Processing of incomplete images by (graph) convolutional neural networks[⋆]

Tomasz Danel[1 [0000−0001−6053−0028]], Marek Śmieja[✉][1 [0000−0003−2027−4132]], Łukasz Struski[1 [0000−0003−4006−356X]], Przemysław Spurek[1 [0000−0003−0097−5521]], and Łukasz Maziarka[1 [0000−0001−6947−8131]]

Faculty of Mathematics and Computer Science
Jagiellonian University
Łojasiewicza 6, 30-428 Krakow, Poland
{tomasz.danel, marek.smieja}@ii.uj.edu.pl

Abstract. We investigate the problem of training neural networks from incomplete images without replacing missing values. For this purpose, we first represent an image as a graph, in which missing pixels are entirely ignored. The graph image representation is processed using a spatial graph convolutional network (SGCN) – a type of graph convolutional networks, which is a proper generalization of classical CNNs operating on images. On one hand, our approach avoids the problem of missing data imputation while, on the other hand, there is a natural correspondence between CNNs and SGCN. Experiments confirm that our approach performs better than analogical CNNs with the imputation of missing values on typical classification and reconstruction tasks.

Keywords: Graph convolutional networks · Convolutional neural networks · Missing data.

1 Introduction

Learning from missing data is one of the basic challenges in machine learning and data analysis [7]. In a typical pipeline, missing data are first replaced by some values (imputation) and next the complete data are used for training a given machine learning model [13]. The above approach depends strictly on the imputation procedure – if we accurately predict missing values, then the other model that operates on completed inputs can obtain good performance. However, it is not obvious how to select imputation method for a given problem, because it is difficult to validate its performance in a real-life scenario. Thus, there appears a natural question: *can we learn from missing data directly without using any imputation at the preprocessing stage?*

While it is difficult to answer this problem in general, a few approaches have already been designed for particular machine learning models [4, 5]. In [1] a

[⋆] A preliminary version of this paper appeared as an extended abstract [2] at the ICML Workshop on The Art of Learning with Missing Values.

modified SVM classifier is trained by scaling the margin according to observed features only. In [8], the embedding mapping of feature-value pairs is constructed together with a classification objective function. Pelckmans et al. [15] model the expected risk, which takes into account the uncertainty of the predicted outputs when missing values are involved. In a similar spirit, a random forest classifier is modified to adjust the voting weights of each tree by estimating the influence of missing data on the decision of the tree [19]. The authors of [9] design an algorithm for kernel classification that performs comparably to the classifier which has access to complete data. Goldberg et al. [6] treat class labels as an additional column in the data matrix and fill missing entries by matrix completion. The work [17] shows how to generalize fully connected neural networks to the case of missing data given only an imprecise Gaussian estimate of missing data. In the similar spirit, RBF kernel can be calculated for missing data [16]. Liu et al. [12] introduce partial convolutions, where the convolution is masked and renormalized to be conditioned on only observed pixels.

In this paper we interpret the image as a graph, in which each node coincides with a visible pixel, while edges connect neighboring pixels, see Figure 1. Since missing values are not mapped to graph nodes, we avoid the problem of missing data imputation. In order to efficiently process such an image representation, we use spatial graph convolutional neural networks (SGCN) [3]. In contrast to typical graph convolutions [10, 18], which consider graph as a relational structure invariant to rotations and translations, SGCN introduces a theoretically-justified mechanism to take into account spatial coordinates of nodes. More precisely, it has been proven that any layer of convolutional neural networks (CNNs) can be represented as a spatial graph convolution. This fact allows us to think about SGCN as a generalization of CNNs, which is able, in particular, to process incomplete images without imputation.

To verify the introduced procedure, we consider MNIST [11] and SVHN [14] image datasets. Experimental results show that SGCN performs better than typical CNNs with imputations on the tasks of image classification and reconstruction.

2 Graph-based model for processing incomplete images

In this section, we introduce our model for processing incomplete images. First, we describe how to interpret images as graphs. Next, we recall basic idea of graph convolutional networks (GCNs). Finally, we show the construction of SGCN and discuss it from an intuitive point of view.

2.1 General idea

Images can be interpreted as vectors (tensors) of fixed sizes. If the values of selected pixels are unknown, then the vector structure is destroyed. To recover this structure, we need to replace missing attributes with some values. Substituting unknown inputs carries the risk of introducing unreliable information and noise to initial data. This may have negative consequences on data interpretation as well as can decrease the performance of subsequent machine learning algorithms applied to completed inputs.

Our idea is to interpret an incomplete image as a graph. Graphs represent a relational structure, in which the number of nodes and edges is not fixed. If some pixels in the image are unknown, then the corresponding graph contains less nodes, but the way it is processed does not change. In consequence, graph-based representation of incomplete images is more natural than using imputation.

It is well-known that CNNs are state-of-the-art feature extractors for images. However, as explained above, it is not obvious how to apply CNNs to incomplete data without replacing missing values. In this paper, we use SGCN, which is a type of graph convolutional networks, that takes spatial coordinates of nodes into account. It has been proven that SGCN can mimic any image convolution and, in consequence, SGCN is able to work comparably to CNNs using analogical network architecture (number of layers, size of filters, etc.).

2.2 Graph-based representation of incomplete images

Formally, the image is represented as a tensor $\boldsymbol{H} = (\mathbf{h}_{ijk}) \in \mathbb{R}^{n \times m \times l}$, where n, m denote height and width of the image, and l is the number of channels. In the case of missing data, we do not have information about pixels values at some coordinates. Thus the incomplete image is denoted by a pair $(\boldsymbol{H}, J)$, where $J \subset \{1, \ldots, n\} \times \{1, \ldots, m\}$ indicates pixels which are unknown. In other words, $\mathbf{h}_{ijk}$ is unknown for every $(i, j) \in J$. Clearly, for a fully-observed image, $J = \emptyset$.

To construct a graph-based image representation, we create a node for every visible pixel of $\boldsymbol{H}$, i.e.

$$V = \{v_{ij} : (i, j) \in J'\},$$

where J' is the set of indices of the observed components. The edge is defined only for nodes that represent adjacent pixels. Formally,

$$E = \{(v_{ij}, v_{pq}) : (i, j) - (p, q) \in \{-1, 0, 1\}^2\}.$$

Observe that for a complete image, every "non-boundary" pixel (node) has exactly 8 neighbors. In the case of incomplete data, the number of neighbors can be smaller, as the unknown pixels are not converted to nodes and, in consequence, the corresponding edge is not created, see Figure 1. The information about pixels brightness is supplied with a feature vector $\mathbf{h}_{ij} \in \mathbb{R}^l$ that corresponds to a node:

$$\boldsymbol{H} = \{\mathbf{h}_{ij} : (i, j) \in J'\}.$$

For a gray-scale image, $\mathbf{h}_{ij} \in \mathbb{R}$, while for a color picture $\mathbf{h}_{ij} \in \mathbb{R}^3$.

2.3 Graph convolutions

Let $G = (V, E, \boldsymbol{H})$ be a graph (representing the image $\boldsymbol{H}$) with n nodes. To avoid multiple indexes in the following description, the node and the corresponding

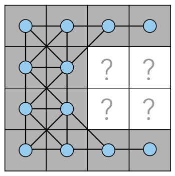

Fig. 1: Graph construction for an incomplete image of the size 4×4 with a missing region of the size 2×2.

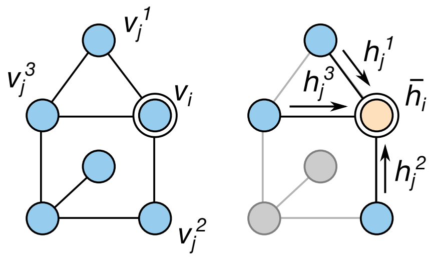

Fig. 2: Basic idea of GCNs. Every filter is responsible for defining a pattern used to aggregate feature vectors from adjacent nodes.

feature vector are denoted by v_i and $\mathbf{h}_i$, respectively, while e_{ij} is the edge between v_i and v_j. To make a natural correspondence between graphs and images, we put $\mathbf{i} = \begin{pmatrix} i_x \\ i_y \end{pmatrix}$ to denote both pixel coordinates and index in graph.

Basic idea of GCNs is to aggregate the information of feature vectors from neighboring nodes over multiple layers, see Figure 2. To build a diverse set of patterns, GCNs use filters for defining a specific aggregation. Information from higher-level neighborhoods are fused by combining many layers together.

The above goal is realized by combining two operations. For each node v_i, feature vectors of its neighbors are first aggregated:

$$\bar{\mathbf{h}}_i = \sum_{(v_i, v_j) \in E} u_{ij} \mathbf{h}_j. \tag{1}$$

Observe that the aggregation is performed only over neighbor nodes, i.e $(v_i, v_j) \in E$. The weights u_{ij} are either trainable [18] or determined from a graph [10]. Next, a standard MLP is applied to transform the intermediate representation $\bar{\boldsymbol{H}} = [\bar{\mathbf{h}}_1 \dots, \bar{\mathbf{h}}_n]$ into the final output of a given layer:

$$\mathrm{MLP}(\bar{\boldsymbol{H}}; \boldsymbol{W}) = \mathrm{ReLU}(\boldsymbol{W}^T \bar{\boldsymbol{H}} + \mathbf{b}), \tag{2}$$

where $\boldsymbol{W} \in \mathbb{R}^{I \times O}$ is a trainable weight matrix, $\boldsymbol{b} \in \mathbb{R}^O$ is a trainable bias vector (added column-wise), and I and O is the size of input and output layer respectively.. A typical GCN is composed of a sequence of graph convolutional layers (described above). Finally, its output is aggregated to the network response

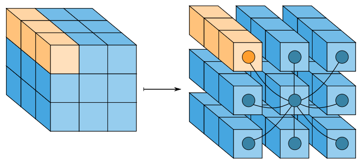

Fig. 3: Convolutional kernel for images (left) can be translated to spatial graph convolutions. Vectors at different positions (see e.g. the orange vector in the figure) are multiplied by different weights. In spatial graph convolutions (right) weights are modified by relative positions of graph neighbors to achieve different weights for each vector.

using a global pooling or a dense layer depending on a given task, e.g. node or graph classification.

2.4 Spatial graph convolutions

In contrast to typical GCNs described above, SGCN uses spatial coordinates of nodes, see Figure 3. In the case of images, spatial coordinates allows to identify a given pixel in the image grid, which is not possible using only the information about neighborhood. What is more important, the convolution defined by SGCN is constructed so that it is able to reflect any convolutional filter of typical CNNs. In other words, any image convolution can be obtained by a specific parametrization of SGCN. This makes a natural correspondence between SGCN and CNNs. This property cannot be obtained by simply adding spatial coordinates to feature vectors in classical GCNs.

From a formal side, SGCN replaces (1) by:

$$\bar{\mathbf{h}}_i(\boldsymbol{U}, \mathbf{b}) = \sum_{(v_i,v_j)\in E} \mathrm{ReLU}\left(\boldsymbol{U}\left[\begin{pmatrix} j_x \\ j_y \end{pmatrix} - \begin{pmatrix} i_x \\ i_y \end{pmatrix}\right] + \mathbf{b}\right) \odot \mathbf{h}_j, \tag{3}$$

where $\boldsymbol{U} \in \mathbb{R}^{I\times 2}$, $\mathbf{b} \in \mathbb{R}^I$ are trainable, and I is the dimension of the previous layer vectors. The operator $\odot$ is element-wise multiplication. The relative positions in the neighborhood are transformed using a linear operation combined with non-linear ReLU function. This is used to weight the feature vectors $\mathbf{h}_j$ in a neighborhood. By the analogy with classical convolution, this transformation can be extended to multiple filters. Let $\boldsymbol{U} = [\boldsymbol{U}_1, \ldots, \boldsymbol{U}_k]$ and $\boldsymbol{B} = [\mathbf{b}_1, \ldots, \mathbf{b}_k]$ define k-filters. The intermediate representation $\bar{\mathbf{h}}_i$ is a vector defined by:

$$\bar{\boldsymbol{H}}_i = \left[\bar{\mathbf{h}}_i(\boldsymbol{U}_1, \mathbf{b}_1), \ldots, \bar{\mathbf{h}}_i(\boldsymbol{U}_k, \mathbf{b}_k)\right].$$

Finally, MLP transformation is applied in the same manner as in (2) to transform these feature vectors.

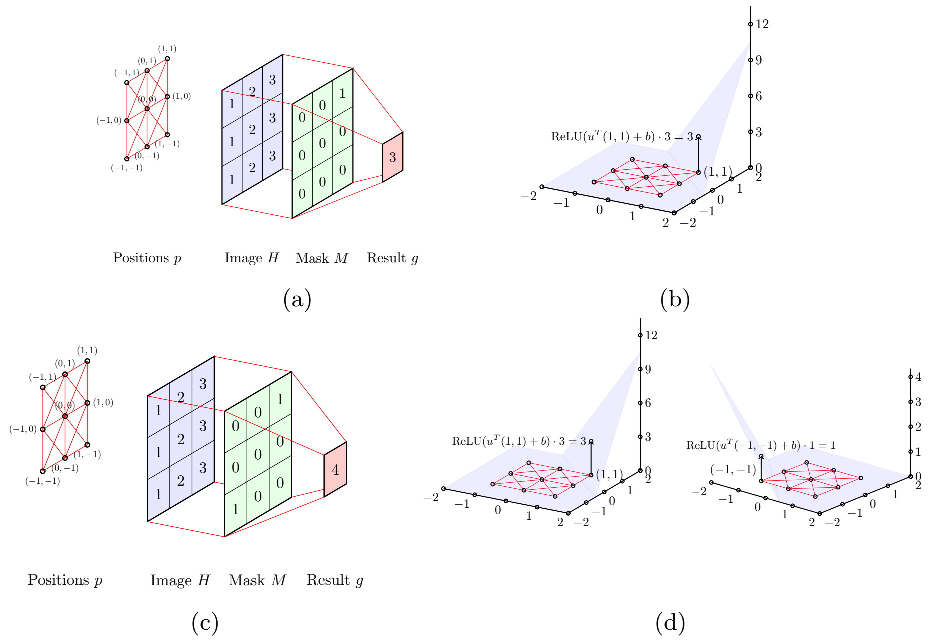
Positions p
Image H
Mask M
Result g
(a)
$\mathrm{ReLU}(u^T(1,1)+b)\cdot 3 = 3$
(b)
(c)
$\mathrm{ReLU}(u^T(-1,-1)+b)\cdot 1 = 1$
(d)

Fig. 4: Replicating two convolution operations by SGCN (top and bottom), see Examples 1 and 2 for details. On the left, the result of applying a convolutional filter $\boldsymbol{M}$ to the image $\boldsymbol{H}$. The positions grid p represents spatial coordinates of the pixels; the neighbors are connected with edges. Analogous convolution can be applied to a spatial graph representation, as shown on the right. In the first case (top) ReLU applied to the linear transformation of the spatial features of the image graph (with $\mathbf{u} = (2,2)^T$ and $b = -3$) allows to select (and possibly modify) the top-right neighbor. In the second case (bottom), convolution operation can be obtained by extracting two opposite corner values (with $\mathbf{u}_1 = (2,2)^T, b_1 = -3$ and $\mathbf{u}_2 = (-2,-2)^T, b_2 = -3$) and summing them.

2.5 Intuition behind Spatial Graph Convolutions

As mentioned, the formulas of SGCN allow to imitate the filters of classical CNNs. While the formal proof of this fact can be found in [3], in this section, we demonstrate this property on toy examples, which help to better understand the reasoning behind SGCN. Let us recall that the classical convolution operation (without pooling) defined by the mask $\boldsymbol{M} = (m_{i'j'})_{i',j' \in \{-k..k\}}$ applied to the image $\boldsymbol{H} = (h_{ij})_{i \in \{1..n\}, j \in \{1..m\}}$ is given by

$$\boldsymbol{M} * \boldsymbol{H} = \boldsymbol{G} = (g_{ij})_{i \in \{1..n\}, j \in \{1..m\}},$$

where

$$g_{ij} = \sum_{\substack{i'=-k..k:\, i+i' \in \{1..n\},\\ j'=-k..k:\, j+j' \in \{1..m\}}} m_{i'j'} h_{i+i',j+j'}.$$

For simplicity, we consider gray-scale images with a single channel.

The following examples illustrate how to construct filters of SGCN that produce identical results to classical convolution operations.

Example 1. First, let us consider a linear convolution given by the mask

$$\boldsymbol{M} = \begin{bmatrix} 0 & 0 & 1 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \end{bmatrix}.$$

Observe that as the result of this convolution on the image $\boldsymbol{H}$, every pixel is exchanged by its right upper neighbor, see Figure 4 (top). Now the image is represented as a graph, where the neighborhood N_i of the pixel with coordinates $\mathbf{i} = \begin{pmatrix} i_x \\ i_y \end{pmatrix}$ is given by the pixels with coordinates $\mathbf{j} = \begin{pmatrix} j_x \\ j_y \end{pmatrix}$ such that $\mathbf{j} - \mathbf{i} \in \{-1, 0, 1\}^2$.

Given a vector[1] $\mathbf{u} \in \mathbb{R}^2$ and a bias $b \in \mathbb{R}$ the (intermediate) graph operation is defined by

$$g_i = \bar{h}_i(\mathbf{u}, b) = \sum_{j \in N_i} \mathrm{ReLU}\left(\mathbf{u}^T \left[\begin{pmatrix} j_x \\ j_y \end{pmatrix} - \begin{pmatrix} i_x \\ i_y \end{pmatrix} \right] + b \right) \cdot h_j.$$

Consider now the case when $\mathbf{u} = 2 \cdot \mathbb{1}, b = -3,$ where $\mathbb{1} = (1,1)^T$. One can easily observe that

$$\mathrm{ReLU}(\mathbf{u}^T \mathbf{z} - b) = \begin{cases} 0, & \text{for } \mathbf{z} \neq \mathbb{1} \\ 1, & \text{for } \mathbf{z} = \mathbb{1}, \end{cases}$$

where $\mathbf{z} = \mathbf{j} - \mathbf{i} \in \{-1, 0, 1\}^2$.

Consequently, we obtain that $g_i = h_{i+\mathbb{1}}$, which equals the result of the considered linear convolution.

Example 2. Now, let us consider the mask, see Figure 4 (bottom):

$$\boldsymbol{M} = \begin{bmatrix} 0 & 0 & 1 \\ 0 & 0 & 0 \\ 1 & 0 & 0 \end{bmatrix}.$$

This convolution cannot be obtained from graph representation using a single transformation as in previous example.

To formulate this convolution, we define two intermediate operations for $k = 1, 2$:

$$\bar{h}_i(\mathbf{u}_k^T, b_k) = \sum_{j \in N_i} \mathrm{ReLU}\left(\mathbf{u}_k \left[\begin{pmatrix} j_x \\ j_y \end{pmatrix} - \begin{pmatrix} i_x \\ i_y \end{pmatrix} \right] + b_k \right) \cdot h_j.$$

[1] For simplicity, we consider an image with a single channel.

where $\mathbf{u}_1 = 2 \cdot \mathbb{1}, b_1 = -3$ and $\mathbf{u}_2 = -2 \cdot \mathbb{1}, b_2 = -3$. The first operation extracts the right upper corner, while the second one extracts the left bottom corner, i.e.

$$\bar{h}_i(\mathbf{u}_1, b_1) = 1 \cdot h_{i+\mathbb{1}} \text{ and } \bar{h}_i(\mathbf{u}_2, b_2) = 1 \cdot h_{i-\mathbb{1}}.$$

Finally, we put

$$\bar{\boldsymbol{H}}_i = \begin{bmatrix} \bar{h}_i(\mathbf{u}_1, b_1) \\ \bar{h}_i(\mathbf{u}_2, b_2) \end{bmatrix} = \begin{bmatrix} h_{i+\mathbb{1}} \\ h_{i-\mathbb{1}} \end{bmatrix}.$$

Making an additional linear transformation defined by (2) with $\mathbf{w}_i = (1, 1)^T$, we obtain:

$$g_i = \mathbf{w}_i^T \bar{\boldsymbol{H}}_i = 1 \cdot h_{i+\mathbb{1}} + 1 \cdot h_{i-\mathbb{1}}.$$

Following the above examples, to obtain the result of applying an arbitrary 3×3 filter, we may need at most 9 operations using (3) (one for replicating each of 9 entries). SGCN is also capable of imitating larger filters, see [3] for details.

3 Experiments

In this section, we evaluate our model on two machine learning tasks and compare it with related approaches.

3.1 Reconstruction

First, as a proof of concept, we take into account MNIST database and consider the problem of restoring corrupted images, in which a part of data is hidden. To prepare this task, for each image of the size 28×28, we remove a square patch of the size 13×13. The location of the patch is uniformly sampled for each image.

The reconstruction models are instantiated using the auto-encoder architecture (AE). In the case of our model, the encoder is implemented as SGCN with 5 spatial graph convolutional layers while the decoder is a typical deconvolutional neural network, which returns the image in the form of tensor. We emphasize that graph neural network is only needed for initial stage of processing (encoder) to avoid replacing missing values. Subsequent stages, e.g. decoder, can be implemented using typical non-graph networks. The result of our model is compared to CNNs combined with typical imputation techniques: (i) **mask**, which is a zero imputation with an additional binary channel indicating unknown pixels (ii) **mean** imputation, where absent attributes are replaced by mean values for a given coordinate (iii) **k-nn** imputation, which substitutes missing features with mean values of those features computed from the k-nearest training samples (we use $k = 5$). For a fair comparison, every architecture (SGCN and CNNs) has the same structure, i.e. the number of layers and filters as well as the type of regularization.

We assume that the complete data are not available in the training phase. Therefore, for all models, the loss is defined as the mean-square error (MSE) calculated outside the missing region. This makes the problem more difficult than

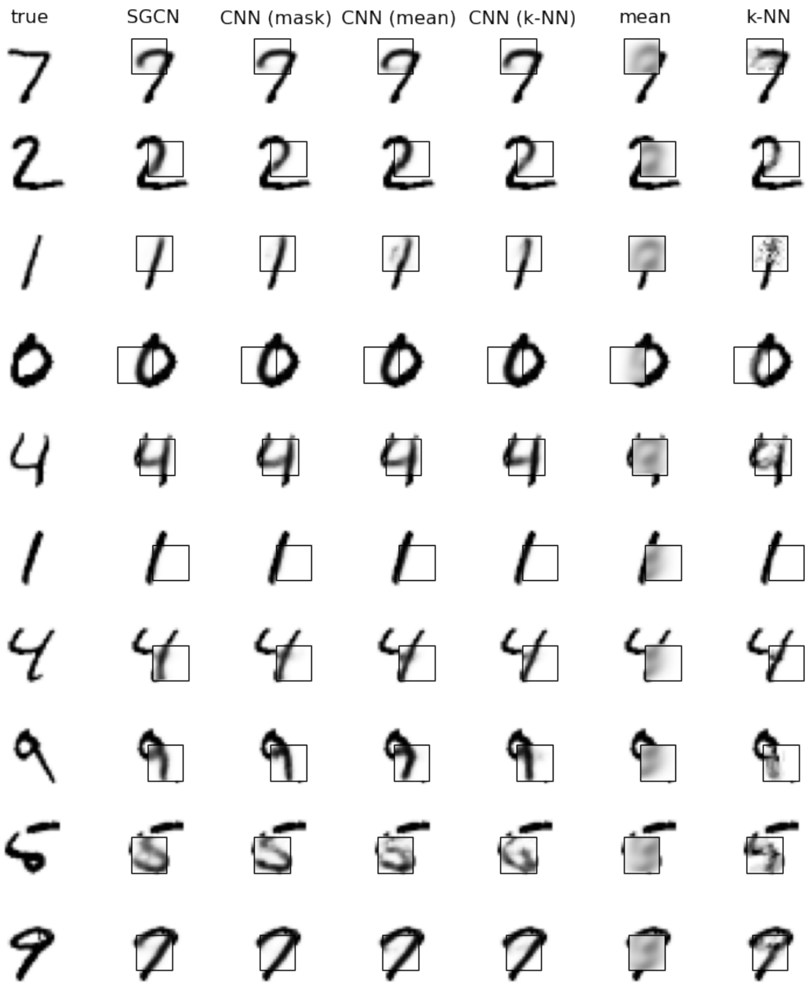

Fig. 5: Reconstructions obtained for MNIST dataset (the first 10 images of test set). To demonstrate the influence of initial imputations on the final reconstructions returned by CNNs, we show the results of applying mean and k-NN imputations (last two columns).

a typical inpainting task. To isolate the effect of processing incomplete images and directly compare SGCN with CNNs, we do not introduce additional losses and use only MSE in training.

It can be seen from the Figure 5 that SGCN gives similar results to CNN (mask). The reconstructions coincide on average with ground-truth and are free of artifacts. There was a problem in restoring digit ”9” (last row), but the same holds for other methods. The results produced by CNN (mean) and CNN (k-NN) are sometimes blurry. To support the visual inspection with quantitative

Table 1: Performance on three machine learning tasks (lower is better). The first row shows mean-square error for reconstructing incomplete MNIST images, while the last two rows present test errors for classifying incomplete MNIST and SVHN images.

Dataset	SGCN	GCN	CNN (mask)	CNN (mean)	CNN (k-NN)
MNIST (MSE)	0.0755	-	0.0760	0.0787	**0.0725**
MNIST (Error)	**4.6%**	31.4%	4.9%	5.9%	5.7%
SVHN (Error)	**16.6%**	74.6%	18.6%	19.9%	22.4%

assessment, we calculate MSE inside the missing region, see the first row of Table 1. Surprisingly, CNN (k-NN) gives the highest resemblance with ground-truth in terms of MSE. While the reconstructions look visually less plausible than the ones returned by SGCN and CNN (mask), the pixel-wise agreement with ground-truth is higher. SGCN gives the second best result. It is important to observe the influence of initial imputation on the performance of CNNs. Since the k-NN imputation is significantly better than the mean imputation[2], the corresponding CNN model is able to restore input image more reliably. However, it is evident that mistakes made by k-NN imputation also negatively affect the performance of CNN (3rd and 9th rows). In contrast, our method is more stable, because it does not depend on imputation strategy. In consequence, it may give worse results than CNN when it is easy to predict missing values, but, at the same time, it should perform better if the imputation problem is more difficult. Another advantage is that SGCN is trained end-to-end (no preprocessing of missing values).

3.2 Classification

In the second experiment, we consider the classification task, in which incomplete data appear in both train and test phase. In addition to gray-scale handwritten digits retrieved from MNIST database, we also use color house-number images of the SVHN dataset. In the case of SVHN images of the size 32×32, we use patches of the size 15×15.

For a comparison, we use CNN models combined with the same imputation techniques as before, but, additionally, we consider "vanilla" **GCN** [10], which is one of the simplest GCNs that ignores spatial coordinates[3]. Classification network is composed of 8 convolutional layers. Each one contains 64 filters of the size 3×3. Batch normalization is used after every convolutional layer. As mentioned, we use analogical architecture for both graph convolutions and typical image convolutions.

[2] k-NN imputation obtains MSE=0.0807 while mean imputation gives MSE=0.1265.

[3] We also experimented with graph attention network [18], but the results did not improve.

It is evident from Table 1 (2nd and 3rd rows) that SGCN performs significantly better than the other version of GCN. It is not surprising because, in contrast to typical GCNs, SGCN introduces information about spatial coordinates to the model. Next observation is that SGCN gives lower errors than CNNs combined with imputation strategies. While the advantage of SGCN over the second best method in the case of MNIST is slight, the difference in accuracy is higher in the case of SVHN, which is significantly harder dataset to classify. As mentioned earlier, it may be difficult to reliably predict missing values for hard tasks, which, in consequence, negatively affects CNN models applied to completed data. Moreover, it can be seen that k-NN imputation is not so beneficial in classification problems as in reconstruction task – CNN (k-NN) performs even worse than CNN (mean) on SVHN. As can be seen, the knowledge about missing pixels is more important for the success of classification CNNs than using specific imputation technique[4]. In contrast to CNN (mask), which uses an additional binary channel to pass the information about unknown values to the neural model, SGCN directly ignores missing pixels, which is more natural.

4 Conclusion

We presented an alternative way of learning neural networks from incomplete images, which does not require replacing missing values at the preprocessing stage. While graph representation of incomplete images avoids the problem of imputation, applying SGCN allows us to reflect the action of classical CNNs. Our model is trained end-to-end without any missing data preprocessing (imputation) as in the case of CNNs. Since the proposed model completely ignores the information about missing values, it is especially useful in the case of complex tasks, where imputation strategies introduce noise and unreliable information. The main disadvantage of our approach is the computational cost of using GCNs. In contrast to classical CNNs, the current implementations of GCNs are less efficient and it is difficult to manage large graphs created from high dimensional images.

Acknowledgements

The work of M. Śmieja was supported by the National Science Centre (Poland) grant no. 2017/25/B/ST6/01271. The work of Ł. Struski was supported by the Foundation for Polish Science Grant No. POIR.04.04.00-00-14DE/18-00 co-financed by the European Union under the European Regional Development Fund. The work of P. Spurek was supported by the National Centre of Science (Poland) Grant No. 2019/33/B/ST6/00894. The work of Ł. Maziarka was supported by the National Science Centre (Poland) grant no. 2018/31/B/ST6/00993.

[4] We verified that combining masking with mean/k-nn imputation does not lead to further improvement of CNNs.

References

1. Chechik, G., Heitz, G., Elidan, G., Abbeel, P., Koller, D.: Max-margin classification of data with absent features. Journal of Machine Learning Research **9**, 1–21 (2008)
2. Danel, T., Śmieja, M., Struski, Ł., Spurek, P., Maziarka, L.: Processing of incomplete images by (graph) convolutional neural networks. In: ICML Workshop on The Art of Learning with Missing Values (Artemiss). p. 6 (2020)
3. Danel, T., Spurek, P., Tabor, J., Śmieja, M., Struski, Ł., Słowik, A., Maziarka, Ł.: Spatial graph convolutional networks. arXiv preprint arXiv:1909.05310 (2020)
4. Dekel, O., Shamir, O., Xiao, L.: Learning to classify with missing and corrupted features. Machine Learning **81**(2), 149–178 (2010)
5. Globerson, A., Roweis, S.: Nightmare at test time: robust learning by feature deletion. In: Proceedings of the International Conference on Machine Learning. pp. 353–360. ACM (2006)
6. Goldberg, A., Recht, B., Xu, J., Nowak, R., Zhu, X.: Transduction with matrix completion: Three birds with one stone. In: Advances in neural information processing systems. pp. 757–765 (2010)
7. Goodfellow, I., Bengio, Y., Courville, A.: Deep learning. MIT press (2016)
8. Grangier, D., Melvin, I.: Feature set embedding for incomplete data. In: Advances in Neural Information Processing Systems. pp. 793–801 (2010)
9. Hazan, E., Livni, R., Mansour, Y.: Classification with low rank and missing data. In: Proceedings of The 32nd International Conference on Machine Learning. pp. 257–266 (2015)
10. Kipf, T.N., Welling, M.: Semi-supervised classification with graph convolutional networks. arXiv preprint arXiv:1609.02907 (2016)
11. LeCun, Y., Bottou, L., Bengio, Y., Haffner, P.: Gradient-based learning applied to document recognition. Proceedings of the IEEE **86**, 2278–2324 (1998)
12. Liu, G., Reda, F.A., Shih, K.J., Wang, T.C., Tao, A., Catanzaro, B.: Image inpainting for irregular holes using partial convolutions. In: Proceedings of the European Conference on Computer Vision (ECCV). pp. 85–100 (2018)
13. McKnight, P.E., McKnight, K.M., Sidani, S., Figueredo, A.J.: Missing data: A gentle introduction. Guilford Press (2007)
14. Netzer, Y., Wang, T., Coates, A., Bissacco, A., Wu, B., Ng, A.Y.: Reading digits in natural images with unsupervised feature learning. In: NIPS Workshop on Deep Learning and Unsupervised Feature Learning (2011)
15. Pelckmans, K., De Brabanter, J., Suykens, J.A., De Moor, B.: Handling missing values in support vector machine classifiers. Neural Networks **18**(5), 684–692 (2005)
16. Śmieja, M., Struski, Ł., Tabor, J., Marzec, M.: Generalized rbf kernel for incomplete data. Knowledge-Based Systems **173**, 150–162 (2019)
17. Śmieja, M., Struski, Ł., Tabor, J., Zieliński, B., Spurek, P.: Processing of missing data by neural networks. In: Advances in Neural Information Processing Systems. pp. 2719–2729 (2018)
18. Veličković, P., Cucurull, G., Casanova, A., Romero, A., Lio, P., Bengio, Y.: Graph attention networks. arXiv preprint arXiv:1710.10903 (2017)
19. Xia, J., Zhang, S., Cai, G., Li, L., Pan, Q., Yan, J., Ning, G.: Adjusted weight voting algorithm for random forests in handling missing values. Pattern Recognition **69**, 52–60 (2017)