NEURAL NETWORKS FOR SHAPE RECOGNITION BY MEDIAL REPRESENTATION

N. Lomov[1]*, S. Arseev[1]

[1] Lomonosov Moscow State University
GSP-1, Leninskie Gory, Moscow, 119991, Russian Federation
nikita-lomov@mail.ru | 9413serg@gmail.com

Commission II, WG II/5

KEY WORDS: Deep Learning, Convolutional Neural Networks, Medial Representation, Skeleton

ABSTRACT:

The article is dedicated to the development of neural networks that process data of a special kind — a medial representation of the shape, which is considered as a special case of an undirected graph. Methods for solving problems that complicate the processing of data of this type by traditional neural networks — different length of input data, heterogeneity of its structure, unordered constituent elements — are proposed. Skeletal counterparts of standard operations used in convolutional neural networks are formulated. Experiments on character recognition for various fonts, on classification of handwritten digits and data compression using the autoencoder-style architecture are carried out.

1. INTRODUCTION

The rapid increase in popularity of deep learning methods causing a quick development of this area started a few years ago with a series of successes in image recognition tasks. Images are usually represented as matrices of points, so they can be considered an example of organized, standardized data. Later, deep learning methods also achieved serious progress in tasks with more complex data which can be represented as an arbitrary length sequences: speech recognition, handwriting recognition, machine translation etc. This was caused by the recurrent neural networks development and usage of specific architectures well suited for these tasks.

Proposed architectures process only structured (by temporal or spatial position) data. However, the problem of processing data less suited for neural networks is still relatively unexplored. A typical example of this type of data are graphs, and in general data of this type often appears in computational geometry: point clouds, polygonal meshes, triangulations, skeletal and boundary representations. The predicament is that the objects to be processed are not ordered and can have an arbitrary size and link structure. A detailed research of this problem and possible approaches to its solution can be found in (Bronstein et al., 2017).

The point of this work is to apply neural network based approach to a special kind of graphs, the skeletons which represent a shape of a binary figure in the form of a certain "carcass"-type basis. Though the skeleton is a special case of an undirected graph, it has some properties (in particular, its geometric nature and sparseness) that can be used with the additional benefit.

Skeletal representations are used in neural network based methods in a very specific context, usually in the tasks of action and gesture recognition (Du et al., 2015) (Li et al., 2017) where skeletons are acquired via sensor devices (e.g. Kinect). These skeletons are intended exclusively for human pose representation and are strictly structured with a fixed number of joints representing specific body parts (head, limbs etc). The authors do not know of any examples where skeletons representing an arbitrary form would be used as an input data. This paper is an attempt to begin research in this area.

*Corresponding author

2. SKELETON PROCESSING WITH NEURAL NETWORK

2.1 Basic Concepts

Skeleton of a binary shape is a set of centers of all maximally inscribed circles of this shape.

Radial function is a function that matches each point of the skeleton to its inscribed circle radius.

Skeleton and radial function together constitute the *medial shape representation*.

Pruning or a regularization of the skeleton is a process of removal of branches that don't significantly contribute to the shape.

As a model of the shape of objects in the image, we will use polygonal figures. In this case a skeleton can be conidered as a geometric graph where edges are segments of straight lines and parabolas. That representation is acquired when the skeleton is built from the Voronoi diagram of linear edge segments (Mestetskiy, 2010). For this task we will represent a skeleton as $S = (V, E)$ where each node $v \in V$ is a triplet (x, y, r) of coordinates and radial function value and each edge $e \in E$ is an unordered pair of nodes.

We will use the extraction of polygonal contours, skeletonization and pruning algorithms for a binary image described in (Mestetskiy, 2009).

2.2 Node Ordering

Our initial goal is being able to process skeletons as an input for a neural network. We see two obstacles on the way there: 1)

skeletons have variable dimension due to having variable edges and notes number and 2) skeleton elements (nodes and edges) have no “natural” order. (Niepert et al., 2016) proposes a solution for this problem: skeleton nodes need to be ordered according to their betweenness centrality value. For tne node v this value is defined in the following way:

$$g(v) = \sum_{s \neq v \neq t} \frac{\sigma_{st}(v)}{\sigma_{st}},$$

where σ_{st} is a number of shortest paths from the node s to the node$t$, and $\sigma_{st}(v)$ is a number of such paths passing through v. Therefore, nodes that have more neighbours than the others and are far enough from terminal nodes will end up in the beginning of the list while isolated and terminal nodes will be in the end of the list. As an additional metric we will use the closeness centrality, $h(v)$, and if both metrics end up equal we will compare the distances (d_{ik}) from candidate nodes to nodes already in the list.

Algorithm 1 Skeleton node ordering

Given: set of nodes $\{v_i\}_{i=1}^n$
for $i \in 1, \ldots, n$ **do**
  $F_i = [g(v_i), h(v_i)]$
$I = [1, \ldots, n]$
$L = [\,]$
while $I \neq [\,]$ **do**
  Sort the list I by F_i in lexicographical order
  $k = \text{pop}(I)$
  Add k to the end of L
  for $i \in I$ **do**
    Add d_{ik} to the end of F_i

Note that the ordering process may be unnecessary if the last layer will apply a symmetric pooling operation to the entire feature map, e.g. max pooling or average pooling.

2.3 Skeleton Unification

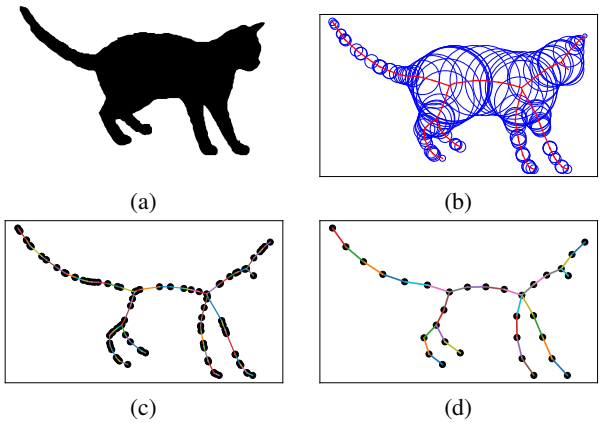

Figure 1. Fixed length skeleton construction.
a) Image b) medial representation c) skeleton d) unified skeleton.

Let’s refer to (Niepert et al., 2016) again: to leave the specific number of nodes n in the graph we can take first n nodes from the list produced on the previous step while keeping their order if there are no less than n nodes, and if there are less we can complete the list with dummy isolated nodes which will have all feature values equal to zeros. Although that’s a decent solution for arbitrary graphs, in the case of a skeleton it will distort its geometrical shape: if there are too many nodes the endings of the branches will be cut. The idea is to use a unified skeleton that will keep she shape of the original skeleton while having all edges have approximately similar length.

Algorithm 2 Skeleton unification

Given: skeleton S, node number n, minimum relative length ε.
If there is more than one connected component in the skeleton, make the skeleton connected using additional edges of minimum length.
Split the skeleton into branches, i.e. lines ending in nodes with degrees of 1 and 3 and not having any such nodes in between. Mark the number of branches as n_e and the number if their ends as n_v.
if $n_v > n$ **then**
  For $n_v - n$ shortest branches merge the ends into one point in the middle of the branch
else
  Determine the length of each branch l_i, total skeleton length L and average length $l = \frac{L}{n_e + n - n_v}$
  For branches shorter than εl merge ends
  Recalculate n_e and n_v
  for $i \in 1, \ldots, n_e$ **do**
    $n_i = \lceil \frac{l_i}{l} \rceil - 1$
  Calculate the values $n_i = \lceil \frac{l_i}{l} \rceil - 1$ that determine the number of vertices that will be added to each branch:
  while $n_v + \sum_{i=1}^{n_e} n_i \geq n$ **do**
    Choose the node with minimum $\frac{l_i}{n_i - 1}$ and reduce n_i by 1
  Add n_i nodes on each branch uniformly

The data preparation process is illustrated in fig. 1.

This procedure will require determining the coordinates of a point that cuts off the fraction of $t \in [0, 1]$ from the edge length when moving from one end to another. This is done trivially for linear and hyperbolic edges, which are segments of straight lines, but causes difficulties for parabolic ones, which are segments of parabolas. Let the parabola equation has the form $x^2 = 2py$ in the local coordinate system. Then the length of the part of the parabola concluded between the abscissas x_1 and x_2, $x_1 \leq x_2$ is calculated as follows:

$$L(x_1, x_2) = \frac{x_2\sqrt{x_2^2 + p^2} - x_1\sqrt{x_1^2 + p^2}}{2p} + \frac{p}{2} \ln \left| \frac{x_2 + \sqrt{x_2^2 + p^2}}{x_1 + \sqrt{x_1^2 + p^2}} \right|.$$

Note that the function $x_2 = L^{-1}(l)$ for a fixed x_1 is not expressed in terms of elementary ones, so the required point cannot be found analytically. Since parabolic edges are usually quite short and can be successfully approximated by linear ones, we find the approximation of the required point: a point lying on a parabola for which the segment connecting the point and the focus of the parabola is located to the OY axis at angle $(1 - t)\alpha_1 + t\alpha_2$, where α_1 and α_2 are the corresponding angles for x_1 and x_2.

2.4 Convolutional Layer

Image convolution operation is a linear combination of pixel features from the chosen pixel neighbourhood. Similarly, we can form a neighbourhood for each node from r closest nodes in the graph and if there are less than r accessble nodes we can complete the neighbourhood using dummy nodes with feature values equal to zeros. To remove the random part from this we use additional criteria:

1. List of the lengths of the shortest paths to nodes already selected for the neighbourhood;
2. Betweenness centrality;
3. Closeness centrality.

The general outline is similar to the skeleton node ordering procedure.

The graph neighbourhood convolution can be formally defined in the following way. Let $F \in \mathbb{R}^{n\times m}$ is the features matrix, $P \in \mathbb{N}_0^{n\times r}$ — neighbourhood elements indices matrix. Then $E(F,P) \in \mathbb{R}^{n\times r\times m}$ and

$$e_{ijk} = \begin{cases} f_{p_{ij}k}, & \text{if } p_{ij} \geq 0 \\ 0, & \text{if } p_{ij} = 0 \end{cases}$$

Now if $W \in \mathbb{R}^{r\times m\times q}$, the convolution $C(E,W) \in \mathbb{R}^{n\times q}$ result is defined as:

$$c_{ij} = \sum_{1\leq t\leq r, 1\leq s\leq m} e_{its} w_{tsj}.$$

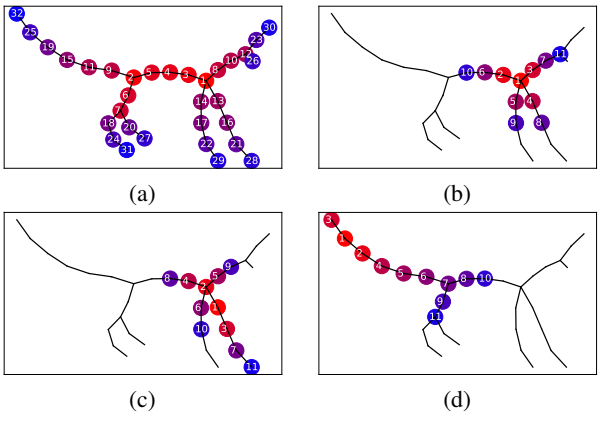

Figure 2. Node ordering (a) and length 11 neighbourhood construction for selected nodes (b,c,d). The color changes from red to blue according to the position in the list.

Neighbourhood construction examples and node ordering result are represented in fig. 2.

2.5 Downsampling Layer

The goal of the downsampling layer is to lower data dimension. We will split the graph nodes into groups in such a way that subgraphs composed of the nodes in each group would be continuous. A method can be proposed that splits nodes into pairs and removes a pair of a terminal node and its adjacent node from the graph if the degree of the latter does not exceed 2. If this is impossible, a number of heuristic rules is used to resolve collisions. This procedure produces a new graph where pairs are considered adjacent if there is at least one edge which connects nodes from different pairs. After repeating this procedure k times we can split the graph nodes set into groups of 2^k (fig. 3).

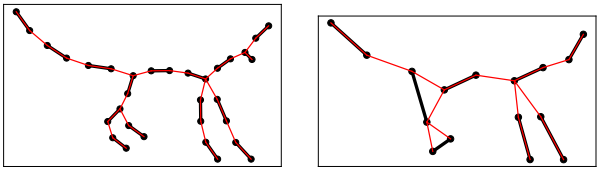

Figure 3. Merging nodes into pairs and graph simplification by replacing a pair of nodes with a single new one

Now we can define the architecture of a “skeletal” network with the same concepts that are used for conventional convolutional networks.

3. AUTOENCODER DATA PREPARATION

To use the autoencoder we need a slightly different data format. If we input the result of the pre-convolution indexation $E(F,P)$, the data would repeat numerous times because the node can appear in the neighbourhoods of several other nodes. Also, this notation loses the adjacency information because the neighbourhood doesn't consist only of adjacent nodes. We propose the following approach: nodes are ordered (as for the convolutional network), then an adjacency matrix is constructed and then, to avoid data redundancy, values that are positioned above the main diagonal of this matrix are appended to the node features. This way for the unified skeleton with n nodes its description serving as an input for the autoencoder would consist of $3n + \frac{n(n-1)}{2}$ values: $3n$ feature values and $\frac{n(n-1)}{2}$ nodes adjacency indicators. The functional to be optimized is defined according to formula **??**.

$$F(Y,\widehat{Y}) = \mathrm{MSE}(Y',\widehat{Y}') + \lambda H(Y'',\widehat{Y}'') = \\ = \frac{1}{3n}\sum_{i=1}^{3n}(y_i - \widehat{y}_i)^2 - \lambda\frac{2}{n(n-1)}\times \\ \times \sum_{i=3n+1}^{3n+n(n-1)/2} (y_i \ln \widehat{y}_i + (1-y_i)\ln(1-\widehat{y}_i)), \quad (1)$$

i.e. $\widehat{Y}''$ will evaluate the probability of corresponding nodes being connected by an edge.

4. ARCHITECTURE WITHOUT UNIFICATION

An alternative way to use convolutional neural networks for processing skeletons is to use a graph convolutional architecture that

does not require graph unification. For comparison, the algorithm was implemented based on the architecture described in (Kipf and Welling, 2016). The network in this architecture is fully-convolutional and consists of a sequence of layers described by the formula:

$$H^{(l)} = \sigma\left(\widetilde{D}^{-\frac{1}{2}}\widetilde{A}\widetilde{D}^{-\frac{1}{2}}H^{(l-1)}W^{(l)}\right).$$

Here l is the layer number, $H^{(l)} \in \mathbb{R}^{N\times D}$ are the activation values for the layer l, $\widetilde{A} = A + I_N$ is the adjacency matrix of the graph with added loops at each node: A is the adjacency matrix, and I_N is the identity matrix of size N. $\widetilde{D}_{ii} = \sum_j \widetilde{A}_{ij}$, $W^{(l)}$ is the trained matrix of weights of the layer l, and σ is an activation function. On each layer, the output of the previous layer (on the first layer, the matrix of attributes of the original graph is used) is multiplied on the left by the symmetric Kirchhoff normalized matrix for the given graph, and on the right — on the matrix of the layer weights.

Since such an operation spreads information only through the immediate neighbors of a graph node, a dummy node is added to the graph, which is connected to all the others. Also, the label of the graph in the classification is determined by the output of the last convolutional layer for this node.

5. EXPERIMENTS

Experiments were condicted on the popular MNIST base of handwritten digits. The images were first binarized with Otsu's method, then their skeletons were calculated which were later pruned with the pruning parameter of 1 pixel. Unified skeletons consisted of 32 nodes. The network architecture comprised the following layers:

1. Convolutional layer of the size $7 \times 3 \times 16$ (neighbourhood size 7, 3 features — x, y, r and 16 filters) + ReLU activation function ($f(x) = x^+ = \max(0, x)$).
2. Convolutional layer of the size $9 \times 7 \times 32$ + ReLU.
3. Fully connected layer of the size 256 + ReLU.
4. Convolutional layer with output feature dimensionality 10.
5. Softmax layer ($\sigma(z)_i = \frac{e^{z_i}}{\sum_{k=1}^{K} e^{z_k}}$) to determine the resulting class label.

The dropout regularization was performed after each convolutional layer, neurons were dropped with 50% probability. The Adam (adaptive moment estimation) method (Kingma and Ba, 2015) was used for optimization, batch size was equal to 100 objects, starting learning rate was equal to 0.001 and was halved exponentially every 20000 iterations. The total number of iterations was 100000.

The classification accuracy was equal to 98.36%. It is somewhat lower than the result that the most effective graph neural networks ChebNet (Defferrard et al., 2016) and MoNet (Monti et al., 2017) show when each graph node corresponds to an image pixel (table 1). However, it should be noted that in that case all graphs have similar size and structure, so the network architecture is close to the conventional convolutional neural networks like (LeCun et al., 1998). Structure non-uniformity only appears when superpixels are extracted from the image, and the skeleton edges can be thought of as a special way to describe these superpixels. Therefore, for arbitrary graphs the proposed method exceeds existing analogues. We also note the compactness of the skeletal description: each graph only has 32 nodes with 3 features; with this description length as a requirement our method's advantages will become even more prominent. Also, it should be noted that in most cases errors were caused by flaws in the skeleton topology (cycles breaking, extra branches appearing) as a result of a flawed segmentation that can be caused by several pixels changing intensity, however, this is not critical if objects with similar defects are present in the training sequence. We also have to admit that the downsampling layer doesn't increase the quality with similar architecture.

Method	LeNet5 (LeCun, 1998)	ChebNet (Deferrand, 2016)	MoNet (Monti, 2017)
Full grid	99.33%	99.14%	99.19%
$\frac{1}{4}$ grid	98.59%	97.70%	98.16%
300 superpixels	–	88.05%	97.30%
150 superpixels	–	80.94%	96.75%
75 superpixels	–	75.62%	91.11%

Table 1. Classification performance for various input formats

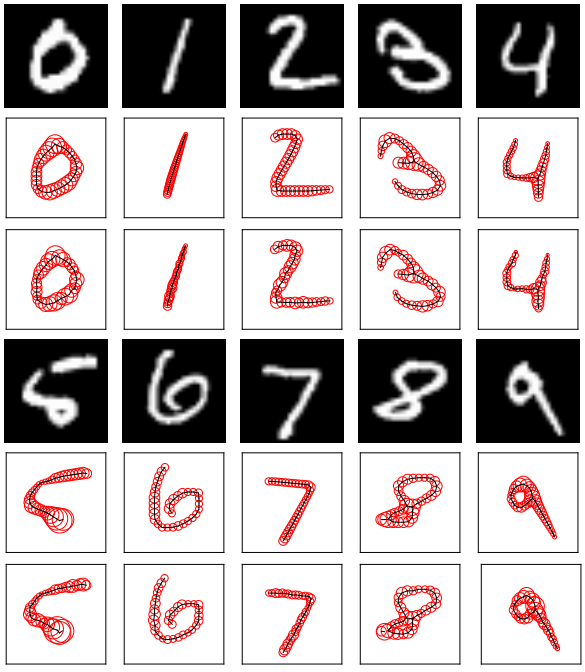

Figure 4. Autoencoder results. Source images (1st and 4th rows), their skeletons (2nd and 5th rows) and skeletons reconstructed with the autoencoder (3rd and 6th rows).

The autoencoder was trained layer by layer using the method described in (Hinton and Salakhutdinov, 2006). Layer sizes sequence was $[592, 800, 400, 200, 30]$, then the same sequence in backwards order. Sigmoidal activation functions were used except the middle layer and the part of the last layer representing the features, the identical function was used there. The parameter λ in optimized functional 1 was set to 10. The skeleton re-

construction examples demonstrated in fig. 4 show that the algorithm determines the node positions well but sometimes makes mistakes in the graph itself by adding extra edges or removing existing ones like with the number 8 skeleton. The edges with the corresponding last layer element value more than 0.5 are shown here.

In the final experiment for architecture of this type we composed the dataset from the ParaType company fonts collection (Yakupov et al., 2015). The font collection consisted of 2543 samples including not only conventional fonts but also decorative and designer fonts where symbols could take quite unorthodox shapes. For each font 26 capital Latin symbols were rasterized with the scale chosen so that the capital H would be 100 pixels tall. Then the skeletons were constructed for each of these images and their bounding box center was moved to the origin point. Images from every fifth font in alphabetical order were considered test images, others were used for training. The skeletal network architecture was mostly similar to the one described earlier, but due to larger variance of skeletal fragments and larger classes count the amount of convolutional layers filters was increased from 16 and 32 to 24 and 48 respectively. This network achieved the classification accuracy of 98.21%.

Figure 5. Atypical fonts examples

As an alternative we trained the convolutional network that operated directly on images. The symbol images were scaled to 32×32 size. The network also consisted of two convolutional layers with ReLU and max-pooling with dropout operation, and two fully connected layers. Convolutional layers sizes were set to 5×5, their filter number to 32 and 64, max-pooling used 2×2 fragments. The first fully connected layer had 1024 neurons. This network achieved the result of 97.96%. It's interesting to note that despite the small size of source images the network showed a tendency for overfitting: classification accuracy on the training set was close to 100%. This effect persisted for similar architecture networks designed for larger source image sizes. The skeletal network had a much less significant gap between the training and testing sets. This is caused by the nonstandart shape fonts where symbols have unique skeletons which the network turns up to be unable to learn. Some examples of such fonts are shown in fig. 5. Thus, most errors are caused by the same “difficult” fonts.

For an alternative architecture without graph unification, the network consisted of the following layers:

1. Two convolutional layers with the dimension of attributes at output 32 + ReLU.
2. Four convolutional layers with dimension 64 + ReLU.
3. A convolutional layer with dimension 10.
4. Softmax function layer for determining the final class label.

Despite the more complex architecture, the classification accuracy was only 90%. This indicates that the chosen approach, which was successfully used in (Kipf and Welling, 2016) for the task of classifying scientific papers, is difficult to transfer to the task of classifying images. This is probably due to the graph convolution operator, as a result of which the information on the relative position of the nodes may be lost. Another source of difficulties is the lack of a couterpart of a fully connected layer for such data, which entails a complication of the architecture.

6. CONCLUSION

In this paper we presented a deep learning method using the graph convolution approach for the image classification task. We proposed an approach allowing us to use the medial representation, which is an arbitrary structured graph, for classification problem solution. The proposed approach, based on the preprocessing and regularization of the medial representation, achieved on MNIST dataset quality comparable to conventional neutal network algorithms, and on synthetic images set from the wide fonts collection its result slightly exceeded the neural network analog which treated the image as a matrix of pixels.

ACKNOWLEDGEMENTS

The work was funded by Russian Foundation of Basic Research grant No. 17-01-00917.

REFERENCES

Bronstein, M., Bruna, J., LeCun, Y., Szlam, A. and Vandergheynst, P., 2017. Geometric deep learning: going beyond euclidean data. *IEEE Signal Processing Magazine* 34(4), pp. 18–42.

Defferrard, M., Bresson, X. and Vandergheynst, P., 2016. Convolutional neural networks on graphs with fast localized spectral filtering. In: *Proceedings of the 30th International Conference on Neural Information Processing Systems*, pp. 3844–3852.

Du, Y., Wang, W. and Wang, L., 2015. Hierarchical recurrent neural network for skeleton based action recognition. In: *2015 IEEE Conference on Computer Vision and Pattern Recognition*, Vol. 34(4), pp. 1110–1118.

Hinton, G. and Salakhutdinov, R., 2006. Reducing the dimensionality of data with neural networks. *Science* (313), pp. 504–507.

Kingma, D. P. and Ba, J., 2015. Adam: A method for stochastic optimization. In: *3rd International Conference on Learning Representations, ICLR 2015, San Diego, CA, USA, May 7-9, 2015, Conference Track Proceedings*.

Kipf, T. N. and Welling, M., 2016. Semi-supervised classification with graph convolutional networks. *arXiv preprint, arXiv:1609.02907*.

LeCun, Y., Bottou, L., Bengio, Y. and Haffner, P., 1998. Gradient-based learning applied to document recognition. In: *Proceedings of the IEEE*, pp. 2278–2324.

Li, C., Wang, P., Wang, S., Hou, Y. and Li, W., 2017. Skeleton-based action recognition using LSTM and CNN. In: *2017 IEEE International Conference on Multimedia & Expo Workshops*, pp. 585–590.

Mestetskiy, L., 2009. *Continuous Morphology of Binary Images: Figures, Skeletons, Circulars [In Russian]*. FIZMATLIT.

Mestetskiy, L., 2010. Skeleton of polygonal figure — representation by a planar linear graph [In Russian]. In: *Proceedings of the 20th International Conference GraphiCon-2010*, IFMO, pp. 222–229.

Monti, F., Boscaini, D., Masci, J., Rodolà, E., Svoboda, J. and Bronstein, M., 2017. Geometric deep learning on graphs and manifolds using mixture model CNNs. In: *2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 5425–5434.

Niepert, M., Ahmed, M. and Kutzkov, K., 2016. Learning convolutional neural networks for graphs. In: *Proceedings of the 33rd International Conference on International Conference on Machine Learning*, Vol. 48, pp. 2014–2023.

Yakupov, E., Petrova, I., Fridman, G., Korolkova, A. and Levin, B., 2015. *2008-2014 — PARATYPE Originals — Digital Typefaces*. Moscow.