Biological activity prediction

Anastasia Sarycheva [1] **Daria Chaplygina** [1] **Alexey Voskoboinikov** [1] **Roman Bychkov** [1] **Sayan Protasov** [1]

Abstract

This project is dedicated to biological activity prediction using deep learning (DL) architectures. We use the database of antiviral activity molecular data (Nikitina et al., 2019) to create datasets. We employ the variety of ways to represent the molecules: feature matrix (Hirohara et al., 2018), molecular descriptors (Moriwaki et al., 2018), molecular fingerprints (Capecchi et al., 2020), molecular graphs (Duvenaud et al., 2015). We explore and compare the corresponding DL architectures: Attention-based Convolutional Neural Natwork (CNN) (Pham & Le, 2019), gradient boosting on decision trees + Multi-Layer Perceptron, transformer-based architectures (Karpov, 2020; Shion Honda, 2019) and graph convolution network (Kipf & Welling, 2016) (GCN) with two different convolution operators. As a result, we compared all models for binary and multi-class classification of biological activity, where the graph convolution network outperforms other deep learning methods for our datasets.

Github repo: https://github.com/antediem/DLbioactivity
Video presentation: Link

Introduction

Various compounds, when interacting with biological systems, can solve problems, improving quality of life, they can cure, or harm. For the needs of the pharmaceutical and food industries, there is a growing need for more sustainable compounds, with improved biological activities (Correia et al., 2019). Biological activity is defined as the capacity of a molecule to achieve a defined biological effect on a target (Jackson et al., 2007).

Biological activity prediction of compounds can be approached using deep learning (DL) architectures, e.g. the architecture proposed in (Pham & Le, 2019), which we used as a baseline.

We consider the relevant problem of antiviral activity prediction. For this, we use the database presented in (Nikitina et al., 2019). It contains molecules capable of harming the target pathogen organisms, i.e. viruses. The database will be described in details in 1.2.

The paper will be structured as follows: 1 discusses the aspects of data representation for molecules, describes the database and data parsing; 3 describes briefly the baseline architecture and changes to it for own datasets; 4 describes the optimization pipeline; 5 describes the employment of fingerprints along with gradient boosting and DNN; 6 describes the approaches based on sequence processing: transformers and 1D convolutions; 7 describes the principles of how graph neural network works and its application in the framework of working with molecules presented as graphs; 8 compares the approaches.

1. Data

1.1. SMILES

SMILES (Simplified Molecular-Input Line-Entry System) (Weininger, 1988) was introduced to represent a molecular structure of a chemical compound with a single line notation. The SMILES notation consists of letters, numbers and characters that specify the atoms, their connectivity, bond order, and chirality (illustration). SMILES specification rules can be summarized as follows: 1) atoms are represented by atomic symbols; 2) single, double, and triple bonds are represented by **-**, **=**, **#**; 3) aromatic atoms are represented in lower case; 4) aliphatic atoms are represented in upper case; 5) branches are enclosed in parentheses; 6) there are special symbols for stereo-chemistry configuration.

1.2. Database structure

We worked with the database containing the data on antiviral activity for more than 260k compounds (in SMILES notation) tested against 195 viral species. (Nikitina et al., 2019) The dataset consisted of assays retrieved from ChEMBL. Each molecule studied in the assay had the binary activity flag, assigned by the authors of the database via automatized information retrieval from the assay text description.

[1]Skolkovo Institute of Science and Technology, Moscow, Russia. Correspondence to: Anastasia Sarycheva <sarycheva.anastasia@gmail.com>.

Final project, Deep learning course, 2021.

The number of assays and species per molecule was not restricted.

1.3. Dataset creation

We processed the data related to the 3 largest viral species in the dataset, namely human immunodeficiency virus-1 (HIV-1), hepatitis C virus (HCV), Influenza A virus. The models were tested on 3 binary datasets and 1 multiclass dataset containing all three species (Fig. 1). The pipeline developed for the data processing could be further applied to other subsets from the considered database.

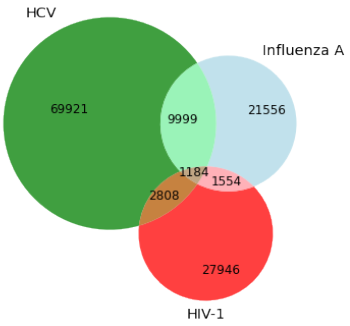

Figure 1. The number of active molecules for each of the considered species.

We first removed all the assays containing both positive and negative activity flags for one molecule to process the data. The percentage of such controversial assays varied from 0.02% to 0.07% for species in our dataset. Next, we selected all the molecules with more than 50% of the assays confirming their activity against the virus. Finally, we studied the molecule length distribution and filtered out all the large molecules, leaving SMILES length of ≥ 250.

1.4. Representation of molecular structures

In the paper (Pham & Le, 2019), which we used as a baseline, SMILES were represented in the form of feature matrix (Hirohara et al., 2018). Each symbol in the isomeric SMILES string was encoded via a 42 dimensional vector (see detailed table). The first 21 features are responsible for encoding an atom in the following manner: 1) atom type (H,C,O,N,other), e.g. H is represented as [1,0,0,0,0]; 2) total number of H atoms attached to the considered atom; 3) atom's degree of unsaturation; 4) atom's formal charge; 5) atom's total valence; 6) if this atom is included in a ring; 7) if this atom is included in an aromatic structure; 8) chirality (R, S, other); 9) hybridization ($s, sp, sp^2, sp^3, sp^3d, sp^3d^2, other$). The rest 21 features are encoding the SMILES symbol which is not an atom. Let us illustrate the feature matrix extraction from canonical SMILES taken from our dataset (Fig. 2).

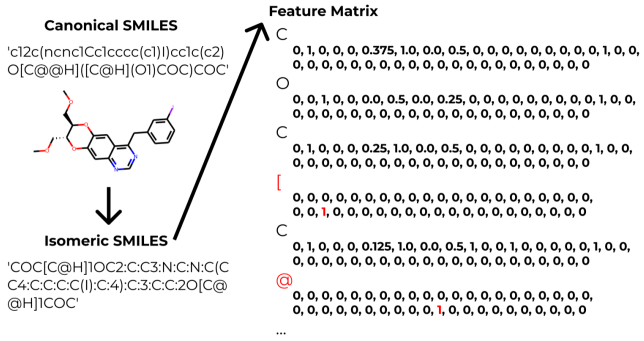

Figure 2. Feature matrix illustration.

In (Pham & Le, 2019), along with feature matrix, the molecular descriptors were used. Molecular descriptors are the numerical representation of chemical information encoded within SMILES. We also calculated them via Mordred (Moriwaki et al., 2018) for our datasets.

Alternatively, molecules can be represented via molecular fingerprints (Capecchi et al., 2020). We employed rdkit library in Python to calculate Morgan (circular) fingerprints (Rogers & Hahn, 2010). To summarize, Morgan algorithm goes through each atom of the molecule and it extracts all possible paths from this atom with a given radius. Each unique path is mapped to a number based on the provided number of bits. We used radius 2 and 1024 bits. While such high-dimensional representation contains maximum information, it is a subject of feature selection, so we employed it for CatBoost+DNN experiments.

The third approach in our work was the use of graph convolutional networks to predict the activity of molecules. A molecular graph is a connected undirected graph that is in one-to-one correspondence with the structural formula of a chemical compound in such a way that the atoms of the molecule correspond to the vertices of the graph, and the chemical bonds between these atoms correspond to the edges of the graph. The concept of "molecular graph" is basic for computational chemistry and chemoinformatics. In order to obtain a molecular graph, we used the following pipeline. To begin with, using the aforementioned rdkit library, we got all the information about the molecule from the SMILES representation. Then, using this information, we obtained a graph using the networkX library. The resulting graph can be easily used for training in the future, since the pytorch geometric framework has a built-in function that converts the graph to tensor type.

2. Metrics

We used multiple metrics to evaluate the performance of our models. The summary of the metrics is in the Table 1.

Table 1. Perfomance Metrics

Metric	Formula
Sensitivity	$\frac{TP}{TP+FN}$
Specificity	$\frac{TN}{TN+FP}$
Accuracy	$\frac{TP+TN}{TP+TN+FP+FN}$
MCC	$\frac{TP \times TN - FP \times FN}{\sqrt{(TP+FP)(TP+FN)(TN+FP)(TN+FN)}}$
AUC	area under ROC curve

AUC is a usual choice of criteria for the unbalanced dataset. For multiclass testing, we used one-vs-one variant with macro averaging. Matthews correlation coefficient (MCC) is a valuable criterion for our problem, as it considers the balance ratios of the four confusion matrix categories and depends on the classifier's threshold. Other metrics provide additional information for the benchmark.

3. Baseline

To run the baseline architecture described in (Pham & Le, 2019) on the provided by the authors data, we added a couple of fixes so it could be executed in Colab with the newest versions of libraries. The metrics on the dataset provided by the authors were in perfect agreement with the paper (Pham & Le, 2019).

For own datasets, we had to modify the baseline code further. Our datasets are much larger than the data in (Pham & Le, 2019), so we had to replace JSON format to fit in 25Gb RAM provided by Colab Pro. Along with data loading changes, we changed the hard-coded channels in layers. The former depend on the dimension of the feature matrix, which in its turn depends on the largest isomeric SMILES string in the dataset.

4. Hyperparameter tuning

Hyperparameter tuning can have a significant effect on the performance of the model. When we reproduced results from the paper (Pham & Le, 2019), results showed us that there is about 0.15 difference between train and validation metric values, which is a clear indication of poor potential generalization capability of neural network. It seems reasonable to try different standard techniques to tackle the overfitting problem, the most straightforward of them - dropout and regularization. However, there is no clear indication, what hyperparameter values should have for this particular task. Therefore, we should perform a search to find the best combination of hyperparameters.

The industry-standard tool for hyperparameter tuning is Ray Tune. It is a Python library, which supports PyTorch (our primary machine learning framework), and it offers state-of-the-art search algorithms and schedulers. After getting familiar with the framework, we choose to try two schedulers - ASHA (Li et al., 2020), which is usually used on smaller problems and has early terminations of the bad trials, and BOHB (Falkner et al., 2018), applicable for larger problems with a small number of hyperparameters.

In the paper (Pham & Le, 2019) authors wrote that they did perform grid search using cross-validation. So, our first attempt to use Ray was to perform a similar procedure but to use random parameter selection and apply it to the modified version of the baseline model. We restricted ourselves to the three parameters: learning rate, dropout, and L2 regularization, as these parameters most likely would affect the generalization ability of the neural network. We ran an ASHA scheduler for 64 cross-validation models with randomly selected parameters and got controversial results. There were no clear trends on how hyperparameters affected validation metrics, but the value of the metrics for some models was unrealistically high 7. The two most probable explanations of this result are either some bug in the code regarding implementation of procedure (target leakage), or that the dataset for this problem was relatively small, and with larger modified models, some of them stochastically got that result. Nonetheless, working with the dataset from the paper (Pham & Le, 2019) was not our primary focus, and, given that this cross-validation procedure took a long time, we proceed to explore models on new datasets.

Table 2. Range of hyperparameter values

Parameter	low	high	scale
Learning rate	5e-6	1e-4	logarithmic
L2 regularisation	5e-6	1	logarithmic
Dropout	0	0.7	linear

The next step was to explore hyperparameters of the base model on the dataset "HIV1 vs. HCV". We choose to explore the base model because it is relatively small, so the tuning process is supposed to be faster than for other architectures, while the same code with minor modifications could be used to tune other models. Same as before, we chose three hyperparameters: learning rate, dropout, and L2 regularization. As a target for the schedulers, first, we chose validation loss. However, after exploring the result on 12 models by ASHA scheduler 8 we suggested that some other performance metric may be preferable. We choose validation MCC, as it was suggested in the paper (Pham & Le, 2019) (MCC considers all classes in the confusion matrix, and it would be helpful if we would like to scale the problem up to the multiclass classification). The results for ASHA and BOHB scedulars (12 models each, BOHB was used only in half of the data due to memory restrictions) with target validation MCC are presented on the Figures 9

and 10, and range of values of hyperparameter are presented on the Table 2.

There are a couple of interesting features one can see from the tables. First, the highest performing models were trained longer and had higher validation loss. This means that early stopping based on the loss (as suggested in the (Pham & Le, 2019)) may harm models, and choosing other target metrics for early stopping or scheduler may be a better idea. Second, dropout and L2 regularisation does not affect much the problem of bad generalization (validation metrics 10 and train metrics 11). They might have some small effect on the model performance, but there are no clear trends other than a decrease in performance with high values of L2 regularisation.

As for tuning Graph neural network, it required more computational power. Ray tune offers ways to parallels computation of multiple models even on a single GPU, but then the memory becomes the limiting factor, resulting in multiple crushes. Thankfully, checkpoints allow to restore the Table 11 and 13. Result suggests that the graph model may favor longer training with a lower learning rate, and regularization and dropout do not improve (maybe even deteriorate) generalization ability.

5. CatBoost and DNN

Gradient boosting (GB) is a machine learning technique. To put it simply, GB model is made of decision trees. The accuracy is "boosted" via addition of trees: if the first tree is trained to output the model target, the second is trained to compensate for the error of the first one, and so on. Then the output of the model is the sum of tree outputs. The CatBoost algorithm was introduced by Yandex in 2017 (Prokhorenkova et al., 2017), and it was shown that CatBoost performs better than other GB methods.

We used CatBoost library for Python to set up two kinds of experiments: 1) we employed CatBoost on its own for the prediction of antiviral compounds classes; 2) we employed CatBoost to extract important features to be further used as DNN input. The reason behind the latter, is that for these experiments we employed Morgan fingerprints of 1024 bits, and while they are informative in terms of overall chemical structure, some bits hold information which is relevant to the antiviral activity: as it was mentioned in 1.4, Morgan algorithm encodes the atoms and their nearest surroundings, which is equivalent to sites of molecule. These sites can be either important in terms of antiviral activity differentiation, or not.

As a DNN architecture, we employed a Multi-Layer Perceptron, its hyper-parameters are detailed in Table 3. The activation function is ReLU, sigmoid function was used for final prediction. We also applied dropout technique (p=0.2 after 1st layer layer, p=0.5 after 2nd layer). Adam optimizer with learning rate of 0.0001 was used.

Table 3. DNN

Layer	Neurons
1st Linear layer	128
2nd Linear layer	64
3rd Linear layer	32

When using CatBoost for feature selection, we employed built-in function for feature importance. Then, for sorted features, we employed threshold (thr) of 0.00, 0.01, and 0.05. The number of the features left is specified in Features row in Tables 4, 5, 6. The results the experiments with our datasets are presented in Tables 4, 5, 6.

According to the results of the experiments, for certain metrics, e.g. Sensitivity and AUC, pre-DNN feature extraction via CatBoost can be considered beneficial. However, the most interesting result is that when we compare the lists of important features (i.e. bits in Morgan fingerprint) between datasets, we see some common values: these sites can be potentially relevant to the specific antiviral classes. We illustrated overall feature importance via Shap values in Figure 3. For example, 314th bit seems to be important for HIV1 antiviral activity, 333rd – for HCV, and 90th – for FLUA. These results, however, should be checked via bit-wise sites visualization and analysis.

6. Transformers

Since SMILES is a row representation of a molecule, it can be naturally processed by NLP techniques. First of all, we made a simple model based on 1D convolutions as a replacement for the initial model which processes the SMILES feature matrix as an image and doesn't take into account connections between all features of neighboring atoms. As we expected, the sequence-based model worked better (at least on the original paper dataset). Since this model shows comparable results, it was used as a convolutional head for the transformer fingerprint described below. Usage of the same convolution operations both on transformer encoded sequence and feature matrix is necessary to compare these approaches.

As a development of the sequence processing idea, transformer application was considered. The idea of using transformers for SMILES processing was presented in (Shion Honda, 2019). Usually, transformers require a lot of data to be trained and molecule datasets are not always large. That is why for cheminformatics problems SMILES transformers are trained not on properties prediction task, but on SMILES canonization problem. In our experiments we worked out both options.

Table 4. Performance comparison for HIV1 vs. HCV dataset: train/test

Metrics	CatBoost	DNN	CatBoost(thr=0.00)+DNN	CatBoost(thr=0.01)+DNN	CatBoost(thr=0.05)+DNN
Accuracy	0.903/0.892	0.982/0.885	0.977/0.885	0.968/0.885	0.931/0.866
AUC	0.830/0.811	0.998/0.915	0.998/0.914	0.996/0.912	0.981/0.912
MCC	0.749/0.714	0.955/0.703	0.943/0.701	0.922/0.702	0.834/0.664
Sensitivity	0.670/0.638	0.986/0.769	0.982/0.765	0.976/0.767	0.932/0.782
Specificity	0.990/0.985	0.980/0.927	0.975/0.928	0.965/0.928	0.931/0.896
Features	1024	1024	906	757	389

Table 5. Performance comparison for FLUA vs. HIV1 dataset: train/test

Metrics	CatBoost	DNN	CatBoost(thr=0.00)+DNN	CatBoost(thr=0.01)+DNN	CatBoost(thr=0.05)+DNN
Accuracy	0.841/0.804	0.961/0.784	0.959/0.801	0.939/0.776	0.860/0.757
AUC	0.840/0.803	0.996/0.882	0.996/0.882	0.991/0.877	0.965/0.873
MCC	0.698/0.621	0.923/0.571	0.919/0.603	0.880/0.556	0.731/0.521
Sensitivity	0.729/0.695	0.985/0.818	0.986/0.787	0.975/0.822	0.946/0.834
Specificity	0.950/0.910	0.937/0.752	0.932/0.814	0.904/0.732	0.776/0.682
Features	1024	1024	889	757	393

Table 6. Performance comparison for HCV vs. FLUA dataset: train/test

Metrics	CatBoost	DNN	CatBoost(thr=0.00)+DNN	CatBoost(thr=0.01)+DNN	CatBoost(thr=0.05)+DNN
Accuracy	0.809/0.791	0.939/0.775	0.935/0.766	0.926/0.772	0.861/0.754
AUC	0.622/0.591	0.988/0.799	0.986/0.798	0.983/0.792	0.948/0.795
MCC	0.394/0.307	0.848/0.403	0.837/0.403	0.818/0.389	0.681/0.394
Sensitivity	0.262/0.208	0.960/0.580	0.954/0.616	0.949/0.560	0.906/0.637
Specificity	0.983/0.975	0.932/0.837	0.928/0.813	0.919/0.839	0.846/0.790
Features	1024	1024	929	818	438

The first approach is fine-tuning after canonization training. There is only one canonical SMILES representation for each molecule and the canonization process transforms any other line related to a particular molecule to its canonical SMILES line. We searched through the papers and git repositories in order to obtain implementation of that technique and found one that provides the PyTorch model with pretrained dictionary. However, it was challenging to launch this model since the vocabulary - object that transforms SMILES symbol to integer number for encoding - was not provided. Following the name of the original paper dataset CHEMBL24, we formed the SMILES-containing file suitable for the vocabulary generation script. After that, the pretrained model was able to launch, however it showed extremely poor performance in comparison with our previous convolutional networks. The main hypothesis for such behavior was the wrong vocabulary since the order of symbols in it is based on their frequencies and, as we later revealed in the paper, authors used not the whole dataset, but the random half of it. Lack of original vocabulary file made it impossible to check this directly and we tried to train the SMILES canonization transformer from scratch. The model converged suspiciously fast. Analyzing the provided code, we revealed that files responsible for the transformer training process do not make canonization, instead the identity transformation was conducted. That is an extremely serious mistake made by the authors and we email them to get the consultation. However, this approach was postponed since several bug fixes did not change the situation and authors didn't respond.

The other approach is the training transformer encoder from scratch. In order to process the encoded state the convolutional network similar to our 1d network was used. The utilized architecture is described in the 7 Additionally, the scheme with distinct branch positional encoding was checked. The hypothesis was that since the molecule is the spatial structure it is better to encode the position not as the number of symbol in the SMILES notation, but as the position of the molecule main body where the atom or a branch (isolated by brackets) is located as shown on fig. 6. However, this modified positional encoding scheme produced metrics 3% worse.

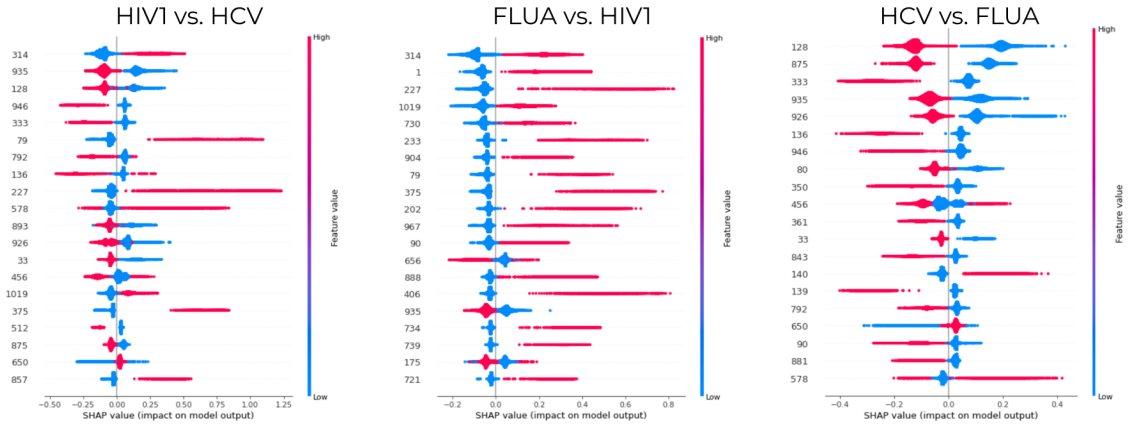

Figure 3. Overall feature importance according to SHAP.

Table 7. Transformer-based branch

Parameter	Value
Hidden dim	64
Number of Layers	4
Heads	4
Vocabulary size	45
Conv layer channels	(64,128)
Conv layer pool	3

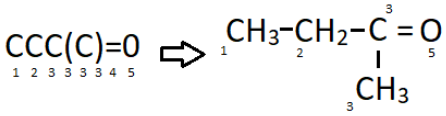

Figure 4. Positional encoding of SMILES taking into account spatial structure

7. Graph neural network

In our work, we decided to abandon classic convolutions and decided to try using graph neural network. 'Convolution' in GCNs is basically the same operation. It refers to multiplying the input neurons with a set of weights that are commonly known as filters or kernels. The filters act as a sliding window across the whole image and enable CNNs to learn features from neighboring cells. Within the same layer, the same filter will be used throughout image, this is referred to as weight sharing.

Using the representation of molecules in the form of a graph, we built a neural network based on calculating the convolution graph. In our case, we used a popular framework called pytorch geometric. Pytorch geometric is a wrapper for the classic pytorch but it provides many different graph convolutions.

In this work, we used 2 different convolution operators.

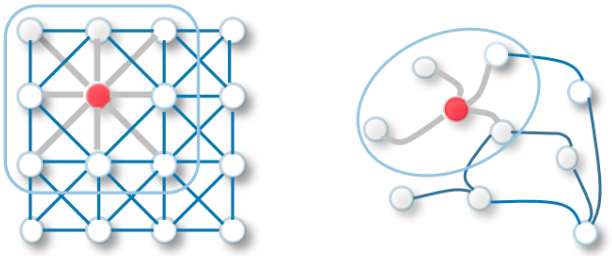

Figure 5. Illustration of 2D Convolutional Neural Networks (left) and Graph Convolutional Networks (right) (Wu et al., 2019)

The first operator that we used was GCNConv - the graph convolutional operator from the "Semi-supervised Classification with Graph Convolutional Networks" paper (Kipf & Welling, 2016). And they also used another operator called MFConv from "Convolutional Networks on Graphs for Learning Molecular Fingerprints" paper (Duvenaud et al., 2015). For both convolutions operators, we had the same architecture (fig. 6). In addition to the convolutional layers, we used linear layers for molecular descriptors, which are then concatenate into a vector and fed to the linear layer for binary classification.

8. Results and Discussion

The results for all the models tested in our study provided in the Table 8. Surprisingly, we can clearly see the dominance of the non-deep learning model, CatBoost. The reason for this could be in the Morgan fingerprints, which were used as input features. Further, these features could be tested with deep-learning models.

The second-best models are Graph-based neural networks, both with MF and GCN convolution types. These models have solid quality and constantly outperform other algo-

Table 8. Final results for all the tested model (train/validation metrics; models with the best validation score are highlighted in bold).

Metrics	Dataset	Baseline	CatBoost	GraphNN(MFConv)	GraphNN(GCNConv)	Conv1D	Transformer
Accuracy	HIV1+HCV	.91/.85	**.90/.89**	.86/.87	.84/.85	.89/.82	.77/.68
	HIV1+FluA	.76/.64	**.84/.80**	.74/.73	.69/.67	.78/.67	.68/.65
	HCV+FluA	.79/.71	**.85/.80**	.75/.73	.71/.69	.79/.77	.64/.55
	HIV1+HCV+FluA	-	**.86/.78**	-	.76/.77	-	-
AUC	HIV1+HCV	.98/.91	.83/.81	**.95/.93**	**.93/.93**	.97./.90	.90/.76
	HIV1+FluA	.95/.80	.84/.80	**.91/.88**	**.89/.88**	.95/.82	.85/.77
	HCV+FluA	.92/.78	.71/.63	**.87/.82**	.83/.81	.88/.76	.64/.55
	HIV1+HCV+FluA	-	.84/.75	-	**.87/.86**	-	-
MCC	HIV1+HCV	.79/.63	.75/.71	.69/.52	.65/.65	**.89/.82**	.55/.33
	HIV1+FluA	.57/.33	**.70/.62**	.52/.48	.44/.40	.60/.37	.45/.39
	HCV+FluA	.58/.36	.56/.37	**.49/.43**	.43/.38	.31/.18	.33/.23
	HIV1+HCV+FluA	-	**.75/.59**	-	.56/.57	-	-
Sensitivity	HIV1+HCV	.94/.80	.67/.64	**.88/.83**	**.85/.83**	.92/.82	.89/.71
	HIV1+FluA	.97/.86	.73/.70	.93/.89	**.93/.92**	.97/.86	.99/.98
	HCV+FluA	.90/.71	.45/.30	.85/.77	.82/.78	**.99/.99**	.81/.79
	HIV1+HCV+FluA	-	**.78/.66**	-	.65/.63	-	-
Specificity	HIV1+HCV	.90/.86	**.99/.98**	.86/.88	.84/.86	.89/.82	.73/.67
	HIV1+FluA	.55/.44	**.95/.91**	.56/.57	.45/.43	.59/.49	.35/.30
	HCV+FluA	.76/.71	**.98/.96**	.72/.72	.68/.66	.14/.06	.58/.48
	HIV1+HCV+FluA	-	**.90/.84**	-	.83/.83	-	-

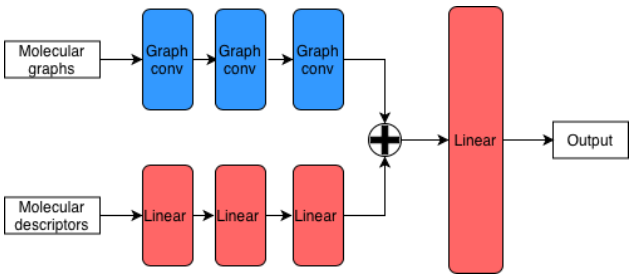

Figure 6. Simplified architecture diagram for a graph convolutional network

rithms in generalizing ability.

Sequence processing models did not show a sufficient quality. Presumably it happens because our models doesn't have enough depth to be trained (and enough data). On the contrary, convolutional networks worked with chemical fingerprints directly and could obtain substance properties from the dataset. Anyway, sequential processing models could benefit from deeper research of molecule sequence representation and more detailed neural networks design.

Another promising direction of future work is to extend the model for more species from the antiviral activity database.

9. Team members contribution

Anastasia Sarycheva. Came up with the initial idea. Initial database processing. Put baseline architecture into Colab for paper's data; adapted baseline architecture for own datasets. Run experiments using baseline for own datasets. Employed CatBoost and DNN.

Daria Chaplygina. I was responsible for data processing and datasets creation. I also contributed to creating a dataloader for the GraphNN model and tested the GraphNN model with GCNConv on 4 datasets.

Roman Bychkov. I made the initial tuning of the code. I implemented 1D-CNN idea. I made all modifications and experiments with transformers. Also I initiated team meeting and generated tasks for team members.

Alexey Voskoboinikov. I explored ray tune library for fine-tuning hyperparameters. I build cross-validation hyperparameter tuning for the base model to replicate results from the paper and tuning procedure for base and graph model on our data.

Saian Protasov. I have been working with the geometric pytorch library. I have worked with the representation of molecules in the form of graphs. GraphNN architecture development using geometric pytorch. Conducted experiments with GraphNN architecture.

10. 3rd party code

- https://pytorch.org/tutorials/beginner/hyperparameter_tuning_tutorial.html
  Parts of the tutorial were used to construct a setup for hyperparameter tuning.

- https://github.com/DSPsleeporg/smiles-transformer
  Code for SMILES-Transformer canonization task.

- https://github.com/lehgtrung/egfr-att
  Baseline architecture, helper functions for metrics calculation, feature matrix calculation (Pham & Le, 2019).

References

Capecchi, A., Probst, D., and Reymond, J.-L. One molecular fingerprint to rule them all: drugs, biomolecules, and the metabolome. *Journal of Cheminformatics*, 12(1):1–15, 2020.

Correia, J., Resende, T., Baptista, D., and Rocha, M. Artificial intelligence in biological activity prediction. In *International Conference on Practical Applications of Computational Biology & Bioinformatics*, pp. 164–172. Springer, 2019.

Duvenaud, D., Maclaurin, D., Aguilera-Iparraguirre, J., Gómez-Bombarelli, R., Hirzel, T., Aspuru-Guzik, A., and Adams, R. P. Convolutional networks on graphs for learning molecular fingerprints. *CoRR*, abs/1509.09292, 2015. URL http://arxiv.org/abs/1509.09292.

Falkner, S., Klein, A., and Hutter, F. Bohb: Robust and efficient hyperparameter optimization at scale. 2018.

Hirohara, M., Saito, Y., Koda, Y., Sato, K., and Sakakibara, Y. Convolutional neural network based on smiles representation of compounds for detecting chemical motif. *BMC bioinformatics*, 19(19):83–94, 2018.

Jackson, C. M., Esnouf, M. P., Winzor, D. J., and Duewer, D. L. Defining and measuring biological activity: applying the principles of metrology. *Accreditation and quality assurance*, 12(6):283–294, 2007.

Karpov, P., G. G. . T. I. Transformer-cnn: Swiss knife for qsar modeling and interpretation. 2020.

Kipf, T. N. and Welling, M. Semi-supervised classification with graph convolutional networks. *CoRR*, abs/1609.02907, 2016. URL http://arxiv.org/abs/1609.02907.

Li, L., Jamieson, K., Rostamizadeh, A., Gonina, E., Hardt, M., Recht, B., and Talwalkar, A. A system for massively parallel hyperparameter tuning. 2020.

Moriwaki, H., Tian, Y.-S., Kawashita, N., and Takagi, T. Mordred: a molecular descriptor calculator. *Journal of cheminformatics*, 10(1):1–14, 2018.

Nikitina, A. A., Orlov, A. A., Kozlovskaya, L. I., Palyulin, V. A., and Osolodkin, D. I. Enhanced taxonomy annotation of antiviral activity data from chembl. *Database*, 2019, 2019.

Pham, H. N. and Le, T. H. Attention-based multi-input deep learning architecture for biological activity prediction: An application in egfr inhibitors. In *2019 11th International Conference on Knowledge and Systems Engineering (KSE)*, pp. 1–9. IEEE, 2019.

Prokhorenkova, L., Gusev, G., Vorobev, A., Dorogush, A. V., and Gulin, A. Catboost: unbiased boosting with categorical features. *arXiv preprint arXiv:1706.09516*, 2017.

Rogers, D. and Hahn, M. Extended-connectivity fingerprints. *Journal of chemical information and modeling*, 50 (5):742–754, 2010.

Shion Honda, Shoi Shi, H. R. U. Smiles transformer: Pre-trained molecular fingerprint for low data drug discovery. 2019.

Weininger, D. Smiles, a chemical language and information system. 1. introduction to methodology and encoding rules. *Journal of chemical information and computer sciences*, 28(1):31–36, 1988.

Wu, Z., Pan, S., Chen, F., Long, G., Zhang, C., and Yu, P. S. A comprehensive survey on graph neural networks. *CoRR*, abs/1901.00596, 2019. URL http://arxiv.org/abs/1901.00596.

A. Hyperparameter tuning tables

	lr	l2reg	dropout	train_loss	val_loss	val_sensitivity	val_specificity	val_accuracy	val_mcc	val_auc	training_iteration
0	0.000050	0.000006	0.288802	0.002996	0.054828	0.983203	0.968186	0.970362	0.892953	0.996783	180
1	0.000050	0.004399	0.038499	0.002546	0.055542	0.974304	0.966093	0.967283	0.881321	0.996038	180
2	0.000101	0.000162	0.003459	0.000308	0.057754	0.941706	0.983841	0.977735	0.911968	0.995197	180
3	0.000013	0.000008	0.136775	0.020143	0.064886	0.989135	0.947924	0.953895	0.844677	0.997010	180
4	0.000112	0.001652	0.109969	0.003311	0.065256	0.972822	0.959981	0.961842	0.864505	0.994686	180
5	0.000099	0.000018	0.211037	0.000894	0.069278	0.974314	0.967098	0.968143	0.884195	0.995576	180
6	0.000143	0.009036	0.159677	0.005150	0.071981	0.961961	0.964753	0.964347	0.871274	0.992542	180
7	0.000309	0.010469	0.199238	0.012018	0.081006	0.950605	0.967683	0.965206	0.871132	0.992854	180
8	0.000292	0.001768	0.012310	0.003041	0.094848	0.907593	0.972791	0.963346	0.857875	0.987999	180
9	0.000066	0.026880	0.140788	0.032940	0.097169	0.971356	0.932689	0.938291	0.810518	0.988794	180
10	0.000016	0.000041	0.328087	0.044983	0.100600	0.996047	0.902464	0.916025	0.756723	0.996492	180
11	0.000081	0.009674	0.397148	0.011001	0.101841	0.992591	0.919791	0.930339	0.788021	0.993378	180
12	0.000028	0.000030	0.446018	0.029951	0.103019	0.996543	0.900202	0.914161	0.752663	0.996707	180
13	0.000437	0.000009	0.178376	0.001462	0.104518	0.928367	0.967851	0.962128	0.858150	0.989650	180
14	0.000071	0.000183	0.567286	0.013873	0.106742	0.975307	0.917868	0.926187	0.774012	0.991504	180
15	0.000285	0.000101	0.485704	0.007089	0.121235	0.913071	0.960150	0.953322	0.829776	0.987757	180
16	0.000550	0.003563	0.035684	0.013571	0.125530	0.917492	0.953534	0.948312	0.817655	0.981420	180
17	0.000984	0.000413	0.531894	0.021653	0.133121	0.912536	0.942316	0.938003	0.782751	0.979182	180
18	0.000283	0.000021	0.517846	0.006761	0.136264	0.870570	0.963830	0.950314	0.807272	0.981254	180
19	0.000756	0.000040	0.228397	0.006329	0.137763	0.873515	0.967683	0.954038	0.821356	0.983223	180
20	0.000006	0.000341	0.187322	0.113134	0.144934	0.990120	0.876674	0.893112	0.705744	0.991239	180
21	0.000005	0.000335	0.159251	0.114269	0.145264	0.989132	0.888228	0.902849	0.724518	0.990745	180
22	0.000161	0.000061	0.752795	0.029591	0.150944	0.893786	0.920045	0.916238	0.726076	0.975114	180
23	0.000091	0.004295	0.678692	0.030926	0.178092	0.966419	0.844529	0.862184	0.647069	0.981001	180
24	0.000633	0.019773	0.485717	0.103101	0.183798	0.982210	0.853736	0.872354	0.666463	0.972479	180
25	0.000273	0.001384	0.703837	0.026212	0.183922	0.833486	0.919208	0.906789	0.678956	0.959292	180
26	0.000469	0.000467	0.196739	0.036494	0.184286	0.928357	0.889555	0.895184	0.730012	0.976338	180
27	0.000428	0.000032	0.598453	0.007457	0.192173	0.804301	0.949347	0.928335	0.723685	0.968913	180
28	0.000328	0.000002	0.576921	0.008098	0.192823	0.792046	0.960818	0.936355	0.748150	0.972260	180
29	0.000269	0.009618	0.750728	0.081702	0.215749	0.951099	0.836491	0.853093	0.621407	0.963628	180
30	0.000023	0.018990	0.578822	0.081846	0.221361	1.000000	0.737191	0.775276	0.538963	0.995457	180
31	0.000003	0.000025	0.080092	0.286233	0.295118	0.895735	0.814885	0.826602	0.553503	0.933230	90

Figure 7. Cross-validation metrics of the best 32 out of 64 base models. Top models have metrics unexpectedly close to 1 and validation loss close to 0.

	lr	l2reg	dropout	training_iteration	train_loss	val_loss	val_sensitivity	val_specificity	val_accuracy	val_mcc	val_auc
0	0.000005	0.007775	0.452802	128	0.223008	0.284818	0.823549	0.846120	0.840050	0.629601	0.916507
1	0.000042	0.000134	0.508670	64	0.147851	0.305358	0.782082	0.904006	0.871214	0.677173	0.916661
2	0.000024	0.000033	0.356980	64	0.141217	0.306471	0.816553	0.852712	0.842987	0.632455	0.916207
3	0.000300	0.000614	0.614658	128	0.224025	0.314895	0.783959	0.877009	0.851983	0.639100	0.907139
4	0.000008	0.000684	0.016450	32	0.170751	0.319662	0.784642	0.870731	0.847577	0.630877	0.903084
5	0.000637	0.000023	0.617275	64	0.180690	0.326102	0.753925	0.908777	0.867129	0.662310	0.911489
6	0.000008	0.007142	0.509299	32	0.336744	0.327195	0.842491	0.739955	0.767533	0.523854	0.892896
7	0.000857	0.000007	0.565517	64	0.149415	0.340633	0.756997	0.923091	0.878419	0.687656	0.915482
8	0.000008	0.241107	0.081641	32	0.113402	0.342847	0.798805	0.861376	0.844547	0.629559	0.901305
9	0.000237	0.000498	0.175179	32	0.113131	0.370779	0.797952	0.869412	0.850193	0.639793	0.912475
10	0.000641	0.082625	0.402806	32	0.489197	0.485721	0.903925	0.347250	0.496971	0.248135	0.787768
11	0.000276	0.143986	0.673845	32	0.544965	0.546870	0.993515	0.007785	0.272903	0.006712	0.664558

Figure 8. Tuning of base model with ASHA scedular and validation loss target.

	lr	l2reg	dropout	training_iteration	train_loss	val_loss	val_sensitivity	val_specificity	val_accuracy	val_mcc	val_auc
0	0.000016	0.000156	0.528336	128	0.145718	0.305243	0.787372	0.899360	0.869240	0.674323	0.918431
1	0.000013	0.076754	0.421737	128	0.172806	0.309982	0.779522	0.891135	0.861116	0.655875	0.900890
2	0.000034	0.001854	0.637467	128	0.146840	0.287889	0.825256	0.853277	0.845741	0.640252	0.916721
3	0.000099	0.038851	0.332137	128	0.308225	0.311829	0.813823	0.846434	0.837663	0.622085	0.907520
4	0.000005	0.000025	0.376992	128	0.231702	0.285483	0.816724	0.842918	0.835873	0.619940	0.913454
5	0.000090	0.100050	0.054485	128	0.340562	0.354040	0.816894	0.791939	0.798651	0.558604	0.889324
6	0.000236	0.056993	0.591475	128	0.432703	0.411353	0.820307	0.678240	0.716449	0.443862	0.850603
7	0.000563	0.034770	0.618875	32	0.453571	0.432286	0.809898	0.669952	0.707591	0.426977	0.831114
8	0.000821	0.097867	0.522401	32	0.543322	0.543180	0.746246	0.431630	0.516247	0.162187	0.667802
9	0.000061	0.324821	0.522496	128	0.549996	0.549517	0.991126	0.010422	0.274188	0.006898	0.654939
10	0.000163	0.670320	0.214254	32	0.556435	0.555833	0.992833	0.007283	0.272352	0.000604	0.639558
11	0.000040	0.213975	0.665953	64	0.548111	0.548147	0.995222	0.004583	0.271021	-0.001273	0.660877

Figure 9. Tuning of base model with ASHA scedular and validation MCC target.

	lr	l2reg	dropout	training_iteration	train_loss	val_loss	val_sensitivity	val_specificity	val_accuracy	val_mcc	val_auc
0	0.000044	0.000556	0.249741	128	0.051535	0.483101	0.746542	0.906148	0.863778	0.651402	0.894626
1	0.000019	0.021477	0.163935	128	0.049893	0.447210	0.741010	0.902524	0.859648	0.641356	0.886254
2	0.000042	0.069777	0.346589	128	0.192141	0.340132	0.768326	0.866783	0.840646	0.610946	0.877645
3	0.000057	0.012363	0.081431	128	0.059512	0.429771	0.764523	0.864909	0.838260	0.605276	0.876755
4	0.000057	0.043998	0.170153	64	0.148471	0.352469	0.788036	0.843539	0.828805	0.595864	0.890270
5	0.000070	0.000175	0.661016	64	0.258704	0.320454	0.801176	0.822794	0.817055	0.580484	0.896443
6	0.000080	0.070236	0.080160	64	0.193623	0.345562	0.765560	0.845789	0.824491	0.580420	0.879690
7	0.000027	0.000074	0.361113	64	0.227755	0.318503	0.824343	0.794801	0.802644	0.566420	0.897612
8	0.000010	0.940119	0.073157	32	0.298475	0.343992	0.846819	0.737441	0.766477	0.523139	0.886413
9	0.000018	0.029831	0.429220	32	0.325946	0.344128	0.872061	0.677331	0.729025	0.486167	0.885543
10	0.000024	0.000010	0.469789	32	0.343688	0.357184	0.863416	0.665334	0.717918	0.467556	0.876322
11	0.000015	0.000025	0.465111	32	0.383430	0.394661	0.904910	0.525244	0.626033	0.386054	0.861431

Figure 10. Tuning of base model with BOHB scedular and validation MCC target, validation metrics

	lr	l2reg	dropout	training_iteration	train_loss	train_sensitivity	train_specificity	train_accuracy	train_mcc	train_auc
0	0.000044	0.000556	0.249741	128	0.051535	0.985207	0.969542	0.973770	0.935797	0.997925
1	0.000019	0.021477	0.163935	128	0.049893	0.988182	0.972245	0.976547	0.942506	0.998306
2	0.000042	0.069777	0.346589	128	0.192141	0.945758	0.905388	0.916284	0.807118	0.977721
3	0.000057	0.012363	0.081431	128	0.059512	0.985207	0.971208	0.974986	0.938613	0.997576
4	0.000057	0.043998	0.170153	64	0.148471	0.944907	0.914597	0.922779	0.819717	0.983221
5	0.000070	0.000175	0.661016	64	0.258704	0.872131	0.832118	0.842918	0.652119	0.937267
6	0.000080	0.070236	0.080160	64	0.193623	0.932579	0.897592	0.907036	0.785895	0.973378
7	0.000027	0.000074	0.361113	64	0.227755	0.896106	0.849092	0.861782	0.692629	0.952225
8	0.000010	0.940119	0.073157	32	0.298475	0.897551	0.777079	0.809597	0.610070	0.931191
9	0.000018	0.029831	0.429220	32	0.325946	0.840078	0.776859	0.793923	0.561583	0.897655
10	0.000024	0.000010	0.469789	32	0.343688	0.832511	0.752970	0.774440	0.529443	0.885153
11	0.000015	0.000025	0.465111	32	0.383430	0.828856	0.694317	0.730632	0.467133	0.858704

Figure 11. Tuning of base model with BOHB scedular and validation MCC target, train metrics

index	lr	l2reg	dropout	training_iteration	train_loss	val_loss	val_sensitivity	val_specificity	val_accuracy	val_mcc	val_auc
0	0.000007	0.000005	0.129619	128	0.229003	0.263235	0.836348	0.862130	0.855196	0.660874	0.926999
1	0.000096	0.004641	0.286295	58	0.267585	0.273885	0.801365	0.881027	0.859602	0.658732	0.920604
2	0.000024	0.026108	0.163128	57	0.236908	0.262004	0.841809	0.854219	0.850881	0.654877	0.926000
3	0.000022	0.073243	0.477044	128	0.326225	0.314559	0.863993	0.789302	0.809390	0.594921	0.918142
4	0.000057	0.021322	0.477708	64	0.314867	0.298136	0.855802	0.785912	0.804709	0.584297	0.913181
5	0.000033	0.222734	0.582932	64	0.375410	0.351609	0.847611	0.749058	0.775565	0.537651	0.894202
6	0.000005	0.001417	0.463218	32	0.415049	0.379371	0.772355	0.779068	0.777263	0.506653	0.863669

Figure 12. Tuning of Graph model with BOHB scedular and validation MCC target, validation metrics

	lr	l2reg	dropout	training_iteration	train_loss	val_loss	val_sensitivity	val_specificity	val_accuracy	val_mcc	val_auc
0	0.000040	0.000021	0.342752	128	0.223459	0.252367	0.829352	0.882471	0.868184	0.683084	0.932507
1	0.000099	0.000031	0.273564	56	0.241538	0.257691	0.821160	0.883099	0.866440	0.677453	0.929383
2	0.000049	0.000027	0.375265	45	0.298600	0.285141	0.782253	0.894274	0.864145	0.662680	0.917205
3	0.000051	0.005045	0.388225	57	0.275756	0.268789	0.827474	0.862130	0.852809	0.653726	0.923795
4	0.000010	0.241852	0.079094	128	0.209751	0.273416	0.852389	0.839967	0.843308	0.645271	0.920998
5	0.000044	0.020757	0.325201	128	0.268705	0.271727	0.843515	0.839716	0.840738	0.637729	0.925170
6	0.000020	0.000006	0.367319	77	0.326337	0.304264	0.802901	0.845932	0.834358	0.612532	0.907809
7	0.000013	0.011756	0.454902	32	0.342267	0.312898	0.815188	0.817491	0.816872	0.587149	0.902803
8	0.000074	0.000017	0.683271	32	0.344125	0.309513	0.814505	0.809643	0.810951	0.577206	0.902612
9	0.000008	0.001363	0.325908	32	0.356665	0.325658	0.805802	0.808011	0.807417	0.568052	0.894086
10	0.000009	0.001440	0.604842	32	0.432808	0.392874	0.823208	0.683262	0.720901	0.451156	0.855787
11	0.000042	0.482847	0.661185	32	0.440781	0.402094	0.941809	0.378767	0.530200	0.312461	0.869642

Figure 13. Tuning of Graph model with BOHB scedular and validation MCC target, validation metrics, 12 models