Project **"Brain Encoding - ViV1T"**

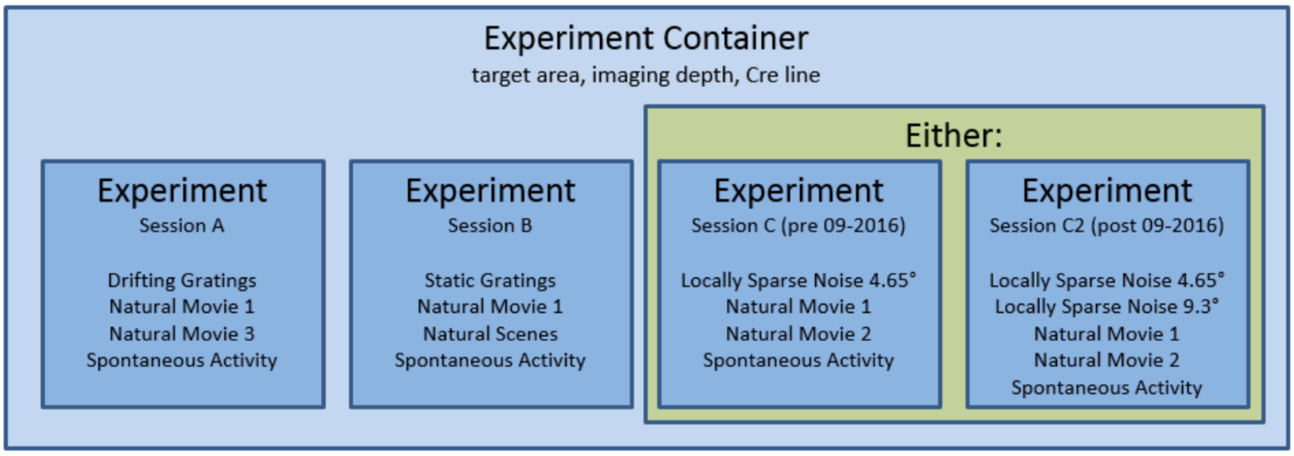
Cognitive Computing and Artificial Intelligence

UniCT - Academic Year 2024/2025
Authors: Giuseppe Leocata, Alberto Provenzano

BRIEF OVERVIEW

After creating our own dataset, we fine-tuned a custom ViV1T model on it using LoRA as the PEFT technique. Finally, we compared the resulting model with a purely mathematical baseline.

ALLEN BRAIN OBSERVATORY DATASET



Through the Allen Institute SDK, we can access the so-called 'containers', which represent three different experiments (or sessions) performed on the same mouse, each involving a different sequence of stimuli. Specifically, a container includes session A, session B, and either session C or C2, depending on the date the experiment was conducted.

Session type 'A' includes the following **stimuli**:

- *Drafting Gratings* → A grating that moves in a direction perpendicular to the orientation of the grating itself. During the experiment, the grating rotates to one of eight possible angles, spaced at 45° intervals, and moves at a speed determined by one of five temporal frequencies, resulting in **40 distinct conditions**. Each condition is presented for 2 seconds, followed by 1 second of gray screen, and repeated 15 times in random order.
- *Natural Movie 1* → A 30-second video clip taken from *Touch of Evil*, a 1958 film directed by Orson Welles.
  - The clip is shown 10 times during the session.

- *Natural Movie 3* → Same concept as the stimulus described above, but here the duration is 120 seconds and its 10 repetitions are non-consecutive.
  Unlike artificial stimuli (such as drifting gratings), natural movies provide a more realistic context for studying how neurons in the visual cortex respond to naturalistic inputs.
- *Spontaneous Activity* → It consists of a period during which the mice are exposed to a screen with uniform mean luminance (gray), without any structured visual stimulation.

Session B, on the other hand, is an experiment in which, in addition to the 'null stimulus' (*Spontaneous Activity*) and a Natural Movie One, natural images (*Natural Scenes*) and static gratings are also presented:

- *Natural Scenes* → the natural images consist of 118 scenes selected from three different databases, each briefly presented for 250 ms, and repeated 50 times in random order.

- *Static Gratings* → these are non-drifting gratings that vary in orientation and spatial frequency (i.e., the spacing between black bars). They are also presented for 250 ms each and repeated 50 times in random order.

Finally, session type C is characterized by a specific type of stimulus: **Locally Sparse Noise**, which consists of black and white spots displayed on a gray background with mean luminance. Each frame contains approximately 11 spots and is presented for 0.25 seconds.
For data collected after September 2016 (type C2), spots twice as large were introduced.

In light of our project goal, we selected a **type A session** as our dataset, from which we extracted only the natural videos.

Focus on the stimuli used in the type A experiment

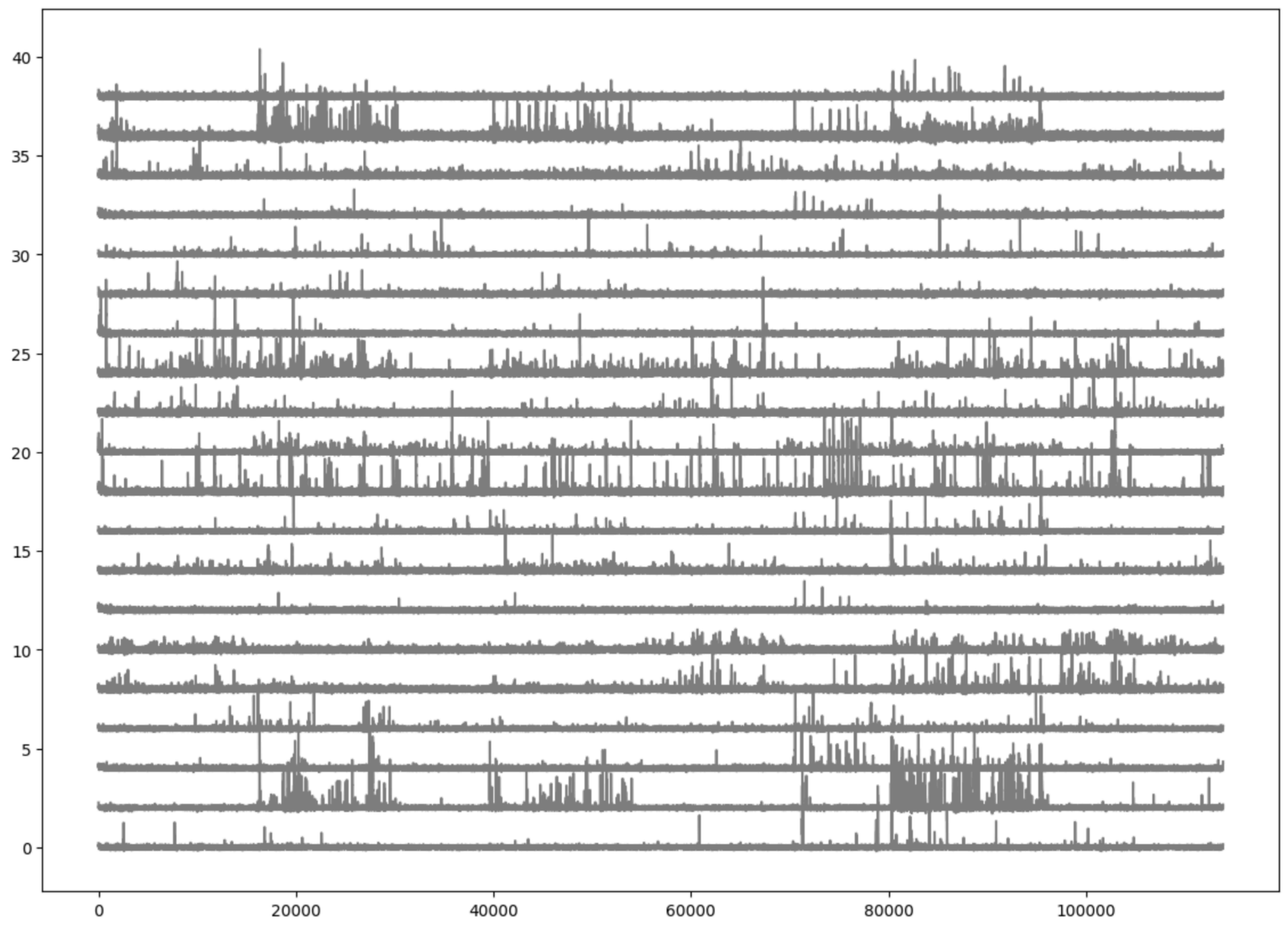
The entire experiment lasts approximately 114,743 frames, and since the **sampling rate** is 30 Hz, this corresponds to about 32 minutes. The **sequence** in which the stimuli were presented to the mouse is shown in the figure on the right. We note that the various trials of Natural Movie 3 are presented at different times (the same applies to the gratings).

```
             stimulus  start     end
0    drifting_gratings    735   18806
1  natural_movie_three  19741   37846
2    natural_movie_one  38751   47810
3    drifting_gratings  48716   66786
4          spontaneous  66936   75866
5  natural_movie_three  75867   93967
6    drifting_gratings  94873  115478
```

Neuronal responses: how they are detected and recorded

The technique used is **two-photon neuronal imaging**, one of the most advanced methods to observe neuronal activity in real time within the brains of living animals. It relies on the physical phenomenon whereby calcium concentration inside a neuron increases upon activation.
Specifically, mice are genetically modified to express a protein called GCaMP6 in a targeted brain region. Chemically, when this protein encounters the influx of calcium—triggered by neuronal activity—it becomes **fluorescent**. The brain area to be recorded is illuminated with low-energy two-photon excitation to **enhance the visibility** of this fluorescence. Through the microscope, changes in fluorescence intensity over time are recorded for identified neurons.

The graph below shows these fluorescence variations for 20 neurons from a random experiment.



DATASET

As previously mentioned, we decided to create the dataset based on a single experiment (a single session) of type A. Specifically, this session is characterized by the following features:

- **Brain area:** primary visual cortex → responsible for the initial processing of visual information.
- **Microscope model:** Nikon A1R.
- **Microscope field of view:** 512 x 512 pixels.
- **Imaging depth:** 275 micrometers → approximately corresponding to layer 4 of the cortex.
- **Introduced gene:** "Rorb" → a specific marker for cortical neurons in layer 4.
- **Experiment date:** February 5, 2016.

Since the custom ViV1T model was originally trained on RGB videos, we decided to filter this session to include only the natural video stimuli (*natural_movie_one* and *natural_movie_three*).
Additionally, we performed a pre-filtering step regarding the experiment selection itself, using the parameters *require_eye_tracking* to select only experiments with eye tracking data available, and *include_failed* (False) to exclude those where eye tracking failed.
This was done to ensure access to information about *pupil size* and *pupil location*, which are essential inputs that allow the model to produce different predictions across trials (without them, the trials would appear identical to the model).
More specifically, pupil size refers to the pupil area measured in square pixels; variations in this measure are correlated with the **mouse's attention level**, which significantly influences neuronal responses. Meanwhile, pupil location indicates the spatial coordinates of the pupil center within the eye tracking camera's field of view, expressed as X and Y coordinates (in pixels). Variations in this measure provide information about **visual exploration patterns**.

The session we identified and filtered has the *ID = 501729039* and contains measurements from *227 neurons*.

Broadly speaking, our goal was to reproduce the dataset from the "Sensorium 2023 Challenge" used by the ViV1T authors to train the model. Consequently, our dataset includes the following inputs:

- *videos*: includes the 10 trials of *natural_movie_one* and *natural_movie_three*.
- *behavior*: a combination of the mouse's running speed and pupil size.
- *pupil_center*: equivalent to the pupil_location from the Allen Institute Observatory.

Data labels (neural responses) were obtained using the `get_corrected_fluorescence_traces()` method of the `session_data` object. Specifically, this method returns corrected fluorescence traces, that is, the amount of intracellular calcium in neurons free from contamination originating from the surrounding neuropil (nervous tissue that surrounds the cell bodies of neurons).
This choice is also consistent with the objective of building a dataset as close as possible to the sensorium; in fact, these traces are non-negative values, unlike dF/F traces (which normalize the signal relative to a baseline and can therefore assume negative values when neural activity drops below the reference level).
From the perspective of the object to be manipulated with python, these measurements consist of a two-dimensional numpy array with dimensions (number_neurons, number_frames) = (227, 140).

Focus on the dataset construction process

We started by retrieving the videos from the session and observed that the natural video of type one is a three-dimensional NumPy array with dimensions (900, 304, 608), while the type three video has dimensions (3600, 304, 608). The first dimension corresponds to the temporal depth, i.e., the number of frames, while the other two represent the height and width of each frame, respectively.

First, we resized the spatial dimensions of the videos to match those expected by the ViV1T model, specifically (36, 64).
Next, since the model was designed to process much shorter videos than ours (140 frames, approximately 4.6 seconds), we chose to **segment**, i.e., split each video into shorter clips.
Because the durations of our videos were not perfect multiples of 4.6 seconds, we discarded 2 seconds (60 frames) from type one and 3.33 seconds (100 frames) from type three, resulting in 6 clips (840 frames) and 25 clips (900 frames), respectively.

To retrieve the information on *pupil_locations*, *pupil_size*, *running_speed*, and *responses* for each clip of the two videos, we proceeded as follows, following the segmentation and considering the 10 repetitions for each full video:

- *Natural_movie_one*: Every 840 frames, we discarded the subsequent 66 frames (6 more than the initially expected 60), since we verified that the total duration of measurements associated with the entire stimulus is 9,060 frames

instead of 9,000. This occurs because, for each stimulus, 1 to 2 seconds of delay are added before and after to capture the complete neuronal response, accounting for the physiological delay in neural conduction.
- *Natural_movie_three*: In this case, the procedure is similar but slightly more complex, since the 10 trials of the full video were presented to the mouse in two separate blocks: 5 in the first and 5 in the second. For the first 5 trials, the total number of frames is 18,105, so we discarded 121 frames after every 3,500-frame segment (100 frames for segmentation and 21 to account for the stimulus recording delay).
  For the remaining 5 trials, the total frame count is 18,100, and in this case we discarded 120 frames between each trial.

Subsequently, we **combined** *pupil_size* and *running_speed (*using the *column_stack()* function from the NumPy library) to obtain what we refer to as "*behavior*".

After selecting the data, we **preprocessed** them using the "Clean Missing Data" technique, which involves removing missing values (NaNs). Specifically, we applied a Data Imputation method that replaces NaNs with the **mean value**.

We then split the data into:
- `train_set`: the first 7 trials of *natural_movie_one* and *natural_movie_three*
- `validation_set`: the 8th trial of *natural_movie_one* and *natural_movie_three*
- `test_set`: the last 2 trials of *natural_movie_one* and *natural_movie_three*

Next, we computed the statistics on the training data in order to perform normalization across all datasets. Specifically, we applied the same data normalization used by the ViV1T authors (as indicated by the parameter `transform_mode=2` in the `args.yaml` file): **min-max normalization** was applied to the *videos*, *behavior*, and *pupil_center*, while **z-score normalization** was applied to the *dF/F* values.

In order to avoid repeating the process each time, we chose to save the datasets in folders with the following hierarchy:

```
train/                      test/                       validation/
└── data/                   └── data/                   └── data/
    ├── videos/                 ├── videos/                 ├── videos/
    ├── behavior/               ├── behavior/               ├── behavior/
    ├── pupil_center/           ├── pupil_center/           ├── pupil_center/
    └── labels/                 └── labels/                 └── labels/
```

Dove, per ogni cartella, i dati sono stati memorizzati come file x.npy dove x indica il numero del dato (i.e. nella cartella videos 0.npy indica la clip 1 del natural_movie_one, 1.npy indica la clip 2 del natural_movie_one ecc.)
Abbiamo, infine, definito la classe `MouseDataset` per la creazione dei dataset effettivi che abbiamo poi utilizzato per il nostro progetto.

In each folder, the data were stored as files named `x.npy`, where *x* indicates the index of the sample (i.e., in the *videos* folder, `0.npy` refers to the first clip of *natural_movie_one*, `1.npy` to the second clip, and so on).

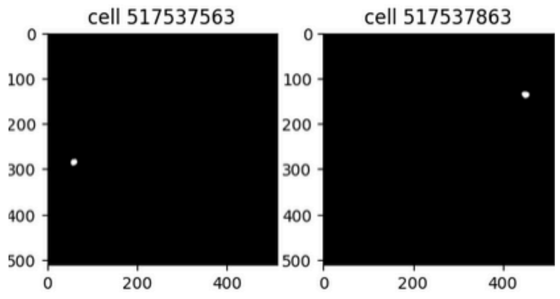
Finally, we defined the `MouseDataset` class to generate the actual datasets used throughout our project.

In the end, our `train_set` consists of **217 samples**, where the first 42 correspond to the first 7 trials of *natural_movie_one*, and the remaining 175 samples correspond to the 7 trials of *natural_movie_three.*
Each sample consists of:
- 1 *clip* with shape (1, 140, 36, 64)
- 1 *behavior* with shape (2, 140)
- 1 *pupil_center* with shape (2, 140)
- 1 *label* with shape (227, 140).

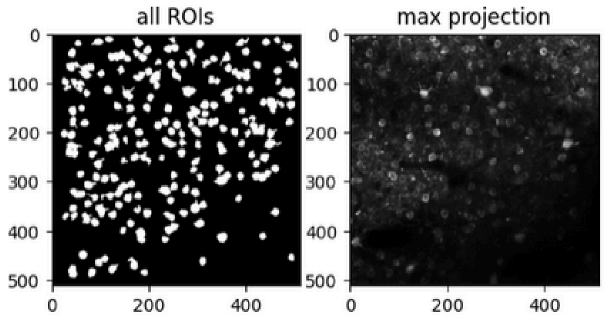
The `validation_set` contains **31 samples** (6 for *natural_movie_one* and 25 for *natural_movie_three*), while the `test_set` contains **62 samples** (12 for *natural_movie_one* and 50 for *natural_movie_three*).

Neuron coordinates

One final element we added to our dataset to fully match the information contained in the dataset used to train the model is the neuron coordinates (relative to the microscope's field of view).
These data are not directly accessible through the Allen Institute SDK, but we derived them from the *ROI masks*: images associated with each neuron that depict its position within the field of view.



Starting from these masks, we obtained the `(x, y)` coordinates by using NumPy's `where()` function, which, given a mask as input, returns the positions along the axes where active pixels are located. We then considered the **midpoint** of these positions as the actual coordinates.



To visualize the overall spatial distribution of neurons in the recorded cortical area, we created a figure by overlaying all 227 masks.

Moreover, using the `get_max_projection()` method, we plotted a mask where, for each pixel, the highest intensity recorded during the experimental session is represented. This mask is mainly used for identifying the neurons that were most active during the experiment.

Finally, we decided to "transform" the coordinates under two aspects:

- The coordinates were negativized, so as to make the center **(0,0)** the **top-right corner**.
- They were each multiplied by a factor of 1.24, so as to have a **620**x**620** scale (instead of 500x500).

Both modifications were made in order to replicate as much as possible the original dataset with which the model was trained, consistently with what has been done so far.

BASELINE

For the reference baseline in our project, we decided to use one that is model-independent (a mathematical baseline). In this case, we only used the `train_set` and `test_set` and defined the function `retrieve_dff_mean_among_trials_for_a_clip`, which:

- It takes as **input**: the video type (one or three), the train_set, and the clip number (from 1 to 6 for *one* and from 1 to 25 for *three*).
- It **returns** a 2D NumPy array with shape (227, 140), where, for each neuron, the value corresponds to the average intensity across the 7 trials related to the selected clip.

We identified two possible approaches.

Sample-based approach

As a first "test," we retrieved:

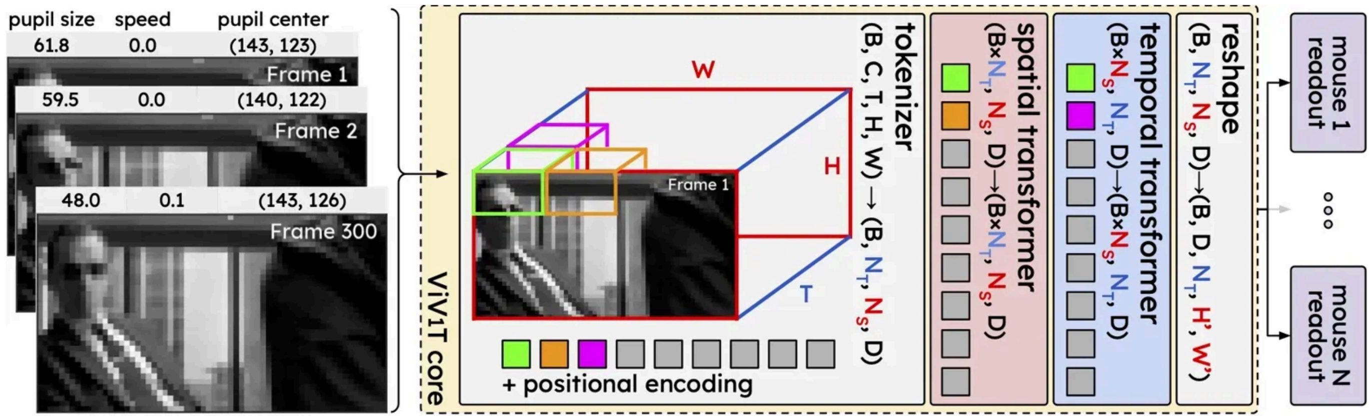
- From the `test_set` side, the **labels** of trials 8 and 9 of the same clip.
- From the `train_set` side, the mean response value of each neuron across the 7 trials, for each frame.
  - This is essentially the NumPy array output by the function discussed above.

Finally, we computed the **Pearson correlation** between this NumPy array and the label of trial 8, and subsequently with that of trial 9, for the "natural_movie_one" video.
We applied the same procedure to the "natural_movie_three" video, thus obtaining **four overall values** to compare with our model. Specifically, in our empirical test we selected clip number 1 for both the "one" and "three" types.

Approach based on the average of the single-trial correlations across all clips in the dataset

After some consideration, we concluded that calculating the baseline across all clips in the dataset is a more appropriate approach, both because it provides more meaningful information compared to the sample-based approach and because the model training follows this metric. In fact, the `compute_metrics` method, called by the HuggingFace Trainer, was implemented precisely to return the mean of the single trial correlations across all `test_set` samples.
Consequently, by evaluating (`evaluate()`) the model on the `test_set`, we obtain a value that can be directly compared with this baseline. Specifically, we first calculated the correlation between the **train_set responses** and the `test_set` clips corresponding to trial 8, and then the correlation between the `train_set` responses and the `test_set` clips corresponding to trial 9.
Finally, to obtain a single correlation value, we averaged the two correlations, thus deriving our baseline, which is **0.022**.

MODEL



The image shown above is the one used by the model's authors to illustrate the **inference process**.

For the video embedding, the authors used the **Tubelet Embeddings** technique — three-dimensional patches that extend the concept of two-dimensional patches along the temporal dimension to preserve motion patterns occurring across consecutive frames.

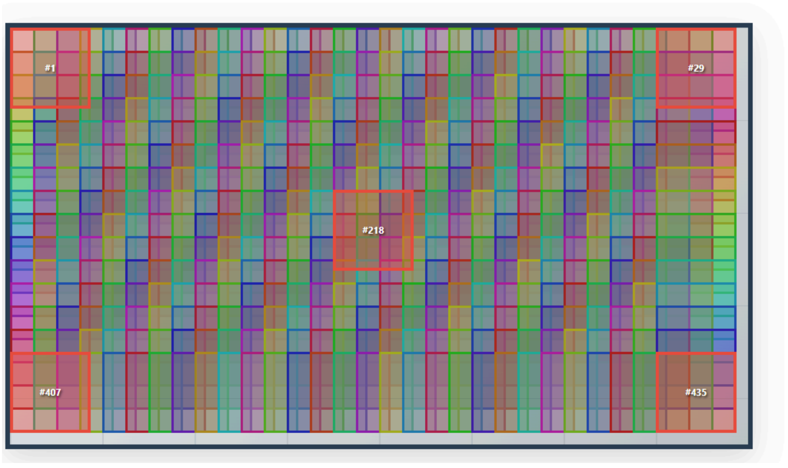
More specifically, the component responsible for this is the "Tokenizer," which takes as input a tensor with shape (B, C, T, H, W) and, through the operation called "Unfold3d," extracts 3D patches by applying three successive operations:

- First unfold along the temporal dimension (T)
- Second unfold along the spatial height dimension (H)
- Third unfold along the spatial width dimension (W)

We have `spatial_patch_size = 7` and `spatial_patch_stride = 2`, which means that each spatial patch has square dimensions of 7x7 pixels and that each patch is spaced 2 pixels apart from the next. In other words, considering a single frame, the number of patches is given by:

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

Therefore, 435 spatial patches are applied per frame.



Now, let's also consider the temporal axis, starting from the conclusion: these 435 patches are "taken" 116 times due to the other two parameters, `core_temporal_patch_size = 25` and `core_temporal_patch_stride = 1`.
This means that each spatial patch on a given frame has a "depth" of 25, and starting from the same spatial coordinates, other patches develop along the temporal axis with a stride of 1.
In simple terms, the i-th patch along the temporal axis starts at a position that is 1 unit away from the starting position of the (i-1)-th patch.

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

It follows that we have 116 temporal patches; this number should be understood as "the number of shifts that a window of width 25 can make within a total length of 140, moving one position at a time."
Indeed, in the formula, we first calculate the difference between the total length and the window size, then divide this quantity by the step or stride, effectively obtaining the number of possible shifts; finally, we add 1 to include the initial position itself.

At the end of the Unfold3d operation, we obtain an output shape where the last dimension corresponds to the size of a single flattened tubelet, which in this case is 6125 (calculated as the product of C, pt, ph, and pw, i.e., 5 × 25 × 7 × 7).
Then, there is an **embedding layer** that transforms each tubelet from its raw representation (6125 dimensions) into a compact embedding representation (112 dimensions).

As a result, we obtain an output with shape (B, Nt, Ns, D) where:

- **B** is the batch size
- **Nt** is the number of temporal patches (116)
- **Ns** is the number of spatial patches per frame (435)
- **D** is the dimension of the compact embedding for each tubelet (112).

The ViViT model implemented by the authors is the **factorized encoder**, which therefore consists of a spatial transformer followed by a temporal transformer.

Spatial Transformer

Its main function is to process the spatial relationships within each frame of the video.
It processes the 435 patches independently at each temporal index (0–115).
Specifically, it receives an input shaped (B×Nt, Ns, D) and, for each temporal position (Nt), computes attention among the Ns patches belonging to the same frame (selected by the current Nt index).

The output of this component still consists of 435 patches for all temporal indices (116), but each patch now contains spatial correlation information with the others.

Temporal Transformer

The input to this component is the output of the spatial transformer but reshaped; specifically, it receives an input shaped (B×Ns, Nt, D). Here, each spatial position is treated as an independent sample in the batch, allowing the attention mechanism to capture temporal dependencies along the sequence of 116 frames for each specific location.

Reshape

Receives as input a tensor with shape (B, Nt, Ns, D) and produces an output tensor with shape (B, D, Nt, H', W'), where H'=15 and W'=29. From the spatial dimension point of view, the **two-dimensional information** is reintroduced (while preserving the information contained in the data) by modifying the **representational structure**.
Specifically, 15 and 29 represent the number of patches along the height and width, respectively (15×29 = 435).
This transformation converts the post-transformer **sequential** representation (where the 435 patches are treated as a linear sequence) into a **spatial** representation necessary for the subsequent modules, which perform operations such as computing distances between patches, essential for correctly mapping visual features to neurons.

MLP Shifter & Readout

ViV1T includes one MLPShifter and one Readout module for each mouse, since these components are customized to handle a specific number of neurons (which can vary between different mice). Because our dataset consists of a single experiment conducted on one mouse, our model has only one Shifter and one Readout.

The Shifter acts as a **dynamic correction mechanism** for the mouse's eye movements during video presentation: eye movements introduce systematic spatial shifts in the perceived visual field, which must be compensated for to achieve more accurate predictions. More specifically, when the mouse moves its eyes, every element in the video is perceived by the retinal neurons at a shifted position compared to where it should be. For example, if an object appears at coordinates (10, 15) in the video but the mouse has shifted its gaze 2 pixels to the right and 1 pixel up, the neurons will perceive that object as if it were at coordinates (12, 16). This creates a systematic misalignment between where we think the mouse is looking and where it is actually looking.

From a practical standpoint, it **predicts** the eye shifts based on behavioral data (pupil position, speed, dilation), producing (Δx, Δy) for each temporal frame—hence the output shape of (116, 2).

The Readout component, instead, takes as input tensors of shape (1, 112, 116, 15, 29) coming from the Rearrange module and produces an output of shape (1, 227, 116), which represents the **raw prediction**.
It operates on each neuron by computing, through its coordinates, where it should "look" within the 15×29 spatial grid. Clearly, this mapping is learned during training and is supported by integrating behavioral information: if each neuron has a preferred "observation" **window** on the 15×29 grid (determined by its anatomical coordinates in the brain), when the mouse moves its eyes in one direction, the model compensates for this movement by shifting the observation windows in the opposite direction.
Similarly, if the mouse changes its running speed, the intensity of the Gaussian window is modulated, and if pupil dilation varies, the shape of the window also changes.
The concept of the window is practically implemented using Gaussian pooling functions: for each neuron, the model creates a Gaussian "bell" centered on its preferred position within the 15×29 grid. This determines the weight assigned to each cell in the grid when calculating the neuron's activation.

Clearly, this process happens for every temporal frame.

The output of the Readout should therefore be interpreted as the application of each neuron's window to the features present in the grid; in other words, all the values in the grid are taken, multiplied by the Gaussian weights, and then summed up: this final value represents how active that neuron should be based on what it "sees" through its window.

Model Adaptation

In order to use the ViV1T, we had to adapt it to our dataset and to do this we modified the "args" to pass to the class constructor. Specifically, we proceeded with the removal of 9 "*mouse_ids*" (leaving only one, the one with id: "A") and adapted the "output shapes" accordingly (this is where the number of neurons considered for each id is specified). This choice allowed us to obtain a model that was lighter in terms of loaded parameters (3M vs 12M) and that makes

predictions based on the brain configuration of the single mouse we examined. Therefore, the head of our model consists of a Shifter and a Readout.
"The adaptation process consisted of the following steps:

- *Loading of the pre-trained model*: we first instantiated the model with the configuration parameters and neuron coordinates.
- *Loading of the ViV1T weights*: we leveraged the pre-trained weights of the ViV1T to perform transfer learning, however since the head of our model is different from that of the ViV1T (because we consider a different mouse), from the checkpoint we recovered exclusively the weights of the ViV1T-core, thus leaving MLP Shifter and Readout with weights from scratch.
- Freezing of all model parameters except the head.

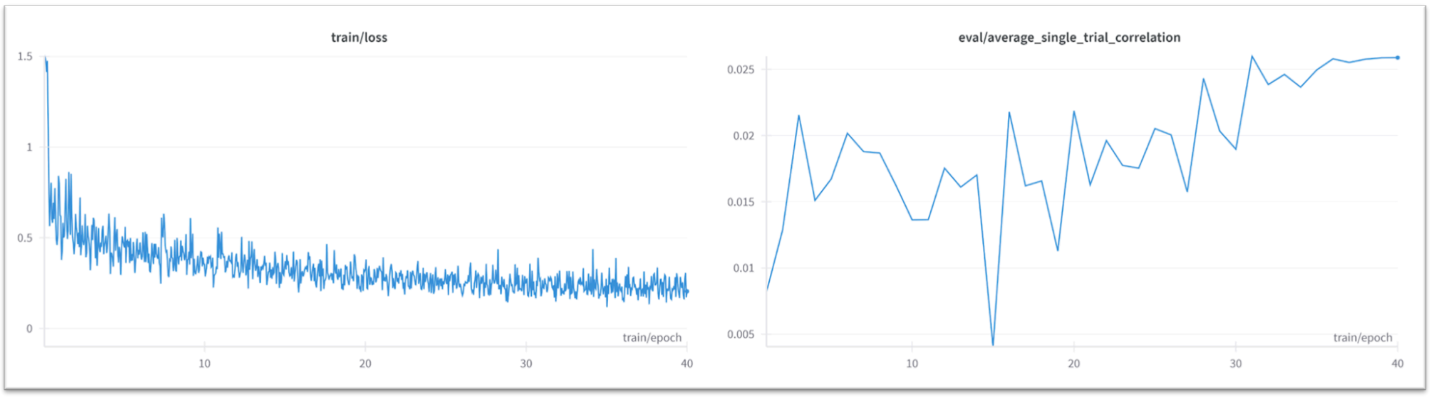
As a metric for model selection of all trainings we used the average of all *single_trial_correlation* on the `test_set`. Furthermore, for the training of the model we used the **MSE Loss** as loss function.

TRAINING

MLP Shifter & Readout

Initially we trained only MLP Shifter and Readout and, since we are dealing with videos and the GPU resources at our disposal are limited, we had to make several compromises: in addition to the video segmentation adopted in the dataset construction phase, we used a *batch_size* = 1, the only value that allowed us to stay within the limits (16.208 GB, P100 GPU on Kaggle).

After performing several trainings (varying the learning rate, the number of epochs and the scheduler) we report below the graphs associated with the training that gave the best score in **test**:



In particular, for this experiment an lr=0.001 and a cosine scheduler were used and the training was conducted for 40 epochs, obtaining, on test, an average single trial correlation value equal to **0.0178**.

The choice of such a high number of epochs is due to 2 reasons: first we are not doing head fine-tuning but we are training the head from scratch; secondly, using a number of epochs equal to 20 (and with all other hyperparameters unchanged) we obtained a decidedly lower score (0.002).

However, this result (**0.0178**) is worse than the baseline (**0.02353**) indicating that in this case the model explains less than the data themselves, and its current parameters are therefore to be considered unsatisfactory.

PEFT (LoRA) training

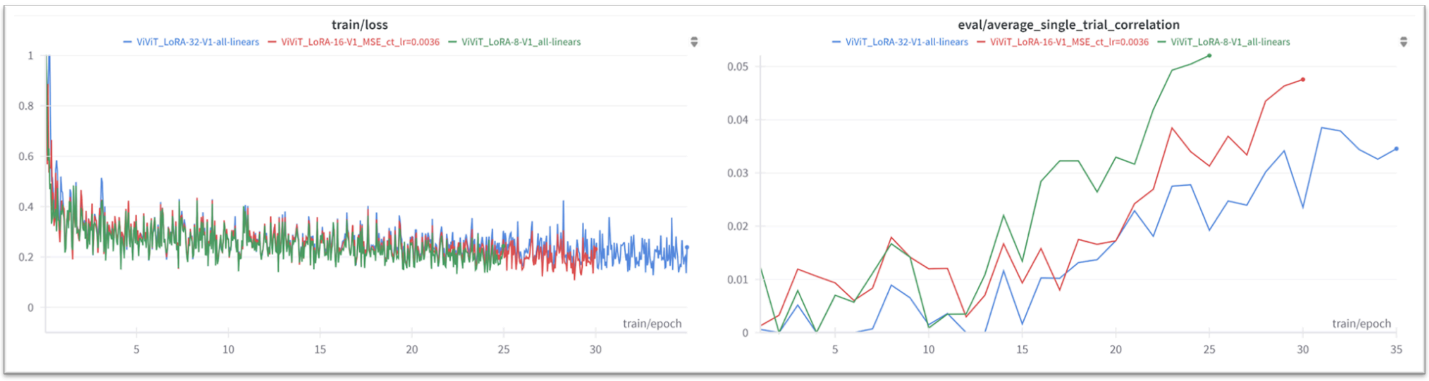
To improve the performance of our model we therefore opted for a PEFT adapter based technique and, specifically, for LoRA. First, we identified what we consider to be the most appropriate linear modules:

- **fused_linear:** particularly interesting because it represents a computational optimization of the transformer architecture: instead of having three separate linear operations for query, key and value (plus an additional projection for the feed-forward network), it computes all four transformations simultaneously. This fusion also creates a **single control point** where LoRA can simultaneously influence all attention and processing mechanisms.
- **attn_out:** final aggregated projection of the multi-head attention output. It determines how information processed by the different attention heads is combined and reintegrated into the main flow of the model.
- **ff_out.2**: corresponds to the final linear layer of the feed-forward network. It is the module responsible for the final transformation that brings the processed representations back to the original dimensional space.

Initially, we tried to perform training including all three modules simultaneously and we varied the **ranks** of matrices A and B (8, 16 and 32) while keeping the **scaling** (alpha/r) constant, thus obtaining the following configurations:

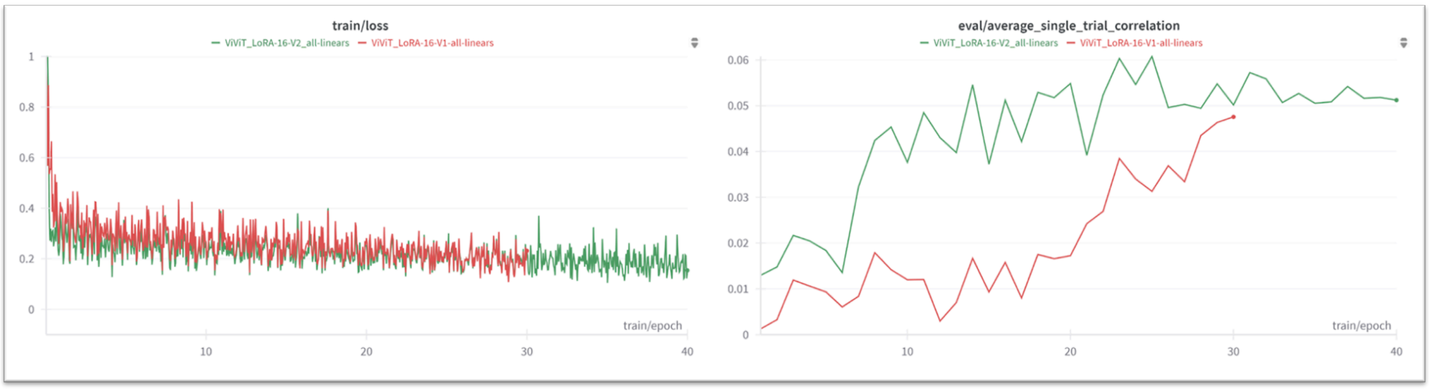
1. r= 8 and lora_alpha = 16
2. r= 16 and lora_alpha = 32
3. r= 32 and lora_alpha = 64

In particular, we started by considering the same learning rate for all three experiments but we differentiated the number of epochs (25 for LoRA-8, 30 for LoRA-16 and 35 for LoRA-32), we show the comparative graphs below:



With these tests we improved the score obtained with only training the MLP Shifter and Readout obtaining, in fact, a score (in test) of **0.033** for LoRA-8, **0.037** for LoRA-16 and **0.038** for LoRA-32. In all 3 cases the baseline (0.023) is exceeded thus demonstrating that the model with LoRA is explaining not only the average of the data but also something more.

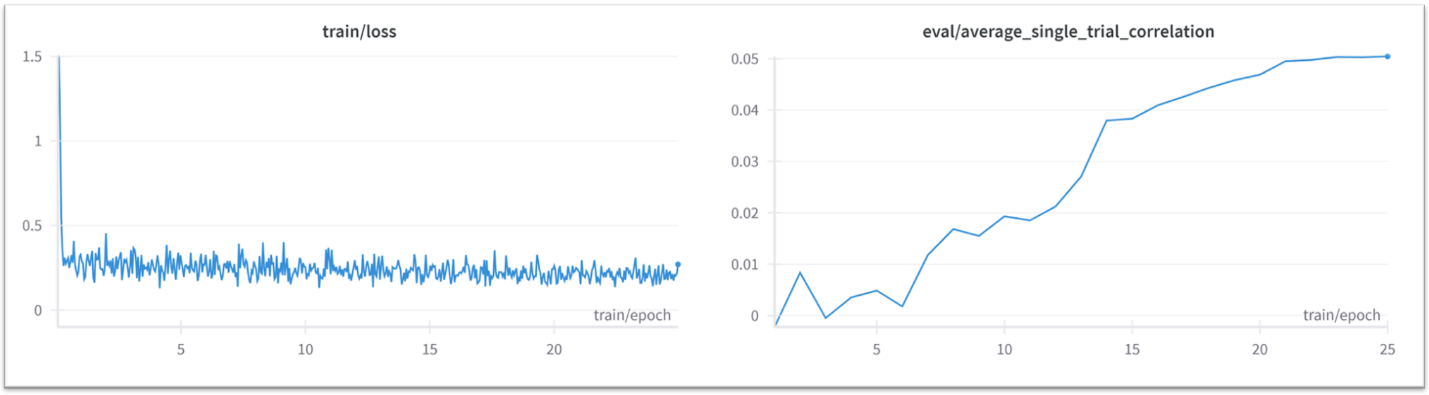
Still leaving all three identified modules as target modules, we made several other tests to try to improve the performance of the models corresponding to the different LoRA configurations, however managing to improve only slightly the score (regarding LoRA-16). In particular, the improvement was obtained using lr=0.001 (lower), obtaining an average single trial correlation value equal to **0.039**.



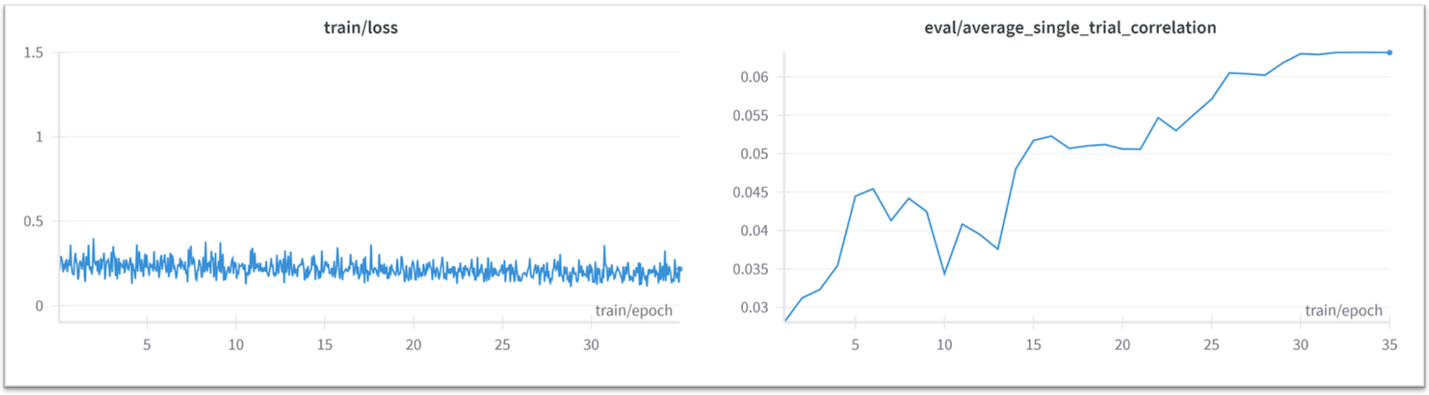
Observing the graph above relating to the correlation on single trials in validation, we can reach the conclusion that a lower learning rate leads to the best value of the previous training in ⅓ of the epochs, a symptom that a more conservative approach in updating the parameters allows the model to reach a more stable convergence and maintain consistent performance for a more extended period during training.

Subsequently, we performed other tests by varying the number of layers where to apply LoRA: we thought it would be more reasonable to apply LoRA **only in the *fused-linear* layers** since they are the ones that involve the Q, K and V matrices and should therefore be sufficient for the purpose of achieving good performance.

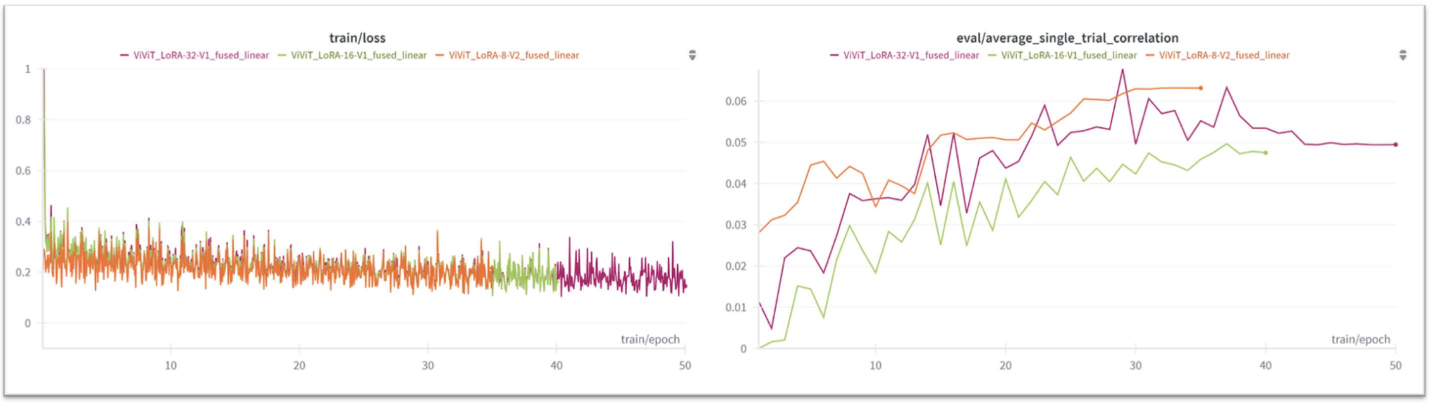
A first experiment was conducted with LoRA-8, in which we used **lr=0.001** with a **cosine scheduler**:



At the end of training a score of **0.035** was obtained and, having found that at epoch 25 the average single trial correlation was still rising, we repeated the training for another 35 epochs starting from those parameters:



In this way, we obtained a score of **0.042**, almost double compared to the baseline (0.023). Given the success with using LoRA only in the *fused linear* layers we tried to apply this strategy also for the LoRA configurations with r=16 and with r=32. Below, the graph showing the comparison of all three different configurations:

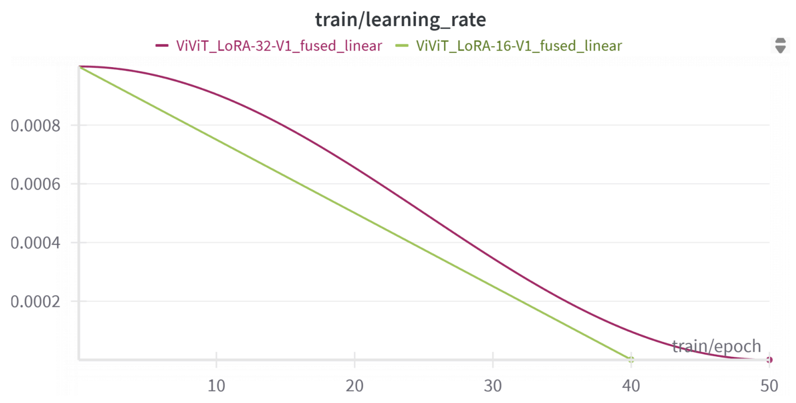


Regarding LoRA-16 we obtained a score of **0.029**, while for LoRA-32 one of **0.033**.

In our opinion, LoRA-8 reported a better result because as the rank increases the number of **trainable parameters** increases and this can lead to greater overfitting, especially in our case where the dataset is small.

However, it should be emphasized that the highest score value obtained (0.042) is the result of the continuation of a first training (which had led to a score of 0.035, still better compared to LoRA-16 and LoRA-32) and, consequently, we can see it as a result due to a single training of 60 epochs where, starting from the 25th (beginning of the subsequent training) the cosine scheduler started again from the maximum value of lr (0.001).

Consequently, if we made a comparison with the learning rates of LoRA-16 and LoRA-32 at epoch 25, we would have that the one of LoRA-8 starts from the maximum while that of LoRA-16 and LoRA-32 will have decreased (respectively to 0.0004 and 0.0003).



Probably, a prolonged training approach or with multiple phases could be beneficial also for configurations with higher rank (provided that adequate regularization strategies are implemented to prevent performance deterioration).

FINAL COMPARISON

RANK	ALPHA	LAYERS	LR	LR SCHEDULER	TRAINABLE%	SCORE
8	16	All	0.0036	Linear	5.56	0.033
8	16	Fused	0.001	Cosine	**3.8**	**0.042**
16	32	All	0.001	Linear	10.53	0.039
16	32	Fused	0.001	Cosine	7.35	0.029
32	64	All	0.0036	Linear	19.1	0.038
32	64	Fused	0.001	Cosine	13.7	0.033

As we can note, the best result was obtained with the lowest percentage of trainable parameters.

CONCLUSIONS

Given the oscillating trend of the various training losses (which is an indicator of non-optimal training), we tried many other different (lower) learning rates but this did not lead to improvements either from the point of view of the curves or from that of the results.
Consequently, we had no choice but to proclaim the unit batch size as the main cause of the oscillation of the train loss. In fact, with batch size=1, the gradient estimated on a single sample is very noisy and can point in very different directions at each step. Therefore, we tried to "simulate" an effective batch size of 7 (so as to have 31 batches for training, which is the same number of batches we have in the validation_set) by setting the *gradient_accumulation_steps* parameter to 7.
Acting on this parameter, the model, during training, will always load 1 single sample at a time on the GPU but the **weight update** does not occur for each sample, but rather after every 7; consequently, the gradients are accumulated sample by sample and only every 7 samples is the `optimizer.step()` instruction executed, which performs the actual parameter update.
Unfortunately, even in this case, although the trends appeared even more dampened, the test results still turned out to be worse than those shown just now, which is why we decided to return to the unit batch size.

We also performed many other tests by changing, for example, **lora_alpha** (to modify the impact of LoRA on the model), fearing that a scaling of 2 (r/lora_alpha) could be too aggressive (obscuring the knowledge of the pre-trained model), but these tests also led us to low scores.

We have therefore reached the conclusion that this score is due to the fact that our dataset, despite having proved to be compatible with the model and equipped with the information necessary for its full functioning, is **too small** (only 1 experiment from session A). In the case of more in-depth research, the dataset could therefore have been extended by searching among the various experiments for those done on the same mouse and making an intersection of the data filtering for the neurons (which have unique ids) in common.

Furthermore, the application of a correct data splitting strategy between `train_set` and `validation_set` (e.g.: k-fold) would certainly have helped.