Protein Superfamily Classification with CNNs

Jeff Ruffolo jruffolo@jhu.edu (mailto:jruffolo@jhu.edu), Rena Liu rliu40@jhu.edu (mailto:rliu40@jhu.edu), Tanner Amundsen tamunds1@jhu.edu (mailto:tamunds1@jhu.edu)

Project mentor: Prashanthi Ravichandran

Applications: Genomics Data

Link to repo: `https://github.com/jeffreyruffolo/ProteinSuperfamPredictor`

Outline and Deliverables

Completed

1. **Must complete 1:** Create input dataset (pre-process structures and one hot encode sequences)

- For each sequence in the SCOP 2 dataset, we downloaded the corresponding PDB file from the RCSB Protein Data Bank. From the PDB files, we extracted the relevant sequence range and atomic coordinates and calculated the pairwise distances between carbon alpha atoms. in "Goals and Data" below (https://colab.research.google.com/drive/1fdmO7ym8ob11q2PGZzc1rPU4zA9Rxg0R#scrollTo=t-2GRU9bDDIT&line=1&uniqifier=1).

1. **Must complete 2:** Train neural network on 20% training dataset (approx. 6000 proteins)

- Discussed and finalized network design in "Results" below (https://colab.research.google.com/drive/1fdmO7ym8ob11q2PGZzc1rPU4zA9Rxg0R#scrollTo=zI53OUQHDE

1. **Must complete 3:** Test different model hyperparameters (e.g. convolution kernel size, number of layers, etc)

- Compared different number of layers in "Results" below (https://colab.research.google.com/drive/1fdmO7ym8ob11q2PGZzc1rPU4zA9Rxg0R#scrollTo=zI53OUQHDE

1. **Expect to complete 2:** Evaluate contributions of sequence and structure features to super-family prediction

- Compared different input combos in "Results" below (https://colab.research.google.com/drive/1fdmO7ym8ob11q2PGZzc1rPU4zA9Rxg0R#scrollTo=zI53OUQHDE

1. **Would like to accomplist 1:** Expand training set to all protein inputs in the SCOP2 data

- Used entire SCOP2 dataset in "Goals and Data" below (https://colab.research.google.com/drive/1fdmO7ym8ob11q2PGZzc1rPU4zA9Rxg0R#scrollTo=t-2GRU9bDDIT&line=1&uniqifier=1) ## Uncompleted Deliveables

1. **Expect to complete 1:** Compare alternative optimization algorithms (e.g. Adam)

- Trouble with SGD optimization implementation, thus could not compare with Adam (Loss converged to NaN and tuning learning rate was more difficult than expected)

1. **Would like to accomplish 2:** Expand to classify most common 100 super-families

- Ran out of time

Goals and Data

Problem: classify proteins into top 50 superfamilies 50 most represented in Worldwide Protein Data Bank (PDB); 51st class encompasses all other superfamilies

Two sources of input:

1. Amino Acid Sequence data
2. Distance matrix that encodes protein structure

Output: Which of Top 50 Superfamilies does the protein fall into?

How does this fit into the concepts of the course? This project is an example of a Supervised learning, multi-class Classification problem like we learned in class. Like many of the supervised learning problems we had in class, we made use of a train/ dev/ test split that allowed us to optimize our model over several epochs and then test our model's accuracy on a held out set of data. Our methods were very similar to HW5 and the deep learning module of the class. We used a convolutional neural network (CNN) which we used in HW5 on image date. Our input was not images but the 2D representations of our data (the distance matrix and a 2D representation of sequence data) can be treated like images in that when we can apply convolutional layers to extract shape- and shift-invaraiant features from the protein structure. Overfitting was a topic brought up in class often - to avoid overfitting we employed early stopping in our CNN.

What are the real-world implications of this data and task? This project may help scientists in understanding the role of newly discovered proteins without experimental work, specifically protein evolution and ancestry. Superfamily is the broadest, evidence-based, evolutionary categorization of proteins that is currently possible. Discovering a protein's superfamily can tell scientists a lot about the proteins structure given that many superfamilies are defined by their consituent proteins' structural similarities. Structure is much more evolutionarily conserved than sequence, so including a distance matrix as input to our model can expand the ancestral inferencing ability of our model.

How is this problem similar to others we've seen in lecture / breakout / homework? Our approach and architecture are similar to image processing on Hw5. We also were informed by the Deep Learning Lectures and Pytorch breakouts.

What makes this problem unique? We are testing a network architecture and using different inputs than recent publications.

What ethical implications does this problem have? We do not think there are strong ethical implications, since data collection and decisions from the results have minimal social application. We considered the ethical implications of protein mutation research if this model were to be expanded to evaluate the effects of perturbations (mutations) on superfamily classification. If this were to happen, this could tell scientists more about protein divergent evolution (little ethical relevance) but not much about predicting future protein mutations (more ethical relevance). So, as it stands, we do not think our model can have any serious ethical implications.

Data was collected through crystallography, and labeled based off the 50 most represented in the Worldwide Protein Data Bank (PDB)

Why did you choose this dataset? The Protein Data Bank (PDB) is the gold standard of structure data for modern protein data. Most major scientific journals require scientists to submit their structure data to the PDB and other renowned databases like SCOP and CATH use structures stored in the PDB. The sheer number of examples at our dispoal (~25,000) as well as the relevant labeling attracted us to this dataset. The PDB does not store structural information in a distance matrix but their coordinate data each residue in a given protein was exactly what we needed to create our distance matrix input.

Data Breakdown

- 25,509 examples
- 70% train, 10% dev, 20% test
- 51 labels (top 50 superfamily + outside of top 50)
- 100 + 1002 = 10,100 features per example

What did you do to pre-process your data? Why?
For the structural data, we created a distance matrix to encode structure which would otherwise be 3D For the protein sequence input, we created a one-hot encoding of sequence due its to length.

What features did you use or choose not to use? Why?
For protein sequences that were longer than 100 residues, we chose to only use a random subsequence (and subset of the distance matrix). An alternative remedy would be to add significant padding to shorter sequences in order to match dimensionality throughout the dataset. We expect that adding such extensive padding would harm model performance, as many shorter sequences would almost entirely consist of useless features and the model may develop biases based on sequence lengths rather than understanding the signature sequences and structural motifs of particular families. Additionally, padding all smaller inputs to match the largest would significantly increase training time, reducing the model variants we could've explored.

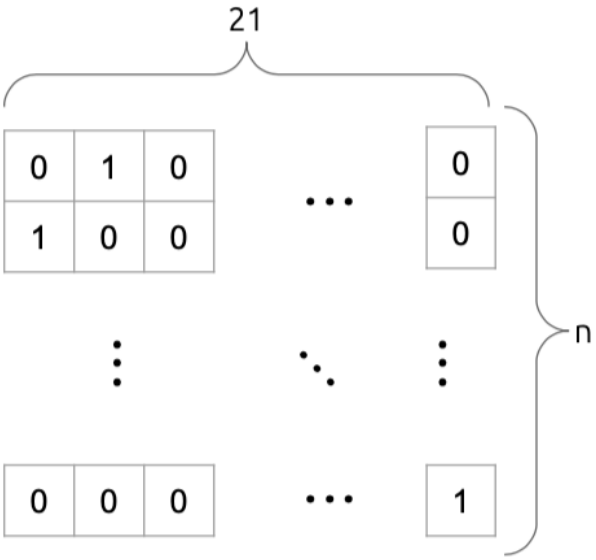
If you have categorical labels, were your datasets class-balanced?
Classes were reasonably balanced among the 50 largest superfamilies, with the most frequent class occuring about four times as often as the least frequent. Model training did not appear to be affected by this slight imbalance, so we did not take further steps to mitigate the imbalance. Under our Results section, we discuss how we accounted for class imbalance in our evaluation of recall, precision, and F1 score. Namely, we make use of scikit.metric classes that adjust how you evaluate recall, precision, and F1 score in the case of imbalanced data.

How did you deal with missing data? What about outliers?
Because the SCOP2 dataset is a carefully curated subset of the known protein structures, we were fortunate to not have to deal with low-quality data with missing features (such as missing coordinates for some CA atoms). However, some of our sequences contained non-canonical amino acids (outside of the regular 20). We chose to encode these outlier amino acids as a single 21st amino acid type. There where however instances of sequence length imbalance. For proteins with fewer than 100 residues, we padded their input both for the sequence encoding and for the distance matrix.

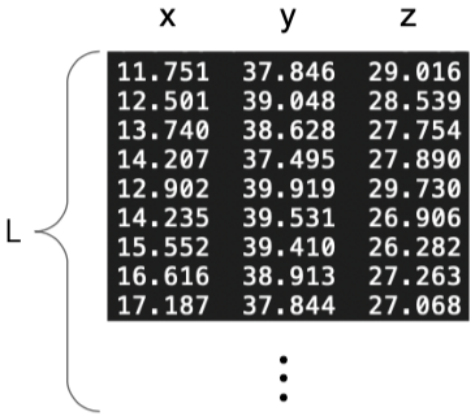
What approach(es) did you use to pre-process your data? Why?
Our main pre-processing was conversion of 3D structures to inter-residue distances. Atomic coordinates corresponding to the protein structure make poor features, as each type of amino acid has a different number of atoms and the 3D position itself is somewhat arbitrary (i.e. translation of the structure changes coordinates but doesn't affect structure). Rather than directly operating on the atomic coordinates for each protein structure, we constructed a pairwise distance matrix between the CA atoms of each amino acid to encode the protein structure [1]. The distance matrix can be treated as an image, and can benefit from techniques developed for image analysis [1].

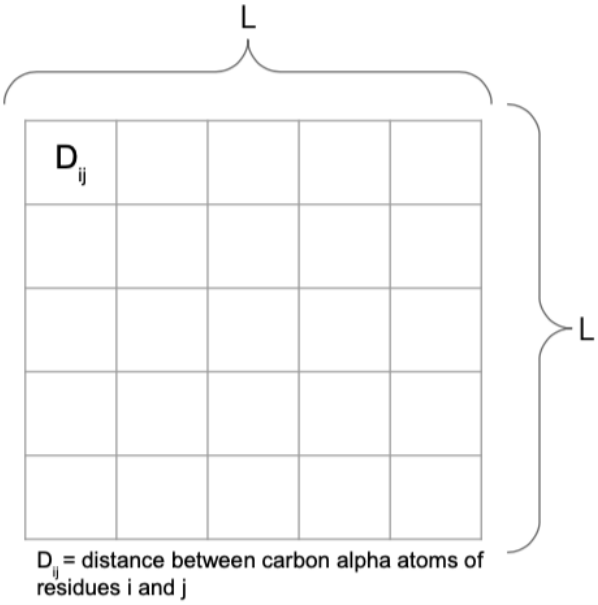
Are your features continuous or categorical? How do you treat these features differently?
Our sequence features are categorical, and we chose to represent each amino acid with a one-hot encoding. The calculated inter-residue distances are continuous features, which we pass directly into the network.

1. **What does it look like? Ex:**
   a. Amino acid sequence:



1. **What does it look like? Ex:**
   a. Distance matrix:

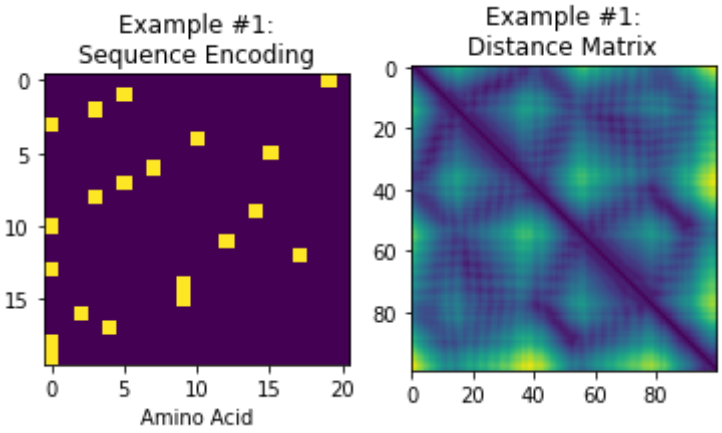


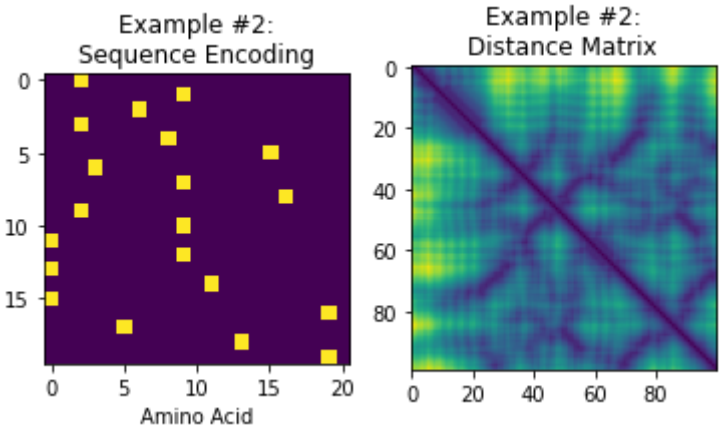


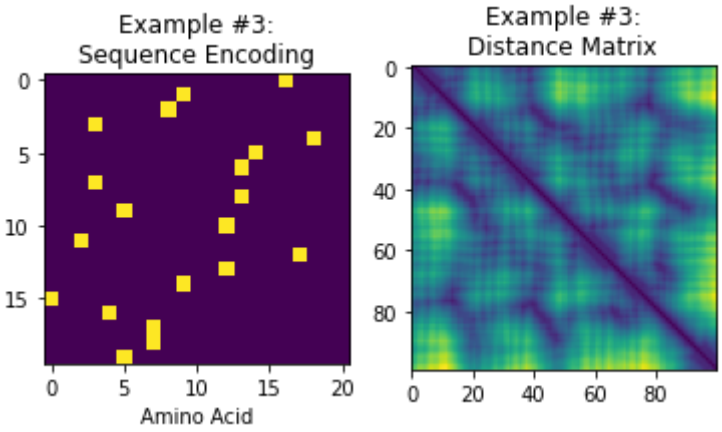
In [ ]:
```
from google.colab import drive
drive.mount('/content/drive', force_remount=True)
```

```
Mounted at /content/drive
```

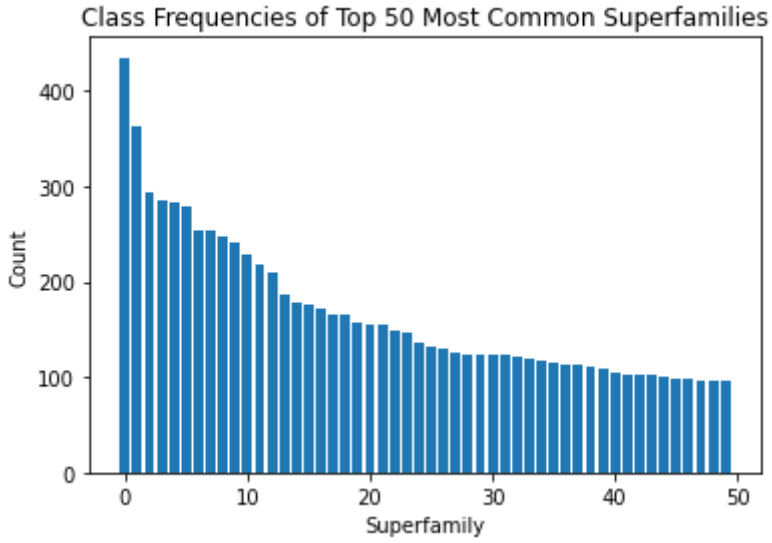
In [ ]:
```
import torch
import matplotlib.pyplot as plt
import sys
sys.path.append("/content/drive/My Drive/ML_Project")
from dataset import ProteinDataset

batch_size = 4
train_split = 0.9
dataset = ProteinDataset('/content/drive/My Drive/ML_Project/data/dataset.h5')
train_split_length = int(len(dataset) * train_split)
torch.manual_seed(0)
train_dataset, validation_dataset = torch.utils.data.random_split(
    dataset,
    [train_split_length, len(dataset) - train_split_length])
train_loader = torch.utils.data.DataLoader(
    train_dataset,
    batch_size=batch_size,
    collate_fn=ProteinDataset.merge_samples_to_minibatch)

# Loading and showing some distance matrices
num_examples = 3
example_count = 0
for (seq_feats, dm_feats), superfams in train_loader:
    if (example_count >= num_examples):
      break
    plt.subplot(1, 2, 1)
    plt.imshow(torch.transpose(seq_feats[0,:,:20], 0, 1).numpy())
    plt.title('Example #{example}:\nSequence Encoding'.format(example=example_count+1))
    plt.xlabel('Amino Acid')
    plt.subplot(1, 2, 2)
    plt.imshow(dm_feats[0])
    plt.title('Example #{example}:\nDistance Matrix'.format(example=example_count+1))
    plt.show()
    plt.close()
    example_count = example_count + 1
```


Example #1:
Sequence Encoding
Amino Acid
Example #1:
Distance Matrix

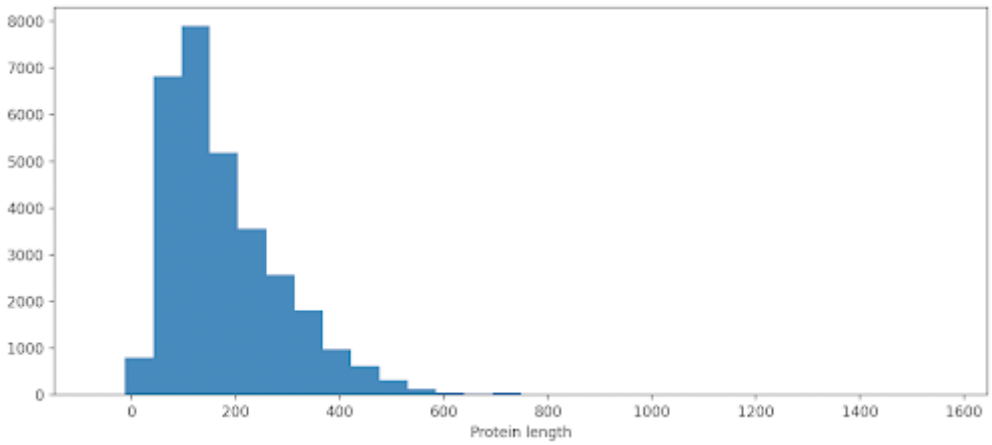

Example #2:
Sequence Encoding
Amino Acid
Example #2:
Distance Matrix


Example #3:
Sequence Encoding
Amino Acid
Example #3:
Distance Matrix

In [ ]:
```
superfam_freq_dict_sorted = dict(sorted(dataset.superfam_freq_dict.items
(), key=lambda item: -item[1]))
freq_dict_keys = list(superfam_freq_dict_sorted.keys())
top_50_freq = list(superfam_freq_dict_sorted.values())[:50]
plt.bar(list(range(50)), top_50_freq)
plt.title("Class Frequencies of Top 50 Most Common Superfamilies")
plt.xlabel("Superfamily")
plt.ylabel("Count")
plt.show()
plt.close()
```



Distribution before vs. after pre-processing: Since our pre-processing methods only reduced dimensions of our inputs by encoding the protein sequence and structural data as a distance matrix and weren't examples feature reduction, we don't have much relevant visualizations for before vs. after. However, to visualize the spread of amino acid sequence length across our examples, refer to the histogram below.



As you can see, the average number of residues in a given protein is between 100 and 200. As discussed above for "how did you deal with missing data?" we padded input proteins that had fewer than 100 residues.

Methods

What was your baseline?
Superfamily prediction is not a well-researched problem so it was difficult to find a baseline for NN-based model accuracy for the specific goal of superfamily prediction. However, CNNs *are* a well-researched tool for things like structure prediction and function prediction. DeepGO is a deep CNN with sequence inputs that classifies proteins by molelcular function (of which there are 693 distinct classes).

In the most updated version of their model (Kulmanov et al. January 2020), they are able to acheive 0.544 F1 score. We chose this as our baseline
Because of the similarities in input (amino acid sequence) and the scope of the classification. Molecular function is highly correlated with protein structure as is protein superfamily. Thus, our goal of superfamily prediction can be reasonably compared to molecular function prediction.

Did you look at related work to contextualize how others methods or baselines have performed on this dataset/task? If so, how did those methods do?
Among the functional classes (BP, MF, and CC) listed in Table 1. of Kulmanov et al., MF or molecular function is the category most comparable to our set of classes (superfamilies) so the accuracies under that column will be a rough baseline goal for us.

The comparison of performance on the first CAFA3 challenge dataset

Method	F_{max}			S_{min}			AUPR		
	MFO	**BPO**	**CCO**	**MFO**	**BPO**	**CCO**	**MFO**	**BPO**	**CCO**
Naive	0.290	0.357	0.562	10.733	25.028	8.465	0.130	0.254	0.456
DiamondBLAST	0.431	0.399	0.506	10.233	25.320	8.800	0.178	0.116	0.142
DiamondScore	0.509	0.427	0.557	9.031	22.860	8.198	0.340	0.267	0.335
DeepGO	0.393	0.435	0.565	9.635	24.181	9.199	0.303	0.385	0.579
DeepGOCNN	0.420	0.378	0.607	9.711	24.234	8.153	0.355	0.323	0.616
DeepGOPlus	**0.544**	**0.469**	**0.623**	**8.724**	**22.573**	**7.823**	**0.487**	**0.404**	**0.627**

Best performance in bold. F_{max} and AUPR, highest; S_{min}, lowest.

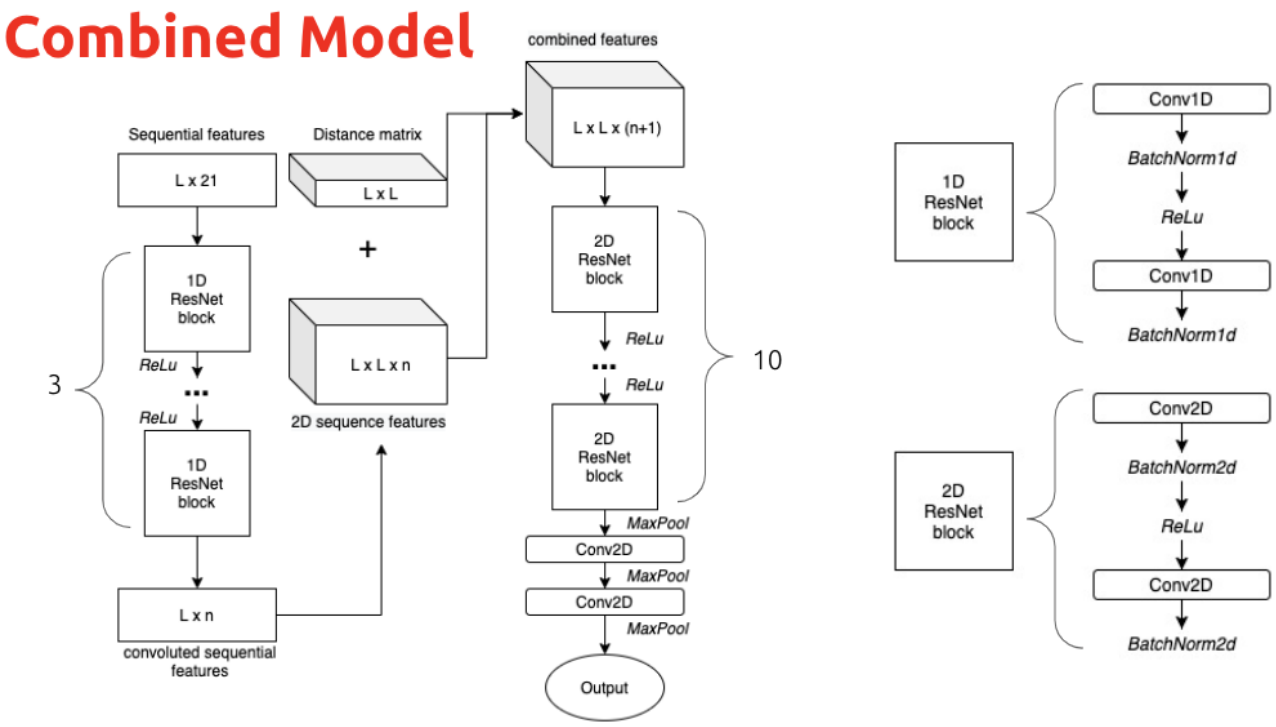
How did you split your data into train and test sets? Why?
Our train, development, and test sets were created by randomly splitting the original dataset into subsets of 70%, 10%, and 20%, respectively. After splitting, we verified that a reasonable class balance was achieved. We split our data (25,509 protein examples) into a 70%, 10%, 20% train, dev, test split. We did this to maximize the the number of unique proteins our model trained on while also allowing for optimization using a dev set (we trained over several epochs). Setting aside 20% of our data as a never-before-seen test set allowed us to make an unbiased evaluation of the final fit of our model after we finish training.

What methods did you choose?

- 1D ResNet blocks for sequence features
- 2D dilated ResNet blocks for pairwise features (distance matrix)
- 2D Convolutional layers between ResNet blocks
- ReLu activations
- MaxPool

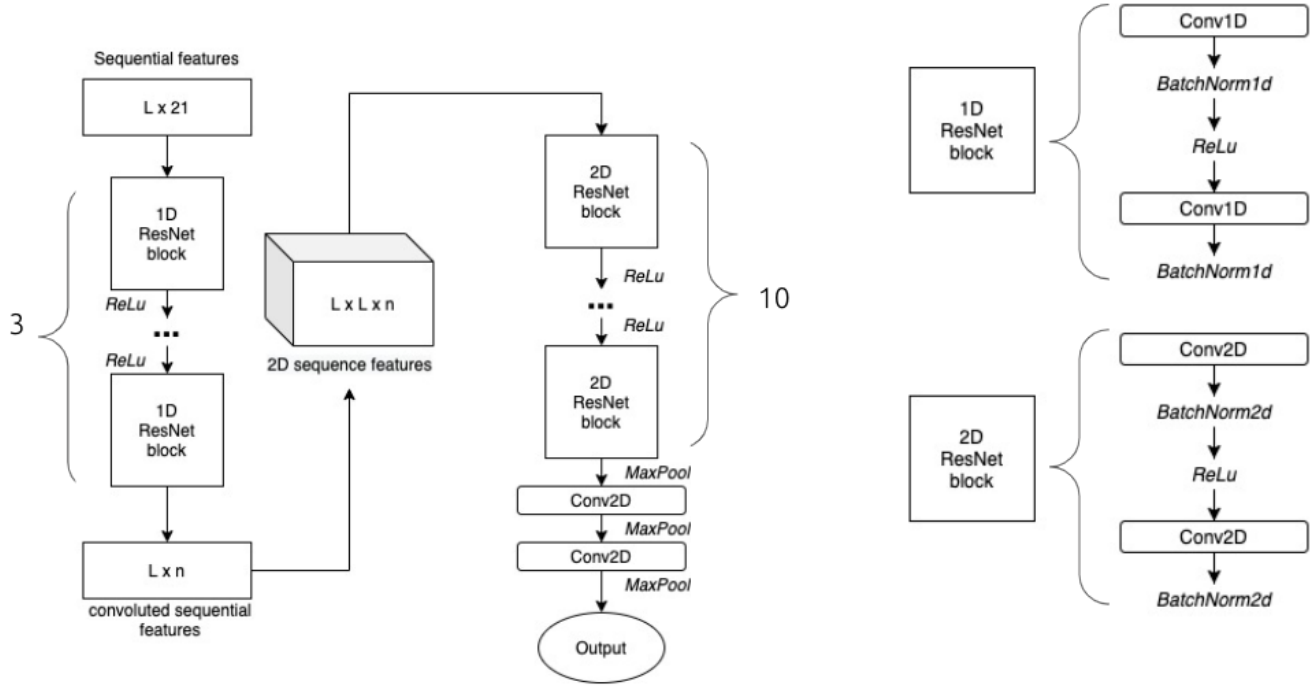
Below is a schema of our Combined Model which takes as input both a sequence encoding and a distance matrix. The sequence feature goes through a series of 1D Res Net blocks (ReLu activations between each layer, 11x11 kernel size at each convolution) and is then tranformed into a 2D representation using torch's unsqueeze() and expand() methods. This 2D sequence representation is then combined with the distance matrix and the combined data then goes through a series of 2D ResNet blocks (ReLu activations between each layer, 5x5 kernel kernel size at each convolution). Following this, the output of the final 2D ResNet block goes through 2 final convolutional layers with MaxPool activations.

To the right of the schema is a breakdown of the contens of the 1D and 2D resnet blocks.



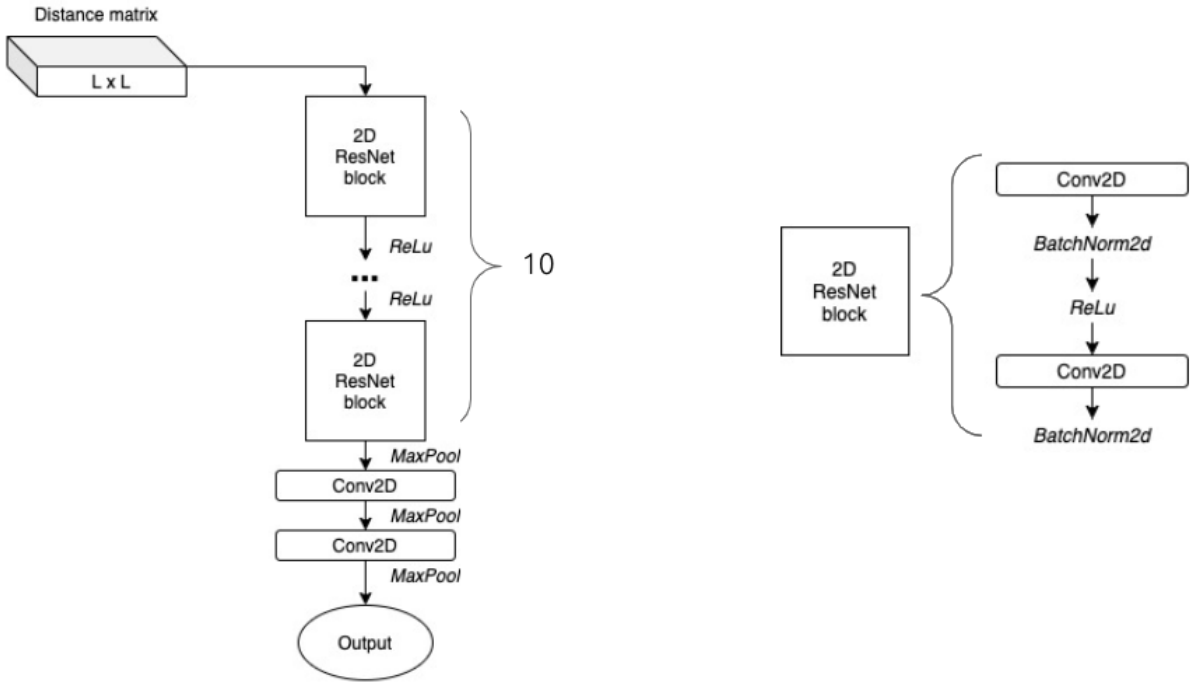
Next, we designed and trained a model that used exclusively sequence data (and not distance matrix data). The schema for that model is below:

Sequence Model



And finally we designed and trained a model that used exclusively the distance matrix as input:

Distance Model



Why did you choose them?
The above methods were chosen because of our problem's similarity to well-researched distance- and sequence-based deep learning for protein applications

Why ResNet?
We used ResNet since it addresses vanishing gradient problem, allows effective training of deeper networks

2D transformations of data - similarities to image classification

Regularization methods? Drop out? Early stopping?
We did do early stopping and batch normalization

What did you use for your loss function to train your models? Did you try multiple loss functions? Why or why not?
We trained our models with categorical cross entropy loss. This worked well and was a natural fit for our classification problem, so we did not need to explore alternatives.

How did you train these methods, and how did you evaluate them?

- Pytorch CNNs
- ReLU Activation
- Cross Entropy Loss
- Optimize with Adam

Evaluation metrics and why they're a reasonable evaluation metric for the task: We evaluated with a confusion matrix since it can tell us whether our model can account for overlap in superfamily properties and understand which superfamilies are most easily confused. Further explanation of the confusion matrix is explained below. From the presentations, another accuracy measure that people seemed to use with multi-class classification is F1 score. We additionally implemented recall based off recommendations from Prashanthi. In our calculation of recall, precision, and F1 score, we used scklearn.metrics functions 'recall_score', 'precision_score', and 'f1_score' with a weight parameter for each that adjusts for class imbalance. This mitigated our class imbalance among superfamilies. It is also worth mentioning that when one uses this weighted correction to calculate recall, precision, and F1 score, it can sometimes happen that the calculated F1 score does not fall between recall and precision (as would be expected since F1 is an average of recall and precision).

Which methods were easy/difficult to implement and train? Why?
We found the network implementation pretty easy, since we reimplemented similar code to Hw 5's image processing. What was harder was deciding exactly what shapes and parameters we wanted to specify for each level of the network architecture from our literation that best suited our application. Additionally, implementing SGD optimization was more difficult than expected since our loss convergence and learning rate tuning was giving us unexpected results.

For each method, what hyperparameters did you evaluate? How sensitive was your model's performance to different hyperparameter settings?
For the 2D blocks the hyperparameters we evaluated were adjusting the number of layers, 10, 6, and 2 layer blocks. We found relatively similar results for each of the layers, with 10 layers performing the best accuracies for overall.

In [ ]:
```
# Code for loss functions, evaluation metrics or link to Git repo
```

Link to repo:
https://github.com/jeffreyruffolo/ProteinSuperfamPredictor/blob/main/psfpred/evaluat

In [ ]:
```
# Code for training models, or link to your Git repository
```

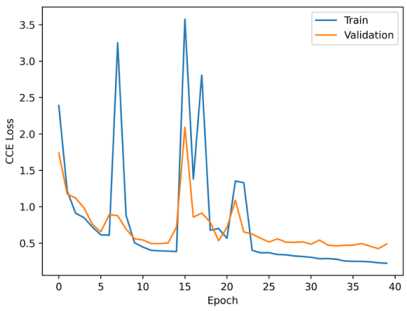
Link to repo:
https://github.com/jeffreyruffolo/ProteinSuperfamPredictor/blob/main/psfpred/train.p

Results

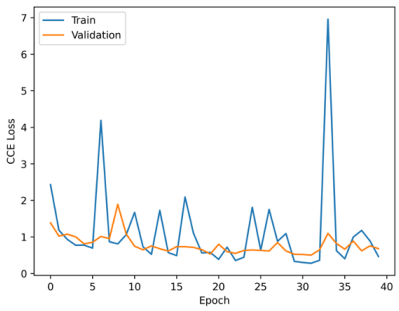
What about these results surprised you? Why?
Most surprising about our results was that the distance matrix input with the protein sequence input did better than sequence input alone, which we expected since we thought the distance matrix encoding structural data would be redundant info from what was already in the sequence input.

Did your models over- or under-fit? How can you tell? What did you do to address these issues?
Shown below are plots after each complete cycle over the training data for the three evaluated hyperparameter configurations. These models did not appear to suffer from over-fitting, and all appeared to have converged to a reasonable degree after 40 epochs.

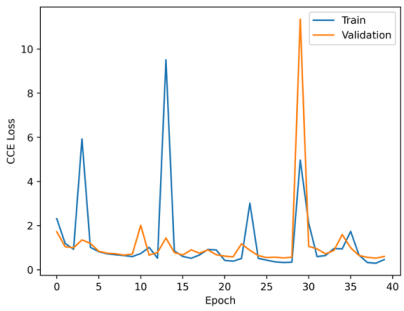
10 2D Blocks



6 2D Blocks

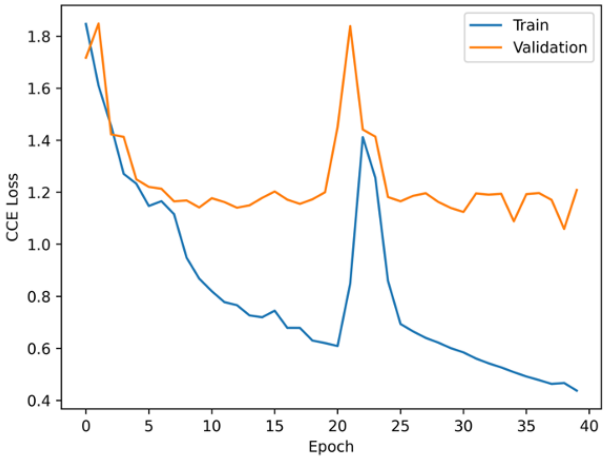


2 2D Blocks

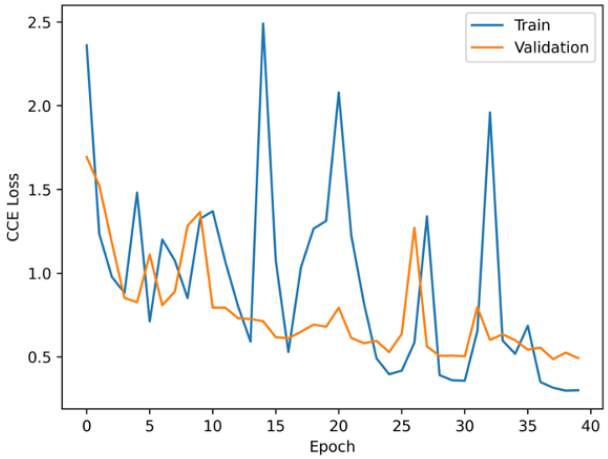


We did see signs of over-fitting when training the sequence-only model variant, shown below. After seven cycles through the training data, validation loss remained constant while training loss continued to decrease. We selected the model saved after seven epochs for later analysis.

Sequence Only



Distance Only



What does the evaluation of your trained models tell you about your data? How do you expect these models might behave differently on different data? Analysis of our loss plots indicate that our model is well-suited to predict superfamilies from distance matrices, but is not utilizing sequence features effectively. Other models, such as the work from Kulmanov et al, use only sequence to achieve accuracy comparable to our own, indicating that sequence information is more valuable than our model would make it seem.
On a different but similar dataset, perhaps that of Kulmanov et al and Gligorijevic et al, we would expect our

model to perform similarly. The aforementioned datasets predict protein functional classifications, which are similar to protein superfamilies though more granular. It is difficult to predict how our model would perform on a different protein classification problem altogether.

Show tables comparing your methods to the baselines.
Note: Values from Kulmanov et al are actually "Fmax" which incorporate a thresholding value, but are similar to F1 score. Because this is the closest baseline available in the literature, we will use these values despite this difference.

Input Comparsons: Accuracies across Different Model Inputs

Overall Labels Model:

Model Input	Recall	Precision	F1
Overall: Combined	.86	.85	.85
Overall: Only Distance Matrix	.86	.86	.85
Overall: Only Protein Sequence	.72	.68	.68
Baseline from Kulmanov et al.			.54

Top 50 Labels Model:

Model Input	Recall	Precision	F1
Top 50: Combined	.65	.85	.72
Top 50: Only Distance Matrix	.65	.89	.73
Top 50: Only Protein Sequence	.32	.60	.37
Baseline from Kulmanov et al.			.54

Hyperparameter Analysis: Accuracies across Different Numbers of Layers

Overall Labels Model:

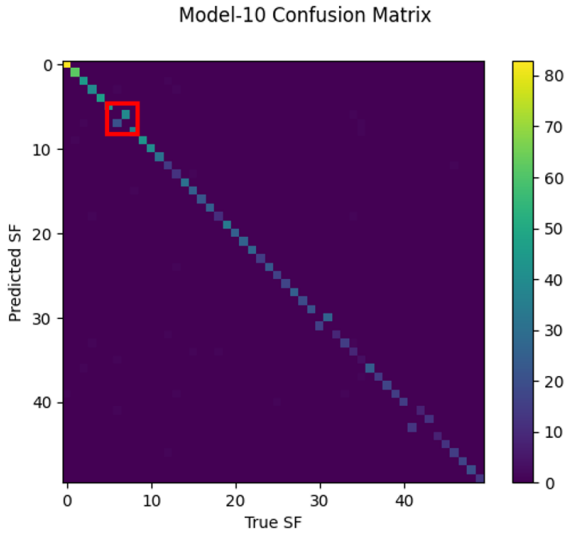
Number of Layers	Recall	Precision	F1
Top 50: 10 Layers	.86	.85	.85
Top 50: 2 Layers	.85	.85	.84
Top 50: 6 Layers	.82	.85	.83
Baseline from Kulmanov et al.			.54

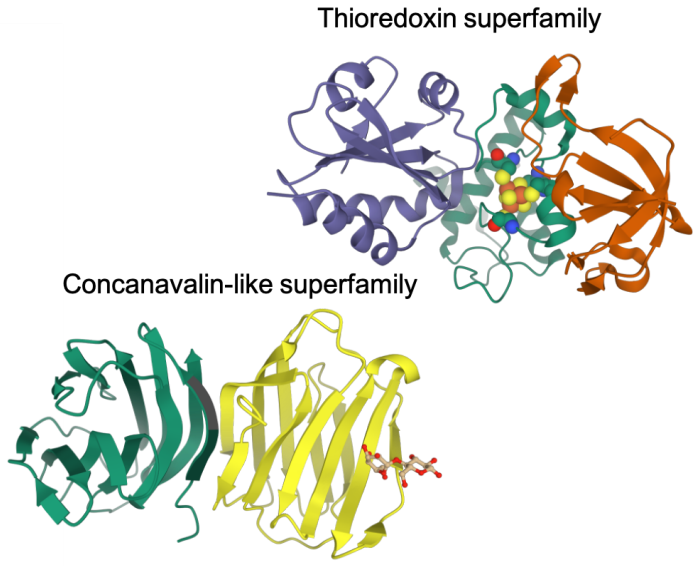
Top 50 Labels Model:

Number of Layers	Recall	Precision	F1
Top 50: 10 Layers	.65	.85	.72
Top 50: 2 Layers	.65	.86	.73
Top 50: 6 Layers	.68	.88	.75
Baseline from Kulmanov et al.			.54

Confusion Matrix

The confusion matrix below helps understand the errors made by our model on test set. The analysis below was performed using the model trained on both sequence and distance features with 10 2D residual blocks. There are three areas along the diagonal were the model appears to have consistently mixed up classification on the test set.
To better understand why these misclassifications occur, we focused specifically on two classes: thioredoxin superfamily and concanavalin-like superfamily. A representative protein from each of these families is shown to the right. Immediately we noticed that both proteins are fairly small and globular (i.e. compact), with significant beta sheet secondary structural elements. Additionally, both protein families interact with small molecules (as shown in the examples). Given these broad similarities, we believe it is reasonable that the model mixed these up, though future versions should correct for this.





How long training took

Training our baseline model (sequence and distance inputs with 10 2D residual blocks) took approximately 24 hours on an NVIDIA k80 GPU. Models with reduced input features and fewer blocks took less time.

Discussion

Note: you don't have to answer all of these, and you can answer other questions if you'd like. We just want you to demonstrate what you've learned from the project.

What aspects of your project did you find most surprising?
Based of initial discussions with Prashanthi, we believed that the extra protein structure data (represented in the distance matrix) would be redundant. However, we found that the distance matrix input did better than sequence alone.

What lessons did you take from this project that you want to remember for the next ML project you work on? Do you think those lessons would transfer to other datasets and/or models? Why or why not?
We realized that finding relevant literature review was pretty important to inform architecture & baselines, so would spend more time on that beforehand in the future. This lesson is definately applicable to other datasets, models, and projects, but would need to be balanced with not being too hesitant to start "quick and dirty" as opposed to designing the best model starting out.

If you had two more weeks to work on this project, what would you do next? Why?
We would expand to top 100 labels to see if that improve accuracies and do use LIME to understand explainability better.

What was the most helpful feedback you received during your presentation? Why? Feedback like "Have you investigated alternate CNN architectures? (for example could convolution and max-pooling layers after reading in the distance matrix better take advantage of it)," "Did you consider using protein amino-acid sequence alignment for protein classification? I would have thought you could do the alignments and compute a similarity score to predict a protein superfamily.," and "Does your model recognize mutated proteins? For example, hemoglobin vs mutated hemoglobin?" are all good suggestions for different hypotheses to test in further work.

References

Papers

1. Xu, J. (2019). Distance-based protein folding powered by deep learning. Proceedings of the National Academy of Sciences, 116(34), 16856-16865.
2. Kulmanov, M., & Hoehndorf, R. (2020). DeepGOPlus: improved protein function prediction from sequence. Bioinformatics, 36(2), 422-429.
3. Gligorijevic, V., Renfrew, P. D., Kosciolek, T., Leman, J. K., Berenberg, D., Vatanen, T., ... & Xavier, R. J. (2020). Structure-based function prediction using graph convolutional networks. bioRxiv, 786236.

Software packages

PyTorch