Sommario

PANORAMICA DEL PROGETTO

Nel campo delle neuroscienze computazionali, il *brain encoding* rappresenta una frontiera fondamentale per comprendere come il cervello elabori stimoli sensoriali complessi. Questo progetto si inserisce in tale contesto con l'obiettivo di adattare e valutare un modello di deep learning, specificamente **ViV1T (Video-Pretrained V1 Transformer)**, per predire l'attività neuronale di un topo in risposta a stimoli video naturalistici.

Il lavoro ha richiesto un significativo processo di data engineering, partendo dalla selezione di una sessione sperimentale di imaging a 2 fotoni dall'Allen Brain Observatory. È stata scelta una sessione (Session A, ID 501729039) che includesse video naturalistici (natural_movie_one e natural_movie_three) e dati fondamentali di eye-tracking, necessari per il funzionamento del modello ViV1T.

È stato quindi costruito un dataset customizzato, segmentando i video e allineandoli temporalmente con i dati comportamentali (velocità di corsa, posizione e dimensione della pupilla) e con le tracce di fluorescenza (le risposte neuronali). Il modello ViV1T pre-addestrato è stato adattato utilizzando una strategia di **transfer learning**, caricando esclusivamente i pesi del "core" (i blocchi Transformer) e addestrando da zero la "testa" specifica per il topo (moduli Shifter e Readout).

Infine, per migliorare le performance, è stata implementata una tecnica di **Parameter-Efficient Fine-Tuning (PEFT)**, nello specifico **LoRA (Low-Rank Adaptation)**. I risultati sono stati valutati rispetto a una **baseline matematica** (la correlazione della risposta media dei trial), dimostrando che l'approccio PEFT supera significativamente sia la baseline sia il solo addestramento dell'head.

Creazione del Dataset

La prima fase del progetto è consistita nella costruzione di un dataset personalizzato. Come richiesto dalla consegna, è stato utilizzato l'Allen Brain Observatory, un repository pubblico di dati di imaging a 2 fotoni che registra l'attività neuronale di topi esposti a diversi stimoli visivi.

L'osservatorio organizza gli esperimenti in "container", che tipicamente includono tre sessioni (A, B, e C/C2) sullo stesso topo, ognuna con stimoli diversi (es. gratings, video, immagini statiche).

L'obiettivo è stato quello di isolare dati compatibili con il modello ViV1T, che è pre-addestrato su stimoli video.

Selezione della Sessione

Dato che l'obiettivo è predire la risposta a *stimoli video*, è stato scelto di filtrare esperimenti di **Tipo A**, poiché questo è l'unico tipo di sessione che include video naturalistici (Natural Movie 1 e Natural Movie 3).

Un requisito fondamentale del modello ViV1T è la necessità di ricevere, oltre al video, dati comportamentali come la **velocità di corsa** e i dati di **eye-tracking** (posizione e dimensione della pupilla). Questi input sono importanti perché permettono al modello di produrre predizioni differenti per diversi *trial* dello stesso video.

È stato quindi ristretta la ricerca alle sole sessioni che includessero dati di eye-tracking affidabili, impostando require_eye_tracking=True e include_failed=False nei parametri di ricerca dell'SDK.

Attraverso questa scansione, è stata identificata la sessione **ID = 501729039**. Questa sessione, registrata in data 05/02/2016, è stata condotta sulla **corteccia visiva primaria (VISp)**, contiene misurazioni valide di eye-tracking e include un numero elevato di neuroni (**227**), rendendola una candidata ideale per il progetto.

Preprocessing dei Dati

Una volta identificata la sessione, il passo successivo è stato estrarre e trasformare i dati grezzi in un formato compatibile con l'architettura del modello ViV1T. Questo processo ha richiesto quattro passaggi principali:

1. Processing dei Video

ViV1T si aspetta in input clip video con dimensioni spaziali e temporali specifiche. I video originali della sessione (natural_movie_one di 900 frame e natural_movie_three di 3600 frame) avevano una risoluzione di 304x608 pixel.

- **Resize Spaziale:** Per prima cosa, ogni frame dei video è stato ridimensionato alle dimensioni attese dal modello, ovvero **36x64 pixel**.

```
from scipy import ndimage

videos = [stimulus_template_movie_one, stimulus_template_movie_three]
target_height = 36
target_width = 64

downsampled_videos = []
for video in videos:
    resized_frames = []
    for frame in video:
        zoom_factors = (target_height / frame.shape[0], target_width / frame.shape[1])
        new_frame = ndimage.zoom(frame, zoom_factors, order=1) #order = 1 Specifica il metodo di interpolazione bilineare
        resized_frames.append(new_frame)

    downsampled_videos.append(np.array(resized_frames))

pprint(downsampled_videos[0].shape)
pprint(downsampled_videos[1].shape)
```

- **Segmentazione Temporale:** Successivamente, i video (ora di 900x36x64 e 3600x36x64) sono stati segmentati temporalmente in clip più corte. La durata richiesta dal modello è di **140 frame** (circa 4.6 secondi). È stata creata una funzione di segmentazione per dividere i video lunghi in clip multiple da 140 frame. Questo ha prodotto 6 clip da natural_movie_one e 25 da natural_movie_three (scartando i frame finali non sufficienti a formare una clip completa).

```
def segment_data_into_clips(items,pos_num_frames=0, target_frames=140) -> (list):
    """prende un array di dati e lo segmenta in "clip" più piccoli tutti della stessa lunghezza."""
    total_frames = items.shape[pos_num_frames]
    num_items = total_frames // target_frames #scartiamo i frame rimanenti
    if len(items.shape) == 3: #(frame, altezza, larghezza).
        return [items[i*target_frames:(i+1)*target_frames, :, :] for i in range(num_items)]
    elif pos_num_frames==1: # le risposte neurali hanno 2 dimensioni (neuroni, frame).
        return [items[:, i*target_frames:(i+1)*target_frames] for i in range(num_items)]
    else:
        return [items[i*target_frames:(i+1)*target_frames, :] for i in range(num_items)]

clips_movie_one = segment_data_into_clips(downsampled_videos[0])
clips_movie_three = segment_data_into_clips(downsampled_videos[1])

print(f"Numero clip in natural_movie_'one': {len(clips_movie_one)}")
print(f"Numero clip in natural_movie_'three': {len(clips_movie_three)}")
```

```
Numero clip in natural_movie_'one': 6
Numero clip in natural_movie_'three': 25
```

2. Dati Comportamentali e Risposte (Labels)

Tenendo conto che ogni video è stato mostrato al topo 10 volte (10 *trial*), sono stati allineati i dati comportamentali e le risposte neuronali con le clip video create

Estrazione: Sono state estratte le seguenti serie temporali dall'oggetto *session_data*:

- **Labels:** Le tracce di fluorescenza corrette, usando get_corrected_fluorescence_traces(). Queste tracce sono valori non negativi che indicano l'attività del calcio intracellulare.
- **Behavior:** Una combinazione della velocità di corsa del topo e della dimensione della pupilla (area in pixel), ottenuta tramite *np.column_stack*.
- **Pupil Center:** Le coordinate (x, y) del centro della pupilla.

Allineamento e Offset: È stato necessario filtrare questi dati per allinearli ai 10 trial di ciascun video. Sono stati usati gli indici di start/end della *stimulus_epoch_table*. È emerso che tra un trial e l'altro erano presenti frame di "pausa" (es. 66 frame per natural_movie_one), che sono stati scartati per ottenere serie temporali continue e pulite per ciascuno dei 10 trial.

Segmentazione: Le serie temporali allineate (behavior, pupil center, labels) sono state a loro volta segmentate in finestre da 140 frame, corrispondenti a ogni clip video.

3. Creazione dei Set (Train/Validation/Test)

Per addestrare e valutare il modello, i 10 trial disponibili sono stati suddivisi come segue:

- **Train Set:** I primi 7 trial di natural_movie_one e natural_movie_three.
- **Validation Set:** L'8° trial di entrambi i video.
- **Test Set:** Gli ultimi 2 trial (9° e 10°) di entrambi i video.

Questa divisione è stata applicata alle liste di clip segmentate. Questo ha prodotto:

- **Train Set:** 217 campioni (42 da movie 1 + 175 da movie 3).
- **Validation Set:** 31 campioni (6 da movie 1 + 25 da movie 3).
- **Test Set:** 62 campioni (12 da movie 1 + 50 da movie 3).

4. Normalizzazione

Per evitare *data leakage*, tutte le statistiche (medie, min, max, std) sono state calcolate **esclusivamente sul Train Set** e poi applicate a tutti e tre i set (Train, Validation, Test).

1. **Imputazione NaN:** I valori mancanti (NaN) nei dati comportamentali sono stati sostituiti con la media calcolata sul train set.
2. **Scaling:** Come indicato dagli autori di ViV1T, è stato applicato:
    - **Min-Max Normalization:** Ai dati video, behavior e pupil_center.
    - **Z-Score (Std Scaling):** Alle labels (fluorescenza), dividendole per la deviazione standard del train set

Classe MouseDataset

Infine, i dati processati e normalizzati sono stati salvati su disco come *file .npy* individuali (un file per clip). È stata definita una classe MouseDataset che eredita da torch.utils.data.Dataset per caricare questi file al modello durante il training.

```
class MouseDataset(Dataset):
    def __init__(self, root_dir):
        # percorsi per le 4 cartelle di dati
        self.videos_dir = os.path.join(root_dir, 'videos')
        self.pupil_dir = os.path.join(root_dir, 'pupil_center')
        self.behavior_dir = os.path.join(root_dir, 'behavior')
        self.labels_dir = os.path.join(root_dir, 'labels')

        # Creo una lista "master" di tutti i file ID (es. '0', '1', '2', ...)
        #    ordinandoli numericamente (es. 1, 2, ... 10, 11)
        self.file_ids = sorted(
            [f.replace('.npy', '') for f in os.listdir(self.videos_dir) if f.endswith('.npy')],
            key=lambda x: int(x)
        )

    def __len__(self):...

    def __getitem__(self, i):
        #Carica un campione specifico i
        file_id = self.file_ids[i]

        # Carica i 4 file .npy corrispondenti usando lo STESSO file_id.
        # Questo garantisce che video, pupilla, comportamento e label
        # siano perfettamente allineati temporalmente.
        video = np.load(os.path.join(self.videos_dir, f'{file_id}.npy'))
        pupil = np.load(os.path.join(self.pupil_dir, f'{file_id}.npy'))
        behavior = np.load(os.path.join(self.behavior_dir, f'{file_id}.npy'))
        label = np.load(os.path.join(self.labels_dir, f'{file_id}.npy'))

        #Aggiunge una dimensione "canale" al video.
        # Forma: (140, 36, 64) -> (1, 140, 36, 64) [Canali, Tempo, Altezza, Larghezza]
        video = video[np.newaxis, ...]
```

```
#Restituisce un dizionario di Tensori PyTorch
return {
    'video': torch.from_numpy(video).float(),
    'pupil_center': torch.from_numpy(pupil).float().transpose(1,0),
    'behavior': torch.from_numpy(behavior).float().transpose(1,0),
    'labels': torch.from_numpy(label).float()
}
```

Ogni campione (sample) restituito da questa classe ha le seguenti dimensioni:

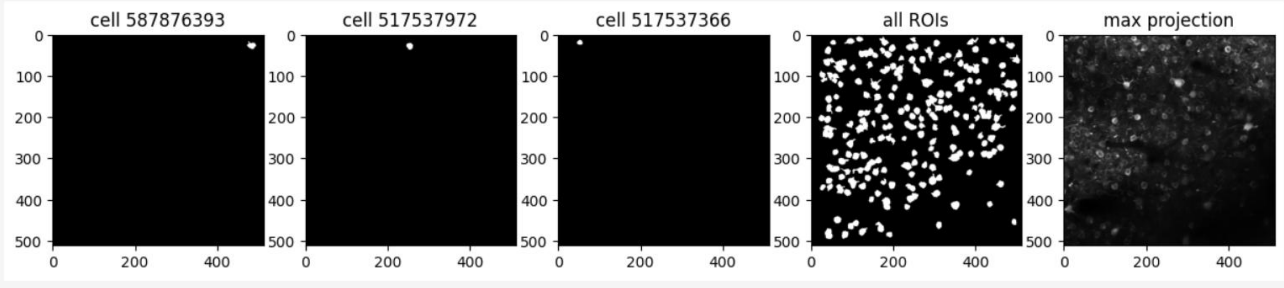
- video: (1, 140, 36, 64)
- behavior: (2, 140)
- pupil_center: (2, 140)
- labels: (227, 140)

Coordinate Neuronali

L'ultimo elemento fondamentale richiesto dal modello ViV1T è **l'input delle coordinate neuronali**. Il modulo *Readout* del modello, infatti, non apprende una predizione "globale", ma utilizza la posizione spaziale specifica di ciascun neurone per campionare in modo intelligente le feature estratte dalla griglia video (15x29).

Questi dati non sono disponibili direttamente tramite l'SDK Allen, ma possono essere derivati dalle maschere ROI (*Region of Interest*).

- **Estrazione:** Per ognuno dei 227 neuroni della sessione, sono state recuperate la sua maschera ROI (get_roi_mask_array), un'immagine 512x512 che mostra la posizione e la forma del neurone nel campo visivo del microscopio.
- **Calcolo del Centroide:** è stato calcolato il centroide (coordinate x, y) di ciascun neurone trovando la media delle posizioni dei pixel attivi (non-zero) all'interno della sua maschera ROI



- grafico 1-3: Questi primi tre grafici mostrano la forma e la posizione esatta di quei 3 neuroni specifici all'interno del campo visivo 512x512
- all ROIs: "appiattisce" i 227 livelli in un'unica immagine, mostrando la mappa completa di ogni singolo neurone registrato in questo esperimento
- max projection: Mostra il valore più luminoso che ogni pixel abbia mai raggiunto. È utile per vedere la struttura del tessuto e i vasi sanguigni

Trasformazione

Le coordinate grezze non sono compatibili con il sistema di coordinate su cui ViV1T è stato pre-addestrato, per questo sono state applicate due trasformazioni per replicare il sistema "Sensorium":

1. Le coordinate sono state scalate di un fattore **1.24** (per mappare la griglia 512x512 a una 620x620).
2. L'origine è stata spostata in alto a destra e i valori sono stati resi negativi.

```python
def convert_to_sensorium(roi_coords_tensor, factor=1.24):
    """
    Converts coordinates from top-left to Sensorium system (top-right, negative)
    """

    x_new = roi_coords_tensor[:, 0] * factor    #620
    y_new = roi_coords_tensor[:, 1] * factor    #620

    x_sensorium = -620 + x_new
    y_sensorium = -y_new

    return torch.stack([x_sensorium, y_sensorium], dim=1)
```

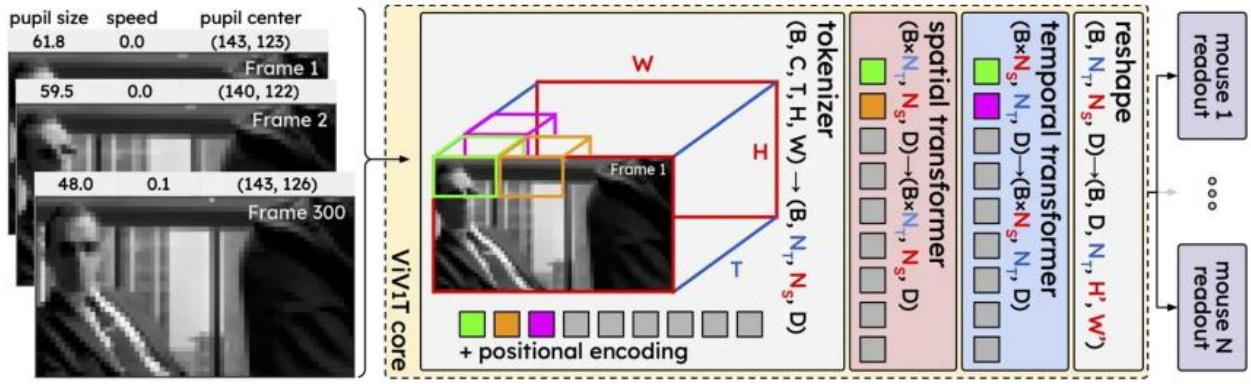
```python
neurons_coordinates_tensor = convert_to_sensorium(neurons_coordinates_tensor)
```

Modello e Architettura (ViV1T)

Come richiesto dalla consegna, il modello centrale di questo progetto è il **ViV1T (Video-Pretrained V1 Transformer)**. Questa architettura è progettata specificamente per il *brain encoding*, ovvero per mappare stimoli video complessi alle risposte dei singoli neuroni nella corteccia visiva (V1).

Il modello si compone di due parti principali: un "Core" universale pre-addestrato e una "Head" specifica per ciascun topo.

Architettura del Modello



L'immagine illustrare il processo di inferenza.

Per rappresentare in modo efficace un video come input per il modello, ViV1T utilizza i **Tubelet Embeddings**, ovvero patch tridimensionali estratte lungo le dimensioni **spaziali** e **temporali**. Questa scelta non è soltanto tecnica, ma ha una forte motivazione biologica: i neuroni visivi, in particolare quelli retinici e delle prime aree visive, rispondono non solo al contenuto spaziale di un'immagine, ma alla dinamica temporale del movimento. Gli stimoli visivi che cambiano nel tempo (ad esempio, un oggetto che si sposta, oppure contrasti che si modificano tra frame consecutivi) attivano pattern neurali che dipendono in modo critico dalla sequenza temporale.

Per questo motivo, ViV1T estende il concetto classico di patch 2D dei Vision Transformer ai **tubelet 3D**, che incorporano informazione temporale e preservano i pattern di movimento presenti nel video: invece di considerare ogni frame come un'entità isolata, il modello acquisisce piccoli "volumi" di video, contenenti più frame, in cui struttura spaziale e dinamica temporale sono strettamente connesse.

L'estrazione dei tubelet è affidata al componente chiamato **Tokenizer**, che riceve un tensore di forma **(B, C, T, H, W)** e applica un'operazione denominata **Unfold3d**.
Tecnicamente, Unfold3d può essere visto come l'estensione tridimensionale dell'operazione di estrazione di patch usata nei CNN o nei ViT tradizionali, ma agisce in modo congiunto su tutte e tre le dimensioni spaziotemporali.

In particolare, Unfold3d:

- scorre il video lungo il **tempo**,
- lungo l'**altezza** dell'immagine,
- e lungo la **larghezza**,

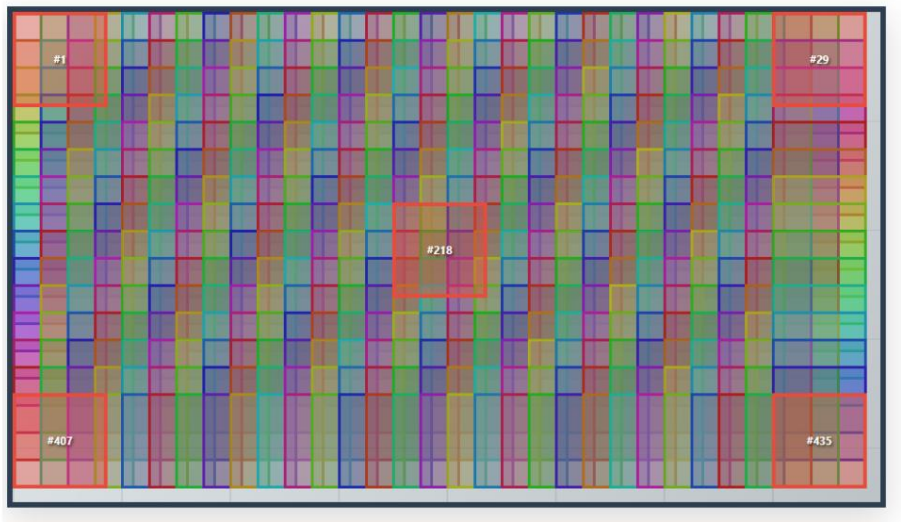
estraendo patch **sovrapposte** secondo i valori di *stride* e *size* specificati.
Questo significa che ogni tubelet è un piccolo blocco tridimensionale che contiene:

1. un insieme di frame consecutivi (profondità temporale),
2. una porzione locale dell'immagine (estensione spaziale),
3. e tutti i canali dell'input.

Abbiamo **spatial_patch_size = 7** e **spatial_patch_stride = 2**, il che significa che ogni patch spaziale ha dimensioni quadrate di **7×7 pixel** e che ogni patch è distanziata di **2 pixel** dalla successiva.
In altre parole, considerando un singolo frame, il numero di patch è dato da:

$$N_s = \left[\frac{H - \text{patch\_size}}{\text{stride}} + 1\right] \times \left[\frac{W - \text{patch\_size}}{\text{stride}} + 1\right]$$

Pertanto, per ogni frame vengono applicate **435 patch spaziali**.



Ora consideriamo anche l'asse temporale, partendo dalla conclusione: queste 435 patch vengono "estratte" **116 volte** a causa degli altri due parametri, **core_temporal_patch_size = 25** e **core_temporal_patch_stride = 1**.
Ciò significa che ogni patch spaziale in un dato frame ha una "profondità" pari a **25**, e a partire dalle stesse coordinate spaziali si sviluppano altre patch lungo l'asse temporale con uno stride di **1**.

In termini semplici, la **i-esima** patch lungo l'asse temporale inizia in una posizione che dista **1 unità** dalla posizione di inizio della patch **(i–1)**.

$$N_t = \frac{T - \text{temporal\_patch\_size}}{\text{temporal\_stride}} + 1$$

Ne consegue che abbiamo **116 patch temporali**; questo numero deve essere interpretato come "il numero di spostamenti che una finestra di ampiezza 25 può effettuare all'interno di una lunghezza totale di 140, muovendosi di una posizione alla volta".
Infatti, nella formula si calcola prima la differenza tra la lunghezza totale e la dimensione della finestra; poi si divide questa quantità per il passo (*stride*), ottenendo di fatto il numero di possibili spostamenti; infine si aggiunge 1 per includere anche la posizione iniziale.

Alla fine dell'operazione **Unfold3d**, otteniamo una forma di output in cui l'ultima dimensione corrisponde alla dimensione di un singolo tubelet appiattito, che in questo caso è **6125,** calcolato come il prodotto di 5 × 25 × 7 × 7.

dove:

- **5** è il numero di canali (**C**) del video di input.
- **25** è la profondità temporale, determinata da *core_temporal_patch_size*.
- **7 × 7** è la dimensione spaziale del patch, determinata da *core_patch_size*.

Ogni tubelet rappresenta quindi un piccolo volume 3D di stimolo composto da 25 frame consecutivi e 5 canali, su un'area locale di 7×7 pixel.

Successivamente, un livello di **embedding** trasforma ogni tubelet dalla sua rappresentazione grezza (6125 dimensioni) in una rappresentazione vettoriale compatta (112 dimensioni).

La scelta di **112** deriva da una combinazione di criteri progettuali:

- **Efficienza computazionale:** una dimensione troppo elevata avrebbe un impatto significativo sulla complessità del Transformer (sia in memoria sia in tempo di calcolo).
  112 rappresenta un compromesso che mantiene sufficiente capacità rappresentativa senza rendere il modello troppo pesante.
- **Coerenza con la configurazione originale del ViV1T:** si tratta di un valore usato dagli autori come *hidden size* del modello base, ottimizzato empiricamente per bilanciare performance e costi computazionali.
- **Trade-off prestazionale:** dimensioni dell'embedding più grandi possono in teoria rappresentare informazioni più ricche, ma aumentano rapidamente il rischio di overfitting, soprattutto con dataset biologici dove le osservazioni non sono numerosissime.
  112 si è dimostrato un valore adeguato nel contesto del nostro dataset.

Di conseguenza, otteniamo un output con forma **(B, Nt, Ns, D)** dove:

- **B** è la dimensione del batch
- **Nt** è il numero di patch temporali (116)
- **Ns** è il numero di patch spaziali per frame (435)
- **D** è la dimensione dell'embedding compatto per ogni tubelet (112)

Spatial Transformer

Il modello ViViT implementato dagli autori è quello con **encoder fattorizzato**, che consiste quindi in un transformer spaziale seguito da un transformer temporale.

Lo **Spatial Transformer** ha la funzione principale di elaborare le relazioni spaziali all'interno di ciascun frame del video.
Esso elabora le 435 patch in modo indipendente per ciascun indice temporale (0-115).

In particolare, riceve un input con forma **(B×Nt, Ns, D)** e, per ogni posizione temporale (Nt), calcola l'attenzione tra le **Ns** patch appartenenti allo stesso frame (selezionato in base all'indice Nt).

L'output di questo componente consiste ancora in **435 patch** per tutti gli indici temporali (116), ma ora ogni patch contiene informazioni di correlazione spaziale con le altre.

Temporal Transformer

L'input di questo componente è l'output del trasformatore spaziale, ma **ristrutturato**; in particolare, riceve un tensore con forma **(B×Ns, Nt, D)**.
Qui, ogni posizione spaziale viene trattata come un campione indipendente del batch, permettendo al meccanismo di attenzione di catturare le dipendenze temporali lungo la sequenza di 116 frame per ciascuna specifica posizione.

Reshape

Riceve in input un tensore di forma **(B, Nt, Ns, D)** e produce un tensore di output con forma **(B, D, Nt, H', W')**, dove **H' = 15** e **W' = 29**.

Dal punto di vista spaziale, l'informazione bidimensionale viene reintrodotta (pur preservando i dati) modificando la struttura rappresentazionale.
In particolare, **15** e **29** rappresentano il numero di patch rispettivamente lungo altezza e larghezza (15 × 29 = 435).

Questa trasformazione converte la rappresentazione sequenziale prodotta dal transformer (dove le 435 patch sono trattate come una sequenza lineare) in una rappresentazione spaziale necessaria per i moduli successivi, che eseguono operazioni come il calcolo delle distanze tra patch, fondamentali per mappare correttamente le caratteristiche visive ai neuroni.

MLP Shifter & Readout

ViV1T include un **MLPShifter** e un **Readout** per ogni topo, poiché questi componenti sono personalizzati per gestire un numero specifico di neuroni (che può variare tra topi diversi).
Poiché il nostro dataset consiste in un singolo esperimento condotto su un solo topo, il nostro modello ha un solo Shifter e un solo Readout.

MLP Shifter

Lo Shifter agisce come un meccanismo di correzione dinamica dei movimenti oculari del topo durante la presentazione del video: i movimenti oculari introducono spostamenti sistematici nel campo visivo percepito, che devono essere compensati per ottenere predizioni più accurate.

Più precisamente, quando il topo muove gli occhi, ogni elemento del video viene percepito dai neuroni retinici in una posizione spostata rispetto a quella reale.
Ad esempio, se un oggetto appare alle coordinate (10, 15) nel video ma il topo sposta lo sguardo di 2 pixel a destra e 1 pixel verso l'alto, i neuroni lo percepiranno come se fosse a (12, 16).
Questo crea un disallineamento sistematico tra dove pensiamo che il topo stia guardando e dove sta realmente guardando.

Dal punto di vista pratico, lo Shifter **predice gli spostamenti oculari** basandosi su dati comportamentali (posizione della pupilla, velocità, dilatazione), producendo **(Δx, Δy)** per ogni frame temporale, da cui la forma di output **(116, 2)**.

Readout

Il componente Readout riceve in input tensori con forma **(1, 112, 116, 15, 29)** provenienti dal modulo Rearrange e produce un output di forma **(1, 227, 116)**, che rappresenta la predizione grezza.

Esso opera su ciascun neurone calcolando, tramite le sue coordinate, dove esso dovrebbe "guardare" all'interno della griglia 15×29.

Chiaramente, questa mappatura viene appresa durante l'allenamento ed è supportata dall'integrazione delle informazioni comportamentali: se ogni neurone ha una finestra di "osservazione" preferita sulla griglia 15×29 (determinata dalle sue coordinate anatomiche nel cervello), quando il topo muove gli occhi in una direzione, il modello compensa questo movimento spostando la finestra di osservazione nella direzione opposta.

Allo stesso modo:

- se la velocità di corsa del topo cambia, l'intensità della finestra gaussiana viene modulata;
- se la dilatazione pupillare varia, cambia anche la forma della finestra.

Il concetto di finestra viene implementato tramite **funzioni di pooling gaussiane**: per ogni neurone, il modello crea una "campana" gaussiana centrata sulla sua posizione preferita all'interno della griglia 15×29.
Ciò determina il peso assegnato a ciascuna cella della griglia nel calcolo dell'attivazione del neurone.

Ovviamente, questo processo avviene per ogni frame temporale.

L'output del Readout deve quindi essere interpretato come l'applicazione della finestra di ciascun neurone alle caratteristiche presenti nella griglia: in altre parole, **tutti i valori della griglia vengono presi, moltiplicati per i pesi gaussiani e poi sommati**.
Questo valore finale rappresenta quanto quel neurone dovrebbe attivarsi in base a ciò che "vede" attraverso la sua finestra.

Adattamento del Modello

Per poter utilizzare ViV1T, è stato necessario adattarlo al nostro dataset e, per farlo, abbiamo modificato gli "args" da passare al costruttore della classe. In particolare, abbiamo proceduto con la rimozione di 9 "mouse_ids" (lasciando soltanto quello con id: "A") e abbiamo adattato di conseguenza gli "output shapes" (dove viene specificato il numero di neuroni considerati per ciascun id).
Questa scelta ci ha permesso di ottenere un modello più leggero in termini di parametri caricati (3M contro 12M) e che formula predizioni basandosi sulla configurazione cerebrale del singolo topo che abbiamo analizzato.
Di conseguenza, la *head* del nostro modello è composta da uno **Shifter** e da un **Readout**.

Il processo di adattamento è consistito nei seguenti passaggi:

- **Caricamento del modello pre-addestrato**: inizialmente abbiamo istanziato il modello con i parametri di configurazione e le coordinate dei neuroni.
- **Caricamento dei pesi del ViV1T**: abbiamo sfruttato i pesi pre-addestrati del ViV1T per effettuare transfer learning; tuttavia, poiché la *head* del nostro modello è diversa da quella del ViV1T (in quanto consideriamo un topo differente), dal checkpoint abbiamo recuperato esclusivamente i pesi del *ViV1T-core*, lasciando invece MLP Shifter e Readout con pesi inizializzati da zero.
- Freezing di tutti i parametri del modello tranne la head.

Come metrica per la selezione del modello durante tutti gli addestramenti è stata utilizzata la media delle *single_trial_correlation* sul *test set*.

La **single_trial_correlation** misura la correlazione tra la **predizione del modello** e l'**attività neurale osservata** per ciascun trial individuale. In pratica, per ogni neurone e per ciascun trial, calcoliamo la **correlazione di Pearson** tra la sequenza temporale della predizione e la sequenza temporale osservata dell'attività neurale. Successivamente, queste correlazioni vengono mediate su tutti i trial e su tutti i neuroni per ottenere un singolo valore di performance, rappresentativo della capacità del modello di catturare le risposte neurali in modo coerente a livello trial-specifico.

Per l'addestramento, il modello è stato ottimizzato utilizzando la **Mean Squared Error (MSE) Loss**

Metodologia di Training e Valutazione

Per addestrare e valutare il modello adattato, è stato necessario definire una baseline di riferimento e personalizzare l'infrastruttura di training di Hugging Face per renderla compatibile con l'architettura di ViV1T e con le nostre metriche specifiche.

Definizione della Baseline

Come richiesto dalla consegna, è stata stabilita una baseline "model-independent" per quantificare il miglioramento apportato dal modello. È stato scelto un approccio basato sulla **media delle risposte dei trial di addestramento**.

La logica è la seguente: la predizione per una data clip video (es. clip 1 del test set) è semplicemente la risposta media registrata per quella stessa clip durante i 7 trial di training. La performance della baseline è quindi la **correlazione di Pearson** tra questa "predizione media" e le risposte *reali* dei trial di test.

Questa metrica è stata calcolata su tutti i campioni del test set (sia per il trial 8 che per il trial 9) ed è stata calcolata la media, ottenendo un valore di correlazione finale della baseline pari a **0.0235**. Qualsiasi modello con uno score superiore a questo valore sta imparando a spiegare la varianza dei dati oltre la semplice media.

Adattamento del Trainer di Hugging Face

Il Trainer standard di Hugging Face non è compatibile "out-of-the-box" con ViV1T. La forward signature del modello (inputs, mouse_id, behaviors, pupil_centers) è diversa da quella attesa dal trainer (che cerca input_ids, attention_mask, ecc.). Inoltre, si doveva definire una logica custom per il calcolo della loss e della metrica di valutazione.

Per risolvere questo, è stato adottato un approccio a tre componenti:

1. ViV1TTrainerWrapper

È stata creato una classe ViViTTrainerWrapper che "avvolge" il modello ViV1T

Questo wrapper gestisce il calcolo della loss:

1. Riceve in input i tensori dal MouseDataset (video, behavior, pupil_center, labels).
2. Passa gli input al modello ViV1T interno (es. lora_model).
3. Ottiene le predizioni (logits).
4. Se sono presenti le labels (cioè, in fase di training o eval), allinea temporalmente predizioni e label e calcola la **MSE Loss**.
5. Restituisce un dizionario contenente loss e logits, come atteso dal Trainer.

```python
class ViViTTrainerWrapper(nn.Module):
    def __init__(self, lora_model, mouse_id='A'):
        super().__init__()
        self.lora_model = lora_model
        self.mouse_id = mouse_id
        self.mse_loss = nn.MSELoss()

    def forward(self, video, behavior, pupil_center, labels=None, **kwargs):
        """
        La firma ora accetta esplicitamente gli output del MouseDataset.
        'labels' è opzionale (None) perché non è presente durante l'inferenza pura,
        ma è presente durante il training e la valutazione.
        """

        # Non c'è più bisogno di kwargs.get()
        # 'inputs' nel modello interno (lora_model) si aspetta il tensore video
        predictions, _ = self.lora_model(
            inputs=video,
            mouse_id=self.mouse_id,
            behaviors=behavior,
            pupil_centers=pupil_center
        )

        loss = None
        if labels is not None:
            # Allineamento temporale
            # Il modello riceve 140 frame, ma ne predice solo 66.
            min_frames = 66
            # Estraggo gli ultimi 66 frame dalle etichette (labels)
            labels_aligned = labels[..., -min_frames:]
            # Estraggo gli ultimi 66 frame dalle predizioni
            predictions_aligned = predictions[..., -min_frames:]
```

```python
        #Calcolo l'errore (MSE) solo su quei 66 frame allineati
        loss = self.mse_loss(predictions_aligned, labels_aligned)

    return {
        "loss": loss,
        "logits": predictions,
    }
```

2. CustomViViTTrainer

È stata creata una sottoclasse di Trainer sovrascrivendo unicamente il *prediction_step*. Questo assicura che durante la valutazione (trainer.evaluate()) venga utilizzata la nostra logica forward personalizzata per generare le predizioni (logits) e le labels corrette da passare alla funzione compute_metrics.

3. compute_metrics

Infine, è stata implementato la funzione compute_metrics, richiesta dalla consegna per calcolare la metrica di "*average single trial correlation*".

Questa funzione riceve le predizioni e le label (già allineate temporalmente grazie al prediction_step), quindi:

1. Itera su ogni campione (clip) nel batch di valutazione.
2. Itera su ogni neurone (227) di quel campione.
3. Calcola la correlazione di Pearson (np.corrcoef) tra la serie temporale predetta e quella reale per quel singolo neurone.
4. Calcola la media delle correlazioni di tutti i neuroni per quel campione.

5. Infine, restituisce la media di queste medie per l'intero batch.

```python
def compute_metrics(eval_pred):
    predictions, labels = eval_pred

    # Allineamento Temporale
    # come nel wrapper della loss valutiamo solo gli ultimi 66 frame, che sono quelli validi.
    min_frames = 66
    labels_aligned = labels[..., -min_frames:]
    predictions_aligned = predictions[..., -min_frames:]


    example_correlations = []

    # Loop per ogni Campione (clip) nel Batch
    for example_idx in range(labels_aligned.shape[0]):

        # Trasponiamo i dati da (neuroni, tempo) a (tempo, neuroni)
        # per facilitare il loop successivo.
        y_true_example = labels_aligned[example_idx].T
        y_pred_example = predictions_aligned[example_idx].T

        correlations = []

        #Loop per ogni Neurone (da 0 a 226)
        for neuron in range(y_true_example.shape[1]):
            # Estrae la serie temporale (66 frame) per UN singolo neurone
            true_vals = y_true_example[:, neuron]
            pred_vals = y_pred_example[:, neuron]

            # Controlla che ci sia varianza (non una linea piatta) per evitare errori
            true_var = np.var(true_vals)
            pred_var = np.var(pred_vals)
```

```python
        if true_var > 1e-10 and pred_var > 1e-10:
            # Calcola la Correlazione di Pearson tra la predizione e la realtà
            corr = np.corrcoef(true_vals, pred_vals)[0, 1]

            # Aggiunge solo se il risultato è un numero valido
            if not np.isnan(corr) and np.isfinite(corr):
                correlations.append(corr)

    # Calcola la correlazione media di tutti i neuroni per questo campione
    mean_corr_example = np.mean(correlations) if correlations else 0.0
    example_correlations.append(mean_corr_example)

# Calcola la media di tutti i campioni nel batch
overall_mean = np.mean(example_correlations) if example_correlations else 0.0

return {
    'eval_average_single_trial_correlation': overall_mean,
    'eval_single_trial_std': np.std(example_correlations) if example_correlations else 0.0,
    'eval_num_examples': len(example_correlations),
}
```

RISULTATI

MLP Shifter & Readout

Inizialmente, sono stati addestrati esclusivamente i moduli **MLP Shifter** e **Readout**, mantenendo congelati i pesi del *ViV1T core* pre-addestrato. Questa strategia è stata adottata per stabilizzare i pesi della "testa" del modello prima di procedere al fine-tuning dell'intera architettura.

Date le limitazioni delle risorse GPU a disposizione, sono state mantenute le scelte effettuate in fase di pre-processing: segmentazione dei video in clip e un batch_size = 1.

Per questo esperimento specifico, sono stati utilizzati i seguenti iperparametri:

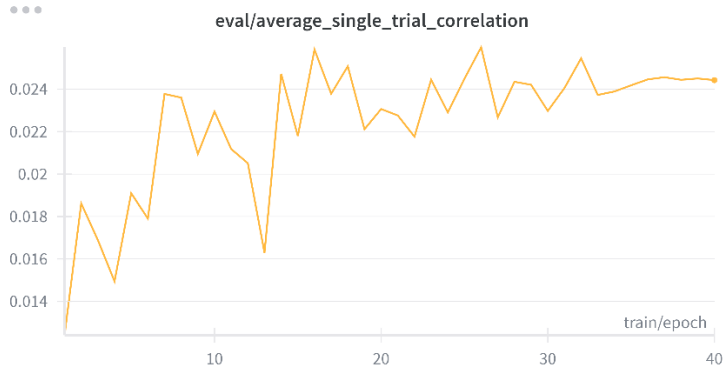
- **Learning Rate:** 0,001
- **Scheduler:** Cosine Annealing
- **Epochs:** 40
- **Loss Function:** MSE Loss
- **Weight Decay:** 0.01

Di seguito vengono riportati i grafici risultanti dal training che ha ottenuto il miglior punteggio sul validation set:

Analisi della Loss (Train/Loss)

Dall'osservazione del grafico della loss, si nota una discesa rapida nelle primissime epoche (da circa 1.0 a 0.3), seguita da una fase di stabilizzazione attorno a valori compresi tra 0.2 e 0.3. È evidente un andamento molto oscillante e "rumoroso" della curva. Tale comportamento è direttamente imputabile all'utilizzo di un batch_size=1: la stima del gradiente su un singolo campione risulta intrinsecamente rumorosa e porta il modello a "rimbalzare" durante la discesa del gradiente, pur mantenendo un trend generale di convergenza.

Analisi della Correlazione (Eval/Average Single Trial Correlation)



Il grafico della correlazione media sui singoli trial mostra un trend crescente, seppur irregolare.

- Partendo da valori bassi (~0.010), il modello guadagna capacità predittiva rapidamente.
- Intorno all'epoca 26 si raggiunge un picco di performance, con una correlazione che tocca valori prossimi a **0.026**.
- Verso la fine del training (epoca 40), la curva si stabilizza attorno a **0.024 - 0.025**.

Risultati e Conclusioni

Nonostante le curve di training mostrassero una convergenza apparente, la valutazione finale del modello sul **test set** ha riportato un valore di correlazione media sui singoli trial pari a **0.0078**.

Tale risultato risulta significativamente inferiore alla **baseline matematica** di riferimento (calcolata a **0.0235**). Questo indica che il modello, nella configurazione *Head-Only*, non è riuscito a generalizzare efficacemente sui dati non visti, fornendo prestazioni peggiori rispetto alla semplice predizione basata sulla media delle risposte. Si deduce, pertanto, che il solo addestramento dello Shifter e del Readout non è sufficiente per adattare le rappresentazioni del *ViV1T core* (pre-addestrato su altri dati) alla specifica fisiologia del topo in esame. Per ottenere risultati soddisfacenti, appare necessario sbloccare e adattare anche i pesi del *core* tramite tecniche di *Fine-Tuning* più profonde (come PEFT/LoRA).

PEFT (LoRA)

Per migliorare le prestazioni del modello, è stato scelto di adottare una tecnica basata su **PEFT (Parameter-Efficient Fine-Tuning)**, nello specifico **LoRA (Low-Rank Adaptation)**.

Inizialmente, sono stati individuati i moduli lineari più appropriati da adattare:

- **fused_linear**: particolarmente interessante perché rappresenta **un'ottimizzazione computazionale** dell'architettura Transformer.
  Invece di avere tre operazioni lineari separate per **query**, **key** e **value** (più una proiezione aggiuntiva per la rete feed-forward), questo modulo calcola tutte e quattro le trasformazioni contemporaneamente.
  Questa fusione crea anche un punto di controllo unico dove LoRA può influenzare simultaneamente tutti i meccanismi di attenzione e di elaborazione.

- **attn_out**: proiezione aggregata finale dell'output della **multi-head attention**. Determina come le informazioni elaborate dalle diverse teste di attenzione vengono combinate e reintegrate nel flusso principale del modello.

- **ff_out.2**: corrisponde all'ultimo layer lineare della rete feed-forward. È il modulo responsabile della trasformazione finale che riporta le rappresentazioni elaborate nello spazio dimensionale originale.

```
LORA_CONFIGS = {
    8: {
        "alpha": 16,
        #"target_modules": ["fused_linear"],
        "target_modules": ["fused_linear", "attn_out", "ff_out.2"],
        #"learning_rate": 0.001,
        "learning_rate": 0.0036,
        #"scheduler": "cosine",
        "scheduler": "linear",
        #"description": "fused_only_cosine"
        "description": "all_layers_linear"
    },
    16: {
        "alpha": 32,
        #"target_modules": ["fused_linear", "attn_out", "ff_out.2"], # "All" layers
        "target_modules": ["fused_linear"],
        "learning_rate": 0.001,
        "scheduler": "linear",
        #"description": "all_layers_linear"
        "description": "fused_linear"
    },
    32: {
        "alpha": 64,
        #"target_modules": ["fused_linear", "attn_out", "ff_out.2"], # "All" layers
        "target_modules": ["fused_linear"], # Solo "Fused"
        #"learning_rate": 0.0036,
        "learning_rate": 0.001,
        #"scheduler": "linear",
        "scheduler": "cosine",
        #"description": "all_layers_linear_highLR"
        "description": "fused_only_cosine"
    }
}
```

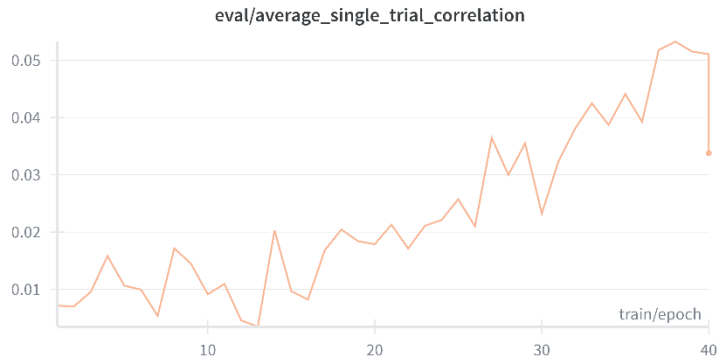
LoRA-8 (All Linear Layers)

Il primo esperimento condotto con questa metodologia ha mirato a massimizzare la capacità di adattamento del modello mantenendo un *Rank* contenuto. È stata scelta una configurazione che applica l'adattamento a **tutti i moduli lineari** del trasformatore (fused_linear, attn_out, ff_out.2), permettendo a LoRA di influenzare sia i meccanismi di attenzione che le reti feed-forward.

Di seguito i dettagli della configurazione e degli iperparametri utilizzati:

- **Rank (r):** 8
- **Alpha (alpha):** 16
- **Target Modules:** All Linear Layers
- **Learning Rate (LR):** 0.0036
- **Scheduler:** Linear
- **Trainable Parameters:** 5.56%

Di seguito vengono riportati i grafici relativi all'andamento del training per questa configurazione:



Analisi delle Curve

- **Loss:** La curva di loss mostra una discesa rapida e costante, stabilizzandosi su valori inferiori rispetto al training *Head-Only*. Nonostante permanga la rumorosità dovuta al batch size unitario, il trend di apprendimento appare molto più robusto.
- **Correlazione:** A differenza del tentativo precedente, la curva di correlazione sul validation set mostra una crescita sostenuta. Intorno all'epoca 15, il modello supera la soglia della baseline (circa 0.023) e continua a migliorare costantemente, raggiungendo picchi superiori a **0.05** verso la fine del training. L'utilizzo di uno scheduler Linear con un Learning Rate iniziale più aggressivo (0.0036) sembra aver favorito una rapida uscita dai minimi locali iniziali.

Risultati sul Test Set La valutazione finale del modello, utilizzando il checkpoint migliore salvato durante il training (checkpoint-8246), ha prodotto i seguenti risultati:

- **Risultato Test Set:** 0.0343
- **Baseline Matematica:** 0.0235

Conclusioni parziali L'introduzione di LoRA ha portato a un miglioramento decisivo. Il modello ha ottenuto uno score sul test set di **0.0343**, superando la baseline matematica di un fattore di circa **1.46 volte**.

Questo risultato conferma che sbloccare i pesi del *core* tramite LoRA permette al modello di riallineare le feature visive apprese durante il pre-training (su altri topi/dataset) alla specifica

topologia neuronale e comportamentale del topo in esame. Il modello non si limita più a predire la media dell'attività, ma inizia a catturare efficacemente le modulazioni specifiche dell'attività neuronale in risposta agli stimoli visivi.

Configurazione LoRA-8 (Fused Linear Only)

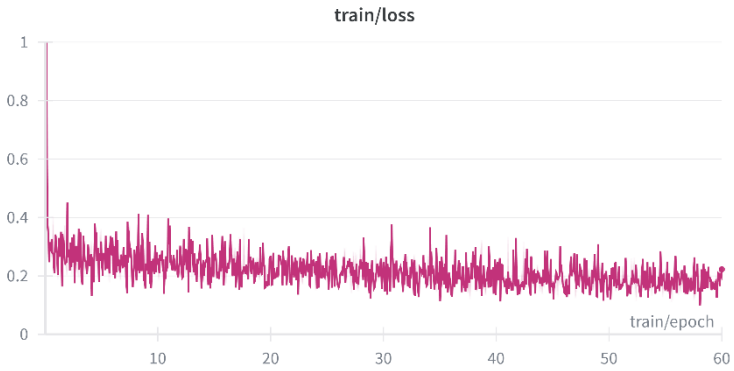
Nel tentativo di ottimizzare ulteriormente il processo di fine-tuning e ridurre il rischio di overfitting, è stato condotto un secondo esperimento mantenendo il Rank a 8, ma restringendo l'applicazione di LoRA esclusivamente ai moduli **fused_linear**. Questi moduli gestiscono le proiezioni di *Query, Key e Value* all'interno del meccanismo di attenzione, e sono ritenuti cruciali per l'adattamento delle rappresentazioni visive.

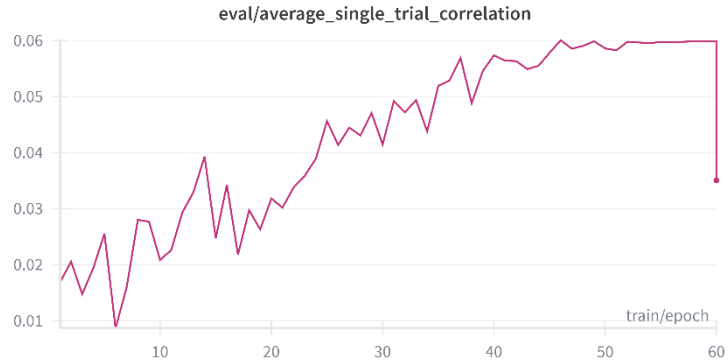
Inoltre, per favorire una convergenza più stabile, si è passati da uno scheduler lineare a uno di tipo **Cosine Annealing**, prolungando il training fino a 60 epoche.

Di seguito i dettagli della configurazione:

- **Rank (r):** 8
- **Alpha (alpha):** 16
- **Target Modules:** Fused Linear Only
- **Learning Rate (LR):** 0.001
- **Scheduler:** Cosine
- **Trainable Parameters:** 3.8%

Di seguito vengono riportati i grafici relativi all'andamento del training:





Analisi delle Curve

- **Loss:** La curva di loss mantiene il caratteristico andamento oscillatorio dovuto al batch size unitario, ma mostra un trend di discesa costante per tutta la durata delle 60 epoche.
- **Correlazione:** L'utilizzo dello scheduler *Cosine* ha prodotto una curva di apprendimento sul validation set molto regolare e crescente. Il modello ha continuato a migliorare le proprie prestazioni ben oltre la trentesima epoca, raggiungendo picchi di correlazione prossimi a **0.06** intorno all'epoca 45-50, prima di una lieve flessione finale. Questo suggerisce che la riduzione progressiva del Learning Rate ha permesso al modello di raffinare i pesi in modo efficace.

Risultati sul Test Set La valutazione finale del modello (checkpoint-9982) ha prodotto i seguenti risultati:

- **Risultato Test Set:** 0.0321
- **Baseline Matematica:** 0.0235

Conclusioni parziali Anche questa configurazione ha superato la baseline matematica, confermando la validità dell'approccio. Tuttavia, il punteggio ottenuto (**0.0321**) risulta leggermente inferiore rispetto alla configurazione precedente che coinvolgeva tutti i layer lineari (**0.0343**).

Ciò indica che, sebbene i moduli di attenzione (*fused_linear*) giochino il ruolo principale nell'adattamento al dominio, l'esclusione dei moduli Feed-Forward (*ff_out*) e di output dell'attenzione (*attn_out*) ha limitato lievemente la capacità espressiva del modello nel mappare le feature visive all'attività neuronale specifica. Nonostante ciò, il risultato rimane positivo, ottenendo performance comparabili con un numero inferiore di parametri addestrabili (3.8% contro 5.56%).

Configurazione LoRA-16 (All Linear Layers)

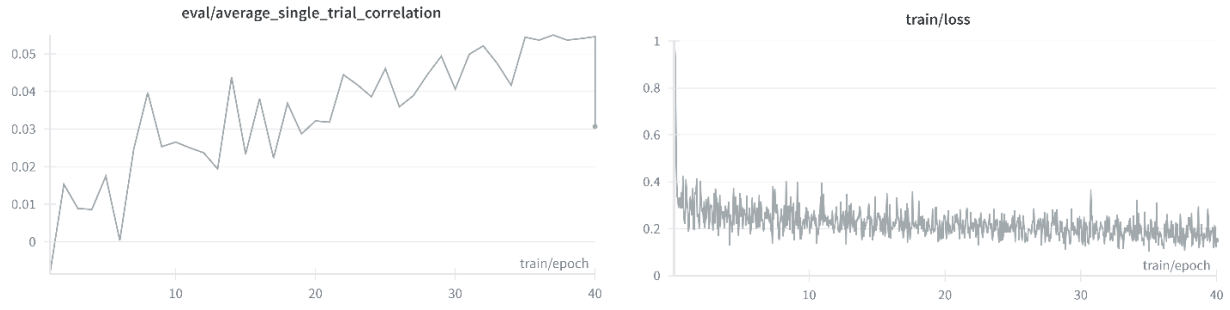
Per valutare l'impatto della capacità del modello sull'apprendimento, è stato effettuato un esperimento incrementando il Rank a 16. Mantenendo l'approccio iniziale, l'adattamento è stato applicato a **tutti i layer lineari**. L'obiettivo era verificare se un maggiore numero di parametri addestrabili (passando dal 5.56% al 10.53%) potesse catturare dettagli più fini della dinamica neuronale.

Per compensare l'aumento di complessità, il Learning Rate è stato ridotto a 0.001, mantenendo uno scheduler lineare.

Di seguito i dettagli della configurazione:

- **Rank (r):** 16
- **Alpha (alpha):** 32
- **Target Modules:** All Linear Layers
- **Learning Rate (LR):** 0.001
- **Scheduler:** Linear
- **Trainable Parameters:** 10.53%

Di seguito vengono riportati i grafici relativi all'andamento del training:



Analisi delle Curve

- **Loss:** La loss mostra una convergenza stabile, con valori che si assestano attorno a 0.2. La riduzione del Learning Rate ha reso la discesa meno ripida nelle fasi iniziali rispetto alla configurazione LoRA-8, ma ha garantito un andamento più controllato.

- **Correlazione:** Sul validation set, il modello ha mostrato buone capacità di generalizzazione, con una curva di correlazione in crescita costante. Si notano picchi interessanti (superiori a **0.05**) verso la fine del training, segno che il modello stava ancora apprendendo efficacemente. Tuttavia, la varianza tra le epoche appare leggermente più marcata rispetto alle configurazioni precedenti.

Risultati sul Test Set La valutazione finale del modello (checkpoint-8029) ha prodotto i seguenti risultati:

- **Risultato Test Set:** 0.0306
- **Baseline Matematica:** 0.0235

Conclusioni parziali Anche con Rank 16 il modello supera agevolmente la baseline matematica. Tuttavia, il punteggio ottenuto (**0.0306**) è inferiore a quello raggiunto con LoRA-8 (0.0343).

Questo risultato suggerisce che l'aumento del numero di parametri addestrabili (quasi raddoppiati rispetto a LoRA-8) non si è tradotto in un guadagno di prestazioni sul test set. È probabile che, dato il dataset relativamente piccolo, un modello più complesso tenda maggiormente all'overfitting, specialmente quando vengono adattati tutti i layer.

Configurazione LoRA-16 (Fused Linear Only)

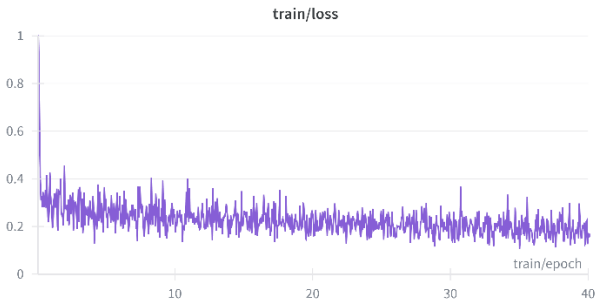
Analogamente a quanto fatto per Rank 8, è stata testata una configurazione di Rank 16 che limita l'applicazione di LoRA ai soli moduli **fused_linear**. Questa scelta mirava a verificare se, concentrando l'adattamento sui meccanismi di attenzione con una maggiore capacità (Rank più alto), si potesse ottenere un compromesso migliore rispetto all'adattamento di tutti i layer.

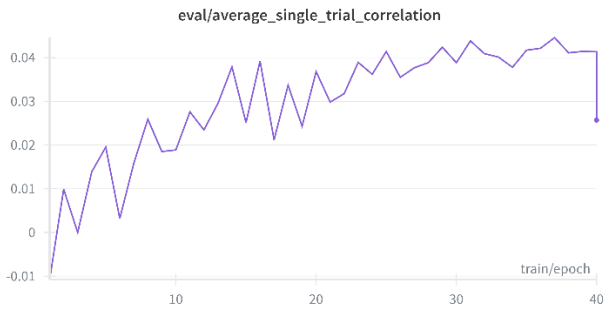
In questo caso, per coerenza con la configurazione precedente di Rank 16, è stato mantenuto un Learning Rate di 0.001 e si è adottato uno scheduler *Cosine* per favorire la stabilità.

Di seguito i dettagli della configurazione:

- **Rank (r):** 16
- **Alpha (alpha):** 32
- **Target Modules:** Fused Linear Only
- **Learning Rate (LR):** 0.001
- **Scheduler:** Cosine
- **Trainable Parameters:** 7.35%

Di seguito vengono riportati i grafici relativi all'andamento del training:





Analisi delle Curve

- **Loss:** La curva di loss mostra un comportamento simile alle altre configurazioni, con una discesa iniziale seguita da oscillazioni attorno a un valore di convergenza.
- **Correlazione:** Sul validation set, l'andamento della correlazione appare meno robusto rispetto alla controparte con Rank 8. Sebbene il trend sia positivo e crescente, si notano fluttuazioni più marcate e i valori massimi raggiunti (attorno a **0.04**) sono inferiori rispetto ai picchi osservati in altre configurazioni.

Risultati sul Test Set La valutazione finale del modello (checkpoint-8029) ha prodotto i seguenti risultati:

- **Risultato Test Set:** 0.0261
- **Baseline Matematica:** 0.0235

Conclusioni parziali Il modello supera la baseline matematica anche in questa configurazione, confermando che l'approccio PEFT è efficace. Tuttavia, il punteggio ottenuto (**0.0261**) è il più basso tra le configurazioni LoRA testate finora.

Questo risultato, inferiore sia alla configurazione "All Linear" con lo stesso Rank (0.0306) che alle configurazioni con Rank 8, suggerisce che per un Rank più elevato (16), limitare l'adattamento ai soli layer *Fused Linear* potrebbe non essere sufficiente a sfruttare appieno la capacità aggiuntiva. Al contrario, potrebbe introdurre un collo di bottiglia dove la maggiore espressività nelle matrici di attenzione non è supportata da un adeguato adattamento nelle proiezioni di output o nei blocchi feed-forward.

Configurazione LoRA-32 (All Linear Layers)

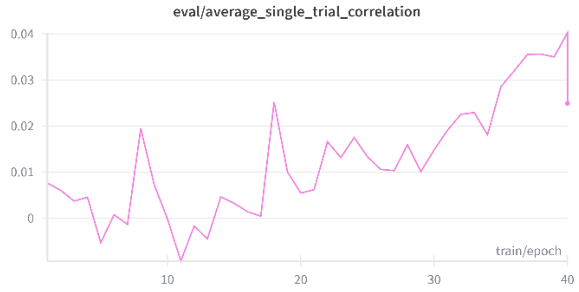
Per completare l'analisi sull'impatto del Rank, è stato condotto un ultimo esperimento incrementando il Rank a 32 e applicando l'adattamento a **tutti i layer lineari**. Questa configurazione rappresenta quella con la maggiore capacità espressiva tra quelle testate, portando la percentuale di parametri addestrabili al 19.1%.

Per sostenere un tale aumento di complessità, si è tornati a un Learning Rate più elevato (0.0036), simile a quello utilizzato con successo nella configurazione LoRA-8, mantenendo uno scheduler lineare.

Di seguito i dettagli della configurazione:

- **Rank (r):** 32
- **Alpha (alpha):** 64
- **Target Modules:** All Linear Layers
- **Learning Rate (LR):** 0.0036
- **Scheduler:** Linear
- **Trainable Parameters:** 19.1%

Di seguito vengono riportati i grafici relativi all'andamento del training:



Analisi delle Curve

- **Loss:** La curva di loss evidenzia un andamento discendente coerente con le altre configurazioni. Tuttavia, si nota una certa persistenza del rumore anche nelle fasi avanzate del training, sintomo che il modello, con così tanti parametri liberi, potrebbe aver faticato a convergere verso un minimo stabile con un batch size unitario.
- **Correlazione:** Il grafico della correlazione sul validation set mostra un andamento peculiare. Dopo una fase iniziale di instabilità (con valori che scendono anche sotto lo zero intorno all'epoca 10), la performance inizia a salire decisamente solo dopo l'epoca 20. Sebbene raggiunga picchi promettenti verso la fine (attorno a **0.04**), la traiettoria di apprendimento appare meno fluida e più soggetta a crolli improvvisi rispetto alle configurazioni con Rank inferiore.

La valutazione finale del modello (checkpoint-8680) ha prodotto i seguenti risultati:

- **Risultato Test Set:** 0.0249
- **Baseline Matematica:** 0.0235

Anche con la massima capacità testata (Rank 32), il modello riesce a superare la baseline matematica, ottenendo uno score di **0.0249**. Tuttavia, questo risultato è inferiore sia alla configurazione LoRA-8 (0.0343) che alla LoRA-16 (0.0306).

Ciò conferma l'ipotesi che, per questo specifico dataset e architettura, aumentare eccessivamente il numero di parametri addestrabili (quasi il 20% del modello totale) è controproducente. L'eccessiva capacità, combinata con la scarsità di dati, ha probabilmente portato il modello a memorizzare il rumore piuttosto che apprendere pattern generalizzabili, risultando in prestazioni sul test set marginalmente superiori alla baseline e inferiori a configurazioni più "leggere" ed efficienti.

Configurazione LoRA-32 (Fused Linear Only)

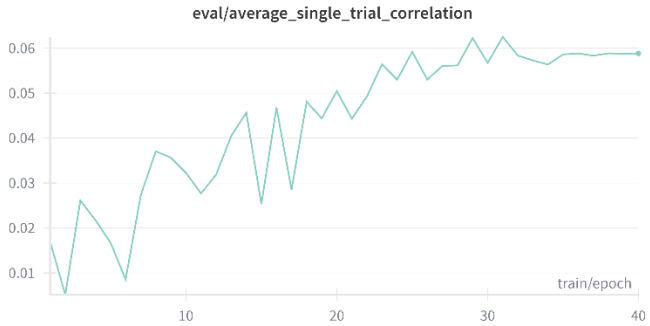
L'ultimo esperimento condotto ha applicato l'adattamento selettivo ai soli moduli **fused_linear** ma con un Rank massimo di 32. Questa configurazione, che coinvolge il 13.7% dei parametri, mirava a verificare se l'aumento della capacità espressiva all'interno dei meccanismi di attenzione potesse compensare la mancanza di adattamento nei layer successivi.

Per questo test, sono stati mantenuti il Learning Rate a e lo scheduler *Cosine*, parametri che si sono dimostrati efficaci nelle configurazioni "Fused Only" precedenti.

Di seguito i dettagli della configurazione:

- **Rank (r):** 32
- **Alpha (alpha):** 64
- **Target Modules:** Fused Linear Only
- **Learning Rate (LR):** 0.001
- **Scheduler:** Cosine
- **Trainable Parameters:** 13.7%

Di seguito vengono riportati i grafici relativi all'andamento del training:



Analisi delle Curve

- **Loss:** La curva di loss mostra una convergenza stabile e progressiva, simile alle altre configurazioni, ma con una varianza apparentemente inferiore nelle epoche finali, suggerendo una buona stabilità del training.
- **Correlazione:** Il grafico della correlazione sul validation set evidenzia un trend di crescita molto positivo. Partendo da valori bassi, la performance migliora costantemente, superando agevolmente la soglia di 0.04 intorno all'epoca 15 e stabilizzandosi su valori prossimi a **0.06**

nella seconda metà del training. Questo andamento suggerisce che il modello è stato in grado di apprendere efficacemente senza incorrere in fenomeni di overfitting precoce.

Risultati sul Test Set La valutazione finale del modello (checkpoint-6727) ha prodotto i seguenti risultati:

- **Risultato Test Set:** 0.0363
- **Baseline Matematica:** 0.0235

Conclusioni parziali Questa configurazione ha prodotto il **miglior risultato assoluto** tra tutti gli esperimenti condotti, con uno score di **0.0363** sul test set (superando la baseline di circa **1.54 volte**).

Questo risultato è particolarmente significativo perché supera sia la configurazione LoRA-8 "All Linear" (0.0343) che la LoRA-32 "All Linear" (0.0249).

Conclusioni: ciò indica che concentrare l'adattamento sui meccanismi di attenzione (*Fused Linear*) con una capacità elevata (Rank 32) è la strategia vincente per questo dataset. Evidentemente, i layer *Feed-Forward* e di output, se adattati con un Rank elevato, tendono a introdurre rumore o overfitting (come visto in LoRA-32 All Linear), mentre i layer di attenzione beneficiano della maggiore espressività per catturare le complesse relazioni spazio-temporali degli stimoli visivi.