# Uncertainty aware Protein-Structure Prediction

Shubham Khatri, Ghalia Rehawi, Mayuran Surendran

January 26, 2022

## 1 Introduction

Protein structure prediction is the process of predicting the three-dimensional shape using the protein amino acid sequence itself. This task is of high importance because structural features can shed light on biological functions of given proteins. Deep Neural Networks have shown great success in predicting protein structures as proven by DeepMind's AlphaFold that won the latest Critical Assessment of Structure Prediction competition (CASP13) [1]. Alphafold uses neural networks in order to predict the backbone torsion angles and pairwise distances between amino acid residuals. The predictions are combined together to form a potential which is used to predict the final tertiary structure of the given protein.

This report is structured as follows. In section 2, we lay out the purpose of our work and research. After that, we describe the dataset which was used for training in section 3. In section 4, we present details on how we pre-processed the data in order to prepare it for training the distance prediction model. Subsequently, section 5 offers insight into the data pipeline used during training whereas section 6 explains and compares the architecture of our model with the architecture of AlphaFold. Sections 7 and 8 describe the training of our model and the experiments conducted in parallel to the training. After that, section 9 describes the implemented testing pipeline and interprets the performance and uncertainty quantification metrics obtained from the model prediction. Afterwards, we argue in section 10 why uncertainty awareness is needed in modern neural networks and introduce two common methods to raise the uncertainty awareness of our model. Moreover, we analyze and interpret the outcome of these methods critically. Finally, in section 11 we present various ideas and recommendations that can be made to increase the performance and uncertainty awareness of the model.

## 2 Purpose

In this work, we use DeepMind's AlphaFold as baseline model. However, we only focus on predicting the pairwise distances of amino acids using both primary features which are the sequences of proteins themselves and evolutionary features

extracted from Multiple Sequence Alignments. Our model, similar to AlphaFold, predicts the distograms which are discretized distances. We only consider distances between 2 and 22 Ångstroms, because this range of distances is further used to predict contacts between two amino acid. Herefore, we use a deep two-dimensional dilated convolutional network with a variable number of residual block groups.

Besides the accuracy of the model, it is often important to know how confident a neural network is in its prediction. However, modern neural networks tend to be poorly calibrated compared to the ones from a decade ago. Therefore, we use two uncertainty quantification methods to raise the uncertainty awareness of the model.

# 3 ProteinNet

ProteinNet is an open-source standardized dataset for training and assessing data-driven models of protein sequence-structure relationships. ProteinNet encompasses sequence, structure and evolutionary information available both in text as well as TFRecords. The dataset also provides a training, validation, testing subset that takes care of the non-independent and non-identically distributed data; a problem that comes from the inherent evolutionary relationships between protein. ProteinNet was created by piggybacking on the CASP series of assessments to create a corresponding series of datasets in which the testing set contains all structures released in a given CASP, and the training set is comprised of all protein structures and sequences publicly available prior to the start date of the respective CASP challenge. The proteins that are provided by the CASP challenge and that are used as testing set in ProteinNet, are divided into multiple categories including template-based modeling (TBM) which contains proteins that are easy to predict and free modeling (FM) that contains proteins which are hard to predict. This categorization provides a measure of difficulty useful for assessing the model's ability to generalize to unseen parts of fold space.

In order to lower the entry barrier and to save significant time on data collection and standardization we use ProteinNet for training, validation and testing purposes.

# 4 Data Preprocessing

**Tertiary structure** As ground truth for our model we use the tertiary information provided by ProteinNet, which is the three-dimensional atomic representation of a protein. This tertiary information is represented as 3D Cartesian coordinates of the backbone atoms N, $C_\alpha$ and C' for each amino acid in the protein sequence. In our implementation, we have calculated the pairwise distances of amino acids based only on the corresponding distance between their $C_\alpha$ atoms. From the resulting distance map of each protein, we further create the distograms by discretizing the inter-residual distance using 64 bins with the

lower bound of the first bin equaling 2 and the upper bound of the $64^{\text{th}}$ bin equaling 22 Ångstroms.

**Primary Features** This feature constitutes one type of input feature for our model and reflects the sequence of amino acids for each protein. Since our model uses a series of convolutional layers, we need to transform the amino acid sequence into the correct shape. Supposing that our our sequence has the length L, the input to the convolution in this case should be of the shape L x L x N, where N = 20, and stands for the one-hot encoded vector for each amino acid. Figure 1 depicts such a matrix.

**Evolutionary Features** We use the PSSM (Position-Specific Scoring Matrices), which are also available in ProteinNet, as input features. For each protein sequence, the PSSM is represented by a 20 dimensional real-valued vector (one dimension for each amino acid, ordered alphabetically). The values reflect the conservation of each amino acid in the sequence i.e. these values reflect the likelihood of substituting an amino acid of the protein with another amino acid while conserving the function of that protein. The information content of each residue of a protein is added as an extra dimension to the PSSM inflating it to a total size of L x 21, where L is the sequence length.

**Masks** Mask are one-bit indicators stating whether the atomic coordinate of an amino acid is available or not. Masks play an essential role during training since we want to avoid the loss function penalizing predictions that were made for unknown atomic positions. We pre-process the masks provided by the ProteinNet to get it into the shape L x L reflecting the pairwise atomic position availability.

| | A | B | A | C |
|---|---|---|---|---|
| A | [1 0 0 0 0 0 0 0 0 0 ..] | [0 0 0 0 0 0 0 0 0 0 ..] | [1 0 0 0 0 0 0 0 0 0 ..] | [0 0 0 0 0 0 0 0 0 0 ..] |
| B | [0 0 0 0 0 0 0 0 0 0 ..] | [0 1 0 0 0 0 0 0 0 0 ..] | [0 0 0 0 0 0 0 0 0 0 ..] | [0 0 0 0 0 0 0 0 0 0 ..] |
| A | [1 0 0 0 0 0 0 0 0 0 ..] | [0 0 0 0 0 0 0 0 0 0 ..] | [1 0 0 0 0 0 0 0 0 0 ..] | [0 0 0 0 0 0 0 0 0 0 ..] |
| C | [0 0 0 0 0 0 0 0 0 0 ..] | [0 0 0 0 0 0 0 0 0 0 ..] | [0 0 0 0 0 0 0 0 0 0 ..] | [0 0 1 0 0 0 0 0 0 0 ..] |

Figure 1: This example shows how we process an input sequence ABAC to obtain the one-hot vector encoding of its amino acids ready for input into the model. Each cell corresponds to the one-hot vector of the amino acid only if both the row and the column amino acids are the same.

# 5 Data Pipeline

**Extracting Crops** Since we need a fixed size of input samples for our training and because protein sequences have different lengths, we crop a fixed sized window of C x C from each protein sequence at a randomly computed index value. We further proceed to extract the corresponding features for the C x C window of amino acids. In case the sequence length is smaller than the chosen crop size (C = 64 in our case) we pad with zeros. The final shape for our input data is now C x C x C where the first two entries correspond to the window size and the third one corresponds to the number of bins. The entire data pipeline for training is depicted in Figure 2.

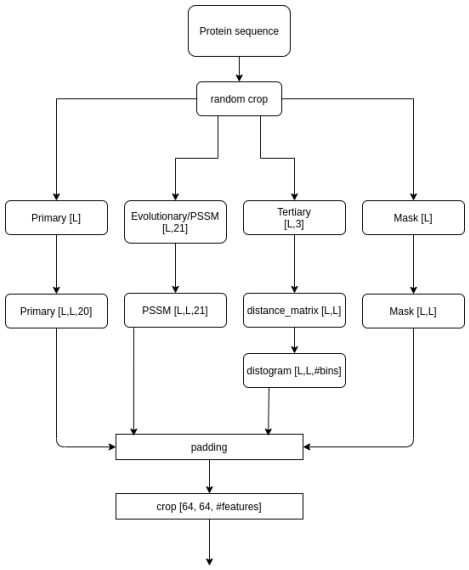



Figure 2: Input data pipeline for training

**Class Imbalance** During the exploratory data analysis, we find that a high percentage of inter-residual distances are larger than 22 Ångstroms.The distribution of distances for the CASP7 subset of the training set can be seen in Figure 3. During the data preprocessing step, all these distances are mapped to the last bin as we are only interested in the range of distances between 2 and 22 Ångstroms. Therefore, our data suffers from an over-representation of the $64^{\text{th}}$ class corresponding to 22 Ångstroms. In section 6, we will discuss how to address this issue.

## 5.1 Training Pipeline

The data preprocessing steps defined in section 4 and the crop extraction method defined above was combined and put together in a data pipeline. This was

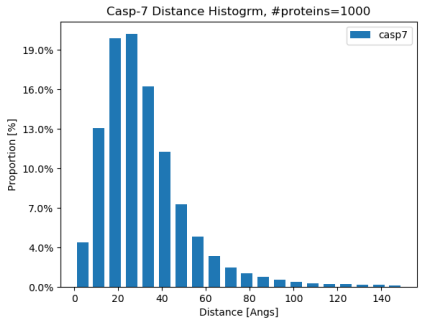



Figure 3: Distance histogram for number of samples in CASP7 training set

done using TF Dataset function alongside with python generators. The main reason behind developing such a pipeline was to tackle the memory issue that arises with the large dataset that we are handling. With the TF Dataset we load the data and perform the preprocessing steps just in time when they were needed, thus avoiding the problem of loading the entire data in the memory. This data pipeline was also optimized by enhancing the parallel processing in the preprocessing steps as well as by the usage of prefetch which allows the GPU to call for preparation of the next batch on CPU while the computation for this batch is being performed. We tested this implementation on a machine with 8GB of RAM and 4 logical cores, we observed with usage of appropriate batch size it was possible to sweep through the entire training set without any issues. We also experimented training data sweeps on other machines with relatively low memory and we observed that the data pipeline was able to perform optimally with the provided resources.

## 5.2 Validation Pipeline

Unlike the training data the ProteinNet validation data is relatively small and contains about 90 samples, each of different lengths. We perform data augmentation by randomly cropping the protein samples such that the number of validation sample is about the user provided ratio of training set. For example if the total number of samples in the training set is 32,000 and the user provided validation ratio is 0.2 then the number of validation samples is about 6400. We also take care that the augmented data is unique as well as the augmented data doesn't change over the epochs

Since augmenting the data to populate the validation set would also ramp up the memory consumption and it quickly becomes unmanageable for the available memory, therefore we also develop a TF dataset pipeline for the validation set. The validation set data pipeline is constructed in following way:

- Compute the number of validation sample available ($N_{avail}$).
- Compute the number of samples which are smaller than crop size ($N_{small}$).

- Compute the desired number of validation samples ($N_{val}$).
- Compute the number of sweeps over validation dataset using following formula

$$N_{sweep} = \left\lceil \frac{N_{val}}{N_{avail} - N_{small}} \right\rceil + 1 \tag{1}$$

  The +1 is to account for the first pass over the data since some of the protein length might be less than the crop size and therefore the number of available samples from the second pass would be less than that of the first pass.
- For the first sweep, generate a random ($x$, $y$) index pair and add it to crop_index list.
- For each subsequent data sweep, if the protein length is larger than the crop size, then generate a random ($x$, $y$) index pair, and if this index pair does not lies in the sublist of indices corresponding to this protein add it to the crop_index list. Else skip the protein.
- After the $N_{sweep}$ we have a crop_index list of size $N_{val}$ which contains the cropping position corresponding to each protein over the $N_{sweeps}$.
- This list is stored as a Numpy array and repeated as many times as the number of epochs for training.
- When a validation sample is requested from the pipeline during the model evaluation, the index pair at the top of the list is popped and the protein is used as the start position of cropping for the preprocessing.

In order to make it scalable and allow multiprocessing during preprocessing a thread safe list access was implemented. This allows only one thread at a time to pop the index pair corresponding to its protein. Once the index pair is received these threads can execute the preprocessing steps in parallel.

Using this validation pipeline we ensure that the validation set remains the same across the epochs and we have sufficient number of validation samples to evaluate the performance of the model. It also provides an efficient and scalable pipeline which can also function on machines with low memory.

# 6 Model Architecture

In this section, we lay out the model architecture that was used to conduct experiments. The model named MiniFold [2] is inspired by the original distance prediction network used in DeepMind's AlphaFold. Similar to AlphaFold, MiniFold is a deep two-dimensional dilated convolutional network with a variable number of residual block groups. A group always includes four residual blocks, because successive residual blocks in a group cycle through dilations of 1, 2, 4, 8 pixels to allow propagation of information quickly across the cropped region [3]. We use the architecture of MiniFold as our baseline architecture.

AlphaFold uses a large architecture with a total number of 220 residual blocks. It has 7 and 48 groups of 256 and 128 channels, respectively. Based on a preliminary analysis with this architecture we found out that it requires about 16 GB of GPU memory to store the weights of the model. Since we were constrained by the available memory and computational power, we had to restrict the maximum number of residual blocks that can be used during training, thus, we adopt the same number of residual blocks as MiniFold.

In contrast to the AlphaFold model, which has over 600 input channels resulting from selection of several features, our model takes an input of 41 channels, when using both primary and evolutionary features, resulting from our choice of dataset, ProteinNet, and the preprocessing steps performed, as described in Section 4. Furthermore, our model does not use the position-specific bias and the auxilliary loss weights that are used by AlphaFold. Moreover, we use Adam instead of Stochastic Gradient Descent for optimization of the model. Figure 4 shows the overall architecture based on the block shown in the Figure 4.

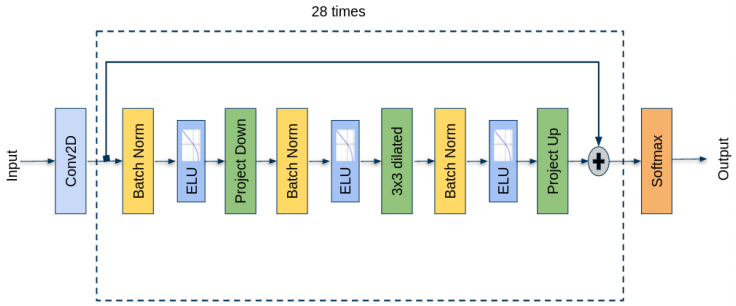



Figure 4: Final model architecture

In order to address the class imbalance problem described in section 5, we introduce class weights during the training. The class weights are calculated by iterating through the validation set and counting the number of occurrences for each of the 64 labels. With the help of these class weights we were able to partially compensate for the misclassification of under-represented classes. Section 9 highlights the differences in the outcome when using class weights as opposed to without it.

The final model parameters used for training are summarized in Table 1.

# 7 Training

The implementation of the model shown in Figure 4 was done using the Tensorflow Keras API. Each of the residual blocks shown in the Figure 4 was followed by a dropout layer using dropout rate mentioned in Table 1. In addition to this, each convolution layer was initialized using He-Initialization, since it has been shown to perform best with ReLU activation [4]. Additionally, kernel regularization

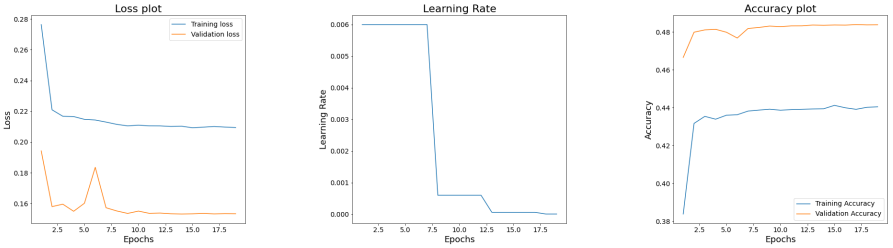
| Hyperparameter | Value |
|---|---|
| Number of blocks and channels | [28, 64] |
| Number of features | 41 |
| Dropout rate | 0.1 |
| Learning rate | 6e-2 with learning rate reduction |
| Kernel initialization | He-Initialization |
| Kernel regularizer | 1e-4 |

Table 1: Hyperparameters of our model

was used in each convolution layer to increase the generalization capability of the model. A learning rate of 0.06 was taken similar to that of AlphaFold. Furthermore, a learning rate reduction scheme was employed which reduce the learning rate at the plateaus.

In addition to the above described model features, one of the key component which was handled during training was the masking tensor. As described in the section 4 the masking tensor describes the reliability of the measurement at a position in the protein and therefore they could be considered similar to sample weights and were incorporated in the computation of loss during the training of the model. We achieved this by passing in a flattened masking tensor from the data generator, since Keras API is limited to 1D sample weights, and then reshaping it to appropriate shape inside the training step. The same logic was used to incorporate the class weights at a later stage to tackle the problem of severe class imbalance described in section 5.

Figure 5 shows the metrics measured during the training of a model. It can be observed that the training and the validation loss strongly drop in the first three epochs, accompanied by a significant increase in the accuracy. Since we employ the learning rate reduction on the validation loss curve plateau we also observe a reduction of learning rate around $7^{\text{th}}$ and $12^{\text{th}}$ epoch where validation loss plateaus.




(a) Training loss plot (b) Learning rate plot (c) Accuracy plot

Figure 5: Training and validation set losses and accuracy, as well as learning rate during the training of model using class weights

# 8 Experiments

In the scope of this work, we conduct various experiments which are outlined in the following paragraphs.

**Overfitting the model** In order to inspect the correctness of the model implementation as described in the section 6, we start with an implementation where we turn off all the regularizers in the model, i.e, we set the dropout rate and the kernel regularization strength equal to 0, and use a random uniform kernel initialization. We call this model Overfitting Model. We test whether the model can overfit on the training data. Figure 6 depicts the loss curve obtained by overfitting the model. We observe that the training process is not stopped early and the training loss is dropping epoch by epoch reaching lower values than the ones reached by the models using dropout and kernel regularization. section 9 compares the performance of this model to the weighted and unweighted model, described below. It must be noted that this implementation was done for sanity check.

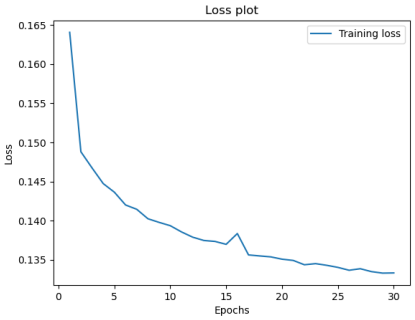



Figure 6: Loss for overfitting model

**Unweighted model** Based on the description provided in section 6, the Unweighted Model is the first implementation. This is an extension of the Overfitting Model that uses the dropout and regularization. The parameters used for this model are the ones described in the Table 1. This model does not deal with the issue arising from the class imbalance. Hence, there are no class weights used for the training of this model.

**Increasing model capacity** In the following step, we experimented with model architectures with higher capacity than the one detailed in Table 1. However, the outcome of the training was not successful as the model overfitted to certain crops consisting of sparse regions i.e. the distance between the given amino acids were almost equal to or larger than 22 Ångstroms. Thus, the model was always predicting the $64^{\text{th}}$ class, which corresponds to this specfic distance,

for any type of crop. This result obtained by using models with higher capacity was the first indicator, that told us that we were facing a class imbalance issue in our dataset, which was described in section 6. Therefore, a confusion matrix obtained from predictions on the testing set was used to inspect this issue further on resulting in proving this hypothesis. Consequently, future models used class weights during to account for this issue.

**Weighted Model** The Weighted Model is an extension of the Unweighted Model. The key difference between the Weighted and the Unweighted Model is that the class weights were introduced to account for the class imbalance problem. The computation of these class weights were done using the method described in section 6.

**Training on CASP11** We tried training our model using CASP11, however due to unknown reasons the training got stopped on the server.

# 9 Results

**Test Pipeline** During testing, we extract non-overlapping 64 x 64 crops (with stride equal to 64) from each protein in the test set. For each one of these crops we extract the primary, evolutionary, tertiary and masking information and create the final test set for the input as well as the ground truth. To evaluate our model's performance on the test set we follow two approaches. First, we use the Scikit-Learn functions named precision, recall, F1 and accuracy-score in a pixel-wise manner on every predicted sample, i.e we calculate the corresponding distograms for each predicted sample of the shape (64 x 64 x 64) [5]. For each pixel we formulate the new y_predict which is now a vector of ones and zeros instead of probabilities. Second, we implement the accuracy, precision, recall and F1-score based on the contact map. Two amino acids are defined to be in contact, if the inter-residual distance is between 2 and 8 Ångstroms. Thus, using this approach we calculate the corresponding contact map for each predicted sample of the shape (64 x 64 x 64) by summing up all the predicted probabilities of the bins that corresponds to bins representing the distance range of 2 to 8 Ångstroms. We predict that the pair of amino acids is in contact, if the sum of the probability masses of these bins is bigger than 0.5. To calculate contact maps from the ground truth data of the shape 64 x 64 x 64, in which the third dimension in this case represents a distogram of the two AA and not probabilities, we transform the sample back into distances and from there we calculate the contact map. Furthermore, we build the test pipeline as a command line tool which makes it easier for the user to do predictions on different CASP sets but also on different categories such TBM, FM, TBM-Hard or all combined. We also extend this evaluation tool with options to let the user choose whether to apply uncertainty quantification methods or not.

**Test Results** Tables 2 and 3 contain the performance metrics of the model on CASP7 to CASP10 based on the contact map and the distogram, respectively. It can be seen that the F1-Score calculated for the contact map drops for all models when using a larger and more diverse dataset. This is to be expected, because the models are not predicting accurately for datasets which they have not been trained on. However, the F1-Score calculated on the distogram rises for all models when using larger and more diverse dataset such as CASP10. This is due to the fact that the test sets of larger datasets have more samples and has a higher mean inter-residual distance. Therefore, those datasets have a higher proportion of inter-residual distances higher than 22 Ångstroms that corresponds to the $64^{\text{th}}$ class. The F1-Score is higher for these datasets, because the model still shows a bias towards this class.

Table 4 sheds light on the uncertainty awareness of the models. It can be seen that compared to the unweighted model the uncertainty of the weighted model is in average higher across all datasets. Therefore, we can conclude that the introduction of class weights during training yields a model which is more certain in its predictions.

Figure 7 shows the confusion matrix on CASP7 test dataset. Even though these results are on the weighted model, there is still a considerable misclassification and bias towards the $64^{\text{th}}$ class. This could be due to the fact that the class weights were calibrated on the validation set and we anticipate a different results on the training set.

| Model | Dataset | Accuracy | Precision | Recall | F1-Score |
|---|---|---|---|---|---|
| Weighted | CASP7 | 98.05 | 62.51 | 22.23 | 32.80 |
| | CASP8 | 98.14 | 62.29 | 20.51 | 30.86 |
| | CASP9 | 98.31 | 59.86 | 18.13 | 27.83 |
| | CASP10 | 98.17 | 57.04 | 15.56 | 24.45 |
| Unweighted | CASP7 | 98.14 | 70.22 | 22.18 | 33.71 |
| | CASP8 | 98.21 | 71.18 | 20.39 | 31.70 |
| | CASP9 | 98.39 | 72.16 | 17.57 | 28.26 |
| | CASP10 | 98.31 | 72.88 | 15.35 | 25.36 |

Table 2: Performance metrics based on contact map in %

| Model | Dataset | Accuracy | Precision | Recall | F1-Score |
|---|---|---|---|---|---|
| Weighted | CASP7 | 57.28 | 41.19 | 57.28 | 46.81 |
| | CASP8 | 59.47 | 44.05 | 59.47 | 49.49 |
| | CASP9 | 64.17 | 49.91 | 64.19 | 55.07 |
| | CASP10 | 63.92 | 50.37 | 63.92 | 55.13 |
| Unweighted | CASP7 | 57.45 | 41.49 | 57.45 | 47.09 |
| | CASP8 | 59.65 | 44.21 | 59.65 | 49.70 |
| | CASP9 | 64.39 | 50.00 | 64.38 | 55.20 |
| | CASP10 | 64.50 | 50.47 | 64.50 | 55.43 |

Table 3: Performance metrics based on distogram in %

| Model | Dataset | ECE | Entropy |
|---|---|---|---|
| Weighted | CASP7 | 0.0624 | 2.5492 |
| | CASP8 | 0.0597 | 2.9860 |
| | CASP9 | 0.0591 | 2.5235 |
| | CASP10 | 0.0641 | 2.8849 |
| Unweighted | CASP7 | 0.0554 | 3.2118 |
| | CASP8 | 0.0528 | 3.0016 |
| | CASP9 | 0.0651 | 2.2177 |
| | CASP10 | 0.0531 | 3.0792 |

Table 4: Uncertainty quantification metrics

**Model Bench-marking** In order to benchmark our model, we compare the contact precision of our predictions to those of AlphaFold and ProSPr [6]. However, it is important to note that this comparison has no solid foundation since both ProSPr and Alphafold were trained on CATH dataset [7] and tested on CASP13, whereas our model was trained and tested using CASP7 provided by ProteinNet. Therefore in Table 5 we report the contact precision on CASP10 dataset since it's closer in difficulty to CASP13 used by Alphafold and ProSPr. Furthermore, it should be mentioned that their cropping procedure might be different to ours, thus, making it very difficult to compare our model to the other two benchmark models

| | AlphaFold | ProSPr | Our Model |
|---|---|---|---|
| Precision | 0.57 | 0.66 | 0.57 |

Table 5: Comparison of contact precision of the three respective models

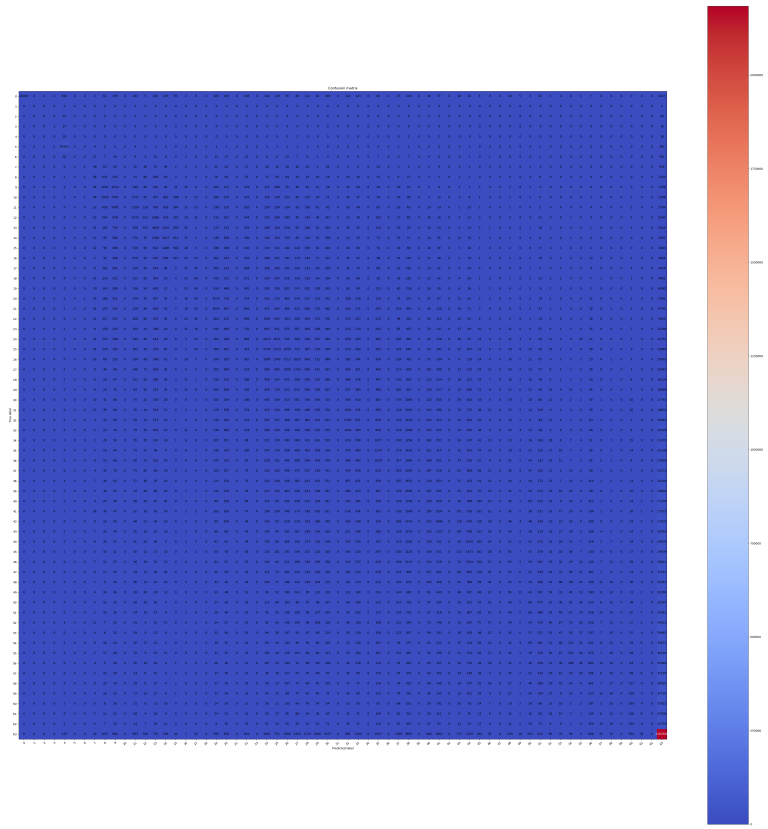


Figure 7: Confusion matrix for CASP7 dataset

# 10 Uncertainty Awareness

In the past years, the performance and accuracy of neural networks has improved drastically. Therefore, they are often part of larger decision making processes entrusted with important decisions. Consequently, a neural network should be able to indicate whether it is confident in its prediction.

In the scope of this work, we use two state-of-the-art algorithms to introduce uncertainty awareness to the distance prediction model. The first algorithm is called Temperature Scaling, which calibrates the prediction of the network using a single parameter. The second algorithm is called Monte-Carlo Dropout which is equivalent to modeling Deep Gaussian Processes.

## 10.1 Temperature Scaling

**Miscalibration of modern neural networks** Guo et al. discover that modern neural networks such as ResNet tend to be less calibrated than models like LeNet i.e. the average accuracy of the model's prediction is much higher

than its average confidence [8] . This phenomenon can be observed in the first row in Figure 8. The second row depicts reliability diagrams. Here, samples are grouped into bins based on their respective prediction confidence by the neural network. For each bin, the average accuracy and average confidence is calculated and plotted along the y-axis. The gaps between both quantities corresponds to the miscalibration of the network. A perfectly calibrated model plots the identity function. Through extensive research, Guo et al. identify that recent advancements in model capacity and the use of batch normalization and weight decay are responsible for the miscalibration of modern neural networks.

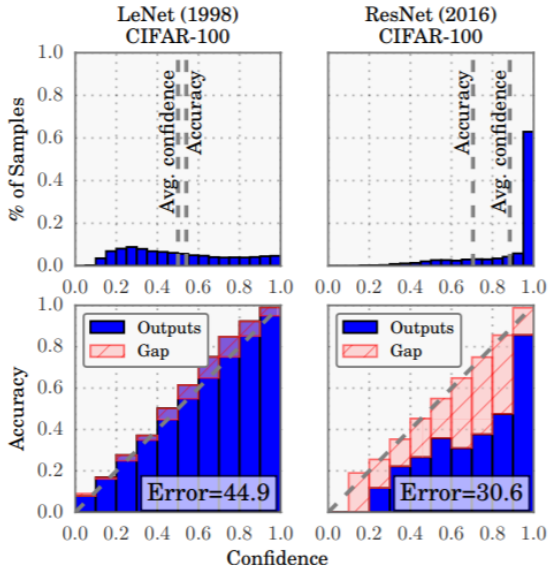



Figure 8: Confidence histograms (top) and reliability diagrams (bottom)

**Expected Calibration Error** Although reliability diagrams have proven to be very useful, it is more convenient to represent the miscalibration of a model using a single scalar. Naeini et al. introduce the Expected Calibration Error (ECE) which approximates the difference in expectation between accuracy and confidence by dividing predictions into $M$ bins. The difference of expectation for each bin is weighted by ratio between the number of predictions of the particular bin and the total number of predictions [9]. This procedure is summarized by the following equation:

$$ECE = \sum_{m=1}^{M} \frac{|B_m|}{n} |acc(B_m) - conf(B_m)| \tag{2}$$

**Algorithm** Temperature Scaling introduces a single scalar $T > 0$ to scale the logit vector $z_i$. This procedure yields a new confidence prediction $\hat{q}_i$ which is given by

$$\hat{q}_i = \max_k \sigma_{sm}\big(z_i/T\big)^{(k)} \tag{3}$$

where $\sigma_{sm}$ is the softmax function. The predictions by the model are not affected by the scaling, thus, the accuracy of the model stays the same.

The temperature is learned on the validation set by minimizing the Negative Log Likelihood.

**Application** In order to increase the uncertainty awareness of the weighted model, we scale the logits by the learned temperature $T$. In this case however, we do not predict one class for each image or crop that is passed through the network, which is the case for models trained on datasets such as MNIST. Here, we deal with pixel-wise classification because we classify the inter-residual distance for each pair of amino-acids. Hence, we need to scale the logits of each pixel by the learned temperature $T$.

We iterate through the validation set once and learn the temperature using batches consisting of 16 crops and for each given batch we perform three optimization steps. Higher batch sizes helps lower the variance during the optimization steps. The ECE as well as entropies before and after applying Temperature Scaling averaged over three runs are given in Table 6.

| Metric | TBM | FM | TBM-Hard | All |
|---|---|---|---|---|
| ECE | 0.0646 | 0.0679 | 0.0583 | 0.0624 |
| TS-ECE | 0.0513 | 0.0808 | 0.0846 | 0.0554 |
| Entropy | 2.5248 | 2.5492 | 2.6159 | 2.5491 |
| TS-Entropy | 2.6533 | 2.6888 | 2.7288 | 2.6888 |
| $N_{crops}$ | 770 | 53 | 111 | 934 |

Table 6: Uncertainty metrics after Temperature Scaling with $T = 1.071277$

It can be seen that the tests using proteins contained in the categories FM and TBM-Hard produce the initial highest entropy value. This outcome was expected as the proteins like the ones contained in these categories are barely represented in the training set. Therefore, the model is more uncertain about distance predictions for proteins contained in those categories. Furthermore, applying temperature scaling on samples contained in categories FM and TBM-Hard increases the ECE. This might be due to the fact that samples like those are not represented enough in the validation set, which was used to learn the temperature. In contrary, the ECE was decreased for test samples contained in the category TBM; similar samples are also represented in the validation set. Applying Temperature Scaling on a testing set containing samples from all categories decreases the ECE. Moreover, the entropy has been increased for all

categories by the application of Temperature Scaling. Thus, it can be concluded that the usage of this uncertainty quantification method is successful. Figure 9 summarizes this conclusion as a reliability diagram.

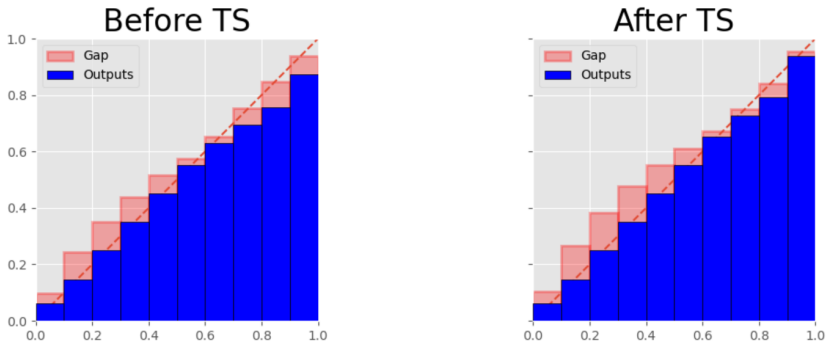



Figure 9: Reliability diagram for CASP7 testing set

## 10.2 Monte-Carlo Dropout

The Monte-Carlo Dropout method (MC Dropout) builds upon the idea of Bayesian networks which provides the flexibility to compute model uncertainity. Yarin Gal, and Zoubin Ghahramani showed in their work, Dropout as a Bayesian Approximation: Representing Model Uncertainty in Deep Learning, how introduction of a dropout layer prior to every weight layer can be treated as a Bayesian approximation of the Gaussian process [10].

### 10.2.1 Theoretical Background

The Monte-Carlo (MC) estimation of an input $x^*$ is given by following equation

$$\mathbf{y^*_{mc}} = \frac{1}{T}\sum_{k=1}^{k=T} \mathbf{y^*_k} \tag{4}$$

where

- $\mathbf{y^*_{mc}}$ is the monte carlo estimate,
- $\mathbf{y^*_k}$ is $\mathrm{k}^{\mathrm{th}}$ sample with random dropout application.
- $T$ is the number of MC sampling.

In order to quantify the confidence of the model on a given input we also look into second moment of the estimation, which is equivalent to looking at the standard deviation of the estimations, $\mathbf{y^*_k}$. The following equation describes the second moment of the estimate.

$$\mathbf{\xi_1^2} = \frac{1}{T}\sum_{k=1}^{k=T} (\mathbf{y^*_k} - \mathbf{y^*_{mc}})^2 \tag{5}$$

where

- $\xi_{\mathbf{1}}$ is the second moment of the estimation

Additionally, to improve upon the uncertainty measure we also estimate the model inherent noise described in the article Engineering Uncertainty Estimation in Neural Networks for Time Series Prediction at Uber[11]. The following equation describes the formulation for inherent noise computation.

Let $X = (x_1, \cdots, x_T)$, $y = y_1, \cdots, y_T$ be the independent test set and the NN be represented by a function $f$, then

$$\xi_{\mathbf{2}}^{\mathbf{2}} = \frac{1}{T} \sum_{k=1}^{k=T} (\mathbf{y_k} - \mathbf{f}(\mathbf{x_k}))^2 \tag{6}$$

where

- $\xi_{\mathbf{2}}$ is the inherent model noise.

Using $\xi_{\mathbf{1}}$ and $\xi_{\mathbf{2}}$ we can define overall model uncertainty measure, $\psi$, as follows

$$\psi = \sqrt{{\xi_{\mathbf{1}}}^2 + {\xi_{\mathbf{2}}}^2} \tag{7}$$

### 10.2.2 Application

The MC Dropout implementation does not modify the training method of the model but only deals with modification of prediction method. In general, the dropout layers are activated during the train time to regularize the model and prevent overfitting but as described above the application of dropout during the estimation is equivalent to GP sampling for a given input. To achieve this we modify the "test_step" method in the Keras API model definition and set the train parameter to True during the prediction phase which makes the model believe that the estimation are being done in the training phase and leads to application of dropout layer during the prediction. Since the dropout layer is applied randomly we perform several prediction on the same sample. These samples are then used to compute the MC estimation, $\mathbf{y^*_{mc}}$, and the model uncertainty, $\psi$.

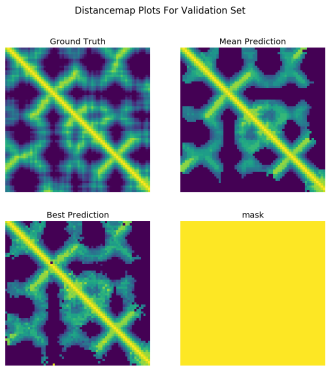
### 10.2.3 Results

We performed the MC Dropout estimation on the CASP7 test data of ProteinNet using the weighted model. We compute the standard model evaluation metrics for each of the category namely, TBM, TBM-Hard, FM and All, using MC Dropout estimates. We also compute the model uncertainty on each of these samples and aggregate it using the formulation described in Equation 5, Equation 7 and Equation 6. Table 7 shows the result obtained on the test set for 500 sampling per data points. We observe that the model entropy computed on the mean prediction increases as compared to that shown in Table 4, this results from a higher uncertainty aware model. Additionally we also observe that the FM

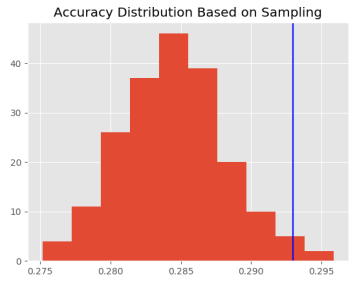
samples have highest entropy as expected. A counter intuitive observation is that the model uncertainty ($\psi$) is lowest for the FM, which was expected to be highest. This results from a relatively low number of FM and TBM-Hard samples availability in the test set which makes this measure a bit unreliable. The bias mainly results from the inherent model noise which is dependent on number of samples in the test set and when a small number of samples are available for testing these measures can quickly become unreliable and therefore this test could be performed on validation set for a better view of the model. We also observe that the overall performance of the model decreases as compared to that of the first model, since the model is evaluated on the MC prediction which is mean of all the prediction (Maximum likelihood estimate of a Gaussian) and therefore represents a sampling from a point based Gaussian Process.

| Metric | TBM | FM | TBM-Hard | All |
|---|---|---|---|---|
| Entropy | 2.5788 | 2.9303 | 2.7432 | 2.9256 |
| Model Uncertainty($\psi$) | $6.6413\ 10^{-5}$ | $1.0177\ 10^{-7}$ | $2.8128\ 10^{-5}$ | $8.6878\ 10^{-5}$ |
| Accuracy (%) | 56.92 | 57.66 | 63.91 | 57.79 |
| Precision (%) | 39.76 | 40.71 | 47.84 | 40.76 |

Table 7: Model performance metrics on distogram with 500 MC Dropout sampling




(a) Distogram plot comparing best prediction, mean prediction and ground truth. The yellow color in the mask depicts a reliable region of inspection.




(b) Sample accuracy plot on distogram. The vertical line depicts the accuracy of the MC mean prediction sample.

Figure 10: MC Dropout evaluation on a sample from the test set.

Figure 10 shows the model evaluation using MC Dropout at one of the test samples. We can observe in the Figure 10a that the predicted off diagonal distances get lower in the value (smoother color transition in the distancemaps) and thereby compromising the accuracy of the the mean prediction, which can be

observed by the gap between the accuracy of the mean prediction and the best prediction, shown in Figure 10b. It can also be observed that the mean prediction accuracy is higher than that of mean accuracy of the samples. This is because we have a multi-input multi-output model, which means the sample accuracy is computed by aggregating all the predictions made for one sample. Therefore we can infer that the our model's prediction of the off-diagonal structure shown in the mean prediction is with high confidence which account for the gap between the mean accuracy of samples and accuracy of the mean prediction.

# 11 Future Steps

In the later stages of our work, we came up with various recommendations and ideas for the future. These will be outlined in the following paragraphs.

**Dealing with higher inter-residual distances** As mentioned in section 5, the dataset suffers from a class imbalance issue which can be attributed to our data preprocessing. Here, we assign inter-residual distances of equal to or larger than 22 Ångstroms to the $64^{\text{th}}$ bin, which results in a class imbalance issue in the dataset. Therefore, another process should be used to deal with these inter-residual distances. For example, protein sequences with amino sequence length higher than a certain threshold can be dropped to keep the number amino acid pairs with very high inter-residual distances to a minimum. Another way would be to modify the masking tensor such that predictions of these certain pixels are not taking into account during training. Also, the class weights could be tuned by iterating several times through the training set rather than iterating once through the validation set.

**Cropping procedure** In the scope of this work, we use random indices to perform the cropping. However, we suspect that problems such as overfitting to a specific class also arises from this way of creating crops for training, because many of the crops are sparse regions i.e. the inter-residual distances are equal to or higher than 22 Ångstroms. A way to address this problem might be to construct random crops that represent amino acids whose place in the amino chain is close to each other i.e. perform crops in the region around the identity. This might increase the probability of constructing more useful crops. Furthermore, one could check during the cropping whether the proportion of pixels corresponding to very high distances is larger than a certain threshold. If that is the case, one might want to take another crop.

**Class-wise Temperature Scaling** As mentioned in subsection 10.1, we deal with pixel-wise classification rather than classifying a whole image or crop. Therefore, we apply the learned temperature to each pixel in the crop. In this case, learning a temperature vector consisting of a temperature per class might yield better results than a single temperature since a scalar temperature only might not be expressive enough to decrease the ECE and increase the entropy

appropriately. In order to train a temperature vector, the current implementation only needs few minor changes.

**Model Misspecification and Noise on Validation Set** As mentioned in the section 10.2.3 the FM and TBM-hard category are very less in number for the test set which makes the Misspecification and Noise computation unreliable for FM and TBM-hard category. This can be tackled by computing these metrics on the validation set which is considerably larger as compared to the test set, also the possibility to introduce data augmentation in the test set could help in producing more samples for these categories.

# References


[1] M. AlQuraishi, "Alphafold at casp13," *Bioinformatics*, vol. 35, no. 22, pp. 4862–4865, 2019.

[2] E. Alcaide, "Minifold: a deeplearning-based mini protein folding engine." `https://github.com/EricAlcaide/MiniFold/`, 2019.

[3] A. Senior, R. Evans, J. Jumper, and et al., "Improved protein structure prediction using potentials from deep learning," *Nature*, vol. 577, 706–710, 2020.

[4] K. He, X. Zhang, S. Ren, and J. Sun, "Delving deep into rectifiers: Surpassing human-level performance on imagenet classification," 2015.

[5] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay, "Scikit-learn: Machine learning in Python," *Journal of Machine Learning Research*, vol. 12, pp. 2825–2830, 2011.

[6] W. M. Billings, B. Hedelius, T. Millecam, D. Wingate, and D. D. Corte, "Prospr: Democratized implementation of alphafold protein distance prediction network," *bioRxiv*, 2019.

[7] N. L. Dawson, T. E. Lewis, S. Das, J. G. Lees, D. Lee, P. Ashford, C. A. Orengo, and I. Sillitoe, "CATH: an expanded resource to predict protein function through structure and sequence," *Nucleic Acids Research*, vol. 45, pp. D289–D295, 11 2016.

[8] C. Guo, G. Pleiss, Y. Sun, and K. Q. Weinberger, "On calibration of modern neural networks," *CoRR*, vol. abs/1706.04599, 2017.

[9] M. P. Naeini, G. F. Cooper, and M. Hauskrecht, "Obtaining well calibrated probabilities using bayesian binning," in *Proceedings of the Twenty-Ninth AAAI Conference on Artificial Intelligence*, AAAI'15, p. 2901–2907, AAAI Press, 2015.

[10] Y. Gal and Z. Ghahramani, "Dropout as a bayesian approximation: Representing model uncertainty in deep learning," 2015.

[11] L. Zhu and N. Laptev, "Engineering Uncertainty Estimation in Neural Networks for Time Series Prediction at Uber." `https://eng.uber.com/neural-networks-uncertainty-estimation/`, 2017. [Online; accessed 6-September-2017].