
**UNIVERSITÀ DEGLI STUDI DI CATANIA**

Dipartimento di Matematica e Informatica

Corso di Laurea Magistrale in Informatica

*Martin Gibilterra, Sabrina Nicosia*


# Confronto tra modelli basati sui transformers per il task di Action Video Classification

Relazione Progetto

Docente: Giovanni Maria **FARINELLA**


Anno Accademico 2023 - 2024

# Abstract

Nel seguente lavoro viene affrontato il task di Action Video Classification, analizzando e confrontando due modelli basati sui transformers: *Multiscale Vision Transformers* e *Video Swin Transformer*. A seguito dell'esposizione dell'architettura base dei transformers e dei due modelli in esame, viene effettuato un fine-tuning a partire dai modelli pre-allenati forniti dalla libreria Pytorch, su un nuovo dataset (Charades). Ciò avviene passando a ciascun modello una clip di 3 secondi estratta da un determinato video del training set. Ultimato il processo di addestramento dei modelli, quest'ultimi saranno sottoposti ad una fase di test nella quale verranno calcolate diverse metriche che permettono di verificare l'efficacia di ciascun modello nel predire correttamente le azioni che avvengono all'interno di ciascuna clip.

# Indice

**1 Video Classification** **3**
1.1 Introduzione . . . . . 3
1.2 Acquisizione dati . . . . . 6

**2 Modelli analizzati: Transformers** **7**
2.1 Multiscale Vision Transformers . . . . . 9
2.1.1 Multi Head Pooling Attention . . . . . 9
2.1.2 Architettura . . . . . 10
2.1.3 MViTv2 . . . . . 13
2.2 Video Swin Transformer . . . . . 15
2.2.1 Architettura . . . . . 15
2.2.2 3D Shifted Window based MSA Module . . . . . 17

**3 Fine-tuning dei modelli** **19**
3.1 Caricamento e pre-processing dei dati . . . . . 20
3.2 Processo di training . . . . . 22
3.3 Risultati ottenuti . . . . . 24

**4 Benchmarking** **26**
4.1 Metriche . . . . . 26
4.2 Risultati ottenuti . . . . . 27

**Conclusione** **30**

**Bibliografia** **32**

# Capitolo 1

# Video Classification

## 1.1 Introduzione

Il task di Video Classification è un processo in cui un sistema assegna automaticamente una o più etichette a un video in base al suo contenuto. In particolare, prevede l'analisi dei frame del video per identificare e classificare le informazioni visive e talvolta uditive. Si tratta di un task fondamentale nel campo della computer vision, in quanto consente ai sistemi di comprendere e interpretare i dati video per varie applicazioni. Il suo scopo può essere più generico, come l'identificazione di categorie come: sport, notizie o musica; oppure, le etichette possono essere più specifiche, come nel caso di attività quali corsa, cucina o ballo. A tal scopo, i modelli prendono in input singole immagini estratte dal video nel tempo, ovvero frame, al fine di classificarli con un'etichetta di classe che rappresenta l'azione o il contesto generico presente.

Nel corso del tempo sono stati adottati diversi modelli in grado di risolvere il task in esame. Di seguito un elenco delle architetture principali:

- **Convolutional Neural Networks (CNNs)**: utilizzate principalmente per estrarre feature spaziali da singoli frame di un video. Sono in grado di identificare pattern come bordi e forme all'interno delle immagini. Applicando una serie di livelli convoluzionali, livelli di pooling e livelli completamente connessi, riescono ad apprendere rappresentazioni gerarchiche dei dati di input, andando ad estrasse man mano pattern più significativi. Nell'approccio più semplice, vengono utilizzate per elaborare ogni frame in modo indipendente in modo da estrarre per ciascuno di essi un'etichetta di classe. Oppure, possono essere combinate le informazioni temprali prima (Early fusion) o dopo (Late Fusion) l'applicazione della CNN in modo da ottenere un singolo valore in output.

- **Recurrent Neural Networks (RNNs)**: sviluppate per catturare le dipendenze temporali presenti nei dati sequenziali; caratteristica che le rende adatte anche per l'analisi di sequenze video. Esse elaborano le sequenze in input un elemento alla volta, mantenendo uno stato nascosto che conserva informazioni sugli elementi precedenti, permettendo così di comprendere il contesto nel tempo. Tuttavia, le RNN tradizionali possono soffrire del cosiddetto *vanishing gradient*, che limita la loro capacità di apprendere dipendenze a lungo termine. Per far fronte a ciò, è stato sviluppato un tipo avanzato di RNN: Long Short-Term Memory (LSTM). Quest'ultime consentendo alla rete di apprendere e mantenere dipendenze a lungo termine. La loro struttura, che include celle di memoria e meccanismi di gating, permette di catturare informazioni rilevanti su lunghi intervalli di tempo, rendendole ancora più efficaci nell'analisi di sequenze di dati.

- **3D Convolutional Neural Networks (3D CNNs)**: estendono le tradizionali CNN 2D per poter catturare caratteristiche sia spaziali che temporali, operando su un cubo come input, che rappresenta pile di frame. Esse applicano convoluzioni 3D, che coprono sia le dimensioni spaziali (altezza e larghezza) che la dimensione temporale (profondità). Ciò consente loro di apprendere direttamente le features relative al movimento. Tale meccanismo risulta efficace per il riconoscimento delle azioni e altre attività in cui la comprensione del movimento tra un frame e il successivo è fondamentale.

- **Transformers**: inizialmente progettati per l'elaborazione del linguaggio naturale (NLP), recentemente sono stati adattati al fine di essere usati per altri task, tra cui anche la classificazione video, essendo in grado di catturare le dipendenze e le interazioni a lungo termine tra i vari frame. A tal scopo, utilizzano meccanismi di self-attention per valutare l'importanza delle diverse parti della sequenza in input, riuscendo così a catturare dipendenze complesse senza essere vincolati dall'elaborazione sequenziale; come invece avveniva nelle RNN.

Negli ultimi anni, nel campo della computer vision è avvenuto un significativo passaggio dalle reti neurali convoluzionali (CNN) alle architetture dei transformer. Questa tendenza è iniziata con l'introduzione dei Vision Transformer (ViT) [1], che modellano globalmente le relazioni spaziali su patch di immagini non sovrapposte con il tradizionale encoder Transformer. Il grande successo di ViT sulle immagini ha portato alla ricerca di architetture simili anche per il riconoscimento di task su video.

Sebbene le CNN siano state a lungo le pionere dell'elaborazione di immagini e video grazie alla loro abilità nell'estrazione di features locali, si trovano in difficoltà nell'affrontare dinamiche temporali e nel cogliere informazioni contestuali tra i frame. Dall'altra parte, i transformers hanno dimostrato la loro superiorità nella gestione di queste sfide, rivoluzionando così l'approccio utilizzato nell'affrontare diverse attività, tra cui vi è anche la classificazione video.

Tuttavia, i transformers, così come le CNN 3D, possono essere computazionali onerosi e richiedere risorse hardware importanti per l'addestramento e l'inferenza. Inoltre, un modello abbastanza complesso spesso richiede un set di dati ampio e diversificato per far sì che il modello riesca a generalizzare bene ed evitare l'overfitting. Data la complessità del modello, comprendere e interpretare le decisioni che prende non è un lavoro elementare, ciò fa sì che la diagnosi degli errori e il miglioramento delle prestazioni possano risultare complessi. Quindi, la selezione della tipologia di modello più adatto per la classificazione video dipende in larga misura dai requisiti specifici del task da svolgere, dalla natura dei dati in esame e dalle risorse computazionali disponibili. Dato che ogni tipologia di modello ha i suoi punti di forza e di debolezza, spesso gli approcci ibridi che combinano diversi modelli forniscono migliori prestazioni per attività di classificazione video complesse, come la combinazione di una CNN con una RNN.

La classificazione video ha un'ampia gamma di applicazioni in vari settori, ognuna delle quali sfrutta la capacità di analizzare e interpretare automaticamente i contenuti video:

- **Sistemi di raccomandazione**: classificando i contenuti dei video, sono in grado di suggerire contenuti in linea con le preferenze dell'utente. Questo permette di mostrare i video più pertinenti agli interessi precedentemente manifestati, migliorando così l'esperienza dell'utente e aumentando la probabilità di coinvolgimento.

- **Sicurezza**: fornisce un ausilio ai sistemi di sorveglianza che, attraverso la classificazione delle attività, riescono a rilevare comportamenti sospetti, accessi non autorizzati o altre anomalie in tempo reale.

- **Sport**: andando a classificare diverse strategie di gioco, azioni e movimenti dei giocatori nei video sportivi, è possibile generare statistiche di vario genere, ma anche fornire un supporto per lo sviluppo di nuove strategie di gioco più mirate e performanti.

Questi sono alcuni esempi, ma al giorno oggi il task di classificazione video funge da ausilio in svariati altri settori, come: campo medico, guida autonoma, agricoltura e campo industriale.

**Action Video Classification** rappresenta una sottocategorìa della classificazione video incentrata sull'identificazione e la categorizzazione delle azioni e delle attività compiute dalle persone presenti nelle sequenze video. Questo processo prevede l'analisi di informazioni spaziali e temporali per riconoscere azioni specifiche, come camminare, correre, saltare o ballare; ma anche attività ancora più dettagliate come, ad esempio, guardare fuori dalla finestra.

## 1.2 Acquisizione dati

Per lo svolgimento del seguente progetto è stato preso in esame il dataset Charades, introdotto nel seguente paper [2], al fine di effettuare il task di Action Video Classification. Esso è stato progettato per il riconoscimento e la comprensione delle azioni compiute in ambienti comuni; in particolare, contiene video di persone che svolgono attività quotidiane e interagiscono con oggetti in ambienti interni. Esso comprende oltre 9.800 video, registrati da volontari, che presentano in genere una sequenza di attività e interazioni con vari oggetti, che simulano scenari domestici realistici. È stato progettato per sfidare i modelli in termini di riconoscimento di attività complesse, comprensione delle dinamiche temporali e gestione delle occlusioni e delle variabili condizioni di illuminazione, così come presenza di sfondi diversi, variazioni nell'aspetto e nel movimento delle persone e il verificarsi simultaneo di più attività.

I video presenti sono ampiamente annotati con diversi tipi di etichette, ma, per lo scopo del progetto verranno utilizzate esclusivamente le etichette relative alle azioni compiute, per un totale di 157 classi. Tuttavia, data la complessità del task, in seguito verrà effettuato un raggruppamento delle classi in modo da ridurne il numero, esponendo e motivando il criterio di aggregazione applicato.

# Capitolo 2

# Modelli analizzati: Transformers

I transformers sono stati introdotti nell'articolo *Attention is All You Need* [3] e da allora hanno riscosso un enorme successo, non solo nel campo dell'elaborazione del linguaggio naturale (NLP), ma anche in svariati altri settori, tra cui la classificazione video.

**Architettura** I transformers sono composti da più blocchi sovrapposti, la cui componente principale è il meccanismo di self-attention, che consente al modello di pesare opportunamente l'importanza di ogni token rispetto a ogni altro token presente nella sequenza in input. Per far ciò, per ogni token in input, vengono calcolati tre valori: *query*, *key* e *value*. L'output di ogni token in input è calcolato andando a moltiplicare il suo valore di *query* per il valore di *key* degli altri token. I valori ottenuti vengono passati ad un softmax, in modo da ottenere valori che siano nel range $[0, 1]$ ed aventi somma 1, che rappresentano il peso che il token in esame darà a ciascun altro token nel calcolo del suo output. Tali valori vengono moltiplicati per il rispettivo *value* di ciascun token della sequenza, ed infine i valori ottenuti verranno sommati. La formula generale che rappresenta tale meccanismo è la seguente:

$$\text{Attention}(Q, K, V) = \text{SoftMax}\left(\frac{QK^T}{\sqrt{d}}\right)V \tag{2.1}$$

dove $Q, K, V$ sono rispettivamente le matrici di *query*, *key* e *value*, e $d$ rappresenta la dimensione delle feature di *key*.

Questo meccanismo è risultato essere molto efficace per catturare relazioni intrinseche e dipendenze all'interno della sequenza. Tuttavia, le dipendenze presenti tra i token posso essere di vario tipo, perciò viene replicato questo

procedimento su più *head*, ovvero teste, eseguite in parallelo, che apprendono contemporaneamente diversi aspetti dei dati di input, migliorando la capacità del modello di comprendere pattern più complessi. Questo meccanismo prende il nome di multi-head self-attention, il cui output viene calcolato andando a concatenare gli output ottenuti da ogni *head* ed effettuando infine una trasformazione lineare. Formalmente l'output viene calcolato nel seguente modo:

$$\text{MultiHead}(Q, K, V) = \text{Concat}(\text{head}_1, \text{head}_2, \ldots, \text{head}_h)W^O \tag{2.2}$$

I transformers non tengono in considerazione l'ordine dei token nel calcolo dell'output, ciò può essere problematico in quanto l'ordine risulta fondamentale in certi ambiti, come nel caso delle parole di un testo. Perciò, viene aggiunto all'input del transformer un riferimento alla posizione espresso come vettore avente la stessa dimensione dell'input embedding, in modo da effettuare una somma con quest'ultimo. Queste rappresentazioni della posizione dei vari token possono essere apprese durante l'addestramento del modello, oppure si possono calcolare mediante funzioni apposite; di solito viene adottata una funzione che effettua una combinazione del seno e del coseno per il calcolo della posizione relativa dei vari tokens.

Dunque, ogni blocco del transformer è composto da un modulo di multi-head self-attention , un layer di normalizzazione, una rete feed-forward e un altro layer di normalizzazione. Inoltre, vengono aggiunte le residual connections che fanno sì che l'input del layer di normalizzazione venga calcolato come la somma dell'output del layer precedente e del suo input originale. Questo permette di mantenere informazioni non alterate lungo il percorso del modello, facilitando l'addestramento e migliorando la propagazione del gradiente.

I trasformatori presentano diversi vantaggi, tra cui: sono in grado di apprendere dipendenze a lungo termine in diverse parti del video, il che è fondamentale per comprendere il contenuto di video complessi; a differenza delle reti neurali ricorrenti (RNN), elaborano tutti i token contemporaneamente ottenendo una maggior efficienza; inoltre, possono prendere in input sequenze aventi dimensioni variabili, il che li rende maggiormente flessibili.

A seguire verranno analizzati due modelli appartenenti alla classe dei transformers per l'esecuzione del task di Action Video Classification: VMiT e Video Swin Transformer. Per ciascuno di essi verrà esposta la struttura generale, spiegate le componenti principali e indicati i miglioramenti introdotti, al fine di comprenderne il funzionamento e i punti di forza di ciascun modello.

## 2.1 Multiscale Vision Transformers

Multiscale Vision Transformers (MViT) [4] è un modello progettato specificamente per l'elaborazione di dati visuali, come immagini e video, che combina l'architettura dei transformers con l'approccio *Multiscale Feature Hierarchies* in cui le features sono rappresentate su più scale. La sua architettura prevede diverse fasi che elaborano l'input su scale diverse. All'inizio la risoluzione dell'input rimane invariata e si hanno un numero ridotto di canali. Man mano che i dati avanzano, il numero di canali aumenta e la risoluzione spaziale diminuisce. Ciò porta alla creazione di una piramide multiscala di features in cui i primi livelli operano ad alta risoluzione spaziale acquisendo informazioni visive più semplici e di basso livello (come bordi e texture); mentre, i livelli più profondi lavorano con una risoluzione spaziale inferiore, ma catturano caratteristiche più complesse e ad alta dimensione (come parti di oggetti). Questo approccio gerarchico consente loro di catturare in modo efficace sia i dettagli fini che i pattern più complessi, raggiungendo prestazioni superiori ed essendo, al tempo stesso, più efficienti in termini di risorse rispetto ad altri modelli di vision transformer. Tuttavia, il vantaggio principale del transformer multiscala nasce dall'estrema natura densa dei segnali visivi, fenomeno che è ancora più pronunciato per segnali visuali spazio-temporali catturati nei video. Un altro notevole vantaggio caratteristico di questi modelli deriva dal fatto che l'informazione temporale ha un impatto significativo nel calcolo dell'output del modello, in quanto, testando il modello mescolando i frame le prestazioni che si ottengono subiscono un netto peggioramento in termini di accuracy, fenomeno che non accade invece con i tradizionali Vision Transformers.

### 2.1.1 Multi Head Pooling Attention

Multi Head Pooling Attention (MHPA) è una variante del meccanismo di self-attention, sviluppata per gestire in modo flessibile diverse risoluzioni all'interno di un transformer block, consentendo al modello di elaborare i dati a vari livelli di dettaglio. Ciò consente al modello di ridurre la risoluzione spaziale mantenendo o aumentando la dimensione del canale man mano che i dati avanzano attraverso i vari livelli. A differenza dell'originale Multi Head Attention (MHA), dove la dimensione dei canali e della risoluzione spazio-temporale rimane costante in tutti gli strati, MHPA include un meccanismo di pooling che riduce la lunghezza della sequenza dell'input, ovvero la risoluzione spazio-temporale concentrandosi sulle caratteristiche essenziali. Seguendo il modulo MHA, vengono calcolati i 3 tensori (Q, K, V) ottenuti come combinazione lineare a partire dall'input, ma quest'ultimi vengono

definiti intermedi dato che su di essi viene poi effettuata un'operazione di *pooling* con un operatore $P$ descritto a seguire.

**Operazione di Pooling** I tensori intermedi vengono quindi sottoposti ad un operazione di pooling mendiante l'operatore $P(\cdot;\Theta)$, che rappresenta il fulcro del modulo MHPA. Esso esegue il pooling sul tensore di input lungo ciascuna delle sue dimensioni. $\Theta$ è un insieme di parametri definito come $\Theta := (k, s, p)$, dove: $k$ è la dimensione del kernel di pooling; $s$ è lo stride; $p$ rappresenta il padding. Il tensore risultante viene appiattito, riducendone così la lunghezza della sequenza, ma conservando la dimensionalità delle features. Di default, il modello utilizza *overlapping* kernels con *shape-preserving* padding $p$ in modo che la lunghezza del tensore in output venga ridotta di un fattore proporzionale alle dimensioni del parametro $s$. In definitiva, il calcolo dell'attenzione, chiamata in questo caso *pooling attenton* 2.1, è calcolata nel seguente modo:

$$PA(\cdot) = \text{Softmax}\left(\frac{P(Q;\Theta_Q)P(K;\Theta_K)^T}{\sqrt{d}}\right)P(V;\Theta_V)$$

I requisiti computazionali e di memoria per i meccanismi di attenzione scalano quadraticamente rispetto alla lunghezza della sequenza. Ciò significa che al raddoppiare della lunghezza della sequenza, la quantità di calcolo e memoria necessaria quadruplica. Applicando il pooling ai tensori *key*, *query* e *value*, la lunghezza della sequenza viene ridotta. Il raggruppamento di questi tensori riduce significativamente la quantità di calcolo e memoria necessaria. Ciò rende il modello maggiormente scalabile ed efficiente per l'elaborazione di sequenze di grandi dimensioni.

### 2.1.2 Architettura

Il modello fa uso soltanato di due tipologie di livelli: MHPA, presentato nella sezione precedente e MLP (Multilayer Perceptron), rete neurale che viene tipicamente utilizzata per l'elaborazione di features dopo il calcolo dell'attenzione. Verrà introdotta inizialmente la struttura del Vision Transformers, dato che il modello in esame possiede le medesime caratteristiche di base.

**Vision Transformer (ViT)** Preso in input un tensore avente dimensione $T \times H \times W$, dove T rappresenta il numero di frame, mentre H e W rispettivamente l'altezza e la lunghezza dei singoli frame, esso viene suddiviso in patch non sovrapposte di dimensione $1 \times 16 \times 16$. Ogni path viene trasformata in un vettore 1-dimensionale a cui viene applicato un layer lineare che lo proietta in

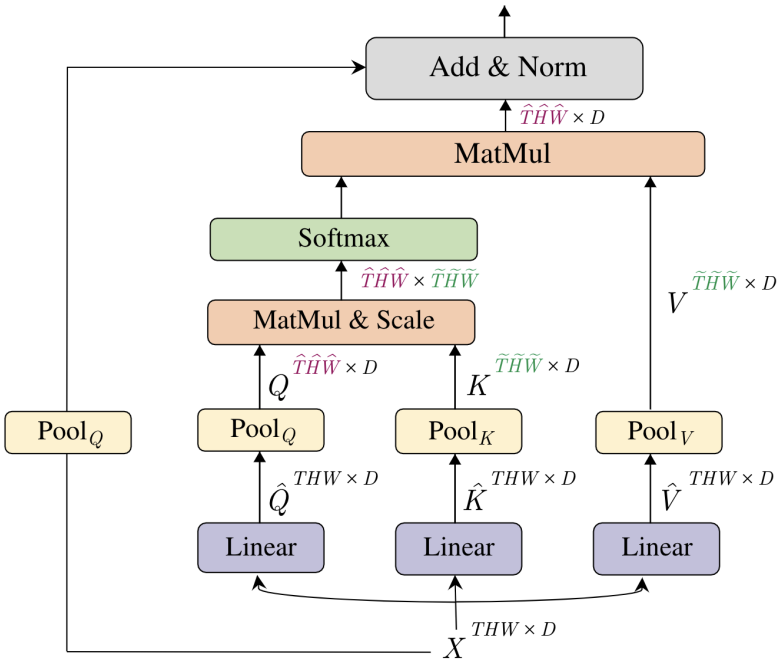



**Figura 2.1:** Rappresentazione del meccanismo di pooling attention.

un altro spazio, ottenendo così una dimensione latente D, che rappresenta la dimensione della rappresentazione interna utilizzata dal transformer. Questo processo è equivalente all'applicazione di una convoluzione avente lo stesso kernel ed uno stride di dimensione $1 \times 16 \times 16$.

Per incorporare informazioni riguardanti la posizione, ad ogni elemento della sequenza, dopo aver ottenuto il rispettivo vettore latente, viene aggiunto un positional embedding $E \in \mathbb{R}^{L \times D}$. Un valore di class embedding da apprendere viene poi aggiunto alla sequenza di patch, portando la sua dimensione al valore L+1. La sequenza ottenuta viene processata sequenzialmente da $N$ transformers blocks, ottenendo una sequenza in output che rappresenta l'embedding della classe, su cui viene applicato un layer lineare per ottenere l'indice della classe di appartenenza del dato in input.

Caratteristica di ViT è che mantiene costante la dimensione del canale e la risoluzione spaziale nei vari blocchi, mentre ciò non avviene nella versione multiscala presentata a seguire.

**Multiscale Vision Transformers (MViT)** L'idea che sta alla base del modello è quella di aumentare progressivamente la risoluzione dei canali, ovvero la profondità delle feature maps, diminuendo allo stesso tempo la risoluzione spazio-temporale. Quindi, l'architettura MViT è stata progettata

in modo da iniziare con informazioni spaziotemporali dettagliate, ottenendo features più semplici nei primi livelli della rete, che progressivamente diventano features più astratte e complesse, in grado di catturare pattern via via più significativi. Ciò consente alla rete di elaborare e apprendere in modo efficiente dai dati, concentrandosi su diversi tipi di informazioni in diverse fasi della rete.

Uno *scale stage* è costituito da un insieme di $N$ transformers blocks che operano alla stessa scala, ovvero rimangono invariate tutte le dimensione (canali, tempo, altezza e larghezza). Le patch, o cubi 3D, vengono proiettati in una dimensione del canale più piccola, rendendo però la lunghezza della sequenza più densa. Ciò fa sì che vengano elaborate più patch, ottenendo una granularità più fine nelle fasi iniziali.

Uno *stage transition* si riferisce al passaggio da una *scale stage* ad un altro all'interno della rete. Durante tale transizione, la dimensione del canale della sequenza elaborata viene aumentata contemporaneamente alla riduzione della sua lunghezza, in modo da riassumere le informazioni spaziali e temporali. I dettagli iniziali vengono aggregati ottenendo rappresentazioni più astratte che sono più significative per task di alto livello come la classificazione o il rilevamento di oggetti.

L'aumento del numero di canali si ottiene modificando la dimensione di output del livello di MLP (Multi-Layer Perceptron) rispetto alla fase precedente, di un fattore proporzionale alla variazione della risoluzione spazio-temporale introdotta. Se la risoluzione spazio-temporale viene ridotta di un fattore 4, allora la dimensione del canale viene aumentata di un fattore 2.

**Pooling** L'operazione di pooling attention offre flessibilità nel regolare la lunghezza dei vettori di *key*, *value* e *query*, e quindi della lunghezza della sequenza di output. Al vettore di *query* Q viene eseguito il pooling mediante la funzione $P(Q; k; p; s)$. Dato che bisogna ridurre la risoluzione solo all'inizio di ogni fase e poi mantenerla costante per il resto della fase, solo il primo modulo di pooling attention opera con uno *stride* maggiore di 1, ovvero esegue effettivamente il pooling riducendo la risoluzione. Gli altri moduli all'interno della stessa fase possiedono uno *stride* avente tutte le componenti a 1, il che significa che non si verifica alcun pooling. In quest'ultimi moduli viene applicato il pooling di *key* e *value* che non influisce sulla lunghezza della sequenza di output, ma è fondamentale per il controllo dei requisiti computazionali. Dato che i tensori *key* e *value* devono avere la stessa lunghezza di affinché il meccanismo di attenzione calcoli correttamente i pesi di attenzione, lo *stride* con cui viene eseguito il pooling deve essere identico. Quindi, di default i parametri di polling saranno identici all'interno della stessa fase, tuttavia in

fasi diverse lo *stride* potrà subire variazioni in base alla scala in cui si sta operando.

**Skip connections** All'interno di un residual block, possono cambiare sia il numero di canali, che la lunghezza della sequenza. Ciò può creare un' incongruenza tra l'input e l'output del blocco. Per far fronte a ciò, viene effettuato il pooling della skip connection, ovvero l'input del residual block, mediante il seguente operatore $P(\cdot; \Theta_Q)$, in modo da sommare questa versione con l'output del modulo MHPA, facendo sì che la dimensione ottenuta coincida con quella dell'output del modulo MHPA.

### 2.1.3 MViTv2

Nel seguente paper [5] viene migliorata l'architettura del Multiscale Vision Transformer, introducendo le seguenti componenti 2.2:

- **Shift-Invariant Positional Embeddings**: prevede l'utilizzo di *decomposed location distances* per creare positional embedding invarianti allo spostamento. Ciò significa che gli embedding sono progettati per essere robusti agli spostamenti o ai cambiamenti nelle posizioni dei token, migliorando la capacità del modello nella gestione di informazioni posizionali variabili.
- **Residual Pooling Connection**: Lo stride del pooling nel calcolo dell'attenzione può influire sulle prestazioni. Per far fronte a ciò, viene introdotta una *residual pooling connection*. Questa connessione compensa gli effetti dello stride del modulo di pooling nel calcolo dell'attenzione, mantenendo o ripristinando informazioni importanti perse a causa di esso.

**Decomposed relative position embedding** MViT si basa su positional embeddings assoluti per fornire informazioni sulla posizione. Ciò significa che ogni token ha un positional embedding fisso che indica la sua posizione assoluta nella sequenza. Però, questo approccio ha un limite perché non tiene conto dell'invarianza allo spostamento, ovvero il principio secondo cui il modello dovrebbe riconoscere lo stesso pattern o feature indipendentemente da dove appare nella sequenza di input. Per risolvere tale difficoltà, vengono introdotte i positional embedding relativi. La posizione relativa tra due elementi di input i e j è codificato in un positional embedding, che viene poi utilizzato nel calcolo del modulo di self-attention. Ciò significa che il meccanismo di

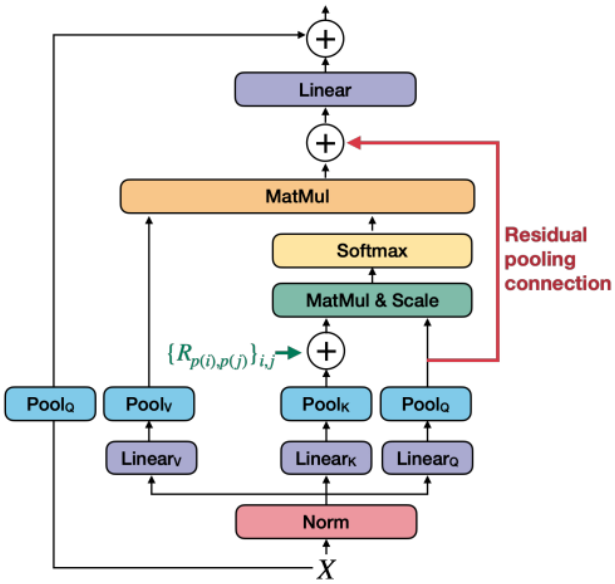



**Figura 2.2:** Rappresentazione dei miglioramenti apportati al meccanismo di Pooling Attention in MViTv2, incorporando i moduli di *decomposed relative position embedding* e *residual pooling connection* nel blocco di attenzione.

attenzione ora tiene conto della distanza relativa tra i token, migliorando la sua capacità di modellare le interazioni in modo shift-invariant.

Tuttavia, il calcolo dei positional embedding relativi è costoso dal punto di vista computazionale perché coinvolge un gran numero di possibili embedding. Perciò, per rendere il processo più efficiente, il calcolo viene scomposto in embedding separati per altezza, larghezza e dimensioni temporali. Questa scomposizione riduce il numero totale di embedding necessari, riducendo così la complessità da $O(T \cdot W \cdot H)$ a $O(T + W + H)$, che è molto più gestibile, soprattutto per input ad alta risoluzione.

**Residual pooling connection** Comporta l'aggiunta del tensore Q *pooled* alla sequenza di output $Z$. Questo migliora il propagarsi delle informazioni e quindi anche l'allenamento dei moduli di attenzione. Il meccanismo originale di attenzione viene quindi modificato incorporando il tensore Q *pooled* nell'output. In particolare, la sequenza di output che si ottiene è la seguente:

$$Z := \text{Attn}(Q, K, V) + Q \tag{2.3}$$

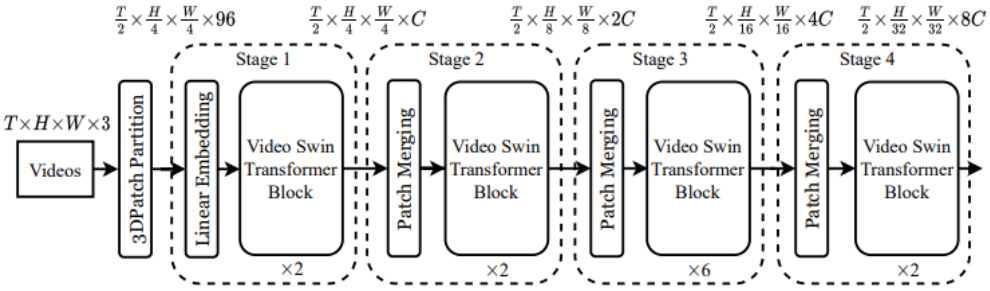



**Figura 2.3:** Struttura generale della versione tiny (Swin-T) di un Video Swin Transformer.

## 2.2 Video Swin Transformer

Questo modello, introdotto nel seguente paper [6], è interamente basato sull'architettura Transformer, a differenza di altri approcci che combinano i Transformers con altri modelli, come le CNN. L'architettura che viene proposta è più efficiente rispetto ai modelli precedenti che separano l'elaborazione spaziale e temporale. Ciò è dovuto al fatto che viene sfruttata l'intrinseca località spaziotemporale dei video, nella quale i pixel che sono più vicini tra loro, sia nella dimensione spaziale che temporale, hanno più probabilità di essere correlati. Grazie a questa proprietà, l'intera self-attention spaziotemporale può essere ben approssimata dalla self-attention calcolata localmente, con un notevole risparmio in termini di complessità computazione e dimensioni del modello.

L'implementazione di questo approccio è basata sull'estensione spaziotemporale del Swin Transformer [7], introdotto per task basati sulla comprensione di immagini, in modo da far sì che il calcolo dell'attenzione locale, applicata solo al dominio spaziale, venga eseguita sul dominio spaziotemporale.

### 2.2.1 Architettura

Come mostrato in figura 2.3, il modello prende in input un video avente una dimensione pari a $T \times H \times W \times 3$ dove: T rappresenta il numero di frame, H e W indicano rispettivamente l'altezza e larghezza di ogni frame, mentre l'ultimo valore indica il numero di canali utilizzati per il colore. A partire dal video in input, per ottenere un set di tokens da passare al modello, ogni token è estratto a partire da una patch 3D avente una dimensione pari a $2 \times 4 \times 4 \times 3$; quindi, si ottengono 96 valori che rappresentano la dimensione

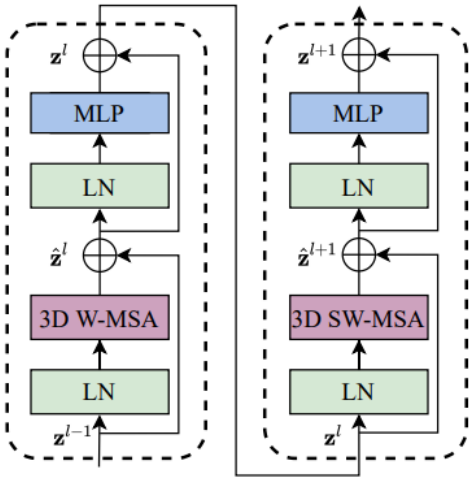



**Figura 2.4:** Illustrazione di due Video Swin Transformer blocks in sequenza.

delle features estratte per ogni patch. Viene suddiviso in questo modo l'intero video, ottenendo un numero di patch, e quindi tokens, pari a $\frac{T}{2} \times \frac{H}{4} \times \frac{W}{4}$. Su tali vettori di features è applicato un layer lineare in modo da mapparli in un embedding space, ovvero uno spazio differente avente dimensione pari a $C$ –parametro da settare opportunamente.

A differenza di altri modelli che effettuano il downsampling della risoluzione temporale per far fronte all'elevata complessità computazionale richiesta, questo modello mantiene l'intera risoluzione temporale durante l'elaborazione; quindi, il numero di fotogrammi T rimane invariato. Ciò fa sì che il modello possa tenere la medesima struttura gerarchica dell'originale Swin Transformer, che consiste in quattro stadi, ciascuno dei quali elabora l'input a diverse scale ed esegue il downsampling spaziale, riducendone le dimensioni di un fattore 2. Quest'ultima operazione viene eseguita dal livello denominato *patch merging* che raggruppa ogni insieme di 2×2 patch limitrofe andando a concatenare le features, ottenendo un vettore di dimensione pari a $4C$. In seguito, viene applicato un livello lineare per ridurre la dimensionalità delle feature concatenate, che ha l'effetto di dimezzarne le dimensioni. Questa riduzione è necessaria per mantenere il modello efficiente e gestibile, riducendo il carico computazionale, preservando al tempo stesso le informazioni importanti.

Il componente principale è rappresentato dal Video Swin Transformer block 2.4 che, invece di utilizzare il modulo di multi-head self-attention (MSA) pre-

| Versione | C | Numero di livelli |
|---|---|---|
| Swin-T | 96 | {2, 2, 6, 2} |
| Swin-S | 96 | {2, 2, 18, 2} |
| Swin-B | 128 | {2, 2, 18, 2} |
| Swin-L | 192 | {2, 2, 18, 2} |

**Tabella 2.1:** Iperparametri delle 4 versioni di Video Swin Transformer.

sente nei tradizionali Transformer blocks, lo sostituisce con un MSA basato su 3D shifted window, mantenendo le altre componenti invariate. Tale variazione è stata introdotta per adattare il meccanismo di self-attention per lavorare su patch 3D, quindi considerando sia le dimensioni spaziali che temporali, con un approccio a finestra (shifted window) per catturare efficacemente sia le dipendenze locali che globali. Dopo il modulo MSA, il blocco include una rete feed-forward (FFN), ovvero un Multilayer Perceptron (MLP) con 2 livelli, che introduce non linearità attraverso la funzione di attivazione GELU (Gaussian Error Linear Unit) applicata tra i due livelli. Viene applicato, inoltre, un layer di normalizzazione, sia prima del modulo MSA, che di FFN, in modo da stabilizzare e velocizzare il processo di addestramento. Dopo ogni modulo viene aggiunta una residual connection, ciò implica che l'input di ciascun modulo viene aggiunto al suo output in modo da mitigare il problema dell'esplosione del gradiente e consentire ai gradienti di fluire attraverso la rete in modo più efficace.

**Varianti** Vengono proposte, così come nell'originale Swin Transformer, 4 versioni anche per il Video Swin Transformer. In tabella 2.1, vengono mostrati i valori degli iperparametri di ogni versione.

### 2.2.2 3D Shifted Window based MSA Module

Dato che rispetto alle immagini i video richiedono un numero nettamente maggiore di token per poter essere rappresentati, un modulo di self-attention globale risulterebbe impraticabile in quanto comporterebbe enormi costi di calcolo e memoria. Perciò, viene introdotto un bias locale al modulo di self-attention in modo da limitare le interazioni tra token all'interno di una regione più piccola e localizzata; così come avveniva nel Swin Transformer originale. Implementare ciò, si è dimostrato essere efficace per task di riconoscimento video.

**Multi-head self-attention su finestre 3D non sovrapposte** Una finestra 3D fa riferimento ad un blocco di token che si estende sia sulle dimensioni spaziali, che sulla dimensione temporale. Un video è quindi rappresentato come un array 3D di tokens avente dimensione $T' \times H' \times W'$. Definita una finestra avente dimensione $P \times M \times M$, dove P rappresenta la profondità temporale della finestra, mentre M è la dimensione della finestra in altezza e larghezza, il video in input viene suddiviso in finestre 3D non sovrapposte che ricoprono l'intero video in modo uniforme. Fatto ciò, il modulo di multi-head self-attention viene eseguito all'interno di ogni finestra 3D in modo indipendente, riducendo così la complessità computazionale.

**3D Shifted Windows** Poiché non sono presenti interazioni tra token in finestre diverse, il modello potrebbe perdere relazioni e dipendenze importanti che si estendono al di fuori della finestra stessa. Ciò può limitare la potenza di rappresentazione del modello, dato che potrebbe non catturare correttamente il contesto globale del video. Per far fronte a ciò, viene introdotto un meccanismo in cui, oltre ad elaborare finestre non sovrapposte, le finestre vengono spostate a intervalli regolari, in modo che ogni token possa interagire anche con token al di fuori della sua finestra. In questo modo, vengono attenzionate anche le relazioni più distanti, mantenendo al tempo stesso l'efficienza computazionale garantita dal modulo di self-attention con finestre non sovrapposte.

**3D Relative Position Bias** L'aggiunta della posizione relativa dei token nel calcolo del modulo di self-attention, si è rilevata essere vantaggiosa in termini di prestazioni per diversi modelli, in quanto fa sì che il modello sia in grado di comprendere meglio le relazioni spaziali e temporali tra i token. Perciò, viene introdotta anche nel modello in esame, andando ad aggiungere un termine di *bias* nel calcolo di ogni *head* di self-attention nel seguente modo:

$$\text{Attention}(Q, K, V) = \text{SoftMax}\left(\frac{QK^T}{\sqrt{d}} + B\right)V \tag{2.4}$$

dove $Q, K, V \in \mathbb{R}^{PM^2 \times d}$ sono le matrici di *query*, *key* e *value*; $d$ rappresenta la dimensione delle feature di *key*, e $PM^2$ indica il numero di tokens all'intero di una finestra 3D.

Poiché la posizione relativa lungo ciascun asse si trova nell'intervallo $[-P+1, P-1]$ (temporale) o $[-M+1, M-1]$ (altezza o larghezza), viene parametrizzata una matrice di *bias* di dimensioni più ridotte, ovvero $\hat{B} \in \mathbb{R}^{(2P-1)\times(2M-1)\times(2M-1)}$, in modo da prendere i valori di $B$ da $\hat{B}$.

# Capitolo 3

# Fine-tuning dei modelli

Agli algoritmi illustrati nel capitolo precedente verrà effettuato un fine-tuning facendo uso di Pytorch, libreria open-source per il linguaggio Python, che mette a disposizioni l'implementazione dei modelli in esame con i rispettivi checkpoint relativi al pre-training effettuato sul dataset Kinetics400. Esso contiene 400 classi di possibili azioni umane, con almeno 400 clip video per ogni azione. Ogni clip dura circa 10 secondi ed è tratta da un diverso video di YouTube. Le azioni sono incentrate sull'uomo e coprono un'ampia gamma di classi, tra cui sia le interazioni uomo-oggetto che quelle uomo-uomo.

Tra i modelli resi disponibili da Pytorch sono stati utilizzati in questa fase, così come nella fase di test, la versione 2 di MViT e la versione tiny del Video Swin Transformer.

**Struttura del codice** Il codice è strutturato in più file python, che gestiscono diverse funzionalità:

- **constants.py**: vengono definite al suo interno le costanti utilizzate nel processo di fine-tuning;
- **charades.py**: definisce la classe `CharadesDataset` che consente di gestire il dataset efficientemente;
- **utils.py**: contiene funzioni e classi di supporto per il campionamento delle clip;
- **transforms.py**: contiene due classi per l'applicazione di determinate trasformazioni ai video;
- **parse.py**: contiene al suo interno una funzione, chiamata `parse_cmd` che si occupa di definire e gestire gli argomenti che vengono passati via riga di comando nell'esecuzione dei file `train.py` e `test.py`;

- **train.py**: contiene il flusso che esegue il processo di fine-tuning, all'interno del quale vengono definite le componenti principali quali funzione di loss, optimizer e scheduler;

- **test.py**: esegue la fase di test dei modelli, andando a calcolare le metriche che permetto di stabilire la bontà di ciascun modello.

Per l'esecuzione del file `train.py` bisogna passare via riga di comando 2 argomenti: `--model`, per selezionare il modello da allenare (mvitv2 o swin_t), e `--collation`, che rappresenta la modalità in cui vengono estratti i frame. Quest'ultimo parametro supporta due valori possibili, ovvero *trimming* o *padding*; di default viene impostato il valore *trimming*. Quest'ultima modalità prende il video che ha meno frame e fa sì che i restanti video dello stesso batch abbiano lo stesso numero di frame, in modo che possano stare nello stesso tensore.

La funzione `parse_cmd` che si occupa di effettuare il parsing dei parametri passati via riga di comando, inizializza il modello specificato dopo l'argomento `--model` con i pesi pre-allenati e modifica l'ultimo layer in modo che restituisca un numero di elementi pari al valore della costante `NUM_CLASSES`. L'argomento `--model` accetta un ulteriore valore, ovvero "local", che permette di caricare un modello presente in locale, specificando il path da cui caricarlo con l'argomento `--path` oppure `-p`. Quest'ultima modalità permette di effettuare il test dei modelli a partire dai checkpoint salvati dopo aver effettuato il training, ma anche di riprendere il training da un determinato checkpoint.

A seguire verrà mostrato il pre-processing effettuato ed analizzata l'implementazione della fase di addestramento dei modelli, spiegando le varie componenti adottate e motivando i parametri utilizzati.

## 3.1 Caricamento e pre-processing dei dati

Al fine di caricare il dataset Charades ed effettuare le operazioni necessarie per poter addestrare i modelli in esame, sono state definite due classi all'interno del file `charades.py`: `CharadesSample` e `CharadesDataset`. La prima classe consente di definire una semplice struttura per rappresentare un singolo campione del dateset. In particolare, contiene i seguenti campi: video, un tensore che rappresenta i frame del video; framerate, la frequenza dei fotogrammi; objects, un elenco di oggetti presenti; actions: le etichette che rappresentano le azioni compiute nel video; timings, un elenco di intervalli che mappano le azioni su fotogrammi specifici del video.

Mentre, `CharadesDataset`, classe personalizzata per contenere il dataset, eredita da IterableDataset, utile quando si lavora con set troppo grandi per essere caricati interamente in memoria. Il metodo `__init__` definito al suo interno per inizializzare il dataset imposta inizialmente un URL per scaricare il dataset Charades, mescola il dataset usando il parametro `seed` impostato a 42 in modo da garantire replicabilità, e per calcolare il numero di video da acquisire viene impostato un numero specificato di batch da selezionare: 225 per il training e 32 per il test, per un totale di video pari rispettivamente a 4050 e 576, dato che la dimensione del batch è settata a 18. Imposta infine una funzione di trasformazione da applicare ai fotogrammi video, definita nel file `constants.py`, che fa sì che il video venga ridimensionato per ottenere una dimensione di 224x224, e che venga trasformato in tensore, usando le classi definite nel file `transforms.py`.

Il metodo `_convert_timings_to_frames` converte i tempi delle azioni (in secondi) in numeri di fotogrammi, in base alla frequenza dei frame del video. Tale operazione consente quindi di allineare le azioni con fotogrammi specifici nel video.

Il metodo principale, `extract_sample`, estrae ed elabora un singolo campione dal dataset, leggendo il file video effettivo dall'archivio zip, se non è già stato estratto. Il file video viene letto utilizzando `OpenCV`, e convertito in formato RGB per poi essere memorizzato nei vari frame. Se impostata, viene applicata la funzione di trasformazione ai vari frame. Vengono convertiti i tempi di azione in numeri di fotogramma utilizzando il metodo precedente. Infine, viene impacchettato tutto ciò che si è ottenuto nell'oggetto `CharadesSample`, che viene restituito dalla funzione.

Un'altra componente chiave è la classe `ClipSampler`, definita nel file `utils.py`, che consente di estrarre un certo numero di clip da un determinato video che prende come input. Ogni clip è composta da un numero di frame da specificare nel parametro `clip_size`, che viene impostato col valore della costante `SAMPLE_CLIP_SIZE`, settata col valore 16.

Essa supporta due strategie di campionamento delle clip:

- **Dilated mode**: estrae una sola clip, il cui indice del frame di partenza viene calcolato in modo random, impostando comunque il seed col valore 42. Inoltre, nell'estrazione dei vari frame che compongono la clip viene usato un fattore di dilatazione che distanzia i vari frame, in modo da coprire un tempo di secondi predefinito, che è stato impostato a 3 secondi. Quindi, in base al frame rate di ogni video, viene calcolato il rispettivo fattore di dilatazione che fa sì che i frame estratti ricoprano un intervallo di tempo pari a 3 secondi.

- **Stride mode**: campiona più di una clip per video, partendo dall'inizio fino al raggiungimento di un numero di clip da specificare nel parametro `max_samples`. Ogni clip viene estratta muovendosi di un numero di passi stabilito dal parametro `stride`, calcolato in base al framerate di ciascun video.

La prima modalità viene eseguita durante la fase di training per alleggerire il processo estraendo una clip ridotta dal video di partenza, che contenga comunque abbastanza informazioni. Mentre, in fase di test, avendo meno campioni da analizzare, viene adottata la seconda modalità che prende un massimo di 4 clip per video.

**Raggruppamento delle classi** Per semplificare il task, le 157 classi presenti nel dataset Charades sono state raggruppate in 21 classi, ognuna delle quali comprende azioni simili oppure che avvengono interagendo col medesimo oggetto. In tabella 3.1 sono elencate le classi prodotte insieme alla loro descrizione.

## 3.2 Processo di training

La fase di training, eseguita nel file `train.py`, definisce il dataset mediante l'oggetto `CharadesDataset` che viene passato al `DataLoader`, insieme al parametro `batch_size` impostato a 18 e `collate_fn`, che esegue la modalità `trimming`. Vengono definite in seguito le componenti chiave, quali: funzione di loss, impostata con la funzione `nn.BCEWithLogitsLoss()`; ottimizzatore, è stato selezionato Adam (Adaptive Moment Estimation), a cui vengono passati i parametri del modello ed il learning rate settato a 0,001; scheduler, che si occupa di far decrescere esponenzialmente il learning rate di un fattore gamma, impostato a 0.5, dopo ogni epoca.

Per quanto riguarda la funzione di loss, è stato impostato il parametro `pos_weight` che permette di gestire dataset sbilanciati pesando ogni classe in base a quanto è presente. In questo modo alle classi meno presenti verrà assegnato un valore maggiore, facendo sì che il modello debba prestarli maggior attenzione. Dato che il dataset viene letto in streaming e non si ha quindi accesso all'intero set di training, per calcolare la presenza di ogni classe si è fatto uso di un file csv che contiene per ogni video una lista di azioni presenti al suo interno. A partire da ciò si è calcolati per ogni classe quante volte fosse presente, ottenendo così il conteggio delle volte positive. Per ottenere i casi negativi si è effettuata una sottrazione tra il numero totale di video e i conteggi positivi. Effettuando la divisione tra casi negativi e positivi si è ottenuto il valore da impostare al parametro `pos_weight`.

| Nome | Descrizione | Indice |
|---|---|---|
| Clothing and Dressing | Azioni relative all'indossare, rimuovere o riordinare vestiti. | 0 |
| Door/Window Interaction | Azioni che comportano l'apertura, chiusura, riparazione di porte o finestre, oppure il passaggio attraverso di esse. | 1 |
| Table Interaction | Azioni relative allo stare seduti, lavorare o riordinare un tavolo. | 2 |
| Phone/Camera Interaction | Azioni che comportano l'uso di un telefono o di una fotocamera, così come il gesto di posarlo o prenderlo. | 3 |
| Bag Handling | Azioni relative all'uso, apertura o chiusura di una borsa, così come posarla. | 4 |
| Book/Reading | Azioni che implicano la lettura, la presa in mano o l'interazione con un libro. | 5 |
| Towel Handling | Azioni relative all'uso o al riordino degli asciugamani. | 6 |
| Box Interaction | Azioni che comportano l'apertura, la chiusura, l'uso o lo spostamento di scatole. | 7 |
| Laptop/Computer Interaction | Azioni che comportano l'utilizzo di un laptop, includendo il gesto di posarlo. | 8 |
| Shoe Interaction | Azioni relative all'indossare, togliere o avere in mano le scarpe. | 9 |
| Food and Eating | Azioni relative alla preparazione o al consumo di cibo. | 10 |
| Blanket/Pillow Interaction | Azioni che consistono nel tenere o riordinare coperte o cuscini. | 11 |
| Shelf Interaction | Azioni relative al riordino o all'organizzazione degli articoli su uno scaffale. | 12 |
| Picture/Photo Interaction | Azioni relative all'interazione con immagini o foto. | 13 |
| Mirror Interaction | Azioni che coinvolgono gli specchi, come guardarsi o pulire lo specchio. | 14 |
| Cleaning and Tidying | Azioni relative alla pulizia, al riordino o all'utilizzo di strumenti di pulizia. | 15 |
| Light Interaction | Azioni relative all'accensione, spegnimento o alla riparazioni di luci. | 16 |
| Drinking/Pouring | Azioni relative al bere o versare in una tazza, un bicchiere o una bottiglia. | 17 |
| Sofa/Bed Interaction | Azioni che coinvolgono sdraiarsi, sedersi o svegliarsi su un divano, un letto o un pavimento. | 18 |
| Emotional Expressions | Azioni che implicano la manifestazione di emozioni come ridere o piangere, ma anche starnutire. | 19 |
| Running | Contiene la sola azione di correre. | 20 |

**Tabella 3.1:** Informazioni relative al raggruppamento in 21 classi a partire delle 157 presenti nel dataset Charades.

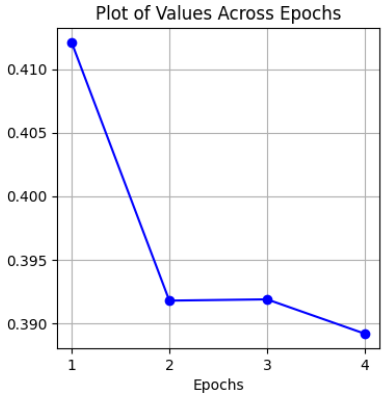
Il flusso di training viene eseguito un numero di volte pari a 4, valore impostato nella costante `NUM_EPOCHS`. All'interno del ciclo, viene inizialmente impostato il modello in modalità di training, eseguendo il comando `model.train()`, ed inizializzate le variabili `epoch_loss` e `num_batches`. Per ogni video del batch, ClipSampler campiona una singola clip, avente un durata di 3 secondi, composta da 16 frame. Le azioni associate a ogni clip vengono convertite in vettori *one-hot* e sommati per formare l'etichetta reale dell'intera clip. Una volta aggregate le informazioni dei video del batch vengono passate al modello le varie clip per ottenere le predizioni su cui viene calcolata la funzione di loss. Quest'ultimo valore viene aggiunto alla variabile `epoch_loss` e viene incrementata la variabile `num_batches`.

I gradienti vengono calcolati usando il comando `loss.backward()`, tuttavia l'ottimizzatore aggiorna i parametri del modello dopo un certo numero di iterazioni, definito dalla costante `GRADIENT_ACCUMULATION_ITERS`, impostata a 4. Ciò consente un training efficace anche con batch di dimensioni ridotte. Inoltre, dopo ogni aggiornamento vengono azzerati i gradienti. Infine, il learning rate viene regolato dallo scheduler, col comando `scheduler.step()`

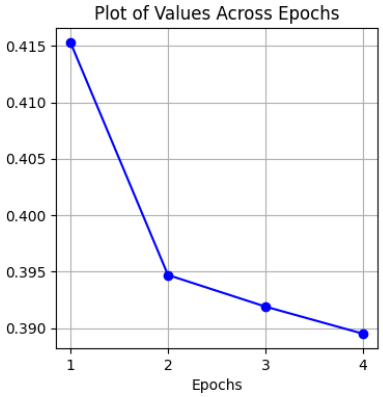
Dopo un certo numero di batch eseguiti, da specificare nella costante `PRINT_BATCH_LOSS_EVERY`, impostata col valore 4, viene visualizzata la loss del batch corrente per tenere traccia dell'evoluzione del modello. Mentre, la loss per ogni epoca viene calcolata dividendo la variabile `epoch_loss` per `num_batches` e visualizzata alla fine di ogni epoca. Prima di passare all'epoca successiva, viene salvato il modello corrente per far sì che possa essere ripreso l'addestramento da una qualsiasi epoca precedente, ma soprattutto per poter fare inferenza una volta ultimato il processo di training.

## 3.3 Risultati ottenuti

Durante il processo di training sono stati mantenuti i valori della funzione di loss per ogni epoca, calcolati come media dei valori ottenuti per ogni batch. I risultati per ciascun modello sono riportati in figura 3.1. L'andamento della curva, per entrambi i modelli, evidenzia un maggior decremento tra la prima e la seconda epoca, mentre, a partire da quest'ultima, la funzione di loss si abbassa di poco, dovuto al fatto che il learning rate viene dimezzato, ma continua comunque a diminuire seppur lentamente. Eppure, rispetto alla prima epoca, il valore della funzione di loss nell'ultima epoca non si discosta più di tanto. Per poter osservare un significativo decremento della funzione di loss sarebbe necessario eseguire un numero decisamente più elevato di epoche, così come aumentare il training set, tuttavia, date le poche risorse




**(a)** Risultati relativi al modello MViTv2




**(b)** Risultati relativi al modello Video Swin Transformer

**Figura 3.1:** Valori di loss ottenuti durante il training dei modelli in esame, per un totale di 4 epoche eseguite.

computazionali, non è stato possibile effettuare ciò. Quindi, i modelli in esame verranno confrontanti con l'ultima epoca di training effettuata, ovvero la quarta, seppur il processo di training sarebbe dovuto continuare o essere modificato per ottenere prestazioni migliori.

# Capitolo 4

# Benchmarking

Dopo aver effettuato il fine-tuning dei modelli verranno sottoposti ad una fase di test in cui verranno calcolate diverse metriche al fine di verificare quale sia l'architettura che ottiene le performance migliori. Il test set è composto da un numero di video pari a 576, per ognuno dei quali viene estratto un numero di clip pari a 4, senza alcun fattore di dilatazione, quindi vengono estratti 16 frame consecutivi, usando la modalità "stride" della classe `ClipSampler`.

Durante questa fase è stata calcolata anche la funzione di loss ottenuta per l'intero set di test, come media dei valori ottenuti per i singoli batch, e sono stati ottenuti i valori 0.41 e 0.42 rispettivamente per MViTv2 e Video Swin Transformer. Tali risultati si avvicinano a ciò che è stato riscontrato in fase di training. Tuttavia, per capire se effettivamente i modelli riescono a generare predizioni corrette sono state adottate diverse metriche che verranno introdotte a seguire.

## 4.1 Metriche

Dato che ogni clip può avere più di un'etichetta di classe, per valutare le predizioni restituite da ciascun modello si è fatto uso della libreria `torcheval` che fornisce l'implementazione di una metrica in grado di calcolare l'accuracy su task multilabel. La metrica in questione, chiamata `TOPKMULTILABELACCURACY`, prende i top `K` elementi aventi probabilità maggiore e imposta la predizione in modo che abbia questi valori a 1 mentre i restanti tutti a 0. Permette di calcolare la frequenza delle `K` label predette che corrispondono esattamente alle etichette target; tuttavia, supporta anche altre modalità di calcolo oltre alla corrispondenza esatta. Di seguito una lista delle modalità disponibili, che verranno calcolate con valori di `K` pari a 2, 3 e 5:

- **exact_match**: è il criterio più rigido nel quale le k etichette predette devono corrispondere esattamente alle etichette reali. Quindi, sia il contenuto che il numero di etichette deve coincidere con le etichette reali per far sì che la predizione venga considerata corretta;

- **hamming**: calcola la frazione di etichette predette che sono corrette, rispetto alla dimensione dell'etichetta reale;

- **overlap**: richiede che l'etichetta stimata abbia valore 1 nella stessa posizione in cui è presente nell'etichetta reale. Se c'è almeno una corrispondenza, la predizione viene considerata corretta;

- **contain**: per considerare corretta l'etichetta predetta richiede che essa posizioni gli elementi col valore 1 nella stessa posizione in cui sono presenti nell'etichetta reale. Le etichette stimate possono avere valori ad 1 in più, ma gli elementi nell'etichetta reale col valore 1 devono essere tutti presenti nelle stesse posizioni nelle predizioni del modello.

- **belong**: richiede che le etichette predette ad 1 appartengano completamente all'insieme delle etichette reali. Tutte le etichette ad 1 predette dal modello devono essere corrette, ma potrebbero comunque esserci più etichette reali ad 1 che non sono state previste nella stima.

Un'altra metrica che viene calcolata è la `MultilabelAccuracy` che prevede l'impostazione di una soglia per considerare ogni classe positiva o negativa nel seguente modo: `torch.where(input < threshold, 0, 1)`, dove `input` rappresenta le predizioni del modello e la threshold viene impostata di default col valore 0.5. Essa supporta le stesse modalità presentate per la metrica precedente che verranno calcoolate con la threshold di default.

## 4.2 Risultati ottenuti

In tabella 4.1 e 4.2 sono mostrati i risultati delle metriche calcolate per ciascun modello analizzato. I valori ottenuti per i due modelli sono abbastanza simili. Riguardo alle metriche calcolate basandosi sulla threshold si hanno prestazioni veramente scadenti: non si ottiene nessun match esatto; le azioni predette non sono interamente presenti nell'etichetta reale in nessun caso; le predizioni non contengono mai tutte le azioni presenti nell'etichetta reale; in una percentuale veramente bassa un'azione predetta è presente tra le azioni presenti nell'etichetta reale. Solo la modalità "hamming" ha ottenuto un valore alto, tuttavia, ciò è dovuto al fatto che calcola quante predizioni

sono corrette rispetto alla cardinalità delle etichette, quindi, considera anche la corrispondenza tra gli 0 presenti, che effettivamente vengono calcolati correttamente, ma ciò accade poiché sono maggiormente presenti nell'etichetta reale, visto che solo pochi valori hanno valore 1, ovvero ci sono poche azioni presenti all'interno di ciascuna clip.

Riguardo alla metrica `TOPKMULTILABELACCURACY`, i risultanti non sono migliori: il match esatto non viene ottenuto mai, per nessun valore di `K`; la modalità di "hamming" parte da un valore intorno allo 0.85 il che significa che, in media, il 25% delle predizioni sono sbagliate, indicando che vi sono circa 3 valori errati nell'etichetta predetta, all'aumentare di `K` il valore decresce, quindi viene introdotto un errore più grande tra etichetta reale e predette prendendo in considerazione più azioni; per la modalità di "overlap" si raggiunge il valore più alto con `K` pari a 5, tuttavia, i valori ottenuti si aggirano intorno allo 0.22, il che significa che solo nel 20% dei casi è presente almeno un'etichetta corretta tra le top-`K` predizioni con probabilità più alta; la modalità "contain" raggiunge il valore 0.16 per `K` pari a 5, indicando che solo in pochi casi tutte le azioni presenti in una clip vengono individuate; infine, in nessun caso vengono le azioni che vengono predette sono tutte corrette, ovvero presenti nell'etichetta reale.

In definitiva, non si è riscontrata alcuna evidente differenza tra le prestazioni dei due modelli dato che hanno ottenuto andamenti e metriche piuttosto simili durante l'analisi effettuata, sia in fase di training che di test. Ciò può essere dovuto all'aver eseguito un numero esiguo di epoche durante il training dei modelli, che non ha permesso di apprendere correttamente dai dati analizzati, ottenendo così prestazioni scadenti durante la fase di test. Da quest'ultima fase è emerso che in termini di recall il modello va leggermente meglio rispetto alla precision rappresentata dalla modalità "belong" che possiede tutti i valori a 0, sia utilizzando la threshold che con le top-`K` predizioni. Tuttavia, i modelli avrebbero ancora molto da apprendere e molto margine di miglioramento eseguendo un training con più dati e più epoche. Inoltre, anche in fase di training sarebbe necessario andare ad estrarre più di una clip in modo da coprire tutte le azioni presenti all'interno del rispettivo video, consentendo così al modello di analizzare più sample per ogni classe.

| Criteria | Threshold (0.5) | Top-2 | Top-3 | Top-5 |
|---|---|---|---|---|
| exact_match | 0.0 | 0.0 | 0.0 | 0.0 |
| hamming | 0.94 | 0.86 | 0.81 | 0.73 |
| overlap | 0.05 | 0.09 | 0.10 | 0.21 |
| contain | 0.0 | 0.07 | 0.08 | 0.16 |
| belong | 0.0 | 0.0 | 0.0 | 0.0 |

**Tabella 4.1:** Risultati relativi alle metriche ottenute testando il modello MViTv2.

| Criteria | Threshold (0.5) | Top-2 | Top-3 | Top-5 |
|---|---|---|---|---|
| exact_match | 0.0 | 0.0 | 0.0 | 0.0 |
| hamming | 0.94 | 0.85 | 0.81 | 0.73 |
| overlap | 0.04 | 0.08 | 0.10 | 0.22 |
| contain | 0.0 | 0.06 | 0.08 | 0.16 |
| belong | 0.0 | 0.0 | 0.0 | 0.0 |

**Tabella 4.2:** Risultati relativi alle metriche ottenute testando il modello Video Swin Transformer.

# Conclusione

Nel seguente lavoro, a seguito dello studio dell'architettura alla base dei due modelli in esame e alle migliorie che ciascuno di essi apporta, è stato effettuato un fine-tuning, partendo dai modelli pre-addestati resi disponibili da Pytorch per il task di Video Classification, per adattarli ad un nuovo dataset, ovvero Charades. Da quest'ultimo, data la numerosità di classi presenti al uso interno, è stato eseguito un raggruppamento delle classi con lo scopo di ridurre la complessità del task in esame, ottenendo così 21 classi dalle 157 iniziali.

Durante la fase di training, a ciascun modello viene passato come input un batch di clip, ciascuna delle quali composta da un numero di frame pari a 16 che ricopre un arco di tempo pari a 3 secondi. La funzione di loss calcolata per ciascuna epoca ha evidenziato nel complesso un andamento decrescente, per entrambi i modelli, ma non sufficiente all'apprendimento necessario per compiere correttamente il task in esame. Infatti, il valore ottenuto alla fine dell'ultima epoca eseguita non si discosta di molto rispetto alla prima epoca, segno del fatto che i modelli hanno bisogno di più esempi per ogni classe, così come più epoche per un maggior apprendimento. Ciò avviene per entrambi i modelli, senza alcuna differenza particolarmente rilevante.

Quanto detto viene confermato dalla fase di test, effettuata campionando da ciascun video 4 clip, che consente di ottenere diverse metriche che mettono in risalto l'incapacità dei modelli nell'individuare le azioni presenti nelle varie clip. In particolare, in pochi casi riescono a ottenere predizioni che abbiamo almeno un'azione corretta, quindi la maggior parte delle volte le azioni con più alta probabilità non sono presenti all'interno dell'etichetta reale.

Non si è riusciti a migliorare i risultati ottenuti in quanto il task in esame richiede grosse risorse computazionali che non sono a nostra disposizione. Se così non fosse, sarebbe stato opportuno aumentare il set di training in modo da comprendere l'intero set di training composto da 7986 campioni, dei quali appena la metà sono stati utilizzati nel training effettuato. Inoltre, si potrebbe andare a campionare più di una clip per video, così come è stato effettuato nella fase di test, in modo da passare al modello tutte le azioni

presenti in ciascun video. Infine, sarebbe stato necessario eseguire un numero maggiore di epoche di training ed aggiustare opportunamente il learning rate in base all'andamento della funzione di loss.

# Bibliografia


[1] Alexey Dosovitskiy, Lucas Beyer, Alexander Kolesnikov, Dirk Weissenborn, Xiaohua Zhai, Thomas Unterthiner, Mostafa Dehghani, Matthias Minderer, Georg Heigold, Sylvain Gelly, Jakob Uszkoreit, and Neil Houlsby. An image is worth 16x16 words: Transformers for image recognition at scale. *CoRR*, abs/2010.11929, 2020.

[2] Gunnar A. Sigurdsson, Gül Varol, Xiaolong Wang, Ali Farhadi, Ivan Laptev, and Abhinav Gupta. Hollywood in homes: Crowdsourcing data collection for activity understanding. *CoRR*, abs/1604.01753, 2016.

[3] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser, and Illia Polosukhin. Attention is all you need. *CoRR*, abs/1706.03762, 2017.

[4] Haoqi Fan, Bo Xiong, Karttikeya Mangalam, Yanghao Li, Zhicheng Yan, Jitendra Malik, and Christoph Feichtenhofer. Multiscale vision transformers. *CoRR*, abs/2104.11227, 2021.

[5] Yanghao Li, Chao-Yuan Wu, Haoqi Fan, Karttikeya Mangalam, Bo Xiong, Jitendra Malik, and Christoph Feichtenhofer. Improved multiscale vision transformers for classification and detection. *CoRR*, abs/2112.01526, 2021.

[6] Ze Liu, Jia Ning, Yue Cao, Yixuan Wei, Zheng Zhang, Stephen Lin, and Han Hu. Video swin transformer. *CoRR*, abs/2106.13230, 2021.

[7] Ze Liu, Yutong Lin, Yue Cao, Han Hu, Yixuan Wei, Zheng Zhang, Stephen Lin, and Baining Guo. Swin transformer: Hierarchical vision transformer using shifted windows. *CoRR*, abs/2103.14030, 2021.