JCIM JOURNAL OF CHEMICAL INFORMATION AND MODELING

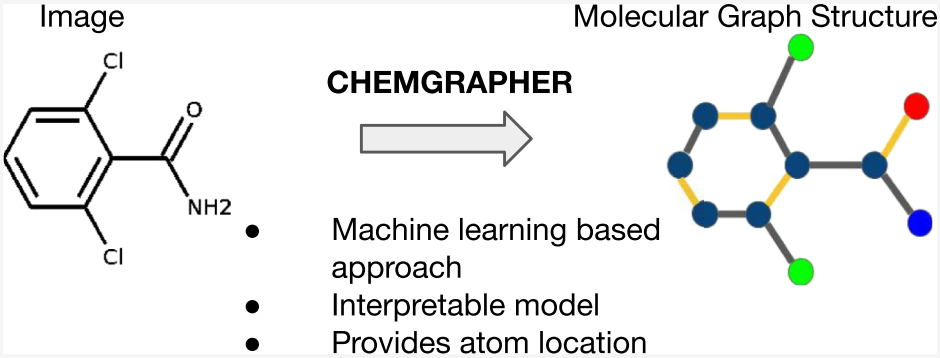
pubs.acs.org/jcim Article

ChemGrapher: Optical Graph Recognition of Chemical Compounds by Deep Learning

Martijn Oldenhof,* Adam Arany,* Yves Moreau,* and Jaak Simm*

Cite This: J. Chem. Inf. Model. 2020, 60, 4506–4517 Read Online

ACCESS | Metrics & More | Article Recommendations | Supporting Information



ABSTRACT: In drug discovery, knowledge of the graph structure of chemical compounds is essential. Many thousands of scientific articles and patents in chemistry and pharmaceutical sciences have investigated chemical compounds, but in many cases, the details of the structure of these chemical compounds are published only as an image. A tool to analyze these images automatically and convert them into a chemical graph structure would be useful for many applications, such as drug discovery. A few such tools are available and they are mostly derived from optical character recognition. However, our evaluation of the performance of these tools reveals that they often make mistakes in recognizing the correct bond multiplicity and stereochemical information. In addition, errors sometimes even lead to missing atoms in the resulting graph. In our work, we address these issues by developing a compound recognition method based on machine learning. More specifically, we develop a deep neural network model for optical compound recognition. The deep learning solution presented here consists of a segmentation model, followed by three classification models that predict atom locations, bonds, and charges. Furthermore, this model not only predicts the graph structure of the molecule but also provides all information necessary to relate each component of the resulting graph to the source image. This solution is scalable and can rapidly process thousands of images. Finally, we empirically compare the proposed method with the well-established tool OSRA[1] and observe significant error reduction.

Downloaded via KU LEUVEN on October 28, 2020 at 09:21:35 (UTC).
See https://pubs.acs.org/sharingguidelines for options on how to legitimately share published articles.

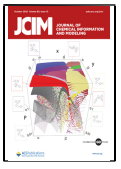
INTRODUCTION

Knowledge of the chemical structure of compounds is central in drug discovery because this structure determines the properties of the compound. It is, for example, used for drug candidate selection. Because investment of billions of euros for research and development is needed to successfully bring a new drug to the market, tools that improve the drug candidate selection process have a significant pharmaceutical impact.

Although chemical structures, which are the familiar graph drawings of molecules, lose some information about the electronic structure of a molecule (which is actually responsible for its chemical properties), they are powerful and effective abstractions. To query such structures or apply machine learning, we need to start from a well-structured data set encoding the graph representation of the chemical compound. This encoding step, which is usually less flexible than an arbitrary drawing, might also lose some information about the chemical structure, but it will provide a solid starting point for further automated processing. Popular formats for representing chemical compounds are, for example, SMILES[2] and MOLfile,[3] which contain all necessary information to build the complete molecular graph. Using these formats, it would, for example, be possible to query databases for specific patterns in chemical compounds. However, many sources, such as scientific journals and patents, do not provide such encodings or do not make them systematically available.

Thousands of scientific publications describe new chemical compounds and investigate their properties. However, the structure of these chemical compounds is usually described in

Received: April 30, 2020
Published: September 14, 2020



ACS Publications © 2020 American Chemical Society

https://dx.doi.org/10.1021/acs.jcim.0c00459
J. Chem. Inf. Model. 2020, 60, 4506–4517

the publication only as an image. This means that today a rich source of data, which would be extremely valuable to develop novel machine-learning approaches or simply query documents more accurately, is largely underexploited. It is therefore important to convert images of chemical structures into these formats. A few tools for recognizing graph structures from chemical compound images are available, such as OSRA,[1] ChemReader,[4] Kekule,[5] CLiDE Pro,[6] and the work of Sadawi *et al.*[7] However, we observed that, using these tools, bond multiplicity and stereochemical information are sometimes lost. Those tools are mainly expert systems using different techniques, such as image processing, optical character recognition, hand-coded rules, or sophisticated algorithms. Modifying or further improving these tools requires significant effort. A tool based on machine learning, which learns directly from training data, would be most valuable. Such a tool could potentially become more accurate than the existing methods and its performance could be improved by increasing the size and the diversity of the data sets, instead of having to modify its code.

Therefore, we propose a new data-driven machine-learning tool that can learn to recognize the chemical structure graph given only an image of the chemical structure. The core of the tool is a deep-learning model. In the work of Staker *et al.*,[8] another deep-learning model was also proposed. However, there the output is only a text sequence representing the graph. By contrast, we focus on directly predicting the graph structure (*i.e.*, identifying all the nodes and the edges and their labels). The positions of these nodes and edges in the resulting graph would correspond to the positions in the original image of the chemical structure, which makes our approach interpretable. The resulting graph can be later translated into any format (*e.g.*, SMILES). Stereochemical information can also be encoded in a 2D representation of a molecule. This stereochemical information is important to differentiate molecules with the same molecular formula, but with a different spatial orientation. To encode this central chirality, different types of lines are used to represent bonds in the 2D representation of a molecule: solid lines, wedge-shaped lines, or dashed lines.[9] It is important that this information is also captured correctly by our graph recognition tool.

In the next sections, we will describe the method, the neural networks it uses, and also how the different networks interact. Then, we describe the data sets used for training. Finally, we focus on the performance of our method and conclude with future work.

RELATED WORK AND BACKGROUND

ChemGrapher is a machine-learning-based image-processing tool, which can be very useful in the drug discovery process. In recent years, machine learning has made a major impact in both the fields of image processing and drug discovery. We will highlight some of the machine-learning techniques used in both fields and how they apply to ChemGrapher.

Image Processing and Machine Learning. A (deep) convolutional neural network[10,11] is the type of network most often preferred (instead of a fully connected network) for image recognition. The number of weights and connections needed by a fully connected network to be able to deal with an 2D array input like an image makes it very expensive in terms of memory, computation, and sample complexity. A convolution neural network, however, only makes "local" connections (*e.g.*, receptive field) with a previous layer which reduces the number of connections and weights needed drastically. These local connections make sense in the processing of an image as most of the correlations in an image are spatially local. Convolutional neural networks are used for several tasks in the field of computer vision including (1) simple image classification, which classifies an image as a whole, and (2) image semantic segmentation, which classifies each pixel in the image. We combine both of these tasks in ChemGrapher.

The first step in the ChemGrapher workflow is semantic segmentation of the image of the chemical compound. The aim of image segmentation is to classify each pixel of the image and assign it to a particular type of segment. In this part, ChemGrapher splits the image in different segments of atoms, bonds, and charge types. The main goal of segmentation of the image is to make the latter step of classification of atoms/bonds/charges easier.

Our work builds upon the recent developments in image segmentation. Different machine-learning approaches[12,13] can be used for the semantic segmentation of images. One well-established approach is U-Net.[12] This approach uses a network that combines a contracting path and an expanding path. Several other works were based on the U-Net approach, such as Jansson *et al.*,[14] where a U-Net is used to extract the vocal component from music. Other works expanded this U-Net approach, such as Çiçek *et al.*,[15] which generalizes the U-Net approach to 3D images.

ChemGrapher, however, uses an alternative approach to U-Net. For the segmentation step, we make use of dilated convolutions,[16] all stacked without decreasing the resolution of the layers. The main advantage of dilated convolutions is that the receptive field can grow exponentially by increasing the dilation in the dilated convolutional operator without increasing the number of parameters. First, this is computationally more efficient than using multiple convolutions or larger kernels, and second, fewer parameters also mean that the network requires less training data.

After the segmentation step, ChemGrapher also uses several classification networks to classify the different segments located in the image segmentation step. There has been a trend to create deeper and deeper neural networks to improve the performance for the classification of images. However, deeper networks could have convergence issues while training because of vanishing gradients.[17] To improve the training capabilities of such deep networks, methods such as residual neural networks (Resnet),[18,19] batch normalization,[20] and ELU[21] have been developed. However, these methods are not needed in our work as the classification networks used in ChemGrapher are relatively shallow, given the segmentation step simplified the classification tasks.

Drug Discovery and Machine Learning. There are several stages in the process of drug discovery. The stages go from basic research and drug candidate selection to the development phase, clinical trials, and finally production. As development progresses further and sunken costs increase, the cost of failure of a project thus increases. "Fail early" is thus important to contain the costs of drug discovery. Predicting risks of failure later in the discovery process (*e.g.*, by predicting toxicity for a compound) without draining the pipeline (enough candidate compounds need to remain available) is essential. Machine-learning techniques can be used at all stages of drug discovery. Chen *et al.*[22] gives a good overview of the recent use of deep learning in drug discovery. We would like to highlight some of these recent applications, which we find interesting in the context of our graph recognition tool.

https://dx.doi.org/10.1021/acs.jcim.0c00459
J. Chem. Inf. Model. 2020, 60, 4506−4517

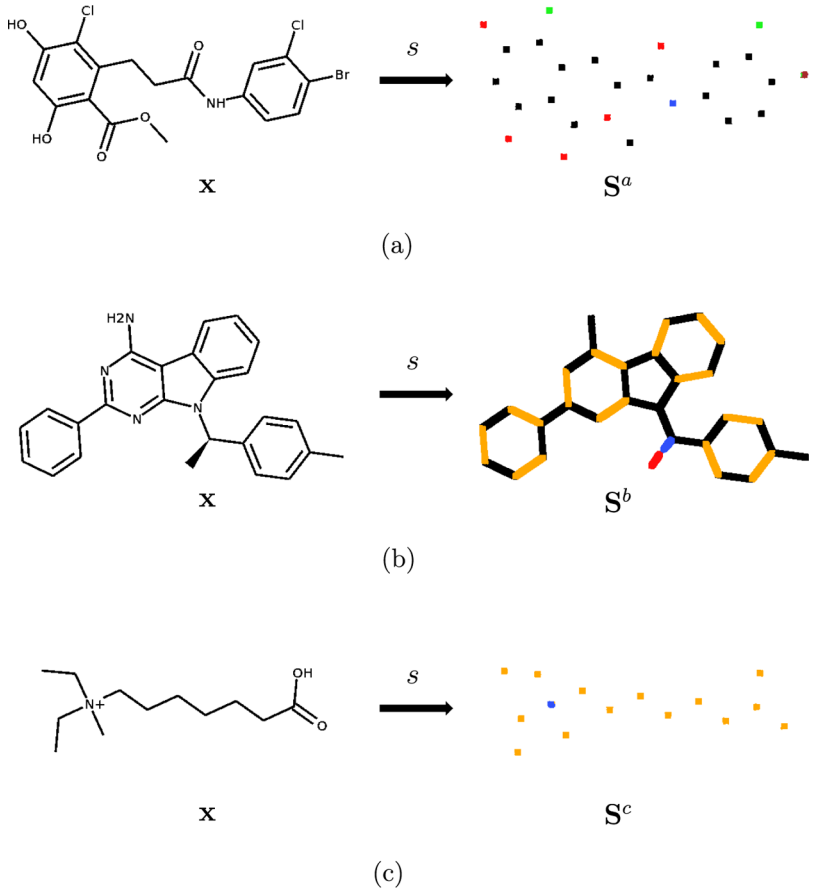


Figure 1. Three figures showing the atom-, bond-, and charge-segmentation. On the left, we have the input image $\mathbf{x}$. On the right, we have the resulting atom ($\mathbf{S}^a$)-, bond ($\mathbf{S}^b$)-, and charge ($\mathbf{S}^c$)-segmentation. (a) Atom segmentation. (b) Bond segmentation. (c) Charge segmentation.

In the first place, there is the work from Xu *et al.*,[23] Winter *et al.*,[24] and Gómez-Bombarelli *et al.*,[25] where unsupervised methods are used to extract features from only SMILES input data. SMILES (simplified molecular identification and line entry system)[2] is a *de facto* standard for textual representation of chemical compounds. SMILES encodes the molecule as the traversal of the spanning tree of its graph. The aforementioned works[23−25] propose unsupervised learning approaches using the autoencoder principle. The resulting vector-based representations of molecules can then be used as input to supervised methods to learn to predict molecular properties (*e.g.*, bioactivity or lipophilicity).

Another interesting method to predict the molecular properties of a chemical compound is to use the neural graph fingerprint presented in Duvenaud *et al.*[26] The neural graph fingerprint is a way to represent and encode a chemical compound. Here, a graph convolutional neural network takes the graph as input and is trained to predict molecular properties. Similarly, more general machine-learning approaches that work directly on the graph representation have been proposed in Kearnes *et al.*,[27] Coley *et al.*,[28] Simm *et al.*,[29] and Pires *et al.*[30]

Large amounts of data are needed to use or train the models mentioned above. It is not always easy to find these data. This is where our tool is useful by extracting graph representations of chemical compounds directly from images. It is also worth mentioning the work presented in Goh *et al.*,[31] where no graph representation of the chemical compound is needed; instead, a machine-learning model is trained to predict the bioactivity directly from the images of chemical structures.

Problem Statement. We now formulate our learning task. The goal of the proposed method is to learn a function that maps an image $\mathbf{x}$ to its graph representation G.

Definition 1. $\mathbf{x} \in \mathbb{R}^{U \times V}$ represents a single-channel 2D image with dimensions $U \times V$.

Generalizations to multiple channels are straightforward if colored images are available.

Definition 2. $G = (V, E)$ represents a graph with labeled vertices V and labeled edges E. The vertices V and the edges E represent the different atoms and bonds, respectively, of the chemical compound in the image x.

For our graph recognition tool to work, we need to learn the following function

$$g(\mathbf{x}) \rightarrow G \tag{1}$$

This function will map a 2D input image of a chemical structure to the graph representation of the molecule. To learn this function, we assume the availability of training data in the form of labeled images of chemical structures. The images are assumed to be labeled pixelwise, therefore knowledge is needed about the pixel coordinates of every vertex (atom) in the resulting graph and the existence of the edges (bonds) and their labels.

MODEL

To learn the map g from data, we build a machine-learning model. The model is split up into different learning tasks: (1) semantic segmentation and (2) segment-type classification.

https://dx.doi.org/10.1021/acs.jcim.0c00459
J. Chem. Inf. Model. 2020, 60, 4506−4517

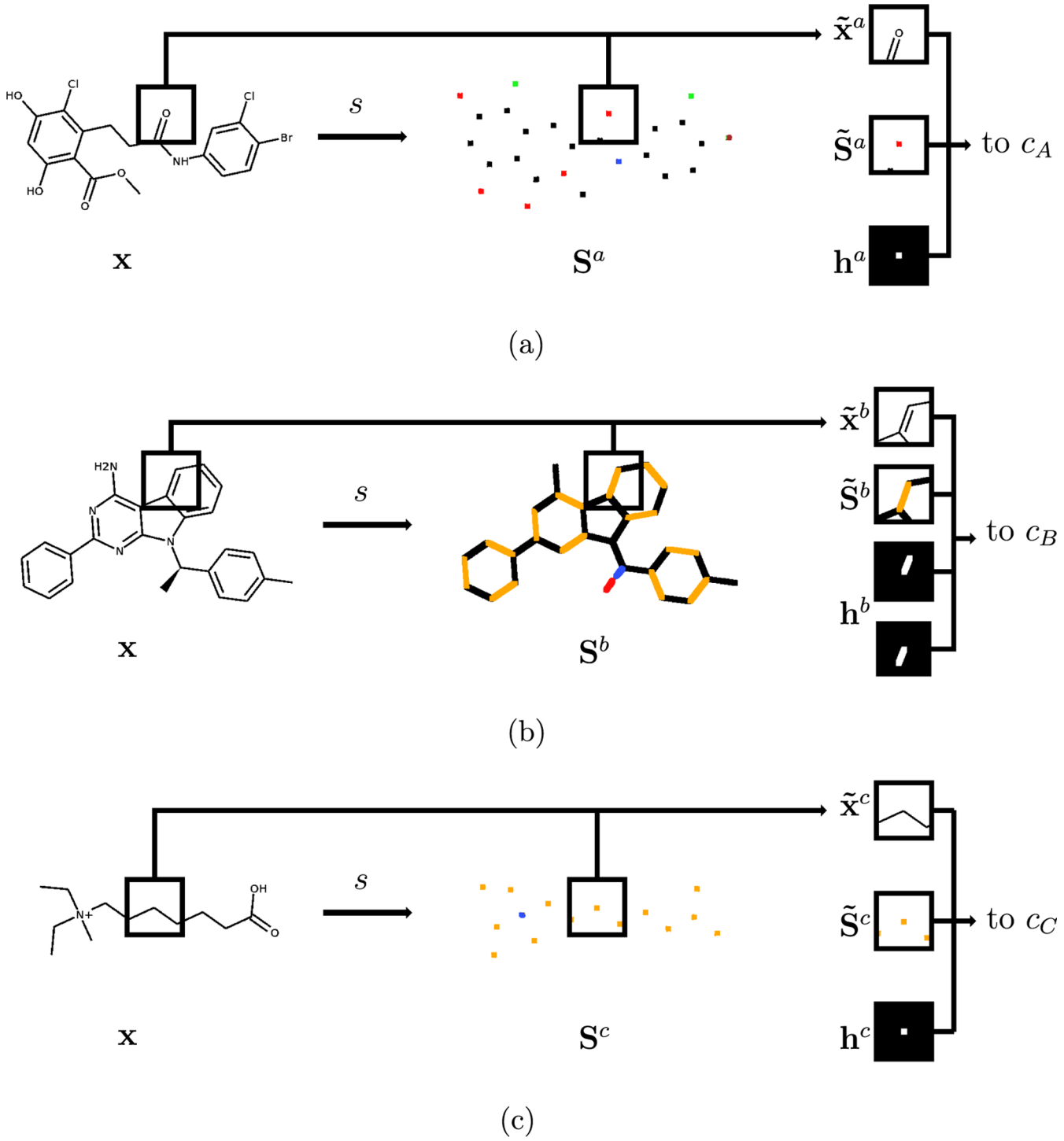


Figure 2. To build the input for the different classifiers c_A, c_B, and c_C, the outputs of the segmentation network $\mathbf{S}^a$, $\mathbf{S}^b$, and $\mathbf{S}^c$ are cut out to feed to the classification network. This cutout is illustrated by a rectangle in the middle ($\tilde{\mathbf{S}}^a$, $\tilde{\mathbf{S}}^b$, and $\tilde{\mathbf{S}}^c$). To this, we also add a cutout of the original image ($\tilde{\mathbf{x}}^a$, $\tilde{\mathbf{x}}^b$, and $\tilde{\mathbf{x}}^c$) together with highlighting the candidate location ($\mathbf{h}^a$, $\mathbf{h}^b$, and $\mathbf{h}^c$) of the segment type to be classified. The complete input for each classifier is shown on the right. (a) Input for the atom-segment-type classifier. (b) Input for the bond-segment-type classifier. (c) Input for the charge-segment-type classifier.

First Task: Semantic Segmentation. The first learning task is to learn to segment a 2D image of a chemical structure, where each segment represents the location of a specific atom, charge, or bond type in the image. The image was already defined in the previous section. Here, the segmentation of this image will be defined.

Definition 3. $\mathbf{S}^a \in \mathbb{R}^{U\times V\times n_a}$, $\mathbf{S}^b \in \mathbb{R}^{U\times V\times n_b}$, and $\mathbf{S}^c \in \mathbb{R}^{U\times V\times n_c}$ represent the atom type, bond type, and charge segmentation of an image. The width U and the height V are the same as in the input image, while n_a, n_b, and n_c, respectively, are the number of atom types, bond types, and charges (including the empty atom-, charge-, and bond-types) present in the compound.

https://dx.doi.org/10.1021/acs.jcim.0c00459
J. Chem. Inf. Model. 2020, 60, 4506−4517

To perform image segmentation, we need to learn a function $s(\mathbf{x})$ that maps the image to these three segmentations

$$s(\mathbf{x}) \rightarrow \mathbf{S}^a, \mathbf{S}^b, \mathbf{S}^c \tag{2}$$

The segmentation function is illustrated in Figure 1. To learn this function, we need to pixelwise label the training elements. This process is explained in the section Data Sets.

For training, we use the cross-entropy loss H. In the case of atom-type segmentation, the cross-entropy loss is calculated and summed for every pixel prediction (thereby fixing u and v) in the following way

$$\text{loss}_a = \sum_{u=1}^{U}\sum_{v=1}^{V} H(y^a_{u,v}, \hat{y}^a_{u,v}) \tag{3}$$

where y^a is the true (one-hot encoded) label and $\hat{y}^a$ is the estimated probability distribution of the labels for atom segmentation.

The losses (loss_b and loss_c) in the case of bond-type segmentation and charge segmentation are calculated in a similar fashion. The total loss is the sum of all partial losses

$$\text{loss}_{\text{total}} = \text{loss}_a + \text{loss}_b + \text{loss}_c \tag{4}$$

Second Task: Segment-Type Classification. A second learning task is necessary to build a final graph. This learning task classifies parts of the segmented image into different possible atom-, bond-, and charge-types. For each segment type (atom, bond, and charge), a different classifier is trained using cutouts of the input $\mathbf{x}$ and predicted segmentation $\mathbf{S}$. The input for each classifier consists of three parts

1. The first part consists of the tensors $\tilde{\mathbf{S}}^a$, $\tilde{\mathbf{S}}^b$, and $\tilde{\mathbf{S}}^c$, which represent a cutout of the tensors $\mathbf{S}^a$, $\mathbf{S}^b$, or $\mathbf{S}^c$.
2. For the second part, we have $\tilde{\mathbf{x}}^a$, $\tilde{\mathbf{x}}^b$, and $\tilde{\mathbf{x}}^c$, which represent cutouts of the original 2D image $\mathbf{x}$.
3. Finally, extra highlights are also created $\mathbf{h}^a$, $\mathbf{h}^b$, and $\mathbf{h}^c$, which highlight the candidate location to be classified. For the bond classifier, the highlight of the candidate location $\mathbf{h}^b$ is split into two parts to encode the direction of the bond, which is necessary to predict the stereoisomeric bond direction.

The different inputs for the different classifiers are illustrated in Figure 2. The functions to be learned by these classifiers are $c_A(\tilde{\mathbf{S}}^a, \tilde{\mathbf{x}}^a, \mathbf{h}^a) \rightarrow Y^a$, $c_B(\tilde{\mathbf{S}}^b, \tilde{\mathbf{x}}^b, \mathbf{h}^b) \rightarrow Y^b$, and $c_C(\tilde{\mathbf{S}}^c, \tilde{\mathbf{x}}^c, \mathbf{h}^c) \rightarrow Y^c$, where $Y^a \in \{0,1\}^{n_a}$, $Y^b \in \{0,1\}^{n_b}$, and $Y^c \in \{0,1\}^{n_c}$ represent the one-hot encoded vectors for the different classifiers. Similar to the segmentation-learning task, we again use the cross-entropy loss here.

GRAPH BUILDING ALGORITHM

Once we have learned the functions described in the previous sections, we need an algorithm to combine the outputs of these functions and build up a final graph structure. We propose an iterative algorithm that first detects all atoms and then identifies bonds using the detected atoms, see Algorithm 1:

Algorithm 1: Graph building algorithm

Data: Image tensor $\mathbf{x}$

Result: Graph G

$\mathbf{S}^a, \mathbf{S}^b, \mathbf{S}^c = s(\mathbf{x})$

atomCandidates = generateAtomCandidates($\mathbf{S}^a$)

$V = []$

for atomCand *in* atomCandidates **do**

  $\tilde{\mathbf{S}}^a, \tilde{\mathbf{x}}^a, \mathbf{h}^a = \text{cutAtomCand}(\text{atomCand}, \mathbf{S}^a, \mathbf{x})$

  $\tilde{\mathbf{S}}^c, \tilde{\mathbf{x}}^c, \mathbf{h}^c = \text{cutAtomCand}(\text{atomCand}, \mathbf{S}^c, \mathbf{x})$

  $Y^a = c_A(\tilde{\mathbf{S}}^a, \tilde{\mathbf{x}}^a, \mathbf{h}^a)$

  $Y^c = c_C(\tilde{\mathbf{S}}^c, \tilde{\mathbf{x}}^c, \mathbf{h}^c)$

  if isNotEmptyAtomType(Y^a) **then**

    V.appendAtom(Y^a, Y^c, atomCand)

  end

end

bondCandidates = generateBondCandidates(V)

$E = []$

for bondCand *in* bondCandidates **do**

  $\tilde{\mathbf{S}}^b, \tilde{\mathbf{x}}^b, \mathbf{h}^b = \text{cutBondCand}(\text{bondCand}, \mathbf{S}^b, \mathbf{x})$

  $Y^b = c_B(\tilde{\mathbf{S}}^b, \tilde{\mathbf{x}}^b, \mathbf{h}^b)$

  if isNotEmptyBondType(Y^b) **then**

    E.appendBond(Y^b, bondCand)

  end

end

In the first phase, the proposed Algorithm 1 will first apply the segmentation function s to the input image. Next, using the segmentation $\mathbf{S}^a$, candidate locations (coordinates) atomCandidates will be generated by generateAtomCandidates. The function generateAtomCandidates will calculate for each segment in $\mathbf{S}^a$ the center of mass. These centers of mass will be the candidate locations (coordinates) to classify. Given these candidate locations, the nodes V of the graph can be built in an iterative way. For this purpose, the segmentations $\mathbf{S}^a$ and $\mathbf{S}^c$ can be cut (cutAtomCand) into smaller segments $\tilde{\mathbf{S}}^a$ and $\tilde{\mathbf{S}}^c$, thanks to every candidate location atomCand. At the same time, the original image $\mathbf{x}$ is also cut (cutAtomCand) into smaller parts $\tilde{\mathbf{x}}^a$ and $\tilde{\mathbf{x}}^c$. Extra highlights, $\mathbf{h}^a$ and $\mathbf{h}^c$, are also created highlighting the candidate location to be classified. Then, the algorithm applies the classification networks c_A and c_C to determine what kind of atom and charge is located at the candidate location. If the candidate location is not empty (isNotEmptyAtomType), the location, atom type Y^a, and charge Y^c will be added to the list of nodes V.

In the second phase, the algorithm will use the identified nodes V to build the edges E of the graph G. For this, first, it needs to generate (generateBondCandidates), the candidate bond locations (assigned to bondCandidates). Similarly, as for the nodes, the bonds E of the graph can be built in an iterative way. For this purpose, for each bond candidate, the segmentation $\mathbf{S}^b$ and the image $\mathbf{x}$ are cropped (by cutBondCand) into smaller tensors $\tilde{\mathbf{S}}^b$ and $\tilde{\mathbf{x}}^b$ based on the location of the candidate bond bondCand. The function cutBondCand also creates an extra tensor $\mathbf{h}^b$ that highlights the bond location to be classified. Finally, the classification network c_B is applied to determine the existence and the type of the candidate. If the candidate bond location is not empty (isNotEmptyBondType), the location and type Y^b will be added to the list of bonds E.

Deep-Learning Implementation. For the graph recognition tool, we employ a combination of different convolutional neural networks.[10] First, we have a semantic segmentation

https://dx.doi.org/10.1021/acs.jcim.0c00459
J. Chem. Inf. Model. 2020, 60, 4506−4517

network using the Dense Prediction Convolutional Network,[16,32] followed by three classification networks. As mentioned in previous sections, the output of the segmentation network is part of the input of the classification networks. Other approaches were also tried: (a) one big segmentation network without classification and (b) three separate segmentation networks (atom, bond, and charge) with three classification networks. These approaches did not perform as well. In the different networks, we tried several kernel sizes and layer structures and iteratively kept the best ones. However, we did not perform an exhaustive search in the hyperparameter space, so there could be more gains if a more thorough search is performed.

Semantic Segmentation Network. Before feeding the image to the segmentation network *s*, it is preprocessed to a binary black and white image. The output of the segmentation network is different channels predicting for every pixel in the image the class to which the pixels belong. The possible classes represent the different atom types, bond types, and charges. For the implementation of this network, we build on the concept of dilated convolution described by Yu and Koltun.[16]

Network Architecture. The network has eight 3 × 3 convolutional layers from which six layers make use of dilation. All convolutional layers are followed by a rectified linear unit (ReLU). The last layer is a linear layer. Padding is used, so that the resolution of the layers does not change. The padding and dilation for the different convolutional layers are summarized in Table 1.

Table 1. Summary of the Layers of the Segmentation Network

layer	Kernel	nonlinearity	padding	dilation
conv1	3 × 3	ReLU	1	no dilation
conv2	3 × 3	ReLU	2	2
conv3	3 × 3	ReLU	4	4
conv4	3 × 3	ReLU	8	8
conv5	3 × 3	ReLU	8	8
conv6	3 × 3	ReLU	4	4
conv7	3 × 3	ReLU	2	2
conv8	3 × 3	ReLU	1	no dilation
Last	1 × 1	none	no padding	no dilation

Classification Networks. For the atom location, the bond prediction, and the charge prediction, we use three separate classification networks. All three networks use part of the output of the segmentation networks in their input, as explained in a previous section. For every image segmentation, the classification network has to run several times to classify all cutouts (resolution 101 × 101) from the candidate locations, resulting in all atom, bond, and charge predictions in the original image, as explained in previous section. The size of the cutouts was chosen to cover a neighborhood of around 2 bond lengths. The average bond length in the original training images is 50 pixels, which resulted in a cutout of size 101 × 101. However, smaller cutouts could work and could be tested in future work. This could reduce the training compute cost.

Network Architecture. The three classification networks have similar layer structures. There are five convolutional layers, where three of them are dilated and one of them (the first one) is actually a depthwise separable convolution.[33] After the convolutional layers, there is always a ReLU layer. The last layer is a linear layer and the layer before that is a max pool layer. All layers are summarized in Table 2.

Table 2. Different Layers in the Classification Network

layer	Kernel	nonlinearity	padding	dilation
depthconv1	3 × 3	ReLU	1	no dilation
conv2	3 × 3	ReLU	2	2
conv3	3 × 3	ReLU	4	4
conv4	3 × 3	ReLU	8	8
conv5	3 × 3	ReLU	1	no dilation
global maxpool	input size	None	no padding	no dilation
Last	1 × 1	None	no padding	no dilation

Data Sets. To build our data sets for the segmentation network and the classification networks, we downloaded and split different chemical structures in the SMILES format from the ChEMBL[34] database. The set of 1.9 million chemical structures is split into four nonoverlapping parts:

1. A training pool for the segmentation network of 1.5 million chemical structures
2. A pool of 300 K chemical structures used for the validation of the segmentation network and training of classification networks
3. A pool of around 35 K chemical structures for the validation of the classification networks
4. Another pool of 35 K chemical structures to test the overall performance

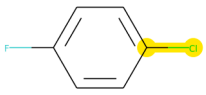
From these pools, we can sample the actual data sets for our different networks. By sampling, we can control the relative frequency of different atom types and bond types in the actual data sets. This is important for the performance of our networks because of data imbalance for the different atom types and bond types, as we will see in the next section.

Segmentation Data Set. For the training of the segmentation network (*s*), we need 2D images of chemical structures together with pixelwise-labeled target values. This type of data set is not available as far as we know, so we need to construct this data set ourselves. Labeling thousands of 2D images of chemical structures pixelwise by hand is not feasible. We thus construct an automatic procedure to generate this data set. For the training data set, we sample around 114,000 chemical compounds in the SMILES format from the ChEMBL training pool in a way that every atom type is present in at least 1000 chemical compounds. Using RDKit[35] in Python, we create the images starting from the SMILES. Furthermore, to create the labeling, we make some modifications[36] in the code of RDKit at the drawing time of the image, so that it additionally produces the necessary labeling and location information needed to create the true segmentation mask for each image. In Table 3 an example is shown of an RDKit-generated image with the corresponding labels. For the validation data set, we use the same procedure. To make the method more robust for style changes, we vary the font, font size, line width, and offset between multiple bonds while creating the images. Additionally, we also add some salt and pepper noise to 10 percent of the images.

Atom Classification Data Set. Once the segmentation network is trained and validated, we can sample from the ChEMBL classification training pool a new data set to feed into the segmentation network. The output of these runs is saved to create the input data set for the next classification networks. As

https://dx.doi.org/10.1021/acs.jcim.0c00459
J. Chem. Inf. Model. 2020, 60, 4506−4517

Table 3. RDKit Labeling Example: On the Left, We Have the Original SMILES; in the Middle, We Have the RDKit-Generated Image; and on the Right, We Have the Resulting Labels Generated by Our Modified[36] Version of RDKit[a]

SMILES	Generated Image	Generated Labels
c1cc(F)ccc1Cl		idx,mol,bond,at1,ch1,at2,ch2,x1,y1,x2,y2 0,0,2.0,C,0,C,0,590,685,376,685 1,0,1.0,C,0,C,0,590,685,697,500 2,0,1.0,C,0,C,0,376,685,269,500 3,0,1.0,C,0,F,0,269,500,55,500 4,0,2.0,C,0,C,0,269,500,376,314 5,0,1.0,C,0,C,0,376,314,590,314 6,0,2.0,C,0,C,0,590,314,697,500 7,0,1.0,C,0,Cl,0,697,500,911,500 8,0,nobond,F,0,F,0,55,500,55,500 9,0,nobond,Cl,0,Cl,0,911,500,911,500

[a]The highlights show the part in the labels that correspond to the highlights in the image and in the original SMILES

already explained in a previous section, the atom classification network (c_A) additionally expects as input the candidate locations (coordinates) to classify. For the training and validation data sets of the classification network c_A, we generate candidate locations based on the true atom location values, but also add locations where no atom is located for the prediction of the empty class. For these locations, we take the middle point of every bond in the data set. As we know that no atom is located in the middle of a bond, these locations can be used for the empty values in the data sets.

Bond Classification Data Set. For the bond prediction network (c_B), we apply a similar technique. In addition to the inputs from the previous segmentation network, the bond prediction network expects the candidate bond locations (coordinates). For the training and validation data of c_B, we generate these candidate locations by going over all possible combinations of pairs of atoms in a molecule within the range of less than two times the average bond length. In case there is a bond between a generated pair of atoms, we label that item with the corresponding bond type. If there is no bond between the pair of atoms, the item is labeled empty.

Charge-Classification Data Set. For the charge-classification network (c_C), the same data sets as the atom-prediction data sets can be used except for the labels. Instead of the atom types, the labels would now be the charge (including the empty charge) of the atom candidate.

EXPERIMENTS AND RESULTS

For implementation and training of different networks, PyTorch[37] was used. All networks were trained using a compute node with two NVIDIA Tesla v100 GPUs with 32 GB of memory. Further training details are summarized in Table 4. For the classification networks, the number of input images (shown in Table 4) is divided into a number of cutouts in order to build the different classification training data sets (atom, charge, and bond candidates), as explained in the previous section.

Table 4. Training Details for Different Networks

network	#input images (K)	#epochs	minibatch size	walltime (h)	learning rate
segm. network	114	5	8	24	0.001
stom clas.	12.4	2	16	8	0.001
charge las.	12.4	2	16	8	0.001
bond clas.	4.4	2	64	4	0.001

For validation, we sample new data sets from the validation ChEMBL pools for the different networks. For the segmentation network, we sample around 12,000 chemical structures. For the validation of the classification networks, fewer chemical structures are needed, so we only sample around 450 chemical structures. Starting from these 450 chemical structures, we generate atom candidates and bond candidates. This results in a data set of around 27,000 atom candidates for atom-type and charge-classification networks and a data set of around 55,000 bond candidate locations for the bond-classification network. We measure the performance on the different networks on these validation data sets.

Performance of the Segmentation Network. For the segmentation network (s), we measure the F1 score[38] for all the pixel predictions for the different atom-, bond-, and charge-types. The F1 score takes into account both precision and recall equally. Figure 3 plots the F1 scores of various atom- and bond-types against their frequencies in the training set. As can be seen from the figure, there is a correlation between the F1 scores and frequencies, which is expected for machine-learning models.

Performance of Classification Networks. For the classification networks, we again use the F1 score to measure the performance for the atom-, bond-, and charge-type classification. Again, we see a correlation between the F1 score and the frequency of the different types in the training data set. We can also empirically see that the F1 score for the classification networks is significantly higher than that for the segmentation networks. This is what we would expect as the segmentation network has to solve a more complicated problem than the classification networks. The segmentation network needs to get all pixels correct, while the classification network only needs to make one prediction for a limited region. Therefore, the classification networks can do a good job even when the segmentation is not perfect. The performance of these classification networks has to be very good as for every graph prediction tens of bond- and atom-classifications have to be made and this would otherwise degrade the overall accuracy rapidly. The results are also summarized in Figure 3.

Overall Graph Accuracy. Now that we know the performance of the different parts, we can combine those building blocks and measure the overall accuracy of the resulting graph predictions. As already mentioned in a previous section, we use Algorithm 1 to build the resulting graph.

Images (size: 1000 × 1000; resolution: 72 dpi) in four different styles (varying fonts, font sizes, line widths, and offset between multiple bonds) are generated using RDKit. Two of the

https://dx.doi.org/10.1021/acs.jcim.0c00459
J. Chem. Inf. Model. 2020, 60, 4506−4517

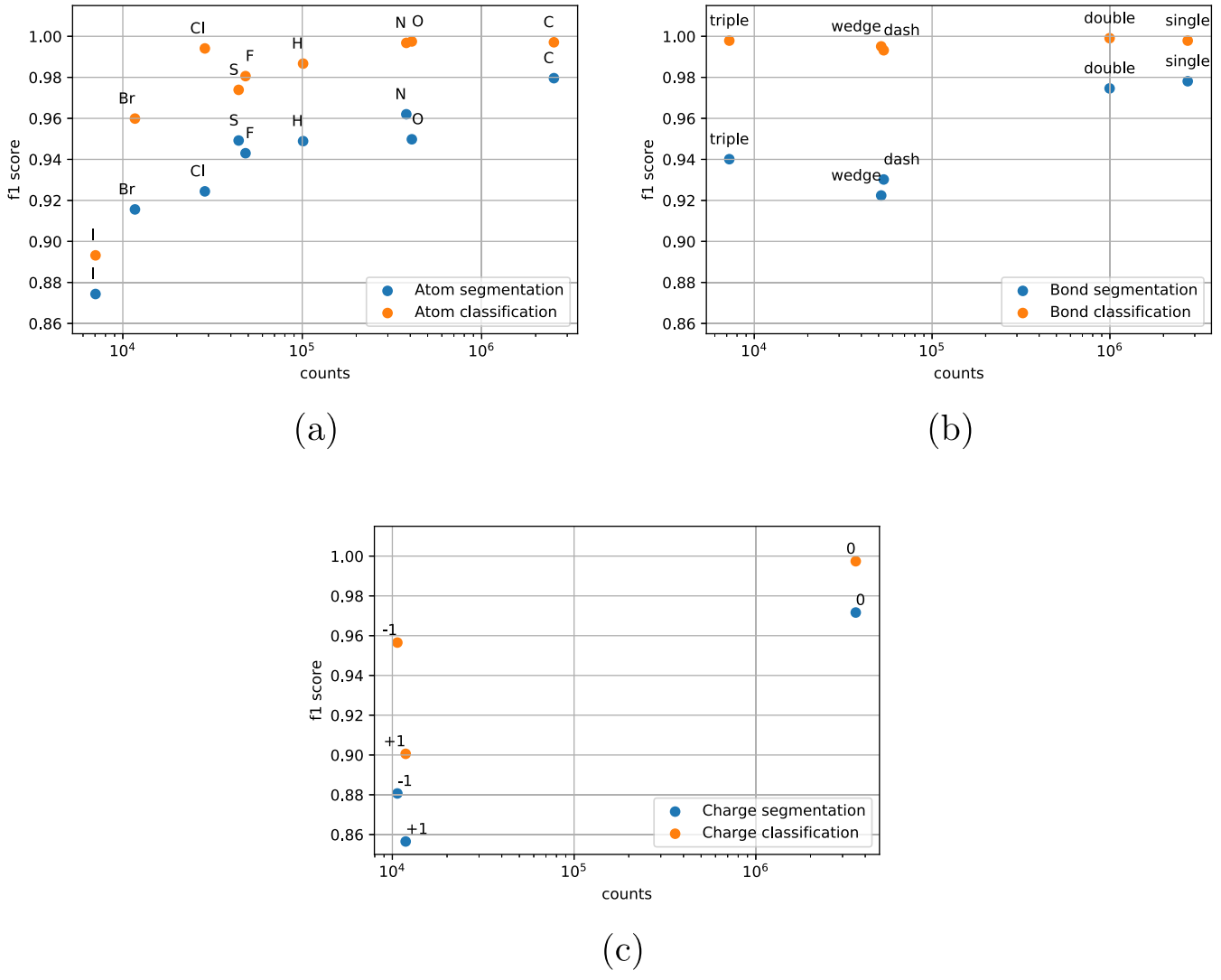


Figure 3. F1 score for segmentation and classification networks. There is clearly a correlation between the performance of the networks on different prediction types and the frequency of the specific type in the training data set. The classification networks perform significantly better than the segmentation networks. (a) Atom-prediction performance (s_A and c_A). (b) Bond-prediction performance (s_B and c_B). (c) Charge-prediction performance (s_C and c_C).

generated styles are also included in the training data set styles, while the other two styles are not. For every style, we generate two sets: (1) one set only has images without stereochemical information encoded in the compounds, while (2) the other set has images where all compounds have stereochemical information encoded. This results in eight sets of 1000 images each to measure the performance on our tool ChemGrapher. We define graph prediction to be correct if there are no mistakes in the resulting graph (*i.e.*, the graph matches exactly the true graph). We therefore compare the true canonical SMILES with the predicted canonical SMILES. For comparison, we use the same data sets to measure the performance of OSRA,[1] version 2.1.0. The results are summarized in Figures 4 and 5. We observe in almost all styles, a higher accuracy of our ChemGrapher tool compared to OSRA, for nonstereo as well as stereo images. For the computations of the ChemGrapher predictions, we used a compute node with one NVIDIA Tesla v100 with 32 GB memory. To predict the canonical SMILES of 1000 images, it took about 27 min walltime. This includes loading and initialization time of the model on the GPU. This could still be improved, as several steps can be executed in parallel. For example, while segmenting the different images using the GPU, the different atom candidates can already be calculated on the CPU for the classification step. These steps are now still implemented sequentially. For comparison, we measured the speed of OSRA on a CPU-only machine (no GPU). Using just 1 core of Intel Xeon CPU E5-2676 v3@2.40 GHz, it took only 19.5 min for OSRA to process the same 1000 images.

Case Study 1: Performance on Journal Article Images. As a case study, we explore how well ChemGrapher performs on images from journal articles. As this kind of data set is not available, we built one manually. We cut out images from journal articles about chemical compounds, preprocess them into the correct input format (size: 1000 × 1000, resolution 72 dpi), and feed them to our tool. We evaluate the resulting graph on the correctness to measure the accuracy. Similarly, as before, we denote a prediction as correct if the graph matches exactly the true graph. For comparison, we execute the same procedure for OSRA v2.1.0. The results of this experiment are summarized in Figure 6. Thus, out of a total of 61 images we tried on ChemGrapher, 12 were incorrectly predicted, while OSRA predicted 19 images incorrectly. This corresponds to a

https://dx.doi.org/10.1021/acs.jcim.0c00459
J. Chem. Inf. Model. 2020, 60, 4506−4517

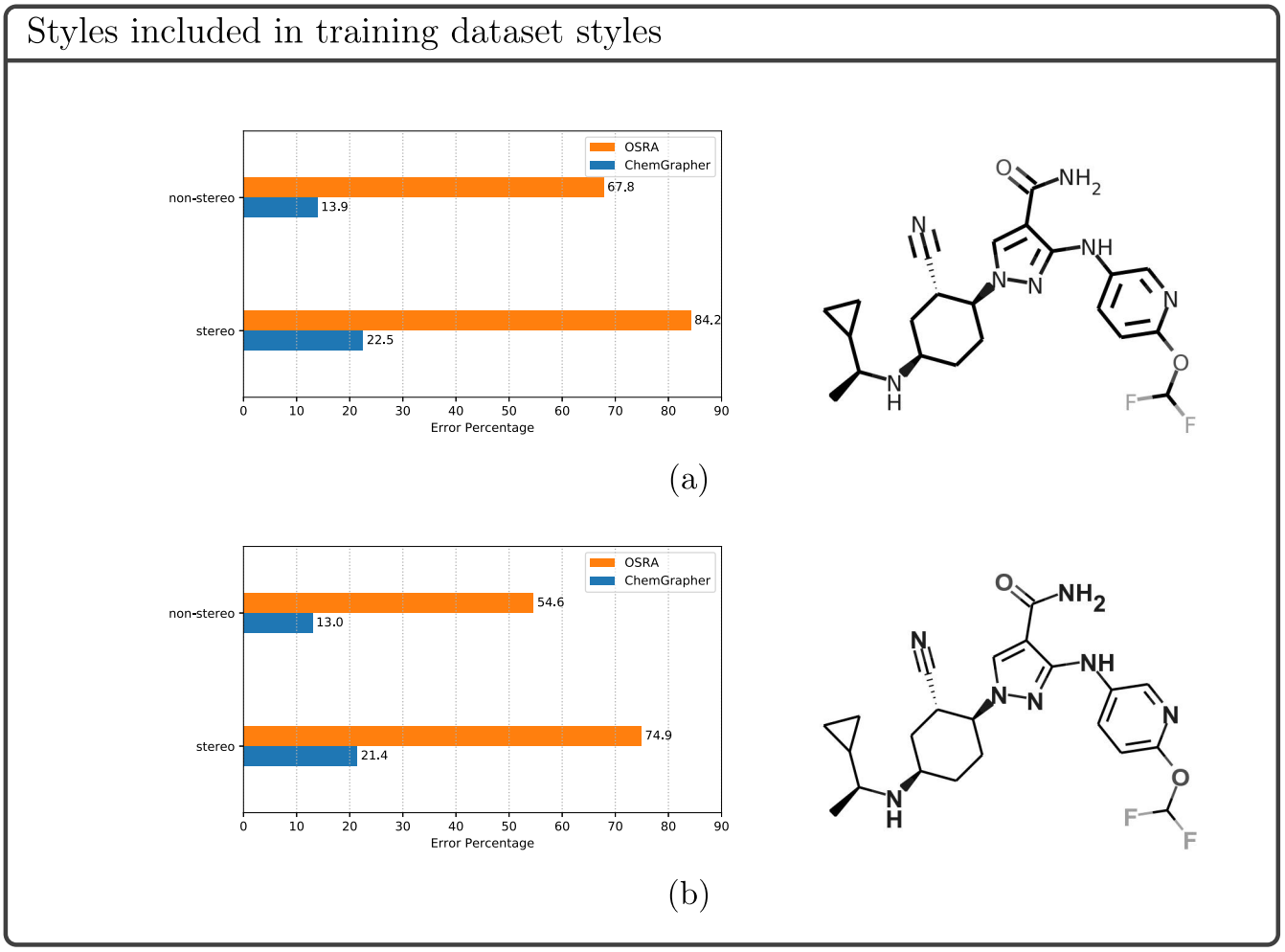


Figure 4. Graph accuracy of our tool compared with OSRA v2.1.0 measured on images (size 1000 × 1000, resolution 72 dpi) generated in different styles using RDKit. For each style, two experiments are performed: one with images without stereochemical information and one with images with stereochemical information. On the left, we observe for each style the results on the error rate. On the right, we observe for each style an example image in that specific style. (a) Error rate for style 1. (b) Error rate for style 2.

significant reduction in error. One observation we make on this case study is that ChemGrapher clearly shows a better performance on images of compounds with only carbon atoms compared to OSRA. For these compounds, typically no letters appear in the image.

Case Study 2: Performance on Maybridge Dataset. As the dataset of case study 1 is fairly limited in variability and size, we also carry out another experiment with a bigger benchmark data set[39] published by the developers of MolRec,[7] which we will call the Maybridge dataset. The Maybridge dataset contains no journal images but scanned images from Maybridge's Catalogues for drug design and discovery. These images typically also have some level of noise such as salt and pepper noise. In order to feed the images to ChemGrapher, we first pad every image to have the size of 1000 × 1000. For every image in the Maybridge dataset, a ground truth compound is given by a corresponding MOL file.[3] We convert this MOL file using RDKit[35] to canonical SMILES. Again, for comparison, we compare the canonical SMILES taken from OSRA and ChemGrapher to measure the accuracy. ChemGrapher predicts 4051 out of 5740 images correctly, which gives an accuracy rate of 70.57%, while for OSRA v2.1.0, we clearly measure a better performance of predicting 4677 images correctly, which gives an accuracy rate of 81.48%. Analyzing the error cases of ChemGrapher, we noticed that a significant amount of images have the superatom nitrogen dioxide (NO_2). As ChemGrapher does not support superatoms, all images with nitrogen dioxide were incorrectly predicted.

As a next experiment, we manually labeled 20 images with superatom nitrogen dioxide of the Maybridge data set pixelwise, up-sampled (100x copies) them to 2000 images, and added them to the training data set of the segmentation network. The superatom nitrogen dioxide is labeled in a way that it represents just another type of regular atom. For the classification networks, we label another set of 20 images containing NO_2, up-sample them (20× copies) to 400 images, and add them to the original RDKit-based classification training data set. We retrain all the networks of ChemGrapher and now obtain correct SMILES prediction for 4747 out of 5700 images (we removed the 40 labeled training images from the total data set of 5740 images), which translates into an accuracy of 83.28%. Therefore, ChemGrapher was able, with a limited amount (40) of labeled images, to increase the graph accuracy significantly. The results are summarized in Figure 7. Note that labeling the images with nitrogen dioxide not only improved the performance of the nitrogen dioxide images (from 0 images being correct to 130 out of 263 images being correct) but also the accuracy for the rest of the images. This is because the labeled images also contained many other atoms and bonds. Access to this information

https://dx.doi.org/10.1021/acs.jcim.0c00459
J. Chem. Inf. Model. 2020, 60, 4506−4517

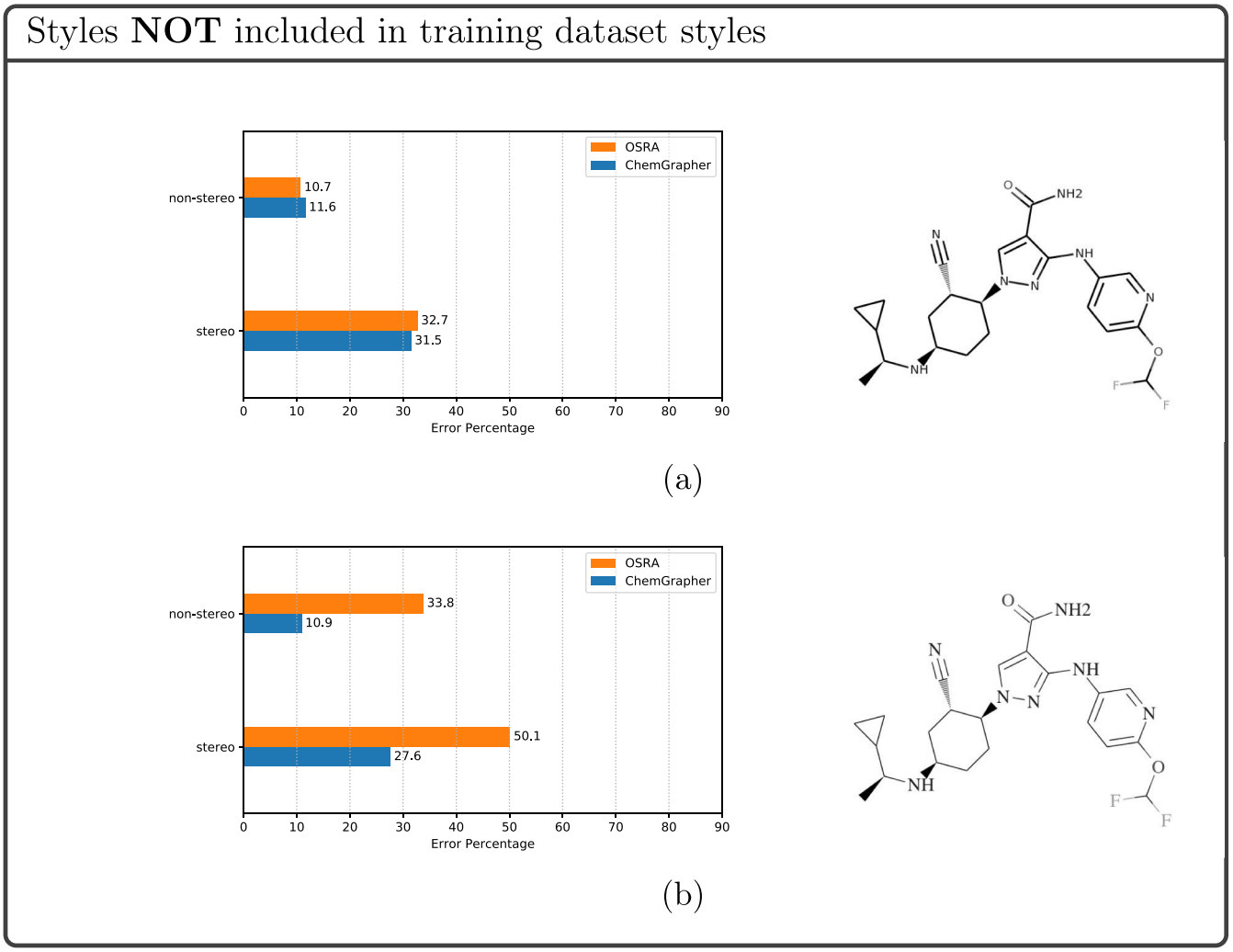


Figure 5. Graph accuracy of our tool compared with OSRA v2.1.0 measured on images (size: 1000 × 1000, resolution: 72 dpi) generated in different styles using RDKit (styles not included in training dataset styles). For each style, two experiments are performed: one with images without stereochemical information and one with images with stereochemical information. On the left, we observe for each style the results on the error rate. On the right, we observe for each style an example image in that specific style. (a) Error rate for style 3. (b) Error rate for style 4.

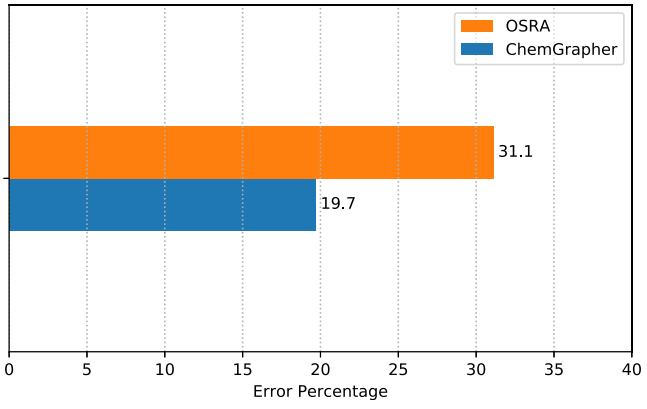


Figure 6. Error rate of our tool ChemGrapher on the test set of journal article images compared with OSRA. From the errors, we learn that there is still room for improvement in future work.

improved the networks to quickly adapt to the specific style (font, line width, *etc.*) of the Maybridge data set.

LIMITATIONS AND FUTURE WORK

ChemGrapher currently handles superatoms as regular atoms. This means that every superatom needs to be explicitly labeled. We leave it as future work to support superatoms with a more efficient approach. The same holds for charges where every charge type (−1, +1, ...) is explicitly labeled as a different charge. ChemGrapher now supports only charges from −1 to +1. In future work, this could also be solved in a more efficient way. Another limitation is that ChemGrapher expects a preprocessed image as input, meaning that chemical compounds should be cropped out of a scanned text page, for example. This image preprocessing module could be built into ChemGrapher as future work to make it a stand-alone tool. Finally, to train the segmentation network, we need a pixelwise-labeled data set. However, this kind of data set is not always available. In this work, we created this data set with RDkit. However, the consequence is that the format of the input image is somewhat biased. We have seen in the case study that ChemGrapher performs reasonably well, although not equally on real images. To handle other kinds of image styles, it might be difficult to find a pixelwise-labeled data set to retrain our networks. Therefore, future work could focus on building a method that can learn

https://dx.doi.org/10.1021/acs.jcim.0c00459
J. Chem. Inf. Model. 2020, 60, 4506−4517

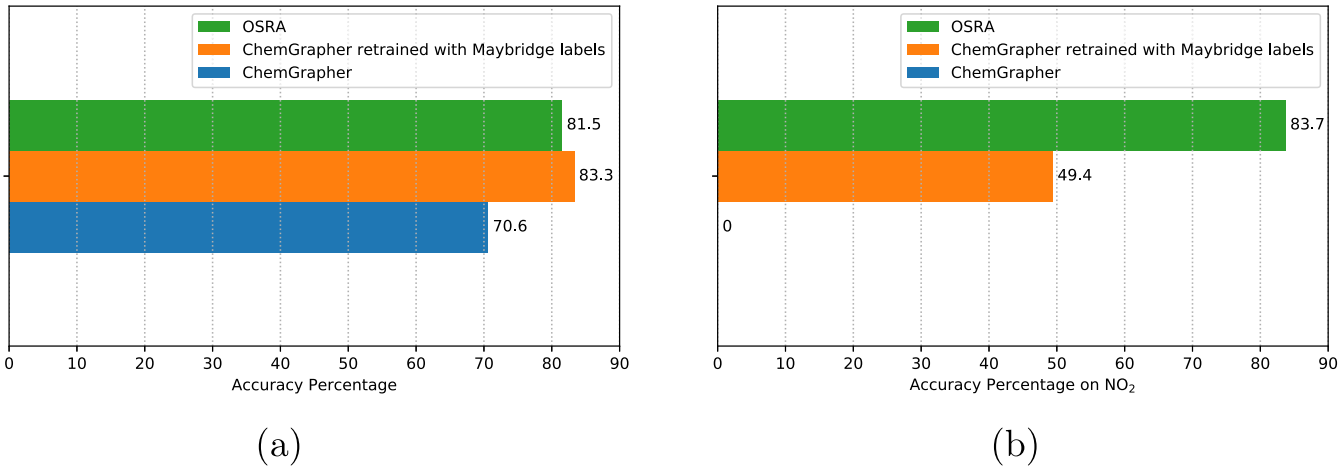


Figure 7. On the left, we have the performance of ChemGrapher on the Maybridge data set without and with retraining (with 40 labeled images from the Maybridge data set) compared to OSRA v2.1.0. On the right, we see the performance of ChemGrapher on NO_2 images ($N = 263$) in the Maybridge data set with and without retraining. For comparison, the performance of OSRA v2.1.0 on the same number of NO_2 images is also shown. (a) Performance on the Maybridge dataset. (b) Accuracy on NO_2 images.

from data that are not labeled pixelwise. The data would only offer a way to verify if the resulting graph is correct or not. A potential candidate for achieving this is to use machine-learning methods from domain adaptation.

CONCLUSIONS

We presented a method to recognize the graph structure of molecules from 2D images of chemical structures using deep learning. This method learns a model directly from data. We have seen that careful data preparation is crucial. Care should be taken to have a balanced data set for the different classes of atoms and bonds. However, even with an imperfectly balanced data set, our deep-learning methods give superior results. To make our method work, we need the classification networks to have an almost perfect accuracy. While the segmentation network can tolerate some errors, for the classification networks, every drop in accuracy can have dramatic results on the overall graph recognition accuracy. We showed empirically that the performance of ChemGrapher is better than that of the well-known tool OSRA[1] and that it also provides detailed information about the layout of the resulting graph to the user. For our deep-learning method to learn accurately, we need pixelwise labels of 2D images of chemical structures. In fact, this pixelwise-labeling of images for the segmentation is actually key to linking the atoms and bonds in the resulting graph back to the source image. This makes our deep-learning approach also interpretable. In the context of drug discovery, such tools are important for information retrieval from patents and journals.

ASSOCIATED CONTENT

Supporting Information

The Supporting Information is available free of charge at https://pubs.acs.org/doi/10.1021/acs.jcim.0c00459.

Eight labeled (SMILES) data sets with 1000 images each generated in four different styles, with each style, containing a set with images without stereochemical information encoded and a set with images encoded with stereochemical information (ZIP)

Data set with 2D images of chemical compounds cut out of journal articles, with the source article provided for every image (ZIP)

AUTHOR INFORMATION

Corresponding Authors

Martijn Oldenhof − *Department of Electrical Engineering (ESAT), STADIUS Center for Dynamical Systems, Signal Processing and Data Analytics, KU Leuven, Leuven 3001, Belgium*; orcid.org/0000-0003-4916-3014; Email: martijn.oldenhof@esat.kuleuven.be

Adam Arany − *Department of Electrical Engineering (ESAT), STADIUS Center for Dynamical Systems, Signal Processing and Data Analytics, KU Leuven, Leuven 3001, Belgium*; Email: adam.arany@esat.kuleuven.be

Yves Moreau − *Department of Electrical Engineering (ESAT), STADIUS Center for Dynamical Systems, Signal Processing and Data Analytics, KU Leuven, Leuven 3001, Belgium*; Email: yves.moreau@esat.kuleuven.be

Jaak Simm − *Department of Electrical Engineering (ESAT), STADIUS Center for Dynamical Systems, Signal Processing and Data Analytics, KU Leuven, Leuven 3001, Belgium*; Email: jaak.simm@esat.kuleuven.be

Complete contact information is available at: https://pubs.acs.org/10.1021/acs.jcim.0c00459

Notes

The authors declare no competing financial interest.

ACKNOWLEDGMENTS

M.O., A.A., Y.M., and J.S. are funded by (1) Research Council KU Leuven: C14/18/092 SymBioSys3; CELSA-HIDUCTION, (2) Innovative Medicines Initiative: MELLODDY, (3) Flemish Government (ELIXIR Belgium, IWT: PhD grants, FWO 06260), and (4) Impulsfonds AI: VR 2019 2203 DOC.0318/1QUATER Kenniscentrum Data en Maatschappij. Computational resources and services used in this work were provided by the VSC (Flemish Supercomputer Center), funded by the Research Foundation—Flanders (FWO) and the Flemish Government—Department EWI. We also gratefully acknowledge the support of NVIDIA Corporation with the donation of the Titan Xp GPU used for this research.

https://dx.doi.org/10.1021/acs.jcim.0c00459
J. Chem. Inf. Model. 2020, 60, 4506−4517

REFERENCES

(1) Filippov, I. V.; Nicklaus, M. C. Optical Structure Recognition Software To Recover Chemical Information: OSRA, An Open Source Solution. *J. Chem. Inf. Model.* **2009**, *49*, 740−743.

(2) Weininger, D. SMILES, a Chemical Language and Information System. 1. Introduction to Methodology and Encoding Rules. *J. Chem. Inf. Comput. Sci.* **1988**, *28*, 31−36.

(3) Dalby, A.; Nourse, J. G.; Hounshell, W. D.; Gushurst, A. K. I.; Grier, D. L.; Leland, B. A.; Laufer, J. Description of Several Chemical Structure File Formats Used by Computer Programs Developed at Molecular Design Limited. *J. Chem. Inf. Comput. Sci.* **1992**, *32*, 244−255.

(4) Park, J.; Rosania, G.; Shedden, K. A.; Nguyen, M.; Lyu, N.; Saitou, K. Automated Extraction of Chemical Structure Information from Digital Raster Images. *Chem. Cent. J.* **2009**, *3*, 4.

(5) McDaniel, J. R.; Balmuth, J. R. Kekule: OCR-Optical Chemical (Structure) Recognition. *J. Chem. Inf. Comput. Sci.* **1992**, *32*, 373−378.

(6) Valko, A. T.; Johnson, A. P. CLiDE Pro: The Latest Generation of CLiDE, a Tool for Optical Chemical Structure Recognition. *J. Chem. Inf. Model.* **2009**, *49*, 780−787.

(7) Sadawi, M. N.; Sexton, A. P.; Sorge, V. Chemical Structure Recognition: A Rule Based Approach. Proceedings of SPIE. *Document Recognition and Retrieval XIX*, 2012; Vol. *8297*.

(8) Staker, J.; Marshall, K.; Abel, R.; McQuaw, C. M. Molecular Structure Extraction from Documents Using Deep Learning. *J. Chem. Inf. Model.* **2019**, *59*, 1017−1029.

(9) Soderberg, T. Chirality and stereoisomers. https://chem.libretexts.org/Bookshelves/Organic_Chemistry/Book%3A_Organic_Chemistry_with_a_Biological_Emphasis_(Soderberg)/Chapter_03%3A_Conformations_and_Stereochemistry/03.3%3A_Stereoisomerism_%E2%80%93_chirality%2C_stereocenters%2C_enantiomers (accessed on 03.06.2019).

(10) LeCun, Y.; Bengio, Y.; Hinton, G. Deep learning. *Nature* **2015**, *521*, 436−444.

(11) LeCun, Y.; Bengio, Y. Convolutional Networks for Images, Speech, and Time Series. *The Handbook of Brain Theory and Neural Networks*, 1995; Vol. *3361*, p 1995.

(12) Ronneberger, O.; Fischer, O.; Brox, T. U-Net: Convolutional Networks for Biomedical Image Segmentation. *Medical Image Computing and Computer-Assisted Intervention (MICCAI)*; MICCAI, 2015; pp 234−241.

(13) He, K.; Gkioxari, G.; Dollár, P.; Girshick, R. Mask R-CNN. **2017**, arXiv.org e-Print archive, arXiv:1703.06870 [cs.CV].

(14) Jansson, A.; Humphrey, E. J.; Montecchio, N.; Bittner, R. M.; Kumar, A.; Weyde, T. Singing Voice Separation with Deep U-Net Convolutional Networks. *ISMIR*, 2017.

(15) Çiçek, Ö.; Abdulkadir, A.; Lienkamp, S. S.; Brox, T.; Ronneberger, O. 3D U-Net: Learning Dense Volumetric Segmentation from Sparse Annotation. **2016**, arXiv.org e-Print archive, arXiv:1606.06650 [cs.CV].

(16) Yu, F.; Koltun, V. Multi-Scale Context Aggregation by Dilated Convolutions. **2016**, arXiv.org e-Print archive, arXiv:1511.07122 [cs.CV].

(17) Bengio, Y.; Simard, P.; Frasconi, P. Learning Long-term Dependencies with Gradient Descent is Difficult. *IEEE Trans. Neural Network.* **1994**, *5*, 157−166.

(18) He, K.; Zhang, X.; Ren, S.; Sun, J. Deep Residual Learning for Image Recognition. *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2016; pp 770−778.

(19) Zhang, Z.; Liu, Q.; Wang, Y. Road Extraction by Deep Residual U-Net. **2017**, arXiv.org e-Print archive, arXiv:1711.10684 [cs.CV].

(20) Ioffe, S.; Szegedy, C. Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift. **2015**, arXiv.org e-Print archive 2015, arXiv:1502.03167 [cs.LG].

(21) Clevert, D.-A.; Unterthiner, T.; Hochreiter, S. Fast and Accurate Deep Network Learning by Exponential Linear Units (ELUs). **2015**, arXiv.org e-Print archive, arXiv:1511.07289 [cs.LG].

(22) Chen, H.; Engkvist, O.; Wang, Y.; Olivecrona, M.; Blaschke, T. The Rise of Deep Learning in Drug Discovery. *Drug Discovery Today* **2018**, *23*, 1241.

(23) Xu, Z.; Wang, S.; Zhu, F.; Huang, J. Seq2Seq Fingerprint: An Unsupervised Deep Molecular Embedding for Drug Discovery. *Proceedings of the 8th ACM International Conference on Bioinformatics, Computational Biology, and Health Informatics*, New York, NY, USA, 2017; pp 285−294.

(24) Winter, R.; Montanari, F.; Noé, F.; Clevert, D.-A. Learning Continuous and Data-driven Molecular Descriptors by Translating Equivalent Chemical Representations. *Chem. Sci.* **2019**, *10*, 1692−1701.

(25) Gómez-Bombarelli, R.; Wei, J. N.; Duvenaud, D.; Hernández-Lobato, J. M.; Sánchez-Lengeling, B.; Sheberla, D.; Aguilera-Iparraguirre, J.; Hirzel, T. D.; Adams, R. P.; Aspuru-Guzik, A. Automatic Chemical Design Using a Data-Driven Continuous Representation of Molecules. *ACS Cent. Sci.* **2018**, *4*, 268−276.

(26) Duvenaud, D. K.; Maclaurin, D.; Iparraguirre, J.; Bombarell, R.; Hirzel, T.; Aspuru-Guzik, A.; Adams, R. P. Convolutional Networks on Graphs for Learning Molecular Fingerprints. *Advances in Neural Information Processing Systems*, 2015; pp 2224−2232.

(27) Kearnes, S.; McCloskey, K.; Berndl, M.; Pande, V.; Riley, P. Molecular Graph Convolutions: Moving Beyond Fingerprints. *J. Comput.-Aided Mol. Des.* **2016**, *30*, 595−608.

(28) Coley, C. W.; Barzilay, R.; Green, W. H.; Jaakkola, T. S.; Jensen, K. F. Convolutional Embedding of Attributed Molecular Graphs for Physical Property Prediction. *J. Chem. Inf. Model.* **2017**, *57*, 1757−1772.

(29) Simm, J.; Arany, A.; De Brouwer, E.; Moreau, Y. Graph Informer Networks for Molecules. **2019**, arXiv.org e-Print archive, arXiv:1907.11318 [stat.ML].

(30) Pires, D. E. V.; Blundell, T. L.; Ascher, D. B. pkCSM: Predicting Small-Molecule Pharmacokinetic and Toxicity Properties Using Graph-Based Signatures. *J. Med. Chem.* **2015**, *58*, 4066.

(31) Goh, G. B.; Siegel, C.; Vishnu, A.; Hodas, N. O.; Baker, N. Chemception: A Deep Neural Network with Minimal Chemistry Knowledge Matches the Performance of Expert-developed QSAR/QSPR Models. **2017**, arXiv.org e-Print archive, arXiv:1706.06689 [stat.ML].

(32) Shelhamer, E.; Long, J.; Darrell, T. Fully Convolutional Networks for Semantic Segmentation. *IEEE Trans. Pattern Anal. Mach. Intell.* **2017**, *39*, 640−651.

(33) Chollet, F. Xception: Deep Learning With Depthwise Separable Convolutions. *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2017.

(34) Gaulton, A.; Hersey, A.; Nowotka, M.; Bento, A. P.; Chambers, J.; Mendez, D.; Mutowo, P.; Atkinson, F.; Bellis, L. J.; Cibrián-Uhalte, E.; Davies, M.; Dedman, N.; Karlsson, A.; Magariños, M. P.; Overington, J. P.; Papadatos, G.; Smit, I.; Leach, A. R. The ChEMBL database in 2017. *Nucleic Acids Res.* **2017**, *45*, D945−D954.

(35) Landrum, G. RDKit: Open-source cheminformatics. http://www.rdkit.org (accessed on 03.06.2019).

(36) Fork of the official sources for the RDKit library. https://github.com/biolearning-stadius/rdkit (accessed on 17.07.2020).

(37) Paszke, A.; Gross, S.; Chintala, S.; Chanan, G.; Yang, E.; DeVito, Z.; Lin, Z.; Desmaison, A.; Antiga, L.; Lerer, A. *Automatic Differentiation in PyTorch*, NIPS-W, 2017.

(38) F1 score. 2020. https://en.wikipedia.org/w/index.php?title=F1_score&oldid=935689499, Page Version ID: 935689499 (accessed on 20.01.2020).

(39) Maybridge dataset. https://www.cs.bham.ac.uk/research/groupings/reasoning/sdag/mol-dataset.php (accessed on 10.06.2020).

https://dx.doi.org/10.1021/acs.jcim.0c00459
J. Chem. Inf. Model. 2020, 60, 4506−4517