo. L'azione identifica il gruppo g^*_{QK} che minimizza:

$$g^*_{\text{QK}} = \arg \min_{\substack{b_i \in \mathscr{B} \\ g_K \in QK_{b_i}}} \sum_{W_g \in g_K} \mathscr{I}(W_g) \tag{2.7}$$

- **Pruning VPROJ (Value-Projection):** Sia $\mathscr{B}$ l'insieme dei blocchi *Transformer* e $VP_{b_i} = \{g \subset \mathscr{W}_{VP,b_i} : |g| = K\}$ l'insieme dei raggruppamenti di cardinalità K di dimensioni dei moduli *Value* e *Output Projection* di tutte le teste di attenzione nel blocco i-esimo. L'azione identifica il gruppo g^*_{VP} che minimizza:

$$g^*_{\text{VP}} = \arg \min_{\substack{b_i \in \mathscr{B} \\ g_K \in VP_{b_i}}} \sum_{W_g \in g_K} \mathscr{I}(W_g) \tag{2.8}$$

- **Pruning HEAD:** Sia $\mathscr{B}$ l'insieme dei blocchi *Transformer* e $H_{b_i} = \{h \in HEADS_{b_i}\}$ l'insieme di tutte le teste di attenzione nel blocco i-esimo. L'azione identifica la head h^* che minimizza:

$$h^* = \arg \min_{\substack{b_i \in \mathscr{B} \\ h \in H_{b_i}}} \sum_{W_g \in h} \mathscr{I}(W_g) \tag{2.9}$$

- **Pruning EMB (Embedding):** Sia $EMB = \{g \subset \mathscr{W}_{EMB} : |g| = K\}$ l'insieme di tutti i raggruppamenti di cardinalità K di dimensioni dell'*Embedding* nell'intero modello *Transformer*. L'azione identifica il gruppo g^*_{EMB} che minimizza:

$$g^*_{\text{EMB}} = \arg \min_{g_K \in EMB} \sum_{W_g \in g_K} \mathscr{I}(W_g) \tag{2.10}$$

  Si noti che, a differenza delle azioni precedenti, il *Pruning* dell'*Embedding* opera sulla dimensione globale del modello. La rimozione di un raggruppamento g_K in questa fase

comporta una potatura della rappresentazione vettoriale che si propaga attraverso l'intera architettura, influenzando la dimensione di tutti i blocchi *Transformer*.

2.3 Derivazione del criterio di importanza

La metrica di importanza utilizzata combina le informazioni sulla magnitudo dei pesi e dei gradienti, al fine di misurare la sensibilità di uno specifico parametro, a partire dai dati in input al modello. I parametri meno sensibili vengono interpretati come di scarso contributo all'output del modello, e di conseguenza vengono potati. La derivazione della metrica di importanza considera un singolo parametro w e il suo effetto sulla funzione di perdita L. In particolare si analizza la variazione di L al variare di w:

$$\Delta L = L(w + \delta w) - L(w) \tag{2.11}$$

$$L(w + \delta w) = L(w) + \Delta L \tag{2.12}$$

Per approssimare il valore della *loss function* con il nuovo valore del parametro $L(w+\delta w)$ si utilizza l'espansione di Taylor al primo ordine:

$$L(w + \delta w) \approx L(w) + \frac{\partial L}{\partial w} \cdot \delta w + \mathcal{O}(w) \tag{2.13}$$

Dato che l'operazione effettuata è la potatura, il valore del parametro perturbato è zero (in quanto viene tagliato) e di conseguenza $\delta w = -w$, da cui possiamo ottenere il valore approssimato della variazione di L:

$$L(w + \delta w) - L(w) \approx (-w) \cdot \frac{\partial L}{\partial w} \tag{2.14}$$

Applicato ad un gruppo di parametri W_g (ad esempio un intero neurone):

$$\Delta L(W_g) = \sum_{w \in W_g} \Delta L(w) = \sum_{w \in W_g} -w \cdot \frac{\partial L}{\partial w} = - \sum_{w \in W_g} w \cdot \frac{\partial L}{\partial w} \tag{2.15}$$

Nonostante l'approssimazione al primo ordine sia ampiamente sfruttata in letteratura per la sua semplicità computazionale, essa introduce un rischio sistematico nel caso del pruning strutturato. Si consideri un gruppo di parametri W_g contenente due parametri w_1 e w_2. Nell'eventualità

in cui le variazioni individuali della perdita siano opposte, ovvero $\Delta L(w_1) \approx -\Delta L(w_2)$, la loro somma algebrica risulterebbe prossima allo zero:

$$\Delta L(W_g) = \sum_{w \in W_g} \Delta L(w) \approx 0 \tag{2.16}$$

Tale risultato porterebbe l'algoritmo a classificare erroneamente il gruppo W_g come irrilevante, quando in realtà è possibile che i singoli parametri possiedano un'elevata sensibilità. Poiché ΔL è una stima lineare e locale della reale superficie della funzione di perdita, fare affidamento sulla somma dei contributi segnati espone a decisioni di potatura errate, dove componenti critiche vengono rimosse a causa di una reciproca compensazione dei gradienti.
La soluzione a questo problema può essere individuata in un approccio più conservativo, utilizzando i quadrati per permettere la somma dei contributi individuali (che saranno così sempre positivi):

$$\mathscr{I}(W_g) = \sum_{w \in W_g} \Delta L(w)^2 = \sum_{w \in W_g} \left(w \cdot \frac{\partial L}{\partial w} \right)^2 \tag{2.17}$$

Notare che il calcolo del gradiente $\frac{\partial L}{\partial w}$ non è banale, infatti accumulando semplicemente i gradienti, si rischia di incorrere in un fenomeno di cancellazione, ad esempio detti b_1 e b_2 i due (unici) *batch* di dati in input, potrebbe accadere che:

$$\frac{\partial L}{\partial w}(b_1) \approx -\frac{\partial L}{\partial w}(b_2) \tag{2.18}$$

Che in fase di accumulo, porterebbe ad un gradiente complessivo circa nullo, azzerando erroneamente l'importanza del parametro.
La soluzione a questo problema può essere facilmente individuata nell'utilizzo del valore atteso di $g^2 = \left(\frac{\partial L}{\partial w} \right)^2$, da cui:

$$\mathscr{I}(w) = w^2 \cdot \mathbb{E}_{x \sim \mathscr{D}}[g^2] = w^2 \cdot \frac{1}{N} \sum_{i=1}^{N} g_i^2 \quad \text{(medie sui batch)} \tag{2.19}$$

con N numero di *batch* in input e g_i il gradiente della funzione di perdita, calcolato sul *batch* i-esimo.
Date queste premesse, è possibile definire la seguente metrica di importanza per un gruppo di

parametri W_g:

$$\mathscr{I}(W_g) = \sum_{w \in W_g} w^2 \cdot \mathbb{E}_{x \sim \mathscr{D}} \left[\left(\frac{\partial L(x)}{\partial w} \right)^2 \right] \tag{2.20}$$

2.4 Fisher Information Matrix ed Hessiano

La metrica di importanza derivata nella sezione precedente combina l'informazione fornita dalla magnitudo dei parametri con l'approssimazione diagonale della **Matrice di Informazione di Fisher**, estremamente utilizzata per determinare l'importanza dei parametri nelle reti neurali (ad es. [3] [5]), così definita:

$$F(w) = \mathbb{E}_{x,y} \left[\nabla L(y, f(x,w)) \cdot \nabla L(y, f(x,w))^T \right] \in \mathbb{R}^{|w| \times |w|} \tag{2.21}$$

Nel quale i singoli elementi rappresentano la correlazione tra i gradienti dei singoli parametri:

$$F(w_i, w_j) = \mathbb{E}_{x,y} \left[\frac{\partial L}{\partial w_i} \cdot \frac{\partial L}{\partial w_j} \right] \tag{2.22}$$

Data la cardinalità dei parametri nei modelli *Transformer*, il calcolo e la memorizzazione della matrice completa risultano computazionalmente proibitivi. Si assume pertanto l'indipendenza tra i parametri, operazione che equivale a considerare esclusivamente la diagonale della matrice:

$$F(w_i, w_j) = 0 \quad \text{ed} \quad F(w_i, w_i) = \mathbb{E}_{x,y} \left[\left(\frac{\partial L}{\partial w_i} \right)^2 \right]$$

Empiricamente, tale valore, detto **Informazione di Fisher**, viene stimato tramite una semplice media su un numero N di *batch* (x_k) in input:

$$F(w) = \frac{1}{N} \sum_{k=1}^{N} \left(\frac{\partial L(x_k)}{\partial w} \right)^2 \tag{2.23}$$

Notare che esiste un profondo legame tra l'*Informazione di Fisher* e l'*Hessiano* della funzione di perdita. Ad una simile conclusione sono giunti anche Theis et. al [5]. Di seguito la dimostrazione formale di questo legame, assumendo che la funzione di perdita sia la *Negative Log-Likelihood*.

Consideriamo un problema di classificazione in cui il modello definisce una distribuzione di probabilità $p(y|x,w)$. Per la proprietà di normalizzazione, la somma delle probabilità su tutte le classi $y \in Y$ è unitaria:

$$\sum_{y \in Y} p(y|x,w) = 1 \tag{2.24}$$

Derivando rispetto ai parametri w:

$$\nabla_w \sum_y p(y|x,w) = 0 \implies \sum_y \nabla_w p(y|x,w) = 0 \tag{2.25}$$

Utilizzando una semplice proprietà delle derivate, per cui $\nabla p = p\frac{\nabla p}{p} = p\nabla \log(p)$, possiamo riscrivere l'espressione come:

$$\sum_y p(y|x,w) \cdot \nabla_w \log p(y|x,w) = 0 \tag{2.26}$$

Per brevità di notazione, assumiamo $p(y|x,w) = p(y)$ e deriviamo una seconda volta rispetto a w:

$$\nabla_w \sum_y p(y) \cdot \nabla_w \log p(y) = 0 \tag{2.27}$$

Applicando la regola del prodotto:

$$\sum_y \left[\nabla_w p(y) \cdot \nabla_w \log p(y)^T + p(y) \cdot \nabla_w^2 \log p(y)\right] = 0 \tag{2.28}$$

Sostituendo nuovamente $\nabla_w p(y) = p(y)\nabla_w \log p(y)$, otteniamo:

$$\sum_y \left[p(y)\nabla_w \log p(y) \cdot \nabla_w \log p(y)^T + p(y) \cdot \nabla_w^2 \log p(y)\right] = 0 \tag{2.29}$$

Raccogliendo $p(y)$, che rappresenta la distribuzione di probabilità della variabile aleatoria y nel valore atteso $\mathbb{E}_y$, arriviamo alla relazione fondamentale:

$$\sum_y p(y) \left[\nabla_w \log p(y) \nabla_w \log p(y)^T + \nabla_w^2 \log p(y)\right] = 0 \tag{2.30}$$

Da questa relazione, si può notare come $\nabla_w \log p(y) \nabla_w \log p(y)^T$ sia proprio il termine legato alla Matrice di Fisher, mentre $\nabla_w^2 \log p(y)$ è il termine legato all'Hessiano. In particolare, si

evince che il valore atteso della Fisher Information e dell'Hessiano sono legati dalla relazione:

$$\mathbb{E}_{x,y}[\text{Fisher}] + \mathbb{E}_{x,y}[\text{Hessiano}] = 0 \implies \mathbb{E}_{x,y}\left[\nabla \log p \nabla \log p^T\right] = -\mathbb{E}_{x,y}\left[\nabla^2 \log p\right] \tag{2.31}$$

Considerato che è assunto l'utilizzo della funzione di perdita Cross-Entropy, che corrisponde alla Negative Log-Likelihood (NLL) nel caso della classificazione:

$$L = -\log p(y_{true}|x, w) \tag{2.32}$$

Le derivate prima e seconda rispetto ai parametri sono dunque:

$$\nabla L = -\nabla \log p, \quad \nabla^2 L = -\nabla^2 \log p \tag{2.33}$$

Sostituendo queste identità nella relazione precedente, otteniamo l'equivalenza finale:

$$\mathbb{E}_{x,y}\left[\nabla L \cdot \nabla L^T\right] = \mathbb{E}_{x,y}\left[\nabla^2 L\right] \tag{2.34}$$

Il che implica, per le singole componenti diagonali:

$$\mathbb{E}_{x,y}\left[\left(\frac{\partial L}{\partial w_i}\right)^2\right] = \mathbb{E}_{x,y}\left[\frac{\partial^2 L}{\partial w_i^2}\right] \tag{2.35}$$

Considerazioni teoriche

Sebbene la teoria garantisca che la matrice di Fisher approssimi l'Hessiano, è importante sottolineare che tale equivalenza è strettamente valida solo quando il modello rappresenta correttamente la distribuzione dei dati reali, ovvero in condizioni di convergenza, come descritto approfonditamente da Kunstner et al. [2]. Nel metodo proposto, questa assunzione è considerata verificata poiché l'estrazione della metrica di importanza avviene su modelli che sono stati sottoposti ad una fase di fine-tuning. In tale stato di stabilità e convergenza, la *Fisher Information* fornisce una stima affidabile della curvatura della funzione di perdita, permettendo di identificare con precisione i parametri la cui rimozione minimizza l'impatto sulle prestazioni del Transformer.

Capitolo 3

Metodo adottato

Capitolo 4

Esperimenti e risultati

In questo capitolo vengono presentati i risultati ottenuti dal framework realizzato, tramite una serie di esperimenti che coinvolgono modelli e dataset differenti.
Tutti gli esperimenti vengono svolti con modelli **ViT**, preaddestrati sul dataset **ImageNet**, che lavorano con *patch* di dimensione **16×16** e immagini in *input* di tipo **RGB** con risoluzione **224×224**. La procedura utilizzata consta di due fasi:

- **FineTuning**: il modello preaddestrato su *ImageNet* viene sottoposto ad una fase di *Fine-Tuning* sul dataset di riferimento per l'esperimento, in questo modo si ottiene allo stesso tempo una **baseline** per confrontare le performance, ma anche un modello di partenza sul quale eseguire il framework di compressione.
- **Compressione**: una volta ottenuto il nuovo modello specializzato sul dominio del problema, viene eseguito il framework di **NAS iterativo** sviluppato, e vengono infine analizzate le performance sul *set* di validazione del dataset di riferimento.

Di seguito vengono presentati i risultati dei vari esperimenti effettuati.

4.1 Test sul dataset CIFAR-100

Il primo dataset utilizzato per testare il framework è stato il **CIFAR100**, che è stato selezionato poiché molto diffuso in letteratura, in cui è utilizzato come **benchmark standard** per compiti di classificazione. Risulta più complesso rispetto al *CIFAR10*, ma mantiene dimensioni ridotte e quindi permette una ottima rapidità di *training*. In particolare *CIFAR100* contiene **60.000**

immagini totali, suddivise in **50.000** immagini di addestramento e **10.000** di validazione, tutte della risoluzione di **32**×**32** e di tipo **RGB**.
Data la ridotta dimensione del dataset, e la grande capacità dei modelli *Vision Transformer*, per scongiurare il rischio di *overfitting*, sono state adottate delle tecniche di **Data Augmentation**, in particolare:

- **Random Resized Crop**: Questa tecnica estrae una porzione casuale dell'immagine originale con un'area compresa tra l'**80%** e il **100%** della dimensione iniziale. Successivamente, il ritaglio viene ridimensionato alla risoluzione *target* di 224×224 pixel.
- **Random Horizontal Flip**: Consiste nel riflettere l'immagine orizzontalmente con una probabilità del **50%**.
- **Color Jitter**: Questa trasformazione applica variazioni casuali alla luminosità, al contrasto e alla saturazione dell'immagine, con un fattore di distorsione impostato a **0.3** per ciascun parametro.

Per i test sul dataset *CIFAR100* è stata effettuata una ulteriore suddivisione del *set* di addestramento con rapporto **90/10**, in modo tale da ottenere un **Train Set** finale di **45.000** immagini e un **Held-Out Set** di **5.000** immagini, quest'ultimo utilizzato per il monitoraggio del processo di compressione.
Al fine di rendere il processo di *Neural Architecture Search* (**NAS**) efficiente e rendere ragionevoli i tempi di esecuzione, ad ogni iterazione viene campionato dal *Train Set* un insieme di **25 immagini per classe** (**2500** totali), in modo da ottenere una buona stima del gradiente, senza compromettere la velocità di esplorazione dello spazio architetturale. Questo insieme costituisce il **Search Set** ed è soggetto alle stesse tecniche di *Augmentation* viste in precedenza, in modo da evitare che il framework individui architetture carenti in termini di **robustezza** e **generalizzazione**. Il campionamento dinamico del *Search Set* ad ogni iterazione è una scelta progettuale essenziale per garantire la robustezza del processo di ricerca. Questa strategia evita che l'algoritmo converga verso architetture eccessivamente specializzate su un *set* statico di campioni, promuovendo invece l'individuazione di *subnet* capaci di generalizzare correttamente sull'intero dominio di *CIFAR-100*. Il ruolo di questo *Search Set* è duplice:

- **Stima della metrica di Importanza**: viene utilizzato per calcolare il valore di importanza di ogni parametro di una *subnet*, corrispondente ad uno specifico nodo della ricerca *Branch and Bound*.

- **Valutazione della Funzione Obiettivo**: viene utilizzato per calcolare l'accuratezza della *subnet*, coinvolta nel calcolo della funzione obiettivo del processo di ricerca.

4.1.1 Test con ViT-Small

In questi test è stato utilizzato un modello **ViT-Small** da **21.7 Mln** di parametri, a cui è stata aggiunta una testa di classificazione da **100 unità**, e successivamente sottoposto a *Fine Tuning* iniziale sul dataset in oggetto, con i seguenti iperparametri:

- **Ottimizzatore AdamW** con un fattore di **decadimento dei pesi** pari a **0.05** e con **learning rate discriminativi**: 0.5×10^{-5} per il *backbone* del *Transformer* e 0.5×10^{-4} per la testa di classificazione.
- **Scheduler** dei *learning rate* di tipo **Cosine Annealing**.
- Dimensione dei **batch** di *training* pari a **128** campioni.
- **30 epoche** di addestramento con **early stopping**.

Per quanto riguarda il framework, tutti i test sono stati effettuati con **15 iterazioni** di compressione. In ogni iterazione, la **profondità** dell'albero di ricerca è stata limitata a **6**, questo implica che, per ogni iterazione, al modello possono essere applicate massimo **6 azioni consecutive** di *pruning*. Viene inoltre utilizzata una **soglia di tolleranza** della fase di *Bound* pari a **0.005**, in modo tale da permettere una ricerca più approfondita dello spazio di ricerca, al costo di un numero lievemente maggiore di nodi esplorati. Infine, la **funzione obiettivo** utilizza un valore di λ pari a **1.0**, attribuito al contributo dei parametri.
La fase di **Recovery Fine-Tuning**, volta a ripristinare le prestazioni dopo il taglio dei parametri, adotta la medesima configurazione di ottimizzazione del *Fine-Tuning* iniziale, con due variazioni: l'impiego di un **learning rate unico** pari a 0.5×10^{-5} per l'intera rete e un limite massimo di **20 epoche** di addestramento, sufficienti per stabilizzare i pesi della *subnet* individuata.
Infine, per quanto concerne i test con la **Knowledge Distillation** (**KD**), è stata implementata una versione *logit-based* seguendo l'approccio proposto da *Hinton et al.* [1]. In questo caso si è scelto di utilizzare il *ViT-Small baseline* come modello **teacher**. La funzione di **loss composita** utilizzata durante il *Recovery Fine-Tuning* integra un termine di distillazione con temperatura $\tau = 4.0$. A tale componente è stato assegnato un **peso relativo pari a** 0.9, privilegiando così il

trasferimento della conoscenza dal modello *teacher* alla *subnet* compressa. Questa configurazione ha permesso di ammorbidire le distribuzioni di probabilità dei *logits*, consentendo al modello *student* di apprendere non solo le etichette corrette, ma anche le relazioni strutturali tra le classi catturate dal modello completo. Di seguito i risultati dei test.

Tabella 4.1: Risultati della compressione di ViT-Small su CIFAR-100.

Modello	**Parametri (M)**	**Top-1 Acc. (%)**	**GFLOPs**	**Throughput (img/s)**	**Latency (ms)**
Baseline	21.67	90.42%	9.20	1137.5	112.5
Pruned	**15.93** (-26.6%)	**89.59%** (-0.83%)	**6.85** (-25.51%)	**1405.5** (+23.56%)	**91.1** (-19.02%)
Pruned + KD	**15.78** (-27.3%)	**89.86%** (-0.56%)	**6.79** (-26.19%)	**1406** (+23.6%)	**91.0** (-19.11%)

Capitolo 5

Caso di studio: deploy su mobile

Conclusioni

Ringraziamenti

Elenco delle figure

Bibliografia

[1] G. Hinton, O. Vinyals e J. Dean. *Distilling the Knowledge in a Neural Network*. 2015. arXiv: 1503.02531 [stat.ML]. URL: https://arxiv.org/abs/1503.02531.

[2] F. Kunstner, L. Balles e P. Hennig. *Limitations of the Empirical Fisher Approximation for Natural Gradient Descent*. 2020. arXiv: 1905.12558 [cs.LG]. URL: https://arxiv.org/abs/1905.12558.

[3] P. Molchanov, A. Mallya, S. Tyree, I. Frosio e J. Kautz. *Importance Estimation for Neural Network Pruning*. 2019. arXiv: 1906.10771 [cs.LG]. URL: https://arxiv.org/abs/1906.10771.

[4] W. Murray e K.-M. Ng. *An algorithm for nonlinear optimization problems with binary variables*. 2010. URL: https://link.springer.com/article/10.1007/s10589-008-9218-1.

[5] L. Theis, I. Korshunova, A. Tejani e F. Huszár. *Faster gaze prediction with dense networks and Fisher pruning*. 2018. arXiv: 1801.05787 [cs.CV]. URL: https://arxiv.org/abs/1801.05787.