

Classificazione della composizione fotografica delle immagini

Relatore: *Dott. Luigi Celona*

Correlatore: *Prof. Raimondo Schettini*

Correlatore: *Prof. Gianluigi Ciocca*

Relazione della prova finale di:

Alessandro Locatelli

Matricola 879362

Anno Accademico 2023-2024

Abstract

La composizione fotografica è un aspetto che spesso viene trascurato ai fini della valutazione della bellezza estetica delle immagini e realizzazione delle stesse da parte di fotografi amatoriali. La modalità con la quale il soggetto e gli elementi ad esso circostante vengono posizionati nello scatto può cambiare completamente il suo aspetto, la sua capacità narrativa, le sensazioni che esso riesce a trasmettere: la simmetria crea armonia e bilanciamento, elementi verticali, tendenti al cielo, trasmettono potenza e forza, profili curvi comunicano dinamismo e movimento, e così via.

Essendo un fondamentale di cui difficilmente il fotografo occasionale è a conoscenza, l'intelligenza artificiale potrebbe essere uno strumento utile ad assisterlo in questo compito e migliorare immediatamente la qualità dei suoi scatti. Attualmente, nei software delle fotocamere dei nostri smartphone esistono già strumenti che ci suggeriscono in tempo reale quale sia lo scatto migliore da effettuare. Questa funzionalità potrebbe essere ancora più potenziata introducendo anche il riconoscimento delle caratteristiche compositive di cui si discuterà in questa relazione.

Recenti sviluppi nel campo del **self-supervised learning** hanno portato alla nascita di DINOv2, un foundational model di MetaAI in grado di produrre features universali utili in un grande range di task a livello visivo (*image classification, instance retrieval, video understanding*) e a livello di pixel (*semantic segmentation, depth estimation*).

L'obiettivo di questa tesi è condurre uno studio delle più recenti tecniche di self-supervision nell'ambito della classificazione delle immagini ed esplorare la possibilità di impiegare questo tipo di approccio nel task di classificazione della composizione fotografica, valutandone il confronto con le metodologie attualmente utilizzate nello stato dell'arte per il medesimo problema.

Indice

1 Composizione fotografica	1
1.1 Cos'è la composizione fotografica	1
1.1.1 Regola dei terzi	1
1.1.2 Leading Lines	2
1.1.3 Forme e patterns	3
1.2 Classificare la composizione	3
1.3 Scopo e utilizzi pratici	5
1.4 Outline	6
2 Stato dell'arte e lavori correlati	7
2.1 Prima del Deep Learning	7
2.2 Fine-tuning di una CNN	8
2.3 Spatial-invariant CNN	8
2.4 Grafi come descrittori del layout	10
3 Self-Supervised Learning	13
3.1 Cos'è il self-supervised learning	13
3.1.1 Pro e contro	14
3.2 Augmentations	15
3.3 Contrastive Learning	16
3.3.1 SimCLR	16
3.3.2 Problemi col contrastive learning	17
3.4 Collasso	18
3.5 Self-Distillation	19
3.5.1 BYOL	20
3.6 DINO	22
3.6.1 Dettagli implementativi	23
3.7 Migliorare DINO	25
3.7.1 BERT	25
3.7.2 iBOT	25
3.8 DINOv2	28
3.8.1 LVD-142M	28
3.8.2 Architettura	29
4 Metodo proposto	31
4.1 Feature Extractor: DINOv2	31
4.2 Features	31
4.2.1 CLS	31
4.2.2 AVG patch	32

4.2.3	Combinare le features di più layers	32
4.3	Classificatore	33
4.4	Loss	33
4.5	Preparazione delle immagini	34
5	Dataset	35
5.1	KU-PCP	35
5.2	LODB	38
5.3	Considerazioni	38
6	Sperimentazione e Risultati	42
6.1	Setup sperimentale	42
6.1.1	Partizione in train, test, validation	42
6.1.2	Metriche	42
6.2	Considerazioni sulle features	44
6.2.1	CLS	44
6.2.2	AVG	45
6.2.3	Concatenazione di CLS e AVG	47
6.3	Risultati finali	47
6.3.1	Linear layer	48
6.3.2	MLP	48
7	Conclusioni e sviluppi futuri	55
7.1	Conclusioni	55
7.1.1	DINOv2	55
7.1.2	Riguardo i datasets	56
7.2	Sviluppi futuri	58
7.2.1	Dataset	58
7.2.2	Fine Tuning	59
7.2.3	Mappe di self-attention	59
7.2.4	Segmentazione delle leading lines	60
Riferimenti		64
Bibliografia	64
Siti	65
A	Transformers e Vision Transformers	66
A.1	Transformers	66
A.1.1	Preparare l'input	67
A.1.2	Encoder	67
A.1.3	Decoder	68
A.1.4	Self-attention	68
A.1.5	Multi-head attention	68
A.2	Vision Transformers (ViT)	69
A.2.1	Token [CLS]	69
A.2.2	Attention	70
A.2.3	Tipologie	70

Elenco delle figure

1.1	Un esempio di regola dei terzi. I due soggetti sono posizionati lungo una delle linee guida, e il punto focale dello scatto si trova sull'intersezione di due di esse.	2
1.2	Esempi di leading lines orizzontali, verticali e curve, rispettivamente.	2
1.3	Esempio di (a) leading lines convergenti, corrispondenti con l'unirsi dei binari all'orizzonte (b) leading lines diagonali.	3
1.4	Esempi di (a) un soggetto triangolare (b) un pattern	4
1.5	Le leading lines curve implicite date dalla forma della staccionata risultano più importanti all'occhio umano rispetto alla separazione fra cielo e terreno.	4
1.6	Immagini classificabili con più tipi di composizione: (a) horizontal leading line e pattern, (b) curva e pattern, (c) verticale e RoT	5
2.1	L'architettura proposta da [14], composta di cinque layer convoluzionali (c_1, c_2, c_3, c_4, c_5) e tre <i>fully connected layers</i> (fc_6, fc_7 , soft-max). Il layer di soft-max produce i punteggi di confidenza per le 9 classi contenute nel dataset KU-PCP.	8
2.2	Framework per la spatial-invariant convolution neural network.	9
2.3	La rotazione dell'immagine viene effettuata introducendo del padding speculare intorno all'immagine, ruotando il risultato e croppando per tornare alle dimensioni originali. Gli autori osservano come, se la classe iniziale dell'immagine è quella di una leading line orizzontale data dall'orizzonte, distorcere l'immagine con una leggera rotazione non cambia il nostro giudizio sulla composizione, la percepiamo ancora come classe orizzontale riconoscendone il contesto.	10
2.4	Workflow seguito da [28].	11
3.1	Un esempio di alcune delle augmentations più utilizzate in ambito di SSL.	15
3.2	Architettura di SimCLR. Questa architettura viene anche detta <i>siamese</i> , perchè composta da 2 sotto-reti che condividono architettura e pesi. L'immagine mostra un esempio per coppie positive.	17
3.3	Valori di accuracy di SimCLR al variare delle augmentations scelte per produrre coppie positive e negative. Si nota quanto diverse combinazioni influiscano sulle performance della rete.	18
3.4	Visualizzazione dello spazio delle features nel caso di collasso dimensionale o completo.	19
3.5	Visualizzazione del workflow utilizzato nella knowledge distillation.	20
3.6	Architettura di BYOL.	21
3.7	Workflow per il calcolo della loss simmetrizzata.	21

3.8	Mappe di self-attention estratte dal token [CLS], sulle teste del blocco di <i>multi-head attention</i> dell'ultimo layer.	23
3.9	Workflow per multi-cropping. A partire da una immagine vengono prodotte 2 viste globali e 6 viste locali. In questo modo aumentano i samples senza ricalcolare altre augmentations.	23
3.10	Architettura di DINO. Rete teacher e student hanno stessa architettura, ma parametri diversi, come BYOL.	24
3.11	Confronto fra le teste di self-attention per il token [CLS] in iBOT e DINO. Si nota come le segmentazioni prodotte da iBOT siano più precise e meno rumorose, e come ciascuna testa rilevi un singolo particolare dettaglio dell'immagine relativa, in maniera più granulare rispetto DINO.	26
3.12	Architettura di iBOT. $h_s^{[CLS]}$ e h_s^{patch} condividono i pesi, così come $h_t^{[CLS]}$ e h_t^{patch} . L'input dello Student viene mascherato solo per l'obiettivo MIM, non per la costruzione di CLS.	27
3.13	Confronto fra iBOT e altri modelli self-supervised. iBOT supera le prestazioni della concorrenza.	27
3.14	Architettura di DINOv2. È la stessa di iBOT (Figura 3.12, con la differenza che i pesi delle teste sono separati questa volta.	28
4.1	Estrazione dei token CLS e patch AVG dal ViT di DINOv2	32
4.2	Concatenazione delle features	32
5.1	Un esempio di immagine per ciascuna classe nel dataset KU-PCP.	37
5.2	Un esempio di immagine per ciascuna classe nel dataset LODB.	41
6.1	Grafico della loss in validation (quinto fold) su LODB e linear layer per la configurazione 4 token CLS e 4 AVG patch, 200 epoche, learning rate 0.003, che viene usata in fase di test. Non ci sono segnali di overfitting.	49
6.2	Grafico della loss in validation (quinto fold) su KU-PCP e linear layer per la configurazione con 4 token CLS e 4 AVG patch, 300 epoche e learning rate 0.003, che viene usata in fase di test. La discesa della loss non mostra segni di overfitting.	50
6.3	Grafico della loss in validation (quinto fold) su KU-PCP e MLP per la configurazione 4 token CLS e 4 AVG patch, 200 epoche, learning rate 0.005, che viene usata in fase di test. Non ci sono segnali di overfitting.	51
6.4	Grafico della loss in validation (quinto fold) su LODB e MLP per la configurazione 4 token CLS e 4 AVG patch, 300 epoche, learning rate 0.002, che viene usata in fase di test. Non ci sono segnali di overfitting.	51
6.5	Esempio di predizione corretta della rete con linear classifier su KU-PCP. Le classi di ground truth sono <i>RoT</i> . e <i>Horizontal</i> . In rosso è segnata la soglia di threshold (posta a 0.5) utilizzata per il calcolo delle metriche.	52
6.6	Predizione con classificatore lineare su KU-PCP. Le classi di ground truth sono <i>Vertical</i> e <i>Symmetric</i> , mentre la nostra rete predice <i>Horizontal</i> e <i>Symmetric</i> . Effettivamente, essendo uno scatto in lontananza della città, somiglia più alle immagini che nel dataset sono classificate come <i>Horizontal</i> perché ritraente un orizzonte, piuttosto che <i>Vertical</i> , di cui solitamente fanno parte immagini di palazzi in vista più ravvicinata. Le prestazioni, quindi, sono anche influenzate da un cattivo o non consistente etichettamento delle immagini nel dataset.	52

6.7	Predizione con classificatore lineare su KU-PCP, ground truth <i>Symmetric</i> e <i>Center</i> . Ancora una volta, l’etichettamento dato all’immagine nel dataset è fuorviante. La rete intuisce la forma dell’edificio e le curve che lo compongono, che sarebbero potute essere altre due valide classi di ground truth per l’immagine in questione.	53
6.8	Predizione con classificatore lineare su LODB, ground truth <i>O3Tri</i> .. Il classificatore è molto confuso sulla classe da assegnare, nessuna raggiunge la soglia al di sopra della quale si considerano le predizioni. Questo risultato scarso è in parte dovuto anche all’etichetta assegnata all’immagine. La rete ha delle intuizioni valide: il primo piano di un soggetto viene spesso associato alla classe <i>Center</i> nel dataset, prendendo i fiori a coppie si potrebbero considerare come posti in diagonale, il fiore rosso a sinistra è in posizione <i>RoT.L</i> , la ripetizione dei petali potrebbe essere visto come <i>Pattern</i> , la forma del fiore come <i>Radi</i> .. Si notano le limitazioni delle classi introdotte da LODB, di granularità troppo elevata e che spesso si sovrappongono l’una all’altra, nonostante le immagini siano per la maggior parte etichettate con una sola label.	53
6.9	Predizione con classificatore lineare su LODB, ground truth <i>O2DiaR</i> .. La rete predice <i>RoT.R</i> , sbagliando completamente. <i>RoT.L</i> sarebbe potuta essere una buona intuizione, l’uomo si trova all’intersezione degli assi sinistro e inferiore della griglia della regola dei terzi, ma con questa predizione il modello mostra di non aver compreso effettivamente la differenza fra classi <i>R</i> e <i>L</i> , indicato anche dal fatto che la seconda probabilità più alta sia quella di <i>O2DiaL</i>	54
6.10	Predizione con classificatore lineare su LODB, ground truth <i>O2Hor</i> .. Anche su immagini la cui composizione risulta essere piuttosto chiara all’occhio umano, il modello fatica a riconoscerla e spalma le sue predizioni su diverse classi, facendo fatica a prendere una decisione.	54
7.1	Esempi di immagini nella classe <i>vertical</i> nella partizione di test KU-PCP. Una quantità importante di immagini che appartengono a questa classe sono fotografie di grattacieli.	56
7.2	Esempi di immagini appartenenti alla classe <i>symmetric</i> nella partizione di test di KU-PCP. A livello semantico, le immagini sono molto simili: montagne, alberi, foreste, riflessi in corpi d’acqua.	56
7.3	Esempi di immagini classificate come <i>vertical</i> in LODB che non corrispondono a leading lines verticali.	57
7.4	Esempi di misclassification/forzatura della composizione. Sono indicate le etichette che LODB assegna a ciascuna.	58
7.5	Alcuni esempi di immagini di KU-PCP le cui mappe di attention permettono di individuare chiaramente la classe assegnata. Sono indicate le classi di ground truth.	61
7.6	Classe <i>vertical</i> . Si nota come l’attention si concentri sul ponte che collega i due grattacieli, ma ignori tutti gli altri edifici circostanti. Questo potrebbe rendere complicato il riconoscimento della sua classe.	61

7.7	Le classi di ground truth sono <i>RoT</i> . e <i>Diagonal</i> . Dalla mappa originale è difficilmente possibile distinguere le due classi, specialmente <i>Diagonal</i> . Una CNN potrebbe fare fatica a causa del rumore che circonda gli elementi chiave. Applicando uno smoothing all’immagine originale e ricalcolando la self-attention su di essa, la mappa prodotta risulta essere più chiara.	62
7.8	Tutte le teste di self-attention dell’ultimo transformer block per un’immagine di KU-PCP, con ground truth <i>Horizontal</i> . In queste caso le migliori sono la prima e la terza, dalle altre è sostanzialmente impossibile dedurre la classe. Non è sempre vero, per altre immagini la prima e terza testa potrebbero essere le peggiori.	63
A.1	Architettura di un transformer.	67
A.2	Workflow per il calcolo della self-attention: (a) scaled dot-product attention, (b) multihead-attention	69
A.3	(a) Architettura di un ViT, (b) Encoder	70

Elenco delle tabelle

3.1	Confronto top-1 e top-5 accuracies in linear evalutation su ImageNet usando una ResNet-50 come encoder fra BYOL e altri metodi di SSL. Si nota come BYOL migliori in maniera rilevante le prestazioni di SimCLR e altri modelli concorrenti.	22
3.2	Confronto top-1 accuracy per linear e kNN sul validation set di ImageNet fra DINO e diversi altri metodi di self-supervision. In grigio i risultati supervisionati, in arancione quelli di DINO.	25
3.3	Un confronto fra iBOT e DINOV2. Si mostrano tutte le novità introdotte in quest'ultimo e il guadagno di prestazioni che comportano.	29
5.1	Distribuzione delle classi nel dataset KU-PCP nella partizione di test. Siccome alcune immagini hanno 3 classi di ground-truth, può essere che un totale sia più piccolo della somma della riga corrispondente.	36
5.2	Distribuzione delle classi nel dataset LODB.	39
6.1	Un confronto fra le prestazioni del layer lineare utilizzando diverse combinazioni di layer di token [CLS]	45
6.2	Metriche per classe del quinto fold nella configurazione $CLS_5 \dots CLS_{12}$ (8 layers), lr = 0.002. Si nota come gli F1-scores più alti siano in corrispondenza delle classi più numerose nel dataset. Allo stesso modo, le classi che hanno alta precision e bassa recall sono quelle meno frequenti o più granulari, che spesso codificano differenze nel numero dei soggetti nell'immagine.	46
6.3	Matrici di confusione del quinto fold per <i>DiaX</i> e <i>O3Tri</i> nella configurazione 8 CLS e lr = 0.002 e nel fold 5. Si nota come <i>DiaX</i> , che conta 861 esempi nel dataset, venga individuata correttamente la maggior parte delle volte, mentre <i>O3Tri</i> , 115 esempi, non venga mai predetta, da cui le metriche azzerate in Tabella 6.2. Si vede anche come in questo specifico fold siano presenti 130 istanze di <i>DiaX</i> , contro solamente 11 di <i>O3Tri</i>	46
6.4	Un confronto fra le prestazioni del layer lineare utilizzando diverse combinazioni di layer di AVG patch	47
6.5	Un confronto fra le prestazioni del layer lineare utilizzando diverse combinazioni di layer di token [CLS] e AVG patch.	47
6.6	Risultati in fase di test utilizzando un linear layer. Tutti i risultati sono presentati con threshold per le probabilità posto a 0.5.	48
6.7	Confronto fra il nostro metodo e gli altri metodi in letteratura sviluppati per la classificazione su KU-PCP discussi nel Capitolo 2.	48

6.8	Metriche per classe su LODB in test con linear layer. Come già discusso precedentemente, si notano F1-score bassi in classi poco frequenti o che codificano il conteggio dei soggetti, e score più elevati in classi più numerose, accompagnate da alte precision e recall.	49
6.9	Metriche per classe in test su KU-PCP con linear layer. A supporto della teoria che le features di DINOv2 riconoscano il contenuto semantico nelle immagini e usino quello per la classificazione, come si esplorerà nel prossimo capitolo, <i>RoT</i> è la classe con prestazioni peggiori ed è anche quella che presenta varianza semantica intraclasse più elevata, al contrario di classi come <i>Symmetric</i> o <i>Pattern</i> , che infatti presentano risultati migliori.	50
6.10	Risultati in fase di test utilizzando un MLP. I valori sono simili a quelli ottenuti tramite un singolo linear layer (Tabella 6.6) con differenze inferiori al 5% nella maggior parte dei casi, che potrebbero essere ridotte ancora di più prolungando l'allenamento o incrementando il learning rate.	51

1

Composizione fotografica

Esistono diversi fattori che dovrebbero essere tenuti in considerazione quando si vogliono catturare fotografie di qualità professionale. Alcuni dei più noti sono la luce, l'esposizione, la messa a fuoco, l'apertura, la scelta delle lenti. Fra questi, però, esiste un aspetto di cui raramente si discute: la composizione o *image layout*.

1.1 Cos'è la composizione fotografica

Con composizione fotografica si intende disposizione strategica e calcolata degli elementi all'interno dell'immagine. Questo è un fattore determinante del modo in cui percepiamo e trasmettiamo il contenuto visivo di uno scatto. La composizione **governa il flusso di informazione**, dirige lo sguardo dell'osservatore verso determinati punti di interesse, influenza profondamente la nostra comprensione e **risposta emotiva** di fronte ad una immagine. Lo scopo ultimo di un fotografo dovrebbe essere quello di produrre una fotografia che sia visivamente piacevole e che allo stesso tempo possa trasmettere un messaggio, una storia, un sentimento. Questo risultato può essere ottenuto tramite un attento posizionamento del soggetto all'interno dello scatto e degli elementi attorno ad esso.

Esistono diverse linee guida ampiamente accettate in ambito fotografico professionale come buone convenzioni da seguire per comporre uno scatto di qualità. Di seguito se ne discutono alcune delle più importanti. Si analizzeranno più nel dettaglio tutte le regole di composizione trattate con ulteriori immagini di esempio quando si andranno a introdurre i datasets utilizzati per svolgere il task di classificazione (Capitolo 5).

1.1.1 Regola dei terzi

La regola dei terzi (RoT, *rule of thirds*) divide l'immagine in una griglia di tre righe e tre colonne, per un totale di nove blocchi di ugual dimensione. L'idea è quella di **posizionare il soggetto** o gli elementi principali della fotografia, i punti focali, lungo le linee divisorie o **nei punti di intersezione** fra esse (Figura 1.1), che sono considerati i più armoniosi e visivamente interessanti per l'occhio umano.

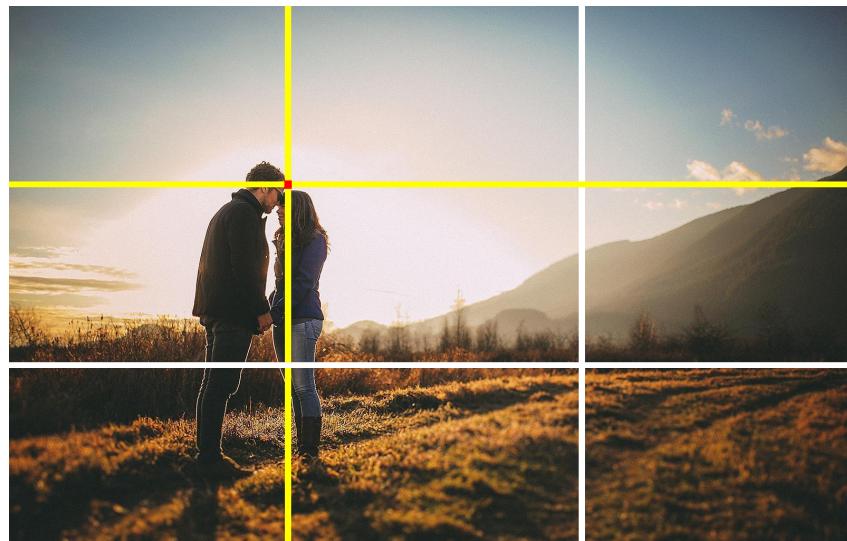


Figura 1.1: Un esempio di regola dei terzi. I due soggetti sono posizionati lungo una delle linee guida, e il punto focale dello scatto si trova sull'intersezione di due di esse.

Posizionare il soggetto in una di queste posizioni, piuttosto che al centro dell'immagine, conferisce un maggiore senso di **equilibrio**, **naturalezza**, aiuta a **guidare lo sguardo** dell'osservatore **verso i punti di interesse** principali della foto. Inoltre, la RoT può aiutare a dare un senso di profondità, di movimento, dinamismo, introduce un elemento di interesse ulteriore che aiuta ad elevare gli scatti.

1.1.2 Leading Lines

Un'altra tecnica spesso utilizzata da fotografi professionisti in ambito di composizione è quella delle *leading lines*. Le leading lines sono delle vere e proprie **linee che guidano l'occhio dell'osservatore attraverso l'immagine** e verso il soggetto dello scatto o un altro punto di interesse che si vuole mettere in risalto. Possono essere linee reali o immaginarie, date dal posizionamento di fiumi, recinzioni, binari ferroviari, strade, linee architettoniche, orizzonti, alberi e altro.

Possiamo distinguere diversi tipi di leading lines, ciascuna delle quali può trasmettere diverse sensazioni nel subconscio dell'osservatore:

- Orizzontali: inconsciamente siamo soliti ad analizzare le immagini linearmente, da sinistra verso destra. Per questo motivo, linee orizzontali possono dare un senso di **comforto**, **stabilità**, **tranquillità**. Alcuni esempi possono essere orizzonti, file di alberi, linee costiere. Possono anche individuare simmetrie all'interno dell'immagine.

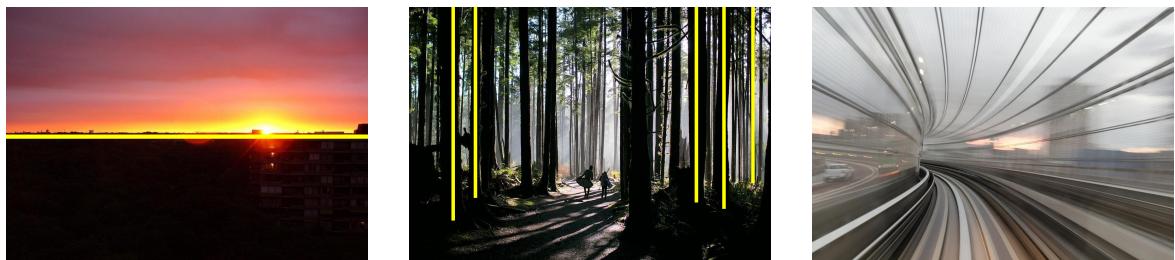


Figura 1.2: Esempi di leading lines orizzontali, verticali e curve, rispettivamente.

magine, come una montagna specchiata sulla superficie piatta di un lago, creando armonia e bilanciamento.

- Verticali: in natura e nel mondo urbano, linee verticali spesso si presentano sotto forma di alberi e alti edifici. Per questo motivo, al contrario del caso precedente, elementi verticali possono indurre una sensazione di sopraffazione, l'occhio è spinto a seguire tanti percorsi in maniera non lineare. Inoltre possono trasmettere **forza, crescita, potenza**.
- Curve: possono guidare lo sguardo in modo più morbido e naturale, creando un senso di **flusso, dinamismo, movimento**, che è difficile da riprodurre in scatti statici. Includono fiumi, sentieri, strade, curve architettoniche. In natura scarseggiano, spesso vengono accentuate artificialmente dall'utilizzo di lenti grandangolari.

Altri tipi di leading lines possono essere diagonali o convergenti, a formare un forte punto focale, come dei binari che convergono in lontananza.

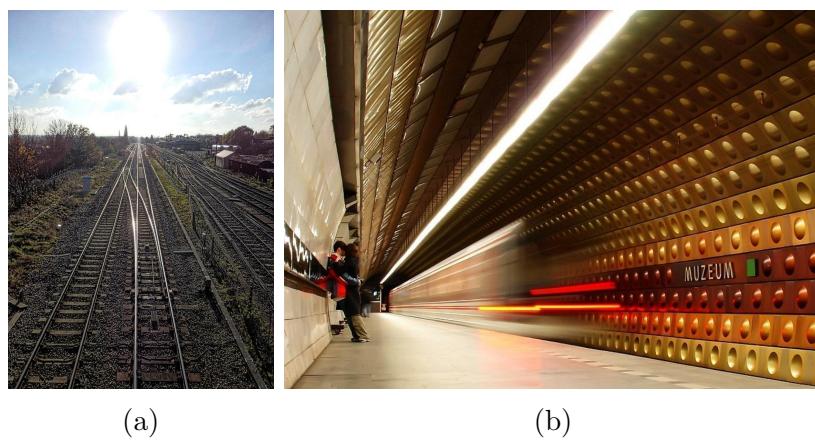


Figura 1.3: Esempio di (a) leading lines convergenti, corrispondenti con l'unirsi dei binari all'orizzonte (b) leading lines diagonali.

1.1.3 Forme e patterns

Un'altra tecnica di composizione dello scatto è quella relativa alla forma dei soggetti e la presenza di *patterns*. Posizionare la camera in modo da esaltare la **ripetizione di oggetti di struttura simile** permette di creare effetti visivi interessanti, accattivanti, di forte impatto (Figura 1.4b.). Anche effettuare lo scatto da angoli appositi che **accentuino delle forme** ben precise, come quella triangolare, possono dare più personalità all'immagine (Figura 1.4a.).

1.2 Classificare la composizione

I fotografi professionisti sfruttano elementi geometrici basilari come punti, linee, forme, per guidarli nella composizione delle immagini. Nonostante ciò, a causa della grande varietà di disposizioni, forme, colori dei soggetti, molti diversi elementi geometrici appaiono nelle fotografie. Per questo motivo, **determinare la regola di composizione**



Figura 1.4: Esempi di (a) un soggetto triangolare (b) un pattern

rilevando direttamente gli elementi geometrici dell’immagine non è un approccio realistico. È necessario avere una comprensione delle foto nella loro interezza per poterne determinare la composizione in maniera affidabile.

Ci sono stati approcci [27] che utilizzano algoritmi convenzionali nell’individuare la classe di composizione delle immagini, sfruttando approcci che determinano quanto siano “ovvi” gli elementi di geometria e di colore nell’immagine, quanto siano prevalenti nel percepire lo scatto. Questo, però, risulta insufficiente nel classificare la composizione: un soggetto umano considera l’importanza contestuale di ciascun elemento tanto quanto la sua rilevanza all’interno dell’immagine. Ad esempio, nella Figura 1.5, l’elemento che risulta essere più ovvio studiando caratteristiche di basso livello, come l’intensità di colore, è lo stacco fra il cielo e il terreno, l’orizzonte. Nonostante ciò, guardando l’immagine, l’elemento che a un utente umano salta immediatamente all’occhio è la staccionata, che può coincidere con una leading line curva, e quindi classificare lo scatto con tale composizione.

Per questo motivo, l’approccio preferenziale è quello di imparare features discriminanti a partire da un grande numero di immagini annotate da esseri umani, piuttosto che delle caratteristiche studiate manualmente, ad hoc. Di conseguenza, negli esperimenti condotti durante lo stage che verranno esposti successivamente, sono stati utilizzate tecniche di *deep learning* sui datasets KU-PCP [13] e LODB [16], le cui origini e caratteristiche saranno approfondite nei Capitoli 2 e 5.



Figura 1.5: Le leading lines curve implicite date dalla forma della staccionata risultano più importanti all’occhio umano rispetto alla separazione fra cielo e terreno.



Figura 1.6: Immagini classificabili con più tipi di composizione: (a) horizontal leading line e pattern, (b) curva e pattern, (c) verticale e RoT

Inoltre, per il task di classificazione, in questa relazione si utilizzeranno features prodotte tramite un modello addestrato con la tecnica della **self-supervision**. Negli ultimi anni, specialmente dopo il 2020, c'è stata una forte accelerazione nella ricerca riguardante i metodi di *self-supervised learning*, che ha portato alla nascita di DINOv2 [18], un foundational model di MetaAI che ha prodotto risultati molti promettenti nell'ambito della classificazione della semantica delle immagini, e non solo. Le sue caratteristiche saranno meglio esplorate nel Capitolo 3. L'analisi proposta nasce dalla curiosità di valutare le prestazioni di DINOv2 sul task presentato e verificare se le features che esso estrae possano essere utilizzate per problemi più di alto livello rispetto al riconoscimento dei soggetti, come appunto l'analisi degli elementi geometrici e spaziali nell'immagine.

Da notare è che quello della composizione è un problema di classificazione *multilabel*, una foto potrebbe presentare l'utilizzo di più tecniche compositive, come illustrato nella Figura 1.6.

1.3 Scopo e utilizzi pratici

Nonostante l'importanza che la composizione fotografica detiene nel migliorare e nel determinare la qualità estetica di uno scatto, questo è un aspetto che raramente viene considerato, soprattutto da parte di fotografi amatoriali e inesperti. È per questo motivo che la creazione di tools che possano assistere l'utente nel compito di comporre abilmente gli scatti potrebbe risultare utile. Pensiamo alle app Fotocamera dei nostri smartphones: e se queste avessero la capacità di guidarci nel produrre scatti migliori oltre il semplice allineamento dell'orizzonte? L'arte fotografica sarebbe molto più accessibile anche per chi non ha necessariamente le basi tecniche che ha un veterano nel settore.

Uno strumento utile in ambito istruttivo per un giovane fotografo potrebbe essere quello di sottoporre ad un sistema automatizzato una immagine statica, già scattata, allo scopo di ottenere un insieme di dettagli e tecniche utilizzate per produrla, da cui imparare. Alternativamente, potrebbe essere usato come fonte di critica dei propri scatti e come punto di riflessione su come poterli migliorare, anche per chi ha già esperienza nel campo.

Un altro spunto potrebbe essere quello di introdurre questo tipo di aiuto in software di editing fotografico (*Photoshop*, ma anche video editing come *Adobe Premiere*), in modo che possa guidare operazioni come il *cropping* o la rotazione di un'immagine, suggeren-

do quale sia l'approccio migliore per ottenere un risultato che rispetti le linee guida di composizione, per una qualità estetica migliorata.

Oltre all'aspetto prettamente fotografico, potremmo anche pensare di estendere questo studio alla grafica e al design. Sarebbe importante l'introduzione di uno strumento che assista il designer nell'impaginazione degli elementi di una grafica, seguendo le stesse regole di leading lines, RoT, forme, patterns, e altri. In ambito di marketing potrebbe aiutare nel suggerire come guidare l'attenzione degli utenti verso gli elementi che si vogliono evidenziare in una inserzione pubblicitaria.

In ambito ricreativo, un tool come quello discusso potrebbe aiutare l'artista a meglio comprendere come trasmettere determinate sensazioni tramite le sue creazioni, esponendogli quale sia la migliore costruzione dell'immagine in base al sentimento che egli vuole ritrarre. Facendo un rapido riferimento ai miei interessi personali, la parte più complicata del produrre un render tramite Blender non è sempre il modellare la scena nei suoi aspetti più tecnici e geometrici, come si potrebbe pensare, bensì il come posizionare la camera nella scena 3D in modo che il risultato finale sia armonico e valorizzi la propria visione artistica.

Un primo passo nel raggiungere lo sviluppo di software che possano guidare l'esperienza del fotografo, dell'artista, del designer, è quello di essere in grado di riconoscere gli elementi compositivi degli scatti, che portano una grande influenza sul valore estetico, e non solo, delle immagini in tutti i settori menzionati.

1.4 Outline

Questa relazione si concentrerà su uno studio dello stato dell'arte nell'ambito della classificazione della composizione delle immagini, nel Capitolo 2. Successivamente si presenterà una panoramica sull'evoluzione delle tecniche di self-supervision che hanno portato alla nascita di DINOv2 nel Capitolo 3. Nel Capitolo 4 si descriverà il metodo utilizzato per ottenere i risultati esposti nel Capitolo 6. La descrizione dei dataset usati si trova al Capitolo 5. Infine, nel Capitolo 7 si faranno delle considerazioni finali sui risultati prodotti e su possibili sviluppi e altre strade da esplorare in futuro.

2

Stato dell'arte e lavori correlati

In questo capitolo si discuteranno brevemente solo alcuni degli approcci più rilevanti al task di *image layout classification*. Quello della composizione delle immagini non è un problema che viene spesso affrontato in ambito di ricerca, di conseguenza le risorse ad esso relative sono in numero limitato. Ciononostante, ci sono stati diversi tentativi nel corso degli anni di migliorare sempre più lo stato dell'arte.

2.1 Prima del Deep Learning

Prima di [14], di cui si discuterà nella sezione successiva, non esistevano metodi automatici e affidabili per il task di classificazione della composizione, a causa della sua natura fortemente soggettiva e talvolta ambigua. Approcci precedenti si basavano solamente su un insieme limitato di aspetti della composizione, come il bilanciamento del peso visivo e la regola dei terzi, o sfruttavano procedure di clustering non supervisionato, senza però definire regole di composizione specifiche. In particolare, per riconoscere la RoT, alcuni metodi lavoravano localizzando il soggetto dell'immagine e calcolando la distanza fra il suo centroide e i punti di intersezione delle linee orizzontali con quelle verticali che formano la griglia (vedi Figura 1.1).

Solamente [27] ha provato a definire un metodo per categorizzare le regole di composizione in una più ampia varietà di classi, 8 in totale, considerando 25 criteri stabiliti sulla base di caratteristiche geometriche calcolate su ciascuna immagine. Questo approccio, però, è realizzato su un piccolo dataset di 80 fotografie e di conseguenza può produrre risultati inaffidabili su insiemi più ampi di immagini. Essendo regole costruite ad hoc per quel ridotto insieme di immagini, queste non possono coprire tutti i possibili casi di configurazioni presenti in un dataset più esteso. Inoltre, questo metodo non permette di classificare più classi di composizione all'interno della stessa immagine, o fatica nel farlo.

Altri approcci suddividono le foto in *patches*, piccole regioni della stessa dimensione, e ne studiano poi il loro ordinamento all'interno dell'immagine e li utilizzano per localizzare soggetti localmente. Tecniche basate sulla valutazione di patches potrebbero però non

tenere conto correttamente dei confini fra soggetti e ancora una volta faticano a valutare le immagini nel loro insieme.

Ci sono stati tentativi di suddividere le immagini tramite algoritmi di segmentazione e successivamente descrivere la composizione tramite le distribuzioni di colore o texture fra regioni, caratterizzandone la posizione nella foto, ma questo non basta a indurne la composizione globale.

2.2 Fine-tuning di una CNN

Il paper *"Photographic composition classification and dominant geometric element detection for outdoor scenes"* [14] del 2017 è la prima istanza di uno studio completo e strutturato della classificazione della composizione delle immagini tramite *deep learning*.

Gli autori del paper utilizzano una CNN *pre-trained* per imparare delle caratteristiche compositive discriminanti da un ampio insieme di immagini *human-annotated*. L'architettura della rete utilizzata è esposta in Figura 2.1. L'allenamento preliminare della ResNet è fatto sul dataset ImageNet [10], formato da 1000 classi, per evitare problemi di overfitting. Solo successivamente si fa un fine-tuning sul dataset KU-PCP, introdotto dal paper, e di cui si discuterà nel dettaglio nella Sezione 5.1. KU-PCP diventerà di fatto il dataset standard per il task di image layout classification.

Il layer di *softmax* prende in input solamente una classe alla volta da considerarsi come ground truth, quindi per affrontare il problema della classificazione multi-label, le immagini che possiedono classi multiple vengono fatte passare nella rete tante volte quante sono le classi di ground truth che possiedono.

La rete produce 9 *confidence scores*, in numero pari alla molteplicità delle classi di composition nel dataset KU-PCP. Successivamente, ogni immagine viene categorizzata con la classe dallo score più alto. Nel caso ci sia un'altra classe che produce un punteggio superiore all'80% di quello della più alta, allora anche questa viene considerata come risultato della classificazione.

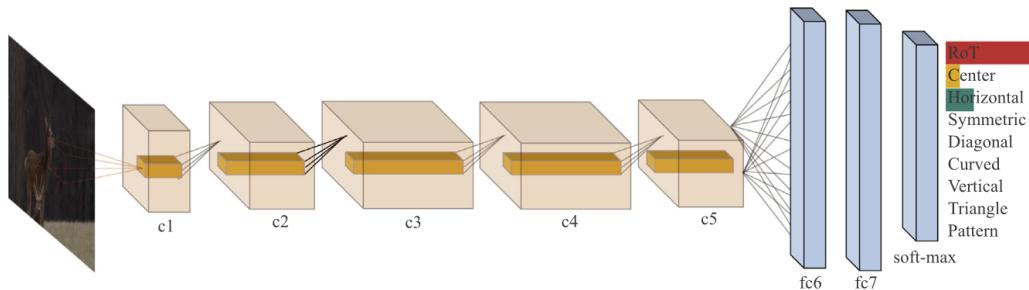


Figura 2.1: L'architettura proposta da [14], composta di cinque layer convoluzionali (c1, c2, c3, c4, c5) e tre *fully connected layers* (fc6, fc7, soft-max). Il layer di soft-max produce i punteggi di confidenza per le 9 classi contenute nel dataset KU-PCP.

2.3 Spatial-invariant CNN

Nel paper *"Spatial-invariant convolutional neural network for photographic composition prediction and automatic correction"* [26] si sostiene che il metodo precedentemente discusso sia indebolito dal fatto che il dataset KU-PCP contenga solamente fotografie pro-

fessionali (uno dei suoi difetti, di cui si discuterà in Sezione 5.2), e che questo crei condizioni "troppo perfette" per il riconoscimento della composizione. Non si tiene conto delle *snapshots*, fotografie scattate rapidamente, con uno smartphone, che possono presentare dei difetti come il non essere perfettamente allineate con l'orizzonte, o avere altre leggere imperfezioni, che caratterizzano la maggior parte delle fotografie che realizziamo nella vita quotidiana. Inoltre, gli autori sostengono che leggere variazioni spaziali, entro un determinato range, non influenzano la capacità di un umano di riconoscere la struttura compositiva dell'immagine (Figura 2.3).

Viene proposta quindi una *spatial-invariant* convolution neural network composta da una architettura denominata Rotation-Shift Transformer Network (RSTN)

Il processo di classificazione proposto è il seguente:

1. **Augmentations:** a partire dalle immagini del dataset KU-PCP, si effettua una rotazione casuale nel range $[-10, +10]^\circ$, che si è osservato non influire sulla percezione della composizione tanto da modificarne la tipologia riconosciuta dagli utenti.
2. **Feature extraction:** si utilizza una ResNet pre-trained su ImageNet per estrarre una *feature map* da utilizzare come descrittore dell'immagine in input nella parte di rete successiva, adibita al riconoscimento della distorsione applicata all'immagine.
3. **Rotation Shift Transformer:** la sua struttura è ispirata dalle *Spatial Transformer Networks* [11]. Si compone di:
 - **Localization Net:** usa 2 layer convoluzionali e 2 fully connected layers per imparare tre parametri: l'angolo di rotazione θ e i coefficienti di traslazione c, f .
 - **Grid generator:** tramite i parametri prodotti dai layer convoluzionali precedenti, si può creare la matrice di trasformazione:

$$M = \begin{bmatrix} \cos \theta & -\sin \theta & c \\ \sin \theta & \cos(\theta) & f \\ 0 & 0 & 1 \end{bmatrix}$$

- **Sampler:** effettuando il prodotto matriciale $M \begin{pmatrix} x_i \\ y_i \\ 1 \end{pmatrix}$, dove (x_i, y_i) sono le coordinate del pixel i nell'immagine augmentata, si può ricostruire l'immagine

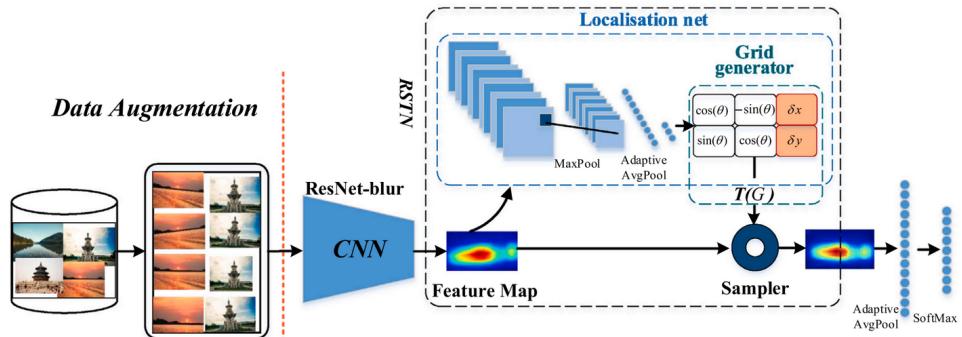


Figura 2.2: Framework per la spatial-invariant convolution neural network.

originale. In questo caso, la moltiplicazione viene fatta direttamente con i pixels della feature map prodotta dalla convoluzione della ResNet.

A questo punto si può passare la feature map ripristinata a un layer di pooling e un softmax, che produce i nove scores di confidenza per la classificazione della composizione.

Questo permette alla rete di imparare a trovare delle trasformazioni che possano correggere le piccole imperfezioni che caratterizzano foto non professionali, snapshots, e di conseguenza classificare le classi di composizione in maniera più robusta.

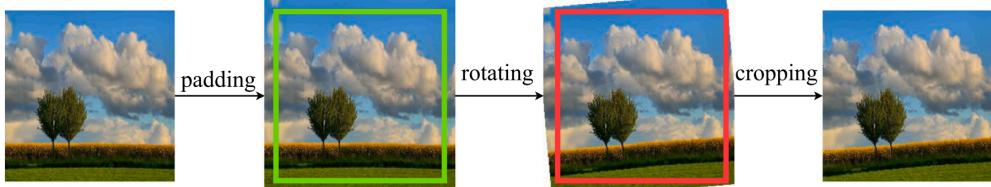


Figura 2.3: La rotazione dell’immagine viene effettuata introducendo del padding speculare intorno all’immagine, ruotando il risultato e croppando per tornare alle dimensioni originali. Gli autori osservano come, se la classe iniziale dell’immagine è quella di una leading line orizzontale data dall’orizzonte, distorcere l’immagine con una leggera rotazione non cambia il nostro giudizio sulla composizione, la percepiamo ancora come classe orizzontale riconoscendone il contesto.

2.4 Grafi come descrittori del layout

Recentemente, nel marzo del 2024, è stato pubblicato il paper *”Self-supervised Photographic Image Layout Representation Learning”* [28]. Gli autori sostengono che grandi problemi di approcci supervisionati nell’ambito della classificazione dell’image layout e task correlati siano il dipendere da datasets costosamente etichettati e la mancanza di adattabilità dei modelli nell’imparare le sfumature più fini della composizione. Per questo motivo si decide di cambiare completamente approccio, cioè di utilizzare la self-supervision, che non necessita di labels.

La novità da Zhao et al. è quella di individuare nelle immagini delle primitive di base che incapsulino l’informazione di layout a diversi livelli, e di mapparle su una struttura a **grafo eterogeneo**. Dopodichè, si introduce un *pretext task* ad hoc, in coppia con una funzione di loss apposita, progettati per poter permettere l’apprendimento autonomo. Con grafo eterogeneo si intende un grafo i cui archi e nodi possono appartenere a domini semantici diversi. I dettagli su self-supervision e pretext task verranno approfonditi nel prossimo capitolo. Stabiliti questi principi, gli autori introducono una architettura di rete in grado di comprimere questi grafi descrittori di ogni immagine in una rappresentazione vettoriale di dimensionalità ridotta, da usare poi per la classificazione.

Un altro importante contributo del paper in oggetto è l’introduzione del dataset **LODB** [16], costruito sulla base di KU-PCP e nato con l’obiettivo di compensare ai più grandi difetti di quest’ultimo. Presenta una più ampia gamma di categorie di layout (17 classi, contro le 9 di KU-PCP) e più semanticamente ricche. I dettagli verranno discussi nella Sezione 5.2.

Di seguito una breve spiegazione delle componenti del workflow. A partire da una immagine χ , i grafi coinvolti sono due:

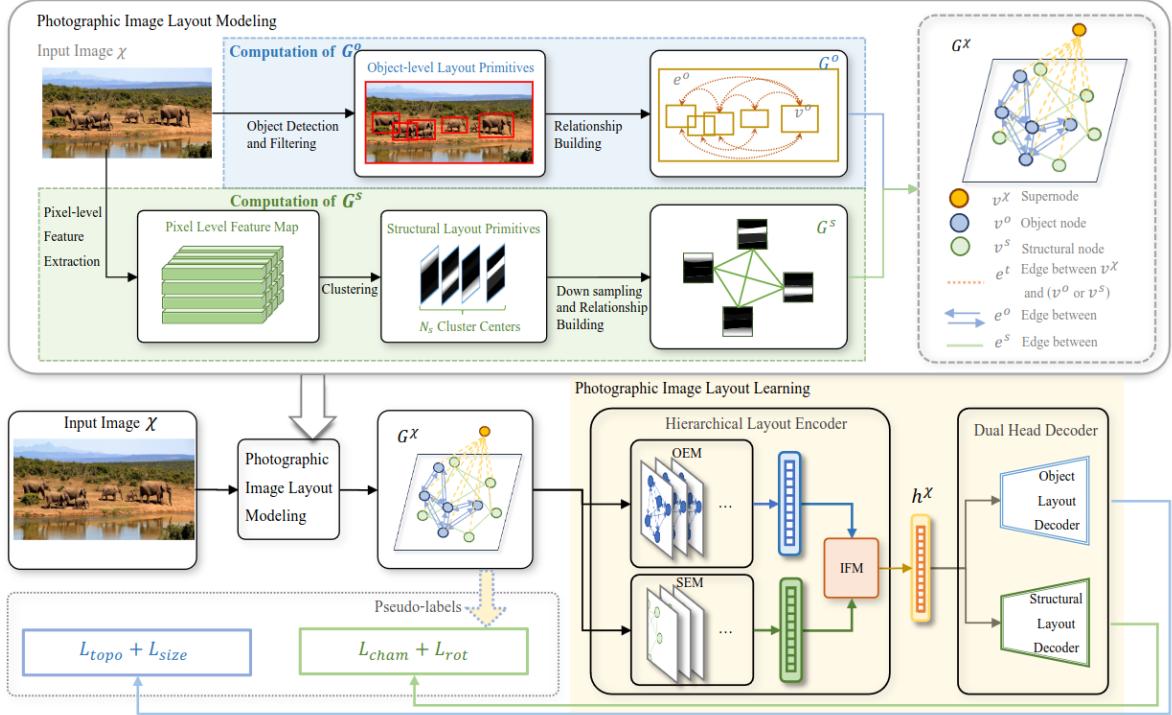


Figura 2.4: Workflow seguito da [28].

- Grafo G^s , relativo alla struttura del layout. Secondo la teoria della Gestalt, primitive visivamente simili o in prossimità nell'immagine vengono percepite come un'unica entità. Per questo motivo, si utilizza una rete pre-allenata come feature extractor da ogni pixel dell'immagine e si fa del clustering sulle feature maps prodotte. Si costruisce il grafo G^s avendo come nodi i clusters individuati, mentre come archi le relazioni di co-occorrenza fra nodi, cioè quanto spesso le primitive strutturali appaiono insieme o in vicinanza all'interno della foto.
- Grafo G^o , relativo agli oggetti nell'immagine. Le immagini presentano generalmente uno o più oggetti salienti, che possono essere identificati tramite le rispettive *bounding boxes*, le cui caratteristiche (dimensione, posizione) costituiranno i nodi del grafo. Gli archi codificano aspetti come le dimensioni relative fra due boxes, la distanza fra loro.

Senza scendere nei dettagli relativi alla funzione di loss utilizzata, si allena la rete in maniera self-supervised a imparare le caratteristiche di V^s , l'insieme dei nodi del primo grafo, utilizzando come informazioni E^s , gli archi dello stesso. Allo stesso modo, si introduce un secondo pretext task volto alla ricostruzione delle relazioni topologiche fra bounding boxes e i relativi centri, e un terzo task dedito ad effettuare regressione sulla dimensione delle boxes. Questo permette alla rete di imparare features molto più espresive dalle immagini che riceve, senza la necessità di labels, godendo di tutti i vantaggi che la self-supervision porta (approfondito in Capitolo 3). La procedura viene realizzata tramite un encoder costituito da una *Graph Attention Network*, argomento che non è stato approfondito nel corso di questo stage.

L'aspetto più importante da trarre da questo studio, oltre l'introduzione di LODB, è la nascita di metodi self-supervised volti alla classificazione della composizione, che sono in grado di imparare features espresive ad essa relative, e soprattutto in maniera

autonoma. Questo rende ancora più rilevante un confronto coi metodi di self-supervised learning già stabiliti e di successo, come DINOv2.

3

Self-Supervised Learning

Il *self-supervised learning* (SSL) è una delle strade attualmente più promettenti nell'avanzare la ricerca in ambito di machine learning. Al contrario dell'apprendimento supervisionato, che è spesso fortemente limitato dalla necessità di creare appositi datasets etichettati, la caratteristica principale della SSL è la capacità di **apprendere** da grandi quantità di **dati non etichettati**. Metodi di SSL in computer vision sono stati in grado di pareggiare, e in alcuni casi anche superare modelli allenati tradizionalmente.

Nel corso di questo capitolo si esporranno le principali tecniche e architetture che hanno portato alla nascita di DINOv2, il modello utilizzato nella fase di sperimentazione (Capitolo 6). Tutti i modelli che verranno esposti vengono sviluppati con obiettivo principale quello della classificazione semantica del contenuto delle immagini, utilizzato come task volto alla creazione di features utilizzabili in altri problemi più specifici.

Esiste una risorsa che raccoglie l'evoluzione e gli aspetti più importanti della self-supervision: *"A Cookbook of Self-Supervised Learning"* [1] di MetaAI.

Il capitolo non è essenziale ai fini della comprensione della fase sperimentale che segue, se si considera DINOv2 come un semplice feature extractor. Nonostante ciò, è stato incluso perchè contestualizza la nascita del modello e ne spiega il funzionamento interno, oltre che presentare una panoramica dello studio che è stato condotto durante il periodo di stage.

3.1 Cos'è il self-supervised learning

La self-supervision consiste nella definizione di un *pretext task*, basato su input **non etichettati**, il cui scopo è la produzione di **rappresentazioni** degli stessi che siano **descrittive e intelligibili**. Un pretext task è un obiettivo di pre-training, costruito in base al problema, che viene stabilito per guidare il modello nell'apprendimento di tali rappresentazioni. In ambito di Natural Language Processing, ad esempio, un obiettivo comune è quello di mascherare una parola nel testo e predire il contesto che la circonda. Questo incoraggia il modello a catturare relazioni fra parole nel testo senza avere bisogno di alcuna label.

Lo scopo ultimo di questi approcci è produrre delle features *general purpose*, utili in una vasta gamma di task. L'ideale è quello di giungere allo sviluppo di *Foundation Models* (FMs): piuttosto che creare da zero delle AI per ciascun task che si desidera risolvere, si vorrebbero introdurre dei modelli allenati su un grande range di dati generali e non etichettati che siano capaci di risolvere compiti disparati con un alto livello di accuratezza. GPT-4 può essere considerato un Foundation Model.

Tutti i modelli che si discuteranno in questo capitolo sono in realtà delle **tecniche di allenamento** di diverse tipologie di *encoders* (ResNet, Transformer, Vision Transformer). Nel momento in cui la fase di training self-supervised termina, **si mantiene solamente l'encoder** da utilizzare come *feature extractor* per altri task più specifici, come quello della classificazione della composizione delle immagini.

3.1.1 Pro e contro

Alcuni dei vantaggi della self-supervision rispetto ad approcci supervisionati sono:

- Apprendimento di rappresentazioni generiche, utili per una ampia gamma di problemi.
- Utilizzo di dati non etichettati:
 - La **disponibilità** di **dataset etichettati** è **scarsa** in numero a causa dei tempi di produzione richiesti e costi coinvolti.
 - L'introduzione di etichette in un dataset **richiede** la presenza di **figure professionali** esperte nel settore, che possano correttamente categorizzare i dati coinvolti. Se pensiamo alla creazione di datasets in ambito medico, è inevitabile l'intervento di uno specialista nel definire correttamente le classi di interesse e come assegnarle ai dati. Questo rende datasets etichettati ancora più costosi da produrre.
 - La quantità di dati esistenti e direttamente accessibili online supera di gran lunga il numero di etichette per gli stessi. Di conseguenza, non avere bisogno di labels permette di **scalare** notevolmente i propri approcci e allenare su decine, se non centinaia, di milioni di immagini.
- Imparare in autonomia rende i modelli più robusti ad *adversarial examples*, esempi progettati appositamente per ingannare i modelli, e al rumore spesso presente all'interno dei datasets, dovuto in parte anche ad etichettamenti errati o ambigui.

Alcuni dei suoi svantaggi:

- Il consumo di risorse di tempo ed energia è elevato per l'allenamento di modelli che usano SSL.
- Sono richiesti enormi quantità di dati per poter raggiungere valori di accuracy paragonabili alle controparti supervisionate.
- Quello della self-supervision è un campo che si basa molto sull'**evidenza empirica**, essendo ancora in piena fase di sviluppo e ricerca. Spesso vengono presentati risultati in cui si può solo ipotizzare la ragione del successo o insuccesso di un modello, senza conferme provenienti da uno studio teorico.

3.2 Augmentations

Gli approcci di SSL si basano in maniera importante sul concetto di *augmentation*. Data una immagine X , con il termine augmentation si indica una versione di X a cui sono state **applicate delle trasformazioni**, fra le quali (Figura 3.1):

- Cropping
- Resizing
- Rotazione
- Flipping
- Introduzione di rumore
- Introduzione di *blur gaussiano*
- Estrazione dell'immagine degli edge tramite filtro di *Sobel*
- Diversi tipi di distorsione dei suoi colori
 - Trasformare l'immagine da RGB a scala di grigi
 - Modifiche nella brightness, contrasto, saturazione, tinta
 - Solarizzazione: inversione del colore di tutti i pixels che hanno un valore sopra una determinata soglia.

Quando successivamente si farà riferimento al termine *view* (o vista) di X si intenderà una augmentation di X .

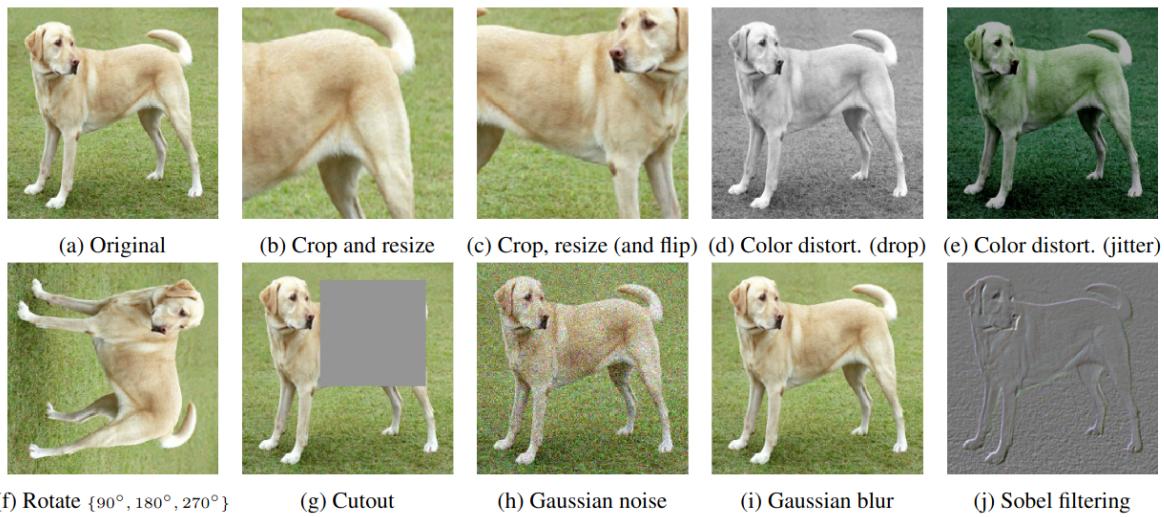


Figura 3.1: Un esempio di alcune delle augmentations più utilizzate in ambito di SSL.

3.3 Contrastive Learning

Il *contrastive learning* è una delle tecniche che stanno alla base della self-supervision.

I metodi supervisionati per i task di classificazione si basano sul confronto fra la classe predetta dal modello e quella reale assegnata al dato, che viene tradotto poi in una funzione di loss che permette al modello di imparare. Nel caso della self-supervision, questo confronto non può avvenire, non esistendo una ground truth. Si devono trovare altri modi per costringere il modello ad apprendere.

Per fare ciò, il contrastive learning introduce il concetto di coppie positive e negative. Data una immagine X :

- **Coppia positiva:** un insieme di due *views* o augmentations di X , quindi generate a partire dalla stessa immagine.
- **Coppia negativa:** una insieme di due immagini non in relazione fra loro, augmentations non provenienti entrambe da X .

L'idea è la seguente. Dato un dataset di immagini, si produce un insieme di augmentations delle stesse. Dopodichè, si formano coppie positive e negative fra le views generate e si allena il modello a differenziare fra le due tipologie, basandosi su determinati criteri di similarità scelti. Gli obiettivi sono:

- **Massimizzare la similitudine fra le rappresentazioni** di immagini provenienti da **copie positive**: siccome si tratta di due varianti di una stessa immagine, vogliamo che abbiano vettori di features più vicini possibile.
- Massimizzare la distanza fra rappresentazioni in coppie negative, essendo composte da due immagini non correlate fra loro.

In questo modo, possiamo stabilire un metro di confronto fra immagini su cui fare *back-propagation* e apprendere, senza avere il bisogno di introdurre labels.

3.3.1 SimCLR

Uno dei modelli più importanti nell'utilizzo della tecnica contrastive è SimCLR ([3]), introdotto nel 2020. Di seguito il suo funzionamento.

Durante la fase di training viene scelto casualmente un *minibatch* di N immagini di esempio dal dataset. Successivamente, per ogni immagine, vengono scelte casualmente due tipologie di augmentation, da un pool di candidati, e vengono applicate. Il risultato sono N coppie positive e $2(N - 1)$ coppie negative. Le coppie di augmentation (x_i, x_j) vengono passate all'interno della rete.

L'architettura si compone di un *encoder* f , cioè una ResNet [9] col compito di estrarre una rappresentazione h dalle immagini, e un projector g , formato da un MLP con 1 layer nascosto e la funzione di attivazione ReLU. Il risultato finale sono due rappresentazioni (z_i, z_j) . Il projector viene aggiunto perché si è verificato sperimentalmente che migliora la performance e la qualità delle features estratte.

A questo punto, per massimizzare l'*agreement* o la *discordanza*, la similitudine fra rappresentazioni, viene utilizzata una apposita funzione di *loss contrastiva*.

$$\ell_{i,j} = -\log \frac{\exp(\text{sim}(z_i, z_j)/\tau)}{\sum_{k=1}^{2N} \mathbb{1}_{k \neq i} \exp(\text{sim}(z_i, z_k)/\tau)} \quad (3.1)$$

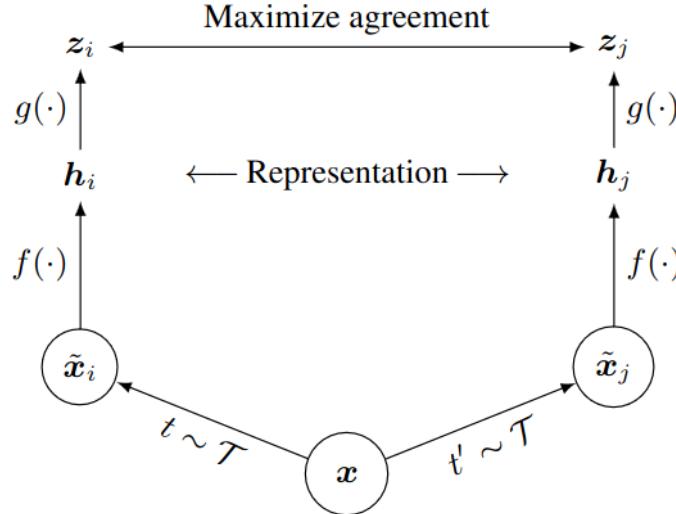


Figura 3.2: Architettura di SimCLR. Questa architettura viene anche detta *siamese*, perché composta da 2 sotto-reti che condividono architettura e pesi. L’immagine mostra un esempio per coppie positive.

dove

$$\text{sim}(u, v) = \frac{u^T v}{\|u\| \|v\|} \quad (3.2)$$

è la *cosine similarity* fra i vettori di rappresentazioni, la misura di similitudine scelta per modulare l’agreement, e

$$\mathbb{1}_{[k \neq i]} = \begin{cases} 1 & \text{iff } k \neq i \\ 0 & \text{else} \end{cases} \quad (3.3)$$

La loss risulta bassa quando le rappresentazioni delle coppie positive sono simili e quelle delle coppie negative distanti, mentre è alta nella situazione inversa. Questo permette alla rete di imparare correttamente ad aumentare o diminuire le distanze fra le rappresentazioni in base al tipo di coppia, e di conseguenza di imparare delle features che siano robuste, espressive, generalizzanti del contenuto di una immagine, a prescindere dai suoi dettagli più fini di colore, posizione del soggetto, rotazione, e così via.

3.3.2 Problemi col contrastive learning

Il contrastive learning, concepito come SimCLR lo implementa, presenta alcuni problemi, soprattutto quello del trattamento riservato alle coppie negative:

- Se le coppie negative sono troppo simili a quelle positive, il rischio è che le rappresentazioni imparate non siano sufficientemente discriminanti, perché il modello non ha la possibilità di migliorare la propria comprensione di cosa è qualificato come coppia negativa, quale sia la differenza fra le due.
- Allo stesso modo, se le coppie negative sono troppo dissimili dai campioni positivi, il pericolo è che si imparino a distinguere solamente le differenze macroscopiche fra due immagini, senza cogliere dettagli piccoli ma rilevanti ai fini della classificazione.



Figura 3.3: Valori di accuracy di SimCLR al variare delle augmentations scelte per produrre coppie positive e negative. Si nota quanto diverse combinazioni influiscano sulle performance della rete.

- È richiesto un bilanciamento numerico fra i due tipi di coppie. Come visto in SimCLR, producendo due viste per immagine il risultato è quello di N coppie positive contro $2(N - 1)$ negative. Questo potrebbe non creare le condizioni adatte affinché si possano imparare features adeguatamente dettagliate, il modello sarebbe in grado di distinguere immagini diverse, ma non di identificare immagini simili. Sarebbe un *overfitting* dei campioni negativi.

Per mitigare questi problemi, molti modelli utilizzano strategie quali l'aumentare la dimensione dei batch, utilizzare delle memory banks che memorizzino rappresentazioni di coppie negative da mini-batch precedenti, per diversificare i campioni senza dover necessariamente ripassarli nell'encoder. Altri utilizzano tecniche di *mining* apposite per recuperare esempi negativi migliori.

Un altro aspetto critico è quello della scelta delle augmentations usate. Le performance possono variare drasticamente in base a quali vengono impiegate (Figura 3.3).

3.4 Collasso

L'idea di base del contrastive learning è efficace, ma come visto, le coppie negative portano con sé diverse criticità a livello gestionale e di performance. Per questo motivo, un corso d'azione che potrebbe risultare naturale seguire è quello di **utilizzare solamente campioni positivi**. Il problema di questo approccio, però, è che ignorare coppie negative può portare a **solutions banali**: se la rete dovesse imparare la stessa rappresentazione per tutte le immagini che riceve, la loss sarebbe minimizzata, l'obiettivo di massimizzazione della similitudine raggiunto e l'apprendimento interrotto. Questo fenomeno prende il nome di *collasso*, cioè quando una rete **converge a rappresentazioni non significative**, come un vettore di features dai valori costanti per tutti i samples analizzati.

Secondo il paper *"Understanding dimensional collapse in contrastive self-supervised learning"* [12] esistono due tipologie di collasso, illustrate in Figura 3.4:

- **Collasso dimensionale**: il vettore di features collassa in uno spazio dimensionale ridotto.

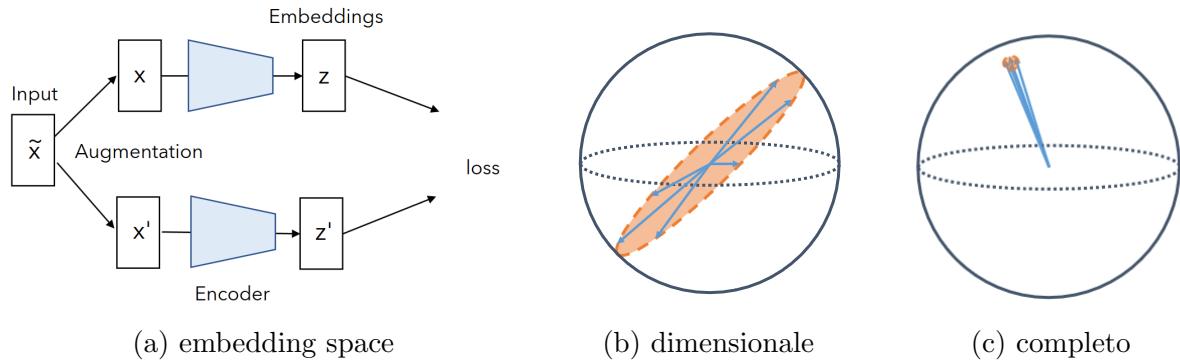


Figura 3.4: Visualizzazione dello spazio delle features nel caso di collasso dimensionale o completo.

- **Collasso completo:** il vettore di features collassa nell'intorno di un solo punto nello spazio delle rappresentazioni.

Sono nati quindi diversi approcci per contrastare questa problematica, prevalente in tutti i modelli basati su self-supervision, e allo stesso tempo condurre un addestramento che non richieda di coppie negative. Una di queste tecniche è quella di *self-distillation*, su cui si basano tutte le architetture di cui si discuterà, compreso DINOv2.

3.5 Self-Distillation

Il concetto di self-distillation deriva a sua volta da un altro concetto, quello di *knowledge distillation*.

La distillation (Figura 3.5) è una tecnica utilizzata in deep learning che permette di **trasferire la conoscenza appresa** da un modello pre-allenato di grandi dimensioni, detto *Teacher*, ad un modello di dimensioni ridotte e ancora da allenare, detto *Student*. In pratica, lo **student** deve imparare a **riprodurre le distribuzioni di probabilità** prodotte **del teacher** relativamente alle classi predette, a fronte dello stesso dato in input (immagini nel nostro caso). L'obiettivo di questa tecnica è ottenere un modello più piccolo, ma dalle performance comparabili a quelle del Teacher, con minori requisiti computazionali e di memoria. Questo aspetto è particolarmente utile per importare reti su dispositivi con risorse limitate, come smartphones o dispositivi IoT.

In ambito di self-supervision, non è tanto l'aspetto relativo alle risorse a interessarci, quanto piuttosto la **struttura della rete** per il trasferimento di conoscenza. L'idea è quella di introdurre **una prima rete** neurale, inizializzata casualmente, che **produca una rappresentazione target** per ciascuna view, e **una seconda rete allenata a replicarli**, data un'altra view della stessa immagine. Questi due ruoli possono essere ricoperti da un Teacher e uno Student, come nella distillation. Una volta terminato l'allenamento della rete, il Teacher viene scartato, e le features prodotte dallo Student saranno quelle definitive.

Si dimostra empiricamente che questo approccio funziona realmente nell'evitare il collasso, ma con un grande compromesso: le rappresentazioni risultanti, prodotte dalla rete Student, sono inadeguate e di bassa qualità. È un risultato prevedibile, avendo allenato una rete ad approssimare il Teacher, che è stato inizializzato casualmente e che

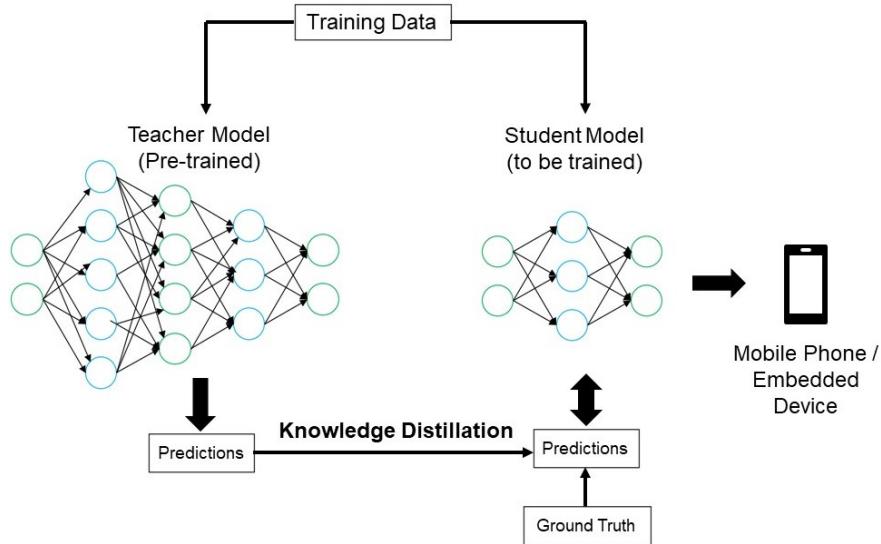


Figura 3.5: Visualizzazione del workflow utilizzato nella knowledge distillation.

non ha modo a sua volta di apprendere. Nonostante ciò, si è verificato sperimentalmente che una rete Student che replica delle rappresentazioni casuali performa meglio, in termini di accuracy nella classificazione, di una rete inizializzata casualmente (18.8% contro 1.4% su ImageNet, rispettivamente, sul modello BYOL, di cui si parlerà a breve). Questo fenomeno è evidenza del fatto che qualcosa di significativo viene effettivamente appreso, pure essendo il Teacher una rete casuale.

Per risolvere il problema del collasso e permettere allo stesso tempo allo student di imparare features di qualità, si introduce la *self-distillation*. Si ripete il processo di **distillation** più volte, fino alla convergenza della rete, imponendo come **nuovi targets** da replicare le **rappresentazioni prodotte dallo student all'iterazione precedente**. L'aspettativa è quella di costruire mano a mano delle features che siano sempre più accurate, evitando contemporaneamente il collasso. L'aggiornamento del teacher viene fatto tramite una *Exponential Moving Average* (EMA) dei pesi della rete studente. In questa configurazione, lo student viene anche detta rete *online*. Notare sempre che non è una soluzione definitiva al collasso, ma una evidenza empirica che i ricercatori hanno osservato nel costruire i loro modelli, fra cui BYOL.

3.5.1 BYOL

Bootstrap your own latent [8] è il modello che introduce l'approccio di self-distillation, che viene declinato nell'architettura in Figura 3.6.

La rete si compone di un encoder f , di un projector g , formato da un MLP con 1 layer hidden e funzione di attivazione ReLU, e predictor q , con la stessa struttura di g , ma presente solo nello student, avendo il compito di predire le rappresentazioni del teacher.

L'aggiornamento dei pesi del teacher tramite EMA dei pesi dello studente viene anche detto *Momentum Encoder*, e viene fatto come segue:

$$\xi \leftarrow \tau \xi + (1 - \tau) \theta \quad (3.4)$$

dove:

- ξ sono i pesi del teacher

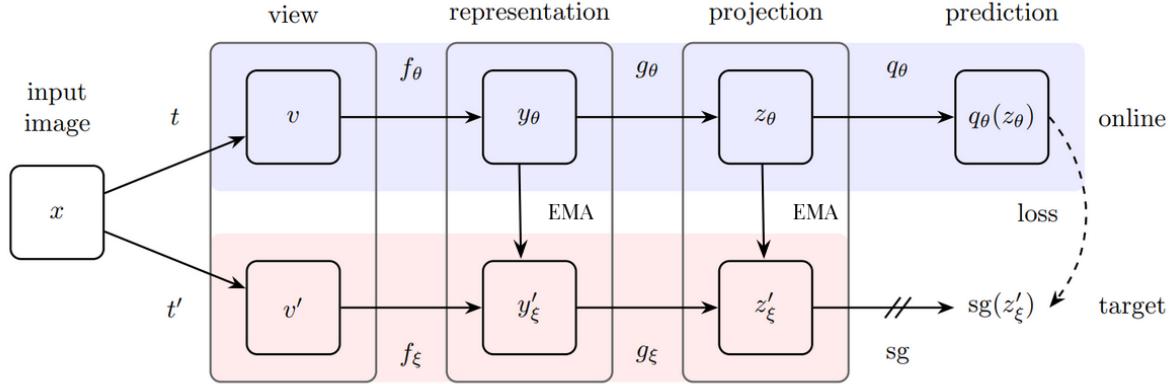


Figura 3.6: Architettura di BYOL.

- θ sono i pesi dello student
- τ è un *momentum coefficient* compreso in $[0, 1]$, che nel caso di BYOL vale inizialmente 0.996 e sale a 1 durante l'allenamento. Il risultato è che il Teacher viene aggiornato principalmente nelle prime iterazioni, fino ad arrivare al punto in cui i suoi pesi vengono fissati e l'allenamento viene fatto solo sullo Student. Lo *stop gradient* posto in cima al teacher impedisce ai pesi di essere aggiornati tramite back-propagation.

La loss utilizzata è una MSE fra la predizione dello Student e la rappresentazione corrispondente del Teacher.

$$L_{\theta,\xi} = \|\bar{q}_\theta(z_\theta) - \bar{z}'_\theta\|^2 \quad (3.5)$$

dove $\bar{q}_\theta(z_\theta)$ è il vettore di predizioni dello Student l2-normalizzato e \bar{z}'_θ è il vettore proiettato del Teacher l2-normalizzato. Questo è il calcolo per una sola coppia di augmentations, a partire da una sola immagine. Per effettuare la back-propagation si dovranno mediare tutti i risultati di loss ottenuti in un batch.

Inoltre, la loss viene simmetrizzata, come mostrato in Figura 3.7. La loss reale sarà $L_{\theta,\xi}^{BYOL} = L_{\theta,\xi} + \tilde{L}_{\theta,\xi}$. Simmetrizzare permette di avere più samples su cui fare addestramento, in maniera efficiente.

Per quanto riguarda il collasso, gli autori osservano che la combinazione di teacher e predictor effettivamente previene la problematica: rimuovere uno dei due risulta realmente nel collasso della rete.

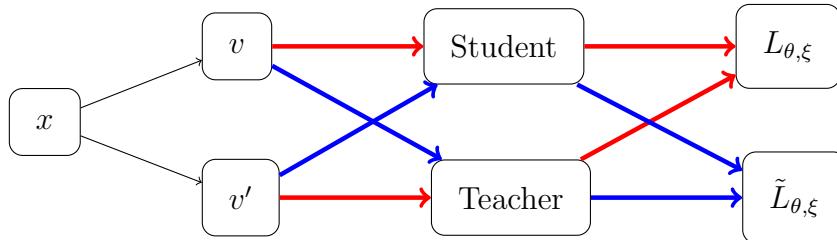


Figura 3.7: Workflow per il calcolo della loss simmetrizzata.

Method	Top-1	Top-5
Local Agg.	60.2	-
PIRL	63.6	-
CPC v2	63.8	85.3
CMC	66.2	87.0
SimCLR	69.3	89.0
MoCo v2	71.1	-
InfoMin Aug.	73.0	91.1
BYOL	74.3	91.6

Tabella 3.1: Confronto top-1 e top-5 accuracies in linear evalutation su ImageNet usando una ResNet-50 come encoder fra BYOL e altri metodi di SSL. Si nota come BYOL migliori in maniera rilevante le prestazioni di SimCLR e altri modelli concorrenti.

Un confronto con SimCLR si trova in Figura 3.1. Inoltre, gli autori verificano che BYOL è molto meno influenzato dalla dimensione dei batch e dalla scelta delle augmentations, raggiungendo l'obiettivo prefissato.

3.6 DINO

DINO (*self-distillation with no labels*) [2] è stato il tentativo da parte di Meta di **conciliare** l'evoluzione della **self-supervision**, in particolare della self-distillation, con il recente avvento dei Transformers [25] in ambito di NLP, e conseguentemente dei **Vision Transformers** (ViT). Per maggiori dettagli su queste due architetture, leggere Appendice A.

L'osservazione che fanno i creatori di DINO è che i ViT sono competitivi con le CNN, ma non mostrano chiari benefici che li rendano una scelta migliore, considerato che:

- Richiedono maggiori quantità di dati
- Sono più costosi in termini di risorse computazionali
- Le features che producono non hanno proprietà particolari. Nelle CNN sappiamo che i primi layers della rete estraggono features di basso livello, come potrebbero essere gli edges, e andando sempre più in profondità si costruiscono features progressivamente più complesse. Nei ViT non abbiamo lo stesso tipo di sicurezza nell'interpretare gli output intermedi della rete.

L'ipotesi formulata è che il **colpevole** di questa mancanza di progresso possa essere la **supervisione**, con cui i ViT sono stati allenati fino a questo punto. La supervisione riduce il ricco contenuto visuale di una immagine a un singolo concetto in un insieme predefinito di categorie, in cui lo classifichiamo. Da qui nasce l'idea di utilizzare la self-distillation, come definito precedentemente, per allenare un vision transformer in maniera self-supervised.

L'intuizione di Meta ha avuto successo, le features prodotte dal ViT **contengono esplicitamente** informazioni sul layout della scena nell'immagine, e in particolare, la **segmentazione** dei suoi oggetti (Figura 3.8). Questa informazione è direttamente accessibile tramite i **blocchi di self-attention** che caratterizzano il funzionamento di un



Figura 3.8: Mappe di self-attention estratte dal token [CLS], sulle teste del blocco di *multi-head attention* dell'ultimo layer.

transformer (Sezione A.1.4). Le features prodotte sono tanto buone da competere con altri modelli nello stato dell'arte con un semplice classificatore kNN.

3.6.1 Dettagli implementativi

Come nei modelli precedenti, a partire da una immagine x si producono delle augmentations. Nel caso di DINO si usa la tecnica del *multi-crop*. Oltre alle trasformazioni già viste nella Sezione 3.2, si introducono i concetti di:

- Vista *locale*: crop dell'immagine di dimensione inferiore al 50% rispetto a quella originale
- Vista *globale*: crop di dimensione $\geq 50\%$

Il **Teacher** riceve in input **solamente viste globali**, mentre lo studente entrambi i tipi. Questo è significativo perché, siccome il teacher conosce solamente immagini che ricoprono più di metà delle controparti originarie e siccome lo **student** impara direttamente da esso, quest'ultimo viene **costretto a imparare feature più generali** anche da crop piccoli, evitando di concentrarsi eccessivamente su dettagli fini e irrilevanti ai fini della classificazione.

L'architettura della rete è esposta in Figura 3.10. Teacher e Student si compongono entrambi di un ViT, con pesi diversi, e un projector. Rimane il momentum encoder, ma al posto del predictor, per evitare il collasso, si introducono due novità all'output del Teacher, complementari fra loro:

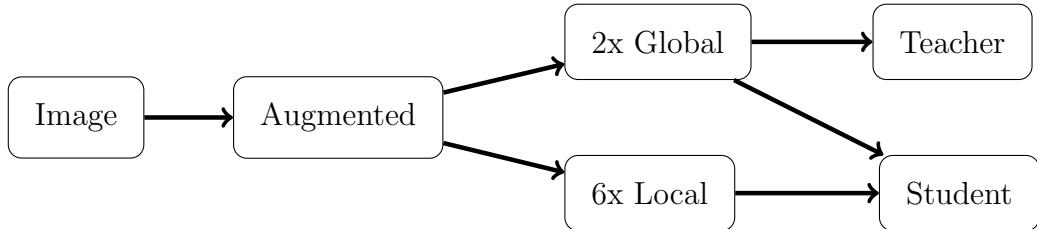


Figura 3.9: Workflow per multi-cropping. A partire da una immagine vengono prodotte 2 viste globali e 6 viste locali. In questo modo aumentano i samples senza ricalcolare altre augmentations.

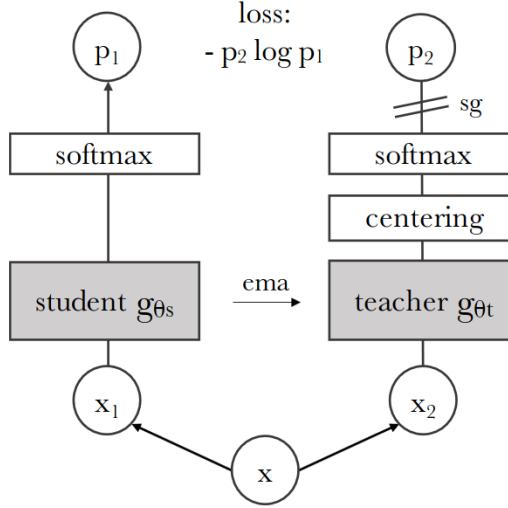


Figura 3.10: Architettura di DINO. Rete teacher e student hanno stessa architettura, ma parametri diversi, come BYOL.

- **Centering:** evita che una sola dimensione nello spazio delle features possa dominare (collazzo dimensionale), ma ha come effetto collaterale l'uniformare le rappresentazioni (collazzo completo). Viene realizzato aggiungendo un bias c a tutte le predizioni del teacher:

$$g_t \leftarrow g_t(x) + c \quad (3.6)$$

dove c viene aggiornato come una exponential moving average dei pesi del teacher

$$c \leftarrow mc + (1 - m) \frac{1}{B} \sum_{i=1}^B g_{\theta_t}(x_i) \quad (3.7)$$

con m parametro che decide il rate di aggiornamento, B è la dimensione del batch, x_i immagine nel batch.

- **Sharpening:** al contrario del centering, evita una distribuzione costante, ma può causare il dominio di una dimensione. Viene fatto utilizzando un valore di *temperatura* basso $0.04 < \tau < 0.07$ nel calcolo del softmax:

$$P_i = \frac{e^{\frac{y_i}{\tau}}}{\sum_{k=1}^n e^{\frac{y_k}{\tau}}} \quad (3.8)$$

dove y_i indica l'attivazione della classe i -esima. Un τ basso fa in modo che la distribuzione di probabilità risultante sia più decisa nelle sue predizioni, meno uniforme.

Infine, la loss utilizzata è una *cross-entropy loss*, con la quale lo Student cerca di riprodurre la distribuzione di probabilità generata dal Teacher:

$$\min_{\theta_s} \sum_{x \in \{x_1^g, x_2^g\}} \sum_{x' \in V, x' \neq x} H(P_t(x), P_s(x')) \quad (3.9)$$

con $H(a, b) = -a \log b$, x^g vista globale, V insieme di tutte le viste, a distribuzione di probabilità da replicare (Teacher), b distribuzione di probabilità predetta (Student).

Method	Arch.	Param.	im/s	Linear	k -NN
Supervised	RN50	23	1237	79.3	79.3
SCLR	RN50	23	1237	69.1	60.7
MoCov2	RN50	23	1237	71.1	61.9
InfoMin	RN50	23	1237	73.0	65.3
BarlowT	RN50	23	1237	73.2	66.0
OBoW	RN50	23	1237	73.8	61.9
BYOL	RN50	23	1237	74.4	64.8
DCv2	RN50	23	1237	75.2	67.1
SwAV	RN50	23	1237	75.3	65.7
DINO	RN50	23	1237	75.3	67.5
Supervised	ViT-S	21	1007	79.8	79.8
BYOL	ViT-S	21	1007	71.4	66.6
MoCov2	ViT-S	21	1007	72.7	64.4
SwAV	ViT-S	21	1007	73.5	66.3
DINO	ViT-S	21	1007	77.0	74.5

Tabella 3.2: Confronto top-1 accuracy per linear e kNN sul validation set di ImageNet fra DINO e diversi altri metodi di self-supervision. In grigio i risultati supervisionati, in arancione quelli di DINO.

3.7 Migliorare DINO

DINO migliora lo stato dell'arte in ambito di self-supervision, ma le sue **performance** risultano ancora **inferiori rispetto a metodi supervisionati** tradizionali (vedi Figura 3.2). Uno dei modelli emersi nel tentativo di evolvere l'approccio adottato da DINO è stato iBOT [29].

3.7.1 BERT

L'idea dei ricercatori dietro iBOT è stata quella di prendere ispirazione dal natural language processing, in particolare da BERT [4], un'evoluzione dei transformer nata nel 2018. La particolarità di BERT sta nel suo paradigma di pre-training, che prende il nome di *masked language modeling* (MLM). Questo consiste nel **mascherare casualmente alcuni dei tokens in input al transformer**, con obiettivo quello di predire l'ID nel vocabolario della parola che sono state mascherate, basandosi solo sul contesto fornito dalla frase. Questo approccio permette di risolvere una problematica dei transformers, ovvero che le parole vengono processate in una sola direzione, da sinistra verso destra, limitando la capacità espressiva delle features. Nei transformer tradizionali, la parte di frase analizzata finora viene utilizzata nella predizione del prossimo token, e se fosse consentita una analisi bidirezionale della frase, il modello avrebbe visibilità delle parole che deve predire, portando a soluzioni banali. MLM risolve il problema sostituendo una parte delle parole nella frase con il token [MASK] e utilizzando una testa di classificazione per ciascuno di essi per allenare la rete a ricostruire la parola originaria.

3.7.2 iBOT

iBOT (*image BERT pre-training with Online Tokenizer*) si ispira a DINO nella sua architettura e a BERT nel stabilire il suo pretext task, che prende il nome di *masked image modeling* (MIM). Data una immagine $x = \{x_i\}_{i=1}^N$, dove N è il numero di patches

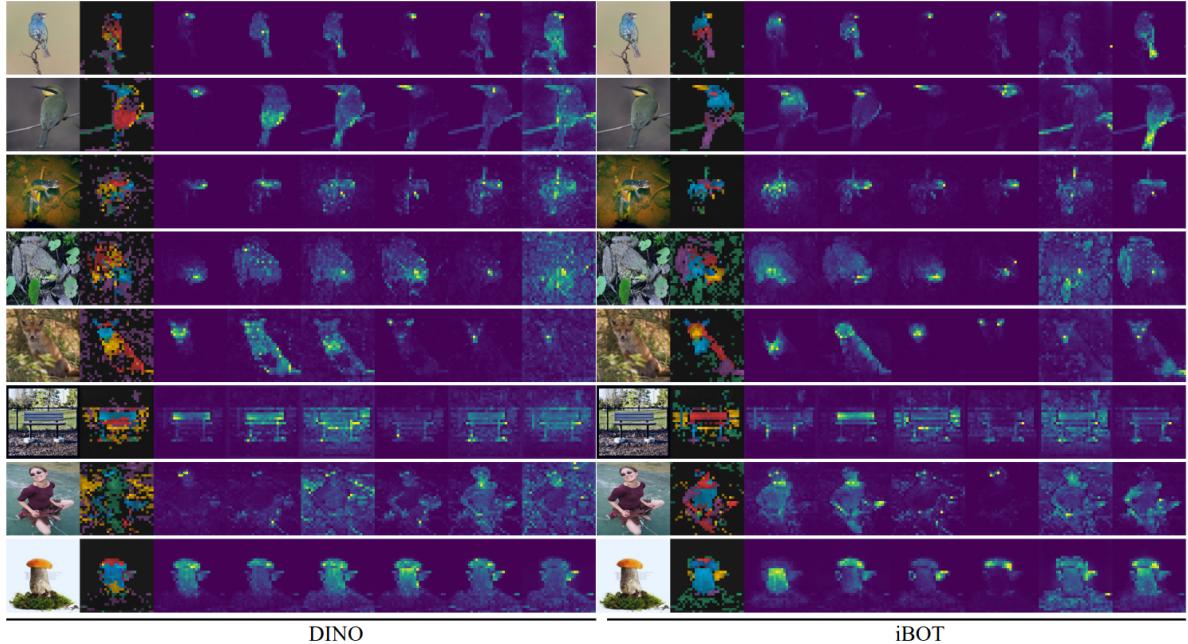


Figura 3.11: Confronto fra le teste di self-attention per il token [CLS] in iBOT e DINO. Si nota come le segmentazioni prodotte da iBOT siano più precise e meno rumorose, e come ciascuna testa rilevi un singolo particolare dettaglio dell’immagine relativa, in maniera più granulare rispetto DINO.

in cui è suddivisa (siamo sembre in ambito di ViT), MIM crea una **maschera binaria casuale** $m \in \{0, 1\}^N$. Se il patch x_i ha corrispondente valore in m_i pari a 1, allora quel patch viene sostituito col token [MASK]. L’**obiettivo** diventa quello di **recuperare le rappresentazioni corrette dei token mascherati**, a partire dall’immagine corrotta. La predizione, quindi, non viene fatta a livello di pixels dell’immagine, approccio che usano i *Masked Auto Encoders* (MAE), ma nello spazio delle features. Apprendere rappresentazioni per i token mascherati usando il contesto che li circonda **conduce a delle features ancora più ricche di significato** rispetto a quelle di DINO (Figura 3.11).

L’architettura della rete (Figura 3.12) rimane simile a quella di DINO, mantenendo la struttura di self-distillation con Student e Teacher e momentum encoder. Il teacher viene anche detto *online tokenizer* e il motivo è che **solo lo student riceve in input i patches mascherati**. Il ruolo del teacher è quello di **produrre le rappresentazioni per il token [MASK] che lo student dovrà replicare**. Si ricostruiscono i token mascherati con la supervisione del teacher.

Date due viste u e v della stessa immagine x , ogni vista passa attraverso i ViT f_s e f_t . Seguono delle teste di proiezione h formate da un MLP a 3 layer. Gli autori provano empiricamente che condividere i pesi delle teste $h^{[CLS]}$ e h^{patch} conduce a risultati migliori, in quanto la semantica ottenuta nella distillation su [CLS] può dare informazione in più nel completare l’obiettivo MIM.

La loss si compone della somma di due cross-entropy:

$$\mathcal{L} = \mathcal{L}_{[CLS]} + \mathcal{L}_{MIM} \quad (3.10)$$

dove $\mathcal{L}_{[CLS]}$ è la distillazione sul token CLS del ViT ed è simmetrizzata (come BYOL, Figura 3.7)

$$\mathcal{L}_{CLS} = -P_{\theta'}^{[CLS]}(v)^T \log P_{\theta}^{[CLS]}(u) \quad (3.11)$$

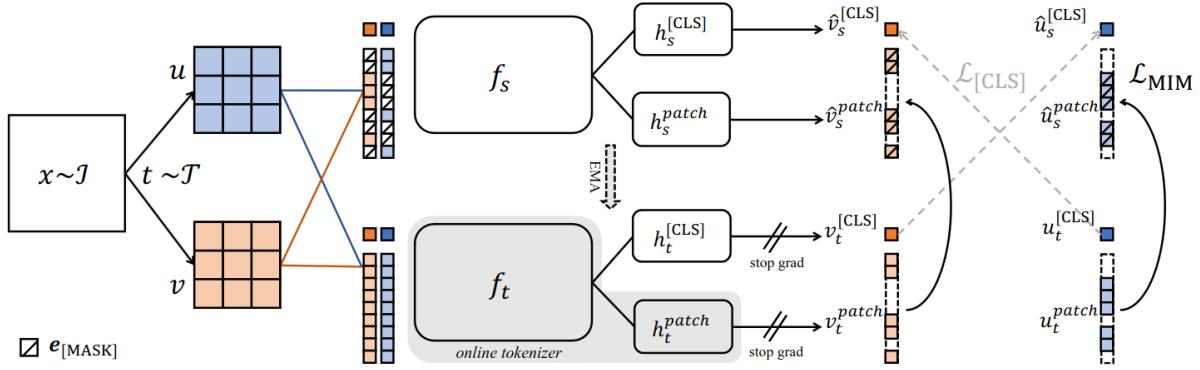


Figura 3.12: Architettura di iBOT. $h_s^{[CLS]}$ e h_s^{patch} condividono i pesi, così come $h_t^{[CLS]}$ e h_t^{patch} . L'input dello Student viene mascherato solo per l'obiettivo MIM, non per la costruzione di CLS.

con θ' i pesi del teacher e θ quelli dello student, e \mathcal{L}_{MIM} è la somma delle loss calcolate fra i patch [MASK] e le rappresentazioni dei rispettivi token prodotte dal teacher:

$$\mathcal{L}_{MIM} = - \sum_{i=1}^N m_i \cdot P_{\theta'}^{patch}(u_i)^T \log P_{\theta}^{patch}(\hat{u}_i) \quad (3.12)$$

iBOT migliora DINO (Figura 3.13) in termini di accuracy, ma è anche maggiormente scalabile, presentando un guadagno in performance maggiore all'aumentare del numero dei parametri.

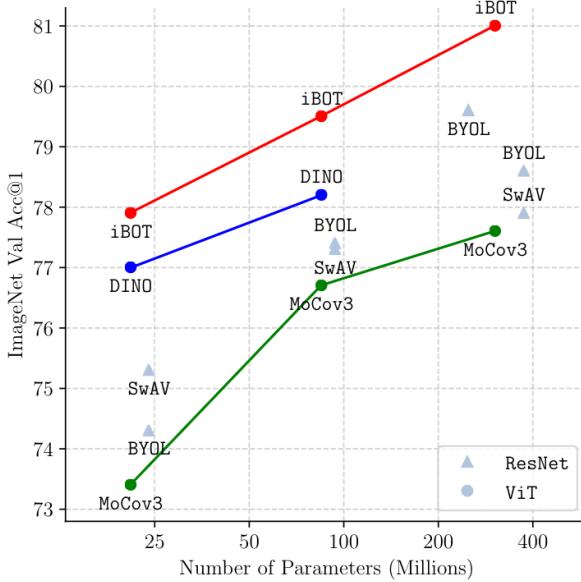


Figura 3.13: Confronto fra iBOT e altri modelli self-supervised. iBOT supera le prestazioni della concorrenza.

3.8 DINOV2

DINOV2 [18] viene pubblicato nell'aprile 2023 come una versione nuova e migliorata del suo predecessore del 2021, costruendo sulla base lasciata da iBOT. Mira ad **accelerare e stabilizzare il training su grande scala** di DINO, introducendo diverse **migliorie e ottimizzazioni nei suoi componenti più tecnici**, come il calcolo della attention o dello *stochastic gradient descent*, che non si andranno ad approfondire in questo elaborato.

3.8.1 LVD-142M

Un aspetto importante dell'innovazione portata da DINOV2 è l'introduzione di un **dataset curato e personalizzato** da utilizzare in fase di pre-training del modello. L'importanza di ciò è data dal fatto che la maggior parte degli avanzamenti in ambito di SSL sono stati fatti nel contesto di datasets di dimensioni ridotte, frequentemente su ImageNet-1k, che contiene 1.2 milioni di immagini nel suo split di training, circa 1300 per ciascuna delle sue mille classi. I tentativi di scalare questi approcci con **datasets non curati** di dimensioni maggiori ha portato inevitabilmente ad un **calo di qualità nelle features** a causa della **assenza di controllo sulla diversità** delle immagini in questi datasets e della loro **qualità**. Per questo motivo, viene costruito il dataset *LVD-142M*, che come dice il nome, si compone di un totale di 142 milioni di immagini, provenienti da sorgenti curate e non. La pipeline utilizzata per costruirlo è meglio spiegata nel paper [18], di seguito una overview:

- **Sorgenti:** si raccolgono più di 1.2 miliardi di immagini, prese da sorgenti curate (diverse varianti di ImageNet, Google Landmarks, Food-101, Caltech 101, e altri) e sorgenti non curate, ottenute tramite web scraping.
- **Deduplication:** si utilizza la pipeline di *copy detection* di Pizzi et al. [20] sui dati non curati, per effettuarne un filtraggio, rimuovere i duplicati o immagini troppo simili fra loro, e aumentare la diversità nel dataset.
- **Retrieval:** LVD-142M viene costruito ottenendo immagini simili a quelle curate dalle sorgenti non curate, già ripulite dalla deduplication. Per fare questo, si calco-

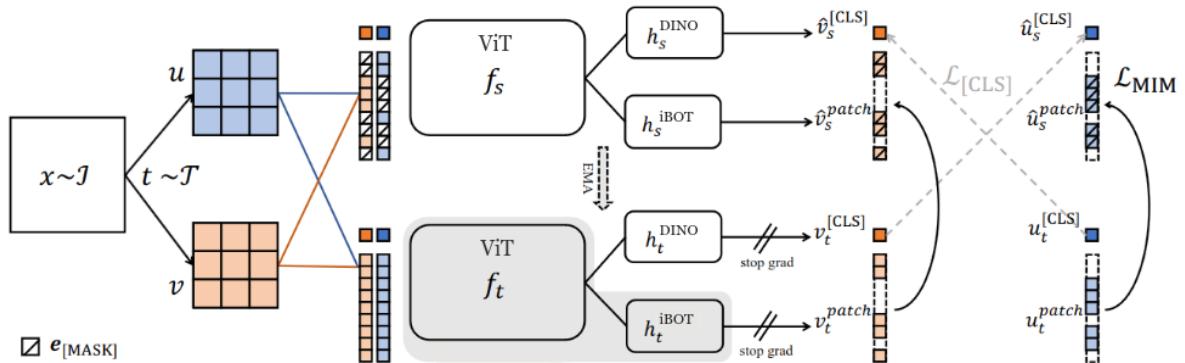


Figura 3.14: Architettura di DINOV2. È la stessa di iBOT (Figura 3.12, con la differenza che i pesi delle teste sono separati questa volta).

	INet-1k k-NN	INet-1k linear
iBOT	72.9	82.3
+ (our reproduction)	74.5 \uparrow 1.6	83.2 \uparrow 0.9
+ LayerScale, Stochastic Depth	75.4 \uparrow 0.9	82.0 \downarrow 1.2
+ 128k prototypes	76.6 \uparrow 1.2	81.9 \downarrow 0.1
+ KoLeo	78.9 \uparrow 2.3	82.5 \uparrow 0.6
+ SwiGLU FFN	78.7 \downarrow 0.2	83.1 \uparrow 0.6
+ Patch size 14	78.9 \uparrow 0.2	83.5 \uparrow 0.4
+ Teacher momentum 0.994	79.4 \uparrow 0.5	83.6 \uparrow 0.1
+ Tweak warmup schedules	80.5 \uparrow 1.1	83.8 \uparrow 0.2
+ Batch size 3k	81.7 \uparrow 1.2	84.7 \uparrow 0.9
+ Sinkhorn-Knopp	81.7 =	84.7 =
+ Untying heads = DINOv2	82.0 \uparrow 0.3	84.5 \downarrow 0.2

Tabella 3.3: Un confronto fra iBOT e DINOv2. Si mostrano tutte le novità introdotte in quest'ultimo e il guadagno di prestazioni che comportano.

lano vettori di features per tutte le immagini utilizzando un ViT-H/16 pre-allenato su ImageNet-22k. Dopodichè, per ogni dataset curato D :

- Se D contiene almeno 1 milione di immagini, si prendono N nearest-neighbors nel set non curato per ogni immagine in D .
- Altrimenti, si effettua un clustering dei dati non curati e si prende un numero M di immagini dal cluster a cui ciascuna immagine di D appartiene, almeno 3.

Il risultato è un dataset curato di grandi dimensioni che permette di scalare DINO nel riconoscere un maggior numero di classi e in maniera più accurata, grazie alla possibilità di presentargli una gamma più ampia di esempi per ciascuna classe.

3.8.2 Architettura

L’architettura di DINOv2 (Figura 3.14) ricalca quella di iBOT. Lo stesso vale per la sua funzione di loss (Equazioni 3.11 e 3.12), una somma del confronto con cross-entropy fra token [CLS] prodotti da Student e Teacher, e fra patch [MASK] e la rispettiva rappresentazione prodotta dal Teacher. Per le augmentation si usa lo stesso protocollo di multi-cropping usato in DINO. Viene utilizzata la testa di proiezione di DINO sull’output del CLS token, mentre la testa di iBOT per i patch mascherati.

Le principali novità introdotte rispetto ad iBOT sono una serie di altre ottimizzazioni che migliorano le prestazioni del modello. Non si scenderà nei dettagli, di seguito solo un elenco di alcuni di essi e il loro apporto:

- **Projection heads:** Gli autori osservano che, su scala più ampia, mantenere gli stessi pesi per le teste risulta penalizzante a livello di performance di classificazione, a differenza di iBOT, in cui migliorava i risultati. Per questo motivo, ogni testa ha dei pesi separati su cui verrà fatto l’addestramento.
- **Sinkhorn Knopp:** una forma migliorata di centering, per evitare il collasso.

- **Koleo Regularization:** un termine aggiunto alla loss che incoraggia una più uniforme distribuzione delle rappresentazioni su tutto lo spazio delle features.
- **Image Resolution:** durante il training, la dimensione delle immagini in input viene aumentata da 224x224 a 512x512 per migliorare per le performance in *down-stream tasks* a livello di pixel, come segmentazione e object detection. Permette di individuare oggetti che a basse risoluzioni risulterebbero troppo piccoli.

4

Metodo proposto

In questo capitolo si discuterà del workflow seguito per ottenere i risultati che saranno esposti nel Capitolo 6. L’implementazione degli aspetti esposti è stata realizzata in Python, con la libreria PyTorch.

4.1 Feature Extractor: DINOv2

Come anticipato, utilizziamo DINOv2 come feature extractor per la nostra sperimentazione. Il modello di DINOv2 pre-allenato è stato scaricato dalla pagina GitHub di Meta [5]. Si tratta sempre di ViT/14, con dimensione dei patches di 14×14 pixels (per maggiori informazioni Appendice A.2).

Dopo che il modello di DINOv2 viene allenato nelle modalità descritte nella Sezione 3.8, si eliminano il Teacher, il mascheramento e le teste di proiezione. Di fatto, quindi, stiamo prendendo i pesi del ViT già allenati e congelati e usandoli per estrarre delle rappresentazioni delle nostre immagini.

4.2 Features

Per la classificazione utilizziamo due diverse tipologie di features estratte da DINOv2: il token [CLS] e l’AVG patch.

4.2.1 CLS

Come meglio descritto nell’Appendice A.2.1, CLS è un token interamente appreso durante l’allenamento di DINOv2, e quindi del ViT sottostante, che raccoglie il contenuto informativo di tutta l’immagine. Per questo motivo, è generalmente utilizzato come feature principale in task di classificazione, specialmente semantica.

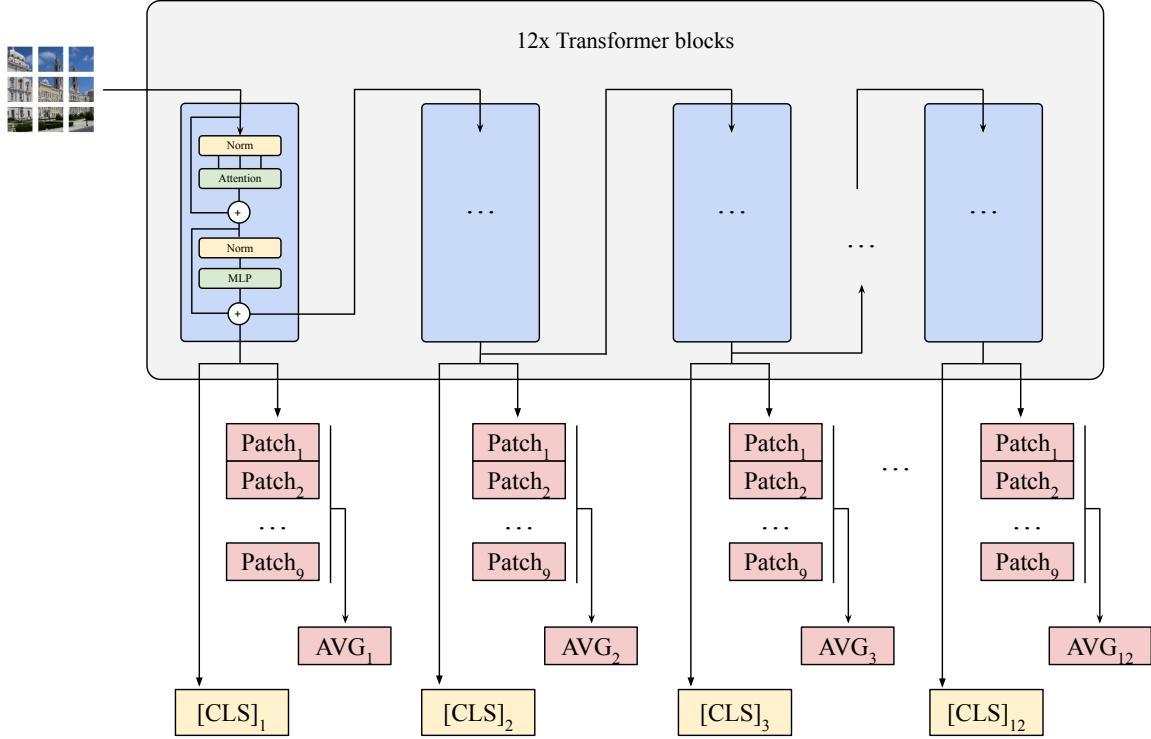


Figura 4.1: Estrazione dei token CLS e patch AVG dal ViT di DINOv2

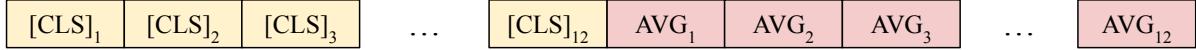


Figura 4.2: Concatenazione delle features

4.2.2 AVG patch

Il ViT produce un vettore di rappresentazioni per ciascuno dei patch in cui è suddiviso nell'immagine. Generalmente questi vengono ignorati, [CLS] è il vettore di embeddings che viene utilizzato nella classificazione (Figura A.3).

Nonostante ciò, sia nel paper che ha introdotto i Vision Transformers [6] che in quello di DINOv1 [2] si sperimenta anche utilizzando come vettore di features la media elemento per elemento delle rappresentazioni dei patches. Questo è uno degli approcci usati prima dell'introduzione di [CLS] da parte di BERT. Si dimostra che con determinati learning rates, questo vettore produce nei ViT risultati che si avvicinano a quelli di CLS.

Per questo motivo utilizziamo anche questo vettore media come feature su cui sperimentare, concatenandolo sequenzialmente al vettore estratto da [CLS].

4.2.3 Combinare le features di più layers

Un'ulteriore sperimentazione che già nella prima versione di DINO viene fatta è quella di non utilizzare solamente l'output preso dal token [CLS] all'ultimo layer, ma di concatenare gli outputs provenienti dai diversi *transformer blocks* precedenti all'ultimo, che compongono il ViT (Figura 4.1). Si osserva che questo può portare ad un miglioramento delle prestazioni del modello. L'intuizione è che i layer precedenti all'ultimo possano raccogliere informazioni a diversi livelli di astrazione, come succede nelle CNN.

Adottiamo questa tecnica sia per il token [CLS] che per l'AVG patch. Supponendo di scegliere un numero l di layers da cui estrarre [CLS] e un numero m di layers su cui fare la media dei patch, il vettore di features finale sarà la concatenazione degli l vettori CLS, seguito dalla concatenazione degli m vettori media (Figura 4.2).

4.3 Classificatore

Usiamo i due datasets descritti nel prossimo capitolo per allenare separatamente il nostro classificatore, con diverse combinazioni delle features proposte. Estraiamo in anticipo tutte le features per tutte le immagini appartenenti a ciascun dataset dal modello di ViT scelto (ViT-S, ViT-B, ViT-L) e le salviamo in un file *.pt*. In questo modo possiamo lanciare più scripts di evaluation senza dover ricalcolare per ciascuno i vettori di rappresentazione per ogni immagine.

Come classificatore proviamo un semplice layer lineare, pratica standard nei protocolli di valutazione dei modelli:

$$y = Wx + b \quad (4.1)$$

dove x il vettore di features scelte, W è la matrice dei pesi del layer e b il bias, entrambi appresi.

Proviamo poi anche un MLP con 1 hidden layer e funzione di attivazione ReLU:

$$y = W_2\sigma(W_1x + b_1) + b_2 \quad (4.2)$$

Aggiungiamo anche un dropout per mitigare il problema dell'overfitting, più probabile in un MLP che in un layer lineare. La dimensione del layer nascosto è pari alla metà di quella del layer in input.

In entrambi i casi, il vettore di features in input viene mappato su un vettore in output di dimensione pari al numero di classi presenti in ciascun dataset (9 e 17).

4.4 Loss

La loss utilizzata è la binary cross-entropy (BCE) with logits. Se x è uno dei logits prodotti in output dall'ultimo layer e y l'etichetta reale corrispondente a quella immagine per quella classe (0/1), allora

$$\text{BCEwithLogits}(x, y) = \text{BCE}(\sigma(x), y) \quad (4.3)$$

dove:

- $\sigma(x)$ è la funzione sigmoide applicata al logit, che lo porta nel range $[0, 1]$, rendendolo interpretabile come una probabilità:

$$p = \sigma(x) = \frac{1}{1 + e^{-x}} \quad (4.4)$$

- La binary cross-entropy è calcolata come:

$$\text{BCE}(p, y) = -(y \log(p) + (1 - y) \log(1 - p)) \quad (4.5)$$

La funzione BCE tratta ogni classe come una classificazione binaria (0 o 1), separatamente dalle altre. Per questo motivo può essere preferibile nella classificazione multi-label e non richiede che le immagini siano passate alla rete più di una volta. Per lo stesso motivo, viene utilizzata sigmoide al posto di softmax: mentre softmax normalizza un intero vettore di predizioni, σ lavora su ciascun logit separatamente, rendendola adatta in contesto di classificazione binaria.

4.5 Preparazione delle immagini

Per ridurre il numero di patches all'interno di una immagine, e quindi il tempo e lo spazio in memoria richiesto per estrarne le features, è necessario ridimensionare tutte le immagini date in input a DINOv2. Scegliamo di porre il lato più lungo di ciascuna di esse a dimensione 500 e di scalare la seconda dimensione di conseguenza. Inoltre, le dimensioni devono essere multiple di 14, in modo che si possano produrre un numero intero di patch 14x14. Per questo motivo, facciamo un crop centrale dell'immagine ridimensionata in modo che i suoi lati risultino multipli di 14.

Per velocizzare ancora di più il processo di estrazione delle features, sarebbe opportuno che le immagini siano quadrate, in modo da poterle mandare in batch alla rete. Questo, però, potrebbe risultare problematico nel contesto della composizione fotografica. Produrre delle immagini quadrate a partire da fotografie rettangolari richiede di fare crop, che inevitabilmente cambia la posizione relativa dei soggetti all'interno della composizione, distruggendo la regola dei terzi. Alternativamente, potremmo pensare di fare uno stretching dell'immagine fino a proporzioni quadrate, ma questo rischia di distorcere le leading lines, soprattutto quelle curve. Si decide quindi di mantenere gli aspect ratio originali, a scapito delle prestazioni.

5

Dataset

Come già menzionato nel Capitolo 2, i datasets che sono stati utilizzati in fase di sperimentazione sono due: KU-PCP (introdotto da [14]) e LODB (introdotto da [28]). Questo capitolo sarà dedicato ad esporre le caratteristiche di ciascuno e le classi di composizione fotografica che ne fanno parte.

5.1 KU-PCP

KU-PCP è un dataset per composizione fotografica che raccoglie immagini provenienti da siti di condivisione di fotografie, fra cui Flickr [7] e Photo.net [19]. È stato costruito dagli autori del paper [14] chiedendo a 18 soggetti umani con conoscenza nel settore di categorizzare le immagini in 9 classi. Di seguito l'elenco e una breve descrizione di ciascuna:

- **RoT**: regola dei terzi, come spiegata nella Sezione 1.1.1. I soggetti vengono posizionati lungo una delle linee che formano la griglia di suddivisione dell'immagine, o ai punti di intersezione fra queste. Guida lo sguardo dell'osservatore nell'immagine, crea armonia ed equilibrio visivo.
- **Center**: il soggetto dello scatto si trova centrato nell'immagine. Può trasmettere una sensazione di stabilità e simmetria, spesso utilizzata per ritratti o per enfatizzare l'importanza del soggetto nella scena.
- **Horizontal**: l'immagine presenta una leading line orizzontale. Spesso corrisponde con orizzonti e paesaggi, si vuole trasmettere un senso di vastità e calma.
- **Symmetric**: gli elementi sono disposti in modo speculare rispetto a un asse centrale. Si basa sull'equilibrio visivo fra due metà dell'immagine, creando un senso di ordine e armonia. È comune in immagini di architettura o riflessioni su corpi d'acqua in natura.
- **Diagonal**: l'immagine presenta una leading line diagonale. Crea dinamismo e movimento nell'immagine, possono aggiungere profondità o prospettiva alla scena.

- **Curve:** come menzionato in Sezione 1.1.2, la composizione sfrutta profili curvi per creare un flusso naturale, movimento.
- **Vertical:** leading lines verticali nell'immagine, spesso riferita ad edifici, alberi, elementi che si sviluppano in altezza e fotografati dal basso verso l'alto.
- **Triangle:** la forma degli oggetti nell'immagine ricorda una figura triangolare. In questa categoria vengono comprese anche leading lines convergenti in distanza, a formare profili triangolari, come una strada che prosegue verso l'orizzonte.
- **Pattern:** composizione basata sulla ripetizione di forme, colori o texture. Crea un forte impatto visivo e un senso di ritmo.

Il dataset è formato da 4.244 fotografie all'aperto. Di queste, circa 80% hanno una singola classe di ground truth, mentre le rimanenti possono avere fino a 3 labels. La Tabella 5.1 mostra il numero di immagini per label, mentre in Figura 5.1 si trova un esempio di immagine per ogni classe.

	<i>RoT</i>	<i>Center</i>	<i>Horiz.</i>	<i>Symm.</i>	<i>Diag.</i>	<i>Curved</i>	<i>Vert.</i>	<i>Triangle</i>	<i>Patt.</i>	<i>Total</i>
<i>RoT</i>	71	10	39	3	5	0	0	0	0	124
<i>Center</i>	10	224	6	9	7	1	4	52	0	310
<i>Horiz.</i>	39	6	97	0	37	21	3	12	0	210
<i>Symm.</i>	3	9	0	78	6	1	3	2	0	100
<i>Diag.</i>	5	7	37	6	117	18	3	1	1	188
<i>Curve</i>	0	1	21	1	18	45	0	7	1	91
<i>Vert.</i>	0	4	3	3	3	0	47	8	0	65
<i>Triangle</i>	0	52	12	2	1	7	8	91	0	171
<i>Patt.</i>	0	0	0	0	1	1	0	0	61	63

Tabella 5.1: Distribuzione delle classi nel dataset KU-PCP nella partizione di test. Siccome alcune immagini hanno 3 classi di ground-truth, può essere che un totale sia più piccolo della somma della riga corrispondente.



(a) RoT



(b) Vertical



(c) Horizontal



(d) Pattern



(e) Curved



(f) Triangle



(g) Center



(h) Symmetric



(i) Diagonal

Figura 5.1: Un esempio di immagine per ciascuna classe nel dataset KU-PCP.

5.2 LODB

Il dataset LODB viene creato dagli autori di [28] appositamente per compensare a quelle che loro individuano come debolezze di KU-PCP:

- **Scarsa varianza semantica intraclasse.** Le immagini che appartengono ad una stessa classe sono troppo simili fra loro, che può condurre i modelli allenati su di esse ad avere una comprensione troppo limitata del concetto che l'etichetta esprime.
- Il dataset contiene solamente **foto professionali**, come osservato da [26] (Sezione 2.3). Questo può creare un ambiente di apprendimento troppo perfetto, rendendo le prestazioni dei modelli scarse a fronte di immagini che presentano leggeri difetti, snapshots.
- **Coarse granularity.** Le classi sono troppo generiche, codificano aspetti di composizione troppi ampi e generici.

Si raccolgono immagini provenienti da diversi altri datasets open source di valutazione dell'estetica delle fotografie, e come per KU-PCP, si lascia che un gruppo di esperti le categorizzino in classi di composizione. Il risultato è un dataset di 6029 immagini e 17 classi, molto più granulari rispetto a KU-PCP. Un totale di 168 immagini possiedono etichetta doppia, non ci sono immagini con 3 o più classi di ground truth. In Tabella 5.2 sono presentate tutte le label introdotte e in Figura 5.2 un esempio per ciascuna.

5.3 Considerazioni

Il dataset KU-PCP è il più utilizzato in ambito di classificazione della composizione fotografica. Di conseguenza esiste un metro di paragone fra i risultati che otteniamo in questo studio e quelli proposti da altri papers. Più volte, però, vengono messi in evidenza i suoi difetti (come detto in Sezione 2.3 e precedentemente nel definire LODB).

LODB è stato introdotto molto recentemente (marzo 2024) insieme a [28], (Sezione 2.4). Essendo così nuovo, ancora non esistono altri paper che ne fanno uso per allenare i propri modelli, quindi non esistono studi sulla sua bontà, al di fuori di quello avanzato dai creatori. L'unico confronto che si può effettuare in termini di prestazioni è quello con i risultati proposti nello stesso paper [28] in cui LODB viene introdotto. Inoltre, come si nota dalla Tabella 5.2, la cardinalità delle classi è molto sbilanciata, passando da classi come *Center* con 947 immagini, a classi come *Oline* con solamente 110.

Visti i pro e contro di entrambi, si è deciso di effettuare la fase sperimentale sia su KU-PCP che su LODB e compararne le prestazioni.

Nome	Descrizione	Numero
<i>Cent.</i>	Soggetto centrato nell'immagine	947
<i>RoT.L</i>	Regola dei terzi, soggetto allineato con la griglia a sinistra	214
<i>RoT.R</i>	Regola dei terzi, soggetto allineato con la griglia a destra	275
<i>O2Dia.L</i>	Due oggetti disposti diagonalmente, dall'angolo in alto a sinistra	301
<i>O2Dia.R</i>	Due oggetti disposti diagonalmente, dall'angolo in alto a destra	293
<i>O2Hor.</i>	Due oggetti posizionati su una linea orizzontale	386
<i>03Li.</i>	Tre oggetti posizionati sulla stessa linea	157
<i>03Tri.</i>	Composizione triangolare formata da 3 oggetti	115
<i>Oline.</i>	Più di tre oggetti posizionati sulla stessa linea	110
<i>Pat.</i>	L'immagine contiene dei patterns	255
<i>DiaL.</i>	Composizione a diagonale, dall'angolo in alto a sinistra	446
<i>DiaR.</i>	Composizione a diagonale, dall'angolo in alto a destra	236
<i>Hor.</i>	Struttura orizzontale	574
<i>Tri.</i>	Struttura triangolare	451
<i>Ver.</i>	Struttura verticale	455
<i>Radi.</i>	Struttura radiale	187
<i>DiaX.</i>	Struttura comprendente entrambe le diagonali	861

Tabella 5.2: Distribuzione delle classi nel dataset LODB.



(a) Cent.



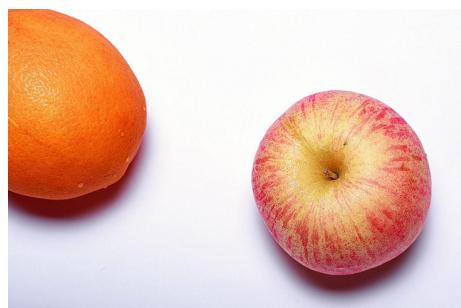
(b) Pat.



(c) RoT.L



(d) RoT.R



(e) O2Dia.L



(f) O2Dia.R



(g) O2Hor.



(h) Hor.



(i) O3Li.



(j) O3Tri.



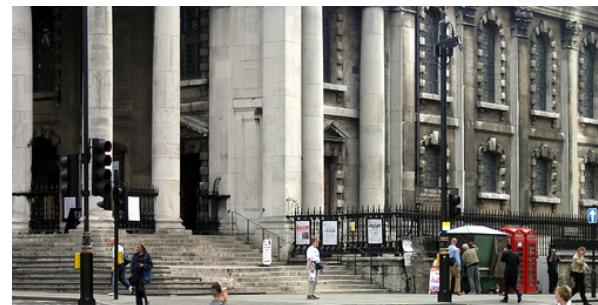
(k) DiaL.



(l) DiaR.



(m) Tri.



(n) Ver.



(o) Radi.



(p) Oline



(q) DiaX.

Figura 5.2: Un esempio di immagine per ciascuna classe nel dataset LODB.

6

Sperimentazione e Risultati

6.1 Setup sperimentale

I test sono stati condotti su una NVIDIA Tesla K80.

6.1.1 Partizione in train, test, validation

Sia per KU-PCP che per LODB, costruiamo le partizioni di training e test destinando a quest'ultima il 20% delle immagini appartenenti a ciascuna classe e il rimanente 80% alla partizione di training. Scegliamo questo approccio perchè le suddivisioni già fornite dagli autori dei paper che hanno introdotto i due dataset risultano essere troppo fortemente sbilanciate da poter dividere ulteriormente per ottenere il validation set (vedi Tabella 5.1).

Avendo dataset di dimensioni ridotte, per creare la partizione di validation si opta per la tecnica del *k-fold cross validation*: si suddivide la partizione di training in k insiemi e si addestra il modello k volte, ogni volta usando uno dei k sottoinsiemi come set di test e i restanti $k - 1$ come set di training. Si valutano le performance del modello su ciascun fold e i risultati finali sono la media delle metriche per fold. Scegliamo di utilizzare $k = 5$. I k insiemi vengono creati casualmente a partire dalla partizione di training.

Tutti i test per la scelta di quale sia la configurazione migliore di layers, learning rates, batch size, epoche da utilizzare vengono fatti sul validation set. I parametri che producono i risultati migliori verranno usati per produrre i risultati definitivi sulla partizione di test.

6.1.2 Metriche

Le metriche utilizzate durante la fase di sperimentazione sono le seguenti: *accuracy* (all), *accuracy* (any), *precision*, *recall*, mAP (*mean average precision*), *F1-score*.

Essendo quello della classificazione della composizione un task multi-label, possiamo decidere di calcolare la accuracy del modello in due modalità, che denominiamo *all* e *any*:

- All: una predizione viene considerata corretta solamente quando il modello predice correttamente tutte le classi di ground truth associate a una immagine. Individuare

correttamente una su due classificazioni per una stessa immagine conta come zero predizioni corrette. Viene calcolata come il rapporto fra il numero immagini le cui classi sono state tutte predette correttamente e il numero totale di immagini nella partizione di test (o validation).

- Any: ogni predizione viene considerata separatamente dalle altre riferite alla stessa immagine. Viene calcolata come il rapporto fra il numero di singole predizioni corrette e il numero di etichette totali nella partizione di test (o validation).

Precision e recall vengono calcolate per ciascuna classe:

- Precision: il numero di esempi positivi correttamente classificati in una classe sul totale di esempi classificati positivamente per la stessa classe.

$$\text{Precision} = \frac{TP}{TP + FP} \quad (6.1)$$

- Recall: il numero di esempi positivi correttamente classificati in una classe sul numero di realmente positivi nella stessa classe.

$$\text{Recall} = \frac{TP}{TP + FN} \quad (6.2)$$

dove:

- TP = true positive
- FP = false positive
- TN = true negative
- FN = false negative

Troviamo poi la *mean precision* (MP) come media delle precision per classe calcolate in ciascun fold. Introduciamo anche l'F1-score come media armonica di precision e recall per classe:

$$F1 = 2 \cdot \frac{\text{precision} \cdot \text{recall}}{\text{precision} + \text{recall}} \quad (6.3)$$

Il mean F1-score (MF1) è la media degli score delle classi in un fold.

Il modello deve essere in grado di predirre più di una label per immagine. Per questo motivo, non possiamo scegliere solamente la classe con la probabilità più alta nel vettore di output della rete e considerare quella come unica predizione effettuata dal modello. È necessario porre un **valore di soglia** sulle probabilità per classe, in modo tale che il vettore delle predizioni possa essere binarizzato: tutte le probabilità al di sopra della soglia sono da considerarsi come classi predette dal modello, mentre quelle al di sotto della soglia saranno ignorate. Le metriche presentate finora verranno calcolate con un valore di threshold **posto a 0.5**: le classi che hanno una probabilità maggiore del 50% di essere vere vengono considerate come predizioni del modello.

In questo contesto, introduciamo la *mean average precision* (mAP). La *average precision* (AP) viene definita come l'area sottesa dalla *curva precision-recall*, che viene ottenuta facendo variare il valore di threshold in un determinato intervallo o insieme di valori e

disegnando sul grafico i punti corrispondenti ai valori di recall (asse x) e precision (asse y) ottenuti a ciascuna soglia, con riferimento ad una classe. Può essere riassunta come:

$$AP = \sum_n (R_n - R_{n-1})P_n \quad (6.4)$$

dove P_n e R_n sono le precision e recall calcolate all' n -esimo threshold. I valori in cui il threshold varia sono le attivazioni uniche nel vettore di predizioni. La mAP corrisponde alla media delle AP per tutte le classi. Questa metrica è stata definita seguendo l'implementazione della libreria di machine learning per Python *scikit-learn* [24].

6.2 Considerazioni sulle features

Si presenta uno studio sulla scelta delle features migliori fra quelle esposte nel Capitolo 4, con riferimento al dataset LODB e sulla partizione di validation. Le stesse considerazioni valgono anche per KU-PCP. I risultati finali in test e su entrambi i datasets verranno esplorati nella prossima sezione.

Tutti gli esperimenti in questa fase di considerazioni vengono eseguiti con la seguente configurazione:

- Batch size: 1024
- Loss: BCE with logits
- Epoche: 100
- Encoder: ViT-S/14
- Optimizer: SDG (*stochastic gradient descent*)
- Features di DINov2 in input a un layer lineare

Le epoche vengono mantenute fisse e viene modificato solamente il *learning rate*, per poter avere un confronto più equo fra le varie configurazioni. Si scelgono 100 epoche perché costituiscono un numero abbastanza alto di iterazioni da permettere un allenamento che metta in risalto le differenze fra le varie configurazioni, ma abbastanza basso da poter di lanciare più test in maniera sufficientemente rapida. Aumentando il numero di epoche di training i modelli possono migliorare le loro prestazioni, ma in fase di validation l'obiettivo è quello di trovare la combinazione di parametri migliori ai fini del testing finale, non quella di produrre già il risultato migliore possibile.

6.2.1 CLS

Si effettua un confronto fra l'utilizzo del token [CLS] dell'ultimo transformer block contro l'estrazione di più tokens da più blocchi. Si osserva che i risultati migliori in accuracy (any) sono ottenuti utilizzando un solo token [CLS], Tabella 6.1.

Osserviamo che aumentando il numero di layer considerati, le accuracy tendono a calare sempre di più. Potremmo attribuire questo fenomeno al fatto che sia altamente probabile che le **features di DINov2 non si adattino al task di classificazione della composizione**, per la modalità in cui il modello è pre-allenato e allo scopo per cui viene allenato, la classificazione semantica. Aumentare il numero di features aggiunge

informazioni irrilevanti, che non fanno altro che inquinare le predizioni del modello e aumentarne la difficoltà nel convergere, portando a risultati pessimi. Questo spiegherebbe anche perché, pure utilizzando un solo token, le prestazioni su molte classi che rappresentano concetti più astratti e meno semanticamente significativi, come *O2Hor*, *DiaL*, rispetto a classi come *Pattern*, *Triangle*, *Hor*, siano scarse. Questo aspetto verrà meglio esplorato nel prossimo capitolo (Sezione 7.1.1).

Inoltre, lo sbilanciamento del dataset LODB influisce fortemente sulla natura dei risultati. Il modello tende a predire più frequentemente le classi di cui ha tanti esempi, come *Center*, *DiaX*, *Hor*. (vedi Tabella 5.2), e a predire raramente (o mai) classi poco frequenti, come *O2DiaL* e *O3Tri*. Questo si nota dalle matrici di confusione per classe (Tabella 6.3) e dal confronto fra precision e recall (Tabella 6.2): alcune classi hanno una alta precision perché il modello le predice solamente nei rari casi in cui è sicuro che siano corrette, probabilmente perché molto simili ai pochi esempi che ha visto in training, ma hanno recall molto bassa perché le predizioni del modello sono sempre orientate verso le classi di cui ha tanti esempi e meno verso classi meno frequenti. Del totale delle istanze di una determinata classe, ne vengono individuate solamente una porzione molto ridotta. Questo fenomeno è riassunto anche dagli F1-scores, alti solo in corrispondenza di classi numerose. Esistono comunque delle eccezioni. Ad esempio, *Pattern* viene riconosciuto in buona parte dei casi, nonostante abbia un numero ridotto di immagini nel dataset, probabilmente perché rappresenta un concetto molto diverso da tutte le altre classi, che lo rende più facilmente riconoscibile e meno confondibile.

Features	lr	Acc. (Any)	Acc. (All)	MP	mAP
CLS ₁₂	0.05	0.52	0.42	0.46	0.49
CLS ₉ ... CLS ₁₂	0.006	0.45	0.44	0.65	0.58
CLS ₅ ... CLS ₁₂	0.002	0.38	0.37	0.58	0.56
CLS ₁ ... CLS ₁₂	0.0005	0.20	0.19	0.33	0.47

Tabella 6.1: Un confronto fra le prestazioni del layer lineare utilizzando diverse combinazioni di layer di token [CLS]

6.2.2 AVG

Usare un patch AVG migliora leggermente le prestazioni della rete in confronto ad utilizzare un token CLS, in termini di accuracy e precision (Tabella 6.4). Il sospetto è che CLS sia meno indicato ai fini della classificazione per un task scollegato alla semantica, perché appreso durante il training di DINO appositamente per racchiudere il significato semantico dell'immagine. Mediare i patch, invece, può fornire informazioni più varie su cui distinguere le diverse classi. In generale, però, anche usando AVG i risultati sono sotto le aspettative e utilizzare più layers porta ad un deterioramento delle prestazioni. Potremmo interpretarlo come ulteriore sintomo del fatto che le features di DINOV2 non siano adatte al task.

Classe	Precision	Recall	F1-score
<i>DiaL.</i>	0.8235	0.1818	0.2979
<i>O2DiaR.</i>	0.2500	0.0208	0.0385
<i>O2Hor.</i>	0.8000	0.1176	0.2051
<i>Hor.</i>	0.9080	0.8495	0.8778
<i>Cent.</i>	0.7079	0.5164	0.5972
<i>Pat.</i>	0.9730	0.7059	0.8182
<i>Radi.</i>	0.0000	0.0000	0.0000
<i>RoT.R</i>	0.7000	0.1795	0.2857
<i>Ver.</i>	0.9273	0.6623	0.7727
<i>O3Li.</i>	0.5000	0.0400	0.0741
<i>RoT.L</i>	0.5000	0.0303	0.0571
<i>DiaR.</i>	0.8182	0.2500	0.3830
<i>DiaX.</i>	0.8952	0.8538	0.8740
<i>O3Tri.</i>	0.0000	0.0000	0.0000
<i>O2DiaL.</i>	0.0000	0.0000	0.0000
<i>Oline</i>	1.0000	0.1818	0.3077
<i>Tri.</i>	0.8529	0.4915	0.6237

Tabella 6.2: Metriche per classe del quinto fold nella configurazione $CLS_5 \dots CLS_{12}$ (8 layers), $lr = 0.002$. Si nota come gli F1-scores più alti siano in corrispondenza delle classi più numerose nel dataset. Allo stesso modo, le classi che hanno alta precision e bassa recall sono quelle meno frequenti o più granulari, che spesso codificano differenze nel numero dei soggetti nell'immagine.

	$\sim DiaX$	$DiaX$		$\sim O3Tri$	$O3Tri$
$\sim DiaX$	805	13	$\sim O3Tri$	937	0
$DiaX$	19	111	$O3Tri$	11	0

Tabella 6.3: Matrici di confusione del quinto fold per *DiaX* e *O3Tri* nella configurazione 8 CLS e $lr = 0.002$ e nel fold 5. Si nota come *DiaX*, che conta 861 esempi nel dataset, venga individuata correttamente la maggior parte delle volte, mentre *O3Tri*, 115 esempi, non venga mai predetta, da cui le metriche azzerate in Tabella 6.2. Si vede anche come in questo specifico fold siano presenti 130 istanze di *DiaX*, contro solamente 11 di *O3Tri*.

Features	lr	Acc. (Any)	Acc. (All)	MP	mAP
AVG ₁₂	0.06	0.52	0.49	0.61	0.59
AVG ₉ ... AVG ₁₂	0.005	0.43	0.42	0.60	0.59
AVG ₅ ... AVG ₁₂	0.002	0.41	0.41	0.55	0.56
AVG ₁ ... AVG ₁₂	0.001	0.35	0.34	0.51	0.54

Tabella 6.4: Un confronto fra le prestazioni del layer lineare utilizzando diverse combinazioni di layer di AVG patch

6.2.3 Concatenazione di CLS e AVG

Si osserva un leggero miglioramento delle prestazioni utilizzando i due tipi di feature in combinazione (Tabella 6.5). Inoltre, i risultati migliorano aumentando il numero di layers considerati, soprattutto in termini di precision. Potremmo stimare che usare i due token in coppia possa fornire due tipologie di informazione diversa che la rete riesce a sfruttare meglio piuttosto che riceverle singolarmente, ma il guadagno in prestazioni è troppo basso per poter giungere a conclusioni significative.

Inoltre, si provano anche a combinare primi e ultimi layers di CLS e AVG, per verificare se in questa modalità sia possibile raccogliere informazioni provenienti da diversi livelli di astrazione, come accade nelle CNN, che possano aiutare in un task scollegato dalla semantica come la composizione. Dai risultati ottenuti, però, non possiamo giungere a questa conclusione.

Features	lr	Acc. (Any)	Acc. (All)	MP	mAP
CLS ₁₂ AVG ₁₂	0.01	0.49	0.46	0.58	0.56
CLS ₉ ... CLS ₁₂ AVG ₉ ... AVG ₁₂	0.005	0.51	0.49	0.64	0.63
CLS ₅ ... CLS ₁₂ AVG ₅ ... AVG ₁₂	0.001	0.42	0.41	0.63	0.60
CLS ₁ ... CLS ₄ CLS ₉ ... CLS ₁₂ AVG ₁ ... AVG ₄ AVG ₉ ... AVG ₁₂	0.001	0.39	0.38	0.56	0.58

Tabella 6.5: Un confronto fra le prestazioni del layer lineare utilizzando diverse combinazioni di layer di token [CLS] e AVG patch.

6.3 Risultati finali

La configurazione di features scelta è di 4 token [CLS] e 4 AVG patch dei layer finali. Con questa e con batch size 1024, SGD e ViT-S/14 si fanno ulteriori prove in validation per stabilire il numero migliore di epoche e learning rate da usare in fase di test.

Si è sperimentato anche usando ViT-B e ViT-L (Appendice A.2.3), ma non si sono osservati miglioramenti significativi delle prestazioni del modello, a fronte di una richiesta di risorse di tempo e memoria fisica nettamente aumentate.

6.3.1 Linear layer

Si riportano i risultati ottenuti su LODB e KUPCP in Tabella 6.6. I risultati su LODB non sono direttamente paragonabili a quelli del paper [28] che lo ha introdotto (discusso in Sezione 2.4) perché le metriche da loro esposte sono legate al metodo da loro utilizzato (graph attention network). Possiamo comunque concludere che le prestazioni siano nettamente al di sotto di quella che potremmo definire una buona prestazione. I risultati su KU-PCP sono migliori, ma ancora inferiori a quelli ottenuti da altri metodi in letteratura (Tabella 6.7). In Tabella 6.9 si riportano le metriche per classe per KU-PCP e in Tabella 6.8 quelle per LODB, nelle Figure 6.1 e 6.2 i grafici delle loss, che non mostrano segni di overfitting. Dalla Figura 6.5 si trovano le predizioni effettuate dalla rete per alcune immagini di entrambi i datasets, che mettono in risalto alcune delle loro criticità.

Dataset	Epoche	lr	Acc. (Any)	Acc. (All)	MP	mAP	MF1
KUPCP	300	0.003	0.83	0.63	0.84	0.81	0.79
LODB	200	0.003	0.56	0.52	0.66	0.65	0.49

Tabella 6.6: Risultati in fase di test utilizzando un linear layer. Tutti i risultati sono presentati con threshold per le probabilità posto a 0.5.

Metodo	Acc. (Any) %
FT_CNN [14]	88.8
RSTN [26]	90.9
DINOv2 + linear	82.9

Tabella 6.7: Confronto fra il nostro metodo e gli altri metodi in letteratura sviluppati per la classificazione su KU-PCP discussi nel Capitolo 2.

6.3.2 MLP

Con MLP come classificatore, facendo diverse prove e modificando la quantità e tipologia di features usate, su KU-PCP si arriva sempre a pareggiare o stare leggermente al di sotto dei risultati ottenuti tramite layer lineare. Su LODB le differenze sono di poco più pronunciate, ma non si discostano in maniera radicale. Anche le differenze osservate a fronte della scelta di diversi layer di features sono minime. Possiamo affermare che non sia richiesto un modello più complesso di un semplice layer lineare per sfruttare al meglio i dati a disposizione. Come già ribadito, il fattore limitante sono le features, non il classificatore.

Si mantiene la configurazione di 4 token CLS e 4 AVG patch per avere un confronto diretto con i risultati ottenuti tramite layer lineare. I valori finali sono raccolti in Tabella 6.10. Anche le metriche per classe sono pressoché invariate da quelle presentate classificando con linear layer, con differenze non significative nel range 1-2%, e con le stesse problematiche mostrate su LODB. Per questo motivo non vengono riportate.

Classe	Precision	Recall	F1-score
<i>DiaL.</i>	0.7333	0.3438	0.4681
<i>O2DiaR.</i>	0.2381	0.0847	0.1250
<i>O2Hor.</i>	0.4545	0.1923	0.2703
<i>Hor.</i>	0.9630	0.8254	0.8889
<i>Cent.</i>	0.8129	0.7079	0.7568
<i>Pat.</i>	0.9091	0.7843	0.8421
<i>Radi.</i>	0.5294	0.4286	0.4737
<i>RoT.R</i>	0.8333	0.5263	0.6452
<i>Ver.</i>	0.9615	0.8242	0.8876
<i>O3Li.</i>	0.4444	0.1143	0.1818
<i>RoT.L</i>	0.7333	0.2500	0.3729
<i>DiaR.</i>	0.7895	0.3261	0.4615
<i>DiaX.</i>	0.9308	0.8605	0.8943
<i>O3Tri.</i>	0.3333	0.0417	0.0741
<i>O2DiaL.</i>	0.1429	0.0159	0.0286
<i>Oline</i>	0.6250	0.2174	0.3226
<i>Tri.</i>	0.8182	0.7079	0.7590

Tabella 6.8: Metriche per classe su LODB in test con linear layer. Come già discusso precedentemente, si notano F1-score bassi in classi poco frequenti o che codificano il conteggio dei soggetti, e score più elevati in classi più numerose, accompagnate da alte precision e recall.

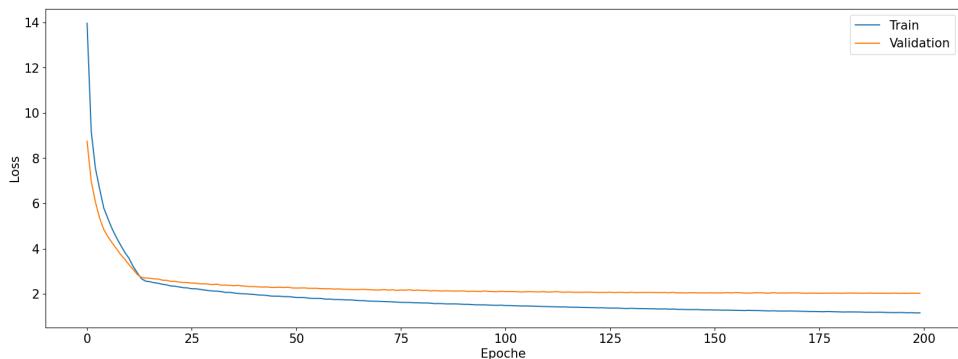


Figura 6.1: Grafico della loss in validation (quinto fold) su LODB e linear layer per la configurazione 4 token CLS e 4 AVG patch, 200 epoche, learning rate 0.003, che viene usata in fase di test. Non ci sono segnali di overfitting.

Classe	Precision	Recall	F1-score
<i>RoT</i>	0.6866	0.7480	0.7160
<i>Vertical</i>	0.8571	0.6364	0.7304
<i>Horizontal</i>	0.8390	0.8190	0.8289
<i>Diagonal</i>	0.7372	0.5344	0.6196
<i>Curved</i>	0.7722	0.6630	0.7135
<i>Triangle</i>	0.8487	0.7544	0.7988
<i>Center</i>	0.9135	0.7814	0.8423
<i>Symmetric</i>	0.9770	0.8586	0.9140
<i>Pattern</i>	0.9683	0.9683	0.9683

Tabella 6.9: Metriche per classe in test su KU-PCP con linear layer. A supporto della teoria che le features di DINOV2 riconoscano il contenuto semantico nelle immagini e usino quello per la classificazione, come si esplorera nel prossimo capitolo, *RoT* è la classe con prestazioni peggiori ed è anche quella che presenta varianza semantica intraclasse più elevata, al contrario di classi come *Symmetric* o *Pattern*, che infatti presentano risultati migliori.

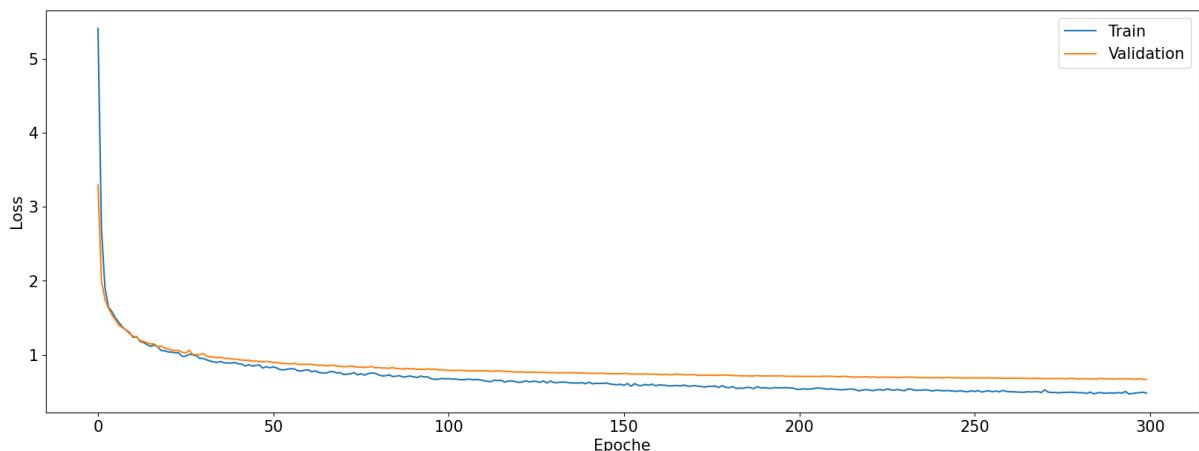


Figura 6.2: Grafico della loss in validation (quinto fold) su KU-PCP e linear layer per la configurazione con 4 token CLS e 4 AVG patch, 300 epoche e learning rate 0.003, che viene usata in fase di test. La discesa della loss non mostra segni di overfitting.

Dataset	Epoche	lr	Acc. (Any)	Acc. (All)	MP	mAP	MF1
KUPCP	200	0.005	0.84	0.54	0.83	0.76	0.77
LODB	300	0.002	0.51	0.50	0.68	0.67	0.44

Tabella 6.10: Risultati in fase di test utilizzando un MLP. I valori sono simili a quelli ottenuti tramite un singolo linear layer (Tabella 6.6) con differenze inferiori al 5% nella maggior parte dei casi, che potrebbero essere ridotte ancora di più prolungando l'allenamento o incrementando il learning rate.

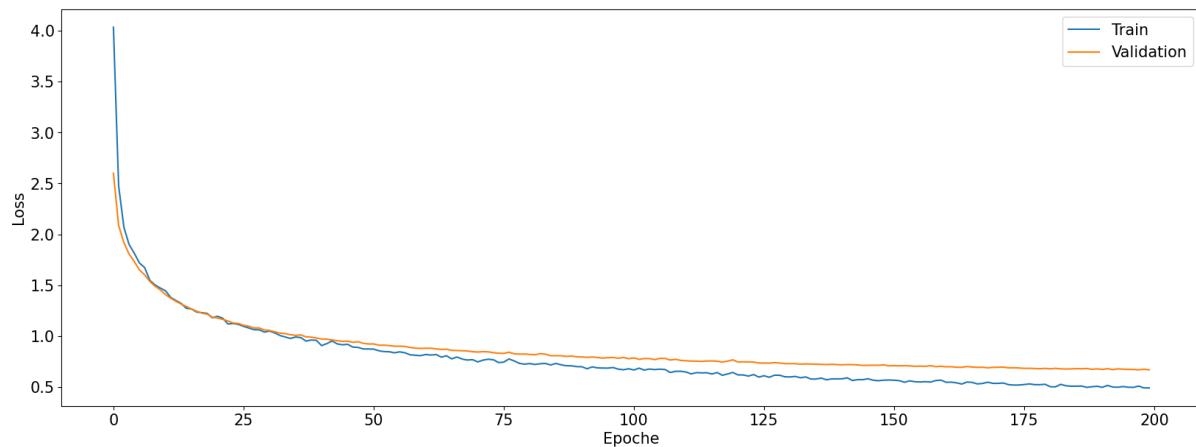


Figura 6.3: Grafico della loss in validation (quinto fold) su KU-PCP e MLP per la configurazione 4 token CLS e 4 AVG patch, 200 epoche, learning rate 0.005, che viene usata in fase di test. Non ci sono segnali di overfitting.

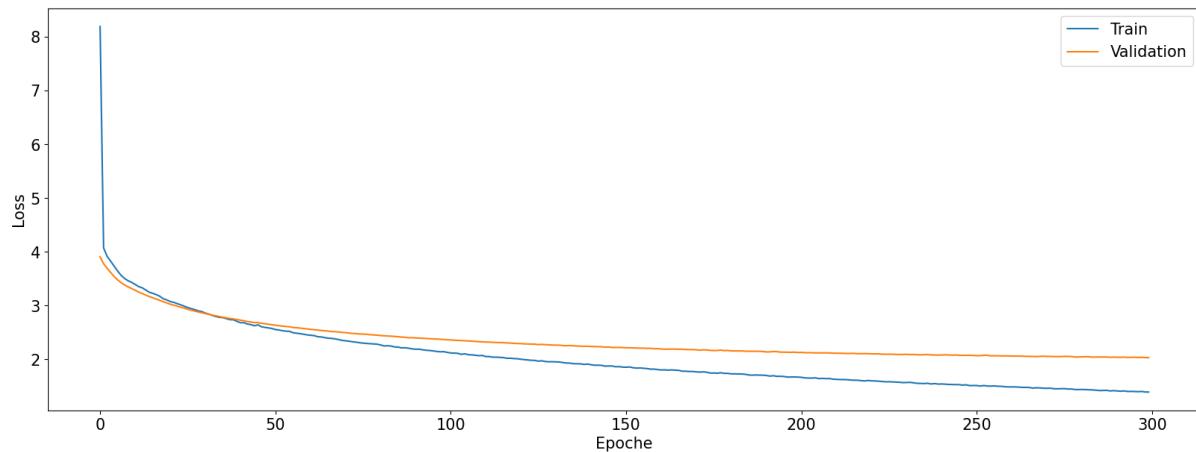


Figura 6.4: Grafico della loss in validation (quinto fold) su LODB e MLP per la configurazione 4 token CLS e 4 AVG patch, 300 epoche, learning rate 0.002, che viene usata in fase di test. Non ci sono segnali di overfitting.

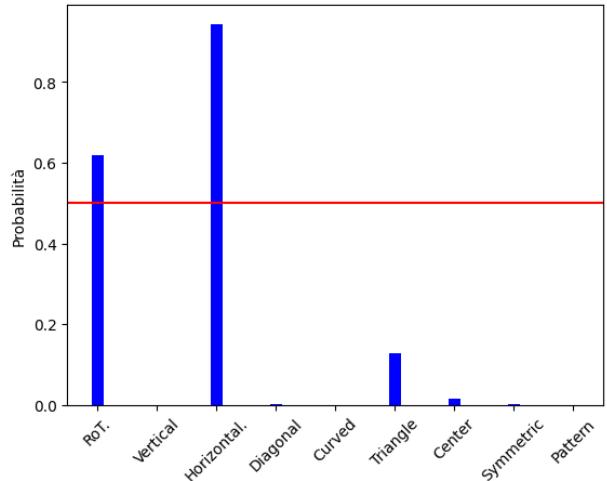


Figura 6.5: Esempio di predizione corretta della rete con linear classifier su KU-PCP. Le classi di ground truth sono *RoT.* e *Horizontal.*. In rosso è segnata la soglia di threshold (posta a 0.5) utilizzata per il calcolo delle metriche.

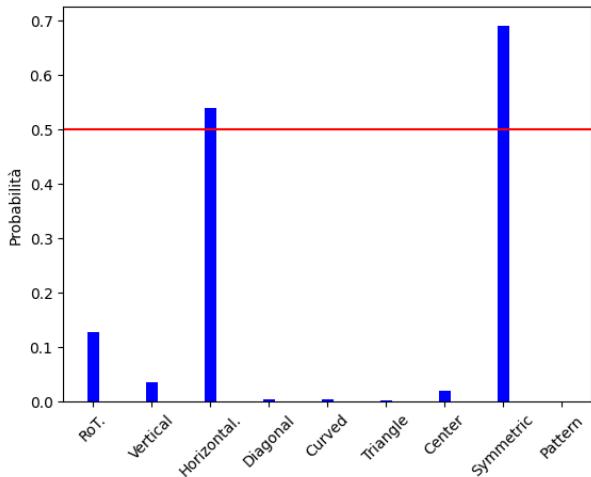


Figura 6.6: Predizione con classificatore lineare su KU-PCP. Le classi di ground truth sono *Vertical* e *Symmetric*, mentre la nostra rete predice *Horizontal* e *Symmetric*. Effettivamente, essendo uno scatto in lontananza della città, somiglia più alle immagini che nel dataset sono classificate come *Horizontal* perché ritraente un orizzonte, piuttosto che *Vertical*, di cui solitamente fanno parte immagini di palazzi in vista più ravvicinata. Le prestazioni, quindi, sono anche influenzate da un cattivo o non consistente etichettamento delle immagini nel dataset.

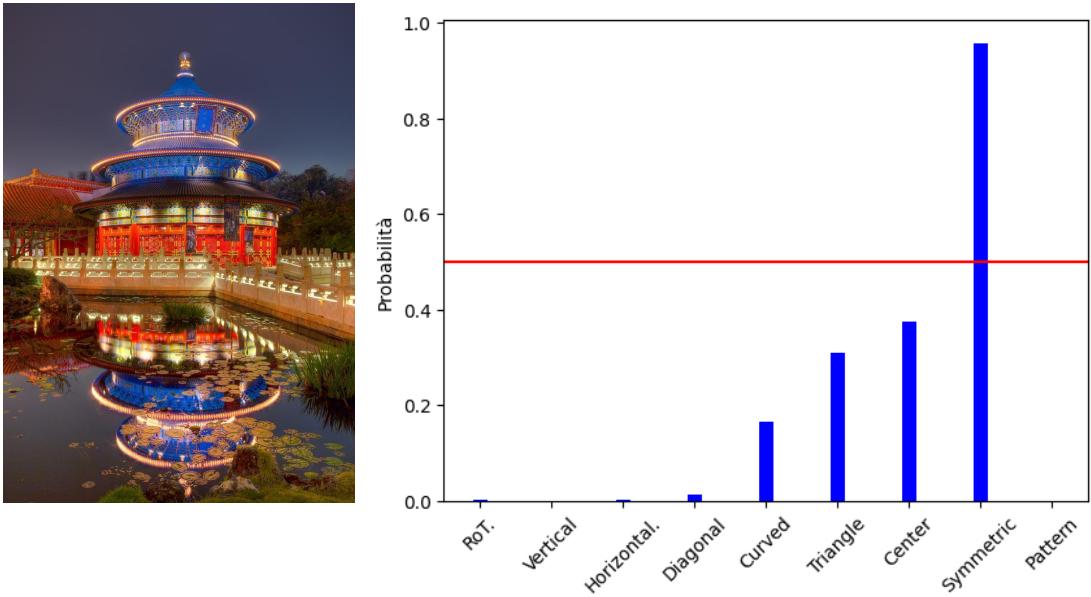


Figura 6.7: Predizione con classificatore lineare su KU-PCP, ground truth *Symmetric* e *Center*. Ancora una volta, l’etichettamento dato all’immagine nel dataset è fuorviante. La rete intuisce la forma dell’edificio e le curve che lo compongono, che sarebbero potute essere altre due valide classi di ground truth per l’immagine in questione.

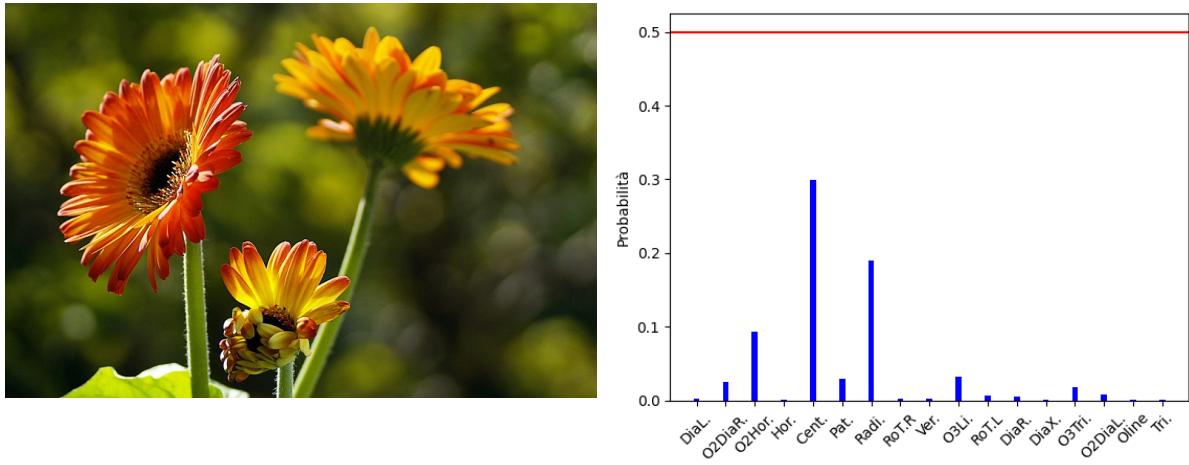


Figura 6.8: Predizione con classificatore lineare su LODB, ground truth *O3Tri..*. Il classificatore è molto confuso sulla classe da assegnare, nessuna raggiunge la soglia al di sopra della quale si considerano le predizioni. Questo risultato scarso è in parte dovuto anche all’etichetta assegnata all’immagine. La rete ha delle intuizioni valide: il primo piano di un soggetto viene spesso associato alla classe *Center* nel dataset, prendendo i fiori a coppia si potrebbero considerare come posti in diagonale, il fiore rosso a sinistra è in posizione *RoT.L*, la ripetizione dei petali potrebbe essere visto come *Pattern*, la forma del fiore come *Radi..*. Si notano le limitazioni delle classi introdotte da LODB, di granularità troppo elevata e che spesso si sovrappongono l’una all’altra, nonostante le immagini siano per la maggior parte etichettate con una sola label.

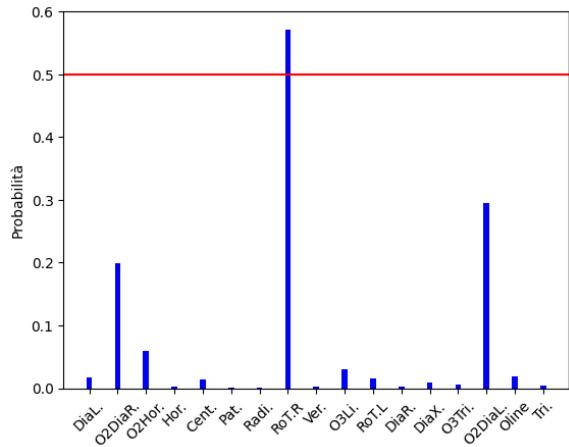


Figura 6.9: Predizione con classificatore lineare su LODB, ground truth *O2Diar..*. La rete predice *RoT.R.*, sbagliando completamente. *RoT.L.* sarebbe potuta essere una buona intuizione, l'uomo si trova all'intersezione degli assi sinistro e inferiore della griglia della regola dei terzi, ma con questa predizione il modello mostra di non aver compreso effettivamente la differenza fra classi *R* e *L*, indicato anche dal fatto che la seconda probabilità più alta sia quella di *O2Dial.*.

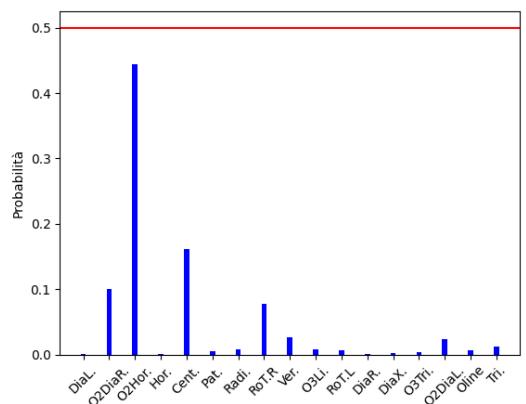


Figura 6.10: Predizione con classificatore lineare su LODB, ground truth *O2Hor..*. Anche su immagini la cui composizione risulta essere piuttosto chiara all'occhio umano, il modello fatica a riconoscerla e spalma le sue predizioni su diverse classi, facendo fatica a prendere una decisione.

7

Conclusioni e sviluppi futuri

7.1 Conclusioni

I risultati sul dataset LODB sono inferiori alle nostre aspettative, mentre per KU-PCP, le prestazioni sono al di sotto dello stato dell’arte, ma le metriche presentano comunque valori alti. Questo risulta sospetto, viste le prestazioni sull’altro dataset. Di seguito si espande su alcune delle motivazioni che potrebbero giustificare questi risultati e si traggono delle conclusioni sull’utilizzo di DINOv2 per il task di riconoscimento della composizione fotografica.

7.1.1 DINOv2

DINOv2 è nato per essere un modello di classificazione semantica dei soggetti in una immagine. Di conseguenza, l’intuizione è che le **features** che esso produce siano troppo **fortemente orientate a descrivere il contenuto semantico** dell’immagine, **piuttosto che** il come l’immagine in sé sia fatta, le sue **caratteristiche di alto livello**, fra cui la composizione. Questo, oltre ad altri fattori di composizione del dataset che si esploreranno in seguito, spiegherebbe le prestazioni molto basse su LODB.

La scarsa varianza intraclass registrata in KU-PCP, invece, è molto probabilmente il motivo per cui le prestazioni su di esso risultano più alte rispetto a LODB. Il modello potrebbe involontariamente associare una determinata classe di composizione al contenuto delle immagini in cui essa viene più spesso utilizzata. Ad esempio, la classe *vertical* coincide nella maggior parte delle immagini a paesaggi urbani, con grandi palazzi e grattacieli, o a paesaggi boschivi, con alti alberi. Il classificatore potrebbe capire queste associazioni (*vertical-grattacielo* o *vertical-albero*), piuttosto che il vero significato di una leading line verticale. Lo stesso vale per la classe *symmetric*, che spesso corrisponde ad un soggetto, frequentemente catene montuose, riflesso sulla superficie di un lago (Figura 7.2), una delle poche situazioni in cui la simmetria si presenta in natura. Anche *horizontal* è rappresentata sempre da orizzonti o separazioni nette fra cielo e area sottostante, e così via per le altre classi. Questo non è necessariamente un problema se l’obiettivo è solamente quello di riconoscere la composizione in foto di tipo paesaggistico. Aspetti come quelli codificati



Figura 7.1: Esempi di immagini nella classe *vertical* nella partizione di test KU-PCP. Una quantità importante di immagini che appartengono a questa classe sono fotografie di grattacieli.

in *vertical* o *symmetric*, infatti, si verificano in natura solamente nelle situazioni descritte (alberi, palazzi, riflessi) e in poche altre rare occasioni. Se invece l’obiettivo è quello di classificare la composizione in una più vasta gamma di tipologie di immagine, questa limitata variazione semantica peggiorerà significativamente le prestazioni del modello. Questa spiegazione motiverebbe il fatto che i risultati peggiori in precision (Tabella 6.9) si hanno nelle classi *RoT*, *Diagonal* e *Curved*, che sono quelle rappresentate da immagini più variegate, che il classificatore non riesce sempre ad associare a soggetti ricorrenti.

Inoltre, come detto, DINOv2 è allenato generalmente a classificare un solo soggetto nell’immagine. Questo potrebbe essere uno dei motivi per il quale i risultati in LODB sulle classi che si basano sul conteggio degli oggetti siano particolarmente bassi, oltre al problema dello sbilanciamento degli esempi visti dalla rete.

7.1.2 Riguardo i datasets

Come anticipato precedentemente e come puntualizzato anche dai creatori di LODB, la varianza semantica intraclass di KU-PCP è un suo grosso punto debole. Lo stesso vale per la qualità troppo professionale delle sue foto. LODB, però, presenta a sua volta un grande numero di problematiche. I suoi creatori sostengono che le classi di KU-PCP siano troppo generiche per codificare tutti gli aspetti della composizione fotografica. Per questo motivo ne introducono altre otto, oltre a modificare alcune delle già esistenti. Dall’analisi del dataset e dei risultati ottenuti, però, questa granularità aggiunta può risultare eccessiva:

- Vengono introdotte più classi che codificano differenze posizionali degli oggetti rispetto allo stesso stile compositivo, come *RoT.L* e *RoT.R*. Queste possono dare



Figura 7.2: Esempi di immagini appartenenti alla classe *symmetric* nella partizione di test di KU-PCP. A livello semantico, le immagini sono molto simili: montagne, alberi, foreste, riflessi in corpi d’acqua.

più dettaglio sul come è fisicamente composta l'immagine, ma non aggiungono particolare rilevanza rispetto ad avere un'unica classe *RoT*, ad esempio. A livello strettamente fotografico, lo scatto finale riceve lo stesso beneficio apportato dalla regola dei terzi sia che il soggetto sia allineato alla parte sinistra della griglia che in quella destra (Figura 1.1). Lo stesso vale per *DiaL.* e *DiaR.*, e non solo.

- Allo stesso modo, si introducono delle classi distinte fra loro solamente dal numero di soggetti presenti nell'immagine. Un esempio sono *O3Li.* e *Oline* con altre classi quali *O2Hor.*, *O2Dia.R*, *O2Dia.L*. Questo è un aspetto strettamente descrittivo dell'immagine e del suo contenuto, piuttosto che una sua caratteristica compositiva, quanto possono esserlo le leading lines o la regola dei terzi.
- L'aggiunta di un gran numero di classi non è accompagnata dall'aggiunta di un numero adeguato di immagini che possano rientrare nelle nuove categorie. Il dataset risulta essere **fortemente sbilanciato** (Tabella 5.2), con classi che contano più di 900 immagini, contro alcune delle nuove classi, che superano di poco le 100. Inoltre, le classi ad avere meno esempi sono proprio quelle introdotte da LODB. Questo si rispecchia pesantemente nei risultati ottenuti (Tabella 6.8).
- Spesso le classi si sovrappongono concettualmente (Figura 6.8), e quando sarebbe ragionevole assegnarne più di una ad una immagine, nella maggior parte delle volte non viene fatto. L'etichettamento è fatto in maniera molto conservativa, meno di 200 immagini presentano etichette doppie e nessuna tripla, quando da una analisi qualitativa del dataset, tante foto ne avrebbero giovato.

Riguardo all'obiettivo di aumentare la varianza intraclass delle immagini, LODB migliora rispetto a KU-PCP. Ad esempio, la classe *vertical*, su cui si è discusso precedentemente, ora comprende in generale strutture che si sviluppano in altezza. Questo, però, a scapito delle leading lines: mentre in KU-PCP *Vertical* corrispondeva spesso a palazzi, alberi uscenti dalla scena, grandi città, che trasmettevano imponenza, forza, vastità, ora qualunque oggetto con altezza maggiore della sua larghezza possiede questa label (Figura 7.3). Inoltre, la classe *Curved* viene eliminata, sostituita dalla classe *Radi.*, che codifica semplicemente oggetti circolari, ancora una volta perdendo il significato di leading line.

LODB si compone ancora in prevalenza di immagini professionali, ma introduce un maggior numero di snapshots e immagini più spontanee. Ci sono dei casi di classificazione errata o forzatura nell'assegnare le etichette (Figura 7.4), che avvengono anche in KU-PCP, come visto in Figura 6.6, che confondono le predizioni del modello. In generale,



Figura 7.3: Esempi di immagini classificate come *vertical* in LODB che non corrispondono a leading lines verticali.



Figura 7.4: Esempi di misclassification/forzatura della composizione. Sono indicate le etichette che LODB assegna a ciascuna.

LODB è **adatto** a una **descrizione** di alto livello **della posizione e del numero degli oggetti** nell’immagine ed è meno orientato alla composizione fotografica classica, come presentata nel Capitolo 1.

7.2 Sviluppi futuri

La risultati insoddisfacenti ottenuti in fase di sperimentazione hanno lasciato spazio a tanti ambiti in cui migliorare e tante idee per poterlo fare. Di seguito alcune delle strade ipotizzate.

7.2.1 Dataset

Un primo ovvio passo potrebbe essere quello di **costruire un nuovo dataset ad hoc per il task**, che migliori KU-PCP e LODB. KU-PCP è stato sostanzialmente l’unico dataset disponibile per la classificazione della composizione fotografica fino a pochi giorni prima dell’inizio dello stage, quando LODB è stato pubblicato. Anche quest’ultimo, però, ha i suoi limiti, come è stato discusso. Un problema che entrambi hanno è il numero ristretto di immagini di cui si compongono (4244 e 6029), a causa del processo di annotazione che richiede supervisione umana, e oltre a ciò, le persone coinvolte devono conoscere le regole di composizione fotografica ed essere in grado di riconoscerle.

Un’idea potrebbe essere quella di mantenere LODB come base, riducendo il numero di classi che lo compongono ed eliminando le immagini la cui classificazione appare ambigua o forzata. Dopodiché, possiamo effettuare uno scraping del web, specialmente di forum di condivisione di fotografia amatoriale e non, come Reddit ([r/pics \[23\]](#), [r/photographs \[22\]](#), [r/photocritique \[21\]](#)). Possiamo applicare le stesse tecniche di filtraggio e rimozione dei duplicati che usa DINOv2 (Sezione 3.8.1). Dovremmo raccogliere così un numero più che sufficiente di immagini.

Rimane il problema di annotare le immagini. Il modo più efficiente per farlo è tramite una forma di crowdsourcing, se l’obiettivo è creare un dataset di dimensioni superiori a LODB, di almeno diecimila immagini. Potremmo creare un questionario che chieda a ciascun utente di classificare un sottoinsieme ridotto e casuale di immagini fra quelle raccolte e con le classi da noi stabilite. Il sondaggio verrebbe pubblicato online, sugli stessi forum menzionati, o condiviso tramite i canali dell’università.

Quando ogni immagine possiederà un buon numero di opinioni su quale sia la sua regola di composizione, per scegliere quale sarà la classe finale da assegnare a ciascuna prendiamo la classe più votata, che deve anche aver superato una determinata quota di

preferenze, almeno 70%. Questo può andare a compensare una mancanza di controllo sul livello di conoscenza della composizione fotografica degli utenti che rispondono al questionario. Nel caso in cui ci siano altre classi che ricevono una percentuale dei voti al di sopra di una determinata soglia sul numero di preferenze ricevute dalla prima classe, allora anche questa rientrerà nella ground truth.

7.2.2 Fine Tuning

Si è ipotizzato che una criticità di DINOv2 per la composizione fotografica sia l’essere troppo orientato alla classificazione semantica, essendo allenato appositamente a quello scopo. Una soluzione ovvia potrebbe essere fare un *finetuning* della rete, allenando solamente gli ultimi layer del ViT sottostante, nella speranza che le features possano racchiudere meglio l’informazione compositiva.

Il problema di questo approccio è che per avere un impatto significativo sulle features è molto probabile che serva un numero di immagini molto superiore rispetto alle poche migliaia in KU-PCP o LODB. Nel paper di DINOv2 [18], gli autori non menzionano finetuning per task specifici, ma in quello di DINO [2] si ottengono buone prestazioni finetunando su dataset come INat18 (600k immagini), ma anche su Cars (16k immagini). Per questo motivo, avrebbe comunque valore fare della sperimentazione coi due datasets a disposizione su DINOv2 con l’encoder più piccolo possibile, un ViT-S, anche se l’aspettativa è che il numero di immagini sia insufficiente.

Una soluzione definitiva al problema sarebbe produrre il dataset sopra menzionato, ma una via più immediata potrebbe essere quella di impiegare delle **augmentations**: flip verticali e orizzontali, leggere rotazioni (come si è discusso nella Sezione 2.3), crop non distruttivi, modifiche che preservino la composizione dell’immagine. Questo può aiutare a rendere il modello più robusto e a creare un maggior numero di samples su cui fare finetuning della rete.

Potremmo impiegare lo stesso protocollo di augmentations anche per allenare il layer lineare o l’MLP usati nella fase sperimentale e verificare se ci sia un miglioramento nelle prestazioni della rete e se sia parzialmente anche il numero di immagini di allenamento a causare le prestazioni mediocri ottenute.

7.2.3 Mappe di self-attention

Una delle caratteristiche più importanti di DINO sono le mappe di self-attention che esso produce (Figura 3.8). Queste permettono al modello di focalizzarsi sulle parti dell’immagine più di rilievo e risultano solitamente nella segmentazione dei soggetti nella foto. Forniscono una semplificazione dell’immagine che potremmo sfruttare per estrarre informazioni sulla posizione dei soggetti (*RoT.*, *Center*), sulla loro forma (*Triangle*, *Pattern*) e sulle leading lines.

La scelta di come usare le mappe è importante. Linearizzarle e passarle ad un classificatore come quelli usati nella fase sperimentale potrebbe distruggere l’informazione spaziale che esse codificano, rimuovendo dettagli molto importanti per il riconoscimento di classi come *RoT.* o *Center*. Per risolvere questo problema, si potrebbe apporre ai layer di classificazione una CNN. Come visto nel Capitolo 2, finora le CNN sono state utilizzate nel risolvere questo task come feature extractor. Sarebbe interessante effettuare un confronto fra un utilizzo di quel tipo, contro una CNN solo ai fini della classificazione e

combinata con un ViT come encoder, per verificare se partire dalle mappe di attention porta a dei vantaggi rispetto che semplicemente utilizzare le immagini originali.

Un problema che potrebbe manifestarsi con la self-attention è che DINOv2 è stato pre-allenato a riconoscere prevalentemente immagini dotate di un soggetto, e per questo motivo, le mappe prodotte su un dataset come KU-PCP, che nella maggior parte dei casi è privo di soggetti in primo piano, potrebbero non essere buone come sperato, soprattutto quando si tratta di leading lines. Questa ipotesi viene parzialmente smentita da una analisi qualitativa delle mappe sul dataset (Figura 7.5). In buona parte dei casi, tramite la self-attention si riesce a distinguere in maniera sufficientemente chiara la classe di ground truth. Esistono però diversi casi in cui l'attention cerca esplicitamente un soggetto nell'immagine, anche quando non esiste, ignorando il contesto generale, come in Figura 7.6. Con un eventuale finetuning, le mappe potrebbero migliorare sotto questo aspetto. In altri casi, la self-attention rischia di risultare troppo rumorosa affinchè la CNN possa riconoscerne chiaramente le classi. Per questo, si potrebbe pensare di applicare un filtro di *smoothing* alle immagini in input al modello, per eliminare parte del dettaglio superfluo e permettere al ViT di concentrarsi sugli elementi più rilevanti nell'immagine, che coincideranno con le parti che sopravvivono alla sfocatura (Figura 7.7).

Un problema che rimane aperto e che dovrà essere esplorato è quello della scelta della testa di self-attention da utilizzare come immagine in input alla CNN. Nell'architettura di un ViT, il calcolo della self-attention viene fatto diverse volte per ciascun transformer block che compone l'encoder (Figura A.2), compreso l'ultimo blocco, da cui estraiamo la mappa finale di self-attention. Nel caso di DINOv2, i ViT ripetono 6 volte il calcolo. Per questo motivo, abbiamo 6 possibili mappe fra cui poter scegliere. Come si nota in Figura 7.8, le differenze fra ciascuna sono importanti e la scelta della immagine migliore impatterà sicuramente sulle prestazioni del classificatore. Un approccio potrebbe essere trattarle come una unica immagine a 6 canali da dare in input alla CNN e verificare che la quantità di informazioni fornite non sia eccessiva e che le mappe più rumorose non inquinino l'output che si avrebbe utilizzando solo quelle più riconoscibili.

7.2.4 Segmentazione delle leading lines

Una volta potenziata la classificazione della composizione con gli approcci menzionati, uno sviluppo interessante potrebbe essere quello di procedere alla segmentazione delle leading lines e delle bounding box dei soggetti nell'immagine, come in Figura 1.2.

Ci sono già stati dei tentativi in letteratura di risolvere questo task, come "*Semantic line detection and its applications*" [15], che introduce SEL (*semantic line dataset*), formato da 1750 immagini annotate con le coordinate del punto di inizio e il punto di fine di ciascuna linea. Il metodo da loro applicato si basa interamente su una CNN. Il vantaggio che potremmo avere usando DINOv2, rispetto ad approcci tradizionali di deep learning, è ancora una volta nelle mappe di self-attention. Come visto in Figura 7.5, l'attention produce una semplificazione notevole dell'immagine, che spesso mette in risalto i contorni dei soggetti, ma anche la separazione fra aree semantiche diverse (Figura 7.8, teste 1 e 3), che può coincidere proprio con una leading line. Una base da cui partire potrebbe essere passare una di queste mappe all'interno di una CNN, come già ipotizzato, e una *regression head* che produca le due coppie di coordinate dei punti per cui passa una linea.

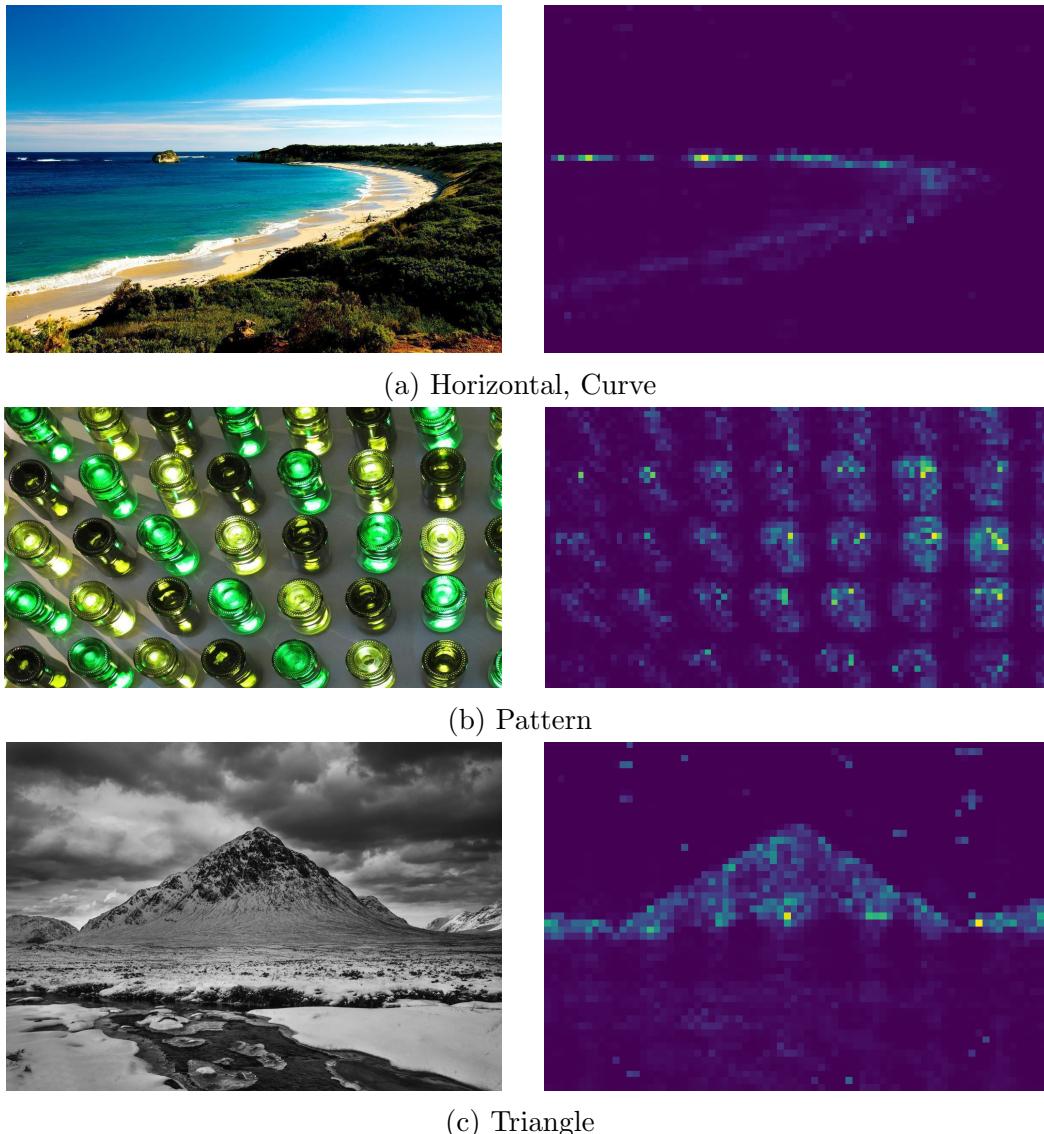


Figura 7.5: Alcuni esempi di immagini di KU-PCP le cui mappe di attention permettono di individuare chiaramente la classe assegnata. Sono indicate le classi di ground truth.



Figura 7.6: Classe *vertical*. Si nota come l'attention si concentri sul ponte che collega i due grattacieli, ma ignori tutti gli altri edifici circostanti. Questo potrebbe rendere complicato il riconoscimento della sua classe.

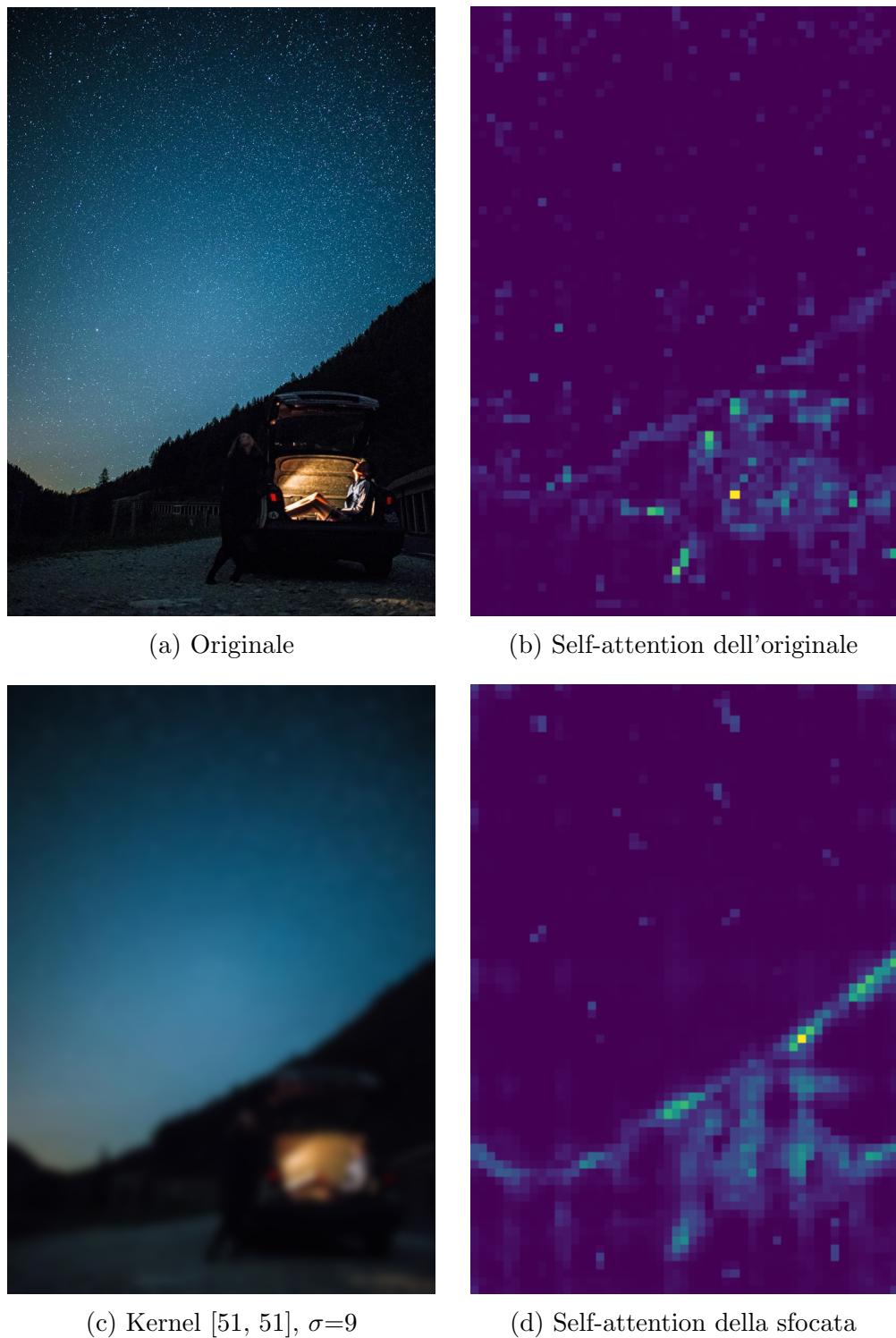


Figura 7.7: Le classi di ground truth sono *RoT*. e *Diagonal*. Dalla mappa originale è difficilmente possibile distinguere le due classi, specialmente *Diagonal*. Una CNN potrebbe fare fatica a causa del rumore che circonda gli elementi chiave. Applicando uno smoothing all’immagine originale e ricalcolando la self-attention su di essa, la mappa prodotta risulta essere più chiara.

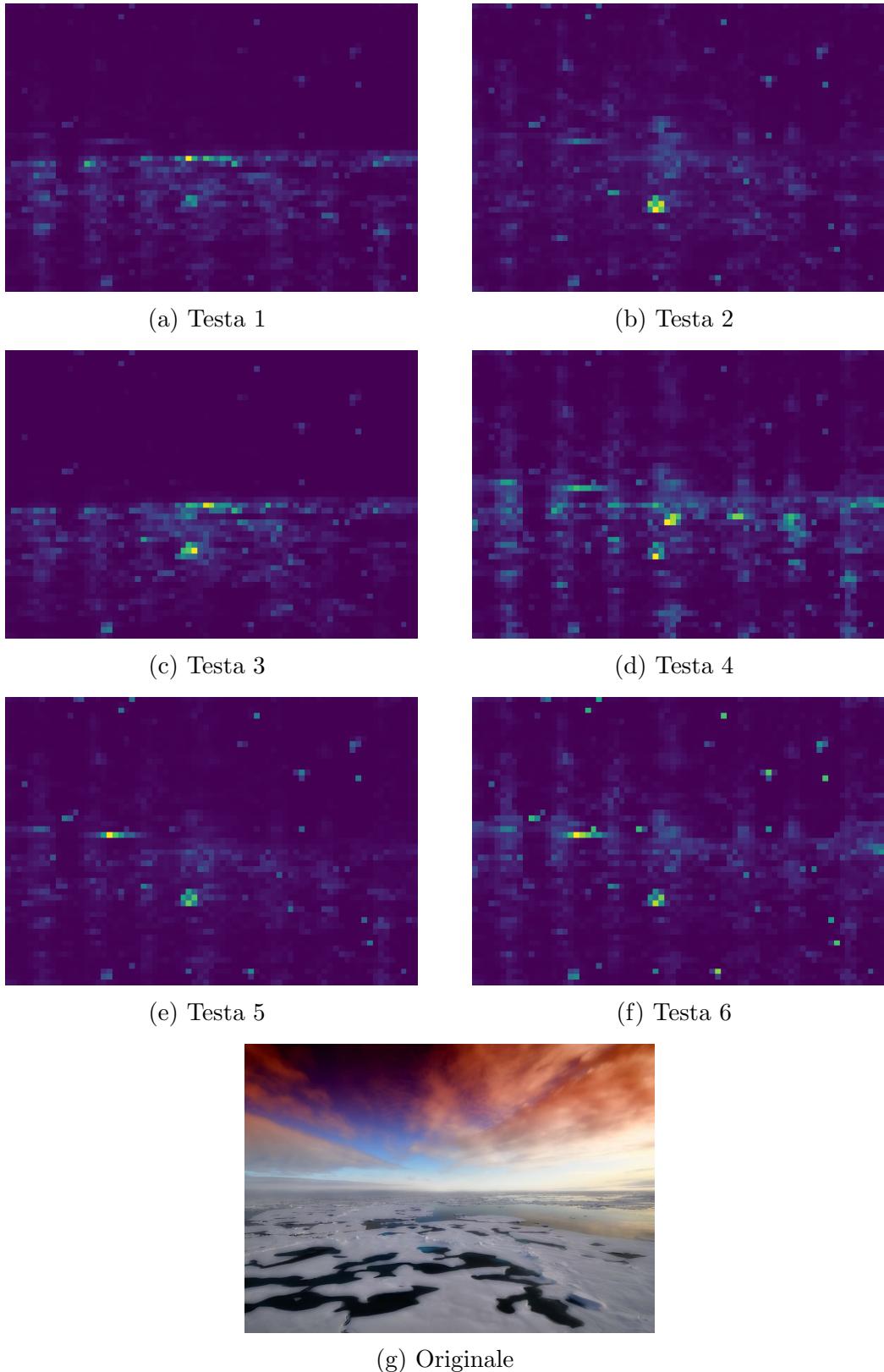


Figura 7.8: Tutte le teste di self-attention dell'ultimo transformer block per un'immagine di KU-PCP, con ground truth *Horizontal*. In queste casu le migliori sono la prima e la terza, dalle altre è sostanzialmente impossibile dedurre la classe. Non è sempre vero, per altre immagini la prima e terza testa potrebbero essere le peggiori.

Riferimenti

Bibliografia

- [1] Randall Balestrieri et al. “A Cookbook of Self-Supervised Learning”. In: *ArXiv* abs/2304.12210 (2023) (cit. a p. 13).
- [2] Mathilde Caron et al. “Emerging properties in self-supervised vision transformers”. In: *Proceedings of the IEEE/CVF international conference on computer vision*. 2021, pp. 9650–9660 (cit. alle pp. 22, 32, 59).
- [3] Ting Chen et al. “A simple framework for contrastive learning of visual representations”. In: *International conference on machine learning*. PMLR. 2020, pp. 1597–1607 (cit. a p. 16).
- [4] Jacob Devlin et al. “BERT: Pre-training of deep bidirectional transformers for language understanding”. In: *arXiv preprint arXiv:1810.04805* (2018) (cit. alle pp. 25, 69).
- [6] Alexey Dosovitskiy et al. “An image is worth 16x16 words: Transformers for image recognition at scale”. In: *arXiv preprint arXiv:2010.11929* (2020) (cit. alle pp. 32, 69).
- [8] Jean-Bastien Grill et al. “Bootstrap your own latent-a new approach to self-supervised learning”. In: *Advances in neural information processing systems* 33 (2020), pp. 21271–21284 (cit. a p. 20).
- [9] Kaiming He et al. “Deep residual learning for image recognition”. In: *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2016, pp. 770–778 (cit. a p. 16).
- [11] Max Jaderberg et al. “Spatial Transformer Networks”. In: *Advances in Neural Information Processing Systems*. A cura di C. Cortes et al. Vol. 28. Curran Associates, Inc., 2015 (cit. a p. 9).
- [12] Li Jing et al. “Understanding dimensional collapse in contrastive self-supervised learning”. In: *arXiv preprint arXiv:2110.09348* (2021) (cit. a p. 18).
- [14] Jun-Tae Lee et al. “Photographic Composition Classification and Dominant Geometric Element Detection for Outdoor Scenes”. In: *Journal of Visual Communication and Image Representation* 55 (mag. 2018) (cit. alle pp. 7, 8, 35, 48).
- [15] Jun-Tae Lee et al. “Semantic line detection and its applications”. In: *Proceedings of the IEEE International Conference on Computer Vision*. 2017, pp. 3229–3237 (cit. a p. 60).
- [17] Tomas Mikolov et al. “Efficient estimation of word representations in vector space”. In: *arXiv preprint arXiv:1301.3781* (2013) (cit. a p. 67).

- [18] Maxime Oquab et al. “Dinov2: Learning robust visual features without supervision”. In: *arXiv preprint arXiv:2304.07193* (2023) (cit. alle pp. 5, 28, 59).
- [20] Ed Pizzi et al. “A self-supervised descriptor for image copy detection”. In: *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*. 2022, pp. 14532–14542 (cit. a p. 28).
- [25] Ashish Vaswani et al. “Attention is all you need”. In: *Advances in neural information processing systems* 30 (2017) (cit. alle pp. 22, 66, 69).
- [26] Yaoting Wang et al. “Spatial-invariant convolutional neural network for photographic composition prediction and automatic correction”. In: *Journal of Visual Communication and Image Representation* 90 (2023), p. 103751. ISSN: 1047-3203 (cit. alle pp. 8, 38, 48).
- [27] Chian-Li Wen e Tsorng-Lin Chia. “The fuzzy approach for classification of the photo composition”. In: *2012 International Conference on Machine Learning and Cybernetics*. Vol. 4. 2012, pp. 1447–1453 (cit. alle pp. 4, 7).
- [28] Zhaoran Zhao et al. “Self-supervised Photographic Image Layout Representation Learning”. In: *arXiv preprint arXiv:2403.03740* (2024) (cit. alle pp. 10, 11, 35, 38, 48).
- [29] Jinghao Zhou et al. “ibot: Image bert pre-training with online tokenizer”. In: *arXiv preprint arXiv:2111.07832* (2021) (cit. a p. 25).

Siti

- [5] *DINOv2 GitHub*. URL: <https://github.com/facebookresearch/dinov2> (cit. a p. 31).
- [7] *Flickr*. URL: <https://www.flickr.net> (cit. a p. 35).
- [10] *ImageNet*. URL: <https://www.image-net.org> (cit. a p. 8).
- [13] *KU-PCP*. URL: https://mcl.korea.ac.kr/research_page/#:~:text=Semantic%20Lines%20and%20Photographic%20Composition (cit. a p. 4).
- [16] *LODB*. URL: https://drive.google.com/file/d/1EfzI04k9TsSsOBwza-EsV0hqfLC2WuN_/view (cit. alle pp. 4, 10).
- [19] *Photo.net*. URL: <https://www.photo.net> (cit. a p. 35).
- [21] *Reddit r/photocritique*. URL: <https://www.reddit.com/r/photocritique/> (cit. a p. 58).
- [22] *Reddit r/photographs*. URL: <https://www.reddit.com/r/photographs/> (cit. a p. 58).
- [23] *Reddit r/pics*. URL: <https://www.reddit.com/r/pics/> (cit. a p. 58).
- [24] *Scikit-learn average_precision_score*. URL: https://scikit-learn.org/stable/modules/generated/sklearn.metrics.average_precision_score.html (cit. a p. 44).

A

Transformers e Vision Transformers

Questa appendice è dedicata a dettagliare maggiormente temi riguardanti transformers e vision transformers (ViT). In particolare, il ViT è la rete che ricopre il ruolo di encoder nell’architettura di DINOv2 (Figura 3.14) e i quali outputs utilizziamo in fase di sperimentazione. Si andranno ad esporre gli aspetti concettuali e l’intuizione dietro il funzionamento dei due, senza scendere nel dettaglio algebrico.

L’introduzione dei transformers nel 2017 ha cambiato il panorama della ricerca in ambito di *natural language processing*, e non solo. Questo nuovo approccio all’intelligenza artificiale, in combinazione con altri fattori come l’aumento della disponibilità e potenza di calcolo delle GPU, ha portato negli ultimi anni ad una rivoluzione che ci ha condotti fino a modelli moderni come GPT, superando le performance di tradizionali approcci di deep learning.

A.1 Transformers

I transformers vengono introdotti per la prima volta nel 2017 dal paper ”*Attention is all you need*” [25]. Questa nuova rete si basa interamente sul concetto di *attention*, applicato al task di traduzione dall’inglese al tedesco.

La struttura di un transformer è illustrata in Figura A.1. Si compone di un insieme di *encoders*, seguiti da un insieme *decoders*. Generalmente, il numero di encoders e decoders scelto è di 6 ciascuno, ma c’è spazio per la sperimentazione. Il blocco di encoders ha il compito di trasformare una sequenza di embeddings (x_1, \dots, x_n) , che identificano la sequenza di tokens della frase, in una sequenza di rappresentazioni $z = (z_1, \dots, z_n)$. Il blocco di decoding, invece, a partire da z genera in output una sequenza di simboli (y_1, \dots, y_m) . Questo viene fatto mappando il vettore di features prodotto dal decoder in un nuovo vettore, lungo quanto il numero di parole nel dizionario di vocaboli che la rete è in grado di predirre, tramite un layer lineare, e utilizzando un softmax per scegliere quale sia la parola più probabile a seguire quella correntemente analizzata. Ad ogni step successivo, il modello consuma i simboli precedentemente generati come input aggiuntivo nel generare altro testo. La loss utilizzata è una cross-entropy.

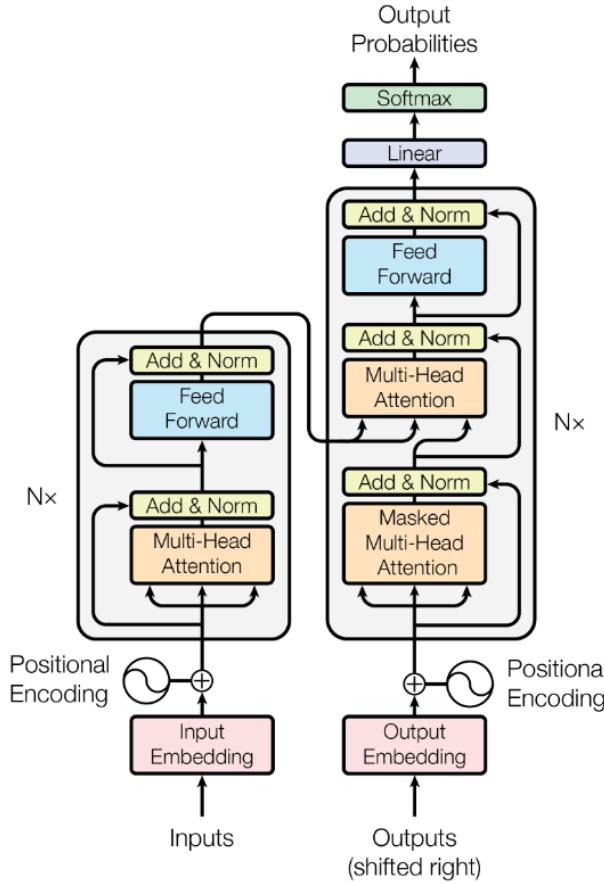


Figura A.1: Architettura di un transformer.

A.1.1 Preparare l'input

Data una frase da tradurre, questa viene suddivisa in *tokens*, che possono essere una singola parola, parti di frasi, punteggiatura. Ciascuno di essi viene trasformato in un vettore numerico univoco per poter essere passato alla rete. Questa traduzione può essere fatta tramite approcci come *word2vec* ([17]), che consiste in una rete neurale poco profonda allenata a mappare parole in uno spazio continuo in cui parole simili sono vicine fra loro. Al vettore risultante viene poi aggiunto un ulteriore vettore, detto *positional encoding*, che codifica la posizione del token nella frase. Questo è importante, in quanto i transformers processano tutti i tokens parallelamente, che è una delle forze di questa architettura, ma questo cancella la nozione naturale di ordine delle parole nella frase.

A.1.2 Encoder

Ogni encoder si compone di 2 sublayers. Il primo è un blocco di *self-attention*, mentre il secondo è una semplice rete fully-connected. Viene utilizzata anche una connessione residuale attorno a ciascuno dei due sublayers, seguita da una layer normalization, la quale ha lo scopo di migliorare velocità e stabilità del training, normalizzando le attivazioni ricevute lungo la dimensione delle features. L'output di ogni sotto-layer è quindi $\text{LayerNorm}(x + \text{Sublayer}(x))$, dove $\text{Sublayer}(x)$ è la funzione implementata dal sublayer stesso. Questo facilita e stabilizza l'addestramento, mitigando fenomeni come il *vanishing gradient*.

shing gradient, quando il gradiente diventa troppo piccolo per permettere apprendimento significativo. Tutti gli outputs, e anche gli embeddings, sono vettori di dimensione 512.

A.1.3 Decoder

Il decoder si compone a sua volta degli stessi due sublayers di cui si compone l'encoder, fra i quali se ne inserisce un terzo adibito a ricevere e calcolare la self-attention dell'output dell'encoder. Le connessioni residuali vengono mantenute.

Se nell'encoder l'input del primo sublayer era la frase da tradurre, tokenizzata e trasformata in embeddings, nel decoder l'input è costituito dalla frase già tradotta, che segue gli stessi passi di trasformazione in vettore continuo prima di entrare nella rete. Per questo motivo, il sublayer che lo riceve deve effettuare un mascheramento della frase, in particolare degli embeddings dei tokens che ancora non sono stati predetti dalla rete (si pongono a $-\infty$). Questo evita che la rete possa conoscere quale sia il risultato della predizione del token corrente prima ancora di averla effettuata, assicurando che una predizione in posizione i dipenda solo dai token di output precedenti a i . È necessario fornire alla rete le traduzioni corrette per i tokens già predetti, in fase di training, per evitare che gli errori di predizione per tokens iniziali nella frase possano propagarsi anche a parole successive, portando a prestazioni pessime.

A.1.4 Self-attention

La *self-attention* è il meccanismo attraverso il quale la rete è in grado di concentrarsi su parti specifiche della frase, assegnando un diverso peso a diversi tokens in base alla loro rilevanza al fine di comprendere il significato del testo. Permette di analizzare le **dipendenze fra parole** tenendo conto delle **relazioni semantiche e sintattiche**, indipendentemente dalla loro posizione della sequenza. Il calcolo è il seguente:

$$\text{Attention}(Q, K, V) = \text{softmax}\left(\frac{QK^T}{\sqrt{d_k}}\right)V$$

dove Q (query), K (key) e V (value) corrispondono allo stessa matrice di embeddings di input, nel primo sublayer, o all'output ricevuto dagli strati precedenti altrimenti. Sostanzialmente, si fa una somma pesata di V , dove il peso è dato dalla compatibilità fra Q e K , la compatibilità fra una parola e tutte le altre parole nella frase, quanto strettamente siano in relazione fra loro. Il valore di compatibilità viene scalato da un fattore $\sqrt{d_k}$, che corrisponde alla radice della dimensione del vettore degli embeddings, che aiuta a stabilizzare il calcolo dei pesi. A livello matematico, il risultato dell'operazione sarà una matrice la cui i -esima riga esprime quanto il token i -esimo sia correlato a ciascuno degli altri tokens nella frase. Valori più alti corrispondono a un rapporto più significativo fra i due tokens.

A.1.5 Multi-head attention

Si introduce anche un meccanismo di *multi-head attention*. Viene realizzato **ripetendo parallelamente il calcolo della self-attention** un numero h di volte e introducendo un layer lineare di proiezione per ciascun Q, K, V , con pesi diversi e appresi in training. Gli output di ciascun blocco self-attention vengono poi concatenati fra loro e il risultato

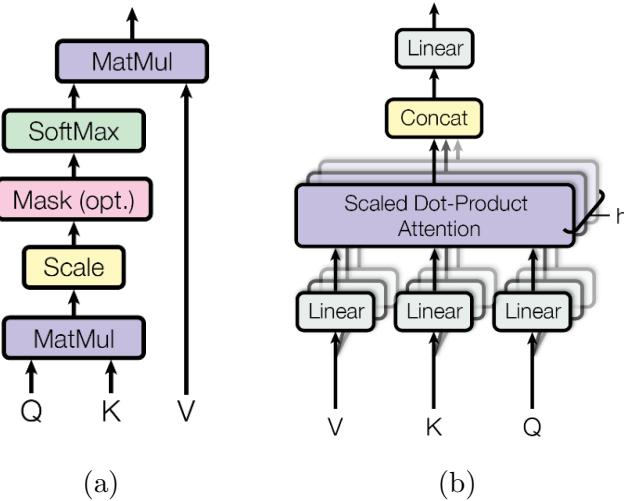


Figura A.2: Workflow per il calcolo della self-attention: (a) scaled dot-product attention, (b) multihead-attention

viene moltiplicato per un'altra matrice di pesi aggiornata durante la back-propagation, prima di continuare verso il sublayer di aggiunta dei residui e layer normalization.

Il motivo per cui viene utilizzata questa tecnica è che utilizzando più teste di attention, il modello può considerare diverse rappresentazioni contemporaneamente, ognuna che pone attenzione su diverse parti della sequenza. All'interno di una frase, una testa potrebbe concentrarsi sul rapporto fra soggetto e verbo, mentre un'altra testa potrebbe concentrarsi sul rapporto fra soggetto e aggettivi ad esso correlati, e così via. Permette alla rete di comprendere contesti complessi.

A.2 Vision Transformers (ViT)

I *vision transformers* [6] nascono nel 2020 dall'osservazione che il concetto di transformer può essere sfruttato anche in ambito di computer vision, su task di image classification, e competere con le prestazioni di modelli tradizionali come le CNN. Si tenta di applicare la stessa struttura del transformer, come descritto da [25], ad un workflow che come input prevede delle immagini, apportando il minor numero di modifiche possibili. Questo viene fatto **suddividendo** ciascuna **immagine** in un **insieme di patches** di dimensione fissa e fornendo in input alla rete una sequenza di embeddings rappresentanti ciascun patch. Queste rappresentazioni saranno trattate allo stesso modo dei tokens utilizzati in NLP e prodotte da un layer lineare.

A.2.1 Token [CLS]

Una novità introdotta dai ViT rispetto ai transformers è l'aggiunta del **token [CLS]**, teorizzata prima ancora da BERT [4]. [CLS] è un token **completamente appreso** durante l'addestramento della rete, che viene **aggiunto all'inizio della sequenza di patches di input**. Durante il passaggio attraverso i livelli del transformer, [CLS] interagisce con gli altri tokens tramite il meccanismo di self-attention, e durante questo processo **accumula informazioni globali riguardo l'intera immagine**. Di fatto, il suo ruolo è quello di aggregare le informazioni racchiuse nella sequenza di patches. Una differenza

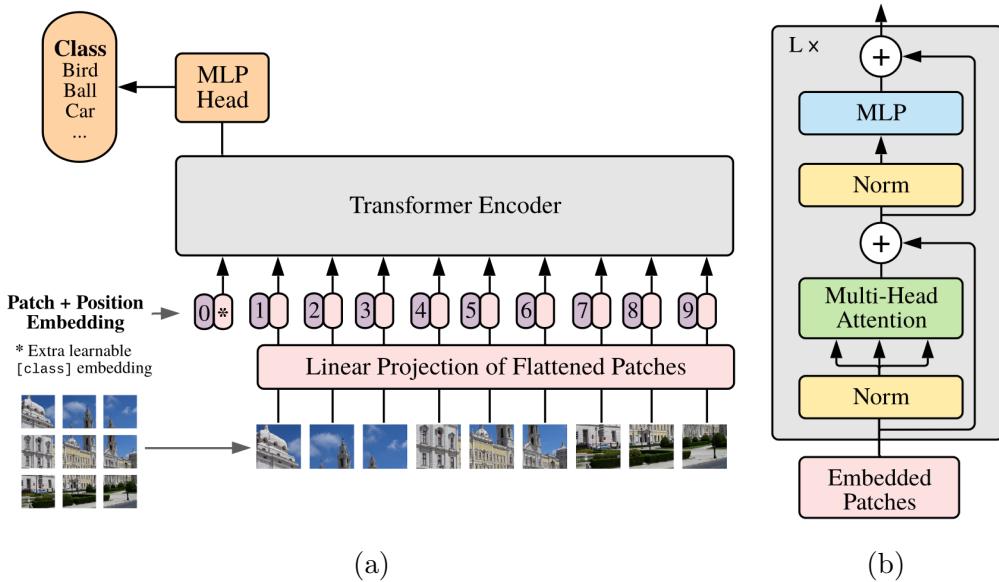


Figura A.3: (a) Architettura di un ViT, (b) Encoder

importante nell’architettura dei ViT rispetto ai transformers è l’assenza di un decoder, essendo l’obiettivo della rete quello della image classification, e non di generare del nuovo contenuto. Per questo, si sceglie di attaccare un MLP proprio all’output di [CLS], grazie alla sua caratteristica di sintesi dell’immagine. La testa di proiezione mappa il vettore ricevuto su un nuovo vettore di attivazioni grande quanto il numero di classi da riconoscere, a cui si applica un softmax e la loss scelta.

A.2.2 Attention

Il concetto di attention rimane il nucleo su cui si fonda anche il ViT. Se nel transformer serviva a individuare le relazioni fra tokens all’interno della frase, in questo caso permette di rilevare il rapporto fra patches nell’immagine. Anche in questo caso, per ottenere le mappe di self-attention (Figura 3.8) si deve accedere ad una matrice all’interno della quale la i -esima riga rappresenta quanto l’ i -esimo patch è correlato a tutti gli altri patch in cui è suddivisa l’immagine. Ad esempio, nella riga 0 troviamo al self-attention riferita al patch [CLS].

A.2.3 Tipologie

Esistono diversi tipi di ViT:

- ViT-S (small):
 - Numero di *transformer blocks*: 12
 - Dimensione del vettore di embeddings: 384
 - Numero di self-attention heads: 6
- ViT-B (base):
 - Numero di *transformer blocks*: 12
 - Dimensione del vettore di embeddings: 768

- Numero di self-attention heads: 12
- ViT-S (large):
 - Numero di *transformer blocks*: 24
 - Dimensione del vettore di embeddings: 1024
 - Numero di self-attention heads: 16

Con il numero di transformer blocks si intende il numero di encoders che formano la rete, come visto in Figura A.1.

Inoltre, con la notazione ViT/ N si specifica le immagini di input vengono suddivise in patch di dimensione $N \times N$ pixels.