MisConv: Convolutional Neural Networks for Missing Data
Supplementary Material

Marcin Przewięźlikowski Marek Śmieja Łukasz Struski Jacek Tabor

Faculty of Mathematics and Computer Science, Jagiellonian University
6 Łojasiewicza Street, 30-348 Kraków, Poland

`marcin.przewiezlikowski@student.uj.edu.pl`
`{marek.smieja, lukasz.struski, jacek.tabor}@uj.edu.pl`

1. Experimental details

In this section, we describe in detail the architectures and hyperparameters of all models used in our experiments. We use the following notation:

- $conv_n$ - a convolution-ReLU-BatchNorm sequence with n convolution filters
- lin_n - a fully connected layer with n output features followed by ReLU
- $down$ / up - downsampling / upsampling convolution (with stride of 2), respectively:
- $drop_p$ - dropout layer with probability p

For simplicity, we omit the reshaping operations which take place between convolutions and fully connected layers.

1.1. Mixture of Factor Analyzers

In this section, we describe the implementation of Deep Mixture of Factor Analyzers (DMFA) and outline the architecture and hyperparameters used for each dataset. For additional information, we refer the reader to [3, 2].

MNIST Following [2], we use DMFA which uses a small convolutional feature extractor and single fully-connected layers to make final predictions of each MFA parameter (μ, A, D). The extractor is a sequence of four convolution layers with ReLU activations, with 16, 32, 64 and 32 filters, respectively. The number of predicted factor analyzers is $l = 4$. The DMFA is trained for 20 epochs, with a learning rate of $4 * 10^{-5}$ and a batch size of 48.

SVHN and CIFAR-10 Due to the increased dimentionality of those datasets, we use a fully convolutional variant of DMFA, which consists of a fully convolutional feature extractor composed of Convolution-ReLU-BatchNorm blocks [2], followed by a downsampling / upsampling convolution with a stride of 2. The network returns three heads that predict (μ, A, D). Thus the extractor consists of the following layers:

$$[conv_{32}] \times 2, down, [conv_{64}] \times 2, down,$$
$$[conv_{128}] \times 4, up, [conv_{64}] \times 2, up, [conv_{32}] \times 2$$

and parameter predictor heads consist of two convolutional layers with 16 filters with a ReLU nonlinearity and a BatchNorm layer between them.

The number of predicted factor analyzers is $l = 4$. We train the DMFA for 100 epochs, with a batch size of 64 and a learning rate of $4 * 10^{-5}$ for the first 10 epochs and $1 * 10^{-5}$ afterwards. Similarly to [2] we note that a fully-convolutional DMFA trained by minimizing only the NLL loss finds it difficult to find a good mean vector μ of the returned density. We mitigate this by supplying the NLL loss with MSE for the first 10 epochs of training.

CelebA For this dataset we use the same setup as in the case of SVHN / CIFAR-10 images, with two differences: a shorter training time - 50 epochs in total, as well as the size of the feature extractor, which is approximately two times bigger:

$$[conv_{32}] \times 4, down, [conv_{64}] \times 3, down,$$
$$[conv_{128}] \times 8, up, [conv_{64}] \times 4, up, [conv_{32}] \times 4$$

Training of DMFA on incomplete data In general, the DMFA model is trained by hiding a portion x_m of a sample x from the model and minimizing the Negative Log-Likelihood of imputation of x_m produced by the model.

Since we work with incomplete images, we cannot evaluate the NLL on x_m, because we do not have access to complete ground-truth data. Thus we simulate incompleteness of the data by hiding the other part x_u of each data sample x. Although the model produces imputations for both x_u and x_m, only the NLL of imputation of $x_u \setminus x_m$ is minimized, because this is an artificially created missing region.

1.2. Image classifier

For image classification we use a model composed of similar Convolution-ReLU-BatchNorm sequences and downsampling convolutions as in the fully convolutional DMFA, followed by two fully connected layers with a ReLU nonlinearity between them.

MNIST and SVHN We use an architecture of: $[conv_{32}, down, [conv_{64}] \times 2, lin_{20}, lin_{10}]$ and train the model with a batch size of 24 and a learning rate of 10^{-3} for 10 epochs in case of MNIST dataset and 25 epochs in case of SVHN dataset.

CIFAR-10 We use an architecture of: $[conv_{32}, drop_{0.3}] \times 2, down, [conv_{64}, drop_{0.3}] \times 3, lin_{128}, drop_{0.3}, lin_{10}]$ and train the model with a batch size of 64 and a learning rate of 10^{-4} for 35 epochs.

1.3. Wasserstein Autoencoder (WAE)

In accordance with [4], encoder and decoder sections of the WAE are fully convolutional, with a fully connected layer at the end of the encoder and the beginning of the encoder for transformation into and from the latent space, whereas the discriminator part of the network consists solely of fully connected layers with ReLU between them and a sigmoid layer at the end.

MNIST Architecture:

- encoder: $conv_{32}, conv_{40}, down, conv_{80}, down, lin_{20}$
- decoder: $lin_{3920}, up, conv_{80}, up, [conv_{40}] \times 2, conv_1$
- discriminator: $[lin_{64}] \times 4, lin_1$

We train the WAE for 20 epochs with a batch size of 128 and learning rate of $4 * 10^{-4}$ for encoder and decoder, and $4 * 10^{-6}$ for the discriminator.

SVHN Architecture:

- encoder: $conv_{32}, [conv_{96}] \times 2, down,$ $[conv_{192}] \times 2, down, lin_{20}$
- decoder: $lin_{12288}, up, [conv_{192}] \times 2, up,$ $[conv_{96}] \times 3, conv_3$
- discriminator: $[lin_{64}] \times 4, lin_1$

We train the WAE for 50 epochs with a batch size of 64 and a learning rate of $4 * 10^{-4}$ for encoder and decoder, and $4 * 10^{-6}$ for the discriminator.

CelebA Architecture:

- encoder: $conv_{32}, [conv_{64}] \times 4,$ $[conv_{128}] \times 4, [conv_{256}] \times 4, [conv512] \times 4, lin_{64}$
- decoder: $lin_{8192}, [conv_{512}] \times 4,$ $[conv_{256}] \times 4, [conv_{128}] \times 4, [conv_{64}] \times 5, conv_3$
- discriminator: $[lin_{512}] \times 4, lin_1$

We train the WAE for 50 epochs with a batch size of 16 and a learning rate of 10^{-4} for encoder and decoder, and $2.5 * 10^{-6}$ for the discriminator.

1.4. Implementation details

The models and experiments have been implemented using the PyTorch framework [1] and run on GeForce GTX 1080 and RTX 2080 NVIDIA GPUs. Every experiment described in this work is runnable on a single GPU.

The code implementing our technique is added to the supplemental material and will be made publicly available when the review period ends. The instructions on how to run the code are written in the README file included with the code.

In addition to our code, in this work we have used the following open-source implementations of baseline imputations: Partial Convolutions[1], SMFA[2], ACFlow[3] and KNN[4].

2. Other missing data tasks

Throughout the experiments, we used **square** missing regions which encompass 1/4 of their area. To benchmark the performance of MisConv on more diverse and difficult tasks, we also consider images where half of the image area was occluded with a trapezoid shape (**trapezoid**), as well as images with 3/4 of pixels randomly removed (**noise**) – see Figure 1. Although the missing regions have different shape, we consequently trained the DMFA by simulating additional missing parts with square shape.

As shown in Table 1, MisConv achieves the best accuracy on all variants of missing data. The above results show promise that MisConv can be leveraged to handle various kinds of missing data problems, even when the percentage of missing data is large.

---

[1] `https://github.com/NVIDIA/partialconv`, available under the BSD3 License

[2] `https://github.com/eitanrich/torch-mfa`

[3] `https://github.com/lupalab/ACFlow`

[4] `https://scikit-learn.org/stable/modules/generated/sklearn.impute.KNNImputer.html`, `https://github.com/rapidsai/cuml`, available under the Apache 2.0 License

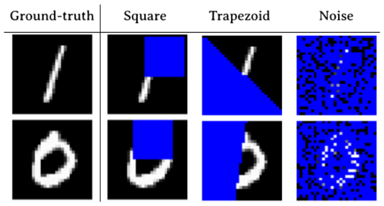

Figure 1: Examples of MNIST images with missing data simulated with square, trapezoid and noise techniques.

Table 1: Classification accuracy when testing on various kinds of incomplete MNIST datasets.

	square	trapezoid	noise
zero	0.91	0.891	0.946
mask	0.926	0.905	0.703
k-NN	0.874	0.854	0.912
PC	0.920	0.674	0.804
ACFlow	0.908	0.799	0.788
MisConv	**0.931**	**0.918**	**0.966**
GT	0.992	0.992	0.992

3. Computational overhead introduced by using MisConv

When computing the expected activation of the convolution based on the MFA representation of the missing data $(\mu, \mathbf{A}, \mathbf{d})$, $\mathbf{M}$ must also be applied to $\mathbf{x}$, μ, $\mathbf{d}$, as well as each of the l vectors $\mathbf{a_1}, ..., \mathbf{a_l}$ of which $\mathbf{A}$ is composed. In consequence, there the total of $3 + l$ vectors are processed by $\mathbf{M}$, which introduces a computational overhead compared with classical convolutions. Since in practice l is usually small, the effective increase in processing time is insignificant.

We demonstrate this by measuring the time it takes to perform a forward pass of missing data through a classical convolutional layer and through MisConv. As in all the above experiments, we select $l = 4$, which should lead to approximately quadrupled time of a forward pass. We check the time of a forward pass for images of sizes 28×28 (as in the MNIST dataset), 32×32 (as in the SVHN and CIFAR-10 datasets) and 64×64 (as in the CelebA dataset), with batch sizes of 16, 32 and 64. For each such setting, we make 5000 measurements of the time of a forward pass of a batch and report the results in Table 2 and Figure 2. It can be seen that in all cases, the overhead introduced by MisConv indeed increases the processing time approximately by a factor of 4. However, the order of magnitude of this increase is milliseconds. It should also be stressed that this overhead affects only the initial convolutional layer in the entire target neural network, which often consists of tens or even hundreds of layers. MisConv is therefore very efficient in practice, because it introduces a small computational overhead only in the initial layer of the entire neural network.

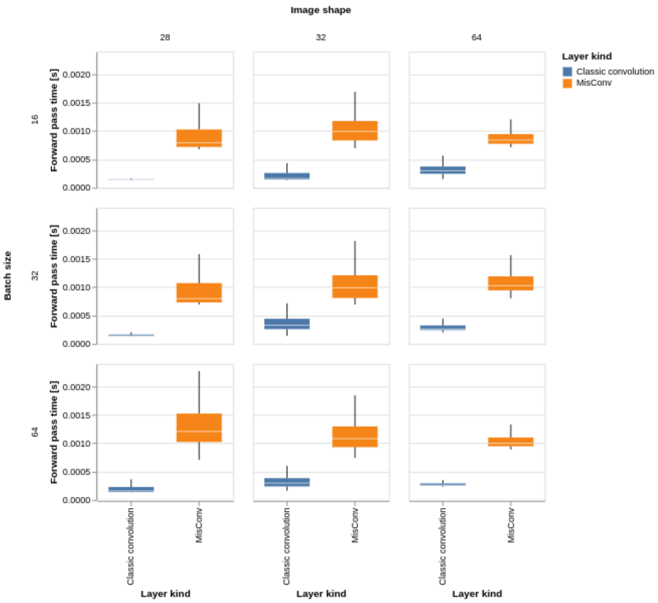

Figure 2: Time of a forward-pass through a classical convolutional layer and MisConv layer, measured for different image sizes in a form of boxplots.

				Layer type	
Datasets	**I**	**M**	**B**	Classic convolution	MisConv
MNIST	28	14	16	0.00016 ± 0.00006	0.00092 ± 0.00030
			32	0.00017 ± 0.00006	0.00097 ± 0.00041
			64	0.00020 ± 0.00008	0.00138 ± 0.00061
SVHN, CIFAR-10	32	16	16	0.00022 ± 0.00012	0.00109 ± 0.00039
			32	0.00037 ± 0.00020	0.00111 ± 0.00046
			64	0.00034 ± 0.00016	0.00121 ± 0.00047
CelebA	64	32	16	0.00034 ± 0.00017	0.00091 ± 0.00025
			32	0.00031 ± 0.00012	0.00115 ± 0.00037
			64	0.00029 ± 0.00008	0.00109 ± 0.00029

Table 2: Time of a forward-pass through a classical convolutional layer and MisConv layer, measured for different image sizes (denoted in the **I** column) with their respective sizes of missing data squares (denoted in the **M** column) and different batch sizes (denoted in the **B** column).

References

[1] Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, Alban Desmaison, Andreas Kopf, Edward Yang, Zachary DeVito, Martin Raison, Alykhan Tejani, Sasank Chilamkurthy, Benoit Steiner, Lu Fang, Junjie Bai, and Soumith Chintala. Pytorch: An imperative style, high-performance deep learning library. In H. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché-Buc, E. Fox, and R. Garnett, editors, *Advances in Neural Information*

Processing Systems 32, pages 8024–8035. Curran Associates, Inc., 2019.

[2] Marcin Przewieźlikowski, Marek Śmieja, and Łukasz Struski. Estimating conditional density of missing values using deep gaussian mixture model. In Haiqin Yang, Kitsuchart Pasupa, Andrew Chi-Sing Leung, James T. Kwok, Jonathan H. Chan, and Irwin King, editors, *Neural Information Processing*, pages 220–231, Cham, 2020. Springer International Publishing.

[3] Marcin Przewieźlikowski, Marek Śmieja, and Łukasz Struski. Estimating conditional density of missing values using deep gaussian mixture model. In *ICML Workshop on The Art of Learning with Missing Values (Artemiss)*, page 7, 2020.

[4] I. Tolstikhin, O. Bousquet, S. Gelly, and B. Schölkopf. Wasserstein auto-encoders. arXiv:1711.01558, 2017.