# USING GENERATIVE AND EXPLAINABLE NEURAL NETWORKS TO INVESTIGATE THE RELATIONSHIP BETWEEN MOTOR CORTEX ACTIVITY AND ANIMAL BEHAVIOR DURING SKILLED REACH LEARNING

**SEAN TANABE**
**Bachelor of Science, University of Ottawa, 2017**

A thesis submitted
in partial fulfilment of the requirements for the degree of

**MASTER OF SCIENCE**

in

**NEUROSCIENCE**

Department of Neuroscience
University of Lethbridge
LETHBRIDGE, ALBERTA, CANADA

USING GENERATIVE AND EXPLAINABLE NEURAL NETWORKS TO INVESTIGATE THE RELATIONSHIP BETWEEN MOTOR CORTEX ACTIVITY AND ANIMAL BEHAVIOR DURING SKILLED REACH LEARNING

SEAN TANABE

Date of Defense: August 20, 2024

| | | |
|---|---|---|
| Dr. Artur Luczak<br>Thesis Supervisor | Professor | Ph.D |
| Dr. Masami Tatsuno<br>Thesis Examination Committee | Professor | Ph.D |
| Dr. Ian Whishaw<br>Thesis Examination Committee | Professor | Ph.D |
| Dr. Aaron Gruber<br>Thesis Examination Committee | Professor | Ph.D |
| Dr. Sergio Pellis<br>Chair, Thesis Examination Committee | Professor | Ph.D |

## ABSTRACT

Understanding complex relations between neuronal activity and animal behavior is one of the most crucial questions in neuroscience. Rapid advancements in Machine Learning (ML) methods offer new powerful tools that can be used to investigate highly non-linear mapping between motor cortex activity and body movements. Here, by using explainable convolutional network (ConvNet) and Generative Adversarial Networks (GAN), we show how neuronal activity can be predicted from raw videos of animal behavior, and interestingly, we show that detailed videos of behaving animals can be recreated from activity of just few selected neurons. Those analyses revealed that the predictability of behavior from neuronal activity (and vice versa) initially increases as an animal learns a new task. However, after the animal performance on a motor task achieves the required accuracy, then "coupling" between neuronal activity and behavior decreases, without degrading task performance. In summary, we aimed to improve the understanding of skilled reach learning by moving past the need for predefined limb markers. By using advanced, data-driven machine learning, we were able to recreate behavioral videos from neural activity and predict neuronal firing patterns from raw videos, providing new insights into the complex processes involved in skilled reach behavior.

To provide all the details about our models and analyses we posted our code with documentation at: https://github.com/seantanabe/GAN_ConvNet_rat_reach_motor_cortex. There are also included original and generated videos and neuronal data.

## CONTRIBUTIONS OF AUTHORS

The authors for this manuscript are; Sean Tanabe, Michael Eckert, Ian Q. Whishaw, Masami Tatsuno, Artur Luczak.

S.T analysed data and wrote the manuscript. M.E designed and conduced experiment, preprocessed data and wrote the manuscript. I.Q.W wrote the manuscript. M.T designed the experiment and wrote the manuscript. A.L contributed analysis and wrote the manuscript.

## ETHICS STATEMENT

All procedures adhered to the Canadian Council for Animal Care and were approved by the University of Lethbridge Animal Welfare Committee.

| Title | Number | Date |
|---|---|---|
| Neural information processing during sleep: memory function of slow-wave sleep and REM sleep | 1307 | April 30, 2013 |

## USE OF GENERATIVE AI

This thesis adheres to SGS Policies and Procedures for the use of generative AI. The manuscript focuses on generative AI, specifically the generation of video data from motor cortical neural signals to advance neuroscience. Separately, ChatGPT-4 generative AI is employed to enhance the readability of the writing. Any text generated by ChatGPT-4 is extensively edited and revised by the authors, ensuring that the final content remains our own work. The research itself is entirely original, with ChatGPT-4 used solely to assist in the presentation of our ideas.

## TABLE OF CONTENTS

## LIST OF FIGURES

## LIST OF ABBREVIATIONS

| | |
|---|---|
| ML | Machine Learning |
| GAN | Generative Adversarial Networks |
| ConvNets | Convolutional Neural Networks |
| cGAN | Conditional GAN |
| LIME | Local Interpretable Model-Agnostic Explanations |
| BMI | brain-machine interfaces |
| FC | fully connected |

# CHAPTER 1: INTRODUCTION

The motor cortex is organized into a 'motor map', where specific motor areas control each body part, as exemplified by Penfield's motor homunculus (Penfield & Rasmussen, 1950), yet its precise function remains debated (Purves et al., 2017). Early experiments demonstrated that electrical stimulation of map regions could activate individual muscles, whereas more recent findings indicate that prolonged stimulation can evoke complex actions, such as reaching (Graziano et al., 2002; Brown & Teskey, 2014; Baldwin et al., 2016). This has sparked a debate over whether cortical maps represent single muscles or complex action sequences. Complicating matters, regions active during movement are also implicated in other functions, including planning (Cisek & Kalaska, 2010), perception (e.g., mirror neurons; Fabbri-Destro & Rizzolatti, 2008), and sensory integration (Scott, 2016). To investigate neuronal activity in the motor cortex, here we employed the new Machine Learning (ML) tools capable of complex input-output mappings which are particularly useful for studying highly non-linear relations between neuronal activity and behavior. Our motivation was to advance the understanding of skilled reach learning by moving beyond the assumption of predefined limb markers. The application of these advanced, data-driven machine learning techniques allowed us to reconstruct behavioral videos directly from neural activity and predict neuronal firing patterns from raw videos, revealing new insights into the dynamics of skilled reach learning.

Single neuron experiments have significantly advanced our understanding of motor control, particularly in decoding how the brain represents movement. Georgopoulos et al. 1988 introduced the neuronal population vector, illustrating that a collective of motor cortical neurons can predict the direction of reaching movements based on their directional

tuning. This foundational work emphasized static movement parameters, such as direction. Hatsopoulos et al. 2007 further developed this by showing that single neurons encode not only instantaneous movement details but also temporally evolving trajectories, indicating their role in encoding more complex movement sequences. Building on these foundational studies, our research leverages advanced machine learning techniques to explore the dynamic relationship between neuronal activity and skilled reach behavior. This approach allows us to move beyond traditional single neuron analyses, offering deeper insights into the complexities of motor control.

In this study, we apply Generative Adversarial Networks (GAN) and explainable Convolutional Neural Networks (ConvNets) to investigate the relationship between behavior and motor cortical activity. GANs produce synthetic data that closely mirrors real data by leveraging the competition between two neural networks: the Generator, which replicates the true data distribution, and the Discriminator, a binary classifier that differentiates between real and fake samples (Goodfellow et al., 2014). GANs, known for their flexibility and efficacy, have seen widespread application beyond image processing and computer vision, extending into healthcare, biology, astronomy, remote sensing, material science, and finance (Dash et al. 2023). Conditional GANs (cGANs), which incorporate additional information, yield more realistic and diverse outputs suited to specific conditions, enhancing their applicability in areas like image translation, text-to-image synthesis, and super-resolution (Mirza and Osindero 2014). This paper introduces an innovative use of cGANs, employing neural data to create behavioral videos, demonstrating the unique capabilities of cGANs in producing contextually relevant synthetic outputs.

The ConvNet has revolutionized image recognition by extracting features from data using convolution structures, inspired by the neuronal architecture of visual systems in mammalian brains, notably emulating the complex cell sequences in a cat's visual cortex (Alzubaidi et al., 2021; Li et al., 2022). This design, influenced by visual perception, enables ConvNets to autonomously identify relevant features without human supervision, leading to extensive applications across various fields such as computer vision, speech processing, and face recognition (Alzubaidi et al., 2021). Despite these advancements, interpreting the reasons behind the models' predictions remains challenging. The Local Interpretable Model-Agnostic Explanations (LIME) explain model addresses this by providing interpretable explanations of which parts of the image were most informative for model decision (Ribeiro et al., 2016). This study leverages ConvNets and the LIME explain model to decode complex movements associated with neuronal activity, presenting an innovative method for extracting receptive fields, thus enhancing our understanding of neural mechanisms behind motor movements.

Here, we use raw videos for analyses and the ML algorithm is allowed to explore which movement components are the most related to neuronal activity. This procedure eliminates experimental bias relating neuron activity with selective body parts and allows for the derivation from videos of the movements most predictive of neuronal patterns. We demonstrated that using such a data-driven approach resulted in discovering how the relation between motor cortex activity and animal behavior is changing during learning a new motor task.

# CHAPTER 2: METHODS

## 2.1 ANIMAL AND SURGERY

All procedures adhered to the Canadian Council for Animal Care and were approved by the University of Lethbridge Animal Welfare Committee. Five adult male Fisher-Brown Norway rats (Rattus norvegicus), aged between four and nine months and weighing 350 to 450 g, were used. The rats, housed under a reverse light-cycle, underwent experiments during their dark cycle. They had continuous access to food and water, except during reaching task training when their diet was limited to 85% of their body weight. Initial hand preference tests were followed by implanting a hyperdrive with 12 tetrodes and 2 reference electrodes into the primary motor cortex, as described in detail in (Eckert et al., 2020) (coordinates: 1.0 mm anterior, 2.5 mm lateral to bregma). The reaching area was confirmed by stimulating the region and observing arm twitches. Post-surgery, rats received 3 days of analgesics (Metacam) and 5 days of antibiotics (Baytril), with a one-week recovery before recording.

## 2.2 RECORDING AND BEHAVIORAL TRAINING

For electrophysiological recordings, we used digital Cheetah SX data acquisition software (Neuralynx, Boseman, Montana). Signals were bandpass filtered (600–6000 Hz) and sampled at 32 kHz. A reference electrode in the drive was placed in the white matter below the cortex. Tetrodes were gradually lowered during a habituation period of one to two weeks to deep layers of the motor cortex. Adjustments were made during the training period to optimize cell yield.

The single pellet reaching task requires an animal to use one hand to reach for, grasp, and bring a single food item to its mouth, similar to a common behavior in both nonhuman primates and humans (Whishaw and Pellis, 1990). The task uses rectangular, clear Plexiglas boxes that allow filming from any angle. Their open design enables the animal to approach and leave the food location freely, choose a hand for reaching, and exhibit adaptive behavioral changes (Whishaw and Pellis, 1990; Alaverdashvili and Whishaw, 2013; Klein et al. 2012). The single-pellet reaching task used a box with a 1.5 cm slot from which the rat retrieved a 45 mg sugar pellet (Bio-Serv, Frenchtown, NJ, USA), placed in a well 1.5 cm from the slot on a 3 cm high shelf.

High-speed infrared cameras recorded the task at 200 fps (Prosilica GigE). Habituation involved closed shelf doors, with the rat occasionally finding pellets on the floor. Rats, naive to skill training, underwent reach training with 60 trials per session, continuing until the asymptotic performance was achieved for three consecutive days (less than 5% increase in performance across 3 days). A trial was deemed successful if the rat grabbed, retrieved, and ate the pellet. All other outcomes, like misses or drops, were classified as unsuccessful.

Spikes were sorted using automated clustering (Rossant et al., 2016) and manually refined (MClust). Sorted unit quality was evaluated based on inter-spike interval histogram,

cross-correlation with other units, waveform shape consistency, firing rate patterns, and L-ratio (Eckert et al., 2020). One of the five rats was excluded due to insufficient firing rates.

### 2.3 PREDICTING CORTICAL ACTIVITY FROM BEHAVIORAL VIDEO

Our objective was to reconstruct motor cortical activity using information from behavioral videos. We focused on reconstructing cortical neuron activity from -400ms to 600ms around the time of reach initiation, in increments of 50ms, applying a Gaussian filter (500ms length, 2.5 alpha). A pre-trained Inception-V3 deep convolutional neural network (ConvNet) (Szegedy et al., 2016) extracted 2,048 high-level features from each frame. Originally trained on the ImageNet dataset, which includes images from 1,000 classes such as "dog," "car," and "make-up," this network demonstrated strong performance on behavioral datasets (Ryait et al., 2019, Torabi et al. 2021). Each neuron was predicted separately (the median number of neurons per rat per session was 20 with a range from 6 to 51). Neurons with spike firing rates below 1Hz were excluded. In the Inception-V3 ConvNet, we retrained a fully connected (FC) last layer with a single unit to reconstruct the activity of a single actual neuron. For those analyses, we used a leave-one-out cross-validation.

Additionally, the LIME explain model was used to analyze the receptive field of neural activity as predicted by ConvNet using leave-one-out cross-validation. The LIME explanation model simplifies understanding neural network predictions, like ConvNet, by altering inputs slightly and observing the impact on the network's predictions (Ribeiro et al. 2016). This involves making small changes to the data that the network processes, and then comparing these adjusted predictions to the original ones. The difference between the new prediction and the original, the explain value, tells us how important that specific piece

of information was for the prediction. Through this comparison, the model learns about the network's behavior. To quicken this learning process, we modified specific segments of the input, such as altering a 4x4 pixel section within a larger 150x150 pixel frame.

## 2.4 RECONSTRUCTION OF BEHAVIORAL VIDEO FROM CORTICAL ACTIVITY

We generated behavioral videos from motor cortical activity with conditional Generative Adversarial Networks (cGAN). GANs train a model using two competing parts: a generator that creates data mimicking the real data distribution, and a discriminator that tries to tell apart real data from the generated data. GANs can be adapted into a conditional model where both parts are influenced by additional information (Mirza and Osindero, 2014). Here, we used GAN conditioned by the neural activity of six neurons with the highest predictability from our ConvNet analysis, to create 128x128 pixel images of a rat reaching, using leave-one-out trial cross-validation. The neural activity is Gaussian-filtered (500ms length, SD: 100ms) -400ms to 600ms around reaching movement initiation, with 20ms interval steps.

To compare the accuracy of movement reconstruction we applied DeepLabCut neural network (Mathis et al., 2018) to track the position of selected body parts in actual and generated videos. Specifically, we traced the position of both hands and the nose (Suppl Figure 1). DeepLabCut combines object recognition and semantic segmentation, using pre-trained ResNets and deconvolutional layers. Specifically, it is a variation of ResNets trained on the ImageNet database, that replaces its classification layer with deconvolutional layers to enhance visual information and generate spatial probability maps. These maps predict the likelihood of body part locations. For specific tasks, the network is fine-tuned

with labeled images, adjusting its weights to accurately locate body parts. We threshold the position likelihood at 0.4 for the nose and hands. The DeepLabCut model was trained solely on actual video and then applied location prediction to both actual and GAN videos. This approach enabled us to quantify the precision of reconstructing behaviors from motor cortical activities by comparing the alignment of actual and GAN-produced limb positions.

### 2.5 DATA AVAILABILITY

To provide all the details about our models and analyses we posted our code with documentation at: https://github.com/seantanabe/GAN_ConvNet_rat_reach_motor_cortex. There are also included original and generated videos and neuronal data.

## CHAPTER 3: RESULTS

To investigate the relation between behavior and neuronal activity, we used data recorded from 4 rats learning a skilled reaching task for a food pellet (Methods) (Whishaw and Pellis, 1990). In the single-pellet reaching task, rats are individually placed in a Plexiglas chamber and are trained to reach through an opening to retrieve sucrose pellets located in an indentation on a shelf attached to the front of the chamber (Fig. 1A) (Whishaw and Pellis, 1990; Ryait et al., 2019). A rat uses a single limb to reach through the opening and grasp a food pellet for eating (Methods). The video is recorded from a frontal view. Simultaneously, the single-unit activity of multiple neurons in the motor cortex was recorded using a head stage with 12 tetrodes (Gothard et al., 1996; Eckert et al., 2020) (Methods). A sample activity of 5 neurons across 65 reaching trials during a single recording session (1 day) is illustrated in Fig. 1B. Animals were trained daily until asymptotic performance was achieved for three consecutive days (less than 5% increase in performance across 3 days; between 10-20 days). To compare the performance of rats across different duration of training periods, for each rat we defined Early, Mid, and Late training days, representing the average data of the first, middle, and last three days of training days, respectively. This is illustrated in Fig. 1C, which shows the improvement in the pellet reaching task from the Early to the Late stage of training (Friedman's test, $\chi^2 = 8$, $p = 0.0183$).

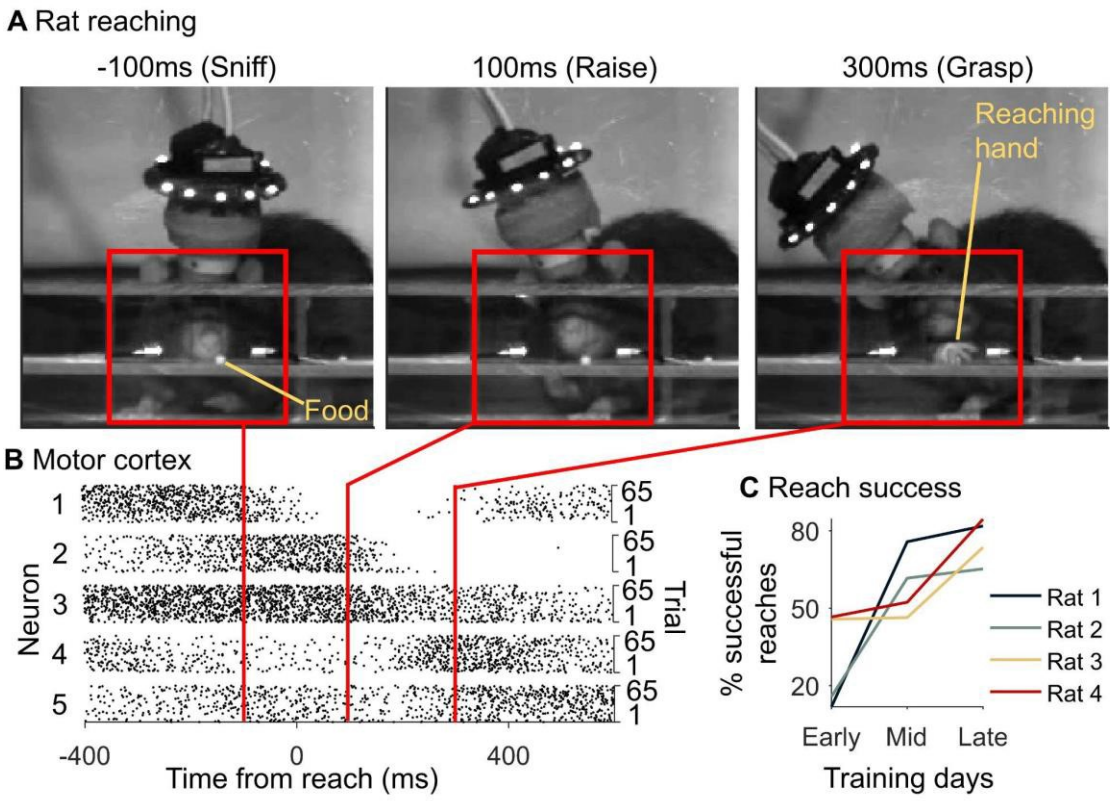



**Figure 1**. A rat performing a reach in the single-pellet reaching task. (**A**) Three sample video frames illustrating the task. Red squares denote the area in the video frame used for neural network analysis, and red lines indicate times of neuronal activity corresponding to those frames. (**B**) Representative activity of 5 neurons in the motor cortex across 65 reach trials. Time 0 indicates the beginning of reach movement. (**C**) The reach accuracy increases across training days, categorized as Early, Mid, and Late training phases.

### 3.1 PREDICTING NEURAL ACTIVITY FROM BEHAVIORAL VIDEOS

To investigate the relation between skilled reaching and neuronal activity, we first asked how well neuronal firing patterns can be predicted from videos of skilled reaching. For that, each video frame was processed by ConvNet to extract 2048 image features (Figure 2A). This is analogous to how the visual system in the brain decomposes retinal

images in a set of features (e.g. neurons responding to specific line orientation, edges, etc.) (Hubel and Wiesel, 1959; Yamins et al., 2014). For computational efficiency we restricted analyses to 150 x 150 pixel window (denoted with a red rectangle in Fig. 1A). Next, we trained a shallow neural network to predict from extracted features, the activity of individual neurons (Methods). For those analyses, the spiking activity of each neuron was smothered with a Gaussian kernel ($\sigma$ = 100 ms), as described in Saleh et al., 2010. From each recording day, 6 to 51 neurons (median 20) from each animal. Examples of actual and predicted activity during a single reaching trial are shown in Figure 2B. To quantify the similarity between actual and predicted activity patterns we used Euclidean distance. The distribution of distances during each behavioral training phase is shown in Figure 2C. For comparison, we also calculated distances between the predicted activity of one neuron and the actual activity of a different neuron during the same trial. The distribution of distances for shuffled data has a significantly larger median distance as compared to original data (Wilcoxon signed-rank test, Early/Mid/Late all $p < 0.00001$, the median is 1.30/1.08/1.22, N = 660/720/600). Interestingly, we observed that the distance values were the lowest during the Mid training days, a finding consistently observed in the results of all four rats (Figure 2D, Friedman's test, $\chi^2 = 6$, $p = 0.0498$). This indicates that the predictability of neuronal activity from behavior is not continuously increasing as animals are learning the task, but rather the predictability peaks in Mid days (see Discussion how this may suggest a change in allocation of neuronal resources during learning phases).

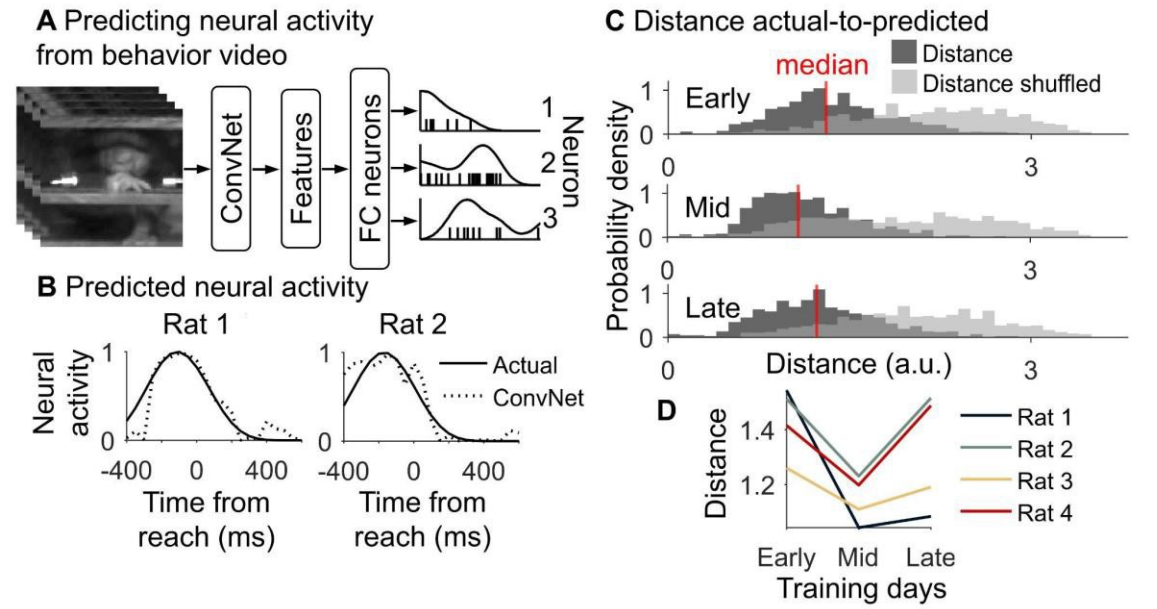



**Figure 2**. Predicting neuronal activity from videos of skilled reaching. (**A**) Pipeline of the predicting neural activity from a video. Each frame is processed by ConvNet, and obtained image features are used by the subsequent network to predict the smoothed activity of individual neurons. (**B**) Representative examples of actual and predicted neuronal activity during a single reaching trial. (**C**) Distribution of distances between actual and predicted motor cortex activity for all rats, plotted separately for Early, Mid and Late training days. Red lines denote the median of those distributions. For comparison, the light gray color shows the distribution for neuron-shuffled data, representing chance level predictions. (**D**) The median distance between actual and predicted activity for each rat across the learning phases of reaching task. The V-shape of those plots shows that surprisingly the best predictions of neuronal activity from videos were obtained at Mid days of training.

## 3.2 DERIVING "RECEPTIVE FIELD" OF NEURONS BY USING AN EXPLAIN MODEL

Having a model predicting neuronal activity from animal behavior can also be used to study how the activity of the motor neurons are related to the movement components of skilled reaching. Specifically, we can apply "explain" methods to our model to find which parts of video frames were the most important for predicting the activity of each neuron. This is similar to deriving a receptive field of a neuron. However, our approach allows for finding highly non-linear relations between video frames and neuronal activity. Here we used the LIME explain method (Ribeiro et al. 2016). The basic idea of the LIME method is that it slightly perturbs the image, and it quantifies the impact of that alternation on the accuracy of the predicted neuronal activity (Methods). This shows how important that specific part of the image was for the prediction. Sample receptive fields obtained with the explain method for 3 representative neurons are shown in Figure 3A. To check the reproducibility of receptive fields, we used cross-validation where we trained our model on different subsets of trials, and applied the explain method to different trials not included in the training set. Sample explain values (receptive fields) obtained for 3 different trials are shown in Figure 3A (columns). To quantify the similarity of receptive fields across trials for each neuron we calculated the Euclidian distance between explain values from different trials. To estimate a chance level similarity, we calculated the Euclidian distance between explain values from the same trial but from different neurons (Figure 3B). We found that the distribution of within-neuron distances was significantly smaller than that of between-neurons (Wilcoxon rank-sum test, $p < 0.0001$, $N_{Within} = 720$, $N_{Between} = 5600$). This

shows that the explain model provides a reliable method for identifying non-linear receptive fields, which are robust and distinctive for each neuron.

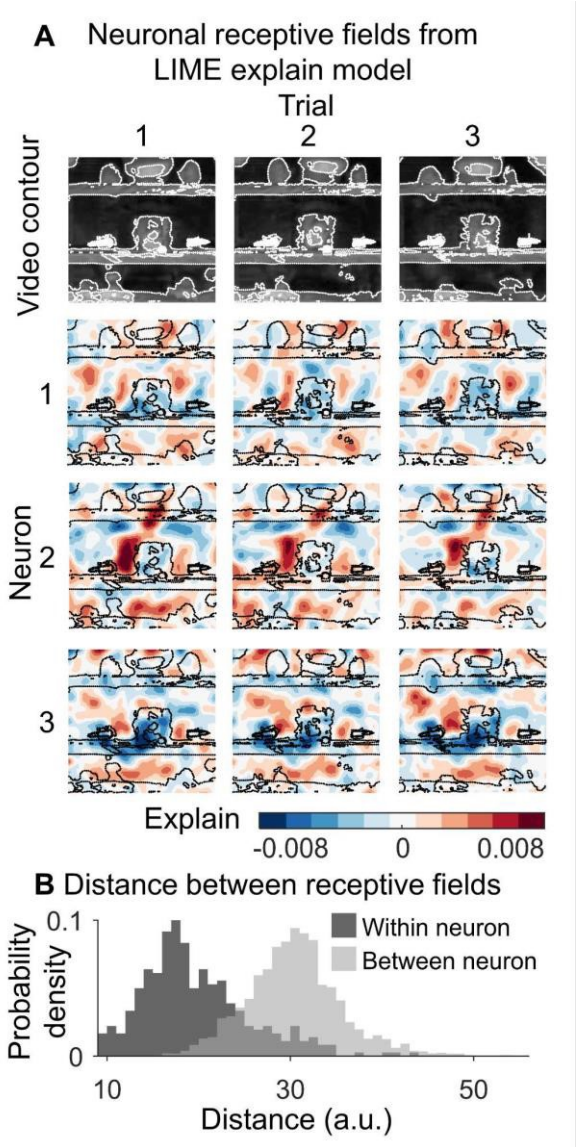



**Figure 3.** Estimating receptive fields of neurons by using an explain model. (**A**) Top row; video frames with superimposed contours of a rat during the start of reaching movement on 3 different trials (columns). Bottom: The same contours as above but color-coded based on which parts of the frame were the most informative for ConvNet to predict

the activity of 3 representative neurons. This is analogous to neuronal receptive field mapping but capable of accounting for non-linear relations. Higher explain values indicate parts of the frame that are the most related to predicting the high firing rate of a neuron. (**B**) Distribution of distances between receptive fields for the same neuron across trials ("within"), and between different neurons ("between"). Note that smaller "within" neuron distance indicates that the receptive field for each neuron is unique and reproducible across trials.

### 3.3 RECONSTRUCTING BEHAVIORAL VIDEOS FROM NEURAL ACTIVITY

As a proof of concept, we also demonstrate how videos of a behaving animal can be reconstructed from neuronal activity. Recent advancements in AI allow users to generate images based on text prompts (Mirza and Osindero 2014). Here, we are using a similar approach, where instead of text, a neuronal activity is given as a "prompt" to generate a video frame. A conditional Generative Adversarial Network (GAN) was trained to reproduce video frames from smoothed neuronal activity (Figure 4A, Methods). Sample frames from original and generated videos are shown in Figure 4B (full videos are included in Supplemental Materials: S1). Interestingly, we found that we could generate good-quality videos of skilled reaching based on the neural activity of only a few neurons which had activity most predictable from videos (see analyses in Figure 2). For computational efficiency, all videos were generated using six neurons, selected for their strong predictability from video features (Methods). This number was chosen because across all rats and training days, the minimum number of neurons with firing rates above 1 Hz was six, ensuring comparable GAN predictions.

To quantify the accuracy of reconstructed movements, we compared trajectories of selected body parts in videos. For that, we used the DeepLabCut neural network (Mathis et al., 2018), which was trained to track the position of the hands (forepaws) and nose (Suppl. Figure 1; Methods). The sample trajectory of the right paw is superimposed on Figure 4B, and it is illustrated in more detail in Figure 4C which shows how the x and y position of the right hand changed in time in the actual and generated videos. To quantify similarity between actual and generated trajectories we calculated the Euclidean distance between trajectories (Figure 4D). For comparison, we also calculated distance between actual and generated trajectories shuffled in time. This allowed us to estimate what would be expected for the distribution of distances if our model would only learn to reproduce images but without learning how frames are related to neuronal activity. We found that actual and generated trajectories were significantly more similar to each other as compared to shuffled data, and this effect was consistent across training phases (Figure 4D, Wilcoxon signed-rank test, Early/Mid/Late all $p < 0.0001$). Consistently with analyses in Fig 2D we also found that for all rats the distance values were the lowest during the Mid training days (Figure 4E, Friedman's test, $\chi^2 = 6$, $p = 0.0498$), indicating the strongest relation between behavior and neuronal activity patterns during that learning phase.

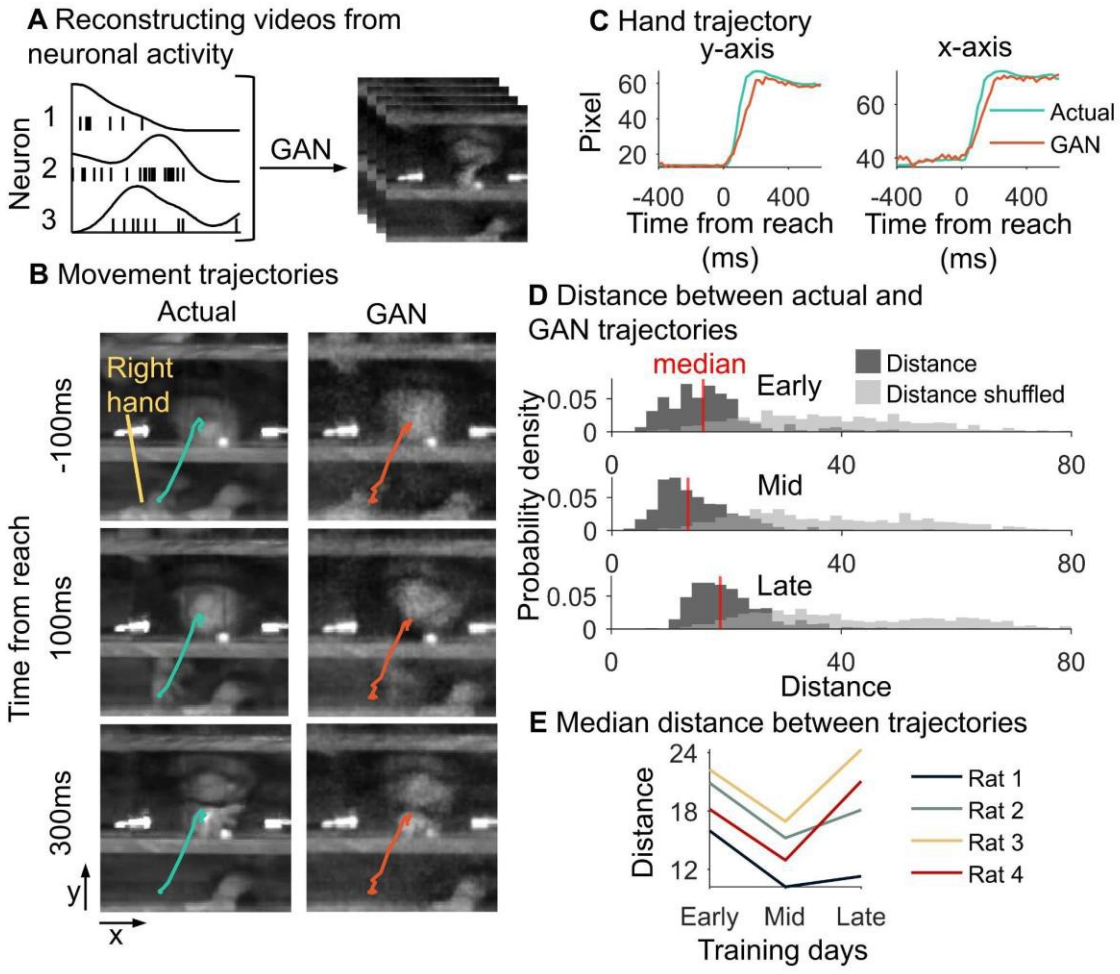



**Figure 4.** Reconstruction of videos from neural activity. (**A**) Illustration of the method: Generative Adversarial Network (GAN) uses smoothed neuronal activity as a conditioning signal to generate video frames. (**B**) Sample frames from the actual and GAN-generated videos. Green and red lines indicate the trajectory of the right hand in actual and generated videos, respectively. (**C**) Illustration of similarity of movement trajectories over time in actual and generated videos. The left panel shows the x-coordinate, and the right panel the y-coordinate of the reaching hand trajectory across time. (**D**) Distribution of distances between actual and GAN-generated reaching-hand positions across all trials and rats, calculated for each training phase. For comparison, the distance between trajectories

expected by chance is calculated by shuffling the order of frames in GAN videos. (**E**) The median distance between actual and GAN generated hand position for each animal across learning phases of reaching task. Consistently with Figure 2D the V-shape of those plots shows that the strongest relation between neuronal activity and behavior in videos was obtained for Mid days of training.

## CHAPTER 4: DISCUSSION

In this study, we leveraged Conditional Generative Adversarial Networks (GAN) and Convolutional Neural Networks (ConvNet) to explore the intricate relationship between motor cortical activity and behavior in rats performing the single-pellet reaching task. The application of these advanced machine learning techniques allowed us to reconstruct behavioral videos from neural activity and predict neuronal firing patterns from raw behavioral videos, and so reveal insights into the dynamics of skilled reach learning.

Our findings demonstrate that the predictability of neuronal activity from behavioral data and vice versa is highest during the middle phase of training. This suggests a non-linear relationship between learning stages and neural-behavioral coupling (Toni et al, 1998). Initially, as the rats learn the task, the coupling between neuronal activity and behavior strengthens, likely due to the increased demand for precise motor control and coordination. However, once the rats achieve proficiency, this coupling decreases even though task performance remains stable. This decoupling may indicate a shift in the neural resources or strategies employed by the motor cortex (Suppl. Figure 2), possibly reflecting a transition from a highly engaged learning state to a more efficient, automated execution state (Hwang et al. 2019).

Microelectrode electrical stimulation of the neocortex shows that the laboratory rat has a large region of frontal neocortex that can be defined as motor cortex, and within this region the forelimb area comprises its largest part (Hall and Lindholm, 1974; Kleim et al; 1998; Neafsey et al, 1984). The neurons in layer V of the rat forelimb area of motor cortex project to the spinal cortex and synapse with premotor neurons but not with motor neurons as occurs in anthropoid primates (Yang and Lemon, 2003). Nevertheless, the rat uses its

forelimb for a wide variety of skilled motor behaviors including reaching for food items with a hand for placement in the mouth for eating and these reaching movements are motorically similar to those used by anthropoid primates including humans (Whishaw et al, 1992; Sacrey et al, 2009). Both electrophysiological and anatomical evidence suggests that motor cortex is involved in both the learning and performance of skilled reaching (Peters et al, 2017; Tennant et al, 2012). To further investigate the contribution of the motor cortex neurons to skilled reaching for food, we employed new ML tools capable of complex input - output mappings that are useful for studying highly non-linear relation between neuronal activity and behavior.

The use of GANs in this context is particularly noteworthy as it marks the first instance of behavioral videos being reconstructed from motor cortical activity. This innovative application of GANs, typically used in image synthesis and enhancement, underscores their potential for generating contextually relevant synthetic outputs based on neural data. The ability to create accurate behavioral reconstructions from sparse neural recordings opens new avenues for understanding and interpreting motor cortex function.

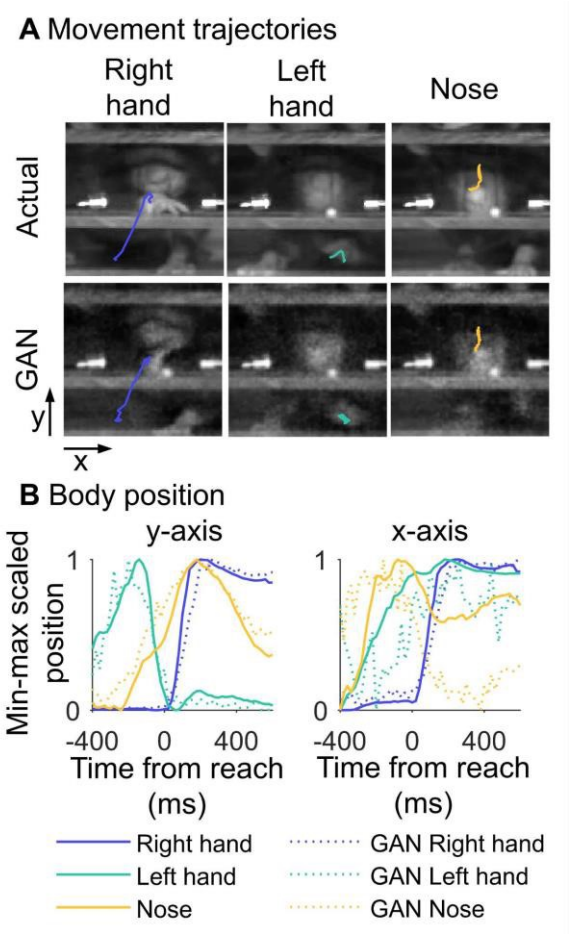
Moreover, the implementation of the LIME model with ConvNet predictions provided a novel approach to identifying the motor cortex's behavioral receptive fields. Traditional methods often align spikes with specific stimuli or behaviors (Hubel and Wiesel, 1959; Alonso and Chen 2008; Luczak et al. 2004), which can overlook the broader context of neural activity. By contrast, our approach, which does not presuppose specific relationships, offers a more comprehensive view of how neuronal activity relates to behavior. This method could significantly enhance our understanding of the neural mechanisms underlying motor control and learning.

These insights have implications for the development of brain-machine interfaces (BMIs) aimed at restoring movement in individuals with motor impairments. Our findings challenge the assumption that motor cortical signals remain stable over extended periods, highlighting the need for adaptive BMI systems that can accommodate changes in neural-behavioral coupling over time (Toni et al, 1998).

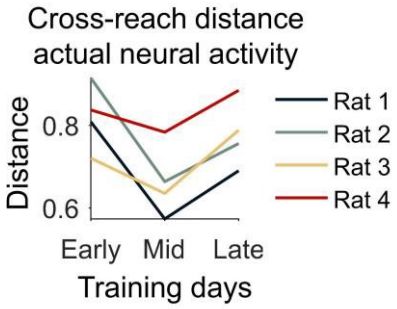
In summary, this study demonstrates the utility of advanced machine learning techniques in unraveling the complex interplay between neuronal activity and behavior. By employing GANs and ConvNets, we have provided new perspectives on motor learning and control, paving the way for future research in neuroprosthetics and neural rehabilitation.

## CHAPTER 5: SUPPLEMENTARY MATERIALS

Behavior video reconstructions from neural activity accurately capture movements of the right hand, left hand, and nose, as evidenced by their average trajectories (Suppl. Figure 1). Analysis of these trajectories over time reveals distinct patterns: the rat first sniffs the food at a lower position (yellow line), then raises its nose to reach for it. Before this, it frequently repositions its non-dominant hand (green line), and finally, extends its dominant hand to grasp the food (blue line).



**Suppl. Figure 1**. A) Rat limb tracking in both actual and GAN-generated videos reveals consistent movements of the right hand, left hand, and nose. B) The mean trajectory over time shows a consistent sequence of limb movements for some rats, progressing from nose sniffing to left-hand repositioning, and finally to right-hand reaching.




**Suppl. Figure 2**. Cross-reach distance of actual neural activity. This figure shows that neuronal activity patterns in the motor cortex were the most similar to each other during Mid training days. This is consistent with the results presented in Figures 2 & 4. In other words, we found that the distance between neural activities across trials on a single day was also the smallest at Mid training phase.

## REFERENCES


Alaverdashvili, M., & Whishaw, I. Q. (2013). A behavioral method for identifying recovery and compensation: hand use in a preclinical stroke model using the single pellet reaching task. Neuroscience and biobehavioral reviews, 37(5), 950–967.

Alonso, J.-M. & Chen, Y. (2008). Receptive field. Scholarpedia. 4 (1): 5393.

Alzubaidi, L., Zhang, J., Humaidi, A. J., Al-Dujaili, A., Duan, Y., Al-Shamma, O., Santamaría, J., Fadhel, M. A., Al-Amidie, M., & Farhan, L. (2021). Review of deep learning: concepts, CNN architectures, challenges, applications, future directions. Journal of big data, 8(1), 53.

Baldwin, M. K., Cooke, D. F., & Krubitzer, L. (2017). Intracortical Microstimulation Maps of Motor, Somatosensory, and Posterior Parietal Cortex in Tree Shrews (Tupaia belangeri) Reveal Complex Movement Representations. Cerebral cortex (New York, N.Y. : 1991), 27(2), 1439–1456.

Brown, A. R., & Teskey, G. C. (2014). Motor cortex is functionally organized as a set of spatially distinct representations for complex movements. The Journal of neuroscience : the official journal of the Society for Neuroscience, 34(41), 13574–13585.

Cisek, P., & Kalaska, J. F. (2010). Neural mechanisms for interacting with a world full of action choices. Annual review of neuroscience, 33, 269–298.

Dash, A., Ye, J., & Wang, G. (2023). A review of Generative Adversarial Networks (GANs) and its applications in a wide variety of disciplines: From Medical to Remote Sensing. IEEE Access.

Eckert, M. J., McNaughton, B. L., & Tatsuno, M. (2020). Neural ensemble reactivation in rapid eye movement and slow-wave sleep coordinate with muscle activity to promote rapid motor skill learning. Philosophical transactions of the Royal Society of London. Series B, Biological sciences, 375(1799), 20190655.

Fabbri-Destro, M., & Rizzolatti, G. (2008). Mirror neurons and mirror systems in monkeys and humans. Physiology (Bethesda, Md.), 23, 171–179.

Georgopoulos, A. P., Kettner, R. E., & Schwartz, A. B. (1988). Primate motor cortex and free arm movements to visual targets in three-dimensional space. II. Coding of the direction of movement by a neuronal population. The Journal of neuroscience : the official journal of the Society for Neuroscience, 8(8), 2928–2937.

Goodfellow, I., Pouget-Abadie, J., Mirza, M., Xu, B., Warde-Farley, D., Ozair, S., ... & Bengio, Y. (2014). Generative adversarial nets. Advances in neural information processing systems, 27.

Gothard, K. M., Skaggs, W. E., Moore, K. M., & McNaughton, B. L. (1996). Binding of hippocampal CA1 neural activity to multiple reference frames in a landmark-based

navigation task. The Journal of neuroscience : the official journal of the Society for Neuroscience, 16(2), 823–835.

Graziano, M. S., Taylor, C. S., & Moore, T. (2002). Complex movements evoked by microstimulation of precentral cortex. Neuron, 34(5), 841–851.

Hall, R.D., & Lindholm, E. (1974). Organization of motor and somatosensory neocortex in the albino rat. Brain Research, 66, 23-38.

Hatsopoulos, N. G., Xu, Q., & Amit, Y. (2007). Encoding of movement fragments in the motor cortex. The Journal of neuroscience : the official journal of the Society for Neuroscience, 27(19), 5105–5114.

Hubel, D. H., & Wiesel, T. N. (1959). Receptive fields of single neurones in the cat's striate cortex. The Journal of physiology, 148(3), 574–591.

Hwang, E. J., Dahlen, J. E., Hu, Y. Y., Aguilar, K., Yu, B., Mukundan, M., Mitani, A., & Komiyama, T. (2019). Disengagement of motor cortex from movement control during long-term learning. Science advances, 5(10), eaay0001.

Kleim, J. A., Barbay, S., & Nudo, R. J. (1998). Functional reorganization of the rat motor cortex following motor skill learning. Journal of neurophysiology, 80(6), 3321–3325.

Klein, A., & Dunnett, S. B. (2012). Analysis of skilled forelimb movement in rats: the single pellet reaching test and staircase test. Current protocols in neuroscience, Chapter 8, Unit8.28.

Li, Z., Liu, F., Yang, W., Peng, S., & Zhou, J. (2022). A Survey of Convolutional Neural Networks: Analysis, Applications, and Prospects. IEEE transactions on neural networks and learning systems, 33(12), 6999–7019.

Luczak, A., Hackett, T. A., Kajikawa, Y., & Laubach, M. (2004). Multivariate receptive field mapping in marmoset auditory cortex. Journal of neuroscience methods, 136(1), 77-85.

Mathis, A., Mamidanna, P., Cury, K. M., Abe, T., Murthy, V. N., Mathis, M. W., & Bethge, M. (2018). DeepLabCut: markerless pose estimation of user-defined body parts with deep learning. Nature neuroscience, 21(9), 1281–1289.

Mirza, M., & Osindero, S. (2014). Conditional generative adversarial nets. arXiv preprint arXiv:1411.1784.

Neafsey, E. J., Bold, E. L., Haas, G., Hurley-Gius, K. M., Quirk, G., Sievert, C. F., & Terreberry, R. R. (1986). The organization of the rat motor cortex: a microstimulation mapping study. Brain research, 396(1), 77–96.

Penfield, W., & Rasmussen, T. (1950). The cerebral cortex of man; a clinical study of localization of function.

Peters, A. J., Liu, H., & Komiyama, T. (2017). Learning in the Rodent Motor Cortex. Annual review of neuroscience, 40, 77–97.

Purves, D., Augustine, G. J., Fitzpatrick, D., Hall, W. C., LaMantia, A.-S., Mooney, R. D., Platt, M. L., & White, L. E. (2017). Neuroscience (6th ed.). Oxford University Press.

Ribeiro, M. T., Singh, S., & Guestrin, C. (2016, August). " Why should i trust you?" Explaining the predictions of any classifier. In Proceedings of the 22nd ACM SIGKDD international conference on knowledge discovery and data mining (pp. 1135-1144).

Rossant, C., Kadir, S. N., Goodman, D. F. M., Schulman, J., Hunter, M. L. D., Saleem, A. B., Grosmark, A., Belluscio, M., Denfield, G. H., Ecker, A. S., Tolias, A. S., Solomon, S., Buzsaki, G., Carandini, M., & Harris, K. D. (2016). Spike sorting for large, dense electrode arrays. Nature neuroscience, 19(4), 634–641.

Ryait, H., Bermudez-Contreras, E., Harvey, M., Faraji, J., Mirza Agha, B., Gomez-Palacio Schjetnan, A., Gruber, A., Doan, J., Mohajerani, M., Metz, G. A. S., Whishaw, I. Q., & Luczak, A. (2019). Data-driven analyses of motor impairments in animal models of neurological disorders. PLoS biology, 17(11), e3000516.

Sacrey, L. A., Alaverdashvili, M., & Whishaw, I. Q. (2009). Similar hand shaping in reaching-for-food (skilled reaching) in rats and humans provides evidence of homology in release, collection, and manipulation movements. Behavioural brain research, 204(1), 153–161.

Saleh, M., Takahashi, K., Amit, Y., & Hatsopoulos, N. G. (2010). Encoding of coordinated grasp trajectories in primary motor cortex. The Journal of neuroscience : the official journal of the Society for Neuroscience, 30(50), 17079–17090.

Scott S. H. (2016). A Functional Taxonomy of Bottom-Up Sensory Feedback Processing for Motor Actions. Trends in neurosciences, 39(8), 512–526.

Szegedy, C., Vanhoucke, V., Ioffe, S., Shlens, J., & Wojna, Z. (2016). Rethinking the inception architecture for computer vision. In Proceedings of the IEEE conference on computer vision and pattern recognition (pp. 2818-2826).

Tennant, K. A., Adkins, D. L., Scalco, M. D., Donlan, N. A., Asay, A. L., Thomas, N., Kleim, J. A., & Jones, T. A. (2012). Skill learning induced plasticity of motor cortical representations is time and age-dependent. Neurobiology of learning and memory, 98(3), 291–302.

Toni, I., Krams, M., Turner, R., & Passingham, R. E. (1998). The time course of changes during motor sequence learning: a whole-brain fMRI study. NeuroImage, 8(1), 50–61.

Torabi, R., Jenkins, S., Harker, A., Whishaw, I. Q., Gibb, R., & Luczak, A. (2021). A Neural Network Reveals Motoric Effects of Maternal Preconception Exposure to Nicotine on Rat Pup Behavior: A New Approach for Movement Disorders Diagnosis. Frontiers in neuroscience, 15, 686767.

Whishaw, I. Q., & Pellis, S. M. (1990). The structure of skilled forelimb reaching in the rat: a proximally driven movement with a single distal rotatory component. Behavioural brain research, 41(1), 49–59.

Whishaw, I. Q., Pellis, S. M., & Gorny, B. P. (1992). Skilled reaching in rats and humans: evidence for parallel development or homology. Behavioural brain research, 47(1), 59–70.

Yamins, D. L., Hong, H., Cadieu, C. F., Solomon, E. A., Seibert, D., & DiCarlo, J. J. (2014). Performance-optimized hierarchical models predict neural responses in higher visual cortex. Proceedings of the National Academy of Sciences of the United States of America, 111(23), 8619–8624.

Yang, H. W., & Lemon, R. N. (2003). An electron microscopic examination of the corticospinal projection to the cervical spinal cord in the rat: lack of evidence for cortico-motoneuronal synapses. Experimental brain research, 149(4), 458–469.

## APPENDIX 1: CONVNET_P1: EXAMPLE OF EXTRACTING FEATURE WITH CONVNET FROM REACH TRIAL VIDEO

```
#%% ConvNet P1: V3 CNN

%reset -f

import os
import numpy as np
import matplotlib.pyplot as plt
%matplotlib inline
import scipy.io as sio
import cv2
from tensorflow.keras.applications.inception_v3 import InceptionV3,
preprocess_input

load_path =
'D:\\20220624_Lethbridge\\20230111_rat_unit_EckertTatsuno\\20240324_sample_cod
e_dt\\rr5\\input_convnet_p1'
save_path =
'D:\\20220624_Lethbridge\\20230111_rat_unit_EckertTatsuno\\20240324_sample_cod
e_dt\\rr5\\output_convnet_p1'


rr_day = [[5,7]] # [rr,day]

frame_diff = 0 # # frames

for rr_day_i in np.arange(len(rr_day)):
    # rr_day_i = 0
    rr_i = rr_day[rr_day_i][0]
    day_i = rr_day[rr_day_i][1]

    os.chdir(load_path)
    tr_adv_ms = np.load('day' + str(day_i) + '_adv_ms_noHandPin.npy')
    N_reach   = np.size(tr_adv_ms)

    base_model = InceptionV3(weights='imagenet', include_top=False, pooling='avg')

    #for i in range(0,N_reach):
    for i in [0]: #range(0,N_reach):
        # i = 0
        print('rr ' + str(rr_i) + ' day ' + str(day_i) + ' reach ' + str(i))
```

```
        frame_reach_vid = np.load("day" + str(day_i) + "_frame_reach_vid_zoom_reach"
+ str(i) + ".npy")
        if frame_diff > 0:
            frame_reach_vid = frame_reach_vid[:,:,frame_diff:] - frame_reach_vid[:,:,:-
frame_diff]
            frame_reach_vid = cv2.normalize(frame_reach_vid, None, alpha = 0, beta =
255, norm_type = cv2.NORM_MINMAX, dtype = cv2.CV_32F)
            frame_reach_vid = frame_reach_vid.astype(np.uint16)
        # plt.figure(); plt.imshow(frame_reach_vid[:,:,300], cmap='gray', vmin=100,
vmax=200); plt.show()

        frame_reach_vid_pre = np.transpose(np.tile(frame_reach_vid, (3,1,1,1)), (3, 1, 2,
0))
        frame_reach_vid_prepro = preprocess_input(frame_reach_vid_pre)
        frame_reach_vid_prepro_i = np.transpose(frame_reach_vid_prepro[:,:,:,0], (1, 2,
0))
        # frame_vid_i_plt = frame_reach_vid_prepro[0,:,:,0]
        # plt.imshow(frame_vid_i_plt, cmap='gray')
        features_vid = base_model.predict(frame_reach_vid_prepro)
        os.chdir(save_path)
        np.save(("day" + str(day_i) + "_frame_reach_vid_zoom_reach" + str(i) +
"_prepro.npy"),frame_reach_vid_prepro_i)
        np.save(("day" + str(day_i) + "_frame_reach_vid_zoom_reach" + str(i) +
"_cnn.npy"),features_vid)
```

## APPENDIX 2: CONVNET_P2: EXAMPLE OF TRAINING DENSE UNIT TO PREDICT NEURAL ACTIVITY FROM CONVNET FEATURES

```
#%% ConvNet P2: train dense, leave-one-out on 10 reach, plot top predicted neural activity

%reset -f

import os
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
%matplotlib qt
import scipy.io as sio
# import cv2
import tensorflow as tf
from tensorflow.keras.models import Sequential, Model
from tensorflow.keras.layers import Dense, InputLayer, Dropout, Flatten, Activation, Input, Lambda, MaxPooling2D, Flatten
from tensorflow.keras.layers import LSTM, Bidirectional, Input, Conv2D, Reshape, RepeatVector, Permute, GRU, SimpleRNN, BatchNormalization
from tensorflow.keras.applications.inception_v3 import InceptionV3, preprocess_input
from tensorflow.keras import optimizers
from tensorflow.keras import backend
from sklearn.model_selection import train_test_split, GroupKFold, StratifiedKFold
from sklearn.utils import shuffle
from sklearn.metrics import classification_report
from keras import backend as K

from sklearn.metrics import r2_score

load_path = 'D:\\20220624_Lethbridge\\20230111_rat_unit_EckertTatsuno\\20240324_sample_code_dt\\rr5\\input_convnet_p2'
save_path = 'D:\\20220624_Lethbridge\\20230111_rat_unit_EckertTatsuno\\20240324_sample_code_dt\\rr5\\output_convnet_p2'

rr_day = [[5,7]] # [rr,day]

N_leaveOne = 10

grid_shift_iPR_vid = [0] #np.arange(-1000,1000+1,50,dtype=int) # ms
grid_iPRDur   = [1000] # ms
grid_iPRIncr  = [50] # ms
```

```
grid_VidDur    = [5] # ms
grid_VidIncr   = [15] # ms
grid_zoom      = [1]
N_lp =
len(grid_shift_iPR_vid)*len(grid_iPRDur)*len(grid_iPRIncr)*len(grid_VidDur)*len(
grid_VidIncr)*len(grid_zoom)
param_lp = np.zeros((N_lp,6))
count = 0
for i in grid_iPRDur:
   for ii in grid_iPRIncr:
      for iii in grid_VidDur:
         for iiii in grid_VidIncr:
            for iiiii in grid_shift_iPR_vid:
               for iiiiii in grid_zoom:
                     param_lp[count,:] = [i, ii, iii, iiii, iiiii, iiiiii]
                     count += 1

for rr_day_i in np.arange(len(rr_day)):
   # rr_day_i = 1
   rr_i = rr_day[rr_day_i][0]
   day_i = rr_day[rr_day_i][1]

   for vid_i in np.arange(N_lp):
      # vid_i = 0
      iPRDur    = int(param_lp[vid_i,0])
      iPRIncr   = int(param_lp[vid_i,1])
      VidDur    = int(param_lp[vid_i,2])
      VidIncr   = int(param_lp[vid_i,3])
      shift_i   = int(param_lp[vid_i,4])
      zoom_i    = int(param_lp[vid_i,5])

      if zoom_i == 1:
         zoom_str = 'zoom_'
      else:
         zoom_str = ''

      os.chdir(load_path)
      spk_count = sio.loadmat('rat' + str(rr_i) + '_day' + str(day_i) +
'_iPR_incr_neuron_iPRDur' + str(iPRDur) + 'ms_iPRIncr' + str(iPRIncr) +
'ms.mat')['iPR_tr_stack_incr']

      N_reach     = np.size(spk_count,1)
      N_neuron_all = np.size(spk_count,0)
      N_incr      = np.size(spk_count,2)
      N_sample    = N_reach*N_incr
```

```
    for i in np.arange(N_reach):
        # i = 0
        print('load reach ' + str(i))
        if shift_i < 0:
            str_n = 'n';
        else:
            str_n = '';
        tmp_fr = sio.loadmat('rat' + str(rr_i) + '_day' + str(day_i) + '_vid_' + zoom_str
+ 'incr_reach_' + str(i) + '_iPRDur' + str(iPRDur) + 'ms_iPRIncr' + str(iPRIncr) +
'ms_' +
                'shiftiPRvid' + str_n + str(abs(shift_i)) + 'ms_cnn.mat')['frame_reach_i']
        if i == 0:
            N_feature = np.size(tmp_fr,2)
            N_frame   = np.size(tmp_fr,1)
            vid_frames_all = np.zeros((N_reach, N_incr, N_frame, N_feature))
        vid_frames_all[i,:,:,:] = tmp_fr


    reach_leaveOne = np.arange(0,N_reach, np.floor(N_reach/N_leaveOne))
    reach_leaveOne = reach_leaveOne[0:10].astype(int)
    groups_reach  = np.zeros((N_reach,1))
    count = 0
    for i_lo in np.arange(1,N_leaveOne+1):
        groups_reach[reach_leaveOne[i_lo-1],:] = i_lo

    os.chdir(save_path)
    np.save(('rat' + str(rr_i) + '_day' + str(day_i) + '_group_reach.npy'),groups_reach)
    r2_neuron = np.zeros((N_neuron_all))
    for i_neur in np.arange(N_neuron_all):
        # i_neur = 3
        spk_count_i = np.expand_dims(spk_count[i_neur,:,:], axis = 0)
        N_neuron = np.size(spk_count_i,0)

        vid_frames_sample = np.zeros((N_sample, N_frame, N_feature))
        spk_count_sample  = np.zeros((N_sample, N_neuron))
        groups_sample     = np.zeros((N_sample,1))
        for i in np.arange(N_reach):
            print('rr ' + str(rr_i) + ' day ' + str(day_i) + ' prepro reach ' + str(i))
            tmp_ind = np.arange((i*N_incr),(i*N_incr+N_incr))
            groups_sample[tmp_ind,0] = np.full(N_incr, groups_reach[i])
            vid_frames_sample[tmp_ind,:,:] = vid_frames_all[i,:,:,:]
            spk_count_sample[tmp_ind,:] = spk_count_i[:,i,:].T

        X = vid_frames_sample
        y = spk_count_sample
        del vid_frames_sample, spk_count_sample
```

```
def coeff_det(y_true, y_pred):
    SS_res =  K.sum(K.square( y_true-y_pred ))
    SS_tot = K.sum(K.square( y_true - K.mean(y_true) ) )
    return ( 1 - SS_res/(SS_tot + K.epsilon()) )

def coeff_det_neurons(y_true, y_pred):
    cd_neur = np.zeros(N_neuron)
    for i in np.arange(N_neuron):
        # i = 0
        SS_res =  K.sum(K.square( y_true[:,i]-y_pred[:,i] ))
        SS_tot = K.sum(K.square( y_true[:,i] - K.mean(K.constant(y_true[:,i]) ) ))
        cd_neur[i] = ( 1 - tf.cast(SS_res, tf.float32)/(SS_tot + K.epsilon()) )
    return cd_neur

class CustomCallback(tf.keras.callbacks.Callback):
    def on_epoch_begin(self, epoch, logs=None):
        print("rr {} day {} loop {} of {} fold {} neuron {} shift {} iPRDur {}
iPRIncr {} VidDur {} VidIncr {} zoom {}".format(rr_i, day_i, vid_i, N_lp, group_i,
i_neur, shift_i, iPRDur, iPRIncr, VidDur, VidIncr, zoom_i))
    def on_epoch_end(self, epoch, logs=None):
        y_pred = model.predict(X_test)
        r2_epoch_neurons[:,epoch] = coeff_det_neurons(y_test, y_pred)



for group_i in np.arange(1,N_leaveOne+1):
    # group_i = 2
    test_index  = np.where(groups_sample == group_i)[0]
    train_index = np.where(groups_sample != group_i)[0]
    X_train, X_test = X[train_index,:,:], X[test_index,:,:]
    y_train, y_test = y[train_index,:],     y[test_index,:]

    backend.clear_session()
    model = Sequential()

    # model.add(Dense(128, activation='relu'))
    # model.add(Dense(16, activation='relu'))
    model.add(Dense(N_neuron))
    model.add(Activation('sigmoid'))
    optimizers.Adam(learning_rate=0.001) #01)
    model.compile(loss='MSE',optimizer='Adam', metrics=[coeff_det])
    # model.summary()

    epochs = 400
    batch_size= 128 # 128 # 256 #512 #1024
```

```
            r2_epoch_neurons = np.zeros((N_neuron,epochs))
            history = model.fit(X_train, y_train, epochs=epochs,
validation_data=(X_test, y_test), batch_size=batch_size,
callbacks=[CustomCallback()]) # batch_size=1024, callbacks=[CustomCallback()])
            y_pred = model.predict(X_test)


            if group_i == 1:
               y_pred_lo = y_pred
               y_test_lo = y_test
            else:
               y_pred_lo = np.concatenate([y_pred_lo, y_pred])
               y_test_lo = np.concatenate([y_test_lo, y_test])

            history_cat = [np.asarray(history.history['loss']),
                  np.asarray(history.history['coeff_det']),
                  np.asarray(history.history['val_loss']),
                  np.asarray(history.history['val_coeff_det'])]

            # plt.figure(figsize=(2.7,2.5))
            # plt.plot(history.history['val_coeff_det'])
            # plt.ylim(0, 1)
            # plt.xlim(0, epochs)
            # plt.grid()

            if shift_i < 0:
               str_n = 'n';
            else:
               str_n = '';

            save_suffix = ('rr' + str(rr_i) + '_day' + str(day_i) + '_neur' + str(i_neur)
                  + '_shift' + str_n + str(np.abs(shift_i)) + 'ms'
                  + '_iPRDur' + str(iPRDur) + 'ms_iPRIncr' + str(iPRIncr) +
'ms_VidDur' + str(VidDur)
                  + 'ms_VidIncr' + str(VidIncr) + 'ms_fold' + str(group_i) + '_zoom' +
str(zoom_i) + '_group' + str(group_i))
            np.save((save_path + '\\predict_' + save_suffix + '.npy'),y_pred)
            model.save(save_path + '\\model_' + save_suffix + '.h5') #, save_format='h5')

         r2_neuron[i_neur] = r2_score(np.squeeze(y_test_lo), np.squeeze(y_pred_lo))
         np.save((save_path + '\\' + 'neur' + str(i_neur)  + '_y_test_ol.npy'),y_test_lo)
         np.save((save_path + '\\' + 'neur' + str(i_neur)  + '_y_pred_ol.npy'),y_pred_lo)

np.save((save_path + '\\r2_neuron.npy'),r2_neuron)

neuron_ind_top6 = np.flip(np.argsort(r2_neuron)[-6:])
```

```
###
### plotting 1 reach's neural activity prediction of best predicted neuron
t_vec = np.arange(-400,600+1,50)
i_neur = neuron_ind_top6[0]
y_test_lo = np.load(save_path + '\\' + 'neur' + str(i_neur)  + '_y_test_ol.npy')
y_pred_lo = np.load(save_path + '\\' + 'neur' + str(i_neur)  + '_y_pred_ol.npy')
reach_i = 2
plt.plot(t_vec,np.squeeze(y_test_lo)[reach_i*21:(reach_i+1)*21], color="black")
plt.plot(t_vec,np.squeeze(y_pred_lo)[reach_i*21:(reach_i+1)*21], color="red")
plt.ylabel("neural activity")
plt.xlabel("time from reach (ms)")
plt.margins(x=0)
plt.tight_layout()

###
### plotting 10 leave-one out reach, for top 6 predicted neurons
count_i = 0
for i_neur in neuron_ind_top6:
    count_i += 1
    axes = plt.subplot(3, 2, count_i)
    y_test_lo = np.load(save_path + '\\' + 'neur' + str(i_neur)  + '_y_test_ol.npy')
    y_pred_lo = np.load(save_path + '\\' + 'neur' + str(i_neur)  + '_y_pred_ol.npy')
    plt.plot(np.squeeze(y_test_lo), color="black")
    plt.plot(np.squeeze(y_pred_lo), color="red")
    plt.margins(x=0)
    plt.tight_layout()
    axes.title.set_text('$R^{2}$ =' + str(round(r2_neuron[i_neur],2)) + ' (neuron ' +
str(i_neur) + ')' )
    if count_i == 5:
        axes.set_ylabel("neural activity")
        axes.set_xlabel("time unit")
```

## APPENDIX 3: GAN_P1: EXAMPLE OF TRAINING CONDITIONAL GAN TO GENERATE RAT VIDEO WITH CONDITION OF 6 NEURON'S ACTIVITY (TOP R2 FROM CONVNET ANALYSIS)

```
#%% GAN P1: conditional GAN and prediction

%reset -f

from skimage.measure import block_reduce
import numpy as np
from numpy import zeros
from numpy import ones
from numpy.random import randn
from numpy.random import randint
from tensorflow.keras.optimizers import Adam
# from tensorflow.keras.optimizers.legacy import Adam
from tensorflow.keras.models import Model
from tensorflow.keras.layers import Input
from tensorflow.keras.layers import Dense
from tensorflow.keras.layers import Reshape
from tensorflow.keras.layers import Flatten
from tensorflow.keras.layers import Conv2D
from tensorflow.keras.layers import Conv2DTranspose
from tensorflow.keras.layers import LeakyReLU
from tensorflow.keras.layers import Dropout
from tensorflow.keras.layers import Embedding
from tensorflow.keras.layers import Concatenate

from matplotlib import pyplot as plt
import scipy.io as sio
import os
import gc
import time

from tensorflow.keras.models import load_model
import scipy.ndimage as ndimage


load_path =
'D:\\20220624_Lethbridge\\20230111_rat_unit_EckertTatsuno\\20240324_sample_code_dt\\rr5\\input_gan_p1_p2'
save_path =
'D:\\20220624_Lethbridge\\20230111_rat_unit_EckertTatsuno\\20240324_sample_code_dt\\rr5\\output_gan_p1'
```

```
rr_day = [[5,7]] # [rr,day]

t_ms_incr = 20 # ms
t_ms_seg = [-400, 600]

Conv_unit = 128

os.chdir(load_path)
r2_all = sio.loadmat('r2_neurons_all_6N.mat')['r2_allall']
for rr_day_i in np.arange(len(rr_day)):
    # rr_day_i = 0
    rr_i = rr_day[rr_day_i][0]
    day_i = rr_day[rr_day_i][1]
    ind_tmp = np.where((r2_all[3,:] == rr_i) & (r2_all[4,:] == day_i))

    str_save_prefix = ('rat' + str(rr_i) + '_day' + str(day_i) +
'_reach_iPR_conditional_generator')

    r2top_neur = r2_all[2,ind_tmp]
    r2top_neur[0][0] = r2top_neur[0][0][0:6]

    os.chdir(load_path)
    iPR = sio.loadmat('rat' + str(rr_i) + '_day' + str(day_i) +
'_iPR_incr_neuron_iPRDur1000ms_iPRIncr' + str(t_ms_incr) + 'ms.mat')
    iPR_n = iPR["iPR_tr_stack_incr"]
    iPR_n = np.squeeze(iPR_n[r2top_neur[0][0]-1,:,:])
    N_iPR   = np.size(iPR_n,2)
    N_reach = np.size(iPR_n,1)
    N_neur  = np.size(iPR_n,0)

    t_frame = np.load('rat' + str(rr_i) + '_day' + str(day_i) + '_frame_reach_unit.npy')
    t_fpms = np.load('rat' + str(rr_i) + '_day' + str(day_i) + '_frame_reach_fpms.npy')
    ms_per_frame = round(1/t_fpms)
    t_ms = t_frame*ms_per_frame


    count = 0
    iPR_ms  = zeros((N_neur,N_reach*N_iPR))
    for i_reach in np.arange(N_reach):
        # i_reach = 0
        print('load reach ' + str(i_reach))

        iPR_ms[:,((i_reach)*N_iPR):((i_reach+1)*N_iPR)] =
np.squeeze(iPR_n[:,i_reach,:])
```

```
        vid_reach = np.load('rat' + str(rr_i) + '_day' + str(day_i) +
'_frame_reach_vid_zoom_reach' + str(i_reach) + '_prepro.npy')
        N_pix_tmp = 128
        shift_pix_x = 10
        shift_pix_y = 20
        vid_reach =
vid_reach[shift_pix_y:(shift_pix_y+N_pix_tmp),shift_pix_x:(shift_pix_x+N_pix_tmp)
,:]

        N_pix = 128 #64


        tmp_abs = abs(t_ms - t_ms_seg[0])
        ind_start = np.where(tmp_abs == min(tmp_abs))[0]
        tmp_abs = abs(t_ms - t_ms_seg[1])
        ind_end   = np.where(tmp_abs == min(tmp_abs))[0]

        ts_frame = np.arange(ind_start, ind_end+1, t_ms_incr/ms_per_frame,dtype=int)
        for i_ind in np.arange(len(ts_frame)):
            if i_reach == 0 and i_ind == 0:
                ts_ms  = zeros(len(ts_frame)*(N_reach))
                ts_vid = zeros((len(ts_frame)*(N_reach),N_pix,N_pix))
            i_fr = ts_frame[i_ind]
            ts_ms[count]      = t_ms[i_fr]
            ts_vid[count,:,:] = vid_reach[:,:,i_fr]
            count = count + 1

        # plt.imshow(vid_reach[:,:,400])
    if ts_ms.ndim == 1:
        ts_vid = ((ts_vid - ts_vid.min()) * (1/(ts_vid.max() - ts_vid.min()) *
255)).astype('uint8')
        ts_vid = ts_vid[:,:,:,np.newaxis]
        ts_vid = np.repeat(ts_vid,3, axis = 3)

        ts_ms = ((ts_ms - ts_ms.min()) * (1/(ts_ms.max() - ts_ms.min()) *
255)).astype('uint8')
        ts_ms = ts_ms[:,np.newaxis]

        iPR_ms = ((iPR_ms - iPR_ms.min()) * (1/(iPR_ms.max() - iPR_ms.min()) *
255)).astype('uint8')
        iPR_ms = iPR_ms.T #[:,np.newaxis]

    # ts_cond = ts_ms
    ts_cond = iPR_ms
```

```
# for i_n in np.arange(N_neur):
#     iPR_ms[:,i_n] = np.gradient(iPR_ms[:,i_n])

# iPR_tmp = iPR_ms[0:65,0]
# iPR_slope_ms = np.gradient(iPR_tmp)
# plt.plot(iPR_tmp)
# plt.plot(iPR_slope_ms)
# plt.plot(np.zeros(65))
# plt.imshow()

## models


def define_discriminator(in_shape=(N_pix,N_pix,3)):

    # label input
    in_label = Input(shape=(np.size(ts_cond,1),))
    n_nodes = in_shape[0] * in_shape[1]
    li = Dense(n_nodes)(in_label)
    li = Reshape((in_shape[0], in_shape[1], 1))(li)

    # image input
    in_image = Input(shape=in_shape)
    # concat label as a channel
    merge = Concatenate()([in_image, li])

    fe = Conv2D(Conv_unit, (3,3), strides=(2,2), padding='same')(merge)
    fe = LeakyReLU(alpha=0.2)(fe)
    fe = Conv2D(Conv_unit, (3,3), strides=(2,2), padding='same')(fe)
    fe = LeakyReLU(alpha=0.2)(fe)
    fe = Conv2D(Conv_unit, (3,3), strides=(2,2), padding='same')(fe)
    fe = LeakyReLU(alpha=0.2)(fe)
    fe = Conv2D(Conv_unit, (3,3), strides=(2,2), padding='same')(fe)
    fe = LeakyReLU(alpha=0.2)(fe)
    # flatten feature maps
    fe = Flatten()(fe)
    # dropout
    fe = Dropout(0.4)(fe)
    # output
    out_layer = Dense(1, activation='sigmoid')(fe)  #Shape=1

    # define model
    model = Model([in_image, in_label], out_layer)

    opt = Adam(lr=0.0002, beta_1=0.5)
```

```
    model.compile(loss='binary_crossentropy', optimizer=opt, metrics=['accuracy'])
    return model


# define generator model

def define_generator(latent_dim):

    # label input
    in_label = Input(shape=(np.size(ts_cond,1),))

    n_nodes = 8 * 8
    li = Dense(n_nodes)(in_label) #1,64
    li = Reshape((8, 8, 1))(li)


    # image generator input
    in_lat = Input(shape=(latent_dim,))  #Input of dimension 100

    n_nodes = Conv_unit * 8 * 8
    gen = Dense(n_nodes)(in_lat)
    gen = LeakyReLU(alpha=0.2)(gen)
    gen = Reshape((8, 8, Conv_unit))(gen)
    # merge image gen and label input
    merge = Concatenate()([gen, li])
    gen = Conv2DTranspose(Conv_unit, (4,4), strides=(2,2), padding='same')(merge)
    gen = LeakyReLU(alpha=0.2)(gen)
    gen = Conv2DTranspose(Conv_unit, (4,4), strides=(2,2), padding='same')(gen)
    gen = LeakyReLU(alpha=0.2)(gen)
    gen = Conv2DTranspose(Conv_unit, (4,4), strides=(2,2), padding='same')(gen)
    gen = LeakyReLU(alpha=0.2)(gen)
    gen = Conv2DTranspose(Conv_unit, (4,4), strides=(2,2), padding='same')(gen)
    gen = LeakyReLU(alpha=0.2)(gen)
    # output
    out_layer = Conv2D(3, (8,8), activation='tanh', padding='same')(gen)
    # define model
    model = Model([in_lat, in_label], out_layer)
    return model


def define_gan(g_model, d_model):
    d_model.trainable = False  #Discriminator is trained separately. So set to not
trainable.


    gen_noise, gen_label = g_model.input
```

```
    # get image output from the generator model
    gen_output = g_model.output

    # generator image output and corresponding input label are inputs to discriminator
    gan_output = d_model([gen_output, gen_label])
    # define gan model as taking noise and label and outputting a classification
    model = Model([gen_noise, gen_label], gan_output)
    # compile model
    opt = Adam(lr=0.0002, beta_1=0.5)
    model.compile(loss='binary_crossentropy', optimizer=opt)
    return model

# load cifar images
def load_real_samples():
    # load dataset
    trainX = ts_vid
    trainy = ts_cond
    # convert to floats and scale
    X = trainX.astype('float32')
    # scale from [0,255] to [-1,1]
    X = (X - 127.5) / 127.5   #Generator uses tanh activation so rescale
                        #original images to -1 to 1 to match the output of generator.
    return [X, trainy]

def generate_real_samples(dataset, n_samples):
    # split into images and labels
    images, labels = dataset
    # choose random instances
    ix = randint(0, images.shape[0], n_samples)
    # select images and labels
    X, labels = images[ix], labels[ix]

    y = ones((n_samples, 1))  #Label=1 indicating they are real
    return [X, labels], y

# generate points in latent space as input for the generator
def generate_latent_points(latent_dim, n_samples):
    # n_samples =100
        # generate points in the latent space
    x_input = randn(latent_dim * n_samples)
        # reshape into a batch of inputs for the network
    z_input = x_input.reshape(n_samples, latent_dim)

    ind_rnd = np.arange(np.size(ts_cond,0))
    np.random.shuffle(ind_rnd)
    labels = ts_cond[ind_rnd[0:n_samples],:]
```

```
    return [z_input, labels]

def generate_fake_samples(generator, latent_dim, n_samples):
    # generate points in latent space
    z_input, labels_input = generate_latent_points(latent_dim, n_samples)
    # predict outputs
    images = generator.predict([z_input, labels_input])
    # create class labels
    y = zeros((n_samples, 1))  #Label=0 indicating they are fake
    return [images, labels_input], y


def train(g_model, d_model, gan_model, dataset, latent_dim, n_epochs=100,
n_batch=32): #128):
    bat_per_epo = int(dataset[0].shape[0] / n_batch)
    half_batch = int(n_batch / 2)  #the discriminator model is updated for a half batch
of real samples
                        #and a half batch of fake samples, combined a single batch.
    count = 0
    train_loss  = zeros((n_epochs*bat_per_epo,7))
    # manually enumerate epochs
    for i in range(n_epochs):
        # enumerate batches over the training set
        for j in range(bat_per_epo):
            start_time = time.time()
            # i = 0
            # j = 0

            [X_real, labels_real], y_real = generate_real_samples(dataset, half_batch)

            d_loss_real, _ = d_model.train_on_batch([X_real, labels_real], y_real)

            # generate 'fake' examples
            [X_fake, labels], y_fake = generate_fake_samples(g_model, latent_dim,
half_batch)
            # update discriminator model weights
            d_loss_fake, _ = d_model.train_on_batch([X_fake, labels], y_fake)

            [z_input, labels_input] = generate_latent_points(latent_dim, n_batch)

            y_gan = ones((n_batch, 1))

            g_loss = gan_model.train_on_batch([z_input, labels_input], y_gan)
            # Print losses on this batch
            t_epoch = (time.time() - start_time)
```

```
            print('rr%d, day%d, Epoch>%d, Batch%d/%d, d1=%.3f, d2=%.3f
g=%.3f, %.3fs/batch' %(rr_i, day_i, i+1, j+1, bat_per_epo, d_loss_real, d_loss_fake,
g_loss, t_epoch))
            train_loss[count,:] = [i+1, j+1, bat_per_epo, d_loss_real, d_loss_fake, g_loss,
t_epoch]
            count = count + 1

            del X_real, X_fake, labels_real, d_loss_real, labels, d_loss_fake, z_input,
labels_input, y_gan, g_loss
            gc.collect()
        # save the generator model
        g_model.save(str_save_prefix + '_' + str(len(ts_ms)) + 'sample_' + str(N_pix) +
'pix_' + str(n_epochs) + 'epochs.h5')
        np.save(str_save_prefix + '_' + str(len(ts_ms)) + 'sample_' + str(N_pix) + 'pix_' +
str(n_epochs) + 'epochs_loss.npy', train_loss)
    #Train the GAN

    # size of the latent space
    latent_dim = 100
    # create the discriminator
    d_model = define_discriminator()
    # create the generator
    g_model = define_generator(latent_dim)
    # create the gan
    gan_model = define_gan(g_model, d_model)
    # load image data
    dataset = load_real_samples()
    # train model
    os.chdir(save_path)
    train(g_model, d_model, gan_model, dataset, latent_dim, n_epochs=100)

    ### save model

    n_epochs = 100
    model = load_model(str_save_prefix + '_' + str(len(ts_ms)) + 'sample_' + str(N_pix)
+ 'pix_' + str(n_epochs) + 'epochs.h5')

    #### save generated image

    labels = ts_cond
    latent_points, labels_tmp = generate_latent_points(100, len(labels))
    X_gan  = model.predict([latent_points, labels])
    # scale from [-1,1] to [0,1]
    X_gan = (X_gan + 1) / 2.0
    X_gan = (X_gan*255).astype(np.uint8)
```

```
    os.chdir(save_path)
    np.save('rat' + str(rr_i) + '_day' + str(day_i) + '_GAN_iPR_images_' + str(N_pix) +
'pix_' + str(t_ms_incr) + 'msIncr.npy', X_gan)
    # plt.imshow(X_gan[100,:,:,:])


    ########################### plot single reach

    ##################### plot GAN reach
    labels = ts_cond[0:N_iPR,:]
    latent_points, labels_tmp = generate_latent_points(100, len(labels))
    X  = model.predict([latent_points, labels])
    # scale from [-1,1] to [0,1]
    X = (X + 1) / 2.0
    X = (X*255).astype(np.uint8)
    t_ms_vec = np.arange(t_ms_seg[0],t_ms_seg[1]+1,t_ms_incr)
    for i in range(N_iPR-1):
        axes = plt.subplot(5, 10, i+1)
        axes.axis('off')
        X_tmp = X[i, :, :, 0]
        axes.imshow(X_tmp, cmap='gray')
        axes.set_title(str(t_ms_vec[i]) + 'ms')

    ##################### plot real reach
    X_real = ts_vid[0:N_iPR,:,:,:]
    t_ms_vec = np.arange(t_ms_seg[0],t_ms_seg[1]+1,t_ms_incr)
    for i in range(N_iPR-1):
        axes = plt.subplot(5, 10, i+1)
        axes.axis('off')
        X_tmp = X_real[i, :, :, 0]
        axes.imshow(X_tmp, cmap='gray')
        axes.set_title(str(t_ms_vec[i]) + 'ms')
```

## APPENDIX 4: GAN_P2: GENERATING GAN IMAGE, SAVE VIDEO, EXAMPLE PLOT

```
#%% GAN P2: generating GAN image, save video, example plot


%reset -f

from skimage.measure import block_reduce
import numpy as np
from numpy import zeros
from numpy import ones
from numpy.random import randn
from numpy.random import randint
from tensorflow.keras.optimizers import Adam
# from tensorflow.keras.optimizers.legacy import Adam
from tensorflow.keras.models import Model
from tensorflow.keras.layers import Input
from tensorflow.keras.layers import Dense
from tensorflow.keras.layers import Reshape
from tensorflow.keras.layers import Flatten
from tensorflow.keras.layers import Conv2D
from tensorflow.keras.layers import Conv2DTranspose
from tensorflow.keras.layers import LeakyReLU
from tensorflow.keras.layers import Dropout
from tensorflow.keras.layers import Embedding
from tensorflow.keras.layers import Concatenate

from matplotlib import pyplot as plt
import scipy.io as sio
import os
import gc
import time

from tensorflow.keras.models import load_model
import scipy.ndimage as ndimage

import cv2


load_path =
'D:\\20220624_Lethbridge\\20230111_rat_unit_EckertTatsuno\\20240324_sample_cod
e_dt\\rr5\\input_gan_p1_p2'
```

```
save_path =
'D:\\20220624_Lethbridge\\20230111_rat_unit_EckertTatsuno\\20240324_sample_cod
e_dt\\rr5\\output_gan_p2'

rr_day = [[5,7]] # [rr,day]

t_ms_incr = 20 # ms
t_ms_seg = [-400, 600]

Conv_unit = 128

os.chdir(load_path)
r2_all = sio.loadmat('r2_neurons_all_6N.mat')['r2_allall']
for rr_day_i in np.arange(len(rr_day)):
    # rr_day_i = 0
    rr_i = rr_day[rr_day_i][0]
    day_i = rr_day[rr_day_i][1]
    ind_tmp = np.where((r2_all[3,:] == rr_i) & (r2_all[4,:] == day_i))

    r2top_neur = r2_all[2,ind_tmp]
    r2top_neur[0][0] = r2top_neur[0][0][0:6]

    os.chdir(load_path)
    iPR = sio.loadmat('rat' + str(rr_i) + '_day' + str(day_i) +
'_iPR_incr_neuron_iPRDur1000ms_iPRIncr' + str(t_ms_incr) + 'ms.mat')
    iPR_n = iPR["iPR_tr_stack_incr"]
    iPR_n = np.squeeze(iPR_n[r2top_neur[0][0]-1,:,:])
    N_iPR  = np.size(iPR_n,2)
    N_reach = np.size(iPR_n,1)
    N_neur  = np.size(iPR_n,0)

    t_frame = np.load('rat' + str(rr_i) + '_day' + str(day_i) + '_frame_reach_unit.npy')
    t_fpms = np.load('rat' + str(rr_i) + '_day' + str(day_i) + '_frame_reach_fpms.npy')
    ms_per_frame = round(1/t_fpms)
    t_ms = t_frame*ms_per_frame


    count = 0
    iPR_ms  = zeros((N_neur,N_reach*N_iPR))
    for i_reach in np.arange(N_reach):
        # i_reach = 0
        print('load reach ' + str(i_reach))

        iPR_ms[:,((i_reach)*N_iPR):((i_reach+1)*N_iPR)] =
np.squeeze(iPR_n[:,i_reach,:])
```

```
        vid_reach = np.load('rat' + str(rr_i) + '_day' + str(day_i) +
'_frame_reach_vid_zoom_reach' + str(i_reach) + '_prepro.npy')
        N_pix_tmp = 128
        shift_pix_x = 10
        shift_pix_y = 20
        vid_reach =
vid_reach[shift_pix_y:(shift_pix_y+N_pix_tmp),shift_pix_x:(shift_pix_x+N_pix_tmp)
,:]

        N_pix = 128 #64


        tmp_abs = abs(t_ms - t_ms_seg[0])
        ind_start = np.where(tmp_abs == min(tmp_abs))[0]
        tmp_abs = abs(t_ms - t_ms_seg[1])
        ind_end   = np.where(tmp_abs == min(tmp_abs))[0]

        ts_frame = np.arange(ind_start, ind_end+1, t_ms_incr/ms_per_frame,dtype=int)
        for i_ind in np.arange(len(ts_frame)):
            if i_reach == 0 and i_ind == 0:
                ts_ms  = zeros(len(ts_frame)*(N_reach))
                ts_vid = zeros((len(ts_frame)*(N_reach),N_pix,N_pix))
            i_fr = ts_frame[i_ind]
            ts_ms[count]      = t_ms[i_fr]
            ts_vid[count,:,:] = vid_reach[:,:,i_fr]
            count = count + 1

        # plt.imshow(vid_reach[:,:,400])
    if ts_ms.ndim == 1:
        ts_vid = ((ts_vid - ts_vid.min()) * (1/(ts_vid.max() - ts_vid.min()) *
255)).astype('uint8')
        ts_vid = ts_vid[:,:,:,np.newaxis]
        ts_vid = np.repeat(ts_vid,3, axis = 3)

        ts_ms = ((ts_ms - ts_ms.min()) * (1/(ts_ms.max() - ts_ms.min()) *
255)).astype('uint8')
        ts_ms = ts_ms[:,np.newaxis]

        iPR_ms = ((iPR_ms - iPR_ms.min()) * (1/(iPR_ms.max() - iPR_ms.min()) *
255)).astype('uint8')
        iPR_ms = iPR_ms.T #[:,np.newaxis]

    # ts_cond = ts_ms
    ts_cond = iPR_ms

    n_epochs = 100
```

```
    str_load_prefix = ('rat' + str(rr_i) + '_day' + str(day_i) +
'_reach_iPR_conditional_generator')
    model = load_model(str_load_prefix + '_' + str(len(ts_ms)) + 'sample_' + str(N_pix)
+ 'pix_' + str(n_epochs) + 'epochs.h5')

    # for i_n in np.arange(N_neur):
    #     iPR_ms[:,i_n] = np.gradient(iPR_ms[:,i_n])

    # iPR_tmp = iPR_ms[0:65,0]
    # iPR_slope_ms = np.gradient(iPR_tmp)
    # plt.plot(iPR_tmp)
    # plt.plot(iPR_slope_ms)
    # plt.plot(np.zeros(65))
    # plt.imshow()

    ## models


    def define_discriminator(in_shape=(N_pix,N_pix,3)):

        # label input
        in_label = Input(shape=(np.size(ts_cond,1),))
        n_nodes = in_shape[0] * in_shape[1]
        li = Dense(n_nodes)(in_label)
        li = Reshape((in_shape[0], in_shape[1], 1))(li)

        # image input
        in_image = Input(shape=in_shape)
        # concat label as a channel
        merge = Concatenate()([in_image, li])

        fe = Conv2D(Conv_unit, (3,3), strides=(2,2), padding='same')(merge)
        fe = LeakyReLU(alpha=0.2)(fe)
        fe = Conv2D(Conv_unit, (3,3), strides=(2,2), padding='same')(fe)
        fe = LeakyReLU(alpha=0.2)(fe)
        fe = Conv2D(Conv_unit, (3,3), strides=(2,2), padding='same')(fe)
        fe = LeakyReLU(alpha=0.2)(fe)
        fe = Conv2D(Conv_unit, (3,3), strides=(2,2), padding='same')(fe)
        fe = LeakyReLU(alpha=0.2)(fe)
        # flatten feature maps
        fe = Flatten()(fe)
        # dropout
        fe = Dropout(0.4)(fe)
        # output
        out_layer = Dense(1, activation='sigmoid')(fe)  #Shape=1
```

```
    # define model
    model = Model([in_image, in_label], out_layer)

    opt = Adam(lr=0.0002, beta_1=0.5)
    model.compile(loss='binary_crossentropy', optimizer=opt, metrics=['accuracy'])
    return model


# define generator model

def define_generator(latent_dim):

        # label input
    in_label = Input(shape=(np.size(ts_cond,1),))

    n_nodes = 8 * 8
    li = Dense(n_nodes)(in_label) #1,64
    li = Reshape((8, 8, 1))(li)


        # image generator input
    in_lat = Input(shape=(latent_dim,))  #Input of dimension 100

    n_nodes = Conv_unit * 8 * 8
    gen = Dense(n_nodes)(in_lat)
    gen = LeakyReLU(alpha=0.2)(gen)
    gen = Reshape((8, 8, Conv_unit))(gen)
        # merge image gen and label input
    merge = Concatenate()([gen, li])
    gen = Conv2DTranspose(Conv_unit, (4,4), strides=(2,2), padding='same')(merge)
    gen = LeakyReLU(alpha=0.2)(gen)
    gen = Conv2DTranspose(Conv_unit, (4,4), strides=(2,2), padding='same')(gen)
    gen = LeakyReLU(alpha=0.2)(gen)
    gen = Conv2DTranspose(Conv_unit, (4,4), strides=(2,2), padding='same')(gen)
    gen = LeakyReLU(alpha=0.2)(gen)
    gen = Conv2DTranspose(Conv_unit, (4,4), strides=(2,2), padding='same')(gen)
    gen = LeakyReLU(alpha=0.2)(gen)
    # output
    out_layer = Conv2D(3, (8,8), activation='tanh', padding='same')(gen)
        # define model
    model = Model([in_lat, in_label], out_layer)
    return model


def define_gan(g_model, d_model):
```

```
    d_model.trainable = False  #Discriminator is trained separately. So set to not
trainable.


    gen_noise, gen_label = g_model.input
    # get image output from the generator model
    gen_output = g_model.output

    # generator image output and corresponding input label are inputs to discriminator
    gan_output = d_model([gen_output, gen_label])
    # define gan model as taking noise and label and outputting a classification
    model = Model([gen_noise, gen_label], gan_output)
    # compile model
    opt = Adam(lr=0.0002, beta_1=0.5)
    model.compile(loss='binary_crossentropy', optimizer=opt)
    return model

# load cifar images
def load_real_samples():
    # load dataset
    trainX = ts_vid
    trainy = ts_cond
    # convert to floats and scale
    X = trainX.astype('float32')
    # scale from [0,255] to [-1,1]
    X = (X - 127.5) / 127.5   #Generator uses tanh activation so rescale
                        #original images to -1 to 1 to match the output of generator.
    return [X, trainy]

def generate_real_samples(dataset, n_samples):
    # split into images and labels
    images, labels = dataset
    # choose random instances
    ix = randint(0, images.shape[0], n_samples)
    # select images and labels
    X, labels = images[ix], labels[ix]

    y = ones((n_samples, 1))  #Label=1 indicating they are real
    return [X, labels], y

# generate points in latent space as input for the generator
def generate_latent_points(latent_dim, n_samples):
    # n_samples =100
        # generate points in the latent space
    x_input = randn(latent_dim * n_samples)
        # reshape into a batch of inputs for the network
```

```
        z_input = x_input.reshape(n_samples, latent_dim)

        ind_rnd = np.arange(np.size(ts_cond,0))
        np.random.shuffle(ind_rnd)
        labels = ts_cond[ind_rnd[0:n_samples],:]
        return [z_input, labels]

    def generate_fake_samples(generator, latent_dim, n_samples):
        # generate points in latent space
        z_input, labels_input = generate_latent_points(latent_dim, n_samples)
        # predict outputs
        images = generator.predict([z_input, labels_input])
        # create class labels
        y = zeros((n_samples, 1))  #Label=0 indicating they are fake
        return [images, labels_input], y

    labels = ts_cond #[0:N_iPR,:]
    latent_points, labels_tmp = generate_latent_points(100, len(labels))
    X_gan  = model.predict([latent_points, labels])
    # scale from [-1,1] to [0,1]
    X_gan = (X_gan + 1) / 2.0
    X_gan = (X_gan*255).astype(np.uint8)


    ###### save GAN generated video data
    os.chdir(save_path)
    np.save('rat' + str(rr_i) + '_day' + str(day_i) + '_GAN_iPR_images_' + str(N_pix) +
'pix_' + str(t_ms_incr) + 'msIncr.npy', X_gan)


    ############## save as avi video

    ####### save real video
    os.chdir(save_path)
    size = np.size(ts_vid,1),np.size(ts_vid,2)
    fps = (1/t_ms_incr)*100*(2/5) #  slowed by 10, limit of mp4
    duration = np.size(ts_vid,0)/fps
    out = cv2.VideoWriter(('real_iPR_rr' + str(rr_i) + '_day' + str(day_i) + '_incr' +
str(t_ms_incr) + 'ms_pix' + str(N_pix) + '.mp4'),
                          cv2.VideoWriter_fourcc(*'MP42'), fps, (size[1], size[0]))
    for i_v in range(np.size(ts_vid,0)):
        print('rat' + str(rr_i) + ' day' + str(day_i) + ' save vid frame ' + str(i_v))
        data = np.squeeze(ts_vid[i_v,:,:,:])
        out.write(data)
    out.release()
```

```
    ts_vid_g = X_gan

    ####### save GAN video
    os.chdir(save_path)
    size = np.size(ts_vid_g,1),np.size(ts_vid_g,2)
    fps = (1/t_ms_incr)*100*(2/5) #  slowed by 10, limit of mp4
    duration = np.size(ts_vid_g,0)/fps
    out = cv2.VideoWriter(('GAN_iPR_rr' + str(rr_i) + '_day' + str(day_i) + '_incr' +
str(t_ms_incr) + 'ms_pix' + str(N_pix) + '.mp4'),
                          cv2.VideoWriter_fourcc(*'MP42'), fps, (size[1], size[0]))
    for i_v in range(np.size(ts_vid_g,0)):
        print('rat' + str(rr_i) + ' day' + str(day_i) + ' save vid frame ' + str(i_v))
        data = np.squeeze(ts_vid_g[i_v,:,:,:])
        out.write(data)
    out.release()

    ############################ plot single reach

    ##################### plot GAN reach
    labels = ts_cond[0:N_iPR,:]
    latent_points, labels_tmp = generate_latent_points(100, len(labels))
    X  = model.predict([latent_points, labels])
    # scale from [-1,1] to [0,1]
    X = (X + 1) / 2.0
    X = (X*255).astype(np.uint8)
    t_ms_vec = np.arange(t_ms_seg[0],t_ms_seg[1]+1,t_ms_incr)
    for i in range(N_iPR-1):
        axes = plt.subplot(5, 10, i+1)
        axes.axis('off')
        X_tmp = X[i, :, :, 0]
        axes.imshow(X_tmp, cmap='gray')
        axes.set_title(str(t_ms_vec[i]) + 'ms')

    ##################### plot real reach
    X_real = ts_vid[0:N_iPR,:,:,:]
    t_ms_vec = np.arange(t_ms_seg[0],t_ms_seg[1]+1,t_ms_incr)
    for i in range(N_iPR-1):
        axes = plt.subplot(5, 10, i+1)
        axes.axis('off')
        X_tmp = X_real[i, :, :, 0]
        axes.imshow(X_tmp, cmap='gray')
        axes.set_title(str(t_ms_vec[i]) + 'ms')
```