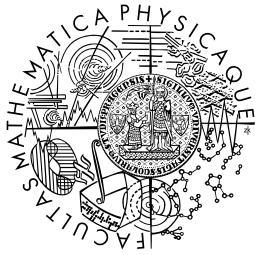

FACULTY
OF MATHEMATICS
AND PHYSICS
Charles University

BACHELOR THESIS

Jakub Parada

Learning V1 targeting optogenetic stimulation protocol for inducing visual perception

Department of Software and Computer Science Education

Supervisor of the bachelor thesis: Mgr. Ján Antolík, Ph.D.
Consultant of the bachelor thesis: Luca Baroni, M.Sc.

Study programme: Computer Science

Study branch: Artificial Intelligence

Prague 2023

I declare that I carried out this bachelor thesis independently, and only with the cited sources, literature and other professional sources. It has not been used to obtain another or the same degree.

I understand that my work relates to the rights and obligations under the Act No. 121/2000 Sb., the Copyright Act, as amended, in particular the fact that the Charles University has the right to conclude a license agreement on the use of this work as a school work pursuant to Section 60 subsection 1 of the Copyright Act.

In ..... date ....

Author's signature

I appreciate all of the support that my supervisor Mgr. Ján Antolík, Ph.D. gave me. His patience, friendly attitude, and will to teach me everything necessary to complete this thesis pleasantly surprised me. I also appreciate his encouragement in the final stretch of the work. Next, I thank Luca Baroni, M.Sc. and Ing. Tibor Rózsa for their theoretical and technical knowledge. When I did not know how to proceed or some academic details were beyond me, they always stepped in and pointed me in the right direction. Lastly, I want to thank Metacentrum for their computational resources and extensive user support in installing all libraries and packages necessary for conducting this thesis.

Title: Learning V1 targeting optogenetic stimulation protocol for inducing visual perception

Author: Jakub Parada

Department: Department of Software and Computer Science Education

Supervisor: Mgr. Ján Antolík, Ph.D., Department of Software and Computer Science Education

Consultant: Luca Baroni, M.Sc., Department of Software and Computer Science Education

Abstract: The Optogenetic stimulation of neurons in the primary visual cortex (V1) is a novel and promising technique for vision restoration for people with acquired blindness. One of the challenges of such a technique is finding artificial stimuli which invoke desired cortical activities. This thesis explores whether neural networks and deep learning can be used for reverse engineering artificial stimuli patterns for optogenetic cortical implant prosthesis (LED) from cortical activity pattern recordings, assuming that similar cortical activity recordings are caused by similar visual stimuli. Various DNN architectures outperforming baseline solutions in stimulus reconstruction will be explored. Loss functions such as MSE and Structural similarity (SSIM) will be used. Questions such as if loss of information in the high-frequency domain of the reconstructed stimuli negatively affects correspondence between the desired cortical activity and the activity elicited by artificial stimuli patterns will be investigated. MSE evaluation metric will be used to determine the degree of similarity between the two types of cortical activities. Due to the limited availability of biological data, we use a model of V1 combined with a model of optogenetic cortical prosthesis (LED) and stimulation developed by et al. [2021] to simulate cortical activity under various stimuli.

Keywords: computational neuroscience, deep learning, image synthesis, Primary visual cortex (V1), optogenetic stimulation

Contents

Introduction

Direct stimulation of neurons inside brain areas responsible for vision, such as in the primary visual cortex (V1), is a promising approach to help people with acquired blindness and irreparably damaged earlier parts of the visual system, such as the eye or optic nerve, regain vision Farnum and Pelled [2020], Liu et al. [2018]. Under the assumption that similar visual stimuli evoke similar cortical activity patterns, one of the challenges of such an approach to vision restoration is understanding how to manipulate the neural activity such that the artificially elicited cortical activity patterns match those elicited through viewing target visual stimulus via an intact visual pathway. For conciseness, artificially elicited cortical activity patterns will be referred to as "artificial activity" (A_{art}), while cortical activity patterns induced by natural vision will be "natural activity" (A_{nat}). Similarly, the stimuli that evoke the two types of activities will be "artificial/natural" stimuli (S_{art}, S_{nat} respectively).

One approach to stimulate the neurons directly is to use an optogenetic cortical implant (LED). In live tissue, the implant would work in the following way: first, a cell population gets transfected to express light-sensitive proteins such as ChannelRhodopsin (ChR); as a result, they become responsive to direct light stimulation. Second, some light pattern (artificial stimulus) is displayed on the LED array superimposed on the cortical surface, resulting in the induction of artificially evoked activity in the neural substrate under the implant. The challenge comes from understanding which artificial stimuli S_{art} on the LED array will elicit the desired artificial activity A_{art}, as under the assumption used, invoking A_{art} to be as close as possible to the A_{nat} would result in functionally the same visual percepts as in the natural vision.

One possible approach would be to use brute force to try all possible artificial stimuli S_{art} and record the artificial activities A_{art}. Afterwards, to artificially induce visual percept matching one produced by natural stimulus S_{nat}, all that is needed is to find the natural activity A_{nat} corresponding to the S_{nat} and use an artificial stimulus S_{art} whose induced artificial activity A_{art} matches A_{nat} as closely as possible. An exact match may not be possible due to the mechanical limitations of the prosthesis.

The problem with such an approach is that it would be computationally unfeasible as the neuron resolution capability of the prosthesis becomes intractable beyond the most trivial cases. Furthermore, using just some subset of possible stimuli S_{art} could limit the range of possible visual percepts expressable by the prosthesis. Therefore, identifying which artificial stimuli evoke artificial activities matching any given cortical activity pattern via some innovative mapping becomes indispensable.

Machine learning, especially neural networks, seems a natural solution for learning such mapping. This thesis investigates several possible architectures of neural networks. The networks take any cortical activity pattern recording and predict artificial stimulus. The training dataset consists only of pairs (S_{art}, A_{art}), which were obtained by simulating the presentation of stimulus s_{art}. The hope is that due to the generalization abilities of neural networks, the artificial stimulus they produce would induce artificial activity close to the given cortical activity

pattern recording. Moreover, previous work has been done on building a forward model that predicts natural activities A_{nat} from natural stimuli S_{nat} citation. This means that a pipeline can be created, where the desired stimulus S_{nat}, the perception of which is to be elicited, is passed through the forward model obtaining the target cortical activity A_{nat}, and the second model - the topic of the thesis - maps this desired cortical activity A_{nat} to the artificial stimulus (induced by the prosthetic system) S_{art}, hence offering a ready-to-use stimulation system.

However, getting sufficient biological data for the data-hungry modern neural network system is very difficult and expensive; therefore, this thesis employs a validated biologically realistic model of V1 along with an optogenetic cortical prosthesis model (LED) cite Antolik capable of simulating manipulation of neural activity via light stimulation. The advantage of using such a model is that the research results will be consistent between the tested methods.

Chapter 1 describes the machine learning fundamentals necessary, a list of the model architectures, losses, metrics and implementation details used. The baseline consists of Linear and MLP architectures. In contrast, tests on more sophisticated convolutional models will show if non-trivial model families can improve the match between induced artificial activity and the desired activity. All model implementations assume cortical activity patterns and artificial stimuli are image tensors. Chapter 2 covers the precise way of obtaining image tensors from cortical activity patterns, the normalization of data, and a description of the final datasets used for training, testing and validation.

Chapter 3 defines the specific experimental settings - experiment pipeline setup and hyperparameters used for network training. After that, follow the individual experiments along with their results.

Chapter 4 examines the success of the experiments, what might have gone wrong and some possible steps for the future. Chapter 5 summarizes the results and topics raised by the discussion.

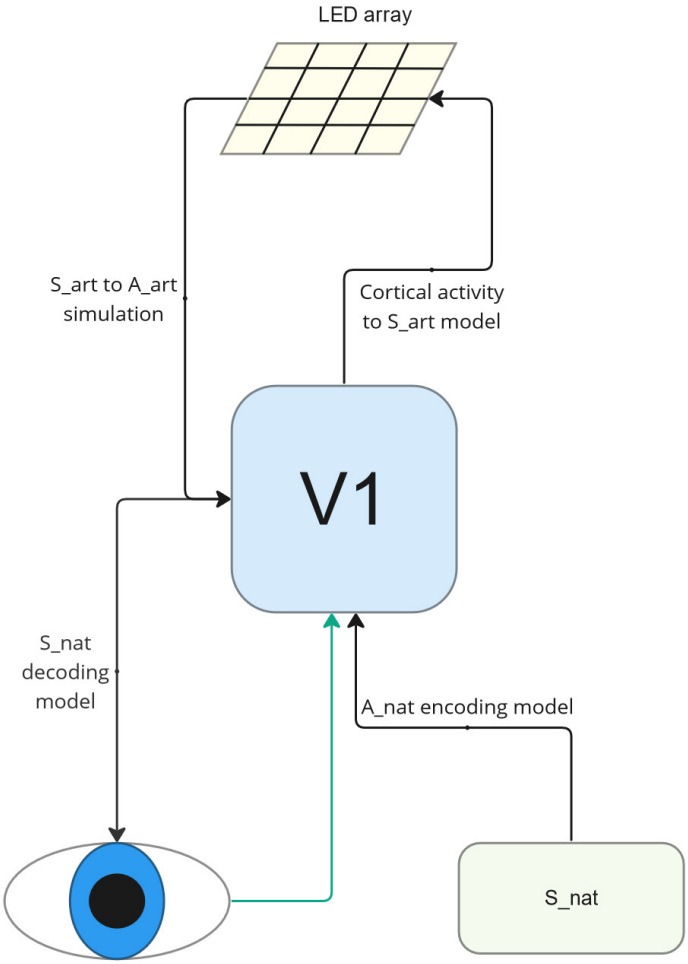

Figure 1: The entire background for this thesis. In an intact visual pathway, visual stimuli start at the eye and get passed to the V1 cortex (indicated by a turquoise arrow), where they invoke cortical activity. Previous work Picek [2022] established mapping from natural stimuli S_{nat} to natural cortex activities A_{nat}. Afterwards Vašek [2023] established model taking cortical activities as an input and producing S_{nat} stimuli on the output. With the help of the Mozaik framework, aby artificial stimuli S_{art} can be converted into artificial activity A_{art}. Now the only missing link is a mapping from cortical activity A_nat to artificial stimuli S_{art} such that artificial activity A_{art} corresponding to the stimulus S_{art} matches the given activity A_{nat}. Finding such mapping would allow for eliciting cortical activity patterns matching those elicited through viewing target visual stimulus via an intact visual pathway by combining it with the model from S_{nat} to A_{nat}. The model for decoding S_{nat} from A_{nat} can also be used to check if the activity A_{art} invoked by S_{art} matches target visual stimulus S_{nat}.

1. Deep Learning and Model Selection for Stimuli Image Prediction

This chapter describes the necessary deep learning knowledge and the choice of models for predicting stimuli images S_{art} from A_{art} used in experiments. Section 1.1 formally defines the main optimisation problem - predicting S_{art} such that corresponding A_{art} is close to given activity A. Section 1.2 describes the various types of machine learning models used to tackle the problem. Linear regression and MLP models are employed as baselines, while subsequent models inspired by Liu et al. [2022] are more complex. Section 1.3 introduces various types of losses and metrics used throughout this work. Examples of used losses and metrics include mean squared error (MSE), mean squared error on deep embeddings from pre-trained networks (deep MSE) and image correlation. Section 1.4 describes implementation details. These include used software libraries, environment and hardware requirements, pre-trained model types, and their exact source used for extracting deep embeddings from images.

1.1 Defining the main optimization objective

The task of alignment of cortical activities A and A_{art} is in this thesis defined in the following way. First, stimulus S_{art} is predicted from activity A. Afterwards, S_{art} is passed into the model of V1 with LED prosthesis to obtain cortical activity A_{art}. The aim is to match A and A_{art} as closely as possible, formally defined as reducing performance measure P computed from the pair. The results of this work can be used in future work aiming to accurately predict stimuli images S_{art} which elicit activity A_{art} equal to given cortical activity from other sources such as natural vision. A description of model training and the reasoning behind architectures selection is below. All definitions in the following part have been adapted from Goodfellow et al. [2016] to suit the context of this thesis, meaning that more general definitions may appear in other literature; however, it is reasonable to modify and restrict them for this work.

1.1.1 What is a model and its training

Definition 1 (Model)**.** *Machine learning model, M, refers to the output of a learning algorithm comprised of model parameters collection ω and algorithm $M(x, \omega)$, which takes input features x along with model parameters ω as input and produces output data y. Models can be parametric or non-parametric. Parametric models use the same set of model parameters with a fixed size for each prediction regardless of the amount of experience E presented to them. Non-parametric models can use variable sets of parameters for each prediction. Examples of parametric models are linear regression and convolution networks, whereas non-parametric models include support vector machines, k-nearest neighbours and decision trees.*

Definition 2 (Learning process)**.** *The learning process comprises presenting experience E to model M with respect to task T and performance measure P. The model learns if its task performance T measured by P improves with experience E.*

Definition 3 (Task T)**.** *In this work machine learning task usually refers to synthesising desired output data y from given input features x. Input features x and output data y are tensors of real numbers. Output data y is desirable iff cortical activity elicited by y resembles x.*

Definition 4 (Performance measure P)**.** *P is a performance measure if it can deterministically produce a quantitative measure of the performance of model M on task T. Usually, such quantitative measure comes as a single scalar number. Examples of performance measures are MSE or image correlation.*

Definition 5 (Dataset)**.** *The dataset is a collection of examples called data points. Each data point is a collection of features which are usually real numbers. Usually, the features are a collection of tensors. Each tensor in the data point can be input or a target for suitable model M depending on the task T. For example, we can use image tensor x as an input and image tensor y as a target in an image mapping task. In contrast, in the task of similarity detection, we can view image tensors x, y on the input as examples of similar images and the job of the model is to learn these associations.*

Definition 6 (Experience E)**.** *In most machine learning algorithms, the definition of experience E can be what kind of data the model can access during the learning process. For example, supervised learning algorithms have access to input and target features during training, meaning they can learn from this experience to produce features similar to target features from a given collection of input features.*

Definition 7 (Supervised training)**.** *Supervised training is a learning process[2] comprised of presenting examples of training data (x, y) to the model M to facilitate minimisation of the loss (differentiable type of performance measure[4] bounded from below) between predictions produced by algorithm $M(x, \omega)$ and targets y. Often validation and test sets are set aside from the training dataset to set hyperparameters of model M and evaluate its performance after learning.*

Definition 8 (Optimizer)**.** *Optimisers used in supervised training are algorithms which take weights ω_1 of parametric model M, loss between the output of the model $M(x, \omega_1)$ and targets y and produce a new set of weights ω_2 which replace ω_1. Over sufficiently enough iterations, such action minimises the loss between $M(x, \omega_2)$ and y. Examples of optimisers are stochastic gradient descent or AdamW introduced in paper Loshchilov and Hutter [2019].*

The usual training loop for parametric models with weights ω in the supervised setting is the following:

1. Choose a batch of examples b from training set.

2. For each data point p in the batch pass the collection of input feature tensors x_p throughout the model M. Aggregate all predicted output feature tensors $o_p = M(x_p, \omega)$ for all data points p into collection o_b. Pass collection o_b along with collection y_b of target feature tensors for all data points in batch b and loss function $\mathcal{L}$ to the optimiser.

3. Optimizer produces a new set of weights ω' which replace the old weights ω.

4. Repeat steps $1 - 3$ until the loss is sufficiently low or other factors, such as the limit to the number of iterations reached, dictate to stop training.

This thesis used a dataset of artificial stimuli and activity patterns to train the models. The hope is that by training artificial stimulus reconstruction from a large enough pool of artificial activities, the models will generalise well on unseen types of cortical activities such as those from natural vision.

1.2 Model types

All model types described below are from the parametric family. Models from this family are better interpretable, have easier manipulation of model capacity and have the possibility of training on GPUs compared to most non-parametric models. Another essential property shared across all model types, regardless of architecture, is that they accept a single image tensor as an input and produce a single image tensor as an output. Image tensor is a 3-dimensional tensor of real numbers, where the first dimension represents channels, whereas the second and third dimensions represent pixel coordinates.

1.2.1 Linear model

A linear model is a model that learns linear mapping from its inputs to its outputs. Modern machine learning frameworks usually implement it as a single Linear/Dense layer without any activation function. During model computation, input image tensor x gets flattened into a vector x', and then the resulting vector y' of affine transformation $Wx' + b$ is reshaped into the image tensor desired output shape. This model type got chosen for its simplicity and the possibility of quantifying whether linear mapping from cortical activity to stimuli images S_{art} suffices to elicit similar cortical responses. The input/output shape details can be found in figure 1.1.

1.2.2 MLP model

MLP model has a series of linear layers and non-linear activation functions in between them. The activation function of choice is Gaussian Error Linear Unit (GELU) first introduced in Hendrycks and Gimpel [2023]. As in the previous model, all input features get flattened into a single vector before passing it through the network. At the end of the computation, the output gets reshaped into an image tensor with the desired shape. The shapes of the tensors during computation can be found in figure 1.2.

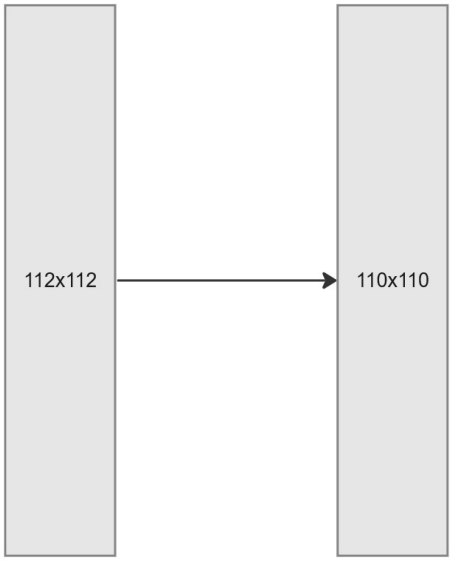

Figure 1.1: Workflow of the linear model with individual input/output tensors sizes.

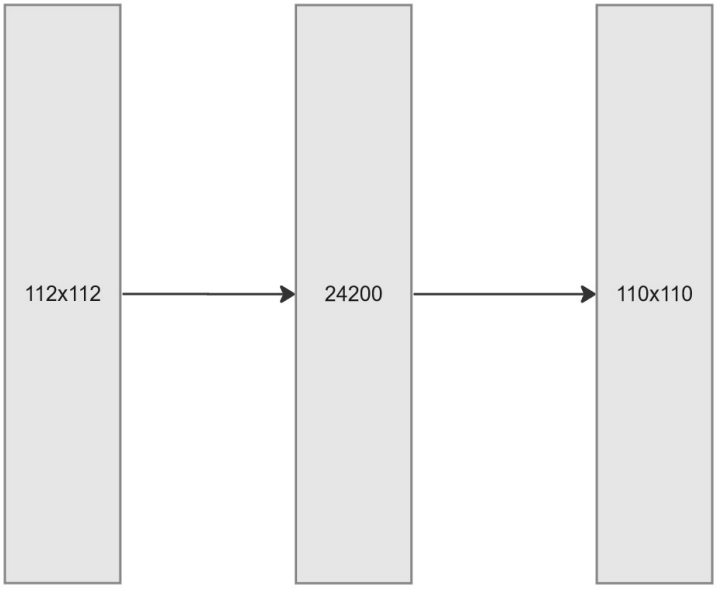

Figure 1.2: Sizes of tensors during computation of the MLP model.

1.2.3 Convolutional models

Unless stated otherwise, all convolutional models investigated consist of three main blocks - encoder, embedding-to-image transformation, and upscaler. The encoder's job is to process input image tensors into meaningful embeddings. The embeddings are used in the embedding-to-image transformation block to obtain an intermediate low-resolution image which is then upscaled by the upscaler block. It is important to note that the low-resolution image has many channels (16 to 64), as this helps to preserve information from the encoder input while meaningfully separating it across multiple locations, since subsequent upscaler layers do not have receptive field large enough to combine features from far away parts of the intermediate image.

1.2.4 CNN V1

The first convolutional model is inspired by the models SRResNet and ConvNeXt introduced in papers Ledig et al. [2017] and Liu et al. [2022], respectively.

The encoder consists of three pairs of (contraction, CNN) blocks. The CNN blocks were taken from the ConvNeXt above model. Each CNN block consists of Depth-wise convolution, Layer norm, point-wise convolution 1, GELU, point-wise convolution 2 and residual connection to the block input. Contraction blocks comprise a convolution layer with kernel size $(2, 2)$ and strides $(2, 2)$, Layer norm and GELU activation.

The embedding-to-image transformation is made of a linear layer followed by GELU activation. The belief is that the encoder produces embeddings which combine activity patterns from various parts of the input activity tensors; as such, there isn't a clear structure to the embeddings anymore. Therefore, the linear layer was chosen for its ability to combine the embedding features arbitrarily.

The upscaler first uses a convolution layer to reduce the number of channels from the transformation block output. The convolution is followed by batch norm and GELU activation. Afterwards, two Expansion blocks using sub-pixel convolution for upscaling separated by one CNN block are responsible for the main brunt of the upscaling. Lastly, one convolution layer followed by tanh activation squashes the number of channels to 1 representing the B/W stimulus image.

The detailed depictions of the CNN block can be found in figure 1.3, of the contraction block in figure 1.4 and the expansion block in figure 1.5. The high-level design of the model architecture is in figure 1.6.

1.2.5 CNN V2

The CNN V2 model 1.7 is very similar to the CNN V1 model. It was constructed by slowly tinkering with the CNN V1 model by including or removing layers to improve performance on the validation set. The main differences are in the upscaler part. The batch normalisation is removed, and after the second expansion block comes another CNN block. These modifications may seem simple but result from systematically eliminating various other changes.

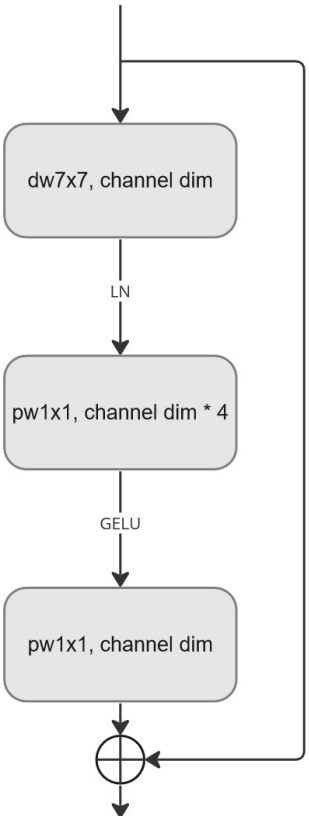

Figure 1.3: CNN block inspired by ConvNeXt. *channel dim* represents the number of input channels for the cnn block.

1.3 Losses and Metrics

The performance of the trained model is often heavily influenced by choice of the loss function. The most straightforward type loss function in image synthesis is MSE between the predicted and target image. However, it suffers from several drawbacks; hence it is usually combined with other loss functions or modified in several key aspects. The experiments performed investigate several such combinations and modifications. All individual parts of all loss functions used are defined below.

1.3.1 MSE

The mean squared error (MSE) between two tensors x_1, x_2 is defined as:

$$\sum (x_1 - x_2)^2 \tag{1.1}$$

Where the subtraction and squaring are applied element-wise, the problem with using only MSE is that in image synthesis tasks, it tends to reconstruct only low-frequency features in the output image. This problem stems from the fact that in image synthesis, there are usually a couple of closely-related options for the appearance of the final image. Hence, the optimiser picks the average from all of them. Another problem is that many pixels matching their targets can suppress a small number of incorrectly bright or dim pixels in their vicinity, as correcting

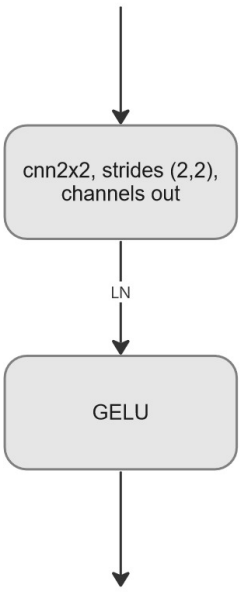

Figure 1.4: Contraction block design. Usually, *channels out* is twice the number of channels in the input tensor.

the value of the minority can lead to the increase of the overall MSE due to slight misalignment in the mostly correct pixels brought by this revision.

1.3.2 Image correlation

Cross-correlation of two images A, B with the same dimensions can be defined as:

$$r = \frac{\sum\left(\left(A - \overline{A}\right) \cdot \left(B - \overline{B}\right)\right)}{\sqrt{\sum\left(A - \overline{A}\right)^2 \cdot \sum\left(B - \overline{B}\right)^2}} \tag{1.2}$$

Where the overline symbolises the mean of the image tensor. All arithmetic operations are applied element-wise, while the final summations sum everything across all dimensions and produce a scalar. The correlation defined like this can be used only as a metric, as the range of its outputs is $[-1, 1]$ where 0 is considered an exact match. Another reason for its unsuitability as a loss function is that two images whose pixel intensities are multiples of each other are indistinguishable by it, meaning that the intensities of the predicted images would be unbounded during training, which leads to divergence of weight sizes.

1.3.3 SSIM

In contrast to MSE, and image correlation, the structural similarity index measure Wang et al. [2004] compares luminance, contrast and structure between the

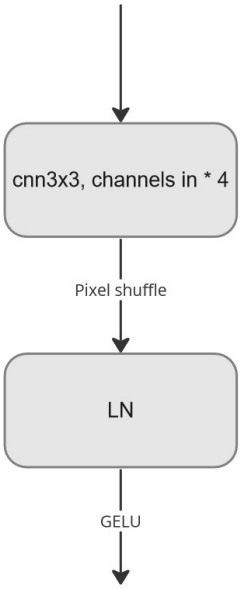

Figure 1.5: Expansion block design. *channels in * 4* with pixel shuffle afterwards create sub-pixel convolution Shi et al. [2016].

samples, meaning that all of the main concerns of the prior losses are mitigated to a certain degree. The structure and contrast components prevent MSE over-smoothing, the luminance component provides the ability to distinguish between images with different intensities, and the overall measure is not dependent on pre-trained embeddings, meaning that it can be applied to grayscale images without issues. The formula for the SSIM is:

$$SSIM(x,y) = \frac{(2\mu_x\mu_y + c_1)(2\sigma_{xy} + c_2)}{\left(\mu_x^2 + \mu_y^2 + c_1\right)\left(\sigma_x^2 + \sigma_y^2 + c_2\right)}$$

where μ_x, μ_y are the mean values of x, y, σ_x, σ_y are standard deviations of x, y, σ_{xy} is the cross-correlation between x, y. The constants c_1 and c_2 help stabilise the division with a weak denominator. The typical choice for them is $c_1 = (0.01 \cdot L)^2$, $c_2 = (0.03 \cdot L)^2$ where L is the data range, meaning that for grayscale images normalised to the range $[0, 1]$ the value of L would be 1. The output of the SSIM index is in the range $[0, 1]$ where 0 indicates complete dissimilarity and 1 is a perfect match. SSIM index can be converted to a loss function suitable as an optimisation target in the following way:

$$\mathcal{L}_{SSIM}(x,y) = 1 - SSIM(x,y)$$

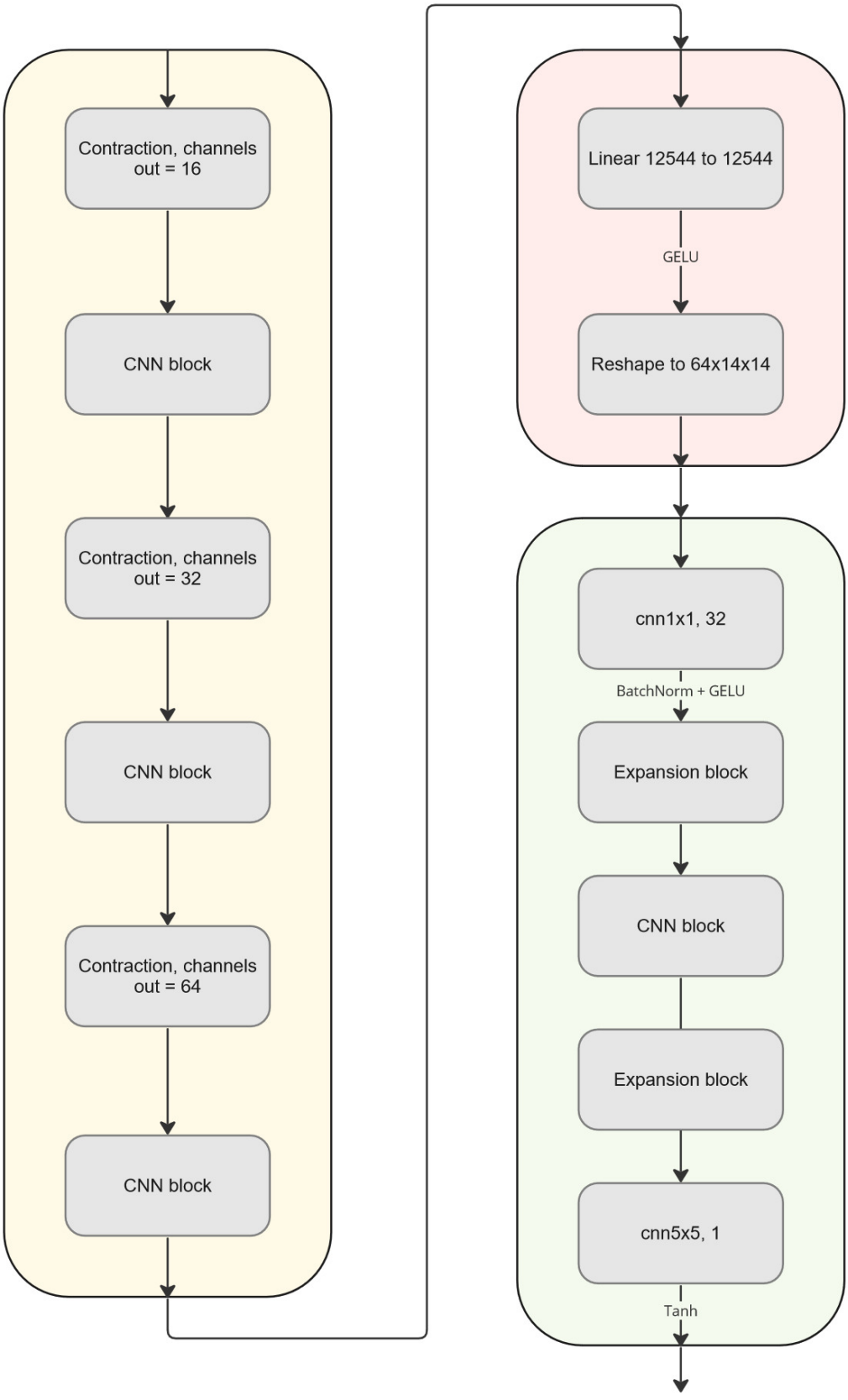

Figure 1.6: Overall CNN V1 architecture.

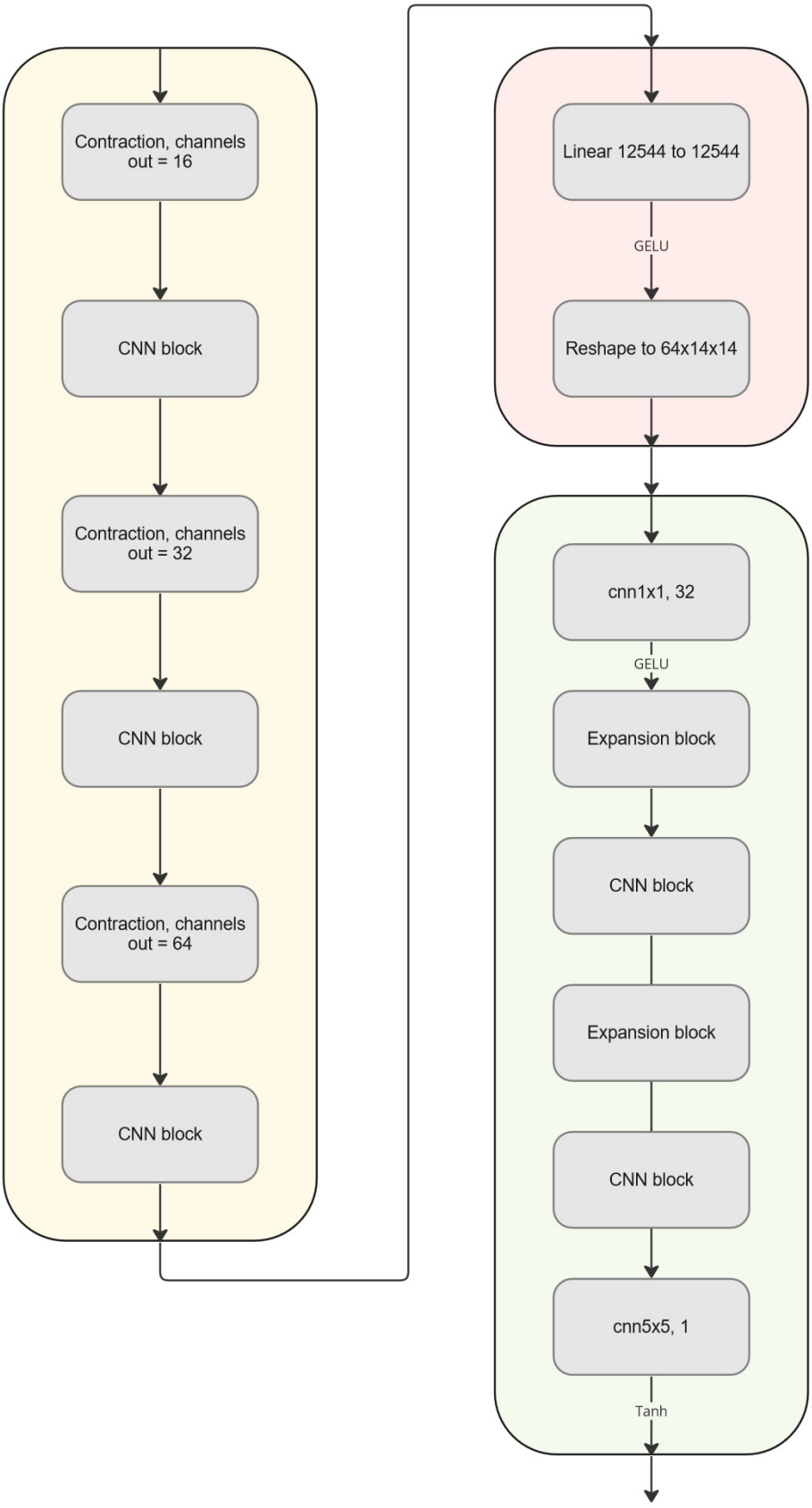

Figure 1.7: Overall CNN V2 architecture.

1.4 Implementation details

All models were implemented via the PyTorch framework in Python 3.10.6. The selected optimiser is AdamW Loshchilov and Hutter [2019]. All hyperparameters of the training, such as learning rate or batch size, are listed directly with the experiments. The simulations of the V1 cortex and the LED prosthesis were done with the Mozaik workflow system[1]. The implementation of SSIM loss was taken from PyTorch-MSSSIM library[2]. The complete code used for this thesis can be found at [3].

Hardware requirements are split into two parts, one for the generation of the dataset and one for the training of the models. The training of the neural networks can be done on a single GPU with at least 16GB of VRAM. The training of each network takes between 60 and 240 minutes. Generation of the dataset and validation of activities A_{art} produced by synthesised stimuli S_{art} were done on the MetaCentrum, as it is CPU bound and a very RAM-intensive process. On a single CPU simulation of activities A_{art} produced by 50 stimuli S_{art} can take up to 24 hours and consume 40GB of RAM throughout the process.

Full acknowledgement for the Metacentrum services: Computational resources were provided by the e-INFRA CZ project (ID:90254), supported by the Ministry of Education, Youth and Sports of the Czech Republic.

[1] `https://github.com/CSNG-MFF/mozaik`

[2] `https://github.com/VainF/pytorch-msssim/`

[3] `https://gitlab.mff.cuni.cz/paradaj/optogenetic_stimulation_protocol/`

2. Dataset for DNN training

The dataset for training the networks predicting reconstructed stimuli from the cortical activity is obtained from a biologically detailed spiking model of the cat's primary visual cortex (et al. [2021]). The dataset consists of data points with two parts: Stimulus S_{art} and Cortical activity A_{art}. In general, Stimuli are cropped images taken from the ImageNet dataset Deng et al. [2009]. These images were chosen because they represent well the statistics of naturalistic images. Cortical activities are temporal recordings of activation spikes of individual cortical neurons; hence more advanced data processing pipeline was used to obtain a single image-like tensor representation for each activity A_{art} recording.

2.1 Stimulus presentation to V1 cortex model

Each stimulus S_{art} presented to the V1 model is a grayscale image with size 110×110 pixels and pixel values in the range 0.0 to 1.0. The stimulus S_{art} is presented through optogenetic cortical prosthesis (LED array) of size 3300.0 µm and LED spacing 30.0 µm. Update interval for the spiking neural network is 1.0 ms, duration of stimulus presentation is 150.0 ms, of which 50.0 ms is onset-time and 100.0 ms is offset-time. The result of each stimulus presentation is artificial activity A_{art} recording.

2.2 Cortical activity

The activity A_{art} is taken from excitatory neurons in layer 2/3 (L2/3). Each neuron has (x, y) positions and a list of times with activation spikes. We can obtain a tensor of cortical activity by binning neuron activation spikes based on the spatial locations of neurons and the temporal location of the activation. One cell (t, i, j) in the tensor represents the firing rate of neurons in temporal bin $[t, t + t_{res})$ and spatial bin (i, j) corresponding to a square with corners (x_i, y_j) and $(x_i + s_{res}, y_j + s_{res})$. t_{res} and s_{res} represent temporal and spatial resolution respectively. x_i and y_j denote affine mapping between indices (i, j) and actual coordinates (x, y) in V1 cortex tissue. The firing rate is calculated as

$$\frac{N(i, j, t, t_{res})}{t_{res}} \tag{2.1}$$

where $N(i, j, t, t_{res})$ is number of neurons that fired in bin (i, j) over time period $[t, t + t_{res})$. For instance $N(i, j, t, t_{res}) = 2$ and $t_{res} = 5$ ms yields rate of $400\,\mathrm{s}^{-1}$.

Means across the temporal dimension were taken to reduce the dataset's size significantly and as the first measure for normalisation of inputs, as a vast majority of cells in the cortical activity tensor have value 0. Working with a single image-like tensor representing A_{art} ensures that proper data pre-processing can be meaningfully applied, as individual slices across the temporal dimension have statistics which are even more skewed than statistics of averaged A_{art}.

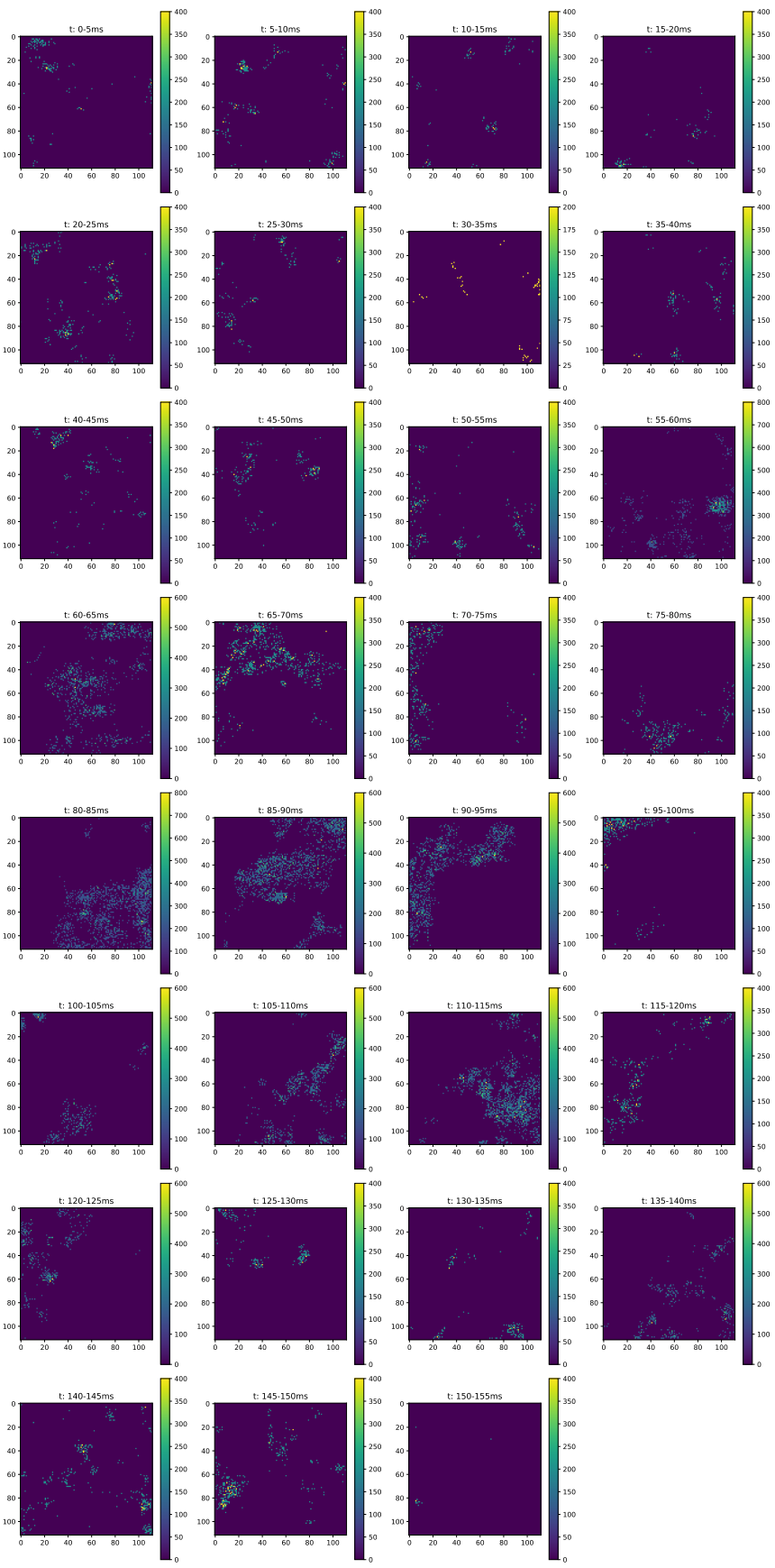

Figure 2.1: Example of binned cortical activity. The plots are organised by increasing time range.

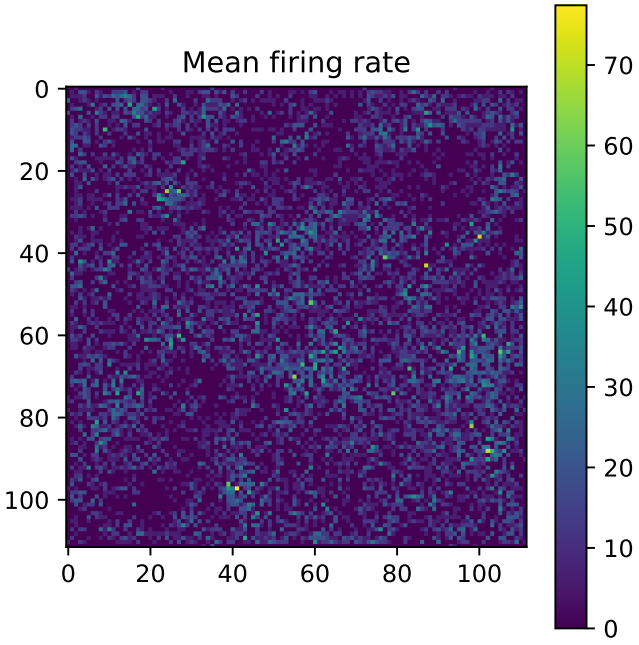

Figure 2.2: Example of cortical activity tensor averaged through the temporal dimension.

2.3 Cortical activity pre-processing

The algorithm written in Python used for a pre-processing tensor of mean activities A_{art} `artificial_cortical_act` is as follows:

```
import numpy as np

artificial_cortical_act = np.load("artificial_cortical_act.npy")
# data pre-processing
artificial_cortical_act = np.log(artificial_cortical_act + 1)
artificial_cortical_act -= np.min(artificial_cortical_act)
artificial_cortical_act /= np.max(artificial_cortical_act)
artificial_cortical_act -= np.mean(artificial_cortical_act)
```

where arithmetic operations $+, -, /$ are applied in element-wise fashion to the `artificial_cortical_act` tensor. Element-wise application fashion means that each element of `artificial_cortical_act` tensor is operated on individually with the corresponding scalar operand on the right side of the operators. For instance `artificial_cortical_act + 1` would add 1 to each element in $4D$ tensor `artificial_cortical_act` of shape (`num_act`, `1`, `rows`, `columns`), where `num_act` is the number of activities A_{art} in the dataset and 1 is number of channels meaning that `artificial_cortical_act` is tensor of image-like tensors with 1 channels representing artificial activities A_{art}.

The intuition behind this style of transformation is that the distribution of average firing rate across all mean tensors of A_{art} is very skewed towards low values - 0.9996% of all firing rate values are below 30, as it can be seen in the figure 2.3. However, a few pixels have firing rate values up to 800 even after taking the mean across the temporal dimension. The natural logarithm breaks

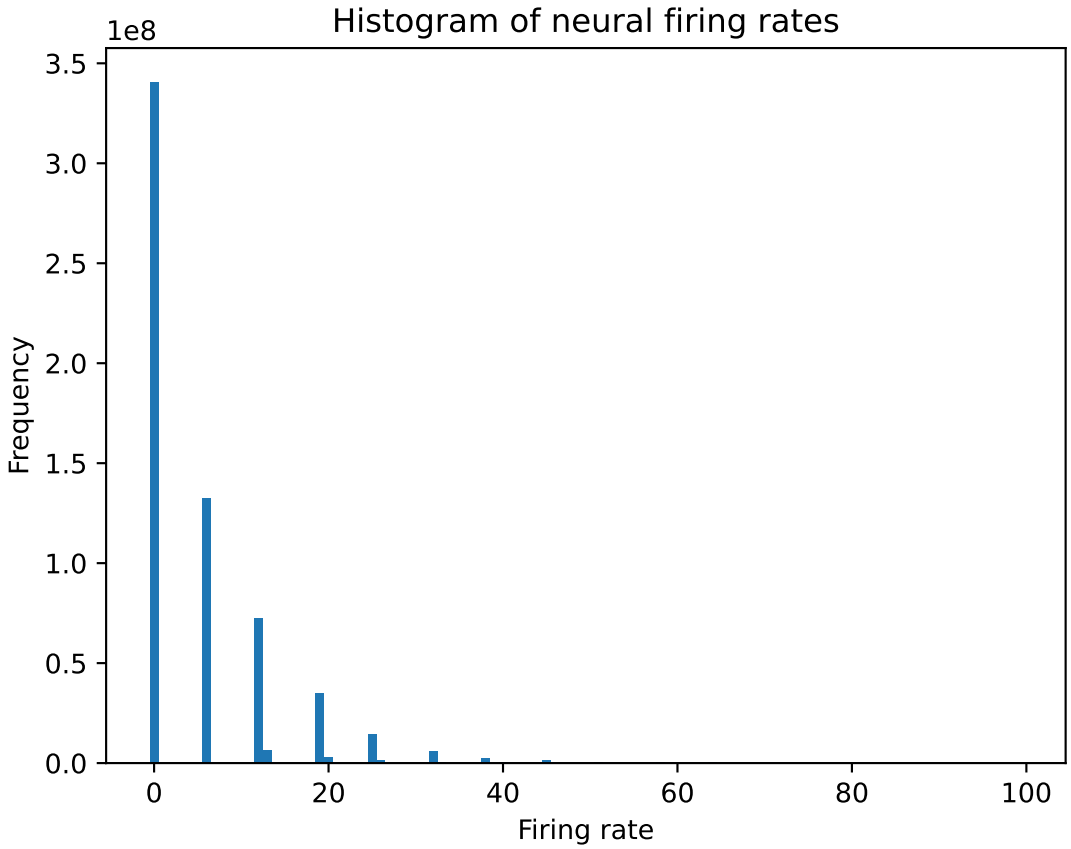

Figure 2.3: Histogram of cortical activity before applying pre-processing transformations.

up this skewness by separating the values by their order of magnitude while not wholly disregarding the firing rates' values. The operations after the logarithm ensure that the range of values of `artificial_cortical_act` is acceptable for the neural networks, i.e. the difference between the largest and smallest value is at most 1 while the arithmetic mean of tensor `artificial_cortical_act` is 0. The histogram for pre-processed cortical activities in figure 2.4 shows a more sensible separation of firing rates.

2.4 Stimulus pre-processing

The general pre-processing steps of stimuli images `stimuli` for network training are:

```
import numpy as np

stimuli = np.load("stimuli.npy")
# data pre-processing
stimuli -= np.mean(stimuli)
```

All pixels in the array of stimuli images are in the range $[0, 1]$ due to previous usage of stimuli array in the simulation of V1 cortex; hence simple data-centring suffices.

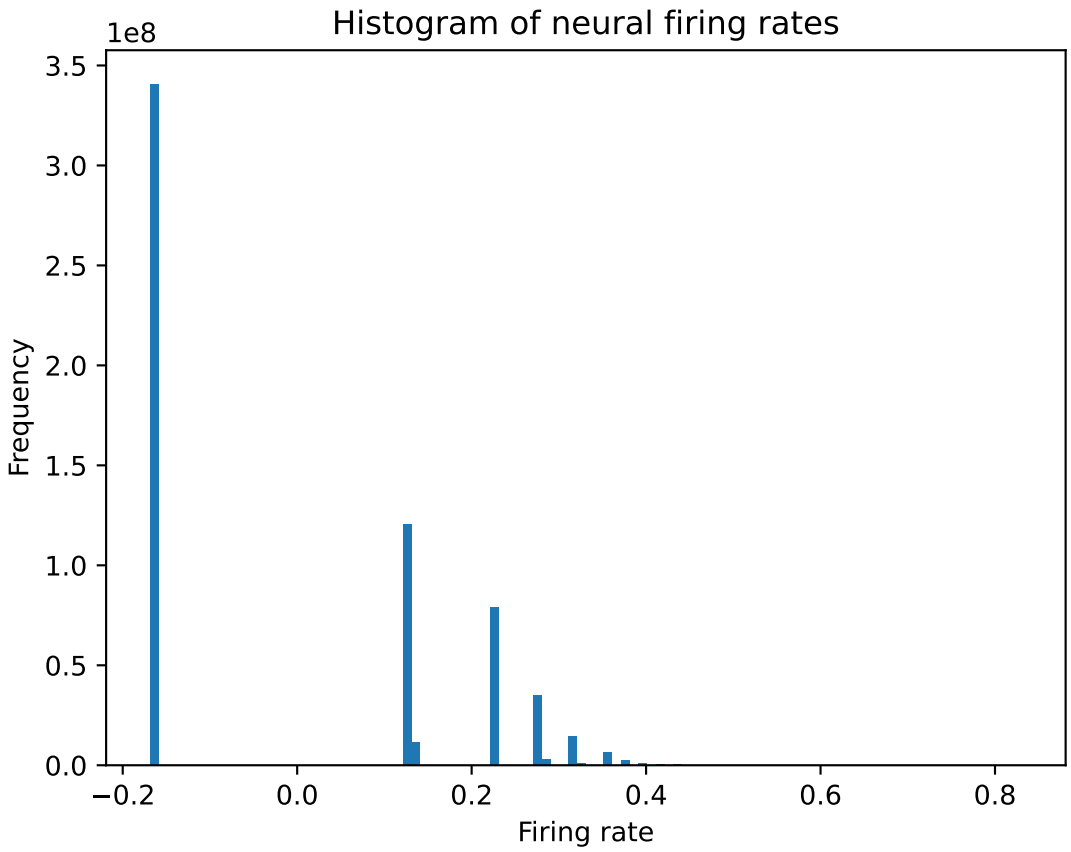

Figure 2.4: Histogram of cortical activity after applying pre-processing transformations.

In the case of the convolutional models, the stimuli are further downsampled by a factor of 2, as the models are designed with bottleneck architecture (they embed the activity tensors to smaller feature tensor, which is then transformed to low-resolution stimulus and upscaled afterwards). The theory is that the fewer moving parts to optimize, the better, hence by working with smaller stimuli, the models can learn the transformation and embeddings more efficiently, as they don't have to do upscaling by a factor of 8, only by the factor of 4. The downsampling is done by averaging 2×2 groups of pixels from stimuli with stride 2.

2.5 Dataset description after pre-processing

The dataset contains 49150 pairs of (artificial stimulus, artificial activity) image-like tensors. The resolution of stimuli is 110×110 pixels, and the resolution of artificial activity tensors is 112×112. The stimuli images and activity tensors are unique; hence they can be split into train, validation, and test sets in a ratio of $80 : 10 : 10$. Below are descriptions and basic statistics for each split:

Train Set:

- Number of samples: 39320
- Stimuli desctiprion: min -0.440, max 0.560, mean 0.0002, std 0.249
- Cortical activity description: min -0.164, max 0.836, mean 0.0001, std 0.189

Validation Set:

- Number of samples: 4915
- Stimuli desctiprion: min -0.440, max 0.560, mean -0.0023, std 0.250
- Cortical activity description: min -0.164, max 0.835, mean -0.0006, std 0.189

Test Set:

- Number of samples: 4915
- Stimuli desctiprion: min -0.440, max 0.560, mean 0.0003, std 0.250
- Cortical activity description: min -0.164, max 0.836, mean -0.0001, std 0.189

3. Experiments + results

Each experiment is structured according to the following outline:

1. First, a machine learning model is trained on mini-batches of size 4 from the training dataset. The validation set was used to manually find the best hyperparameters, such as the sizes of layers and loss weights during training. Hence, the validation error metrics are as low as possible. All hyperparameters are listed at the start of each experiment description.

2. The trained model is used to predict artificial stimuli S_{model} from A_{art} from the test set, which are given to a model of the optogenetic cortical prosthesis. Here S_model notation is used instead of S_{art} to indicate that S_{art} serve as gold labels of the dataset, while S_{model} is the actual stimulus produced by the machine learning models and A_{model} is the cortical activity invoked from S_{model} through the optogenetic cortical prosthesis.

3. The resulting cortical activity tensors A_{model} and test set activities A_{test} are compared to each other with the MSE and SSIM metrics. Both tensors are pre-processed in the same way, i.e. natural logarithm is applied to each element of the tensors. Subsequently, both activities are divided by the maximum value from A_{test}. A_{test} is a subset of A_{art} used in the test set. In contrast to the cortical activity pre-processing process described in section 2.3, subtraction of the mean is omitted, as it would not substantially impact the MSE nor SSIM metrics.

The validation score is measured as the similarity between S_{model}, while the test score is measured as the similarity between A_{art} and activity A_{model}. Visualizations of the validation and test differences are shown in the following figures: Figure 3.1 shows one pair from the validation set and their difference map, while Figure 3.2 shows one pair from the test set and their difference map.

3.1 Linear Model

Only one experiment with the linear model was performed. The hyperparameters for the experiment are in the table below.

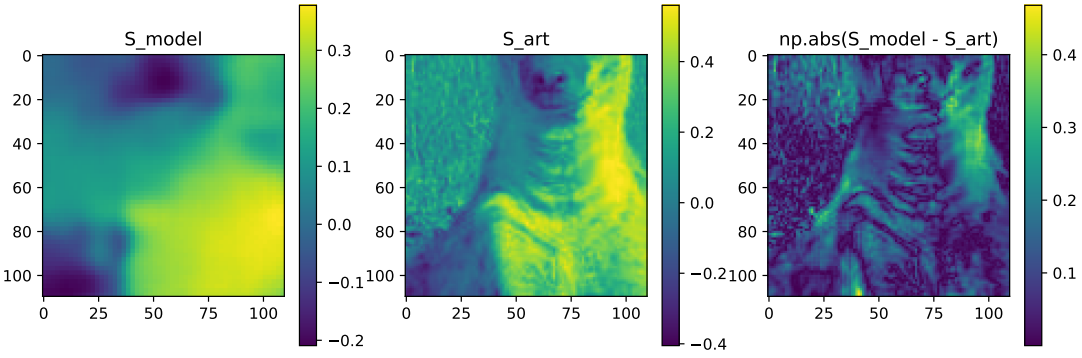

Figure 3.1: Example of (S_{model}, S_{art}) pair used for validation score.

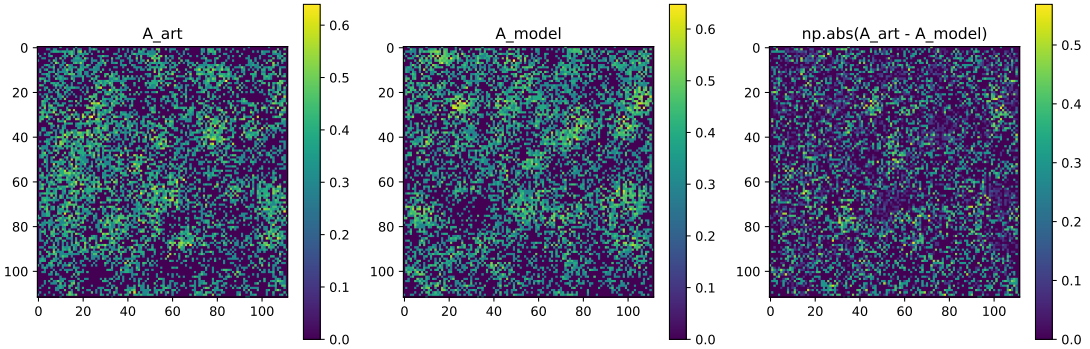

Figure 3.2: Example of (A_{model}, A_{art}) pair used for test score.

Optimizer	AdamW
Learning rate	1e-3
Batch size	4
Weight decay (l2 regularization)	1e-4
Training loss functions	MSE, SSIM
Loss weights	$1, 1$
Number of epochs	5

Table 3.1: Hyperparameters used for the linear model training.

3.1.1 Training Results

The table 3.2 depicts the model evaluation results on the validation set after each epoch. The best validation performance was observed after epoch 4; hence, model weights from this epoch were used to evaluate activity comparisons. Overall, the linear model is insufficient for general stimulus reconstruction. MSE and $\mathcal{L}_{SSIM}$ in the table 3.2 indicate that all reconstructed stimuli are almost entirely dissimilar from the artificial stimuli in the validation set. Even after the best epoch, the $\mathcal{L}_{SSIM}$ validation result is 0.987, where 1 signifies complete dissimilarity. Similarly, an MSE of 0.310 signifies that something very bad happened, as even two random uniform variables from the range $(0, 1)$ have expected MSE of $\frac{1}{6} \approx 0.16$ between them.

Epoch	MSE	$\mathcal{L}_{SSIM}$
1	0.328	0.988
2	0.320	0.988
3	0.320	0.987
4	0.310	0.987
5	0.323	0.987

Table 3.2: Linear model training results on the validation set after each epoch.

3.1.2 Activity comparison

The metrics comparing A_{linear} and A_{test} are summarized in the table 3.3. $\mathcal{L}_{SSIM}$ and MSE denote the average SSIM and MSE loss across all (A_{linear}, A_{test}) pairs.

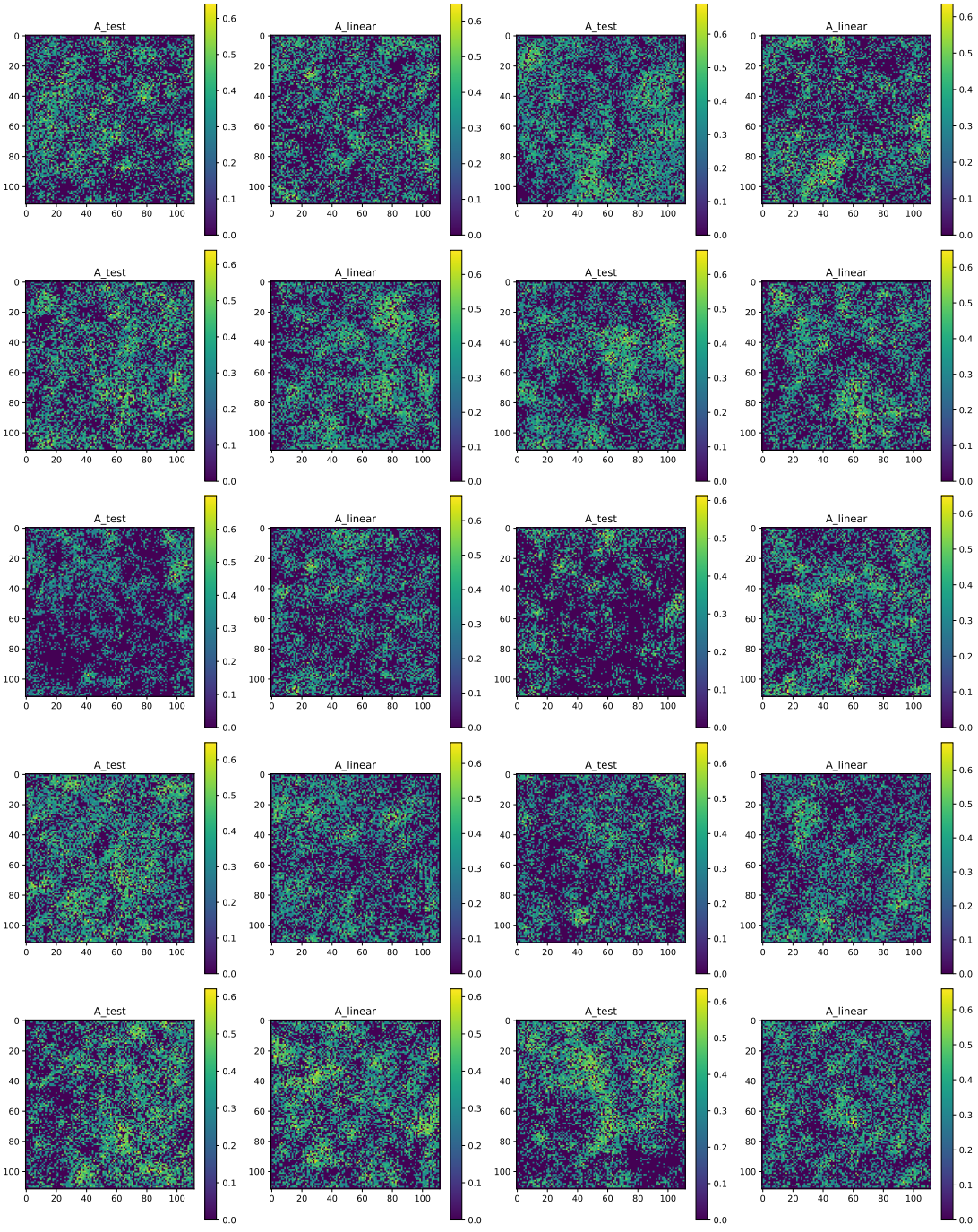

Figure 3.3: Comparison of the first 10 activity tensors from A_{test} and A_{linear}. The first and second columns form the first 5 pairs, while the 3rd and 4th columns form the second five pairs.

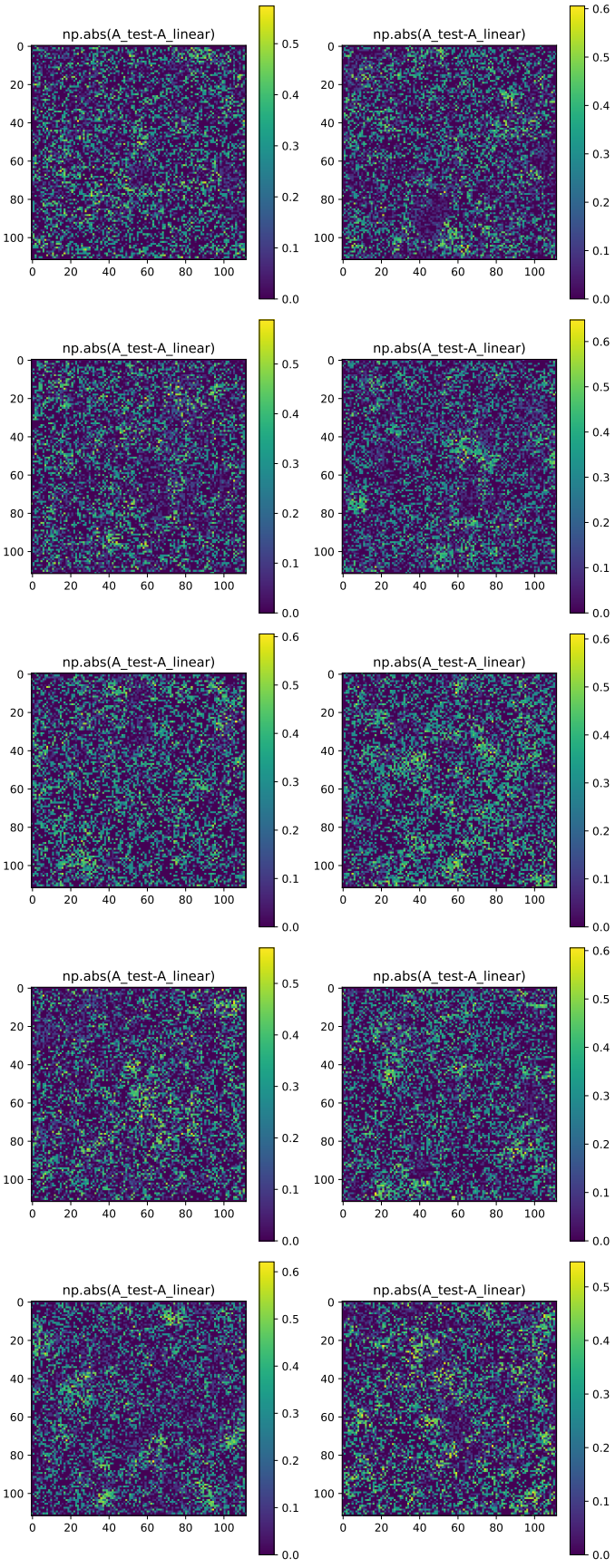

Figure 3.4: Absolute difference (error map) of the first 10 activity tensor pairs from A_{test} and A_{linear}.

The other four rows give context to these values. *Mean A_{xx} loss* $\mathcal{L}$ denotes the mean loss $\mathcal{L}$ across all pairs of activities sampled from A_{xx}. Given the context, the results of the linear model align with expectations from the training - sampling a random stimulus from the training set as the model output would probably be functionally the same as using the Linear model.

$\mathcal{L}_{SSIM}$	0.471
MSE	0.036
Mean A_{linear} SSIM	0.531
Mean A_{test} SSIM	0.417
Mean A_{linear} MSE	0.032
Mean A_{test} MSE	0.039

Table 3.3: Average difference between A_{linear} and A_{test} described by MSE and SSIM metrics. MSE of 0 and SSIM of 1 mean no difference.

3.2 MLP model

Similarly to the linear model, only one final experiment with MLP was performed. The choice of hyperparameters, especially increasing the hidden layer size, didn't seem to impact the model's generalisation capacity during training; as such, the most simple version of hyperparameters, as depicted in table 3.4 was chosen. Compared to the linear model, only 3 training epochs were done for the MLP predictor, as it tended to overfit the training data and not generalise well on the unseen validation data.

Optimizer	AdamW
Learning rate	1e-3
Batch size	4
Weight decay (l2 regularization)	1e-4
Training loss functions	MSE, SSIM
Loss weights	$1, 1$
Number of epochs	3
Hidden layer size	[24200,]

Table 3.4: Hyperparameters used for the MLP model training.

3.2.1 Training results

The training results in table 3.5 indicate that the MLP model can generalise better on previously unseen stimuli from the validation set than the linear model.

3.2.2 Activity comparison

Surprisingly the improved capacity of the MLP model to generate stimuli didn't seem to help at bridging the gap between A_{mlp} and A_{test} as shown in table 3.6. Although the results look almost the same, the MLP model comes slightly on

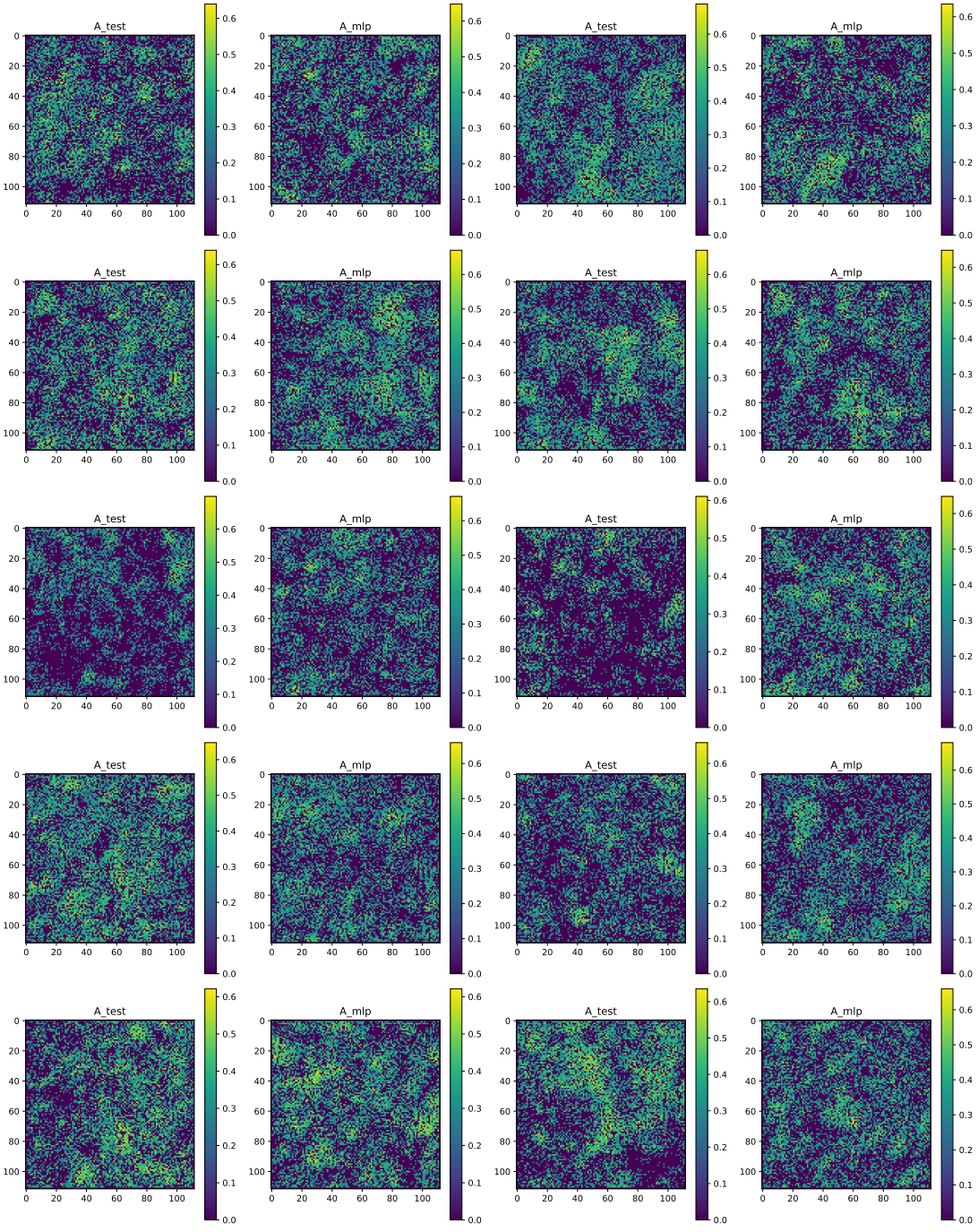

Figure 3.5: Comparison of the first 10 activity tensors from A_{test} and A_{mlp}. The first and second columns form the first 5 pairs, while 3rd and 4th columns form the second five pairs.

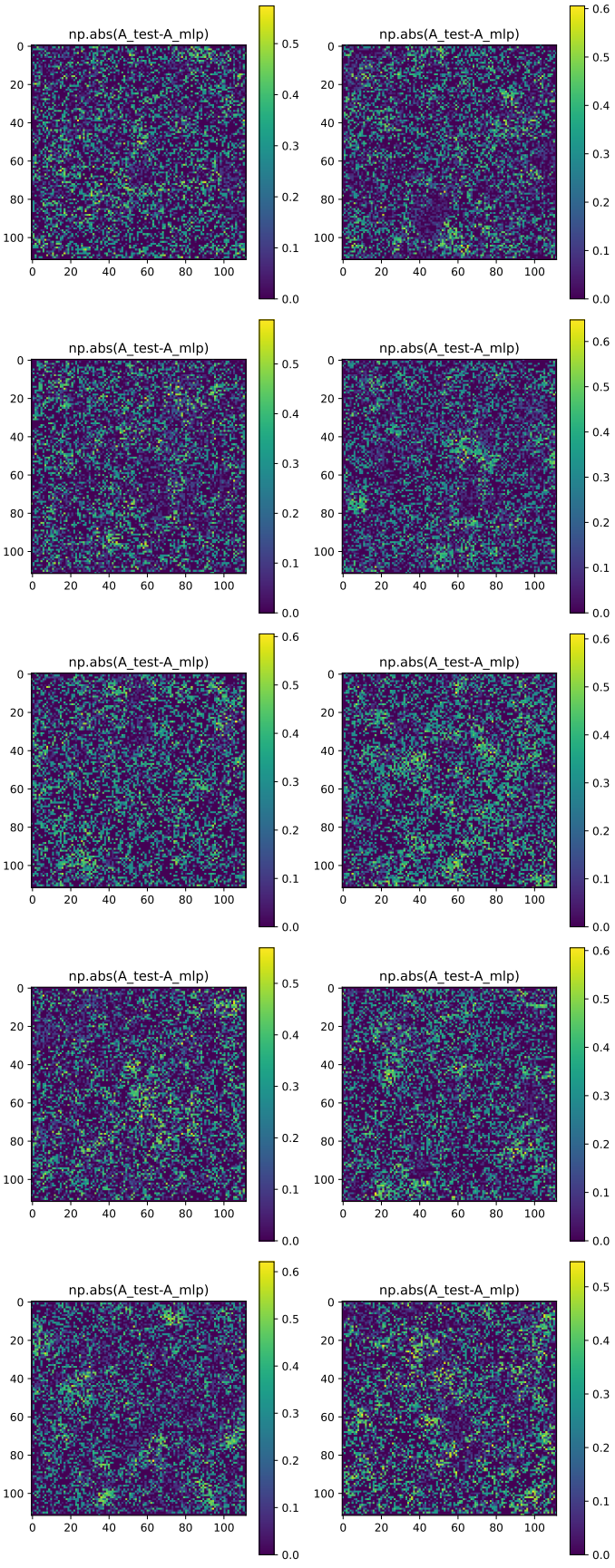

Figure 3.6: Absolute difference (error map) of the first 10 activity tensor pairs from A_{test} and A_{mlp}.

Epoch	MSE	$\mathcal{L}_{SSIM}$
1	0.035	0.736
2	0.035	0.734
3	0.034	0.735

Table 3.5: MLP model training results on the validation set after each epoch.

top. This still does not change the conclusion that even the MLP model cannot generalise well on previously unseen stimuli.

$\mathcal{L}_{SSIM}$	0.470
MSE	0.035
Mean A_{mlp} SSIM	0.531
Mean A_{test} SSIM	0.417
Mean A_{mlp} MSE	0.032
Mean A_{test} MSE	0.039

Table 3.6: Average difference between A_{mlp} and A_{test} described by MSE and SSIM metrics. MSE of 0 and SSIM of 1 mean no difference.

3.3 CNN V1 model

The performance in stimuli synthesis of the convolutional models is superior to the previous two methods; as such, the weight of the MSE loss was increased to 10 to make it relevant during the training. The larger weight of the MSE loss during training allowed for faster convergence of the model parameters. Most of this effect came from the first epoch, as it allowed the model to quickly find some reasonable setting of its parameters such that the MSE would go down quickly. In the subsequent epochs, the SSIM loss component dominated the MSE component. Empirically the best performance was observed with 4 epochs of training, as by design, the CNN V1 model has very high capacity, meaning that it can quickly overfit on the training dataset. The advantage of using a convolutional model with high capacity in this task is that combined with a small number of epochs which serve as early stopping, much better generalisation performance can be achieved than with a smaller model version and more training epochs.

Optimizer	AdamW
Learning rate	1e-3
Batch size	4
Weight decay (l2 regularization)	1e-4
Training loss functions	MSE, SSIM
Loss weights	$10, 1$
Number of epochs	4

Table 3.7: Hyperparameters used for the CNN V1 model training.

3.3.1 Training results

The training results of the CNN V1 model displayed in the table 3.8 are substantially better than those of the MLP model 3.5. This indicates that more advanced feature extraction, such as the one in the CNN V1 model, is necessary to predict stimuli from the cortical activities. Another interesting point is that the model seems to learn most of its encodings during the first epoch, which indicates that a more extensive training dataset might help with the generalisation performance, as subsequent epochs improve the validation performance only marginally, even though the training loss went down, meaning that the model could successfully learn to predict stimuli from the training set. At the same time, it relatively struggled to generalise onto the validation set.

Epoch	MSE	$\mathcal{L}_{SSIM}$
1	0.020	0.603
2	0.019	0.585
3	0.017	0.585
4	0.018	0.578

Table 3.8: CNN V1 model training results on the validation set after each epoch.

3.3.2 Activity comparison

The results in the table 3.9 paint the picture that even the performance of the CNN V1 model at synthesising stimuli is not enough to invoke cortical activity A_{cnnv1} similar to the activity A_{test}. This might indicate that more precise stimulus reconstruction is required to invoke chosen cortical activity responses.

$\mathcal{L}_{SSIM}$	0.471
MSE	0.036
Mean A_{cnnv1} SSIM	0.531
Mean A_{test} SSIM	0.417
Mean A_{cnnv1} MSE	0.032
Mean A_{test} MSE	0.039

Table 3.9: Average difference between A_{cnnv1} and A_{test} described by MSE and SSIM metrics. MSE of 0 and SSIM of 1 mean no difference.

3.4 CNN V2 model

For similar reasons as in the CNN V1 model, the MSE loss weight was increased to 50. Apart from that, standard values of hyperparameters were chosen 3.10.

3.4.1 Training results

Compared to the CNN V1 model, the CNN V2 model training seems even more stagnant after the first epoch, only oscillating slightly. For the final activity

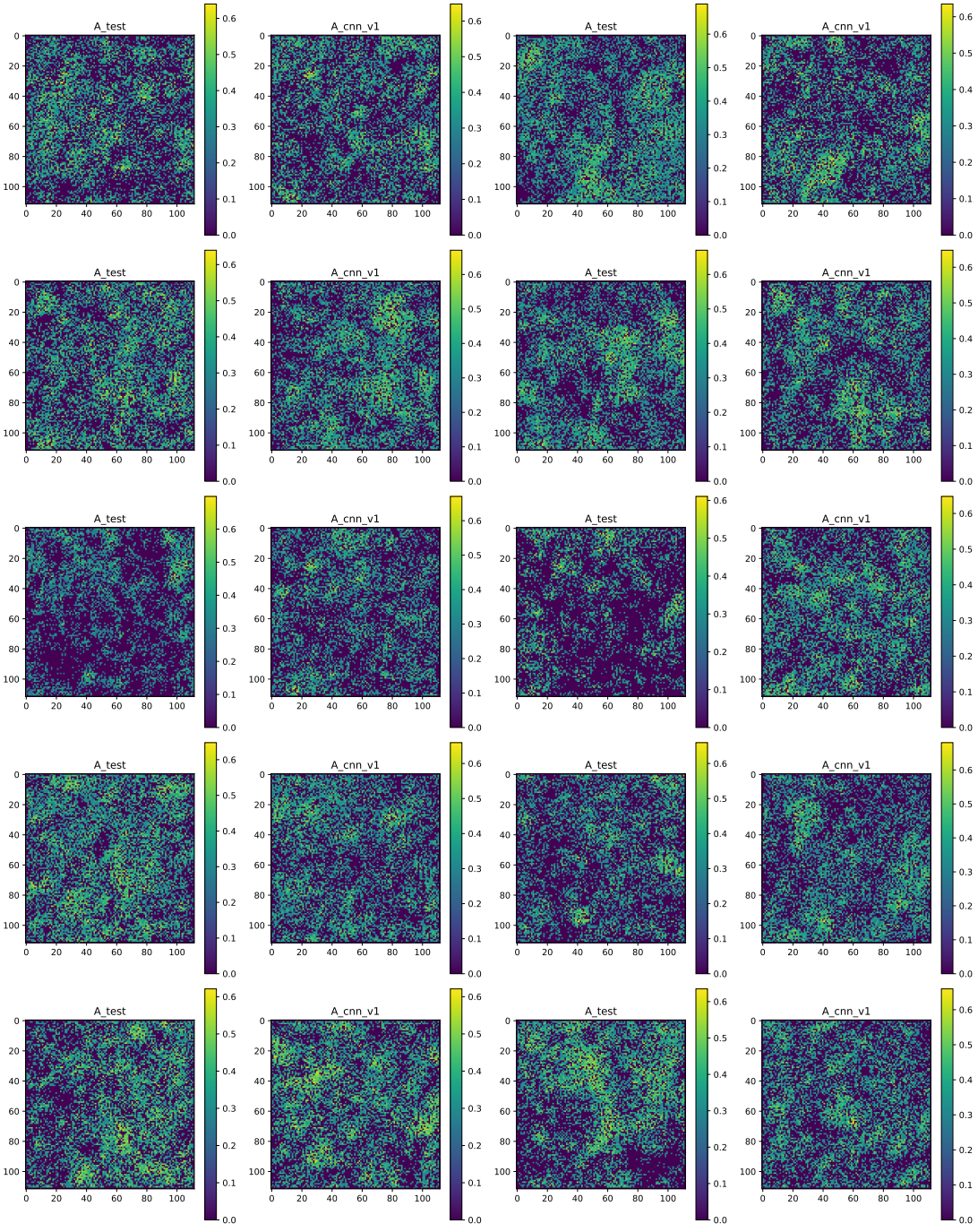

Figure 3.7: Comparison of the first 10 activity tensors from A_{test} and A_{cnnv1}. The first and second columns form the first 5 pairs, while 3rd and 4th columns form the second five pairs.

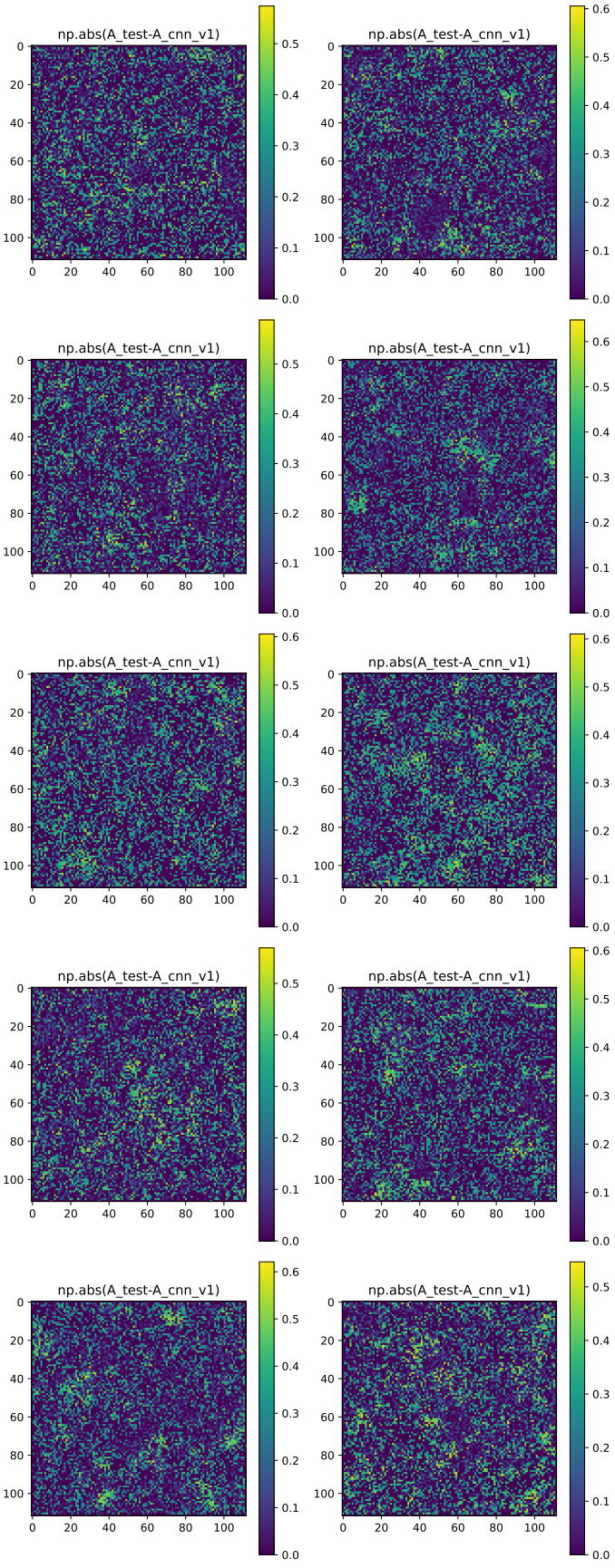

Figure 3.8: Absolute difference (error map) of the first 10 activity tensor pairs from A_{test} and A_{cnnv1}.

Optimizer	AdamW
Learning rate	1e-3
Batch size	4
Weight decay (l2 regularization)	1e-4
Training loss functions	MSE, SSIM
Loss weights	$50, 1$
Number of epochs	5

Table 3.10: Hyperparameters used for the CNN V2 model training.

comparison, model weights after the 3rd epoch were taken, even if only marginally the validation performance of $\mathcal{L}_{SSIM}$ 0.572 was the best out of all the epochs.

Epoch	MSE	$\mathcal{L}_{SSIM}$
1	0.016	0.574
2	0.016	0.573
3	0.016	0.572
4	0.016	0.574
4	0.016	0.575

Table 3.11: CNN V2 model training results on the validation set after each epoch.

3.4.2 Activity comparison

Out of all the models investigated in this thesis, the model CNN V2 showed the best MSE result regarding activity comparisons. This may be caused by the fact that the average difference between activities A_{cnnv2} in both MSE and SSIM metrics is the highest across all the models, indicating more entropy information in the A_{cnnv2} distribution.

$\mathcal{L}_{SSIM}$	0.474
MSE	0.032
Mean A_{cnnv2} SSIM	0.447
Mean A_{test} SSIM	0.417
Mean A_{cnnv2} MSE	0.037
Mean A_{test} MSE	0.039

Table 3.12: Average difference between A_{cnnv2} and A_{test} described by MSE and SSIM metrics. MSE of 0 and SSIM of 1 mean no difference.

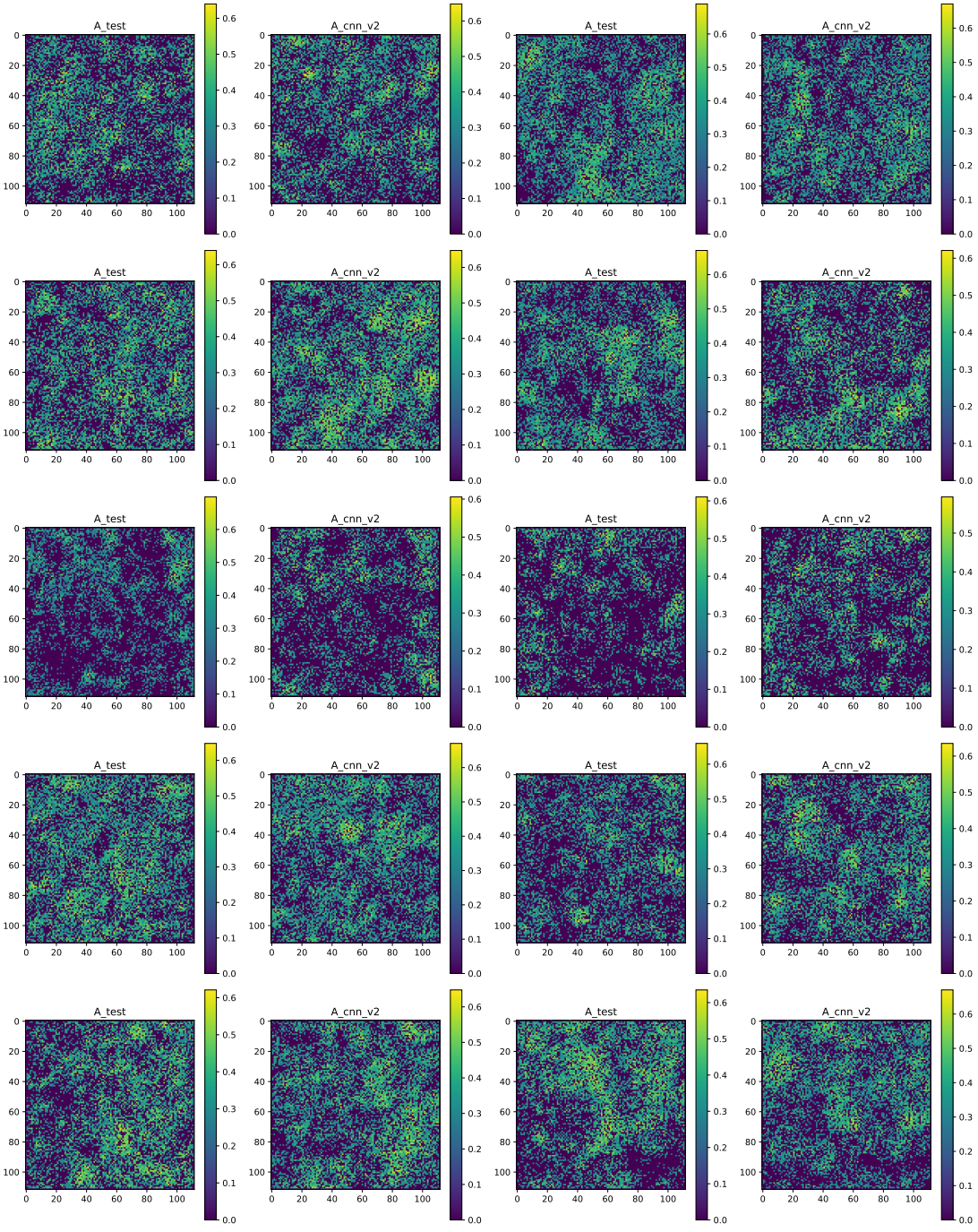

Figure 3.9: Comparison of the first 10 activity tensors from A_{test} and A_{cnnv2}. The first and second columns form the first 5 pairs, while 3rd and 4th columns form the second five pairs.

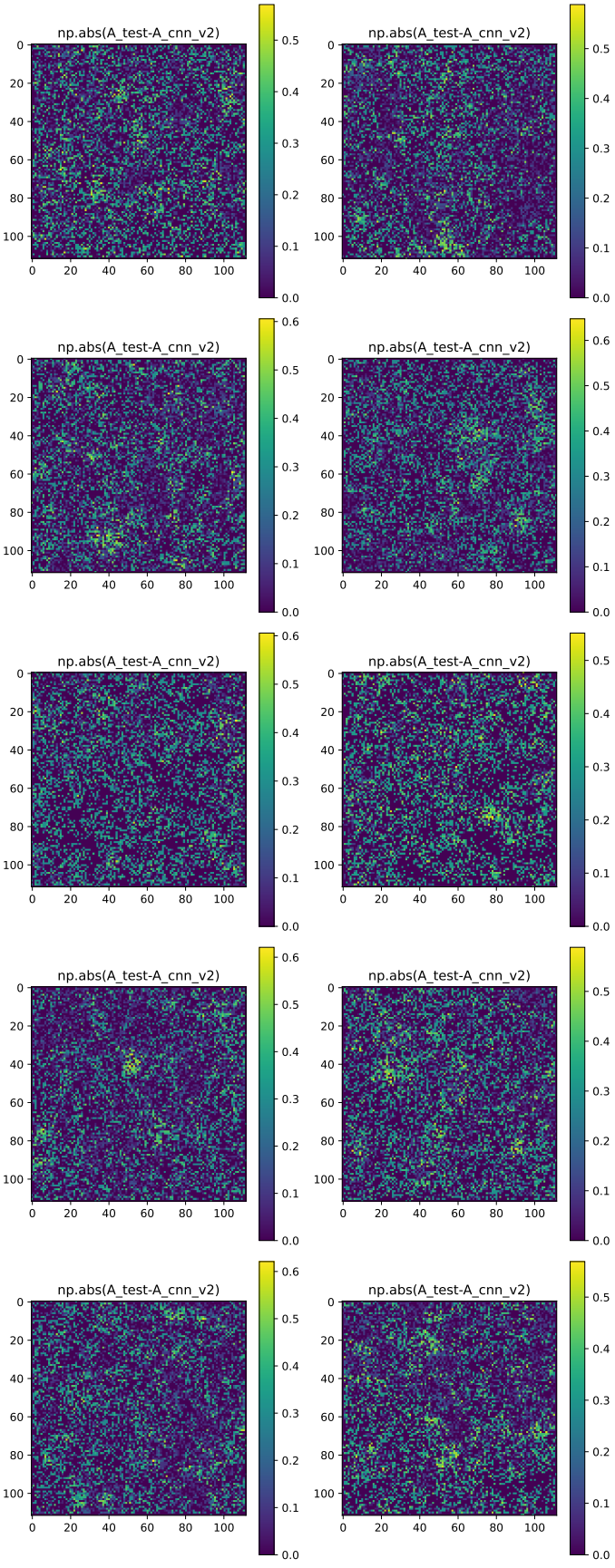

Figure 3.10: Absolute difference (error map) of the first 10 activity tensor pairs from A_{test} and A_{cnnv2}.

4. Discussion

4.1 Possible improvements

The model training results show that further improving the stimuli prediction is necessary for better generalisation performance. A manual look at example predictions such as 4.1 or 4.2 indicates that not only do the models struggle to match the overall dynamic range of the target stimuli, but they are also capable of synthesising only low-resolution features, meaning that any high-frequency data that is present in most stimuli is inevitably lost. Some of the possible causes for these phenomenons include:

1. Insufficient training data. The space of possible stimuli images and cortical activity patterns might be very heterogeneous, meaning that copious amounts of data would be required to cover a fraction of the possible stimuli and cortical activity patterns spaces. This hypothesis seems quite plausible, as mammalian vision is very advanced; it is capable of perceiving countless types of images from the environment; as such, it might be a futile effort to try reverse-engineering it with just 39000 pairs of (stimuli, cortical activity) pairs.

   Another reason there might be insufficient training data is that most image synthesis algorithms require big datasets to achieve good results, as high-frequency details require much more examples than low-frequency image features. This factor is disconnected from the previous hypothesis, as it might be relatively easy to distinguish various types of cortical activity patterns. Still, it is tough to synthesise highly detailed stimuli images with such a small dataset.

   The training data depends heavily on the computational resources and the amount of software friction with simultaneously simulating many experiments. For example, running more than 1000 simulations in parallel requires distributed strategies for disk management, as some simulations may crash if they cannot obtain write permission to hard-disk for a long time. Both of these factors played a significant role in creating the dataset.

2. The CNN architecture is fundamentally flawed in the settings of this task. The CNN architecture was chosen for its historical significance in vision neuroscience research. Real neurons are much more complex, meaning that a more advanced model might be necessary to decode stimuli from cortical activity patterns successfully. For instance, vision transformers, which can focus attention only on the crucial parts, might be better suited for the task. Another consideration is that with CNN architecture, it takes time for all of the crucial pieces of information from the input to mix, meaning that CNN models usually have relatively high capacity. This may negatively impact the performance, as there is a limited amount of training data, and advanced convolutional models can remember all examples from the training set while not working well on unseen data.

   The last thing worth mentioning with the architecture design is that the

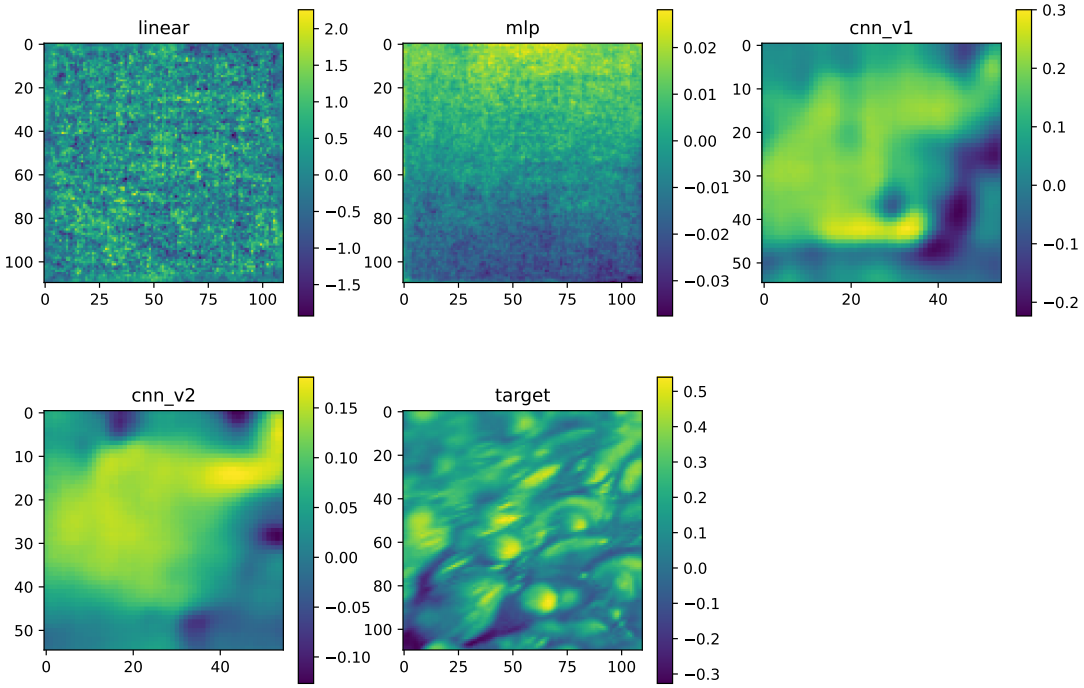

Figure 4.1: Example #1 of stimuli prediction from the test set.

encoder-decoder approach might not be the best overall. Still, a diffusion model that keeps the identical resolution throughout the activity to stimulus prediction pipeline would be more suitable, as the activity patterns can be viewed as a form of structured noise.

3. The loss functions used in this thesis give too weak signals for guiding the networks in the right direction. In the literature, countless loss functions focus on specific problems. For example, in the paper Ledig et al. [2017], they use both DEEP MSE loss and perceptual loss, allowing them to achieve much better results in image upscaling than with simple loss functions such as MSE. Specifically, image upscaling is deeply connected with the current model architectures; hence any process that can add high-frequency data into the predicted stimulus A_{model} might significantly improve the results.

4. Cortical implant is not able to stimulate the cortex sufficiently precisely. Suppose the cortical implant cannot achieve artificially evoked cortical patterns similar to those evoked through natural vision due to precision issues. In that case, it may indeed be impossible to match A_{nat} activity patterns with mappings generated on A_{art}, S_{art} pairs. Such imprecision would also explain why even if most of the low-level features of S_{art} are matched by S_{model}, the validation performance is abysmal for all model types, as the simulation may be too chaotic.

Solving these concerns may bring the dream of substituting a non-functional eye closer to reality.

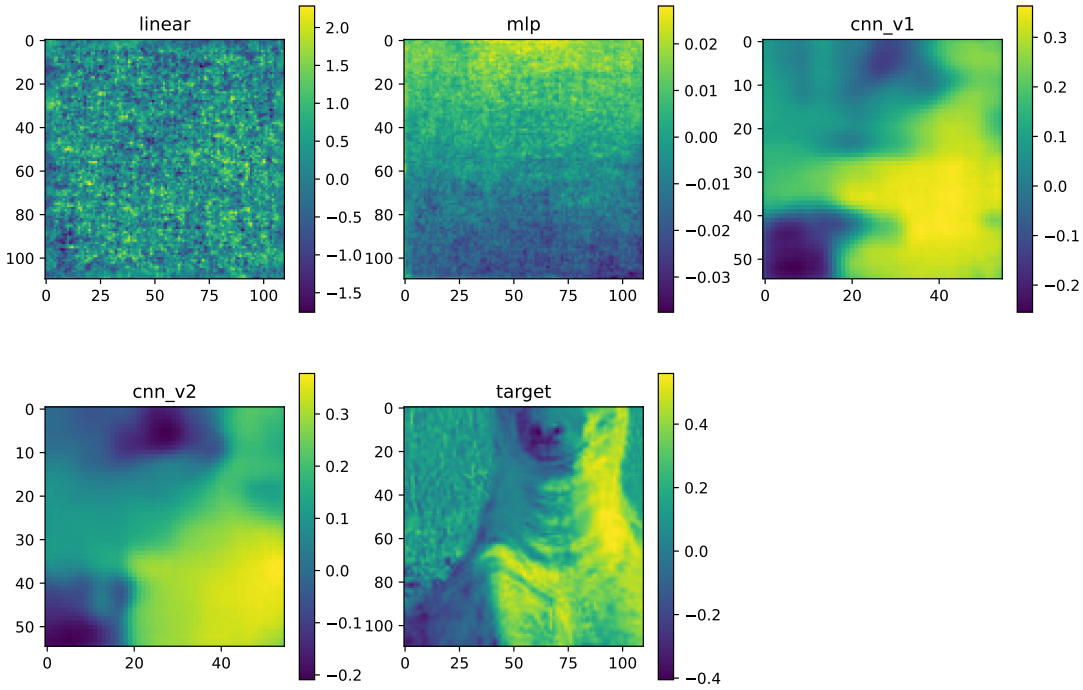

Figure 4.2: Example #2 of stimuli prediction from the test set.

4.2 The training set imbalance

Another aspect which might have negatively impacted the results is the sparsity of the dataset. Even after applying various operations to increase the variance of A_{art} activities used for training, a vast majority of A_{art} tensor cells are filled with values corresponding to a firing rate of 0. This issue might stem from an uneven sampling of neuron positions across the cortex in the Mozaik simulations. Averaging the data from multiple trials with various random seeds can relieve this issue and produce a more information-rich dataset.

4.3 Future work

Future work can involve taking cortical activities from actual A_{nat} distribution as a test set to validate whether the model performance indeed can generalise well from training on A_{art}, S_{art} pairs. Another approach that could be investigated in the future is to build an ensemble of models in boosting way, meaning that the first model would learn to produce artificial stimuli S_{model_1} as close to the gold targets S_{art}. In contrast, subsequent models would be trained on the differences between the gold labels S_{art} and the sum of previous models' predictions. An ensemble of weak learners might learn to model the complex distribution S_{art} from A_{art} surprisingly well. Besides these two points, all arguments mentioned in the 4.1 section must be investigated.

Conclusion

The whole focus of this thesis was on this single cornerstone challenge from the beginning: *Understanding how to manipulate the neural activity such that the artificially elicited cortical activity patterns match those elicited through viewing the target visual stimulus via an intact visual pathway.* The machine learning models investigated provide baselines for future research and shed some light on the issues with learning the innovative mapping between cortical activities and artificial stimuli. While the results presented are somewhat lacklustre, they helped pave the road for future research. Mainly the points raised in the discussion (chapter 4) provide valuable insights for future research.

One of the significant successes of this thesis was setting up various scripts that can help run the Mozaik framework on the Metacentrum infrastructure. This gives rise to the possibility of using more extensive and varied datasets.

Bibliography

Jia Deng, Wei Dong, Richard Socher, Li-Jia Li, Kai Li, and Li Fei-Fei. Imagenet: A large-scale hierarchical image database. In *2009 IEEE conference on computer vision and pattern recognition*, pages 248–255. Ieee, 2009.

Jan Antolík et al. Assessment of optogenetically-driven strategies for prosthetic restoration of cortical vision in large-scale neural simulation of v1. *Scientific reports*, 11:1–18, 2021.

Alexander Farnum and Galit Pelled. New vision for visual prostheses. *Front Neurosci*, 14:36, February 2020.

Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning*. MIT Press, 2016. `http://www.deeplearningbook.org`.

Dan Hendrycks and Kevin Gimpel. Gaussian error linear units (gelus), 2023.

Christian Ledig, Lucas Theis, Ferenc Huszar, Jose Caballero, Andrew Cunningham, Alejandro Acosta, Andrew Aitken, Alykhan Tejani, Johannes Totz, Zehan Wang, and Wenzhe Shi. Photo-realistic single image super-resolution using a generative adversarial network, 2017.

Zhichao Liu, Ruili Huang, Ruth Roberts, and Weida Tong. Toxicogenomics: A 2020 vision. *Trends Pharmacol Sci*, 40(2):92–103, December 2018.

Zhuang Liu, Hanzi Mao, Chao-Yuan Wu, Christoph Feichtenhofer, Trevor Darrell, and Saining Xie. A convnet for the 2020s, 2022.

Ilya Loshchilov and Frank Hutter. Decoupled weight decay regularization, 2019.

Martin Picek. Rotation-equivariant convolutional neural network for design of visual prosthetic stimulation protocol, 2022.

Wenzhe Shi, Jose Caballero, Ferenc Huszár, Johannes Totz, Andrew P. Aitken, Rob Bishop, Daniel Rueckert, and Zehan Wang. Real-time single image and video super-resolution using an efficient sub-pixel convolutional neural network, 2016.

Vojtěch Vašek. Decoding visual stimuli from cortical activity, 2023.

Z. Wang, A. C. Bovik, H. R. Sheikh, and E. P. Simoncelli. Image Quality Assessment: From Error Visibility to Structural Similarity. *IEEE Transactions on Image Processing*, 13(4):600–612, April 2004. doi: 10.1109/TIP.2003.819861.

List of Figures

List of Tables

List of Abbreviations

DNN	Deep neural network
MSE	Mean square error
SSIM	Structural similarity index metric
V1	visual area 1 in the visual cortex
MLP	multilayer perceptron
CNN	convolutional neural network
LN	layer normalisation
GELU	Gaussian Error Linear Unit