ARTIFICIAL INTELLIGENCE AND COGNITIVE COMPUTING

V1TA

Train a V1T-like architecture to predict neural activations using Allen Brain Observatory Dataset

TEAM: Fernando Riccioli, Nunzio Fornitto, Angelo Frasca

1000069299, 1000002901, 1000067615

Cognitive Computing and Artificial Intelligence Course - University of Catania, Italy

VITA

OUTLINE

Introduction

Understanding the Dataset

Adapting Allen to V1T

V1T Architecture

Training

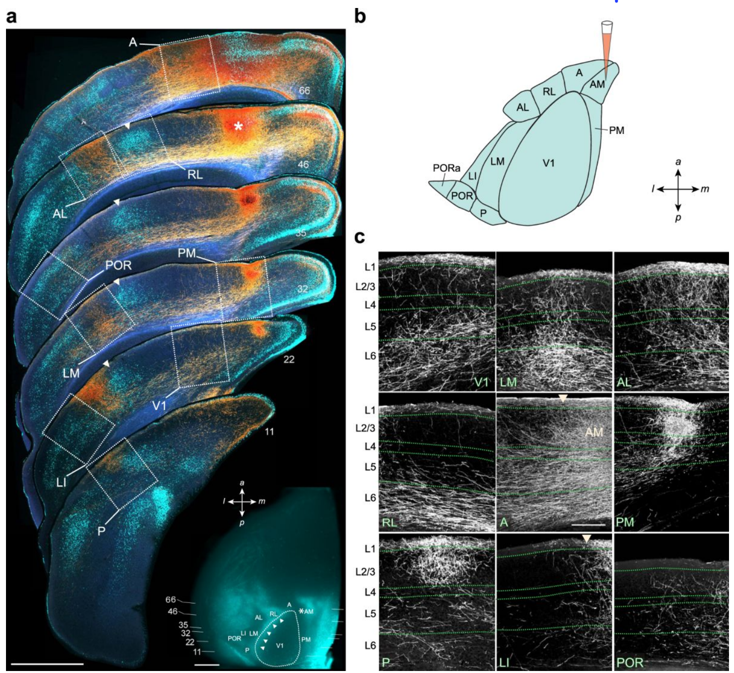
Conclusions

V1T Explained

A Vision Transformer architecture that predicts neural activation in mice.

The name comes from the area of the brain called Primary Visual Cortex (V1)

This is the area which is subject to our study.



Goal of the Project

V1T is designed to work with Sensorium Dataset or Franke 22 Dataset

Our goal is to adapt the Allen Brain Observatory Dataset to make it work with V1T, understand it and train the model.

Tools and Resources

V1T: large-scale mouse V1 response prediction using a Vision Transformer

Bryan M. Li[1] bryan.li@ed.ac.uk
Isabel M. Cornacchia[1] isabel.cornacchia@ed.ac.uk
Nathalie L. Rochefort[2,3] n.rochefort@ed.ac.uk
Arno Onken[1] aonken@ed.ac.uk

[1] *School of Informatics, University of Edinburgh*
[2] *Centre for Discovery Brain Sciences, University of Edinburgh*
[3] *Simons Initiative for the Developing Brain, University of Edinburgh*

Reviewed on OpenReview: https://openreview.net/forum?id=qHZs2p4ZD4

Abstract

Accurate predictive models of the visual cortex neural response to natural visual stimuli remain a challenge in computational neuroscience. In this work, we introduce V1T, a novel Vision Transformer based architecture that learns a shared visual and behavioral representation across animals. We evaluate our model on two large datasets recorded from mouse primary visual cortex and outperform previous convolution-based models by more than 12.7% in prediction performance. Moreover, we show that the self-attention weights learned by the Transformer correlate with the population receptive fields. Our model thus sets a new benchmark for neural response prediction and can be used jointly with behavioral and neural recordings to reveal meaningful characteristic features of the visual cortex. Code available at github.com/bryanlimy/V1T.

Original V1T Paper (Bryan M. Li)

V1T source code: [bryanlimy/V1T: Code for "V1T: Large-scale mouse V1 response prediction using a Vision Transformer"]

V1T paper: [V1T: large-scale mouse V1 response prediction using a Vision Transformer | OpenReview]

Google Colab Pro: https://colab.research.google.com/

Allen Dataset Explanation & Examples: [AllenInstitute/brain_observatory_examples | Github]

VITA

UNDERSTANDING THE DATASET

Allen Brain Observatory

Frame 81 from Natural Scene Table

What happens during a Session

A mouse is shown a sequence of images or videos. These can either be natural or artificial (called gratings).

Each image is presented for **250 ms** before switching to another.

In the session each image is presented 50 times at random intervals.

All images and videos are in grayscale.

VITA

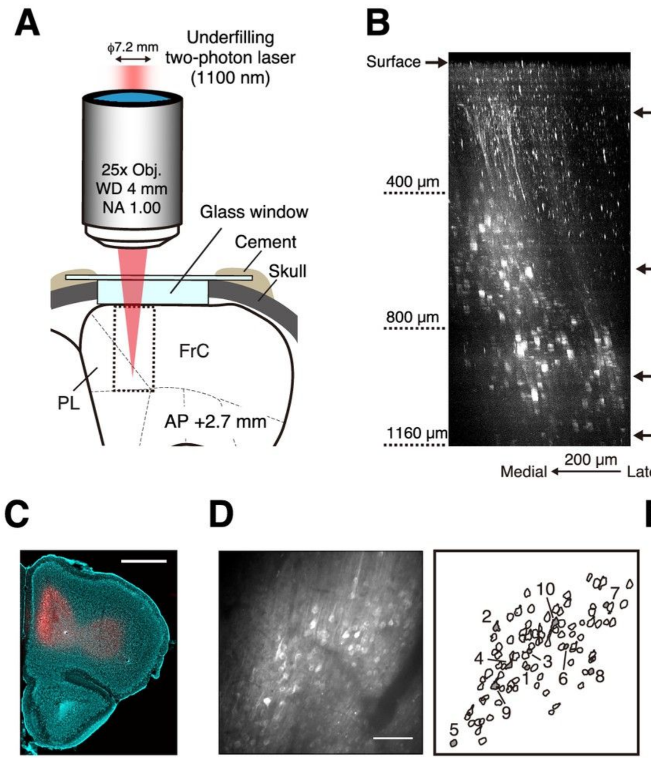
Neural Activation & Behavior

Neural activation is measured by calcium imaging: calcium protein enters the neuron when it's active and makes it fluorescent.

A microscope (2 photon) measures the variation of fluorescence. This measure is called **ΔF/F**.

But neural activation depends on the current state of the brain as well.

This is why to have better performance during training we need behavioral information such as **running speed** or pupil dilation.



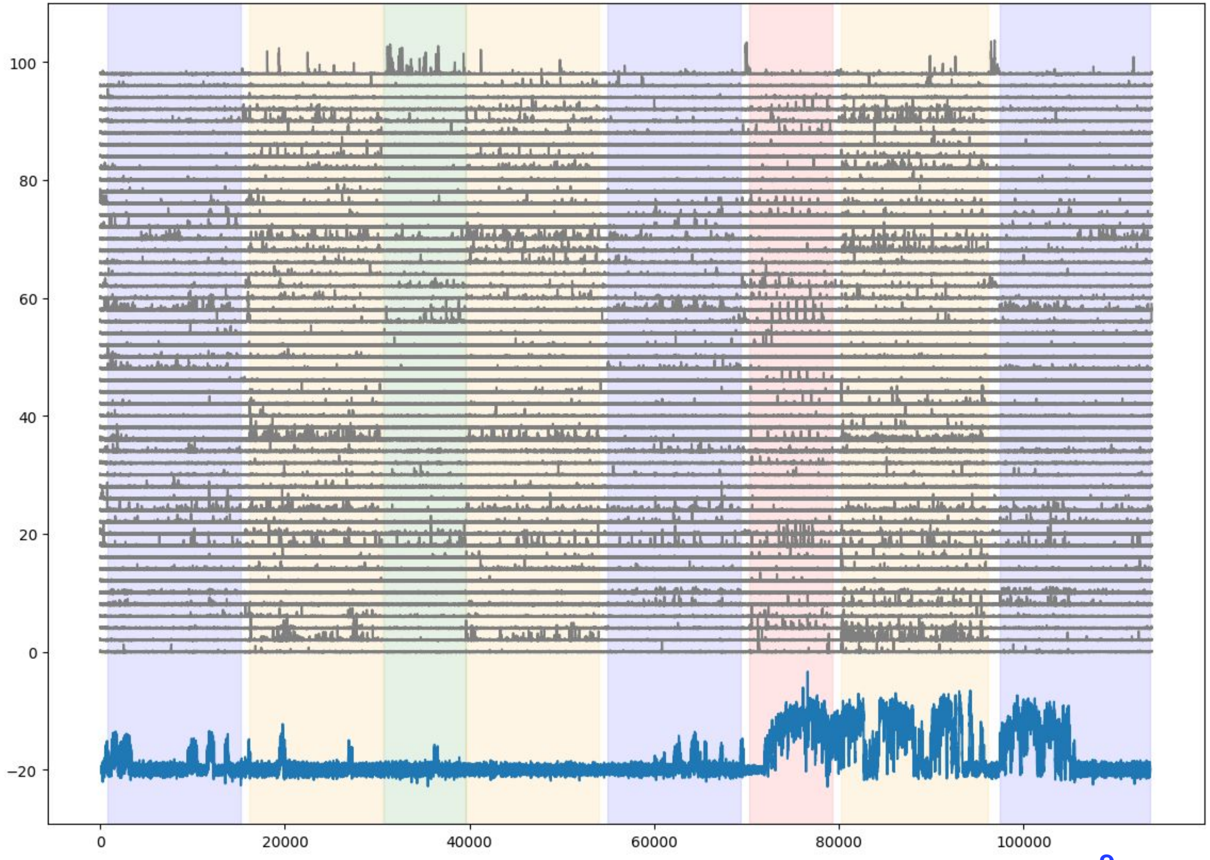
Data Plot

This is a partial plot of the data

X-Axis: frames
Y-Axis: ΔF/F and running speed (cm/s)

Total Neurons: 174
Total Frames: 11388

Duration in minutes: 63.3

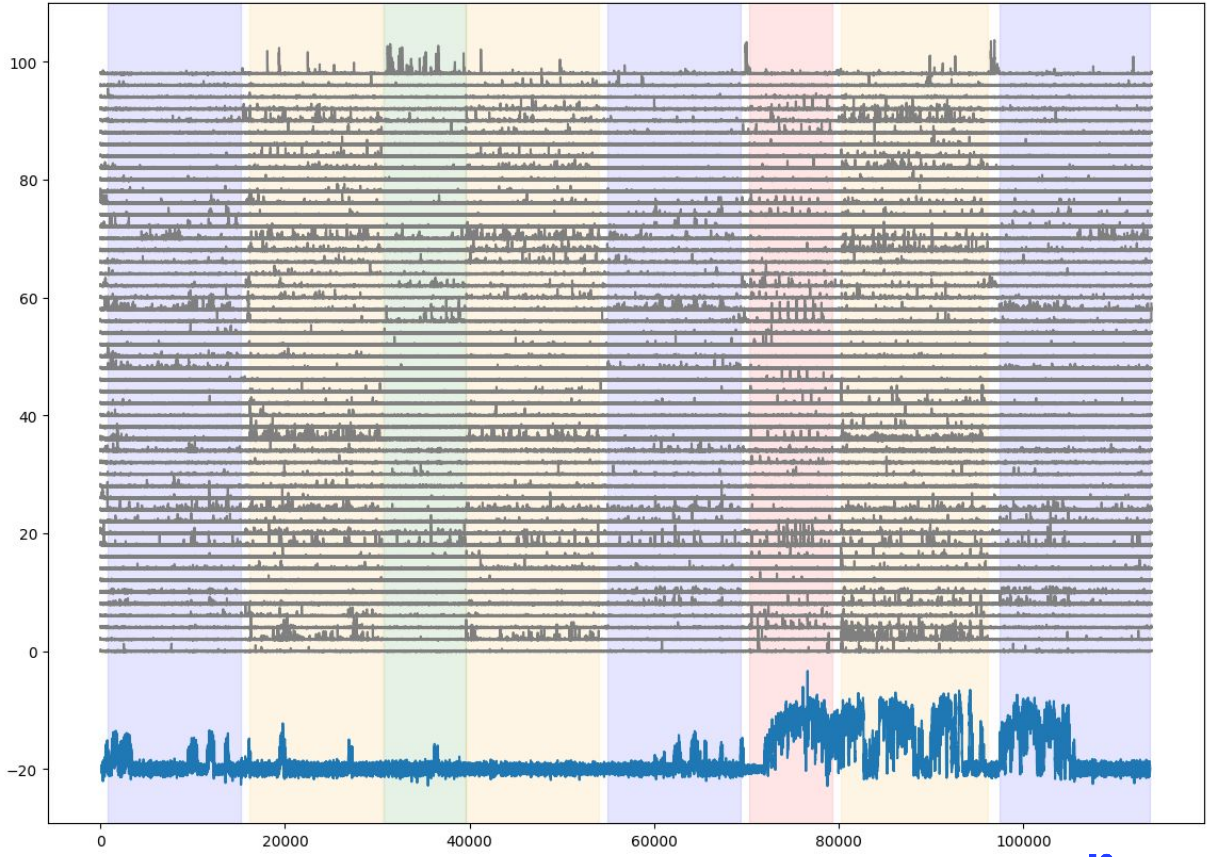


Data Plot

X-Axis: frames
Y-Axis: ΔF/F and running speed (cm/s)

Different colours are different "stimulus epochs".

Yellow are natural images and blue are static gratings.



ADAPTING ALLEN TO VIT

Data & Metadata Processing

AllenSDK

Allen Brain Observatory dataset is available using AllenSDK

```
!pip install allensdk
```

Within the code itself we can choose which sessions to use

```
session_id = 501559087
```

And get the data we need (**ΔF/F, running speed, images**)

```
ts, dff = data_set.get_dff_traces()
dxcm, tsd = data_set.get_running_speed()
stim_table = data_set.get_stimulus_table("natural_scenes")
```

+

Inizializzazione

```
[ ] # Installazione SDK per gestire mediante API il Dataset Allen Institute
    !pip install allensdk
```

```
[ ] !pip uninstall tensorboard -y
    !pip install tensorboard==2.10.0
    !pip show tensorboard
```

```
[ ] # Codice per ottenere un esperimento/sessione specifico dal Dataset di Allen Institute
    # per mezzo della API get_ophys_experiment_data
    import matplotlib.pyplot    as plt
    from allensdk.core.brain_observatory_cache import BrainObservatoryCache

    manifest_file = r'/brain_observatory/manifest.json'
    boc = BrainObservatoryCache()
    session_id = 501559087
    data_set = boc.get_ophys_experiment_data(ophys_experiment_id=session_id)
```

```
[ ] # Dal Dataset Allen Institute estraiamo : 1) i segnali dF/F (risposte neuronali)
    # e relativi Timestamp, 2) velocità di corsa istantanee del topo dell'esperimento
    #(cm/s), 3) i periodi in cui erano attivi gli stimoli visivi,  4) la tabella delle
    # immagini presentate con i relativi Timestamp e i Frame di inizio e fine, 5) gli
    #eventi neuronali(in pratica i dati grezzi dai quali si ottengono i segnali dF/F)
    ts, dff = data_set.get_dff_traces()
    dxcm, tsd = data_set.get_running_speed()
    stim_epoch = data_set.get_stimulus_epoch_table()
    stim_table = data_set.get_stimulus_table("natural_scenes")
    events = boc.get_ophys_experiment_events(ophys_experiment_id=session_id)
```

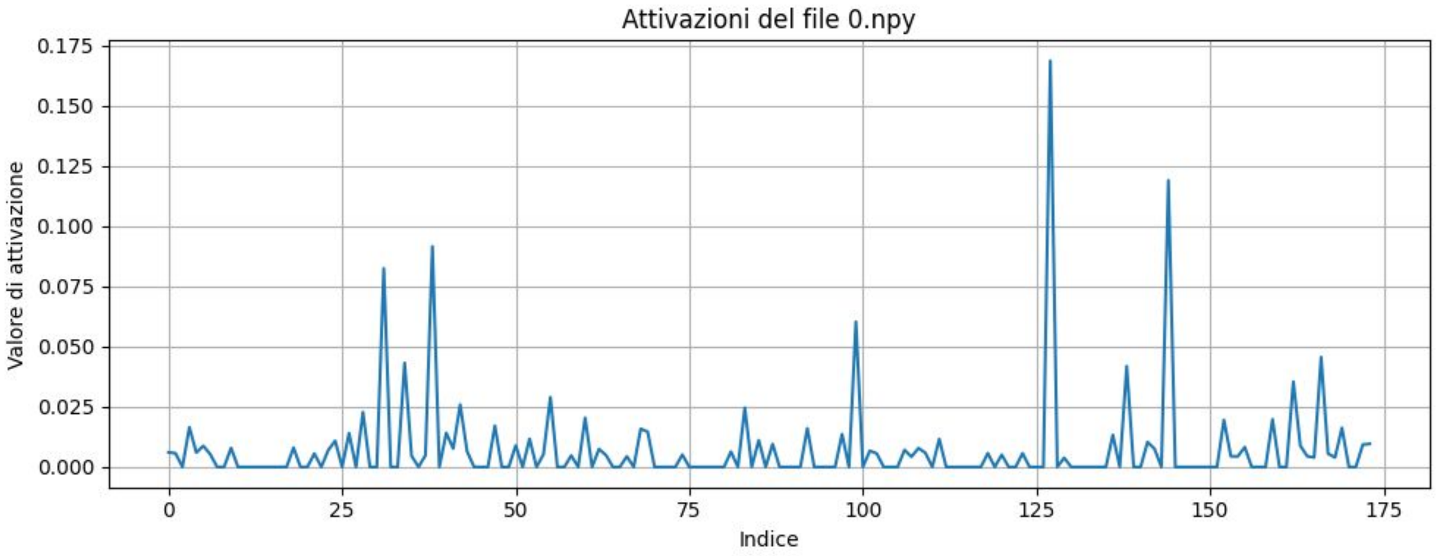
From notebook.ipynb

Processing Stimuli

Neural activations are saved as a single value taken every ~33ms.

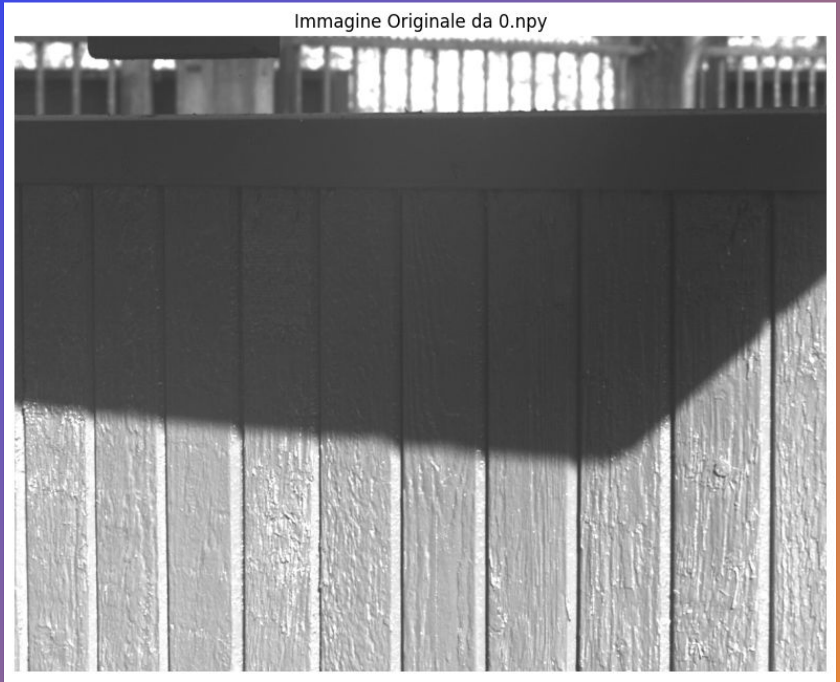
To use them in V1T we **average over 500ms**, similar to how it's done in V1T original dataset.

The results are saved in .npy files, each of which contains 174 values. One for each neuron



VITA

Extracting Stimulus Images



We chose to use only **natural scenes**.

These are found in "`natural_scene_template`"

After they are loaded, they are processed and saved in .npy format.

The number of unique images is 118, each with a resolution of 918×1174.

The total number of `.npy` files for the images is 5950 (one for each trial).

VITA

Grey Images

During the experiment the mouse was shown a grey screen between some images.

These are not available in the dataset, so as a bonus we created them **from scratch**.

These are a total of 50 images shown a different times during the experiment.

These are created using a grey value of `grey_value = 128`

VITA

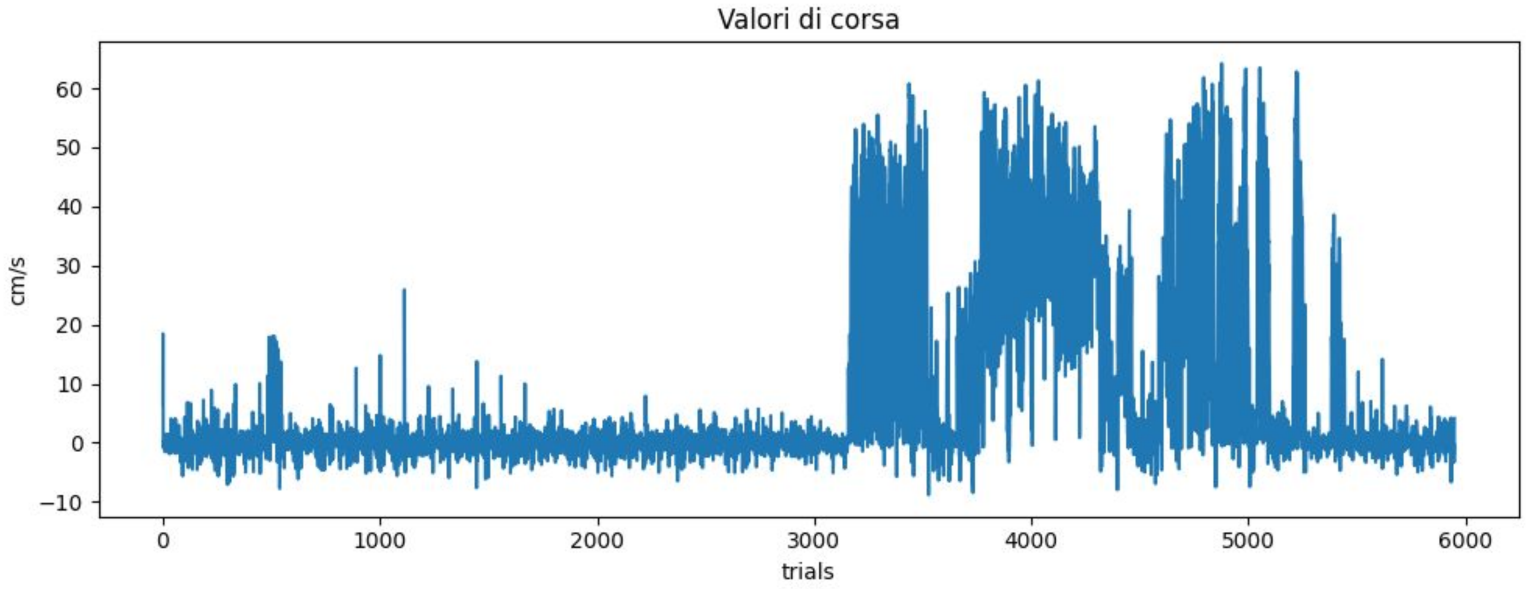
Image Resizing for V1T

The original images from the Allen dataset are 918×1174 pixels.

Since the V1T model expects input of size **144×256** we resize all images.

Running speed

For each trial we have a corresponding running speed (in **cm/s**)



VITA

Pupil Size

Pupil size is **not available** for our current experiment session.

However we left the methods to get pupil size size for other Allen Institute experiments as integral part of our code.

This makes it possible to add pupil size in a V1T project with a different experiment session.

```python
manifest_file = r'/brain_observatory/manifest.json'
boc = BrainObservatoryCache()
session_id = 510536157 #sessione differente
data_set = boc.get_ophys_experiment_data(ophys_experiment_id=session_id)

time, pupil_size = data_set.get_pupil_size()
time, pupil_location = data_set.get_pupil_location()
print(f"Shape di pupil_size: {pupil_size.shape}")
print(f"Shape di pupil_location: {pupil_location.shape}")

print("Prime 5 posizioni della pupilla (x, y) in cm:")
print(pupil_location[:5])

print("Prime 5 dimensioni della pupilla in pixel:")
print(pupil_size[:5])
```

```
Shape di pupil_size: (105951,)
Shape di pupil_location: (105951, 2)
Prime 5 posizioni della pupilla (x, y) in cm:
[[-6.1912417 23.951511 ]
 [-6.179465  23.758945 ]
 [-6.0974994 23.484304 ]
 [-6.1179376 23.902225 ]
 [-5.9426255 24.288134 ]]
Prime 5 dimensioni della pupilla in pixel:
[6677.7744 6664.99   6650.8438 6677.8315 6594.969 ]
```

Code Snippet to Load Pupil Size

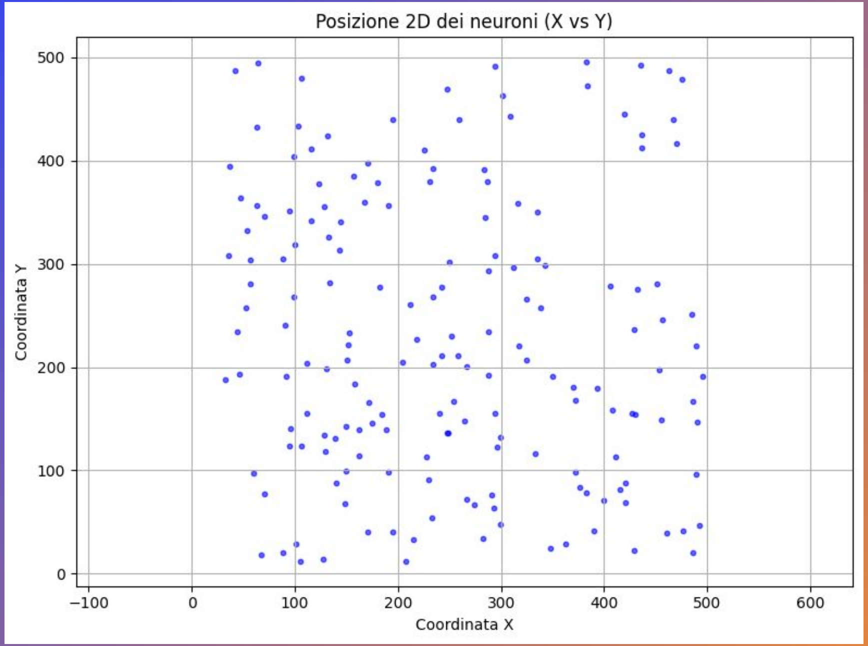
Behavior

Since pupil size and pupil derivative are unavailable we create an array with **running speed as the first value** and 0. as second and third value.

```
Informazioni sul file di comportamento Allen: 0.npy
  Percorso completo: /content/V1T/data/allen/501559087/data/behavior/0.npy
  Shape dei dati di comportamento: (3,)
  Tipo di dati (dtype): float32

Valori del vettore comportamento:
[6.439405 0.       0.      ]
```

VITA



Neuron Metadata

The first file we need is "`unit_ids.npy`" containing a numerical ID for each neuron.

```
[1, 2, 3, ... 173, 174]
```

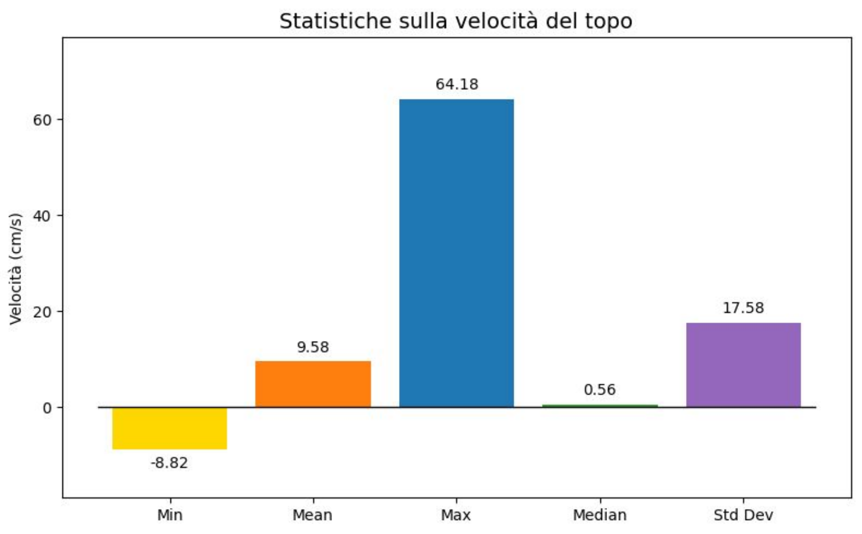
The second is "`cell_motor_coordinates.npy`" which contain the 3D coordinates for each neuron.

```
[292.91147   63.505207]
[274.3859    66.493774]
            …
[492.45386   47.046154]
```

Depth is the same (175μm) for each one.

This is used to in the Gaussian Readout module to enhance training performance.

VITA



Statistics

For images, behavior and neural responses we need to compute manually

`'min.npy', 'max.npy', 'mean.npy', 'median.npy', 'std.npy'`

These are used by the model for **normalization** and other processes before training.

For running speed we get:

```
Min: -8.82, Max: 64.18, Mean: 9.58,
Median: 0.56, Std: 17.58
```

VITA

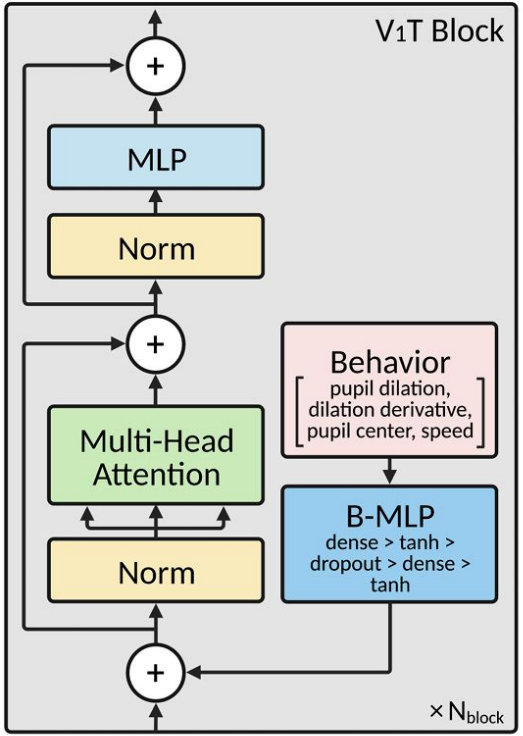
VIT ARCHITECTURE

An In Depth Look

V1T Architecture

- Input: images (1 channel, 36×64)
- Output: 174 predicted neural responses
- Approach: Vision Transformer + Gaussian2D Readout

In this project, the model takes images as input and predicts the response of **174 neurons**. It's based on a Vision Transformer and a readout mechanism called Gaussian2D.



Full Model Structure

This is the complete structure of the model.
Now we'll analyze each block step by step.

```
Model                         [16, 174]
├─ImageCropper: 1-1           [16, 1, 36, 64]
├─ViTCore: 1-2                [16, 155, 29, 57]
├─Readouts: 1-3               [16, 174]
├─ELU1: 1-4                   [16, 174]
```

Preprocessing: Image Cropper

Resizes the image to 36×64
Prepares images for the transformer

```
├─ImageCropper: 1-1                                   [16, 1, 36, 64]
│    └─Resize: 2-1                                    [16, 1, 36, 64]
```

The first step is an optional resizing. Input images can be resized to 36×64 pixels for efficiency

Encoding: Image2 Patches

Splits image into patches
Projects each patch into a 155-dimensional embedding

```
|    └─Image2Patches: 2-2                      [16, 1654, 155]        256,525
|    |    └─Sequential: 3-1                     [16, 1653, 155]        --
|    |    |    └─Unfold: 4-1                    [16, 64, 1653]         --
|    |    |    └─Rearrange: 4-2                 [16, 1653, 64]         --
|    |    |    └─Linear: 4-3                    [16, 1653, 155]        10,075
|    |    └─Dropout: 3-2                        [16, 1654, 155]        --
```

Each image is divided into small patches, and each patch is projected into an embedding vector. This is the first step toward converting an image into a sequence

Vision Transformer Core

4 Transformer blocks: Attention + MLP
Enables global interactions between patches

```
|    └─Transformer: 2-3                          [16, 1654, 155]      --
|    |    └─ModuleList: 3-17                     --                   (recursive)
|    |    |    └─ModuleDict: 4-5                 --                   (recursive)
|    |    |    |    └─Attention: 5-1             [16, 1654, 155]      384,865
|    |    └─DropPath: 3-4                        [16, 1654, 155]      --
|    |    |    └─MLP: 5-2                        [16, 1654, 155]      152,233
|    |    └─DropPath: 3-6                        [16, 1654, 155]      --
... (repeat 4 times: attention + MLP)
```

The core of the model is the Transformer. Each patch can communicate with all others through self-attention, learning spatial relationships

Rearranging Features

Transforms sequence back to spatial format

Shape: [1654, 155] → [155, 29, 57]

```
|    └Rearrange: 2-4                                      [16, 155, 29, 57]
```

Once the patch embeddings have been processed, we rearrange them into a 2D spatial map, which will be used by the readout layer

Gaussian2D Readout

Each neuron "looks" at a (x, y) location
Gaussian sampling + neuron-specific MLP

```
├─Readouts: 1-3                          [16, 174]      --
│    └─Gaussian2DReadout: 2-5            [16, 174]      27,840
│    │    └─Sequential: 3-19             [174, 2]       --
│    │    │    └─Linear: 4-12            [174, 30]      90
│    │    │    └─ELU: 4-13               [174, 30]      --
│    │    │    └─Linear: 4-14            [174, 2]       62
│    │    │    └─Tanh: 4-15              [174, 2]       --
```

Each neuron is associated with a **spatial location** on the 2D feature map. Around that location, we apply a Gaussian sampling and use a small MLP to compute the output

Final Non-Linearity

Applies ELU on final output
Helps mimic biological neuron firing pattern

```
├─ELU1: 1-4                                          [16, 174]
|     └─ELU: 2-6                                      [16, 174]
```

Finally, the output goes through an **ELU activation**. This helps the model better approximate the non-linear behavior of real neural responses

In `tensorboard.py` riga 19 modificare `"seaborn-v0_8-deep"`

In `tensorboard.py` commentare la riga 322 `f"pupil center..."` e mettere la virgola alla fine della riga precedente

In `data.py` aggiungere:

```
ALLEN = {
    "P": "501559087",
}
```

In `data.py` nella funzione `get_mouse2path` modificare:

```
assert ds_name in ("sensorium", "franke2022", "allen")
return SENSORIUM if ds_name == "sensorium" else FRANKE2022 if ds_name == "franke2022" else ALLEN
```

In `data.py` nella funzione `get_mouse_ids` aggiungere:

```
case "allen":
    all_animals = list(ALLEN.keys())
    if not args.mouse_ids:
        args.mouse_ids = all_animals
    for mouse_id in args.mouse_ids:
        assert mouse_id in all_animals
```

In `data.py` linea 220 al posto dell'else `np.arange(5950)`

```
    # load image IDs
    if ds_name == "sensorium":
        metadata["image_ids"] = load_trial("frame_image_id.npy")
    else:
        metadata['image_ids'] = np.arange(5950)
```

Necessary Changes

To make V1T work with a different dataset we need to **change the original code**.

As an example we add checks and dictionaries related to our new dataset in `data.py`.

And we create new `.npy` files for image data.

VITA

Necessary Changes

In `data.py` commentare o eliminare questa parte di codice alla riga 226

```
# load trial timestamps
#if load_timestamps:
#    if ds_name == "sensorium":
#        metadata["trial_ts"] = load_trial("frame_trial_ts.npy")
#    else:
#        metadata["trial_ts"] = load_trial("colorframeprojector_trial_ts.npy")
#    metadata["trial_ts"] = str2datetime(metadata["trial_ts"])

# load animal ID
#animal_ids = np.unique(load_neuron("animal_ids.npy"))
#assert len(animal_ids) == 1, f"Multiple animal ID in {os.path.dirname(meta_dir)}."
#metadata["animal_id"] = animal_ids[0]
```

In `data.py` riga 300 in `MiceDataset`

```
assert self.ds_name in ("sensorium", "franke2022", "allen")
```

In `data.py` aggiungere alla riga 395

```
def transform_behavior(self, behavior: np.ndarray):
    """Standardize behaviour in modo sicuro"""
    stats = self.behavior_stats
    std = stats["std"]

    # Evita divisione per 0 o NaN
    safe_std = np.where((std == 0) | np.isnan(std), 1.0, std)

    return behavior / safe_std
```

Adapting a foreign dataset to V1T has taken a considerable amount of our time.

Before training we did a lot of **trial and error** to find the file configuration to make V1T start.

As we can see on the left, we study and make changes to different parts of code such as troubling functions and datasets.

VITA

INTERACTIONS WITH VIT'S AUTHOR

Thoughts and Recommendations

Important Files

Bryan Li <bryanlimy@gmail.com>
a me

gio 12 giu, 12:13

Hi Nunzio,

Thank you for reaching out!

I haven't ran the V1T in a while so I am recalling this from memory. There are files in the metadata folder that are important for model training so it is best for you to extract them from the allen brain dataset. There are also files in the metadata folder that is only needed for the Sensorium 2022 challenge so you can skip those or just create a dummy file so the code runs.

Files that are important

- meta/statistics: all the files in meta/statistics (e.g. meta/statistics/behavior/all/max.npy, meta/statistics/behavior/all/mean.npy, etc.) are important. They record basic statistics of the training set, which are then used to preprocess (i.e. normalizing or standardizing) the input and output of the model during model training and inference. For instance: Natural images, recorded responses, and behavioral variables (i.e. pupil dilation, dilation derivative, pupil center, running speed) were standardized using the mean and standard deviation measured from the training set. You should be able to compute these statistics from the training set (however you want to define it) of the allen brain dataset.
- meta/trials/tiers.npy: this file inform the training, validation and test split of the dataset, so you will have to create this file for the data split you want to use.
- meta/neurons/cell_motor_coordinates.npy: this file has the neuron coordinates in 3D space. This is needed for the Gaussian readout in the model. I am not sure if this is available in the allen brain dataset? You might have to use a different readout if this is not available.

Files that are not important and I think you can just create a placeholder file

- meta/neurons/unit_ids.npy: this record the neuron IDs, I think you can just do np.arange(num_neurons) to replace that.
- meta/trial_idx.npy: this record the trial IDs, I think you can just do np.arange(num_trials).

frame_trial_ds.npy and animal_ids.npy are not used at all, you can either create dummy files, or simply comment out those lines of code!

Best
Bryan

Placeholder and Unavailable Files

Bryan Li <bryanlimy@gmail.com>
a me

13 giu 2025, 17:20

Does the Allen brain dataset have pupil center? And other behaviour variables such as pupil dilation, pupil dilation derivatives and running speed?

From our experiments, training the model with behaviours significantly improve performance. If you have some behaviour variables but not all, you might have to modify the input to the model. You shouldn't create dummy files for behaviours because they are actually being used by the model.

An alternative is to train the model without behaviour. You can do that by using —behaviour_mode=0. But the model won't be as good.

Best,
Bryan

Since data as pupil center, pupil dilation and pupil derivatives are **not available** for our session, we can use `--behavior_mode=1` but we measure no significant improvement

According to our research and Bryan's findings this explains our training results.

TRAINING

Highlights

Training Parameters

The model can be run using different configurations, passed through parameters. The parameters that have been set for the various training tests are:

Parameters	Type	Default	Description
--dataset	str	REQUIRED	Path to the data folder
--output_dir	str	REQUIRED	Where to save results/model
--behavior_mode	int	REQUIRED	Behavior Mode (0-4)
--resize_image	int	1	If 1, resize to (36×64), if 0, leave (144×256)
--gray_scale	flag	False	If True, converts image to grayscale
--limit_data	int	None	Limit the number of samples

Training Parameters

Other important parameters for training are the following

Parameter	Type	Default	Description
--epochs	int	400	Maximum number of training epochs
--batch_size	int	8	Global batch size
--criterion	str	poisson	Loss function
--core	str	REQUIRED	”conv", "vit", "cct", "stn", "stacked2d"
--readout	str	REQUIRED	"gaussian2d"
--shift_mode	int	0-4	

Training sessions

Several **training sessions** were conducted, each with different training parameters. In the next slides we show a **summary** of the most significant.

From the first session to the last, performance improves, both because of the increasing number of data samples and more suitable hyperparameters

```
[ ] #Sessione 1
    !python train.py --dataset data/allen --output_dir runs/v1t_model --core vit
    --epochs 300 --readout gaussian2d --behavior_mode 0 --shift_mode 0 --batch_size
    8 --limit_data 1000 --verbose 1 --lr 0.005
```

```
[ ] #Sessione 2
    !python train.py --dataset data/allen --output_dir runs/v1t_model --core vit
    --epochs 300 --readout gaussian2d --behavior_mode 0 --shift_mode 0 --batch_size
    8 --limit_data 1000 --verbose 1 --lr 0.005 --criterion correlation
```

```
[ ] #Sessione 3
    !python train.py --dataset data/allen --output_dir runs/v1t_model --core vit
    --readout gaussian2d --behavior_mode 0 --shift_mode 0 --epochs 300 --batch_size
    8 --limit_data 3000 --lr 0.008 --adam_beta2 0.98 --criterion correlation
```

```
[ ] #Sessione 4
    !python train.py --dataset data/allen --output_dir runs/v1t_finetune --core vit
    --readout gaussian2d --behavior_mode 0 --shift_mode 0 --resize_image 1
    --batch_size 16 --lr 0.0005 --core_lr 0.0005 --epochs 300 --adam_beta1 0.9
    --adam_beta2 0.98 --adam_eps 1e-6 --save_plots --verbose 3
```

```
[ ] #Sessione 5
    !python train.py --dataset data/allen --output_dir runs/v1t_model --core vit
    --readout gaussian2d --behavior_mode 0 --shift_mode 0 --batch_size 16 --verbose
    3 --lr 0.004 --core_lr 0.007 --epochs 300 --save_plots
```

```
[ ] #Sessione 6
    !python train.py --dataset data/allen --output_dir runs/v1t_model --core vit
    --readout gaussian2d --behavior_mode 1 --shift_mode 0 --batch_size 16 --verbose
    3 --lr 0.004 --core_lr 0.007 --epochs 300 --save_plots
```

Training Sessions Summary

Session 1

Limit data is used to train quickly due to timing and computing power restriction.

`core_lr` takes the default value, since it's not explicitly set. The **batch size** has been reduced to 8.

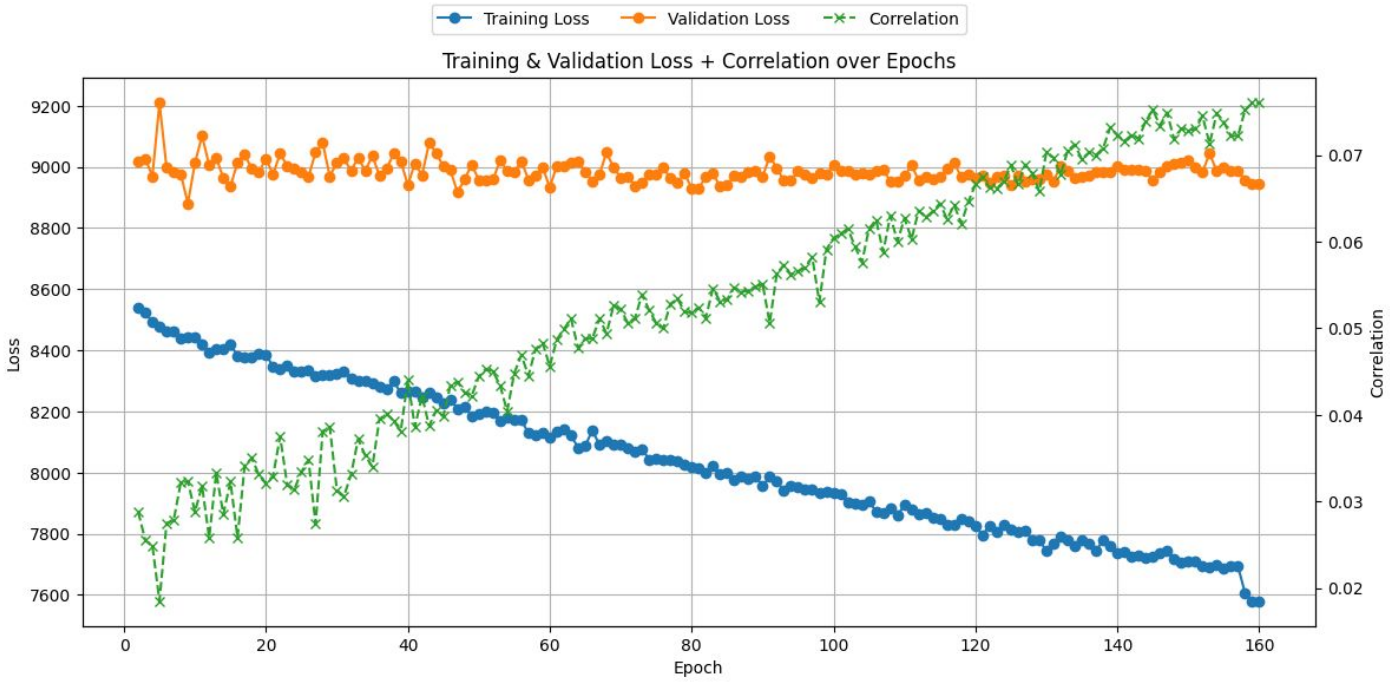
```
!python train.py --dataset data/allen --output_dir runs/v1t_model
--core vit --epochs 300 --readout gaussian2d --behavior_mode 0
--shift_mode 0 --batch_size 8 --limit_data 1000 --verbose 1 --lr 0.005
```

Session 1

Epoch 1
Train loss: 8366
Validation loss: 8937
Correlation: 0.0251

Epoch 160
Train loss: 7579
Validation loss: 8943
correlation: 0.0760

The run was stopped at epoch 160, as the model dynamically reduces the lr.



Session 2

In this session we try a **different loss function**, this is set via the criterion parameter.

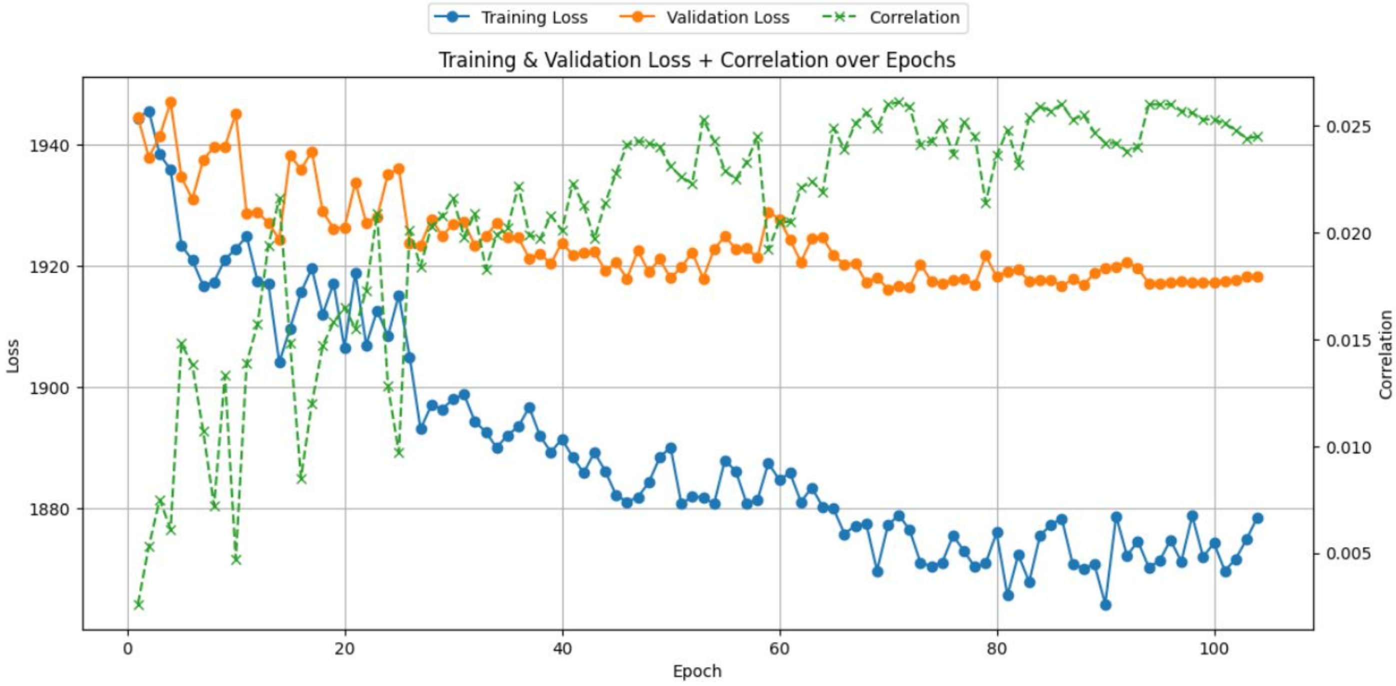
Most of the other parameters remain the same, while some are tweaked slightly

```
!python train.py --dataset data/allen --output_dir runs/v1t_model
--core vit --epochs 300 --readout gaussian2d --behavior_mode 0
--shift_mode 0 --batch_size 8 --limit_data 1000 --verbose 1 --lr 0.005
--criterion correlation
```

Session 2

Epoch 1
Train loss: 1944
Validation loss: 1944
correlation: 0.0026

Epoch 104
Train loss: 1878
Validation loss: 1918
correlation: 0.0245



Session 3

In this session we increase the **learning rate** and the **limit data** parameter.

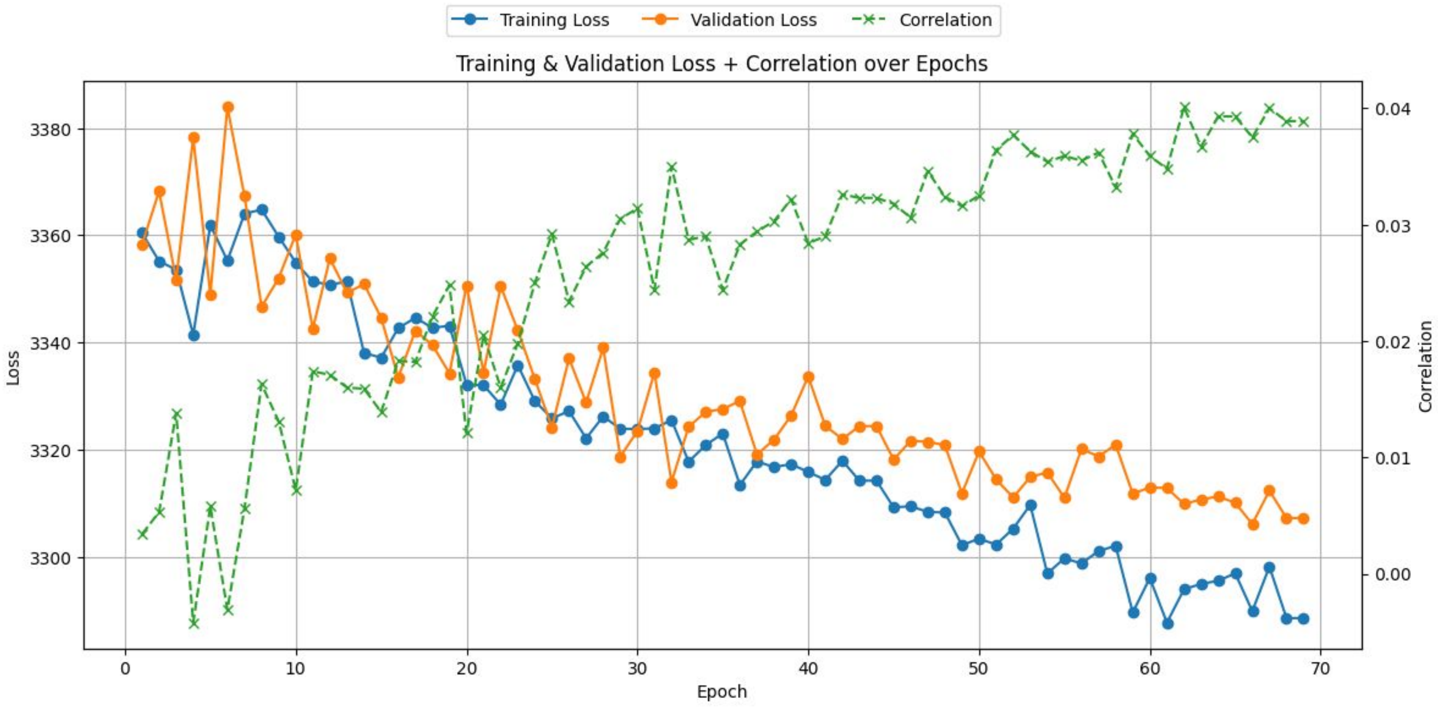
We also change the optimizer: now it's more conservative in automatically adjusting the learning rates (because the variance estimate changes more slowly).

```
!python train.py --dataset data/allen --output_dir runs/v1t_model
--core vit --readout gaussian2d --behavior_mode 0 --shift_mode 0
--epochs 300 --batch_size 8 --limit_data 3000 --lr 0.008 --adam_beta2
0.98 --criterion correlation
```

Session 3

Epoch 1
Train loss: 3361
Validation loss: 3367
correlation: 0.0068

Epoch 69
Train loss: 3288
Validation loss: 3307
correlation: 0.0402



Session 4

In this session we lower the learning rate and try **image resizing** (to 36x64).

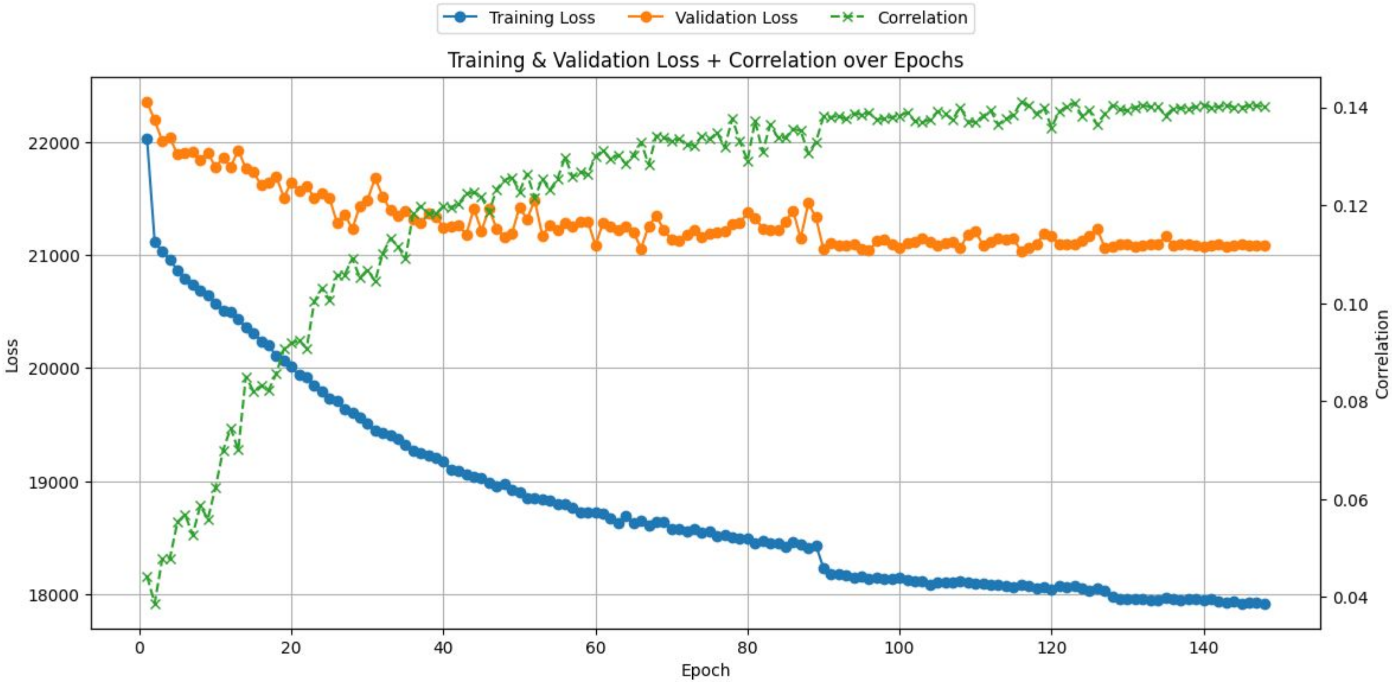
We also changed the **verbose** parameter to give us more informations about training.

```
!python train.py --dataset data/allen --output_dir runs/v1t_finetune
--core vit --readout gaussian2d --behavior_mode 0 --shift_mode 0
--resize_image 1 --batch_size 16 --lr 0.0005 --core_lr 0.0005 --epochs
300 --adam_beta1 0.9 --adam_beta2 0.98 --adam_eps 1e-6 --save_plots
--verbose 3
```

Session 4

Epoch 1
Train loss: 22028
Validation loss: 22353
correlation: 0.0442

Epoch 148
Train loss: 17924
Validation loss: 21102
correlation: 0.1402



Session 5

In this session we use two separate learning rates (readout and core). This is done by **removing limit data**.

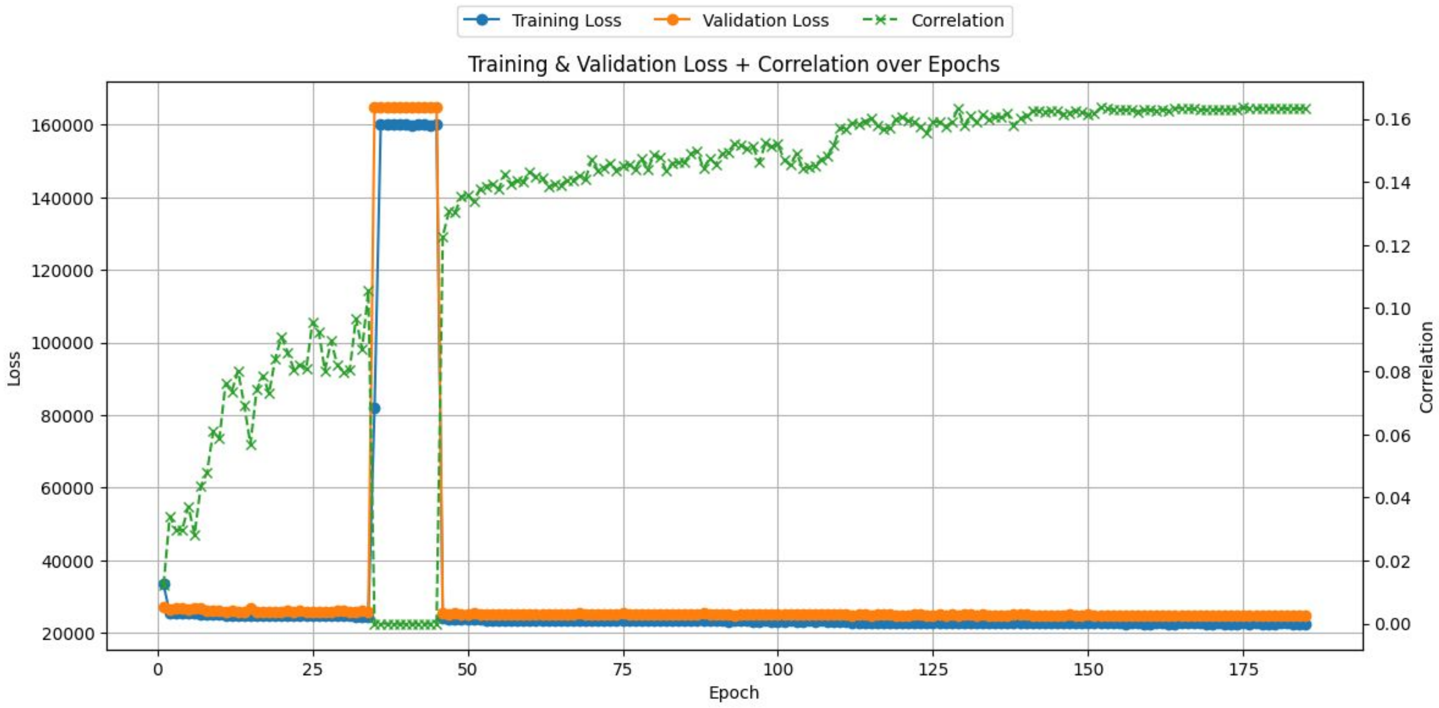
We also switch to Colab Pro and use all the samples available since we now have the computational power to do it.

```
!python train.py --dataset data/allen --output_dir runs/v1t_model
--core vit --readout gaussian2d --behavior_mode 0 --shift_mode 0
--batch_size 16 --verbose 3 --lr 0.004 --core_lr 0.007 --epochs 300
--save_plots
```

Session 5

Epoch 1
Train loss: 33400
Validation loss: 27170
correlation: 0.0121

Epoch 185
Train loss: 22594
Validation loss: 24921
correlation: 0.1633



Session 6

We change the original V1T code and we're now able to use **behaviors** during training, particularly running speed.

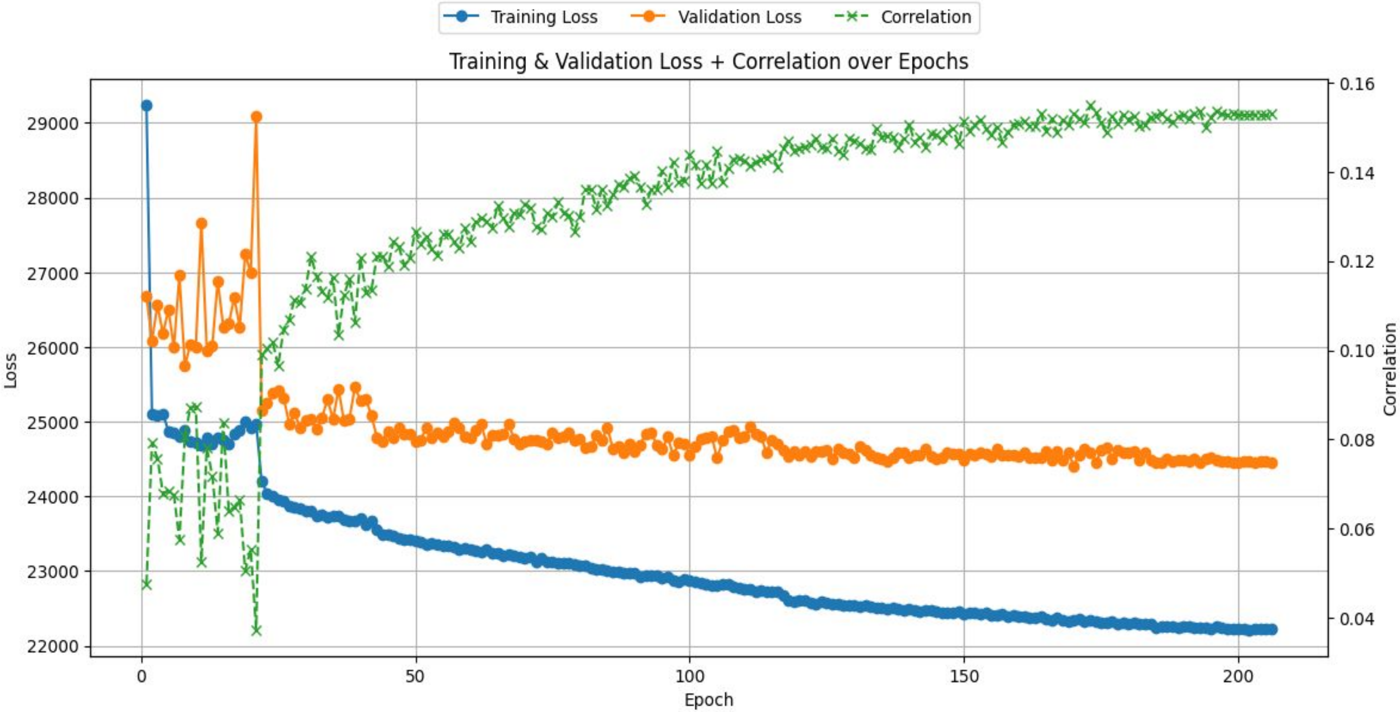
However this doesn't have much impact on performance because pupil data is not available.

```
!python train.py --dataset data/allen --output_dir runs/v1t_model
--core vit --readout gaussian2d --behavior_mode 1 --shift_mode 0
--batch_size 16 --verbose 3 --lr 0.004 --core_lr 0.007 --epochs 300
--save_plots
```

Session 6

Epoch 1
Train loss: 29232
Validation loss: 26684
correlation: 0.0476

Epoch 206
Train loss: 22218
Validation loss: 24460
correlation 0.1529



CONCLUSIONS

And Closing Statements

Best Parameters

```
options:
  -h, --help            show this help message and exit
  --dataset DATASET     path to directory where the dataset is stored.
  --output_dir OUTPUT_DIR
  --mouse_ids MOUSE_IDS [MOUSE_IDS ...]
                        Mouse to use for training.
  --behavior_mode {0,1,2,3,4}
                        behavior mode:0: do not include behavior1: concat
                        behavior with natural image2: add latent behavior
                        variables to each ViT block3: add latent behavior +
                        pupil centers to each ViT block4: separate BehaviorMLP
                        for each animal
  --center_crop CENTER_CROP
                        crop the center of the image to (scale * height,
                        scale, width)
  --resize_image {0,1}  resize image mode:0: no resizing, return full image
                        (1, 144, 256)1: resize image to (1, 36, 64)
  --gray_scale          convert colored image to gray-scale
  --limit_data LIMIT_DATA
                        limit the number of training samples.
  --num_workers NUM_WORKERS
                        number of works for DataLoader.
  --epochs EPOCHS       maximum epochs to train the model.
  --batch_size BATCH_SIZE
  --micro_batch_size MICRO_BATCH_SIZE
                        micro batch size to train the model. if the model is
                        being trained on CUDA device and micro batch size 0 is
                        provided, then automatically increase micro batch size
                        until OOM.
```

Training Parameters (Epochs & Batch Size)

We tried different values for core learning rate and readout learning rate, increasing the batch size to 16, and set the readout gaussian 2d improved the correlation.

In particular, the best readout suggested by the Bryan Li is **Gaussian 2D**. This takes into account the neuron coordinates from `cell_motor_coordinates.npy`.

This readout accounts for spatial arrangement of neurons to model their responses

As for the best starting lr for the core part it is 0.004, for the readout part it is 0.007.

Conclusions

In addition to the sessions shown, other training sessions were carried out as well. We tried **different lr**, **lr_core**, batch size and **loss**.

The correlation value remains static because of unavailable behaviors.

As brain response is heavily influenced by its current state.

The session with best results is the 5th, reaching a correlation value 0.16

```
--criterion CRITERION
                      criterion (loss function) to use.
--ds_scale {0,1}      scale loss by the size of the dataset
--pretrain_core PRETRAIN_CORE
                      path to directory where pre-trained core model is
                      stored.
--save_plots          save plots to --output_dir
--dpi DPI             matplotlib figure DPI
--format {pdf,svg,png}
                      file format when --save_plots
--use_wandb
--wandb_group WANDB_GROUP
--clear_output_dir    overwrite content in --output_dir
--verbose {0,1,2,3}
--core CORE           The core module to use.
--readout READOUT     The readout module to use.
```

Training Parameters (Loss & Readout)

VITA

THANK YOU

Fernando Riccioli, Nunzio Fornitto, Angelo Frasca