Università degli Studi di Bergamo

Dipartimento di Ingegneria Gestionale, dell'Informazione e della Produzione

Corso di laurea in Ingegneria Informatica

Valutazione delle prestazioni dei Vision Transformer per la classificazione di immagini

Candidato:

Alessandro Belotti

Matricola n.

1066721

Relatore:

Prof. Michele Ermidoro

Correlatore:

Prof. Leandro Pitturelli

Anno Accademico

2021/2022

ABSTRACT

L'evoluzione tecnologica nel campo del Machine Learning è molto rapida, la ricerca scientifica produce numerosi modelli di reti neurali e tecniche di Deep Learning sempre più performanti in termini di accuratezza e velocità.
In questa trattazione farò una breve introduzione in cui verrà definito cosa è la Computer Vision e in che ramo dell'Intelligenza Artificiale si colloca. Presenterò poi una delle più recenti tecniche di Machine Learning chiamata Attention, come è nata e quali applicazioni ha nella Computer Vision.
Effettuerò delle prove sperimentali che andranno a valutare le performance di questo approccio rispetto a un tradizionale approccio convoluzionale, a parità di dataset e iperparametri di training.

INDICE

Lista delle Figure

Capitolo 1

Introduzione

In un algoritmo tradizionale l'approccio open box richiede che il programmatore specifichi il comportamento del software in qualsiasi condizione.
In molti casi la conoscenza completa di un problema non è nota a priori. Basta pensare all'algoritmo di ranking di Google, è impensabile definire come ordinare i risultati di una ricerca nel web, le variabili in gioco sarebbero troppe e in costante cambiamento. Anche se esistesse una soluzione tradizionale, questa non sarebbe scalabile.
Il Machine Learning fornisce un diverso tipo di approccio in grado di risolvere problemi di questo tipo.

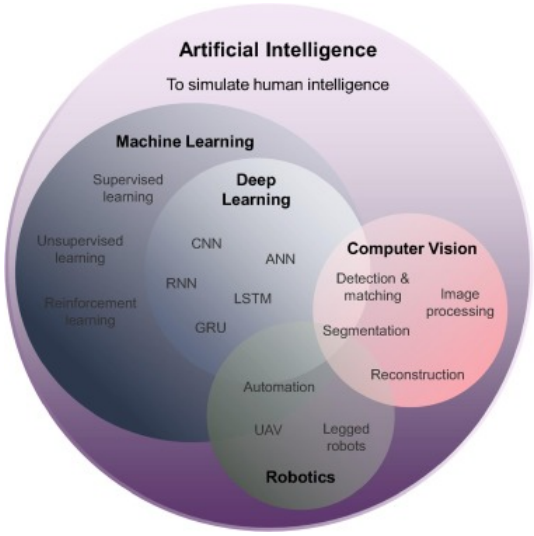

Figura 1.1: ML appartiene al mondo dell'AI, questo studio si concentrerà su una parte di questo chiamato Computer Vision.

Gli algoritmi di Machine Learning si discostano dai classici algoritmi, sono in-

fatti programmati per "imparare dall'esperienza", ossia trarre informazioni da una grande quantità di dati in modo da costruire un modello del problema.
Questi algoritmi forniscono un approccio black box che permette di ottenere una soluzione generale ad un problema.

1.1 Deep Learning e Computer Vision

Esistono numerose tecniche di Machine Learning, questo studio si concentrerà sulle Reti Neurali.
Una Rete Neurale Artificiale è un modello matematico che prende ispirazioni dalla rete neurale biologica. Ogni neurone (chiamato nodo) è una funzione matematica a cui viene assegnato un certo peso. I nodi sono collegati tra loro e vengono attivati (la funzione matematica è applicata al dato) solo se viene superato un certo valore di soglia predefinito.
I nodi vengono raggruppati in layer.

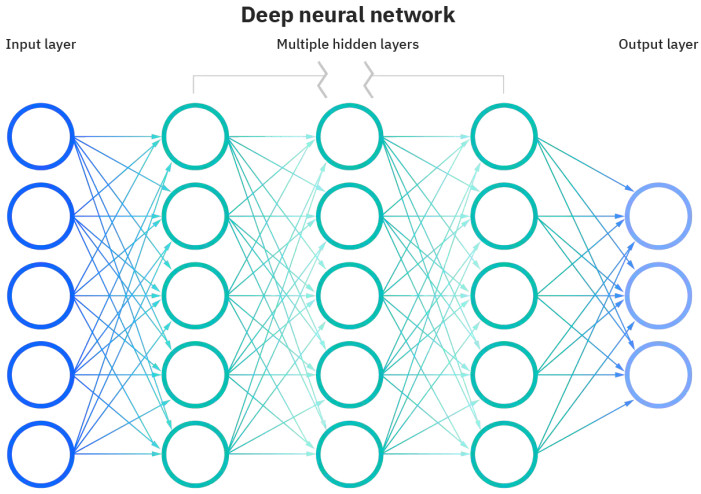

Figura 1.2: I layer possono essere di 3 tipologie.

La funzione di attivazione introduce delle non-linearità al modello. Se per esempio vogliamo costruire un modello matematico che approssimi l'andamento del PM10

nell'aria, una rete neurale senza funzioni d'attivazioni è un semplice modello di regressione che approssima l'andamento con una retta. Questo è un modello impreciso, aggiungendo delle funzioni di attivazione l'approssimazione diventa più precisa e non lineare.

L'approccio con Rete Neurale è una tecnica sofisticata, sia in termini di quantità necessaria di dati che di potenza computazionale, e rientra in un sottocampo del Machine Learning chiamato Deep Learning (vedi figura 1.1).

Deep indica infatti l'alto numero di layer di una Rete Neurale. Immaginiamo un problema di classificazione, in cui il software deve etichettare un oggetto in base alla classe di appartenenza (riconoscere in un segnale stradale il limite di velocità da rispettare).

L'associazione corretta alla classe di appartenenza viene fatta attraverso l'utilizzo di features, una particolare caratteristica dell'oggetto che permette di identificarlo correttamente (in un'immagine di un elefante la proboscide è una feature importante).

Un approccio tradizionale di ML richiede che il programmatore in fase di training specifichi le caratteristiche identificative dell'oggetto (le features appunto). Nel Deep Learning invece l'estrazione delle features avviene automaticamente durante il training.

Questo particolare tipo di problema appartiene al mondo della Computer Vision, materia oggetto di questa tesi.

Grazie alla Computer Vision è possibile riconoscere, classificare ed estrarre oggetti da immagini o video.

1.2 Natural Language Processing

Il Natural Language Processing (NLP) è una ramo dell'AI che si occupa di comprendere ed interpretare il linguaggio umano. Fanno parte di NLP tutti gli algoritmi di traduzione, riconoscimento del linguaggio e correzione automatica.
Grazie all'NLP è possibile realizzare sistemi di assistenza vocale come Siri o Alexa, analizzare documenti o post su Instagram (il cosidetto Sentiment Analysis).
Il metodo dell'Attention ha trovato le prime applicazioni proprio in questo ramo dell'AI e basa il suo funzionamento sull'assegnare pesi diversi agli elementi di una sequenza di input. Così facendo il modello focalizza l'attenzione sui dati ritenuti più importanti.
Nel capitolo successivo vedremo nel dettaglio in cosa consiste questa tecnica e che varianti esistono. Prima è però opportuno presentare il motivo per cui è nato il metodo dell'Attention, parlando dello Stato dell'Arte riguardante le prime tecniche di Deep Learning usate nell'NLP.
Dopo aver dato una spiegazione teorica esaustiva si passerà alla parte sperimentale di questa tesi, presentando le tecnologie utilizzate e i modelli scelti.

Capitolo 2

Stato dell'arte

In questo capitolo verranno spiegati i bisogni che hanno portato alla nascita della tecnica dell'Attention e successivamente alla creazione del modello Transformer. Sarà presentata l'evoluzione dello Stato dell'Arte nel mondo del Language Translation e successivamente nel mondo della Visione Artificiale.
A conclusione di questa sezione verranno elencati i dettagli dei modelli di Rete Neurale scelti nelle prove sperimentali.

2.1 RNN e Seq2Seq

Le Recurrent Neural Network sono un tipo di Rete Neurale che lavora con dati sequenziali o serie temporali. La loro caratteristica è quella di avere dei layers in cui i neuroni prendono come input i loro stessi output (recurrent neuron vs feedforward neuron delle CNN). Questo rende le RNN ottime per fare previsione.
Altra loro caratteristica è quella di accettare come input vettori di lunghezza arbitraria (a differenza delle CNN).
RNN è stata la prima Rete Neurale ad essere usata per l'NLP, e nel tempo ha dato vita a diverse varianti, la più comune è stata l'architettura Long Short-Term Memory (LSTM, Hochreiter and Schmidhuber (1997)).
Infatti le RNN avevano come limite quello di non processare correttamente parole riferite allo stesso contesto ma distanti tra loro nella frase.
Una diversa applicazione delle RNN in questo campo è stata proposta nei paper Cho et al. (2014) e Sutskever et al. (2014).

Questo modello prevede l'utilizzo di due RNN in sequenza, la prima chiamata Encoder, la seconda Decoder. Per questo motivo viene anche chiamato modello Sequence-to-Sequence (Seq2Seq).

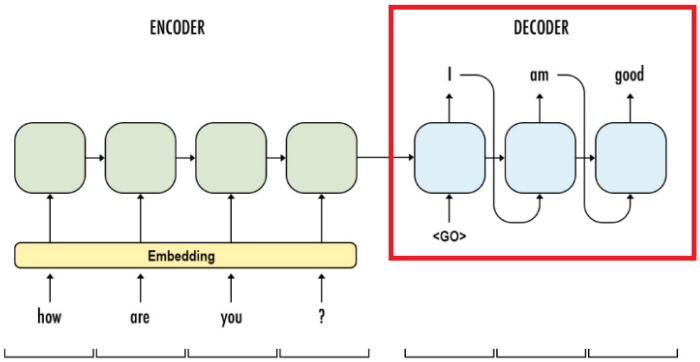

Figura 2.1: Rappresentazione del modello Seq2Seq

Prendendo in esempio l'applicazione di questo modello in un problema di Neural Machine Translation si osserva come ogni parola della frase viene passata alla RNN dell'Encoder. Ad ogni iterazione oltre alla parola la RNN prende in input un vettore, che altro non è che l'output del precedente processamento.

Il risultato della fase di Encoder è un vettore chiamato Context che diventa l'input della RNN di Decoder. Questa, dopo aver fatto un lavoro analogo all'Encoder, manda in output la traduzione della frase.

Il passaggio del Context da Encoder a Decoder (un vettore di numeri che rappresenta la codifica dell'input) rappresenta un collo di bottiglia per lunghe sequenze, questo viene risolto applicando la tecnica dell'Attention.

2.2 Seq2Seq con Attention

La soluzione proposta nei paper Bahdanau et al. (2014) e Luong et al. (2015) introduce nell'architettura Seq2Seq vista precedentemente la tecnica dell'Attention. Questo permette al modello di focalizzarsi sulle parti rilevanti della stringa di input prima di effettuare la traduzione.
Le principali differenze con il Seq2Seq tradizionale sono:

- Tutti gli hidden state generati dall'encoder sono passati al decoder (non solo l'ultimo).
- Il modello è in grado di dare più peso a certe parole rispetto che ad altre, il decoder assegna degli scores agli hidden state ricevuti.

Questa tecnica permette di parallelizzare il passaggio degli hidden state tra encoder e decoder, ottenendo quindi un significativo aumento delle performance generali. Il passaggio solamente dell'ultimo hidden state al decoder va bene quando il vettore in input è di ridotta dimensione. Nel caso opposto il modello perderebbe in termini di prestazioni.

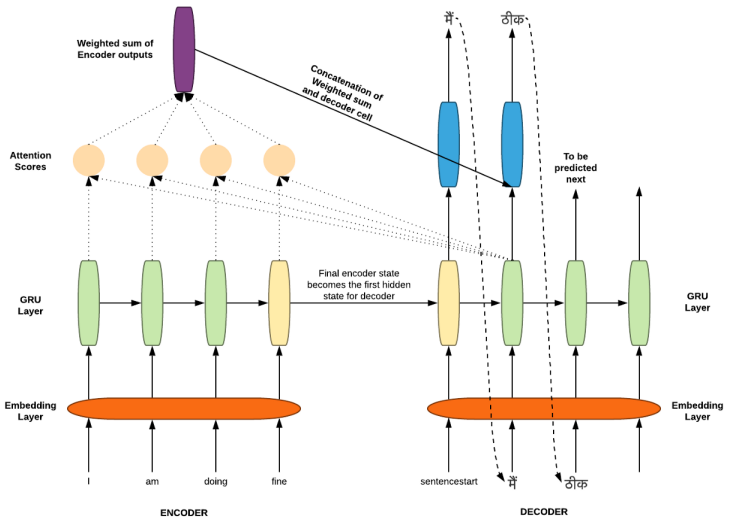

Figura 2.2: Per ogni step del Decoder tutti gli hidden state dell'Encoder vengono sommati e passati alla RNN del Decoder. Questo vettore verrà poi concatenato al Decoder hidden state ed elaborato nuovamente.

Nei paper sopra citati viene dimostrato il miglioramento delle performance ottenuto a seguito dell'introduzione dell'Attention nel modello Encoder-Decoder. Negli ultimi anni è però nato un nuovo e più veloce modello sulla base del paper Vaswani et al. (2017) .

2.3 Self-Attention nel modello Transformer

L'utilizzo di Attention combinato con le RNN fu largamente usata nei problemi di NLP fino al 2018. In questo anno dei ricercatori di Google pubblicarono uno studio (Vaswani et al. (2017)) in cui dimostrarono che per codificare una sequenza di caratteri era sufficiente solamente la tecnica dell'Attention.

Non sono più necessarie RNN e una codifica sequenziale, una corretta interpretazione di una stringa di input può essere fatta utilizzando solo l'Attention. Nel paper di Google venne quindi proposta una modifica di questa tecnica e prese il nome Self-Attention Mechanism, usato poi nel modello Transformer presentato nello stesso articolo.

Mantiene la struttura encoder/decoder, ognuno formato da 6 layer (i layer non compongono una RNN). Il Self-Attention Mechanism è usato nel layer omonimo. La figura 2.3 rappresenta i layer del modello Transformer.

La velocità di questo modello è garantita dal fatto che più parole possono essere elaborate contemporaneamente all'interno dei layer (nel modello Encoder/Decoder era elaborata una parola alla volta).Le principali architetture che fanno uso del modello Trasformer sono BERT e GPT.

La stringa in input all'Encoder (parte sinistra di figura 2.3) attraversa il layer Multi-Head Attention, che non è altro che una sequenza di layer Attention. Successivamente attraversa un layer Feed Forward.

Questa struttura è identica anche nel Decoder (parte destra di figura 2.3), con l'unica differenza che fra questi layer ve ne è un terzo, usato per porre il focus sulla

parte rilevante della frase (come nel modello Seq2Seq con Attention).

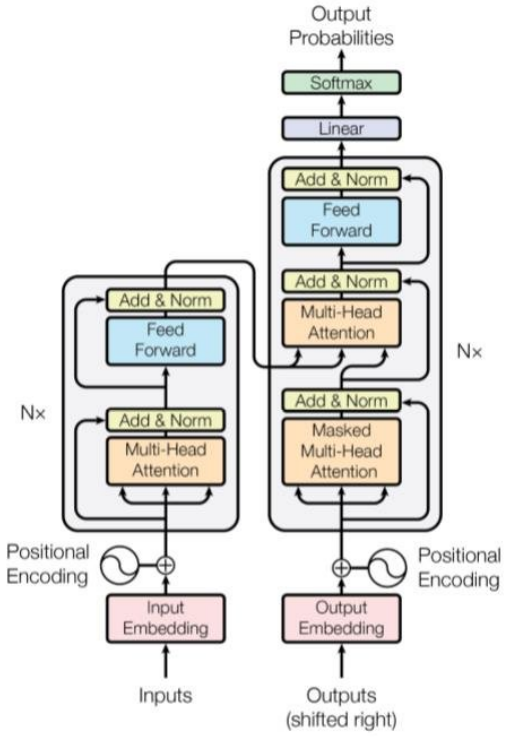

Figura 2.3: Il Transformer Model presentato nel paper.

Approfondiamo il cuore del modello Transformer, il layer Multi-Head Attention:

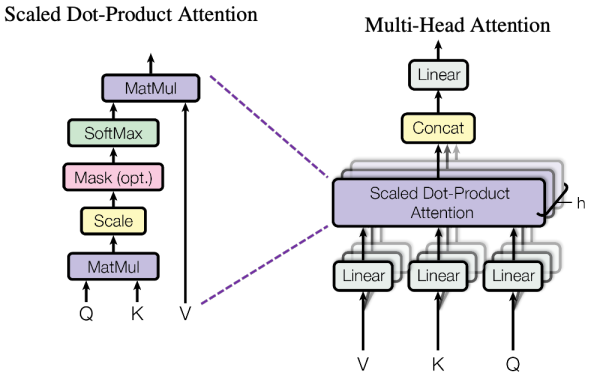

Figura 2.4: Layer Multi-Head Attention.

In figura 2.4 osserviamo che l'input del layer sono 3 valori: V, K e Q. Di seguito le operazioni fatte all'interno del layer:

- Ogni input (parola della frase) ha 3 rappresentazioni:Key, Value e Query. Queste sono ottenute moltiplicando gli input per dei pesi presi da una distribuzione Gaussiana.
- Moltiplichiamo la Query di ogni input per la matrice di tutte le Key, otteniamo così l'attention score.
- Applichiamo la funzione Softmax agli n scores ottenuti (dati n input).
- Moltiplichiamo i nuovi attention scores con il rispettivo Value (per ognuno degli input).
- Il risultato ottenuto è l'output di questo layer.

2.4 Vision Transformer

Il recente paper (Dosovitskiy et al., 2010) mostra come sia possibile applicare il modello Transformer all'Image Recognition.
L'immagine in input viene suddivisa in patches passate poi alla rete. Per mantenere l'informazione posizionale ad ogni patch è associato un vettore numerico indicante la sua posizione nell'immagine generale. Queste sono utilizzate come le parole di una frase (come nel problema di NLP visto al paragrafo precedente).

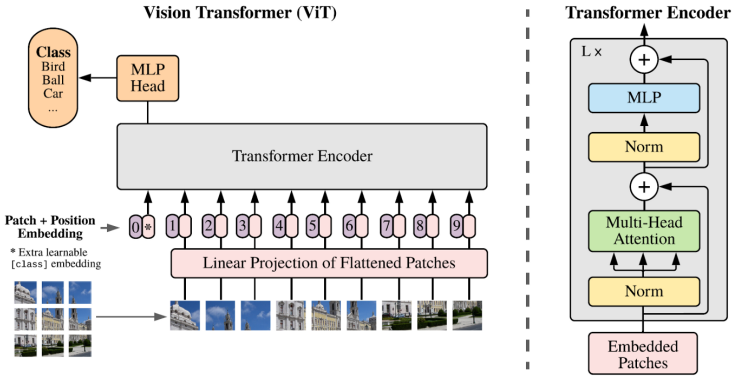

Figura 2.5: Il modello Vision Transformer.

Il blocco di Encoder è identico a quello del Transformer originale presentato nel paragrafo precedente (quello del paper Vaswani et al. (2017)).
Sempre nello stesso studio sono state presentate tre varianti, diverse nel numero di layer e parametri di training:

Model	Layers	Hidden Size D	MLP Size	Heads	Params
ViT - Base	12	768	3072	12	86M
ViT - Large	24	1024	4096	16	307M
ViT - Huge	32	1280	5120	16	632M

La colonna Heads si riferisce al numero di blocchi di Encoder della rete, mentre MLP size alla dimensione del vettore che passa nel blocco multi-layer perceptron (rete neurale in cui il mapping tra input e output non è lineare). Hidden size D indi-

ca invece la dimensione della patch, o più precisamente la dimensione della stringa identificativa della patch.

Questi modelli sono disponibili in tutti i principali framework di Machine Learning. In questo studio si è scelto di utilizzare Pytorch, una framework python sviluppato da Meta che fornisce modelli pre-addestrati e funzioni utili nella Computer Vision e nell'NLP.

Il modello scelto di Vision Transformer è instanziato usando il pacchetto torchvision.models. Il modello della rete sarà poi già pre-addestrato con dei pesi predefiniti (eventualmente modificabili).

Pytorch mette a disposizioni diversi costruttori di modelli in base alla dimensione della patch in input alla rete (se la patch size è 16x16, si utilizzerà vit_b_16 o vit_l_16).

2.5 Modelli utilizzati

Il pacchetto torchvision offre numerosi modelli pre-addestrati per diversi utilizzi, come ad esempio la classificazione di immagini.
I modelli disponibili sono pre-addestrati con specifici pesi, settabili attraverso una API della libreria models.
A parità di dataset e iperparametri la documentazione ufficiale di Pytorch fornisce 2 differenti valori di accuracy:

- **Acc@1**: data la probabilità di appartenenza ad ogni classe dell'immagine di test, la classificazione viene considerata corretta se la label corrisponde a quella a più alta priorità.
- **Acc@5**: data la probabilità di appartenenza ad ogni classe dell'immagine di test, la classificazione viene considerata corretta se la label corrisponde ad almeno una fra le 5 a più alta priorità.

In questo studio è stato scelto di settare le reti pre-addestrate ai valori di default, ossia quelli ottenuti effettuando il training sul dataset ImageNet.
Durante il training di una rete infatti ad ogni ingresso di ciascun nodo viene assegnato un certo peso. Utilizzare reti già addestrate significa ottenere un modello in cui il training è già stato fatto su un dataset generale, e quindi i nodi hanno già dei pesi settati. Questo modello pre-addestrato viene poi adattato al nostro dataset modificando i layer finali della rete, che sono quelli di classificazione. I layer pre-addestrati vengono invece mantenuti con i pesi già precedentemente calcolati. Questi layer possono avere qualsiasi funzione di attivazione, nel caso in cui si inseriscano solo layer lineari invece si ricade nel caso di Regressione Lineare.
Questa tecnica è chiamata **Transfer Learning** e tra gli enormi vantaggi ha quello di evitare di spendere tempo e risorse computazionali nel training completo della rete.
La scelta sui layer da inserire nella testa è abbastanza libera, la letteratura scientifica

non dà precise indicazioni. Pratica comune nei modelli di classificazione è utilizzare in coppia layer di Dropout con una funzione di attivazione. In questo studio si è deciso di utilizzare come funzione di attivazione la ReLU, definita come la parte positiva dei suoi argomenti (se negativi restituisce zero):

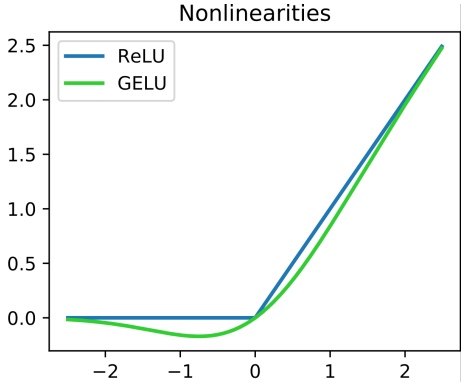

Figura 2.6: La funzione ReLU (rectified linear unit) a confronto con GELU.

Oltre a questi sono stati inseriti layer Lineari in modo da ridurre il numero di feature in output, così da poter effettuare la classificazione sul numero di classi corretto (per maggiori dettagli vedere la figura 2.7 e l'Appendice).

2.5.1 Vision Transformer Base

Il modello segue l'architettura a layer della figura 2.5: il primo layer applica una convoluzione alla patch di input riducendo le dimensioni a 16x16.
Vi sono alcuni layer Dropout usati per settare a zero casualmente alcuni elementi di input. La probabilità di questo azzeramento è messa a zero come riportato nel paper Dosovitskiy et al. (2010).
Come mostrato nella tabella 2.4 il modello contiene 12 layer di Encoder, ognuno con la seguente struttura:

- **Conv2d**: il primo layer della rete applica una convoluzone bi-dimensionale al dato di input. Il metodo Pytorch per questo layer richiede come paramentri il numero dei canali in input e in output (3 e 768), il valore del kernel (16x16) e dello stride (16x16) della convoluzione da applicare.

- **LayerNorm**: applica una normalizzazione al batch in input (lo setta a 768, come scritto nella colonna Hidden size di 2.4).
- **MultiheadAttention**: applica la tecnica di Self Attention come descritta nel paper sopra citato .
- **Dropout**: setta casualmente a zero con probabilità p alcuni elementi del tensore di input.
- **LayerNorm**: secondo layer di normalizzazione del blocco encoder.
- **MLP Block**: implementa il Multi-Layer-Perceptron (un insieme di layer fully connected). Questo blocco è composto da:
  - Linear: applica una trasformazione lineare al tensore in ingresso modificando il numero di features da 768 a 3072.
  - GELU: applica l'Errore Lineare Gaussiano al dato (vedere figura 2.6).
  - Dropout: la probabilità di azzeramento è zero.
  - Linear: uguale al precedente layer omonimo solo che le features sono modificate da 3072 a 768.
  - Dropout: uguale al precedente layer omonimo.

Lo schema generale dei layer della rete è il seguente:

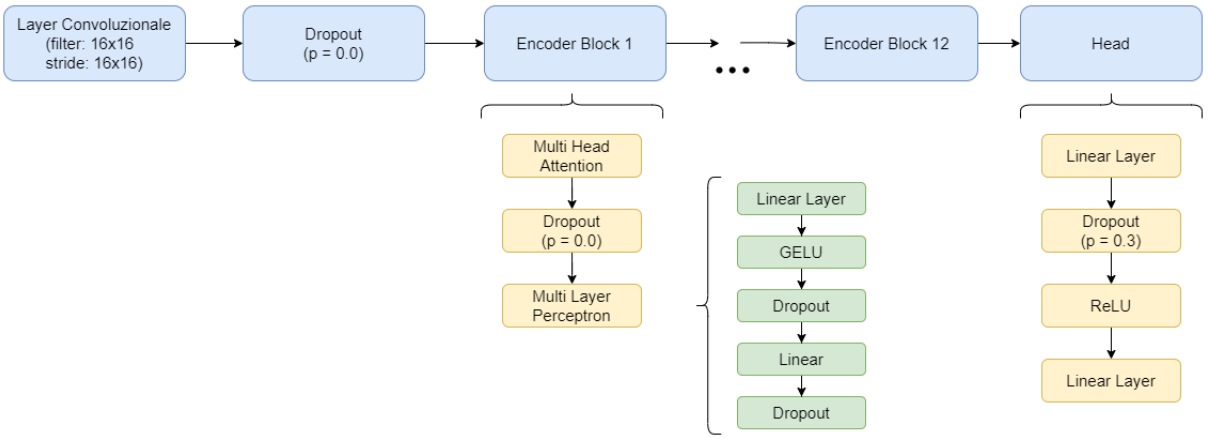

Figura 2.7: Schema della rete Vision Transformer.

Per quanto riguarda la Head della rete sono state effettuate prove con i quattro layer sopra citati, schematizzati in figura 2.7.

Il modello scelto pre-addestrato con ImageNet ha le seguenti caratteristiche :

Acc@1	70.662 %
Acc@5	95.318 %
Numero di Parametri	86 567 656
Input Size	3 x 224 x 224

I precedenti valori sono presi dalla documentazione di Pytorch.

2.5.2 Vision Transformer Large

La struttura generale di questo modello rimane coerente con l'immagine 2.5 mostrata nel paragrafo precedente.
Come per il modello Vit_b_16 l'immagine di input è suddivisa in patch di dimensione 16x16 nel layer convoluzionale. Successivamente si susseguono in sequenza 24 layer di Encoder, in cui ognuno ha la seguente struttura:

- **Conv2d**: i parametri per questo modello sono in_channel = 3, out_channel = 768, kernel_size e stride di 16x16.
- **LayerNorm**: stavolta la normalizzazione è fatta a 1024 come in scritto in tabella 2.4.
- **MultiheadAttention**
- **Dropout**
- **LayerNorm**
- **MLP Block**: questo blocco è composto da:
  - Linear: la trasformazione lineare modifica il tensore da 1024 a 4096.
  - GELU
  - Dropout
  - Linear: modifica da 4096 a 1024.
  - Dropout

Le principali differenze con il precedente modello Transformer(come da tabella 2.4) sono:

- Un maggiore numero di layer di Encoder (24 invece di 12).
- La stringa identificativa della patch (contente anche l'informazione posizionale della singola patch, che da output del layer N-1 diventa input del layer N) è di 1024 invece che 768.

- Il numero di parametri di training aumenta enormemente (da 86M a 307M).
- Ogni blocco Multi-Layer-Perceptron (MLP Block) elabora vettori di dimensioni 4096 (e non 3072 come in Vit_B).

Essendo lo schema della rete molto simile a ViT - Base si faccia riferimento alla figura 2.7).

Il modello pre-addestrato scelto è il seguente:

Acc@1	79.662 %
Acc@5	94.638 %
Numero di Parametri	304 326 632
Input Size	3 x 224 x 224

I precedenti valori sono presi dalla documentazione di Pytorch.

2.5.3 ResNet50

ResNet (Residual Network) è una innovativa rete neurale a convoluzione introdotta nel 2015 nel seguente paper He et al. (2016). La sua diffusione nel mondo della Computer Vision è anche dovuta al fatto che ResNet vinse la challenge di ImageNet nel 2015 (ILSVRC2015).
Vi sono diverse varianti di ResNet originale, quella scelta in questo lavoro è Resnet50.

Le principali caratteristiche sono:

- Il modello è composto da 50 layers.
- La rete ha 25M di parametri.
- Il blocco principale è chiamato bottleneck e viene ripetuto diverse volte all'interno del modello
- Ogni bottleneck block possiede 3 layer convoluzionali (in coppia con una funzione di normalizzazione) e una funzione ReLu.

Di seguito vengono presentati i layer del modello ResNet50:

- **Conv2d**: i parametri per questo modello sono in_channel = 3, out_channel = 64, kernel_size = 7x7, stride = 2x2 e padding = 3x3.
- **BatchNorm2d**: applica una normalizzazione al batch in input (lo setta a 64).
- **ReLU**: applica una funzione ReLU al dato in input.
- **MaxPool2d**: calcola il valore massimo di ogni patch in input.
- **Bottleneck**: è un blocco di layer che contiente 3 coppie di layer convoluzionale e di normalizzazione. Il blocco termina con un layer ReLU. Ci sono 16 blocchi di questo tipo nel modello.

Lo schema generale dei layer della rete è il seguente:

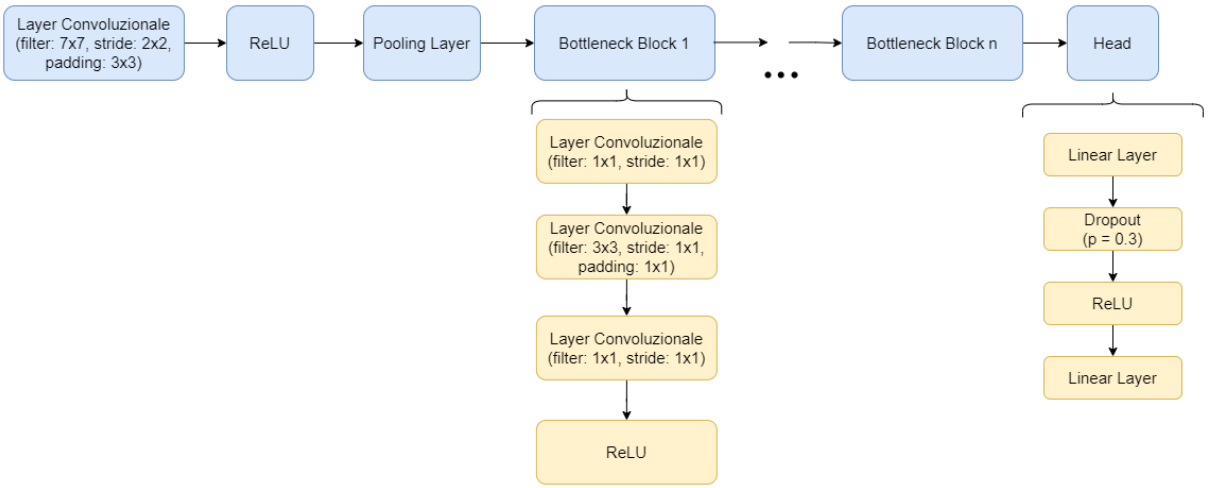

Figura 2.8: Schema della rete ResNet50.

I layer per la classificazione inseriti nella testa del modello sono identici a quelli dei modelli ViT, sono schematizzati in figura 2.8.
Il modello pre-addestrato scelto è il seguente:

Acc@1	76.13 %
Acc@5	92.862 %
Numero di Parametri	25 557 032
Input Size	3 x 224 x 224

I precedenti valori sono presi dalla documentazione di Pytorch.

Capitolo 3

Prove Sperimentali

Lo scopo di questo progetto di tesi è mostrare la superiorità in termini prestazionali dei modelli Vision Trasformer rispetto ai tradizionali modelli convoluzionali.
Per fare questo sono stati fatti diversi notebook in Colab al fine di valutare Accuracy e Loss di alcune reti neurali a parità di iperparametri e dataset.
Le varie reti sono state utilizzate per fare Object Classification su dataset di dimensione crescente sia in termini di immagini che di classi.
Anche nel paper Dosovitskiy et al. (2010) di Vision Transformer è stato effettuato uno studio comparativo con le CNN. Questo ha dimostrato come la scelta del dataset di training incide molto sulle prestazioni del modello. Di seguito l'immagine che mostra l'andamento dell'accuracy in relazioni a dataset di dimensioni crescenti:

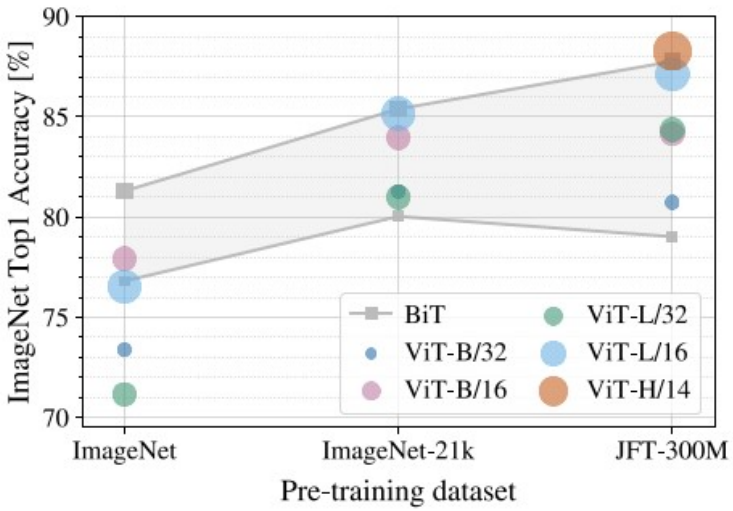

Figura 3.1: Andamento di Accuracy in relazione al dataset (BiT è un modello convoluzionale, paper Dosovitskiy et al. (2010)).

3.1 Dataset

Tutti i dataset sono stati presi da Kaggle, una piattaforma open source con una sezione interamente dedicata ai dataset per Computer Vision.
Di seguito una breve presentazione dei dati utilizzati e delle loro caratteristiche.
Ogni immagine è stata elaborata per essere un valido input della rete. Il lavoro di preparazione del dataset consiste nel trasformare ogni immagine in un tensore, normalizzarlo ed effettuare un resize di 224x224. Ogni tensore ha dimensione (3,224,224), dove tre indica che l'immagine è a colori. Questo è ciò che è passato come input della rete.
Per caricare correttamente il dataset è necessario che le immagini siano correttamente suddivise in cartelle, ognuna delle quali rappresenta la label (classe di appartenenza) dell'immagine.
In questa tesi è stato scelto di suddividere il dataset in 2 parti:

- Training Dataset: immagini utilizzate durante l'addestramento del modello.
- Testing Dataset: immagini utilizzate per valutare il modello (è un dataset completamente sconosciuto, non ancora passato al modello).

3.1.1 Corn or Maize Leaf Disease Dataset

Di seguito le 4 classi di immagini all'interno del dataset:

- Common Rust - 1306 immagini.
- Gray Leaf Spot - 574 immagini.
- Blight -1146 immagini.
- Healthy - 1162 immagini.

Il dataset è stato suddiviso in Training Dataset (80% delle immagini) e Testing Dataset (20% delle immagini).

- **Train dataset**: 3350
- **Test dataset**: 838

Ecco alcune immagini di esempio:

Figura 3.2: Alcune immagini casuali di Corn or Maize Leaf Disease Dataset.

Questo è il dataset di dimensioni minori, oltre ad avere poche immagini ha anche solamente 4 classi. Queste caratteristiche saranno molto incisive durante il training. Corn or Maize Leaf Disease Dataset risulta essere bilanciato per tre classi su quattro, ossia la maggior parte delle classi sono composte da un numero di immagini simile rispetto all'intero dataset di training:

Nome della classe	**Percentuale**
Blight	27.97
Common_Rust	30.66
Gray_Leaf_Spot	13.4
Healthy	27.97

3.1.2 Pet Dataset

Questo dataset contiene 37 classi di razze animali e per ogni classe sono presenti circa 200 immagini.

Anche in questo caso la suddivisione del dataset in training e testing ha seguito la regola 80% - 20%.

- **Train dataset**: 5912
- **Test dataset**: 1478

Ecco alcune immagini di esempio:

Figura 3.3: Alcune immagini casuali di Pet Dataset.

Questo dataset non aveva la suddivisione delle immagini nelle cartelle delle label. Questa informazione era infatti contenuta nella stringa del nome.

Il lavoro di pre-processing è stato quindi quello di estrarre i nomi delle label dai nomi delle immagini, creare le cartelle in modo coerente con le label e inserire ogni immagine nella cartella corretta.

Questo dataset è bilanciato, ossia il numero di immagini per ogni classe è all'incirca

simile. Di seguito è mostrata la tabella contenente le percentuali di immagini in una classe rispetto all'intero dataset di training:

Nome della classe	**Percentuale**
great_pyrenees	2.64
scottish_terrier	2.59
pomeranian	2.52
basset_hound	2.6
german_shorthaired	2.71
miniature_pinscher	2.76
shiba_inu	2.71
english_setter	2.81
american_pit_bull_terrier	2.6
chihuahua	2.74
boxer	2.62
Russian_Blue	2.55
Siamese	2.77
Abyssinian	2.62
Bengal	2.84
Egyptian_Mau	2.52
Ragdoll	2.69
Sphynx	2.89
pug	2.66
Bombay	2.77
samoyed	2.98
Maine_Coon	2.72
British_Shorthair	2.84
saint_bernard	2.59
yorkshire_terrier	2.71
havanese	2.69
american_bulldog	2.66
japanese_chin	2.64
newfoundland	2.69
keeshond	2.66
Birman	2.64
beagle	2.86
wheaten_terrier	2.67
staffordshire_bull_terrier	2.66
english_cocker_spaniel	2.89
Persian	2.82
leonberger	2.67

3.1.3 Flower Dataset

Il dataset raggruppa 102 categorie di fiori e ogni classe ha tra le 40 e le 258 immagini, questo è il dataset più numeroso in termini di classi. Il tutto è già suddiviso in Training e Testing Dataset secondo le percentuali 90% - 10%.
Per effettuare un confronto prestazionale coerente tra i dataset le percentuali delle immagini di training e testing di Flower Dataset sono state sistemate. Per fare questo sono state spostate un numero sufficiente di immagini dal Training Dataset al Testing Dataset in modo da ottenere una suddivisione di 80% - 20%. Sono quindi state scelte casualmente 650 immagini dalle cartelle di train e spostate nelle omonime cartelle di testing (il nome della cartella di destinazione deve essere quello della label corretta).

- **Train dataset**: 5902
- **Test dataset**: 1468

Ecco alcune immagini di esempio:

Figura 3.4: Alcune immagini casuali di Flower Dataset.

Anche questo dataset, come i precedenti, risulta essere bilanciato. Per le percentuali di ogni classe vedere l'Appendice (tabella 4.2).

3.2 Scelta degli iperparametri

La scelta degli iperparametri durante il training è fortemente collegato alle prestazioni che il modello avrà.
Si è quindi scelto di confrontare i diversi modelli di Vision Trasformer e CNN anche al variare di iperparametri oltre che di dataset.
Questi sono i valori che maggiormente incidono sulle performance del modello:

- Scelta dell'ottimizzatore e dei relativi Learning Rate e Momentum.
- Numero di Thread usati per caricare il dataset.
- Dimensione del Batch Size.
- Numero di Epoche.

L'ottimizzatore viene utilizzato durante il training per settare al meglio i pesi della Rete Neurale, ossia in modo tale che si aumenti l'accuracy generale della rete. Per scegliere i pesi ottimali l'ottimizzatore utilizza la Loss Function.
Pytorch ne mette a disposizione diversi, ma in base a quanto fatto nel paper Dosovitskiy et al. (2010) di Vision Transformer (nella parte relativa al fine-tuning della rete) si è scelto di utilizzare SGD (Stochastic Gradient Descent Optimizer).

Il **Gradient Descendent** è un algoritmo di ottimizzazione in grado di trovare il punto minimo di una funzione di costo. I pesi sono inizialmente settati ad un valore casuale e saranno aggiornati ad ogni iterazione in modo da raggiungere il minimo della Cost Function. Il gradiente nullo indica che si è raggiunto il minimo di questa funzione. I pesi sono aggiornati in base al valore del gradiente (aggiornato ad ogni

iterazione) e del Learning Rate (si mantiene costante).
Il Learning rate indica di quanto dovranno essere aggiornati i pesi della rete ad ogni iterazione (vedi formula 3.1). Ha un valore costante che può variare tra 0 e 1, più piccolo è e minore è la ripartizione dell'errore sul calcolo dei nuovi pesi della rete. Solitamente il valore di default è 0.001 (scelto infatti per questo studio sperimentale). Il Momentum è invece un parametro opzionale che aiuta l'ottimizzatore SGD ad accelerare il tempo di convergenza al valore minimo della funzione di costo, un buon valore usato spesso per il Momentum è 0.9 (usato in questo studio).
La formula è la seguente:

$$\boldsymbol{\theta} = \boldsymbol{\theta} - \boldsymbol{\eta} \cdot \nabla_{\boldsymbol{\theta}} J\left(\boldsymbol{\theta}; \boldsymbol{x}^{(i)}; \boldsymbol{y}^{(i)}\right) \tag{3.1}$$

Il Learning Rate $\boldsymbol{\eta}$ è moltiplicato per il gradiente della funzione di costo $J\left(\boldsymbol{\theta}; \boldsymbol{x}^{(i)}; \boldsymbol{y}^{(i)}\right)$ (J è la Cost Function, $\boldsymbol{\theta}$ è il parametro del modello che minimizza la funzione di costo e $\boldsymbol{x}^{(i)}$ e $\boldsymbol{y}^{(i)}$ indicano rispettivamente l'i-esimo dato in input e la sua label).

Altri parametri sono settabili durante la definizione del DataLoader. Questa classe permette di creare un oggetto iterabile del dataset, in modo da poterlo usare in modo semplice all'interno dello script. In questa fase viene definito il Batch Size e il numero di Thread usati.
Il Batch Size indica il numero di parti in cui viene suddiviso il dataset originale. Questi sotto-dataset vengono passati uno alla volta alla rete, è facile capire come questo parametro influenzi molto le prestazioni della Rete.
Il loading del dataset può essere fatto da un unico processo o da più sotto-processi (chiamati appunto Thread). Uno dei parametri del DataLoader è proprio il numero di questi Thread. In tutti i test fatti il numero di sotto-processi è stato mantenuto costante a 4.
Il numero di epoche corrisponde invece al numero di volte in cui l'intero dataset (tutti i Batch del dataset) sono passati alla rete.

3.3 Prestazioni dei modelli

Ognuno dei modelli appena descritti ha lavorato con dataset molto diversi fra loro, sia in termini di dimensioni che in termini di numero di classi. Questo ha influenzato i risultati prestazionali delle 3 reti.

Nella tabella a seguire vengono presentati i valori di Accuracy ottenuti dai vari modelli:

VIT_B_16	**Dataset**	**Train Accuracy**	**Test Accuracy**
Batch Size = 8 **Num Epochs** = 25	Leaf Diseases	98.8%	98.9%
	Pet	92.8%	92.0%
	Flower	93.3%	93.4%
Batch Size = 20 **Num Epochs** = 25	Leaf Diseases	97.1%	96.9%
	Pet	90.8%	90.9%
	Flower	88.7%	89.3%

VIT_L_16	**Dataset**	**Train Accuracy**	**Test Accuracy**
Batch Size = 8 **Num Epochs** = 25	Leaf Diseases	99.6%	99.9%
	Pet	93.0%	93.5%
	Flower	95.9%	96.0%
Batch Size = 20 **Num Epochs** = 25	Leaf Diseases	98.9%	99.2%
	Pet	90.5%	92.4%
	Flower	92.2%	92.5%

RESNET50	**Dataset**	**Train Accuracy**	**Test Accuracy**
Batch Size = 8 **Num Epochs** = 25	Leaf Diseases	91.9%	89.9%
	Pet	82.7%	83.4%
	Flower	80.0%	81.4%
Batch Size = 20 **Num Epochs** = 25	Leaf Diseases	93.0%	93.2%
	Pet	77.1%	77.3%
	Flower	82.5%	81.7%

3.3.1 Prestazioni di Leaf Dataset

Passiamo ora ad un'analisi più dettagliata dell'andamento delle performance durante il training. Per quanto riguarda il dataset delle malattie sulle piante di mais i risultati di Accuracy ottenuti con i dati di training sono stati i seguenti:

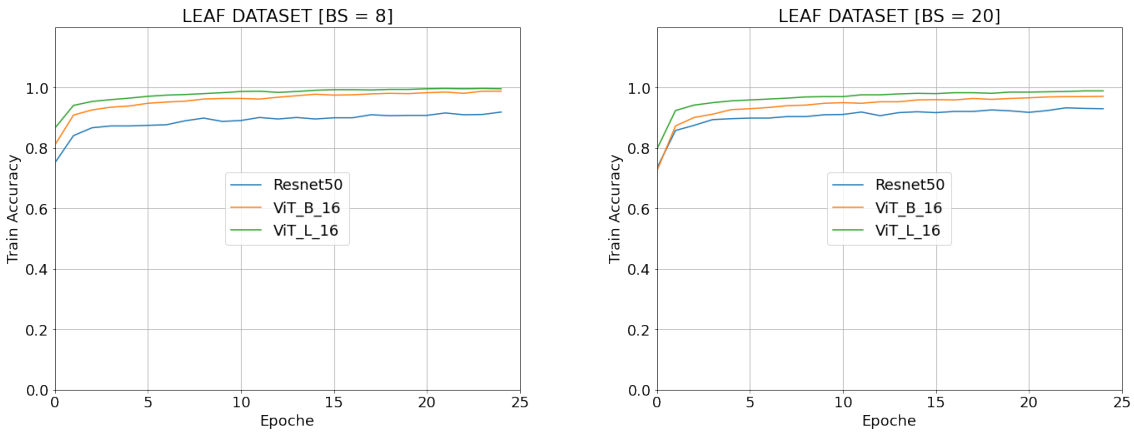

Figura 3.5: Train Accuracy di Leaf Dataset.

L'andamento di Accuracy è molto simile per tutte e tre le reti, per arrivare a regime tutte e tre le reti impiegano poche epoche. Per quanto riguarda il training con Batch Size di 8 ViT - Large risulta il più accurato fin dalla prima epoca. Aumentando il Batch Size ResNet50 arriva ad un valore di Accuracy più simile alle altre 2 reti. Fin da subito è visibile come i modelli Vision Transformer siano migliori del modello convoluzionale ResNet. Per quanto riguarda la Loss i grafici sono i seguenti:

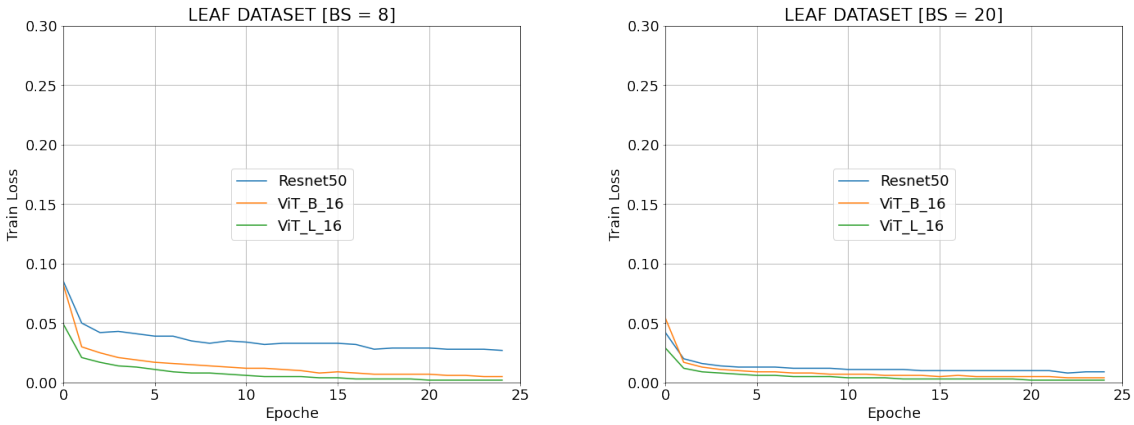

Figura 3.6: Train Loss di Leaf Dataset.

I grafici mostrati sono riferiti ai dati di training, quelli di test non mostrano alcuna differenza evidente.

L'andamento della Loss (figura 3.6) per ResNet50 ha un andamento costantemente peggiore rispetto alle altre due reti Transformer quando il Batch Size è settato a 8. Questa caratteristica viene meno quando questo parametro ha valore 20, infatti per molte epoche ResNet50 mantiene un valore di Loss molto simile ai modelli ViT.

Un'altra fattore che si vede nei grafici è che l'aumentare del Batch Size porta a una diminuzione del valore di inizio di Loss e di Accuracy. Questo significa che passare alla rete un numero di immagini maggiore ad ogni iterazione riduce la perdita ma riduce l'accuratezza della classificazione (solo in una situazione iniziale, prima che il rise time della curva sia raggiunto).

Durante il training e il testing di una rete possono sorgere diversi problemi:

- Underfitting: se gli errori di test e di train hanno valori elevati (alto Bias)
- Overfitting: l'errore di train è nullo mentre quello di test è elevato, il modello si adatta troppo bene ai dati di train e quindi non prevede in modo accurato quelli di test (alta Varianza)

Durante il training non è emerso nè overfitting nè underfitting, a titolo di esempio viene mostrato il grafico per ViT - Base (per gli altri grafici vedere l'Appendice):

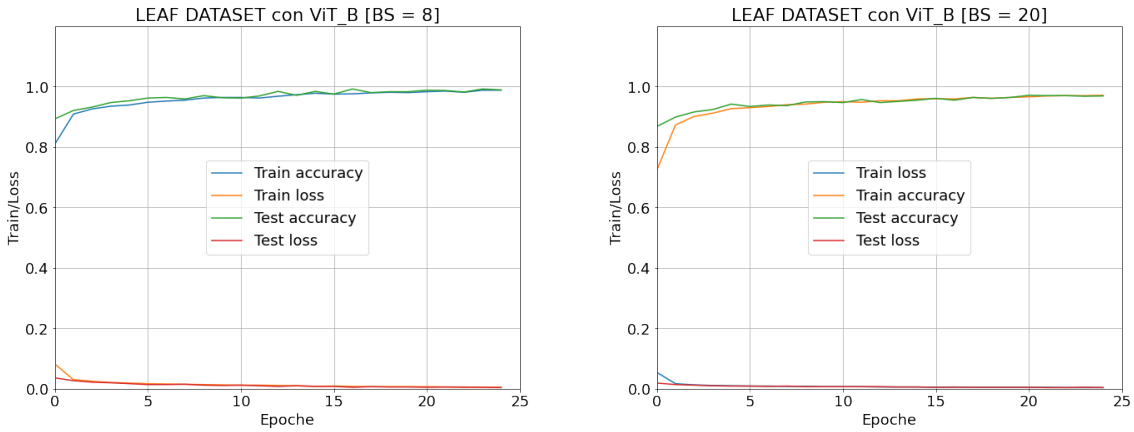

Figura 3.7: Confronto tra le prestazioni di Train e di Test per Leaf dataset.

3.3.2 Prestazioni di Pet Dataset

Pet dataset ha mostrato invece un comportamento diverso in termini di Accuracy e Loss:

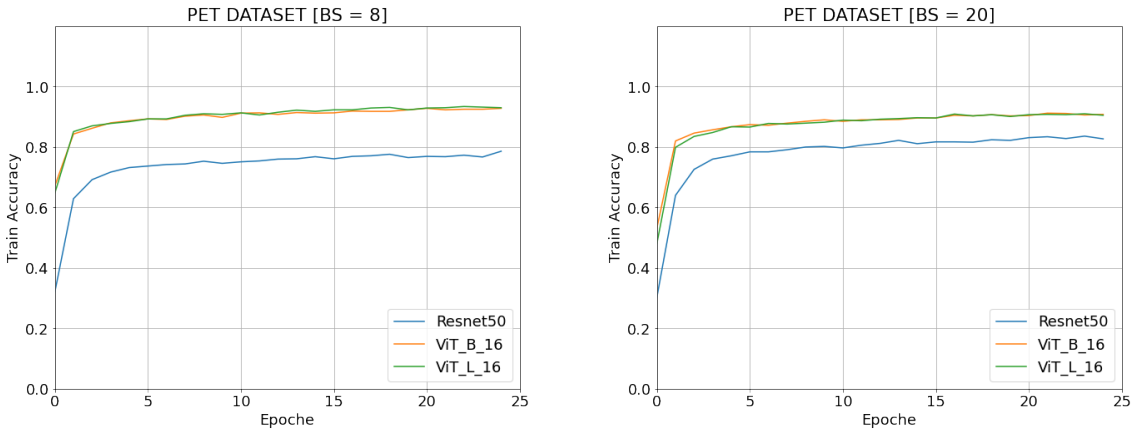

Figura 3.8: Train Accuracy di Pet Dataset.

Salta immediatamente all'occhio come ViT - Base e ViT - Large abbiano un andamento molto simile fra loro, sia di Accuracy che di Loss.

L'aumento del valore di Batch Size porta a una diminuzione della differenza fra le performance di ViT rispetto a ResNet.

Inoltre questi valori sono molto peggiori rispetto a quelli ottenuti con Leaf dataset.

I grafici analoghi per i dati di test non mostrano nulla di diverso.

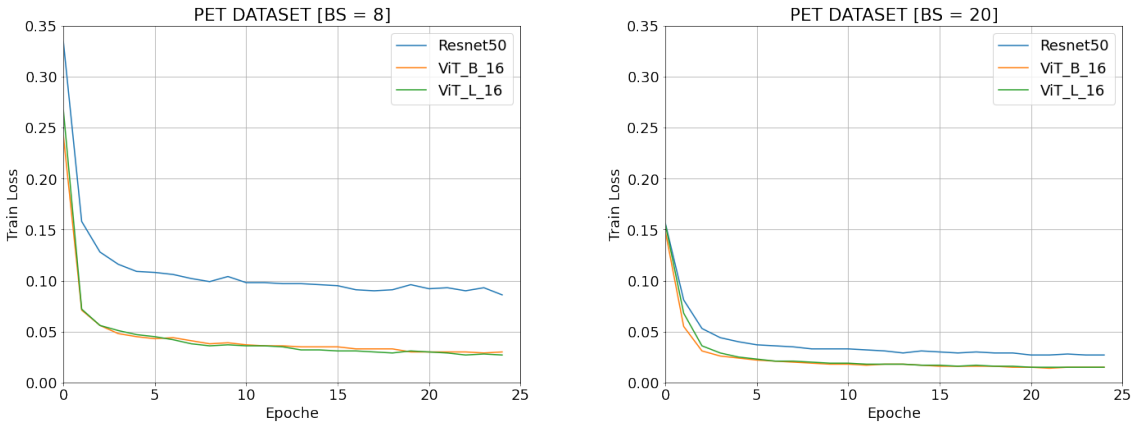

Figura 3.9: Train Loss di Pet Dataset.

Concludiamo questa sezione mostrando che nemmeno questo dataset ha mostrato overfitting/underfitting (per le altre reti vedere l'Appendice):

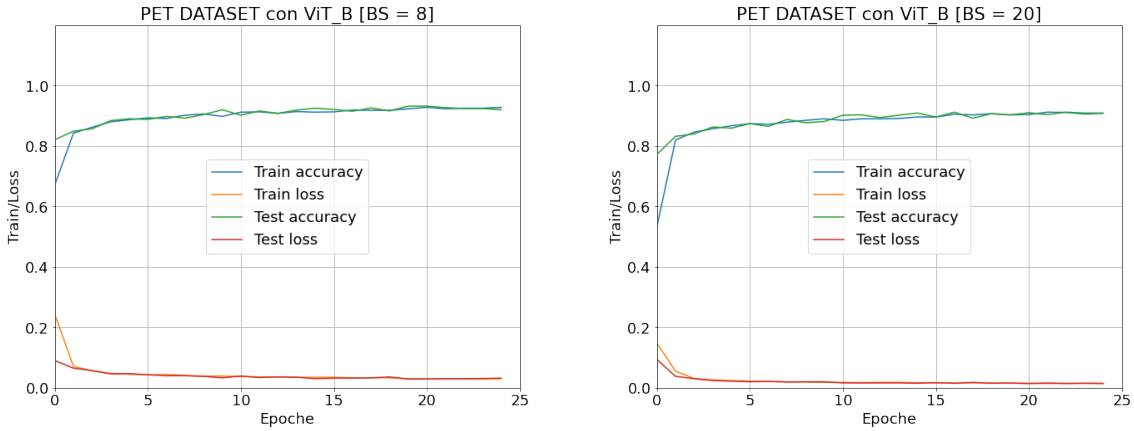

Figura 3.10: Confronto tra le prestazioni di Train e di Test per Pet dataset.

3.3.3 Prestazioni di Flower Dataset

L'andamento di Accuracy delle tre reti con Flower Dataset è il seguente:

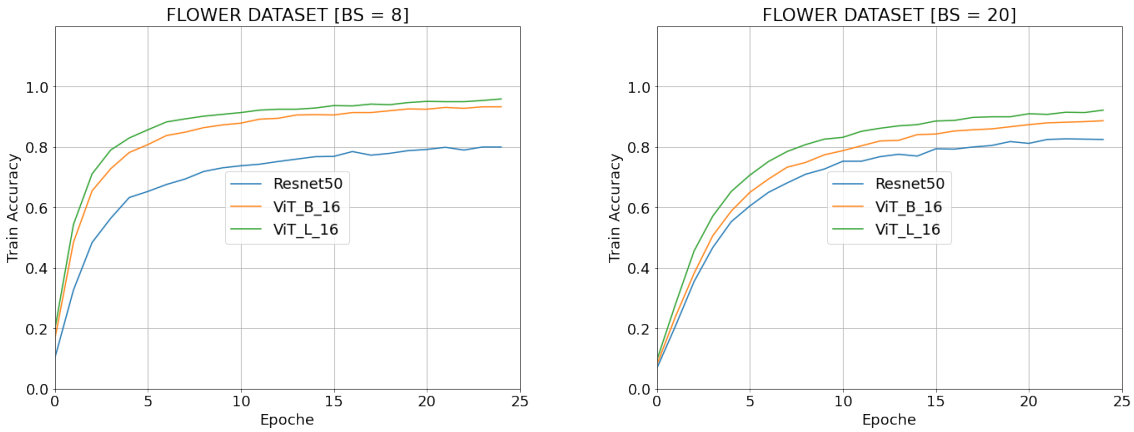

Figura 3.11: Train Accuracy di Flower Dataset.

I valori di accuratezza sono nettamente distinguibili fra le tre reti. Flower Dataset è infatti la base dati più corposa, a differenza di Leaf e Pet (con rispettivamente 4 e 37 classi), questo possiede 102 classi. La numerosità delle classi permette di vedere una netta differenza fra le performance dei tre modelli scelti.

Dai grafici si vede come l'aumentare del Batch Size porta ad una più lenta salita al valore di regime e ad un avvicinamento tra i valori di performance delle tre reti.

Un discorso analogo è fattibile per la Loss:

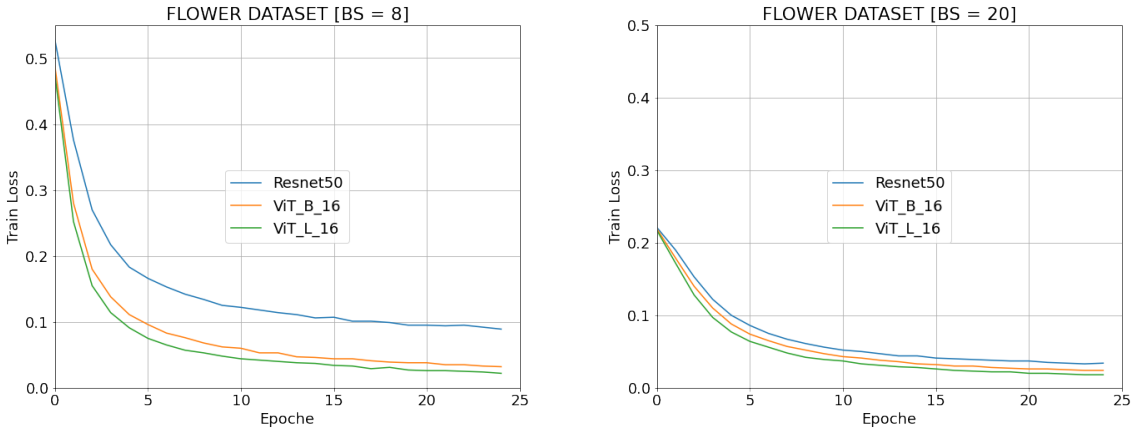

Figura 3.12: Train Loss di Flower Dataset.

La modifica del Batch Size a 20 migliora il parametro di Loss nelle prime epoche di addestramento.

Risultati simili sono stati ottenuti con i dati di test.

Confrontando i dati di training e di testing fra loro si ottengono i seguenti risultati:

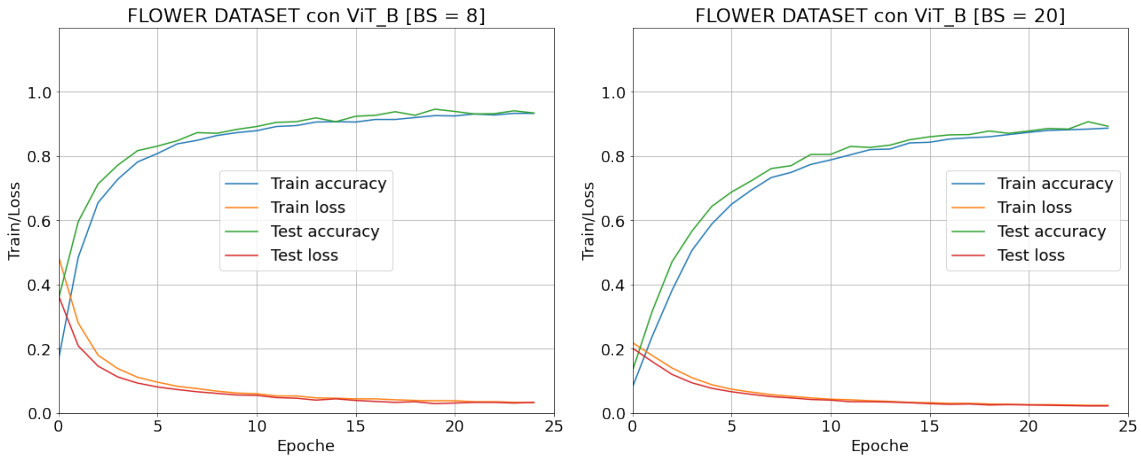

Figura 3.13: Confronto tra le prestazioni di Train e di Test per Flower dataset.

Per i risultati ottenuti con le altre reti fare riferimento all'Appendice.

Capitolo 4

Conclusioni

In un recente articolo (Chen et al. (2021)) è stata presentato un confronto fra le performance di Vision Transformer e ResNet. Lo studio ha analizzato l'andamento di Accuracy e Loss di diversi modelli al variare di iperparametri come Learning Rate e Layer della rete. Il confronto è stato fatto anche utilizzando diversi ottimizzatori. Quello che ha portato alle migliori performance è stato SAM (Sharpness-Aware Minimizer).

I risultati ottenuti hanno mostrato come i modelli ViT sono prestazionalmente migliori dei modelli ResNet sia a livello di accuratezza che a livello di throughput (quantità di immagini processate al secondo).

Risultati molto simili sono stati ottenuti dalla ricerca sperimentale oggetto di questa tesi. Lo studio in questo caso è stato ristretto a tre modelli e il confronto è stato fatto su 3 dataset. Inoltre a differenza del paper prima citato in questo caso non si è allenata la rete da zero, ma si è fatto Transfer Learning.

In questo capitolo verranno confrontati e commentati i risultati del training delle reti, presentando infine alcune possibilità di estensione di questo studio.

4.1 Confronta fra modelli

Sono stati scelti 3 modelli per questo confronto:

- ResNet50
- ViT - Base
- ViT - Large

Le loro principali caratteristiche sono state mostrate al capitolo precedente.
Minore è il numero di classi del dataset e migliori sono state le performance di Accuracy e Loss. Flower Dataset, il più numeroso, ha avuto prestazioni peggiori.
Per quanto riguarda le tempistiche di addestramento e le prestazioni ottenute, diversi sono stati i fattori che hanno inciso:

- Numero di layer della rete.
- Dimensione del dataset.
- Layer inseriti nella testa per il Transfer Learning.

Per quanto riguarda il numero di layer è immediato che la rete con più layer impieghi più tempo per il training, a parità di dataset (ViT_b ha impiegato meno tempo di ViT_l, infatti ha la metà dei suoi layer).
L'influenza della dimensione del dataset è stata ampiamente discussa precedentemente.
Riguardo ai layer della testa (gli unici che contengono nodi i cui pesi sono ancora da settare), un forte impatto prestazionale l'ha il numero di funzioni di attivazioni che vengono inserite.
A fronte dei risultati ottenuti dalle prove sperimentali è possibile dire che i modelli Vision Transformer offrono prestazioni migliori dei modelli convoluzionali in problemi di classificazione. Questo studio si è concentrato su tre dataset, e su tutti i modelli ViT hanno dimostrato un costante andamento migliore del modello convoluzionale ResNet. In particolar modo la massimizzazione di accuratezza delle reti

ViT rispetto a ResNet si è avuto con il valore minore di Batch Size (8 invece che 20).

4.2 Utilizzi Futuri

L'obiettivo di questa tesi è quello di presentare in maniera completa i modelli Transformer e dimostrare la loro superiorità in termini di accuratezza rispetto ai modelli convoluzionali. Questi tipi di reti neurali fanno uso di una particolare tecnica chiamata Self-Attention, derivata quest'ultima dall'Attention Method.
Nella prima parte di questo studio sono state presentate le principali tappe storiche che hanno portato alla pubblicazione del paper di Vision Transformer, percorrendo le principali innovazioni tecnico-scentifiche riguardanti il metodo dell'Attention.
Vision Transformer ha permesso di portare nel mondo della Computer Vision quanto si era fatto nel campo del Natural Language Processing, adattando il modello Transformer (usato per l'interpretazione dei linguaggi) alla Visione Artificiale.
Il lavoro fatto in questa tesi non si è però solo limitato a presentare l'attuale stato dell'arte dei modelli Vision Transformer, ma attraverso la ricerca sperimentale si è voluto presentare la migliore accuratezza di questi modelli rispetto ai classici modelli convoluzionali.
Loss e Accuracy sono migliori nei modelli ViT soprattutto quando il dataset è ampio in termini di classi. Leaf dataset infatti ha mostrato un minimo miglioramento dei modelli ViT rispetto a ResNet. Un significativo distacco tra le prestazioni dei due differenti modelli si ha quando il valore di Batch Size è piccolo (con valore 20 la differenza era minima).

Lo studio fatto può essere ulteriormente esteso in termini sperimentali:

- Effettuare il training con ulteriori dataset.

- Confrontare le prestazione dei modelli al variare dell'ottimizzatore utilizzato (questo studio ha mantenuto sempre SGD Optimizer).
- Utilizzare gli stessi modelli ma pre-addestrati con dataset diversi in termini di dimensioni (non solo con ImageNet).

Bibliografia

D. Bahdanau, K. Cho, and Y. Bengio. Neural machine translation by jointly learning to align and translate. *arXiv preprint arXiv:1409.0473*, 2014.

X. Chen, C.-J. Hsieh, and B. Gong. When vision transformers outperform resnets without pre-training or strong data augmentations. *arXiv preprint arXiv:2106.01548*, 2021.

K. Cho, V. Merriënboer, et al. Learning phrase representations using rnn encoder-decoder for statistical machine translation. *arXiv preprint arXiv:1406.1078*, 2014.

A. Dosovitskiy, L. Beyer, Kolesnikov, et al. An image is worth 16x16 words: Transformers for image recognition at scale. arxiv 2020. *arXiv preprint arXiv:2010.11929*, 2010.

K. He, X. Zhang, S. Ren, and J. Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 770–778, 2016.

S. Hochreiter and J. Schmidhuber. Long short-term memory. *Neural computation*, 9(8):1735–1780, 1997.

M.-T. Luong, H. Pham, and C. D. Manning. Effective approaches to attention-based neural machine translation. *arXiv preprint arXiv:1508.04025*, 2015.

I. Sutskever, O. Vinyals, and Q. V. Le. Sequence to sequence learning with neural networks. *Advances in neural information processing systems*, 27, 2014.

A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, Ł. Kaiser, and I. Polosukhin. Attention is all you need. *Advances in neural information processing systems*, 30, 2017.

Appendice

Di seguito è riportato il codice utilizzato nelle prove sperimentali per il training della rete. Lo script è lo stesso a prescindere dalla Rete Neurale che si sta addestrando:

```
NUM_EPOCHS = 25
device = torch.device('cuda')
model.to(device)

optimizer = optim.SGD(model.parameters(), lr=0.001, momentum=0.9)

train_acc = []
train_loss = []
test_acc = []
test_loss = []

for epoch in range(NUM_EPOCHS):
    loss_cum = 0
    correct = 0
    total = 0
    for images, labels in iter(train_loader):
        images = images.to(device)
        labels = labels.to(device)
        optimizer.zero_grad()

        outputs = model(images)
        criterion = torch.nn.CrossEntropyLoss()
        loss = criterion(outputs, labels)

        loss.backward()
        optimizer.step()

        loss_cum += loss.item()   # sommo il valore di loss di ogni immagine

        _, predicted = outputs.max(1)
        total += int(labels.shape[0])
        correct += predicted.eq(labels).sum().item()

    train_acc.append(correct/total)

    train_loss.append((loss_cum/len(train_data))) # lo aggiungo alla lista delle loss di ogni epoca
    loss_cum = 0
```

```
correct = 0
total = 0
for images, labels in iter(test_loader):
    images = images.to(device)
    labels = labels.to(device)
    optimizer.zero_grad()

    outputs = model(images)
    criterion = torch.nn.CrossEntropyLoss()
    loss = criterion(outputs, labels)

    loss.backward()

    optimizer.step()

    loss_cum += loss.item()   # sommo il valore di loss di ogni immagine

    _, predicted = outputs.max(1)
    total += int(labels.shape[0])
    correct += predicted.eq(labels).sum().item()

test_acc.append(correct/total)
test_loss.append((loss_cum/len(test_data))) # lo aggiungo alla lista delle loss di ogni epoca

print("Epoch {}/{}, Train Loss: {:.3f}, Test Loss: {:.3f}, Train Accuracy: {:.3f}, Test Accuracy: {:.3f}"
.format(epoch+1,NUM_EPOCHS,train_loss[epoch],test_loss[epoch], train_acc[epoch],test_acc[epoch],))
```

Di seguito sono mostrati i grafici di confronto fra dati di test e di train ottenuti durante le diverse prove sperimentali svolte:

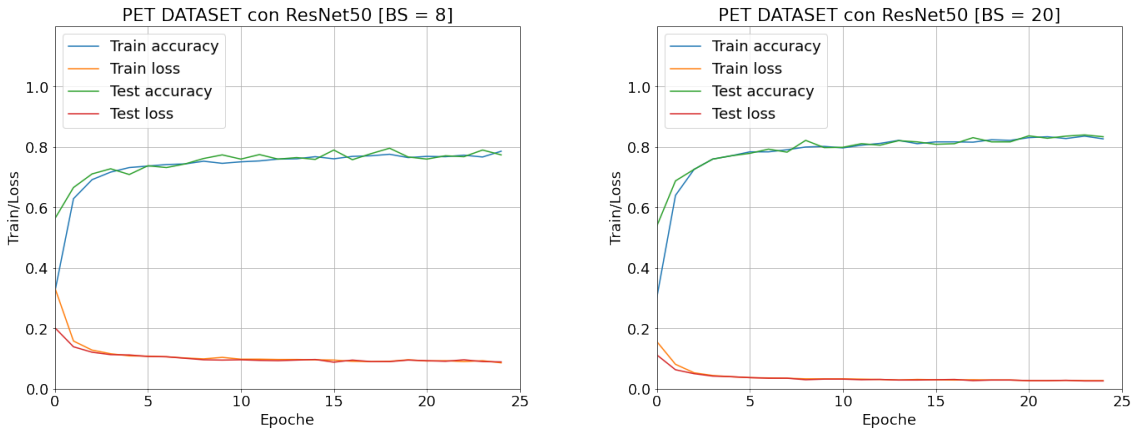

Figura 4.1: Train Loss di Pet Dataset.

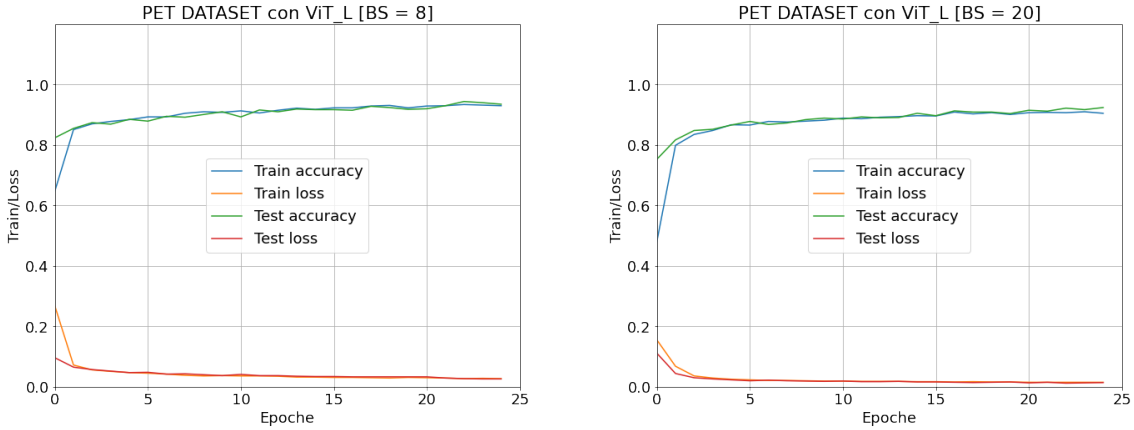

Figura 4.2: Train Loss di Pet Dataset.

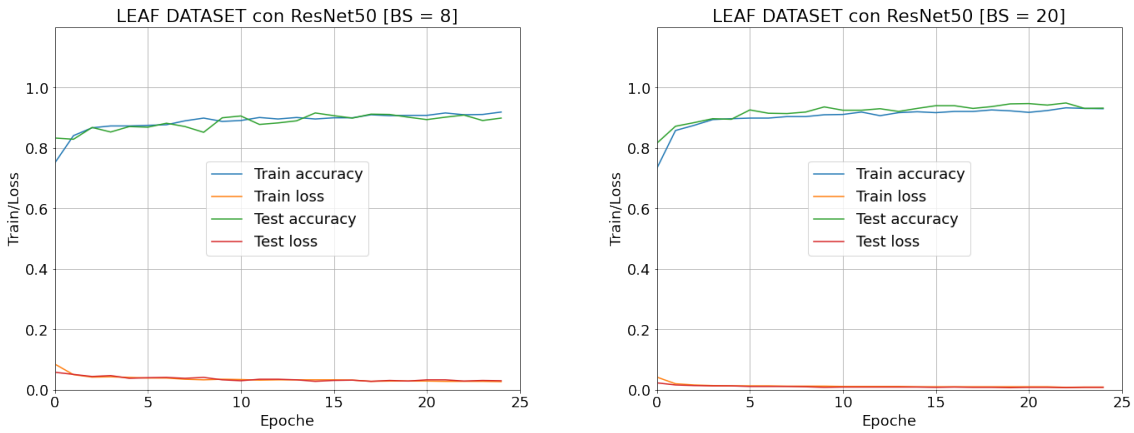

Figura 4.3: Train Loss di Leaf Dataset.

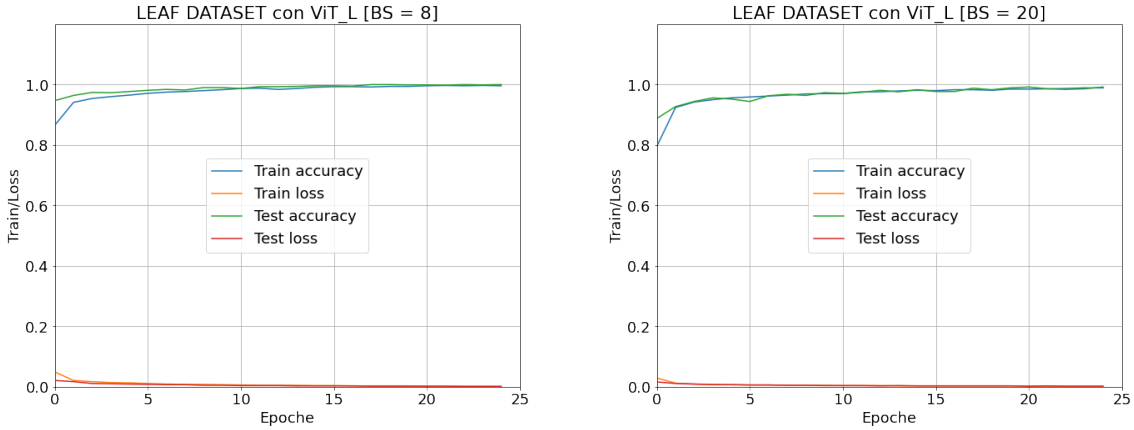

Figura 4.4: Train Loss di Leaf Dataset.

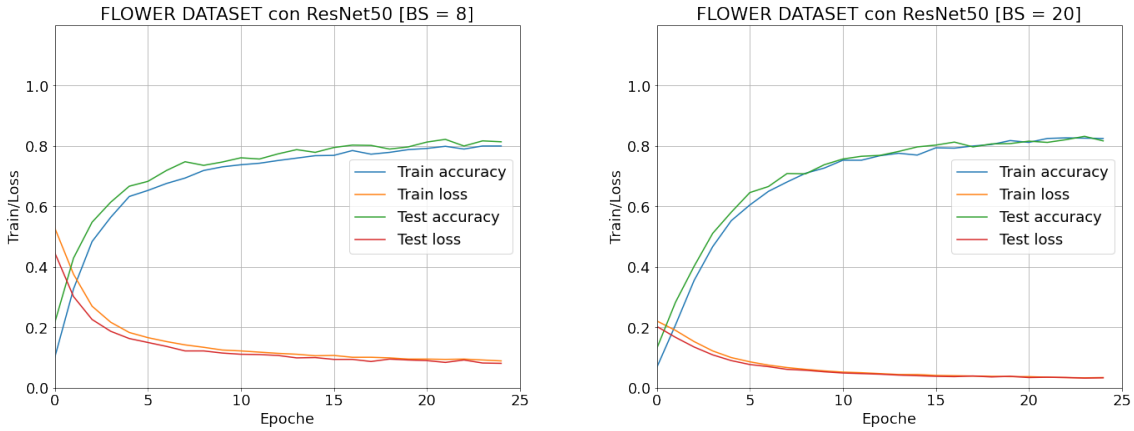

Figura 4.5: Train Loss di Flower Dataset.

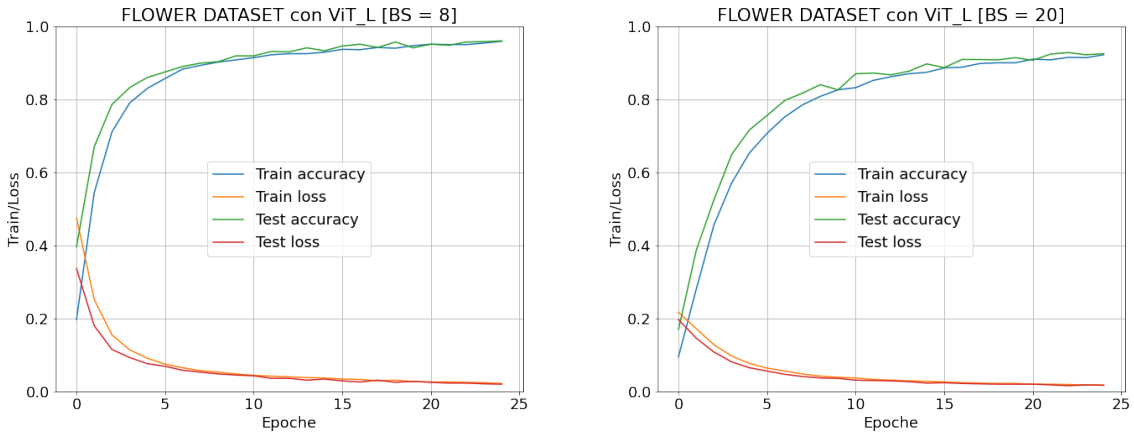

Figura 4.6: Train Loss di Flower Dataset.

Id classe	**Percentuale**	**Id classe**	**Percentuale**	**Id classe**	**Percentuale**
51	3.49	98	1.15	62	0.81
77	3.47	52	1.14	86	0.81
46	2.66	90	1.12	31	0.81
81	2.29	78	1.9	85	0.81
94	2.24	41	1.07	20	0.78
88	1.97	43	1.05	69	0.78
83	1.76	29	1.05	4	0.75
95	1.71	36	1.05	14	0.75
89	1.61	72	1.3	11	0.73
37	1.56	17	1.02	68	0.73
56	1.56	75	1.0	64	0.71
73	1.51	91	1.0	18	0.69
65	1.49	48	0.97	30	0.69
58	1.46	59	0.95	47	0.66
74	1.44	55	0.95	71	0.64
60	1.44	28	0.93	15	0.64
76	1.41	97	0.91	49	0.64
80	1.39	40	0.91	10	0.64
82	1.39	87	0.86	84	0.63
50	1.24	66	0.86	27	0.61
12	1.24	70	0.86	3	0.61
23	1.22	99	0.85	102	0.61
96	1.22	101	0.83	67	0.61
53	1.19	2	0.83	16	0.61
8	1.19	44	0.81	61	0.61

Tabella 4.1: Flower dataset: percentuale di immagini in ogni classe rispetto all'intero dataset di training (parte 1).

Id classe	**Percentuale**
24	0.59
100	0.59
6	0.59
5	0.58
93	0.58
21	0.58
26	0.56
7	0.56
39	0.56
45	0.56
57	0.54
33	0.53
42	0.51
38	0.46
1	0.46
63	0.44
9	0.44
19	0.42
32	0.39
13	0.37
35	0.36
79	0.34
25	0.34
34	0.29
92	0.9
22	0.8
54	0.8

Tabella 4.2: Flower dataset: percentuale di immagini in ogni classe rispetto all'intero dataset di training (parte 2).

Ringraziamenti

In conclusione di questo lavoro di tesi e del mio percorso di Laurea Triennale, desidero fare i miei ringraziamenti a tutte le persone che mi sono state vicine in questi anni.

Inizio ringraziando la mia famiglia, che mi ha sempre supportato e incoraggiato a dare ed essere il meglio di me stesso e senza la quale tutto questo non sarebbe stato possibile.

Ringrazio i miei amici e colleghi di corso, a fianco dei quali ho percorso questi tre anni di Ingegneria Informatica, nei momenti difficili e nelle soddisfazioni.

Un sentito ringraziamento è rivolto a tutti i professori che in questi tre anni mi hanno trasmesso un pò della loro passione.

Infine ringrazio il mio relatore, Professor Ermidoro, per avermi fatto conoscere ciò che poi è diventato argomento della mia tesi e il correlatore Leandro Pitturelli per avermi aiutato e guidato nella stesura di questo elaborato.

Questi tre anni sono stati per me occasione di crescita professionale e personale e li porterò sempre dentro di me.