Robustness to Error in Autoencoders: MNIST and Large-Scale Simulation Stimuli

Olesia Altunina,
MSc in Life Science and Technology,
École polytechnique fédérale de Lausanne

Supervisors:
Dr. Eilif Muller,
Dr. Giuseppe Chindemi,
Francesco Casalegno,
Blue Brain Project

Introduction

Fetching latent low-dimensional representations from high-dimensional data has been long known to facilitate the performance of the machine learning (ML) models used to solve classification tasks. A particular example of such fetching rooted in the concept of artificial neural networks (ANNs) is an autoencoder, which aims at minimising the difference between the actual input and the reconstruction of this input from a lower-dimensional code, which the model learns using the back-propagation algorithm (Ballard 1987).

An autoencoder consists of two generally independent modules, an *encoder*, which transforms a high-dimensional input vector into a low-dimensional *representation* (or *code*), and a decoder, which restores the original vector from this representation. The loss function used for optimisation is some kind of a difference between the decoded and the original vectors, for example, mean squared error (Bengio, Courville, and Vincent 2013). However, if the inputs are bound, like images, which contain values from 0 to 255, binary cross-entropy would be a better choice to ensure similarly bounded reconstructions. Activation functions used throughout the network, are typically ReLUs, except for the lest layer. For the inputs which are continuous in nature, the output layer has linear activations, whereas sigmoid better suits binary data.

To date, there exist a few variations of autoencoders, including deep, sparse, convolutional, denoising, contractive and variational (Goodfellow 2016), which emphasise different properties to get relevant representations for each particular task.

Autoencoder learning is considered unsupervised or self-supervised in a sense that it requires no external labelling. However, just by training an autoencoder, one might get representations that restore the data well but fail at the task for which the compression was needed in the first place. In this respect, good representations are the representations that yield good results at the task for which they are used. To fine-tune representations to the task at hand, a pre-trained encoder part of the autoencoder can be stacked with the model of interest and trained together. For example, this can be used to boost performance of the autoencoder-based classifiers.

Large scale simulations of the brain represent an evolving field which aims at reproducing the brain's fundamental properties within the framework of a computational model. Sturdy methods of assessing the qualitative properties of such models are now crystallising, and new approaches are being considered. To evaluate learning properties of the model, three general indicators are used: invariance, pattern completion and error correction.

In this project, we focus on the error correction property, which implies tolerance to the distorted input, checking which autoencoder architectures satisfy this property. We chose an autoencoder framework because the large scale simulation model, which is an isolated model of 7 columns of

the sensorimotor cortex, implies unsupervised learning: it is been presented with artificially generated sequences of inputs, each being a noisy version of a particular pattern, and in the course of training, due to plasticity mechanisms involved, the model learns to give distinct output depending on the pattern to which the current input belongs, i.e. it transforms the input into the definite output for the corresponding pattern.

The way the performance of the model is estimated, is by comparing the trained model with the naive in terms of the learned parameters and the output produced. In the case of error correction paradigm, the distorted inputs are presented to the system, and the outputs are compared to those produced by the original data. The system has the error correction property, if, up to some degree of distortion, it produces the same output as if the error was not introduced.

In terms of ANNs, we use an autoencoder stacked with a classifier, which is hopefully similar to the readout of the model, and assess whether it satisfies the error correction property and find the architecture that yields the best results in this respect. For this we use MNIST as a conventional “default” dataset to start with, and then apply the same approach to the stimuli generated for the model described above.

Methods

This project was implemented in Python 3.6.3. ANNs were built and trained in Keras 2.2.4. Visualisations were done in Matplotlib 2.0.2. For storage purposes, Pickle 4.0 was used. Also, we used Numpy 1.15.4. A helper class AppendedHistory was used for recording and storing learning histories. The project folder contains .ipynb files which constitute the main part of the project, .py helpers, the input data for the stimuli part and the output data such as the trained models, learning histories and relevant visualisations.

MNIST

Data

In this part, the aim was to analyse various autoencoder architectures in terms of their robustness to missing pixels in the MNIST dataset [ref], which contains 28x28 images of handwritten digits, with 60 000 training and 10 000 testing samples. The error was introduced by randomly sampling (784 - missing_pixels) pixels without replacement in each image. The number of missing pixels was from 0 to 784.

Autoencoders

The code for this part is based on this Keras tutorial, which features a few commonly used autoencoder architectures. The following autoencoder architectures were tested:

- Sparse 784-1000-784
- Simple 784-10-784
- Deep (symmetric, hidden layer sizes are equidistant and depend on the number of hidden layers)
  - 784-397-10-…
  - 784-526-268-10-…
  - 784-590-397-203-10-…
  - 784-629-474-319-164-10-…
- Convolutional (symmetric with upsampling)
  - Convolution: 16 filters 3x3
  - Max-pooling 2x2
  - Convolution: 8 filters 3x3
  - Max-pooling 2x2
  - Convolution: 8 filters 3x3
  - Max-pooling 2x2
- Denoising (symmetric with upsampling)
  - Convolution: 32 filters 3x3

- Max-pooling 2x2
- Convolution: 32 filters 3x3
- Max-pooling 2x2

For MNIST, activations are bounded ([0, 255], normalised to [0, 1]) and non-binary in nature. Therefore, for autoencoders, we used sigmoid activation function for the output layer with mean squared error, as discussed in the introduction. The activation function for the hidden layers was ReLU. Each model was trained for 5 epochs with batch_size=32, which provided sufficient retrieval.

To assess the relative quality of the represented architectures, R^2 statistic was used. Pixels of the output vectors (pictures) were treated as outputs of regression models. R^2 value of explained variance (Wikipedia) was calculated for each pixel throughout the dataset using the following formula:

$$R^2 \equiv 1 - \frac{SS_{\text{res}}}{SS_{\text{tot}}}, \text{ where}$$

$$SS_{\text{tot}} = \sum_i (y_i - \bar{y})^2,$$

$$SS_{\text{res}} = \sum_i (y_i - f_i)^2 = \sum_i e_i^2,$$

$$\bar{y} = \frac{1}{n} \sum_{i=1}^{n} y_i \,.$$

Positive values of R^2 were kept and visualised, negative values (including $-\infty$) put to zero. The information about the amount of negative values can be derived from $N_{\text{pos}}/N_{\text{tot}}$ ratio, which accompanies each relevant plot.

Readout

To assess the robustness to missing pixels, we used a logistic regression classifier built on top of the autoencoder. We considered two cases: when the classifier was trained on the encoded representations of the autoencoder (with autoencoder's weights frozen) and when the classifier was built on top of the encoder, yielding fine-tuning of the encoder weights as well as classification training.

Large-scale Simulation Stimuli

Data

In this part of the project, we compare the results from the first part with the same types of architectures implemented for the input dataset generated for the large-scale simulation of 7 columns of the sensorimotor cortex. The spacial structure of the input is defined with a set of 2D coordinates, each of which represents a specific dendritic fibre. All the fibres have their unique global IDs (GIDs).

Two initial files were given: project_xz.pkl file contains a dictionary with the GIDs as keys and their respective coordinates (list of 2 floats) as values; input.dat file contains a list of spiking events, each of which is presented as a list containing the time from the beginning of the simulation in ms and the corresponding GID. Starting from 1 s, the meaningful spiking begins, each 200 ms representing a single input frame (presentation instance). To convert the original data into a dataset suitable for training of an ANN, we binned all the spiking events within 200 ms intervals skipping the first second, summed up all spikes for each fibre within each bin and normalised the dataset by dividing it with its maximum value. After that, one-hot encoding has been performed, yielding 145 2170-dimensional vectors of the input data. The data is labeled into 8 classes, each corresponding to a particular activity pattern. The instances of each pattern is a bit different from each other, allowing some variation into the dataset.

Autoencoders

The following autoencoder architectures were tested in this part:

- Sparse
  - 2170-8-2170
  - 2170-16-2170
  - 2170-32-2170
  - 2170-64-2170
  - 2170-128-2170
  - 2170-512-2170
- Simple 2170-8-2170
- Deep (symmetric, hidden layer sizes are equidistant and depend on the number of hidden layers)
  - 1 hidden layer
  - 2 hidden layers
  - 3 hidden layers
  - 4 hidden layers

Convolutional approach could also be implemented, however, due to the fact that spacial structure cannot be exactly represented using pixel notation (integer coordinates), one has to create a new dataset of images suitable for subsequent training.

Results
MNIST

First, to grade the performance of the autoencoders in the terms other than their loss values, let us look at the R^2 scores distribution throughout the pixels of the output images (Fig. 1). From the figure, we can see that all of the models explain the variance of the dataset quite well, with the relevant area (where the digits are placed) covered and with mean values of R^2 from 0.573 in the deep autoencoder with two hidden layers to 0.869 in the denoising autoencoder. The models that have the best R^2 scores are denoising, sparse and convolutional autoencoders.
If we now look at Fig. 2, where the original and restored images are shown, we can see that these same models yield the decoding that in the sharpest and closest to the original. This results, however, do not correlate with the loss values (Fig. 3) that we get after 5 epochs of training. Loss values, therefore, are not quite as telling as the R^2 score.

Next, let us compare the performance of the readout models with frozen weights and with fine-tuning. Looking at the values for 0 missing pixels in Fig. 5, we can notice that there is a significant variance in the model accuracies for the frozen weights case, while in the fine-tuning case, the initial performances all lie close to 1 (except for the simple autoencoder, which naturally does the poorest).

In terms of the robustness, the best model with frozen weights is the autoencoder with one hidden layer, which outperforms the readout on the input at about ~440 missing pixels. With fine-tuning, all the deep models seem to perform better than the readout in the input up to ~700 missing pixels, and even after that they do just as well as the input readout. The sparse autoencoder also does quite well here, outperforming the baseline up until ~600 missing pixels.

Notably, all the deep models (one to four hidden layers) had similar performance, especially in the case of fine-tuning.

Surprisingly, the convolutional models, including denoising autoencoder, which had the best R^2 score, turned out to have the worst robustness in the fine-tuning case, though the convolutional model performed about the same as the deep models with the weights frozen (unlike denoising autoencoder, which had the steepest drop in both cases).

Fig. 1: R^2 values distribution for each autoencoder model

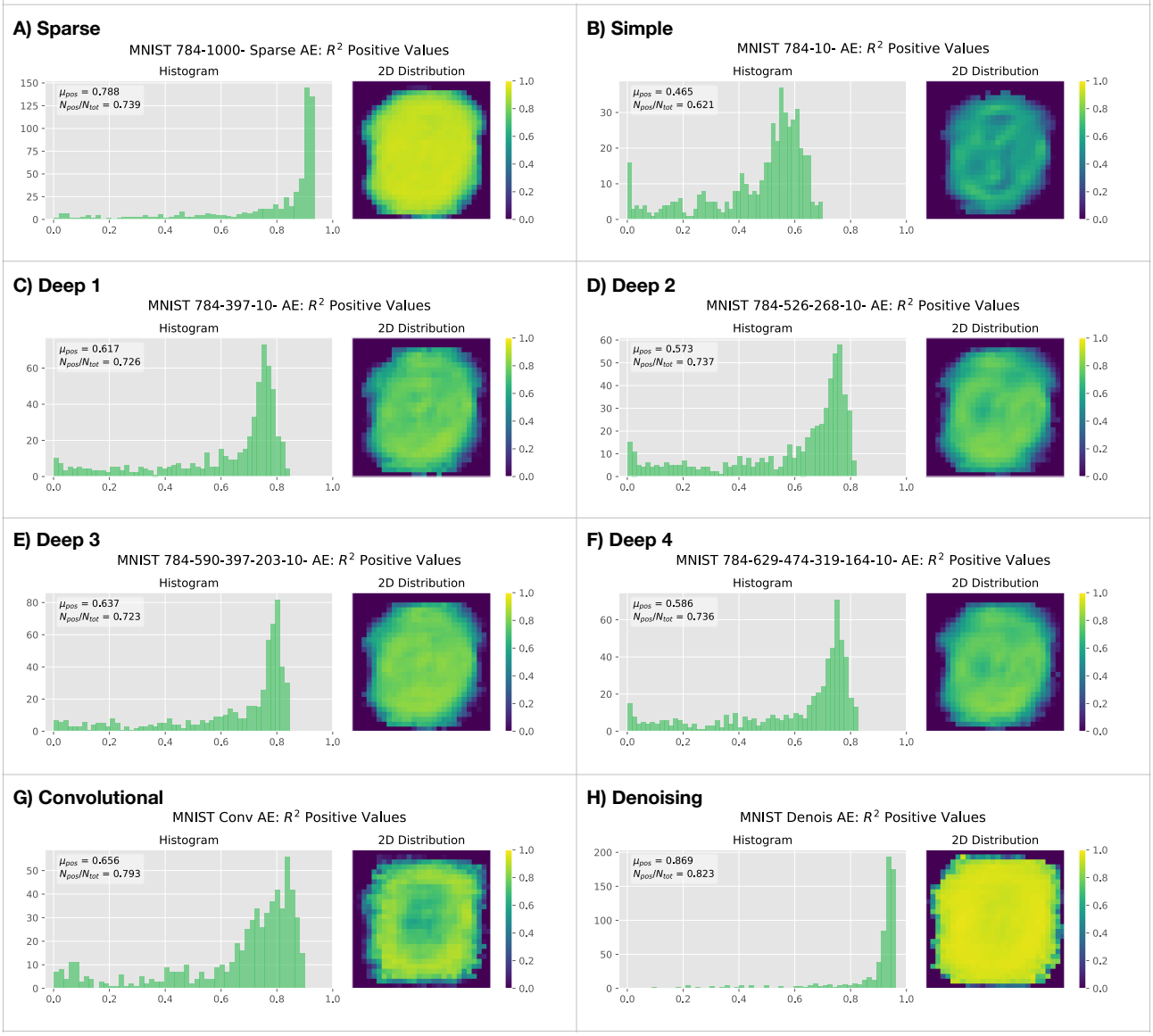
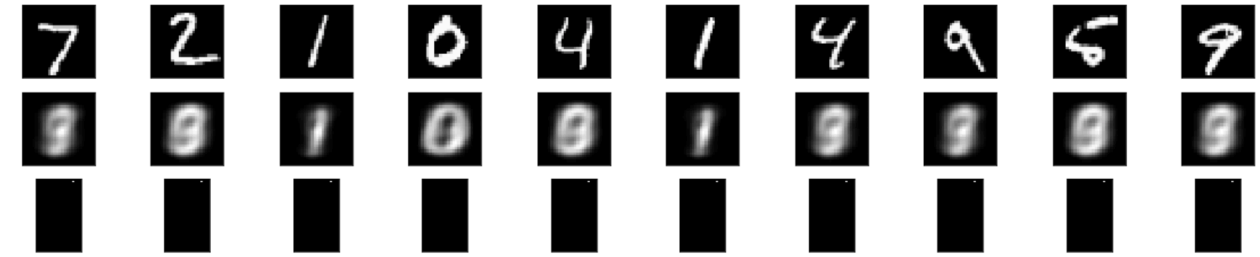


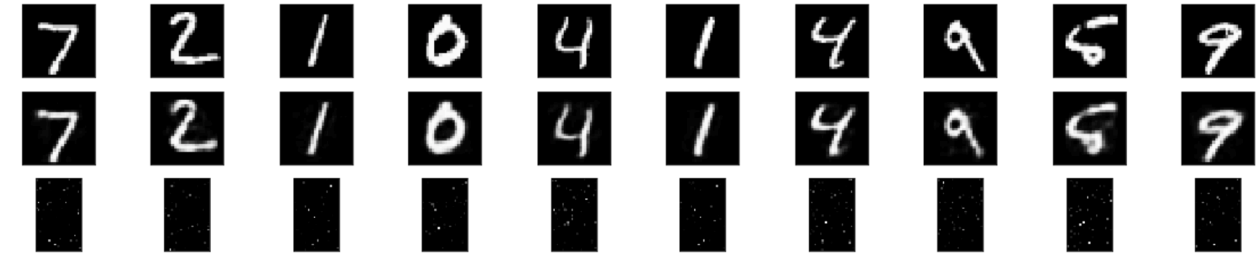
Fig. 2: 10 random original inputs, reconstructed inputs, and representations for each autoencoder model

A) Sparse

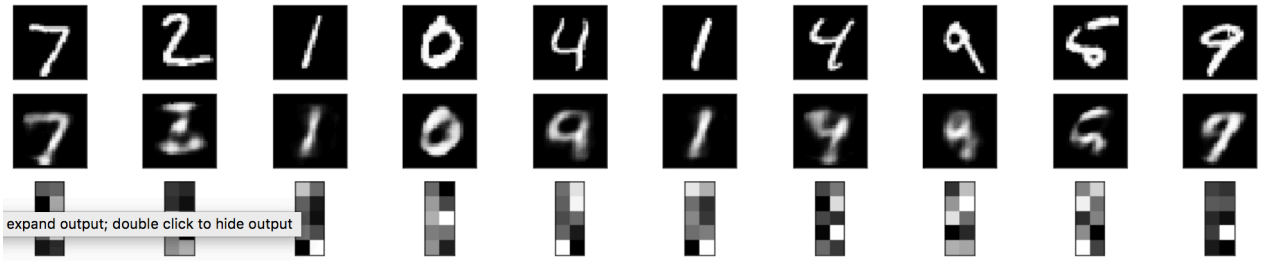
`sparsity = 0.0001`



`sparsity = 1e-05`

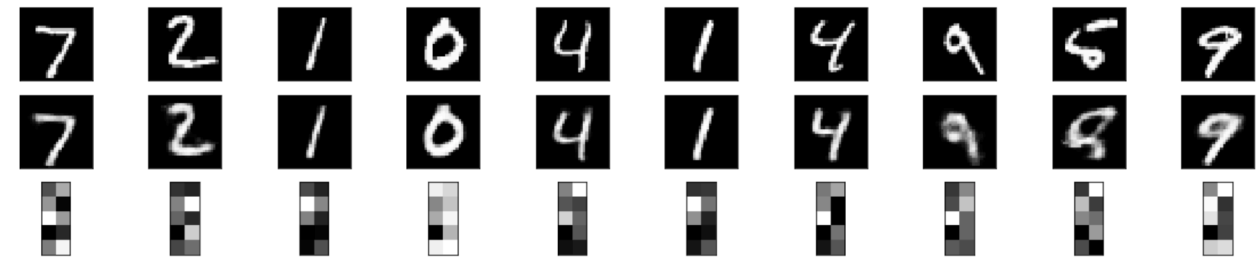


B) Simple

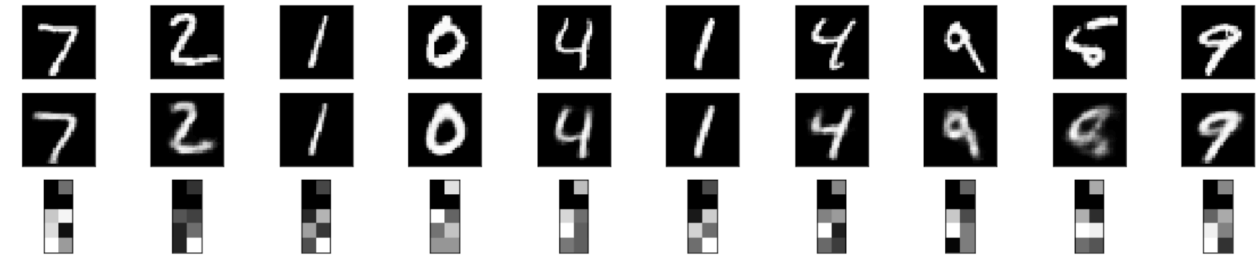


C, D, E, F) Deep

`num_hid = 1`



`num_hid = 2`



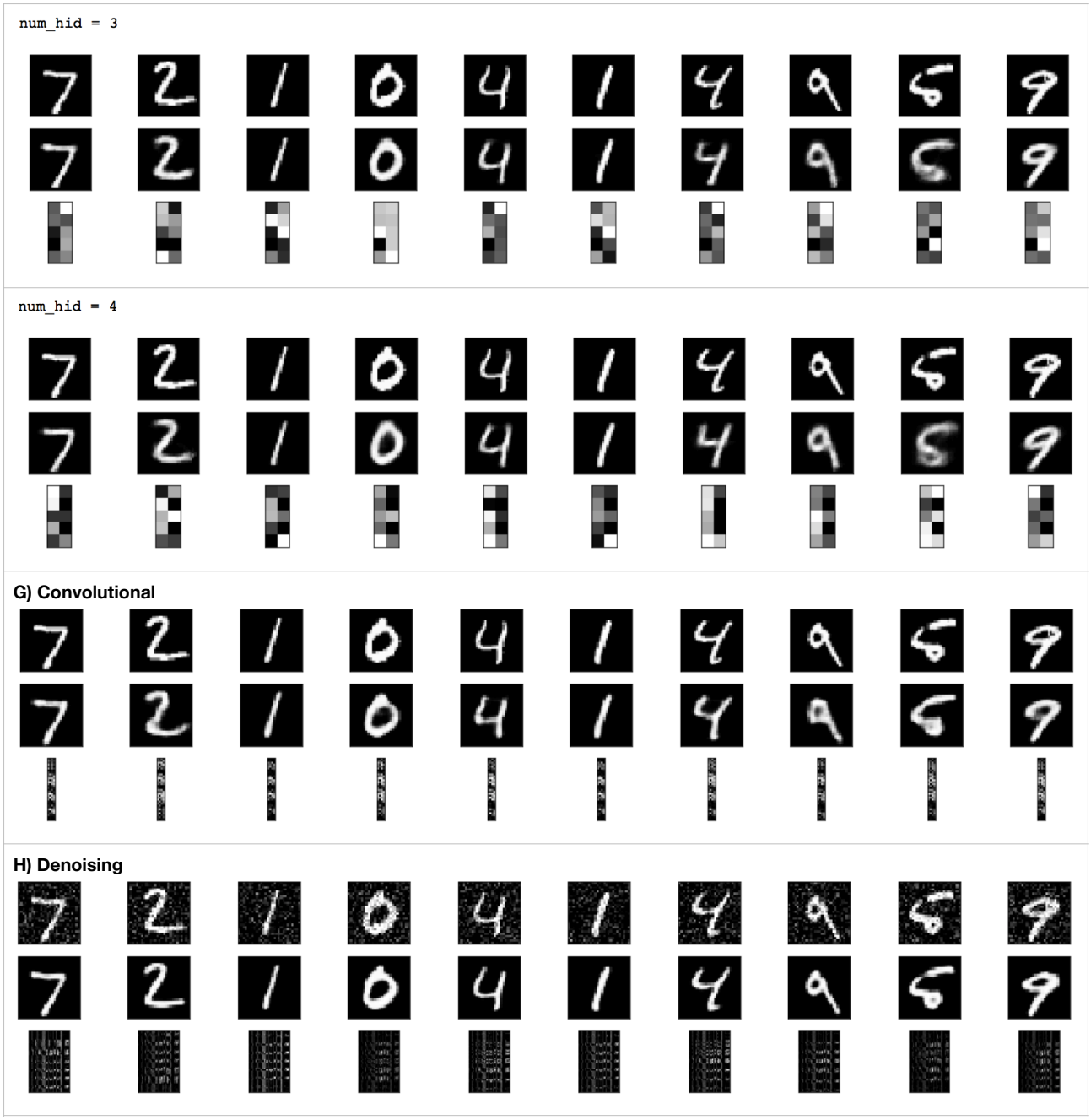
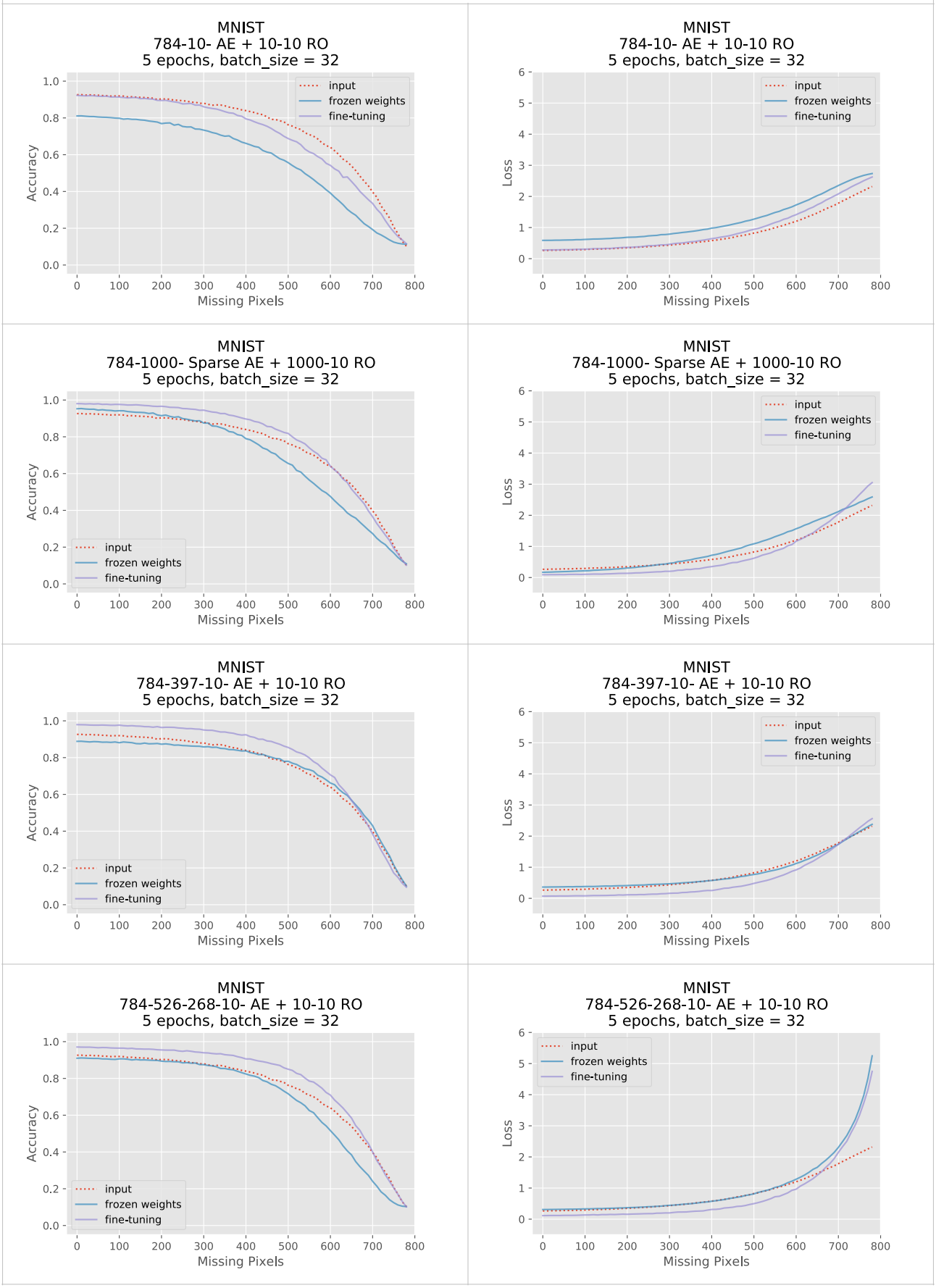
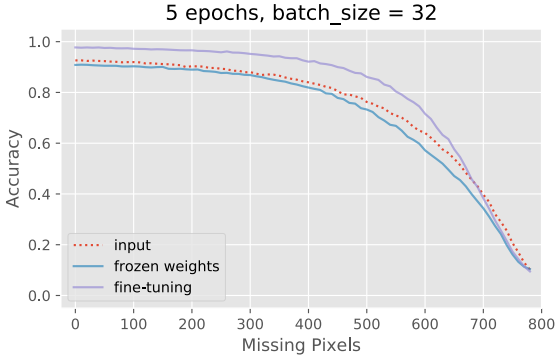


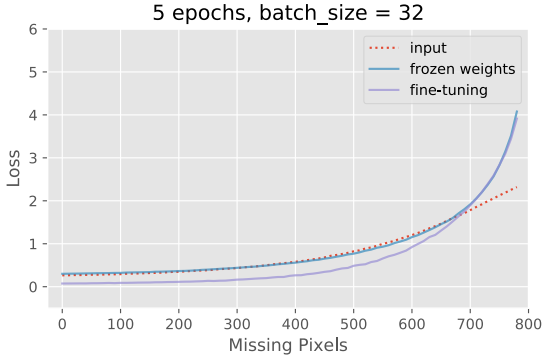
Fig. 3: Loss values of autoencoder models

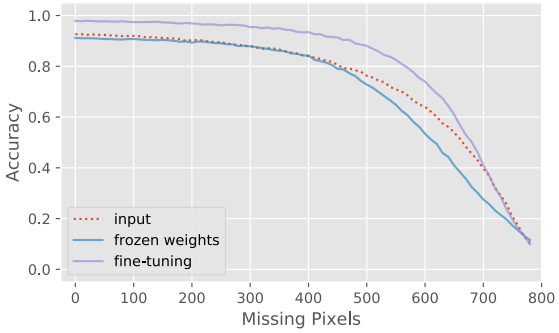
	Loss	
Autoencoder	**Training**	**Validation**
Sparse	0.0119	0.0116
Simple	0.0306	0.0300
Deep 1	0.0169	0.0167
Deep 2	0.0185	0.0184
Deep 3	0.0149	0.0149
Deep 4	0.0182	0.0177
Convolutional	0.1113	0.1074
Denoising	0.0746	0.0731

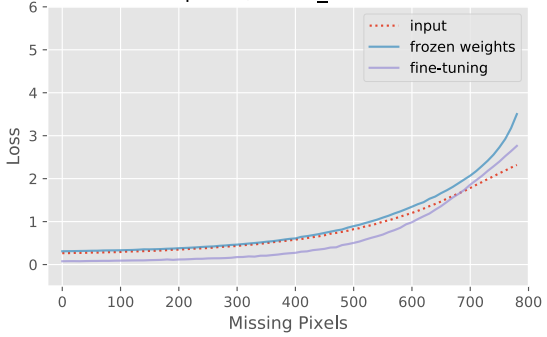
Fig. 4: Accuracy and loss of each autoencoder model with readout as a function of missing pixels

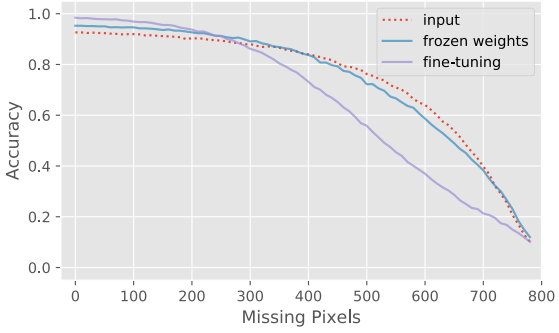


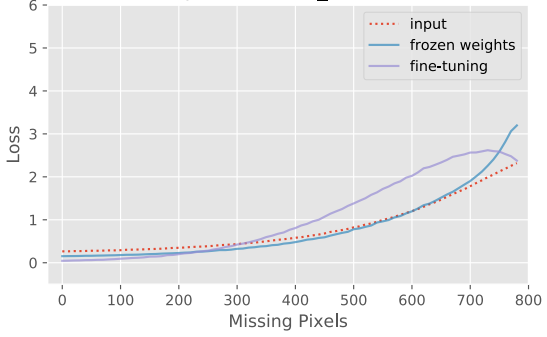

MNIST
784-590-397-203-10- AE + 10-10 RO
5 epochs, batch_size = 32
Accuracy
Missing Pixels
input
frozen weights
fine-tuning

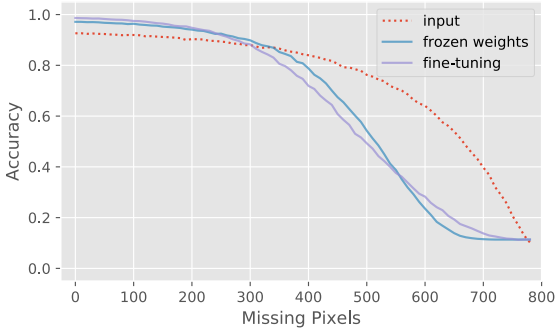

MNIST
784-590-397-203-10- AE + 10-10 RO
5 epochs, batch_size = 32
Loss
Missing Pixels
input
frozen weights
fine-tuning

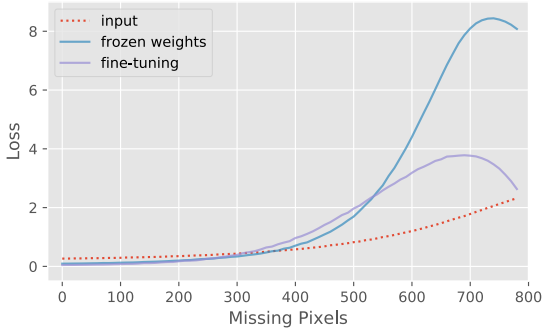

MNIST
784-629-474-319-164-10- AE + 10-10 RO
5 epochs, batch_size = 32
Accuracy
Missing Pixels
input
frozen weights
fine-tuning


MNIST
784-629-474-319-164-10- AE + 10-10 RO
5 epochs, batch_size = 32
Loss
Missing Pixels
input
frozen weights
fine-tuning


MNIST
Conv AE + 4x4x8-10 RO
5 epochs, batch_size = 32
Accuracy
Missing Pixels
input
frozen weights
fine-tuning


MNIST
Conv AE + 4x4x8-10 RO
5 epochs, batch_size = 32
Loss
Missing Pixels
input
frozen weights
fine-tuning


MNIST
Denois AE + 7x7x32-10 Noisy RO
5 epochs, batch_size = 32
Accuracy
Missing Pixels
input
frozen weights
fine-tuning


MNIST
Denois AE + 7x7x32-10 Noisy RO
5 epochs, batch_size = 32
Loss
Missing Pixels
input
frozen weights
fine-tuning

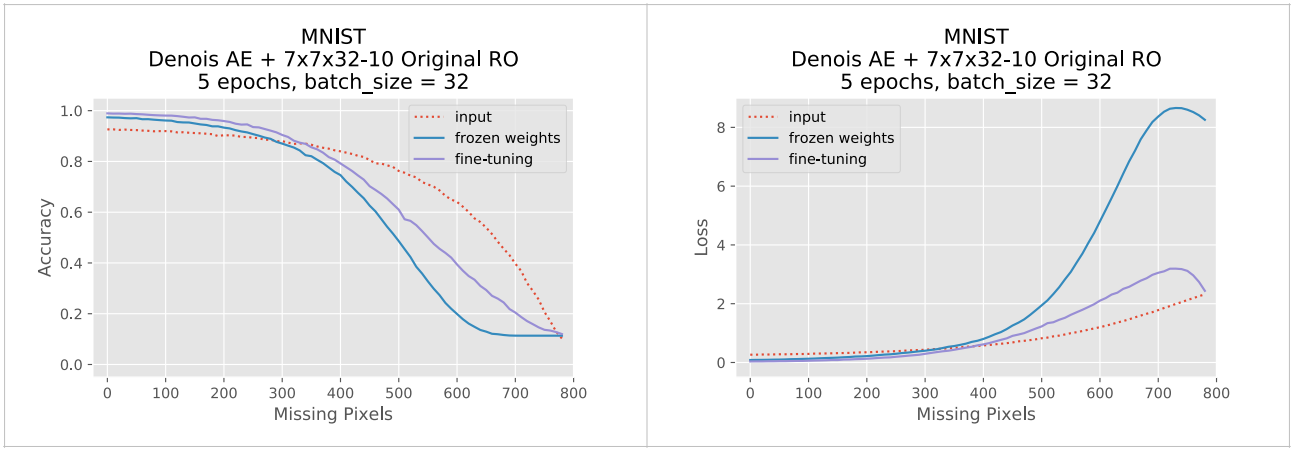
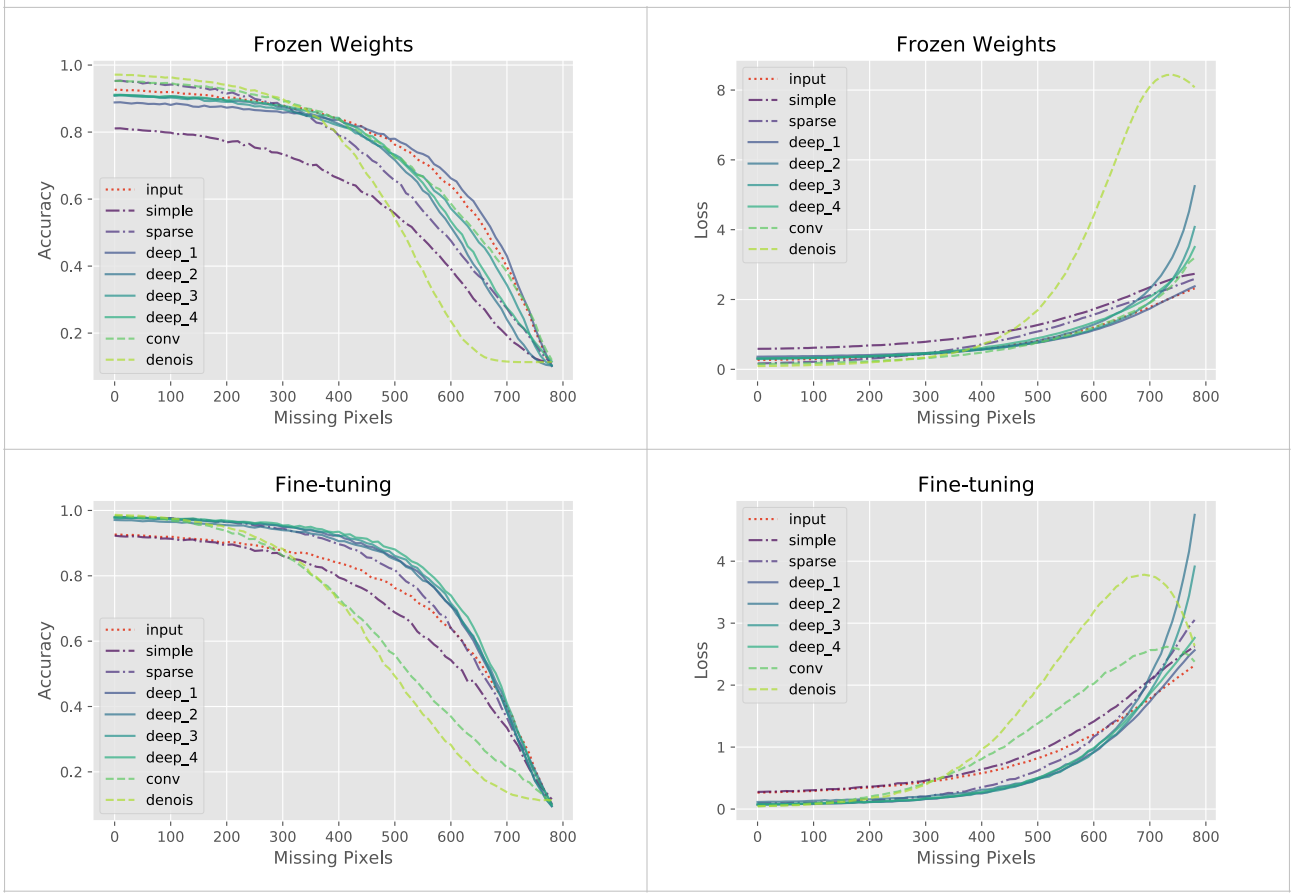


Fig. 5: Summary of robustness plots (Fig. 3)



Large-Scale Simulation Stimuli

First, in Fig. 1A an example of a pattern is presented. To get this picture, we summed all the vectors corresponding to a particular pattern and then normalised the result. Thus we get the blobs that are active in each instance of the given pattern versus the background noise of the arbitrarily activated blobs.

In Fig. 1B, we give an instance of representation values distributions for the pattern presented in Fig. 1A. The dynamics of encoding the input data with missing blobs with the same autoencoder is given in Fig. 1B1, B2. Notable, the more blobs are missing, the closer to zero the representation values (which should be the case because the more missing blobs, the more zeros are in the input vector).

Fig. 1 C, D are to show that there is not much of a difference in the performance of a simple autoencoder with the bottleneck sizes of 8 and 128, neither qualitatively nor statistically, although the two sizes were chosen for the relative difference in the loss value after training.

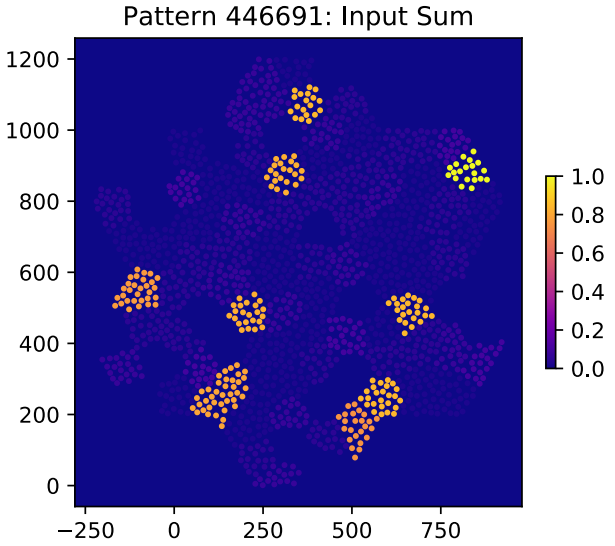
The robustness of the models was estimated using the readout logistic classifier, as in MNIST case. The summary table of the results is presented in Fig. 2.

First, we see that no autoencoder model is more robust that a simple logistic classifier trained on the input. Moreover, if we look at the deep autoencoder models, we can notice that the more hidden layers in the autoencoder, the less performing and robust the readout. This suggests that the data itself is separable with a single hyperplane, so more hidden layers only complicate the matters.

Second, it is worth noticing that the variance in the robustness for the sparse and simple autoencoders is quite high and depends on the number of missing blobs, with higher numbers having more variance in for both the input readout and the readouts on the autoencoders. For deep autoencoders this tendency is not the case.

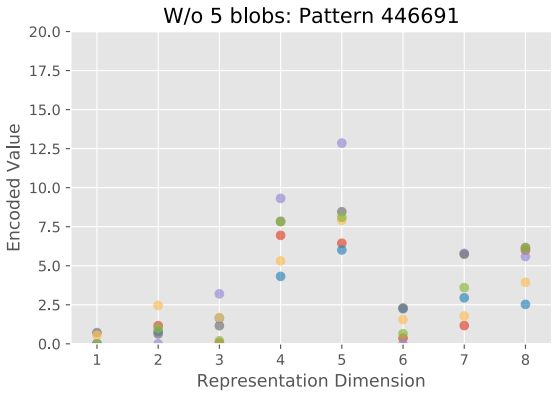
Sparse 2170-128- autoencoder's performance in terms of robustness is the closest to the one of the baseline.

Fig.1: Qualitative description of an autoencoder performance on the stimuli dataset

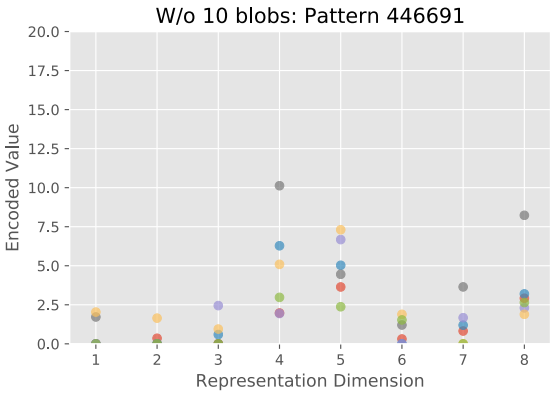


A) Sum of all instances of a single pattern

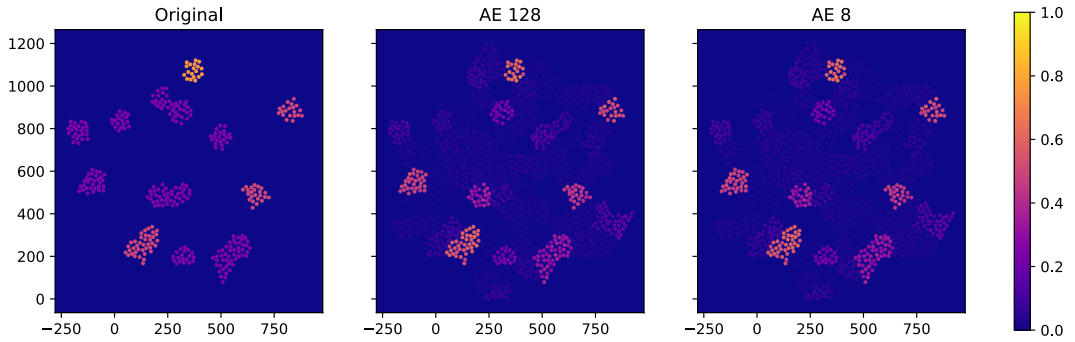
B) Values distributions for 8 representations of a single pattern



B1) Same distributions when 5 blobs are missing



B2) Same distributions when 5 blobs are missing



C) Example of the quality of the learning: patterns decoded with a simple autoencoder

```
np.mean(x_test), np.mean(x_test_decoded), np.mean(x_test_decoded_2)
```

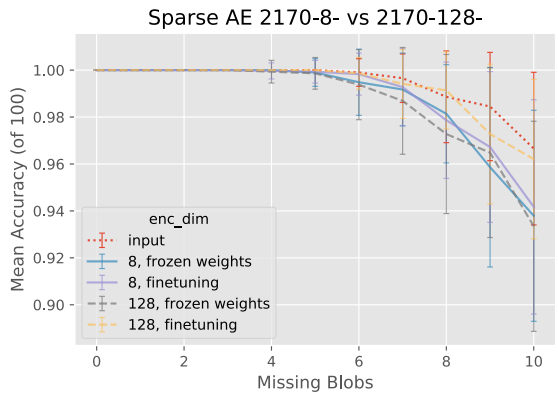
```
(0.062176227554425555, 0.07088892, 0.07088892)
```

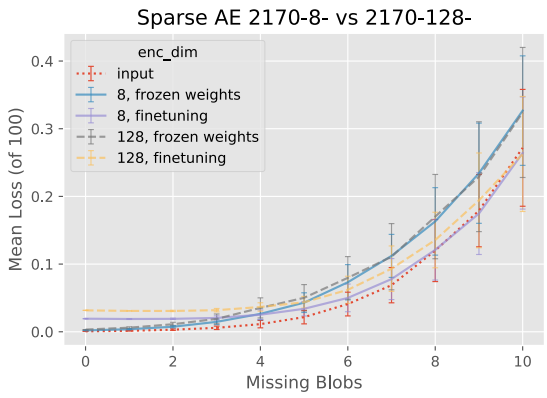
```
np.std(x_test), np.std(x_test_decoded), np.std(x_test_decoded_2)
```

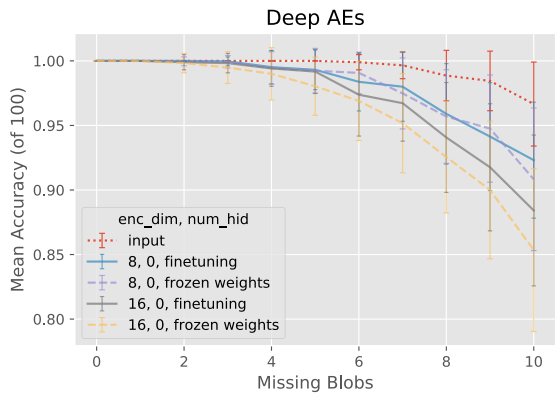
```
(0.1465359569451785, 0.1314671, 0.1314671)
```

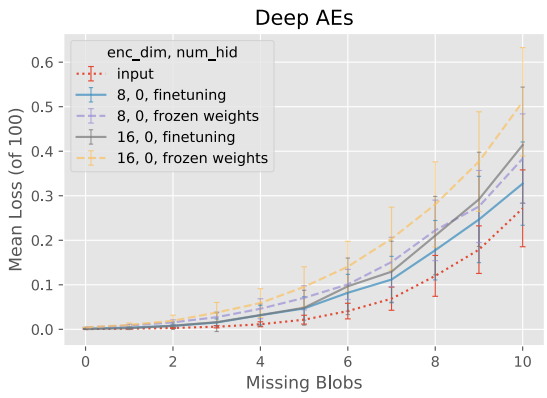
D) The difference between original statistics and decoded by a simple autoencoder. *x_test_decoded* — 8 dimensional AE, *x_decoded_2* — 128 dimensional AE

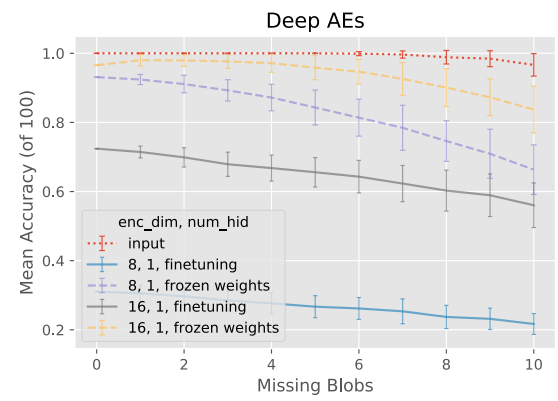
Fig. 2: Mean accuracy and loss for each autoencoder model as a function of the number of missing blobs

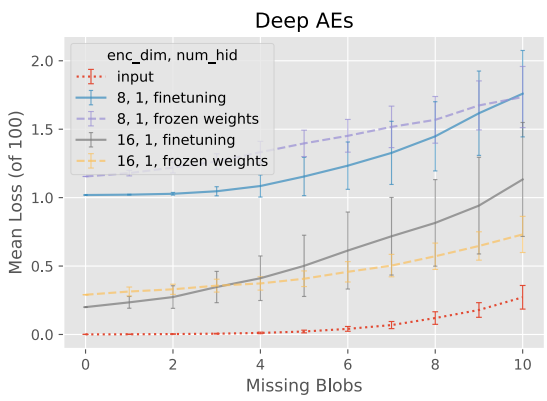


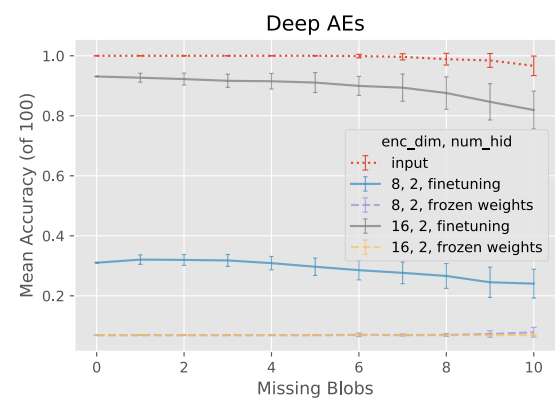


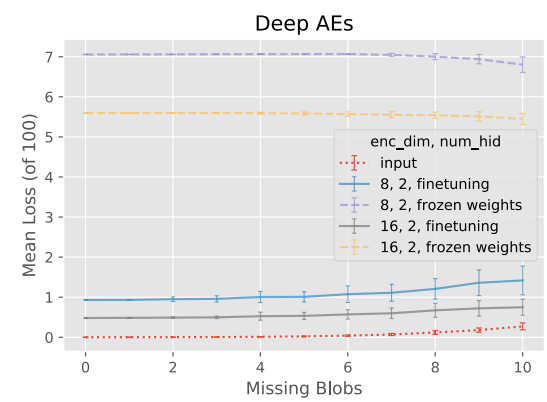


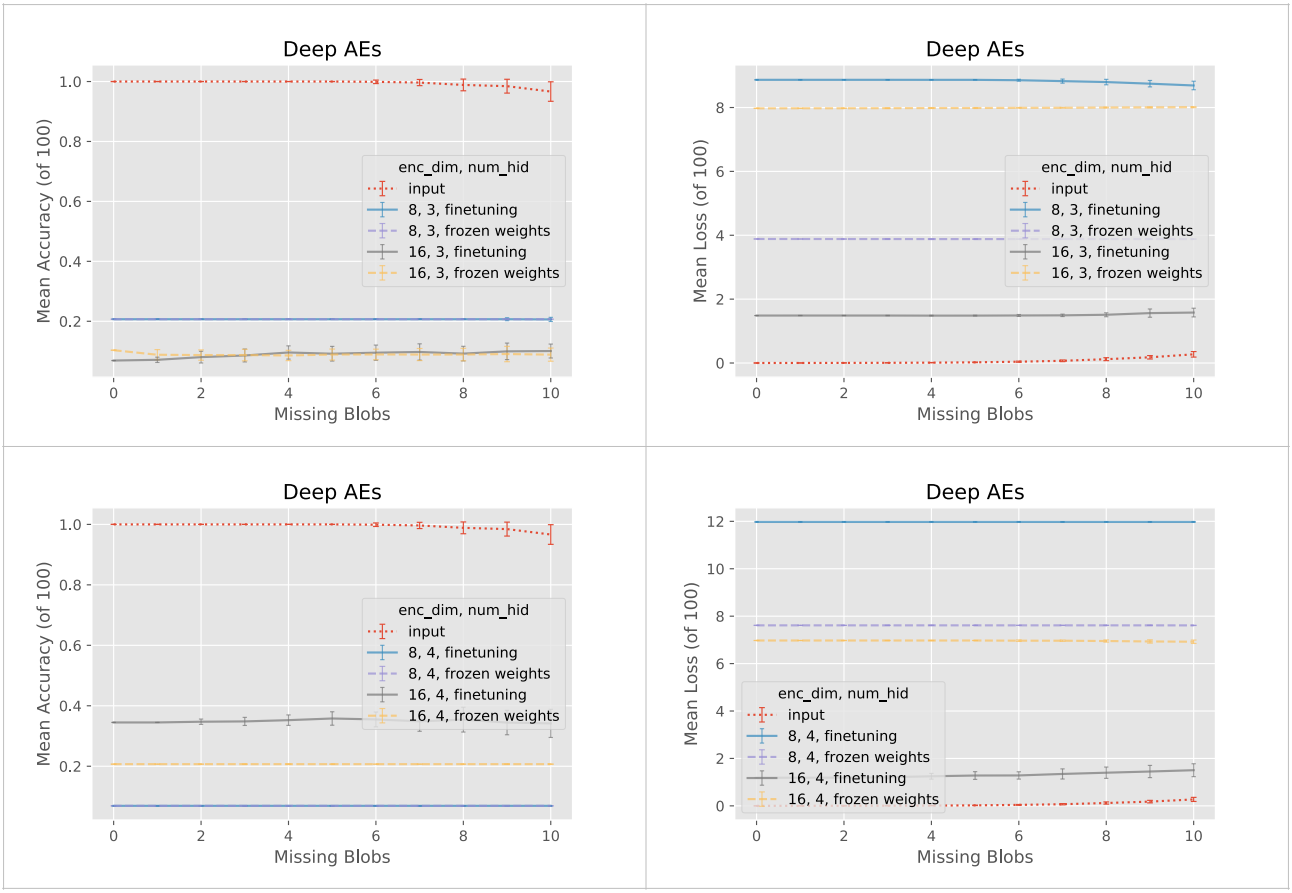











Deep AEs
Mean Accuracy (of 100)
Missing Blobs
enc_dim, num_hid
input
8, 3, finetuning
8, 3, frozen weights
16, 3, finetuning
16, 3, frozen weights
Deep AEs
Mean Loss (of 100)
Missing Blobs
enc_dim, num_hid
input
8, 3, finetuning
8, 3, frozen weights
16, 3, finetuning
16, 3, frozen weights
Deep AEs
Mean Accuracy (of 100)
Missing Blobs
enc_dim, num_hid
input
8, 4, finetuning
8, 4, frozen weights
16, 4, finetuning
16, 4, frozen weights
Deep AEs
Mean Loss (of 100)
Missing Blobs
enc_dim, num_hid
input
8, 4, finetuning
8, 4, frozen weights
16, 4, finetuning
16, 4, frozen weights

Discussion

The results that we have got for both the MNIST and the stimuli datasets turned out to be quite unexpected. We hypothesised that a simple logistic classifier would be comparable to a “naive network” in terms of the biological modelling, i.e. providing a baseline for the learning happening in the system, however, it might only be true for the datasets that are not highly linearly separable, as was our case. In this case it might be more telling to use an untrained autoencoder of the same architecture as the model of interest.

The second issue with this type of model comparison is whether to compare models trained for a fixed number of epochs and with fixed batch sizes, or to compare the best models, or to somehow weigh the training conditions with the number of trainable parameters. We suspect, for example, that the poor robustness of convolutional models has to do with the significantly larger number of trainable parameters comparing to the vector-based models.

Nevertheless, it was quite interesting to visualise for the stimuli dataset the learning outcomes of an autoencoder, such as representation dimension distributions, pattern sums and learning weights, as well as the structure of the input data. Also for the MNIST dataset, the visualisation of the R^2 provided a better metric for grading the model than just the loss value.

To sum up, error correction property might be another way of looking at model grading in machine learning, however, the autoencoder framework does not yet allow for proper analogy with the simulations of the cortical learning systems. Further work needs to be done to create a sufficient framework for this kind of endeavours, bearing in mind the other two general indicators of learning, invariance and pattern completion.

Acknowledgements

This project turned out to be quite demanding for us in terms of maintaining the structural organisation and presenting both the current state of affairs and the final outcome (which was not completely successful on our part), which surely taught us a couple good lessons. We thank our supervisor, Dr. Eilif Muller, for a daring project idea and good guidance, Dr. Giuseppe Chindemi for curating the part related to the INCITE project at BBP, and Francesco Casalegno for bringing structure into the project meetings and the feedback on our critical thinking in the machine learning domain.

Reference List

Ballard, Dana. 1987. “Modular Learning in Neural Networks.”

Bengio, Y., A. Courville, and P. Vincent. 2013. “Representation Learning: A Review and New Perspectives.” IEEE Transactions on Pattern Analysis and Machine Intelligence 35 (8): 1798–1828. https://doi.org/10.1109/TPAMI.2013.50.

Chollet, Francois. 2016. “Building Autoencoders in Keras.” https://blog.keras.io/building-autoencoders-in-keras.html

Goodfellow, I., and Bengio, Y., and Courville, A. 2016. “Deep Learning.” MIT Press. http://www.deeplearningbook.org

Wikipedia. “Coefficient of Determination.” https://en.wikipedia.org/wiki/Coefficient_of_determination