Flattening the curve - How to get better results with small deep-mutational-scanning datasets

Gregor Wirnsberger[1], Iva Pritišanac[2], Gustav Oberdorfer[3], Karl Gruber[1*]

1 Institute of Molecular Biosciences, University of Graz, Graz, Austria
2 Institute of Molecular Biology and Biochemistry, Medical University of Graz, Graz, Austria
3 Institute of Biochemistry, Graz University of Technology, Graz, Austria

*karl.gruber@uni-graz.at

Abstract

Proteins get used in a wide variety of applications. For some applications, a specific protein property needs to be improved. Deep mutational scanning data can then be used to train a neural network, which can predict the effect of variants that have not been experimentally determined. Many approaches use the protein's sequence in one form or another as data representation for their prediction method. Since proteins are not one-dimensional strings but feature a complex 3D structure containing interacting residues, using the protein's structure as input brings the advantage of already encoding the interactions rather than having to learn them. Here we present a new representation for describing protein structures using stacked 2D contact maps. The contact maps represent residue interactions, their evolutionary conservation, and changes in interactions that occur due to mutations. Additionally, we provide different methods to improve the performance when training neural networks on small (50 - 2000 data points big) deep mutational scanning data sets. To test this approach, we trained three neural network architectures on three deep mutational scanning datasets and assessed their performance compared to a neural network trained on protein sequences alone. We demonstrate the utility of this representation for training neural networks to predict the outcome of deep mutational scanning experiments. The use of structural representation, as well as data augmentation and pre-training, substantially reduced the training data requirements and led to improved performance on smaller data sets and comparable performance when bigger data sets are available compared to the state-of-the-art sequence convolutional neural network, which demonstrates the power of our approach. This work could pave the way to expanded usage of deep mutational scanning by substantially reducing experimental requirements and opens the door for smaller facilities that might not have access to high throughput screening.

Author summary

We introduce a novel approach to improve predictions of protein properties based on deep mutational scanning (DMS) data. Our method utilizes a unique data representation for protein structures through stacked 2D contact maps, which capture residue interactions and alterations in these interactions due to amino acid substitutions, as well as evolutionary conservation. We also tested several techniques to enhance the performance of neural networks when trained on small datasets. By training three neural network architectures on three public DMS datasets, we demonstrate the value of

integrating protein structure information in training neural networks to predict the effects of amino acid substitutions. Utilizing structural information along with data augmentation and pre-training significantly improves performance on smaller datasets compared to state-of-the-art sequence convolutional neural networks while achieving similar performance on larger datasets. Thus, our approach could substantially reduce the experimental input needed and provide valuable predictions even for smaller DMS datasets when high-throughput screening is not feasible.

Introduction

Proteins are found in viruses, bacteria, plants, and humans and fulfil a large number of different functions and tasks. Due to their large number of diverse functions, they can be used in many areas. However, the naturally occurring proteins are often not optimal for the new desired use. Therefore, certain amino acids get exchanged to allow for an improvement of their function. This can be the increase in the brightness [1], the change in the binding target of an antibody [2], and much more.

Amino acid exchanges in proteins can alter their activity and a variety of their properties. To assess their effect, mutagenesis can be a useful method. In order to get more data on genotype-phenotype coupling, deep mutational scanning is a viable method. It allows the creation of large datasets describing the effects of mutations in a protein of interest [3]. Deep mutational scanning experiments feature a combination of some kind of protein display that preserves a physical link from a protein to its encoding nucleic acid sequence and high-throughput sequencing in order to generate data for as many as 10^5 variants of a protein. This is done by enforcing a selective pressure based on the protein's function on a diverse library (collection of protein variants). They get sequenced before and after the selective pressure was applied. The high-throughput sequencing quantifies the abundance of each variant. Variants with beneficial mutations will be enriched throughout the selection, and variants with deleterious mutations become depleted and thus offers the possibility to quantify the fitness of a large sequence diversity for a protein of interest [4]. Deep mutational scanning is used in a variety of different settings. Some examples of the various applications of this method are; examining the sequence determinants of the Aβ aggregation in Alzheimer's disease [5], the binding behaviour of proteins [6], the forecast of the evolutionary fates of human H3N2 influenza variants [7], the optimisation of antimicrobial peptides [8] to determining the effects of mutations in SARS-CoV-2 proteins [9] [10].

With the increasing number of deep mutational scanning experiments, the number of predictive methods also increases. Thus, many methods already specialise in predicting the effects of mutations in proteins. Some of them use only evolutionary data and no experimentally determined data to predict the functional consequences of amino acid substitutions. For this kind of modelling, approaches like the use of profile hidden Markov model [11], Potts model (EVmutation [12]) or a variational auto-encoder (DeepSequence [13]) are used. Others are natural language processing models, which are strongly influenced by the training approaches used in their field of origin. They get pre-trained in an unsupervised manner on a large amount of data and then fine-tuned on the prediction task. Here, models like LSTM [14] and transformer [15] are used. Then some models use decision tree ensembles (Envision [16]) trained on deep mutational scanning data or use Gaussian processes [17] for predictions. All of them act on protein sequences but do not use the protein's structure in their learning process. Another important aspect in training ML models is training efficient encoding of the underlying data. In the case of proteins, this can be the amino acid sequence alone without any 3D information [18], a graph representation of the protein structure [18], or voxel-based spatial structural encoding [19]. In recent years, models used in natural

language processing have increasingly been applied to problems with proteins. Although these models are very powerful and can produce great results, they tend to need a massive number of parameters, leading to high memory and computation requirements [20].

Since protein structure is more conserved than sequence [21], we created a - to our knowledge - new encoding for protein structures to take advantage of the information contained in the 3D structure. The encoding consists of 2D contact maps representing different physico-chemical properties of amino acids and their accompanying interaction, as well as the evolutionary conservation of each interacting residue in the structure (Section 0.2). In addition, this encoding allows the use of standard architectures for image classification networks, thus giving access to a large number of different architectures that can be used to solve this problem. Furthermore, we create a helpful pre-training and data augmentation protocol that helps to improve results when only a small amount of data is available (Fig. 1).

To study the applicability as well as the performance of our approach when only a limited number of training data is available, we trained different architectures on training datasets ranging from 50 to 6000 samples. We compared the performance using the same architecture with sequence input and our structure representation, as well as the influence of pre-training and data augmentation. We also investigated the performance of architectures with a smaller number of parameters. This showed that our representation alone gives an advantage but that data augmentation and pre-training are important for good performance. We also showed that our workflow could perform well with an architecture that has a greatly reduced number of parameters.

Materials and methods

0.1 Data

For our work, we used deep mutational scanning data, which was already prepared and used in [18]. In detail, we used data from avGFP, Pab1 and GB1 since these proteins yielded the best results in this paper, which makes them well-suited for comparison since the quality of the data set doesn't influence the results. For the structure input, we also used the provided structures from [18] to be sure the sequence and, therefore, the structure matches the deep mutational scanning data. These data sets contain nonsense mutations. We decided not to use assay scores of a protein that features one or more nonsense mutations since this would mean using scores of a protein fragment, and this, therefore, doesn't reflect the property of the wild-type protein containing a certain mutation. Therefore we decided to alter the data sets by excluding all nonsense mutations during training, validation and testing.

0.2 Interactions and their encoding

To emulate the effect of different mutations in a protein, we create interaction matrices that use a set of different amino acid properties to describe the interactions between residues in a protein and their change due to mutations. Additionally, a matrix that encodes the evolutionary conservation of interacting residues and an index matrix are used. Visual representations of the individual matrices, using Pab1 as an example, can be found in Fig. S1.

0.2.1 Distance Matrix

To know which residues interact with each other, the protein's crystal structure is used. The coordinates of all atoms of the protein structure are stored in the corresponding

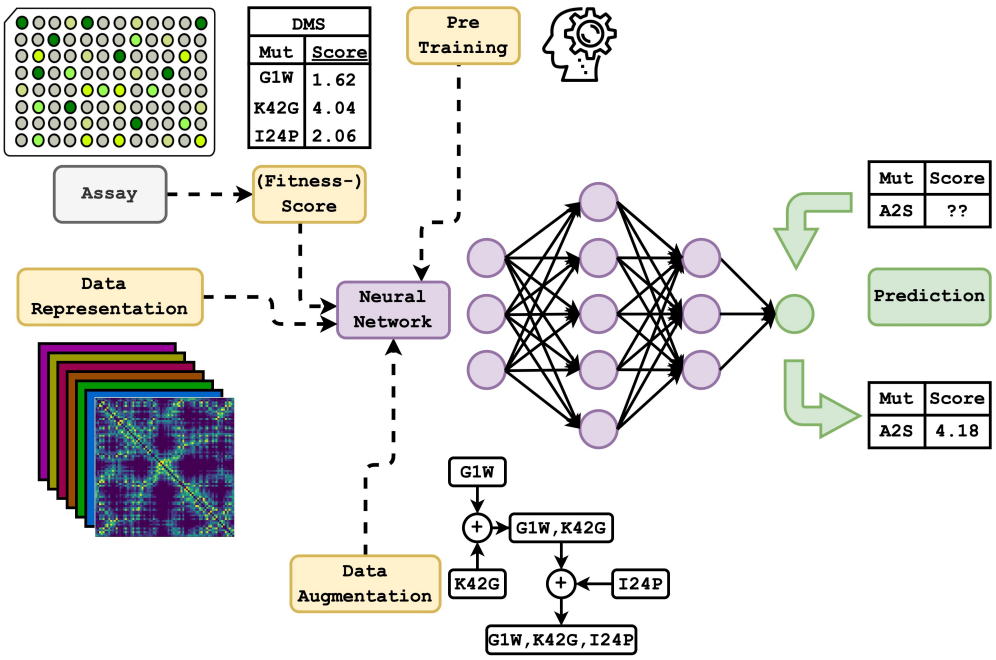

Fig 1. Overview of the training and prediction workflow. Initially, models are pre-trained on predicting a pseudo score that arises from the data representation (consisting of stacked 2D contact maps representing different physico-chemical properties and evolutionary information). This helps the model adjust its weights to the kind of prediction it will later be used for while not requiring additional data acquisition. Data augmentation is applied to up-size the training data to improve the prediction quality further. This is then used to train the network on experimentally determined (fitness-) scores of the protein of interest. In the end, the trained model can be used to predict these scores and, therefore, the effects of amino acid exchanges in the protein that were not experimentally determined. It is also possible to omit pretraining and data augmentation and train the network solely on experimentally determined data. Three different network architectures were used in this study, but they can be easily changed to any architecture of choice that accepts the input in the form of the data representation.

pdb file [28].

$$d_{\mathrm{ij}} = \sqrt{(x_i - x_j)^2 + (y_i - y_j)^2 + (z_i - z_j)^2} \tag{1}$$

These are used to calculate the distance (d_{ij}) between all atoms of the protein (Eq: 1). Specifying interacting residues is done by checking the distance of the closest side chain atoms between $Residue_i$ and $Residue_j$ for all residues. Using this approach, the distances between all residues are calculated, and a $n \times n$ distance matrix (D), where n denotes the sequence length, is created. The created matrix is symmetric around the diagonal. This matrix is then converted to a boolean matrix. Therefore a threshold ($\mathrm{dist_{th}}$) of 20 Å is used. The conditions are: $\mathrm{d_{ij}} \leq \mathrm{dist_{th}}$ gets converted to True and $\mathrm{d_{ij}} > \mathrm{dist_{th}}$ to False. The resulting matrix (M) is used in the factor matrix (F) as a mask to set interactions of residues farther apart than the distance threshold to zero.

$$F_{\mathrm{ij}} = 1 - \frac{d_{\mathrm{ij}}}{max(D)} \tag{2}$$

Where d_{ij} denotes the distance between two residues and *max(D)* the biggest distance seen in the structure. The distance matrix gets also used to scale the "strength" of the interactions as a factor matrix (F) (Eq: 2) in all subsequent matrices (apart from the position matrix (P)) by calculating the Hadamard product of F with each interaction matrix. This leads to a higher value for close interactions and a lower one for interactions of residues that are further apart and, in addition, masks interactions that originate from residues that are further apart than $\mathrm{dist}_{\mathrm{th}}$.

0.2.2 Index Matrix

Convolution neural networks (CNN) are translation invariant. This is one of the features that make them powerful in image recognition tasks since they can find patterns they have learned anywhere in an image and not rely on their position. In our case, this translation invariance is an undesirable feature because the positions of the interactions matter. To combat this issue, we introduce a simple position matrix (P). It describes the position of each interaction in the matrices based on the index matrix I (Fig. 2). Its not interacting residues are masked using the M matrix, where each True value is converted to 1 and each False to 0. To calculate P, the Hadamard product of M and I is formed.

$$I = \begin{bmatrix} 0 & 1 & 2 & 3 \\ 1 & 0 & 6 & 7 \\ 2 & 6 & 0 & 11 \\ 3 & 7 & 11 & 0 \end{bmatrix}$$

Fig 2. Sample index matrix before normalising the values to be in the range $[0, 1]$

0.2.3 Hydrogen Bonding

The number of hydrogen bonds is one of the factors that determine the stability of a protein. Therefore it is a crucial kind of interaction since mutations that introduce hydrogen bonding capabilities or remove them will thus alter this property. Not all amino acids have the same capability of forming hydrogen bonds with their side chain. Some can only act as a donor(K, R, W), some as an acceptor (D, E), some as donors or acceptors (H, N, Q, S, T, Y) and some are not able to form hydrogen bonds with their side chain at all (A, C, F, G, I, L, M, P, V).
The hydrogen bonding capability values for each amino acid are used to calculate the "interaction value" (Eq: 3) for each residue against each residue. This is used to determine if hydrogen bonding is possible (acceptor - donor, acceptor - acceptor/donor, donor - acceptor/donor, acceptor/donor - acceptor/donor interactions). If so, the interaction gets marked as True, otherwise as False. This creates a $n \times n$ boolean hydrogen bonding matrix (B) describing possible hydrogen bonds inside a protein. The values get changed to 1 for True and 0 for False for further computations.

$$B_{\mathrm{ij}} = b_{\mathrm{i}} \times b_{\mathrm{j}} \tag{3}$$

Where b denotes the hydrogen bonding capability of a certain residue.

0.2.4 Hydrophobicity

Proteins often contain a hydrophobic core and a hydrophilic outside that interacts with its surroundings. The hydrophobic core plays an important role in the folding process of

a protein. Therefore, mutations that change the hydrophobicity in certain areas of a protein can have positive and negative effects.
The hydrophobicity values are obtained from Parrot [31]. The hydrophobicity values range from -4.5 for arginine to 4.5 for isoleucine.

$$H_{\mathrm{ij}} = 1 - \frac{|h_{\mathrm{i}} - h_{\mathrm{j}}|}{9} \tag{4}$$

Where h denotes the hydrophobicity of a certain residue, and 9 is the maximum possible hydrophobicity difference.
The hydrophobicity matrix (H) (Eq: 4) describes how well-interacting residues match in terms of their hydrophobicity.

0.2.5 Charge

There are three main types of amino acids in terms of charge. They can be divided into neutral (A, C, F, G, I, L, M, N, P, Q, S, T, W, Y), positively (R, H, K) and negatively (D, E) charged. Salt bridges, which are interactions of residues of opposite charge, are, besides hydrogen bonds, another type of interaction that is important for the stability of a protein. On the other side, amino acids with the same charge may repel each other and thereby destabilise the structure of a protein. To obtain the charge matrix (C) (Eq: 5), interactions of positively charged amino acids get a value of 1 assigned, interactions of amino acids carrying the same charge get a value of -1 assigned, and all other interactions get a value of 0 assigned.

$$C_{\mathrm{ij}} = c_{\mathrm{i}} \times c_{\mathrm{j}} \times -1 \tag{5}$$

Where c denotes the charge "value" of a certain residue encoded.

0.2.6 Surface accessible side chain area

Amino acids feature a variety of different sizes of their side chain. This is reflected in the difference in their surface accessible side chain area (SASA). The bigger the SASA, the higher the possibility for a (strong) interaction. Therefore a mutation that changes the interaction area between two interacting residues can have an influence on their interaction strength. The SASA values are obtained from Parrot [31], ranging from 0 for glycine to 254 for tryptophan.

$$A_{\mathrm{ij}} = \frac{a_{\mathrm{i}} + a_{\mathrm{j}}}{max_{\mathrm{SASA}} \times 2} \tag{6}$$

a denotes the interaction area of a certain residue and max_{SASA} the maximum SASA value for an amino acid.
The interaction area matrix (A) (Eq: 6) describes the interaction area between residues.

0.2.7 Clashes

Amino acids also differ in the length of their side chains. That means certain mutations can lead to potential "holes" in a protein if the side chains get shorter or potential clashes because the side chains are too long for the space between them.

$$X_{\mathrm{ij}} = \frac{\Delta l_{\mathrm{i}} + \Delta l_{\mathrm{j}} + d_{\mathrm{ij}}}{2 \times max_{\mathrm{l}} + dist_{\mathrm{th}}} \tag{7}$$

Δl denotes the change in the side chain length at a certain residue position from wild-type to the variant, max_{l} the maximum side chain length and $dist_{\mathrm{th}}$ the maximum allowed distance between two residues to count as interacting.

Side chain lengths range from 0 Å for glycine to 8.28 Å for arginine. The values of the side chain length are obtained by measuring the distance from the Cα to the side chain atom with the biggest distance using Pymol [32]. The clash matrix (X) (Eq: 7) represents the distances between interacting side chain residues. If a mutation leads to a distance between two residues that is closer than the distance between them in the wild-type, a negative length value is created. This makes this matrix, besides the charge matrix C, the only matrix where values are in the range of [-1, 1] instead of [0, 1].

0.2.8 Evolution

To make use of the evolutionary information that can be found in blastP [33], we create a matrix (E)(Eq: 8) based on the conservation of amino acids at each sequence position. Therefore we use the result of a PBLAST search against the wild-type protein sequence and align the sequences and the wild-type sequence using the multiple sequence alignment tool clustal omega [34]. From this, the duplicated sequences get removed. To calculate the conservation at each sequence position, all present amino acids get counted at each position in the aligned wild-type sequence, and their number gets divided by the total number of amino acids at the specific sequence position. Amino acids that are not present at this position get a value of zero assigned. To evaluate the conservation of an interaction, the conservation scores of the interacting residues get multiplied.

$$E_{\mathrm{ij}} = e_{\mathrm{i}} \times e_{\mathrm{j}} \tag{8}$$

e denotes the conservation score of the current amino acid at its specific sequence position.
Figure S1 shows an example of all interaction matrices (B, H, C, A, X) for Pab1 containing the mutation "N127R, A178H, G177S, A178G, G188H, E195K, L133M, P125S" as well as the position matrix (P), the interaction matrix (M) which describes which residues interact with each other and the distance matrix (D).

0.3 Network architectures

0.3.1 Simple CNN

Since we wanted to compare our structure representation to the sequence convolution approach (Section: 0.3.4), a LeNet5 [22] - like convolutional neural network (Fig. S16) was used. It contains a feature extraction part containing three 2D convolution layers with 16, 32 and 64 filters and a kernel size of 3×3, each followed by a max pooling layer. After that comes a flatten layer and a classifier part consisting of 4 fully connected layers with 128, 256, 256 and 64 nodes and a single output node. We used the leaky rectified linear unit (leaky RELU) as the activation function for all layers in the model. Zero-padding is used throughout the whole network. This model is referred to as "simple CNN".

0.3.2 DenseNet

To compare the performance to a more recently described architecture, we chose to use a DenseNet [23] - like architecture (Fig. S17), which will be referred to as "DenseNet". Here the core building block consists of a 2D convolution layer with 128 filters and a kernel size of 1×1, followed by a 2D convolution layer with 12 filters and a kernel size of 3×3. Zero-padding is used throughout the whole network to keep each layer's input and output dimensions the same. The input into the first 2D convolution layer and the output of the second get concatenated. This is repeated 4 (*block depth*) times and is then followed by a 2D average pooling layer with a kernel size of 2×2. All this combined

is one block, and this is repeated 4 (*block number*) times. In the end, a 2D global average pooling layer is followed by a fully connected network with 128, 128, and 64 nodes per layer leading into one output node. Additionally, we used an "intro layer" for avGFP, which consists of a 2D convolution layer with 128 filters, a kernel size of 3×3 and a stride of 2 followed by a 2D max pooling layer with a kernel size of 3 and a stride of 2 at the beginning of the network. This reduces the size of the input and thereby reduces the number of computations needed in the rest of the network. In contrast to the original DenseNet we omitted batch normalisation because it led to worse performance and used the leaky RELU instead of RELU.

0.3.3 SepConvMixer

To test the performance of a network with as few parameters as possible, we implemented an architecture (Fig. S18) similar to ConvMixer [24]. Sequence convolution requires up to 82 times, simple CNN up 185 times, and DenseNet up to 21 times the number of parameters in our settings (Table 4). The two main contributors to the reduction of the number of parameters were the possibility of using a smaller fully connected network as classifier as well as the use of 2D separable convolution layers. The latter first perform a depth-wise spatial convolution, which acts separately on each input channel and is followed by a point-wise convolution to mix the resulting output channels. The network starts with one 2D separable convolution layer with 32 filters where we used a kernel size of 3×3 and a stride of 1 for smaller proteins (like Pab1 and GB1) and a kernel size of 9×9 and a stride of 9 for bigger proteins (avGFP). This is 9(*depth*) times followed by a block consisting of 2 2D separable convolution layers with 32 filters and a kernel size of 3×3. The input into the first, the output of the first, and the output of the second layer get added at the end of the block. This is followed by a 2D global max pooling layer and a fully connected network consisting of 128- and 64-node layers followed by a single-node output layer. We used the leaky RELU as well as zero-padding to keep the dimensions the same throughout the whole network. The used *depth* was 9. The "down-sampling" (a kernel size of 9×9 with a stride of 9 in the first layer) for bigger proteins slightly reduces the performance but is a worthy trade-off to reduce the computational cost.

0.3.4 Sequence convolution

For result comparison, we used the network architectures of [18] as specified in their main experiments (/pub/regression_args/PROTEIN_main_cnn.txt [35]). Apart from enabling early stopping and restricting the length of the training to 100 epochs, we chose the default parameters when using the /code/regression.py. This is referred to as "sequence convolution".

0.3.5 Implementation

Our models are implemented using Python v3.10, TensorFlow v2.9.1, and Keras v2.9.0

0.4 Training

The training was done using the mean absolute error as the metric, Adam as optimiser with a learning rate of 0.001 and a maximum number of epochs of 100. Furthermore, we have used a limit to stop the training if the mean absolute error did not improve by at least 0.01 over 20 epochs. The batch size for the training was 32 and parallelised by using 12 central processing unit (CPU) cores of an AMD Ryzen Threadripper 3960X. The training was performed on an Nvidia RTX A5000 graphics processing unit (GPU). For pretraining, we limited the maximum number of epochs to 70.

The training of sequence convolution was done using Google Colaboratory with the use of a provided GPU.

0.5 Experimental setup

For each training, we used three different data sets, which were all obtained from the original data sets of the proteins: a train, a tune and a test set. The test set always consists of 5000 randomly chosen unique entries each. The tune has one-fifth of the size of the training data set for our architectures and of always 5000 entries for sequence convolution. The train data sets contained 50, 100, 250, 500, 1000, 2000 or 6000 entries for all training runs. The train data set is the data set that the network uses to train on. The tune set is used to calculate the validation statistics during training, and the test set is used to calculate the statistics of the performance of the network after training. For the training of simple CNN, SepConvMixer and DenseNet we used data augmentation (Section: 0.6) as described below, as well as pre-training (Section: 0.7). Training simple CNN, SepConvMixer, DenseNet and sequence convolution was done on three randomly chosen subsets of the whole protein data sets to construct the train, tune and test sets to avoid picking one that suites one architecture better by chance. For sequence convolution, we used the same train, tune and test sets as for the training of simple CNN, SepConvMixer and DenseNet, however, we have not used data augmentation and transfer learning during its training process.

0.6 Data augmentation

Since neural networks learn better with more data, we used a simple data augmentation method to obtain more training data from small data sets. This method uses the given experimental data, e.g. Table 1, shuffles it and adds it to the original not shuffled data to create new augmented variants like shown in Table 2. This is done four times, and the newly created data is stored. This data is then used as input data to perform the same action three times where, after each round, the newly created data is used as the new input data in the next round. From this newly created augmented data, as many samples are drawn as needed to get a maximum of 20000 training samples when the original data is added ($aug_{\text{used}} = 20000 - n_{\text{original}}$ where aug_{used} is the number of augmented samples used and $\text{n}_{\text{original}}$ the number of original data). If the augmentation doesn't produce enough data to reach a combined number of 20000 samples after the original data is added, the whole augmented data is used. It didn't show good results when increasing the number of runs to produce +20000 samples when the original data set is not big enough to reach the number of samples with the number of runs described above. This kind of data augmentation produces pseudo labels for data and assumes an additive effect of mutations. These labels are not as accurate as measurements from an assay but create labels close enough to the true values to improve the learning capability of our networks. We also tried training the networks only on augmented data and fine-tuning them on the original small data sets. This showed worse performance than training them with the original and augmented data combined.

Table 1. Sample data for augmentation

variant	number mutations	score
K1L,S3A	2	-0.3
R23H,W19F	2	0.1
C5G,A7L	2	-1.0

Shows data samples later used for an example of data augmentation.

Table 2. Sample augmentation

augmented variant	number mutations	augmented score
K1L,S3A,R23H,W19F	4	-0.2
R23H,W19F,C5G,A7L	4	-0.9
C5G,A7L,K1L,S3A	4	-1.3

Shows how data points of 1 are added during data augmentation.

0.7 Pre-training

To overcome the need for big data sets, we used pre-training to obtain better results while training on small data sets. We used it only for simple CNN, SepConvMixer and DenseNet. The transfer of weights of the feature extraction part of a network trained on a whole dataset of another protein yields better performance than starting from a completely untrained network. However, to pre-train a network on data that is more closely correlated to the protein of interest, we created a pseudo-score that can be calculated without the need for experimental data (section: 0.7.1). After training the model on the pseudo data, the weights of the feature extractor got transferred to an untrained network, frozen, and a new classifier was trained. The same was done with a trainable feature extractor. During initial tests, the reduction of the learning rate did not improve the performance. Therefore we omitted it in further studies. Transferring the weights of the whole pre-trained model, including the classifier, showed worse performance. We also tested networks pre-trained on other proteins, e.g. pre-trained on avGFP and trained on Pab1, but our pre-training method proved to be more effective.

0.7.1 Pseudo Score

In order to calculate the pseudo score for the pre-training, the wild-type of the protein gets encoded in the same way as for training the network. The same is done for all possible single and double mutants of the protein. To calculate the pseudo score of a variant, the encoded wild type gets element-wise subtracted from the encoded variant matrix. Of all these values, the absolute value is taken and summed up over all matrices. This gets divided by 100 to shift the values into the range of the real deep mutational scanning scores. 40000 of these created data points are randomly chosen and used to pre-train the models. These pseudo scores show a Pearson's R of around -0.5 to the original deep mutational scanning data for the different datasets.

0.8 Recall Performance

To access the recall performance of simple CNN, SepConvMixer and DenseNet when trained on different-sized training datasets (Section: 0.5), we used the pre-trained models without data augmentation since this is one of the best-performing settings. The models were trained on different-sized training datasets (50 - 6000 data points) or 80% of the whole datasets. Then the test data set, which consists of only variants that the models have never seen before, was used to access the recall performance by letting the models predict the scores and checking how many of the predicted top-scoring variants were actually part of the actual top scoring 100 variants of the test dataset, given a certain budget (Fig. 4 & S15). If one then ranks all variants according to their predicted (fitness-) score, the budget refers to the number of best variants predicted by the network from all variants, which are examined to see whether they occur in the actual 100 best variants.

0.9 Single Mutation Effect Prediction

To test how many training samples a network needs to get an idea of the effect of single mutations, all single mutations of the deep mutational scanning dataset of GB1 were used as ground truth. Then pre-trained SepConvMixer models were trained on different numbers of training samples of the original deep mutational scanning datasets (50 -6000 data points). These models, as well as only the pre-trained model of SepConvMixer were asked to predict the score of every single mutation present. This was done for GB1 because this dataset consists of all possible single mutations, whereas the Pab1 and avGFP datasets would not yield a comparable ground truth due to missing single-point mutations.

0.10 Generalization

To test the models' capabilities in predicting mutants with a higher number of mutations than they were trained on, the avGFP dataset was used. This dataset is the only one containing variants with up to 14 mutations. Therefore, the training and tune sets consisted of 10,221 and 2,556 data points, respectively, featuring only single and double mutants. The test set consisted of 38,937 variants containing three to 14 mutations. The models were trained under four different settings: from scratch, meaning no pre-training or data augmentation; only with pre-training on our pseudo-score, which contains only scores for single and double mutants; only with data augmentation; and lastly with pre-training and data augmentation combined. The training was done three times with different random seeds to check the prediction consistency. Pearson's R values between the true and the predicted scores of the test set mutants were computed to evaluate the performance.

0.11 Data Availability

The data used to train all networks and the code can be found on GitHub (`https://github.com/ugSUBMARINE/image-dms`) and is licensed under the MIT license. It is set up so that it can easily be used to reproduce the results of this publication with all scripts used to obtain the results and to be easily used to pre-train and train our networks (or any other network of interest) on a new dataset.

Results

We tested new ways of encoding protein structure and improving the training on small data sets. To this end, a simple convolutional neural network with a LeNet5 [22] -like architecture (simple CNN, Section: 0.3.1), a DenseNet [23] -like Network (DenseNet, Section: 0.3.2) and a network heavily inspired by ConvMixer [24] (SepConvMixer, Section: 0.3.3) were used. Furthermore, two methods, data augmentation and pre-training, were tested for their applicability to deep mutational scanning (DMS) data. To assess their performance, a state-of-the-art sequence convolution model [18] (sequence convolution) was trained with the same data sets, and the results were compared. Furthermore, we tested the recall performance of the Top 100 variants in a data set as well as the predictive performance of the effect of single mutations for models trained on different amounts of data. To evaluate the "real world applicability", we conducted a synthetic experiment to see how well the three different models generalised to variants containing three or more mutations when they were only trained on data containing single- and double mutants.

0.12 Neural network training

In order to evaluate the performance of the three different architectures as well as of the original sequence convolution, each model was trained on three different data sets. In addition, different-sized parts of the datasets were used to determine the influence of the amount of training data. This showed improved performance of all three architectures for smaller datasets and the positive impact of pre-training and data augmentation on predictive performance. For larger datasets, all approaches perform almost equivalently. For all tests, three data sets, avGFP, Pab1 and GB1 from [18], were used. These were modified not to include nonsense mutations. From these modified data sets, three subsets were created: the train, tune and test sets. The train data set varied in size from 50 to 6000, whereas the tune and test data sets always consisted of 5000 entries each for the sequence convolution. For simple CNN, DenseNet and SepConvMixer, a smaller tune set size of one-fifth of the train data set size was used to test whether the size of the tune data impacts the performance. The test set size of 5000 entries to assess the models' performance was kept the same. The train set is used for training the models. The tune set is used as validation data to check the loss and metrics of the model after each epoch for data it has not seen during training. The test set is used after training to calculate the statistics of the model's performance since this contains data the model has never seen before. All tests were performed three times with randomly chosen samples of the whole modified data sets, and the median of the results was used to assess the performance. Three main performance metrics are used: mean squared error (MSE), Pearson's correlation coefficient and Spearman's correlation coefficient, with the main focus on Pearson's correlation coefficient. No dedicated hyper-parameter tuning was done, but those that had proven to be the best after some initial testing were used. To test the impact of an "intro layer" like in the original DenseNet, which is a normal 2D convolution layer with a kernel size of 3×3 and a stride of 2 followed by a 2D max pooling layer with a kernel size of 3×3 and a stride of 2, we chose to include this in the training of avGFP but not for Pab1 and GB1. The same was done for SepConvMixer, where the first separable convolution layer has either a kernel size of 3 and a stride of 1 or, for avGFP, a kernel size of 9 and a stride of 9. The use of an "intro layer" reduces the performance for smaller proteins like Pab1 and GB1 slightly but is needed and a good trade-off to be computationally efficient for proteins of the size of avGFP and bigger. Here the decrease in performance is smaller.

In order to improve predictions on small data sets, two methods were applied: data augmentation and pre-training. Data augmentation was already shown to be important when the data set size is small [25]. Since the proposed data representation is not translation and rotation invariant, it was not possible to use the same data augmentation methods (e.g. rotation, crop, flip, transpose, etc.) as used in image processing. Hence, a simpler data augmentation method was used that sums up scores of existing data (Section: 0.6). Another method to improve a model's performance is pre-training. Here a model can be pre-trained unsupervised if a lot of unlabelled data is available [26] or supervised on a big labelled data set with similar content to the data one is interested in and then fine-tuned on the data set of interest [27]. Since the proposed data representation already captures some variation due to mutations in a protein, we created a simple pre-training procedure, where the models are pre-trained on a pseudo-score that arises from the representations itself (Section 0.7.1).

In the figures (3 and S2 - S9), the median of the three training runs for each data point is shown. The graphs in the top row show either the Pearson-, the Spearman correlation coefficient or the MSE for the predictions on the test set made by the models. The bottom row shows the relative performance compared to the sequence convolution. The data set size always refers to the amount of data from the original split and is not related to the data set size after data augmentation. For the MSE, the relative

performance was calculated with $p_{\mathrm{MSE}} = 2 - (MSE_{\mathrm{i}}/MSE_{\mathrm{seqconv}})$ where p is the relative performance, the MSE_{i} the MSE of a model to compare to and MSE_{seqconv} the MSE of the sequence convolution. The relative performance of the correlation coefficients was calculated $p_{\mathrm{R}} = R_{\mathrm{i}}/R_{\mathrm{seqconv}}$ where p is the relative performance, R_i the correlation coefficient for the model to compare to and $R_{seqconv}$ the correlation coefficient of the sequence convolution model.

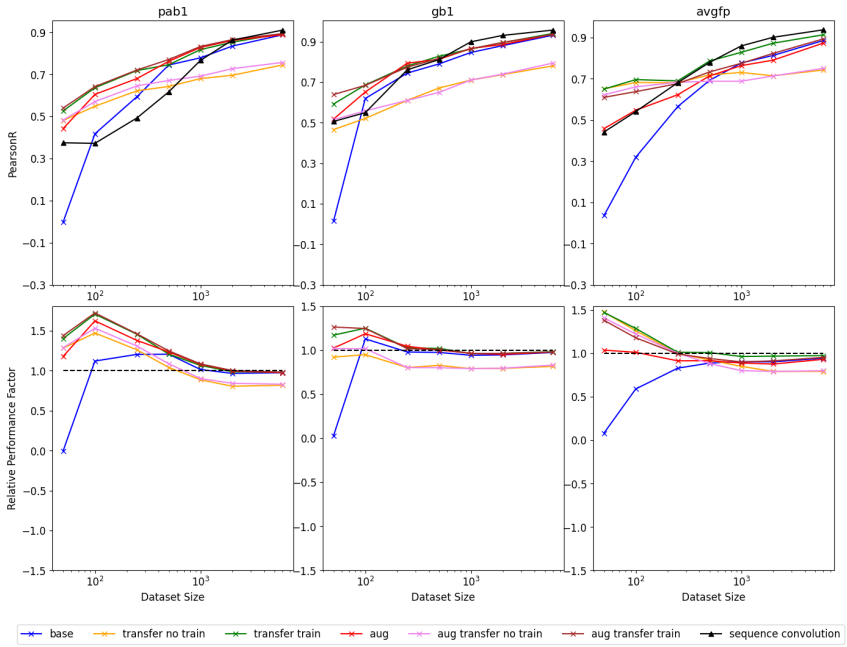

Fig 3. Pearson's R for predictions of the test data set of SepConvMixer for all three proteins in the upper row, as well as the relative performance compared to sequence convolution in the lower row. Here sequence convolution is indicated as a black dashed line at 1×of its own performance. Label descriptions can be found in Table 3
.

Table 3. Label description for result plots.

label	augmentation	transfer	train CL
sequence convolution	No	No	Yes
base	No	No	No
transfer no train conv	No	Yes	No
transfer train conv	No	Yes	Yes
aug	Yes	No	No
aug transfer no train conv	Yes	Yes	No
aug transfer train conv	Yes	Yes	Yes

Augmentation specifies whether data augmentation was used, transfer whether pre-training was used and train CL whether the convolution layers were trainable or not when pre-training was used.

In general, the more data the networks get to train, the better they perform, and the less important the approach becomes since they tend to perform almost equivalently (>= 2000 training samples). Another trend that can be observed is that the more original data the networks get, the less important augmentation and pre-training become to achieve the same training results.

The best performances were obtained when the networks were pre-trained, and the weights of the convolutional layer were not frozen in the subsequent training runs. Data augmentation had an additional positive effect. On smaller datasets (<= 500 training samples), the difference in the performance of a chosen method is more pronounced. For example, for Pab1 and avGFP, using data augmentation and freezing the convolutional layers show a better performance in simple CNN (Fig. S7) but show a worse performance when the training dataset gets bigger. In contrast, this method leads to an overall worse performance in DenseNet (Fig. S4) and SepConvMixer. This is especially true for SeqConvMixer, and can be caused by the low number of trainable parameters (13k) for the network under this setting (Fig. 3). Looking at the method that produces the best results, training a pre-trained network and using data augmentation, DenseNet has a similar performance overall to simple CNN and SepConvMixer. A performance improvement from DenseNet can be seen in small datasets (Fig. S10). When the number of training samples gets over 500, the performances of all architectures are almost identical. One fact that stands out about DenseNet is that it takes at least 6000 samples to show the same performance as sequence convolution when no pre-training and data augmentation is used. In contrast, simple CNN without pre-training and data augmentations needs 250 to 500 training samples to show the same performance as sequence convolution or even surpass it. SepConvMixer shows the same trend, starting to surpass sequence convolution at 100 training samples without pre-training and data augmentation on Pab1 and GB1 but needs pre-training in order to show improved performance on avGFP (Fig. 3).

In general, comparing the correlations of all three networks to sequence convolution, the best methods (using pre-training with and without data augmentation) perform at worst 0.95×of sequence convolution but most of the time better, especially on small data sets. In general, the difference in performance for different methods is less pronounced in simple CNN than in DenseNet and SepConvMixer.

Data augmentation works well for data sets that feature only single and double mutants, such as Pab1 and GB1. When the dataset already consists of variants with more than two mutations (up to 14 in the case of avGFP), the approach doesn't work as well when the training data size surpasses 250 entries. This might be caused by the additive nature of the data augmentation used in our pipeline. Since adding two single mutants is more likely to be additive in real life compared to adding two variants, both carrying 12 mutations on their own.

When comparing the number of parameters (Table: 4) for sequence convolution and simple CNN, the protein sequence length is the determining factor. The bigger the protein, the more will the simple CNN exceed the sequence convolution in terms of the number of parameters. For DenseNet and SepConvMixer the number of parameters will stay constant and only change due to the use or absence of the first introduction layer. This difference is due to the use of a flatten layer in the sequence convolution and simple CNN after their feature extraction part, whereas DenseNet and SepConvMixer both use a global pooling layer instead that always has the same size, regardless of the input data.

0.12.1 Performance Comparison

Comparing sequence convolution to simple CNN shows the advantage of incorporating the structure in the model learning process because these two approaches use the same kind of network architecture. For training set sizes >= 100, simple CNN without pre-training and data augmentation shows a performance up to 1.35×of the performance of sequence convolution or at least the same performance in terms of Pearson's R (Fig. S7). For training sets containing 50 data points, it shows a relative performance p_{R} between zero and 1. SepConvMixer performs similarly to simple CNN and does not rely on data augmentation and transfer learning to show improvements over sequence

Table 4. Number of parameters of each network.

architecture	protein	trainable parameter
sequence convolution	Pab1	990k
simple CNN	Pab1	803k
DenseNet	Pab1	714k
SepConvMixer	Pab1	37k
sequence convolution	avGFP	3.118k
simple CNN	avGFP	7.029k
DenseNet	avGFP	799k
SepConvMixer	avGFP	38k

Number of trainable parameters of the three different architectures for two different proteins: Pab1 (75 amino acids) and avGFP (237 amino acids).

convolution despite its low number of parameters. The usage of pre-training and data augmentation improves the performance, especially on smaller datasets and can lead to 1.2×and up a maximum performance of 1.75×compared to the sequence convolution (Fig. 3). At worst, it performs comparably to sequence convolution over all dataset sizes. Using a more complex architecture (DenseNet) can improve the performance on smaller dataset sizes ($<= 250$) but needs at least pre-training to reach that level of performance (Fig. S10). Looking at the difference in performance between simple CNN, SepConvMixer and DenseNet, one can see that DenseNet can improve the performance for smaller datasets when pre-training and/or data augmentation is used. On the other hand, when none of these methods is used, DenseNet shows a strongly reduced performance and higher variability in its results (Figs. S11 - S13).

0.13 Predictive Performance

The recall performances of models trained on up to 6000 data points show a similar overall performance. When asked to predict all single mutation effects, a SepConvMixer trained on 250 data points will already be able to qualitatively predict advantageous and deleterious amino acid exchanges.

Comparing the recall performance of the top 100 variants of the test data set, one can see that the pre-trained models of simple CNN, DenseNet and SepConvMix perform very similarly (Fig. S15). For SepConvMix trained on 6000 training samples, it takes a budget size between 70 to 1040 samples to recall 60 % of the top 100 variants. When trained on 500 training samples, it needs between 500 to 1270 samples to match this performance, and when trained on 50 training samples, 1390 up to 1700 samples are required to reach this performance (Fig. 4).

Testing the performance on predicting the effects of single mutations of pre-trained SepConvMixer networks trained on reduced dataset sizes (Section: 0.5) shows under visual comparison that for GB1, a protein with a sequence length of 56 amino acids, models trained on 250 training samples start to have a good idea of which single mutations has a positive and which has a negative effect (Fig. 5). This comparison was only possible for GB1 since its data set contains all possible single mutants.

0.14 Generalization

Examining the performance of models trained on only single and double mutants in predicting mutants that have more than two amino acid exchanges again shows the advantage of pre-training and data augmentation.

In order to determine how well the different neural networks are applicable in “real lab applications”, a synthetic experiment was conducted. The various networks (simple

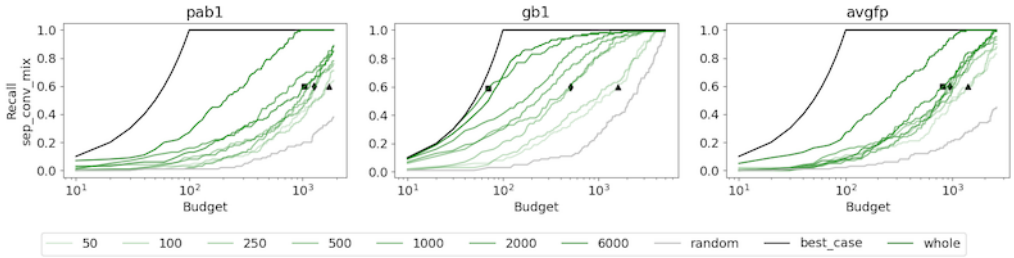

Fig 4. Recall of the top 100 test set mutations given a certain budget (number of predictions that may contain the true Top 100) for SepConvMixer. The models were trained on train data sets containing 50 - 6000 data points or on 80% of the whole data set, which is labelled "whole". The 60% recall performances when trained on 6000 data points are shown as ■, as ♦ when trained on 500 data points and as ▲ when trained on 50 data points.

CNN, DenseNet, SepConvMixer) were trained on 10.221 single and double mutants of avGFP, with and without pre-training and/or data augmentation. Then they were asked to predict the test dataset that contains variants featuring a minimum of three and a maximum of 14 mutations. This led to a maximum performance in terms of Pearson's R of 0.835 (Fig. 6). Over all settings, simple CNN showed the best results, followed by SepConvMixer. DenseNet showed the worst performance. Looking at the consistency of the results, SepConvMixer outperforms the other two networks. In accordance with previous results, the models can improve their predictions when pre-training and data augmentation are used. Also in line with previous experiments, the best setting combination is pre-training combined with data augmentation. Under these settings, all models perform the best and deliver the same performance.

Discussion

It was previously shown that incorporating the structure in the form of a graph and training a graph neural network to predict deep mutational scanning results achieves the same performance as sequence convolution [18]. By introducing our new protein structure representation, we could show that even without pre-training and data augmentation, a better correlation between experimentally determined and predicted scores can be achieved using the same architecture with a smaller number of filters. Furthermore, we can show that the predictive performance can be greatly improved by two straightforward but effective methods of pre-training and data augmentation. Even though the pre-training is very effective, it is not enough to freeze the convolution layers and only let the fully connected layers be trainable. This shows already worse performance after the train sample size exceeds 500 samples, even though simple CNN can compensate better because the major part of its architecture consists of fully connected layers. The current way our contact maps are generated is used as a fast and simple approximation of the changes happening in the structure of a protein due to amino acid substitutions. The matrix representing the charge interactions does not take into account the protonation state of the amino acids, and the hydrophobicity matrix is only a simple scale and does not take the side chain surroundings into account. These are just two examples where improvements in the protein structure representation can still be made. This might, in turn, help the network to predict mutational effects even better due to a more realistic representation. Data augmentation can be advantageous when used with datasets containing only single and double mutants and network architectures with a smaller number of parameters. When the network architecture with

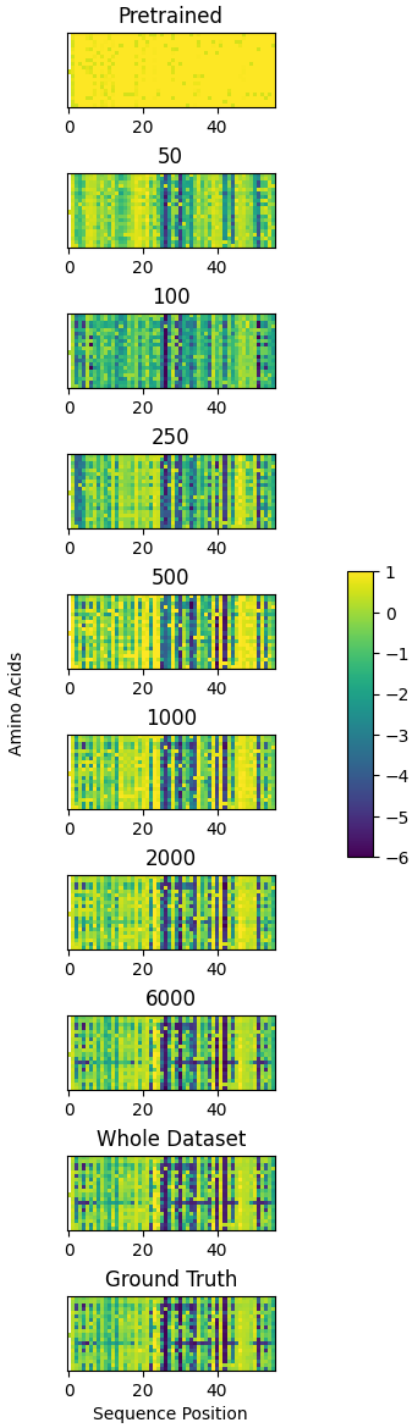

Fig 5. Prediction of the mutational effect of each single mutation at each sequence position using SepConvMixer on the example of GB1. The pre-trained models were trained on training data sets consisting of 50 to 6000 or 80% of the whole GB1 dataset and asked to predict the score of every possible single mutation of GB1. For comparison, the actual measured data are shown as ground truth and the result of a model that was only pre-trained. Figure S14 shows the difference of all predictions to the ground truth. On the y-axis, the amino acids are alphabetically ordered.

the highest number of parameters is used (Simple CNN for avGFP), one can see that this network, when no pre-training is used, over-fits the augmented data. Since this data resembles synthetically generated fitness scores that are not always correct, it has to be used with architectures that use fewer parameters. In contrast, the smallest network (SepConvMixer) still manages to perform decently when only data augmentation is used (Fig. 6). Using different protein sequence alignment databases

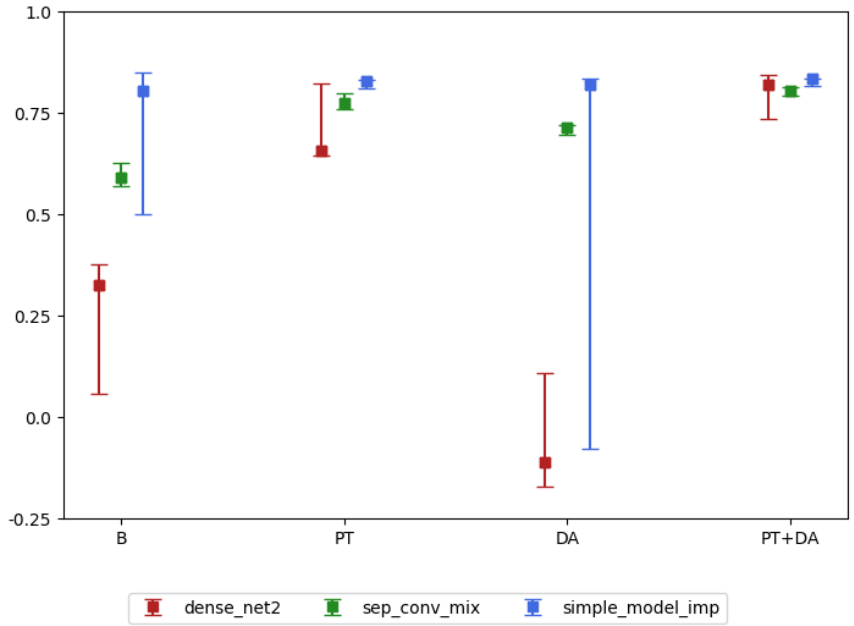

Fig 6. PearsonR of predictions for variants of avGFP containing three to 14 mutations when the networks were only trained on single and double mutants (B: no pre-training and no data augmentation, PT: with pre-training, DA: with data augmentation, PT+DA: with pre-training and data augmentation)

(non-redundant and experimental, which is a 90% clustered version of the non-redundant database) did not change the training results significantly. Interestingly, SepConvMixer performed almost the same as the other architectures despite its much smaller number of parameters. This is promising since fewer parameters reduce the risk of over-fitting. Therefore, the network should be better able to generalise to unseen data. Furthermore, this network will need fewer computational resources. We also showed that more modern network architectures compared to simple CNN could improve the performance when the training sample size is small. Since no dedicated hyper-parameter tuning was performed, an increase in the performance of the models is still possible. A word on protein structure: With the advancement of programs like AlphaFold [29] and RoseTTAFold [30], we can assume that there is a trustworthy structure for most proteins. Even homology modelling might be sufficient to supply a decent protein structure that can be used to create the structure representation. Another advantage of our encoding is the possibility to encode "average" structures derived from molecular dynamics simulations or structures with optimized rotamer positions which leads to an even more natural representation of the protein structure and could potentially create an even better encoding through the interaction matrices. Reducing data sizes necessary to produce comparable or even better results is crucial to reducing time, cost and resources in the wet lab.

Acknowledgement

The authors thank Prof. Thomas Pock and Prof. Robert Kourist for fruitful discussions. Funding was provided by the Austrian Science Fund (FWF) through project DOC-130

(doc.funds BioMolStruct - Biomolecular Structures and Interactions).

References

1. Heim R, Tsien RY. Engineering green fluorescent protein for improved brightness, longer wavelengths and fluorescence resonance energy transfer. Current Biology. 1996;6(2):178–182. doi:https://doi.org/10.1016/S0960-9822(02)00450-5.
2. Binz HK, Amstutz P, Plückthun A. Engineering novel binding proteins from nonimmunoglobulin domains. Nature Biotechnology. 2005;23(10):1257–1268. doi:10.1038/nbt1127.
3. Fowler DM, Fields S. Deep mutational scanning: a new style of protein science. Nature Methods. 2014;11:801–807. doi:https://doi.org/10.1038/nmeth.3027.
4. Araya CL, Fowler DM. Deep mutational scanning: assessing protein function on a massive scale. Trends in Biotechnology. 2011;29(9):435–442. doi:https://doi.org/10.1016/j.tibtech.2011.04.003.
5. Gray VE, Sitko K, Kameni FZN, Williamson M, Stephany JJ, Hasle N, et al. Elucidating the Molecular Determinants of A beta Aggregation with Deep Mutational Scanning. G3 Genes—Genomes—Genetics. 2019;9(11):3683–3689. doi:10.1534/g3.119.400535.
6. Fowler DM, Araya CL, Fleischman SJ, Kellog EH, Stephany JJ, Baker D, et al. High-resolution mapping of protein sequence-function relationships. Nature Methods. 2010;7:741–746. doi:https://doi.org/10.1038/nmeth.1492.
7. Lee JM, Huddleston J, Doud MB, Hooper KA, Wu NC, Bedford T, et al. Deep mutational scanning of hemagglutinin helps predict evolutionary fates of human H3N2 influenza variants. Proceedings of the National Academy of Sciences. 2018;115(35):E8276–E8285. doi:10.1073/pnas.1806133115.
8. Koch P, Schmitt S, Heynisch A, Gumpinger A, Wüthrich I, Gysin M, et al. Optimization of the antimicrobial peptide Bac7 by deep mutational scanning. BMC Biology. 2022;20:114. doi:https://doi.org/10.1186/s12915-022-01304-4.
9. Starr TN, Greaney AJ, Hilton SK, Crawford KHD, Navarro MJ, Bowen JE, et al. Deep mutational scanning of SARS-CoV-2 receptor binding domain reveals constraints on folding and ACE2 binding. bioRxiv. 2020;doi:10.1101/2020.06.17.157982.
10. Frank F, Keen MM, Rao A, Bassit L, Liu X, Bowers HB, et al. Deep mutational scanning identifies SARS-CoV-2 Nucleocapsid escape mutations of currently available rapid antigen tests. Cell. 2022;185(19):3603–3616.e13. doi:https://doi.org/10.1016/j.cell.2022.08.010.
11. Shihab HA, Gough J, Cooper DN, Stenson PD, Barker GLA, Edwards KJ, et al. Predicting the Functional, Molecular, and Phenotypic Consequences of Amino Acid Substitutions using Hidden Markov Models. Human Mutation. 2013;34(1):57–65. doi:https://doi.org/10.1002/humu.22225.
12. Hopf TA, Ingraham JB, Poelwijk FJ, Schärfe CPI, Springer M, Sander C, et al. Mutation effects predicted from sequence co-variation. Nature Biotechnology. 2017;35(2):128–135. doi:10.1038/nbt.3769.

13. Riesselman AJ, Ingraham JB, Marks DS. Deep generative models of genetic variation capture the effects of mutations. Nature Methods. 2018;15(10):816–822. doi:10.1038/s41592-018-0138-4.

14. Biswas S, Khimulya G, Alley EC, Esvelt KM, Church GM. Low-N protein engineering with data-efficient deep learning. Nature Methods. 2021;18(4):389–396. doi:10.1038/s41592-021-01100-y.

15. Rives A, Meier J, Sercu T, Goyal S, Lin Z, Liu J, et al. Biological structure and function emerge from scaling unsupervised learning to 250 million protein sequences. Proceedings of the National Academy of Sciences. 2021;118(15):e2016239118. doi:10.1073/pnas.2016239118.

16. Gray VE, Hause RJ, Luebeck J, Shendure J, Fowler DM. Quantitative Missense Variant Effect Prediction Using Large-Scale Mutagenesis Data. Cell Systems. 2018;6(1):116–124.e3. doi:https://doi.org/10.1016/j.cels.2017.11.003.

17. Romero PA, Krause A, Arnold FH. Navigating the protein fitness landscape with Gaussian processes. Proceedings of the National Academy of Sciences. 2013;110(3):E193–E201. doi:10.1073/pnas.1215251110.

18. Gelman S, Fahlberg SA, Heinzelman P, Romero PA, Gitter A. Neural networks to learn protein sequence-function relationships from deep mutational scanning data. Proceedings of the National Academy of Sciences. 2021;118(48):e2104878118. doi:10.1073/pnas.2104878118.

19. Amidi A, Amidi S, Vlachakis D, Megalooikonomou V, Paragios N, Zacharaki E. EnzyNet: enzyme classification using 3D convolutional neural networks on spatial representation. PeerJ. 2018;doi:10.7717/peerj.4750.

20. Ofer D, Brandes N, Linial M. The language of proteins: NLP, machine learning & protein sequences. Computational and Structural Biotechnology Journal. 2021;19:1750–1758. doi:https://doi.org/10.1016/j.csbj.2021.03.022.

21. Illergård K, Ardell DH, Elofsson A. Structure is three to ten times more conserved than sequence—A study of structural response in protein cores. Proteins: Structure, Function, and Bioinformatics. 2009;77(3):499–508. doi:https://doi.org/10.1002/prot.22458.

22. Lecun Y, Bottou L, Bengio Y, Haffner P. Gradient-based learning applied to document recognition. Proceedings of the IEEE. 1998;86(11):2278–2324. doi:10.1109/5.726791.

23. Huang G, Liu Z, Van Der Maaten L, Weinberger KQ. Densely Connected Convolutional Networks. In: 2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR); 2017. p. 2261–2269.

24. Trockman A, Kolter JZ. Patches Are All You Need? CoRR. 2022;abs/2201.09792.

25. Moreno-Barea FJ, Jerez JM, Franco L. Improving classification accuracy using data augmentation on small data sets. Expert Systems with Applications. 2020;161:113696. doi:https://doi.org/10.1016/j.eswa.2020.113696.

26. Caron M, Bojanowski P, Mairal J, Joulin A. Unsupervised Pre-Training of Image Features on Non-Curated Data. In: Proceedings of the IEEE/CVF International Conference on Computer Vision (ICCV); 2019.

27. Iorga C, Neagoe VE. A Deep CNN Approach with Transfer Learning for Image Recognition. In: 2019 11th International Conference on Electronics, Computers and Artificial Intelligence (ECAI); 2019. p. 1–6.

28. Berman H, Henrick K, Nakamura H. Announcing the worldwide Protein Data Bank. Nature Structural & Molecular Biology. 2003;10(12):980–980. doi:10.1038/nsb1203-980.

29. Jumper J, Evans R, Pritzel A, Green T, Figurnov M, Ronneberger O, et al. Highly accurate protein structure prediction with AlphaFold. nature. 2021;589:583–589. doi:10.1038/s41586-021-03819-2.

30. Baek M, DiMaio F, Anishchenko I, Dauparas J, Ovchinnikov S, Lee GR, et al. Accurate prediction of protein structures and interactions using a three-track neural network. Science. 2021;373(6557):871–876. doi:10.1126/science.abj8754.

31. Griffith D. PARROT: Protein Analysis using RecuRrent neural networks On Training data; 2020. Available from: `https://github.com/idptools/parrot/blob/master/parrot/encode_sequence.py`.

32. Schrödinger, LLC. The PyMOL Molecular Graphics System, Version 1.8; 2015.

33. Johnson M, Zaretskaya I, Raytselis Y, Merezhuk Y, McGinnis S, Madden TL. NCBI BLAST: a better web interface. Nucleic Acids Research. 2008;.

34. McWilliam H, Li W, Uludag M, Squizzato S, Park YM, Buso N, et al. Analysis Tool Web Services from the EMBL-EBI. Nucleic Acids Research. 2013;.

35. Gelman S, Fahlberg SA, Heinzelman P, Romero PA, Gitter A. Neural networks for deep mutational scanning data; 2021. Available from: `https://github.com/gitter-lab/nn4dms`.