Peptide Fingerprint Prediction

Vallari Shende
Computational Biology Department
Carnegie Mellon University
vvs@andrew.cmu.edu

Anushka Sinha
Computational Biology Department
Carnegie Mellon University
anushka3@andrew.cmu.edu

Rohan Chawla
Computer Science Department
Carnegie Mellon University
rchawla2@andrew.cmu.edu

1 Introduction

Tandem mass spectrometry (MS2) is an analytical technique used to identify novel peptides and quantify known ones. It works by breaking down proteins into smaller peptide fragments which provides us a detailed analysis of their amino acid sequences. In MS2, a spectrum of ions is produced because peptides are first ionized and then fragmented. Each spectrum is essentially a fingerprint of a peptide. It helps us understand the protein composition of cells and tissues, as well as identify changes in protein expression under different physiological conditions.

A significant challenge within this field is the accurate identification of the exact amino acid sequence of peptides, which is critical for interpreting the data generated by MS2. In the computational framework for this problem, the input would be Tandem Mass Spectra (MS2) data which includes mass-to-charge ratios (m/z) and intensities of the fragmented ions and the output would be the peptide sequence that best matches the observed MS2 data.

There are two common techniques used to address this challenge. The first is in silico database search and the second is de novo prediction.

In silico database search compares the experimental MS2 spectra of peptides against a pre-built database containing theoretical spectra of known peptide sequences. It assesses how well the experimental spectra match the theoretical spectra by typically scoring the matches based on similarities in mass-to-charge ratios (m/z) and intensities of the ions. One limitation of this approach is its dependence on existing peptide databases, restricting its ability to identify novel peptides from the sample. Another limitation of this approach is that as the size of the database increases, the search becomes more computationally intensive. Moreover, there is an inherent risk of encountering false positives and false negatives.

De novo prediction identifies the amino acid sequence of a peptide directly from its MS2 spectrum without relying on a pre-existing database of known peptide sequences. It interprets the fragmentation pattern observed in the MS2 spectra by recognizing the mass differences that correspond to amino acid residues. Algorithms for this approach are used to predict the most likely peptide sequence that could have produced the observed spectrum. One limitation of this approach is that although it is fast, it typically contains errors. Additionally, the success of de novo prediction heavily depends on the quality of the MS2 spectra. Poor quality or highly complex spectra can lead to incorrect peptide sequence predictions.

Since identifying the whole peptide sequence from the MS2 spectrum presents a significant challenge, we aimed for a more feasible task which is to identify a set of 3-mers that most accurately match the observed MS2 data. Consequently, the updated computational framework for this problem will continue to utilize Tandem Mass Spectra (MS2) data, which includes mass-to-charge ratios (m/z) and intensities of fragmented ions, as input. However, it will now output a set of peptide 3-mers that best match the observed MS2 data. The objective of this project is to develop a deep learning model trained on MS2 spectra data, where each spectrum corresponds to a known peptide sequence. This will enable the model to accurately predict a set of 3-mers from new MS2 spectra, enhancing our understanding and identification of peptides in complex MS2 data.

We obtained the data for this project from National Institute of Standards and Technology (NIST) Peptide Spectral Library. NIST peptide libraries are extensive collections of annotated mass spectral references from different organisms and proteins. The spectra are generated using tandem mass spectrometers, which utilize liquid chromatography separations followed by electrospray ionization. For training, we utilized mass spectra data derived from human samples, comprising approximately $340,356$ total spectra resulting in a proteome coverage of 22.31%. Of these total spectra, $207,910$ represent unique peptide sequences. For validation, we used mass spectra obtained from mouse samples comprising $149,458$ total number of spectra resulting in 10.33% of proteome coverage. Of the total spectra, $94,055$ correspond to unique peptide sequences.

2 Methods.

In our project, we adopted two distinct approaches for analyzing the spectra, each treating the data in a unique manner. In the first approach, we considered the spectra from the mass spectrometer as a collection of individual peaks. Here, we treated peaks as distinct features, and our objective was to create a feature vector for each sample based on these peaks. This method allowed us to leverage specific peak patterns for classification or analysis tasks.

Conversely, in our second approach, we treated the spectra as sequential data. Instead of focusing on individual peaks, we analyzed the entire spectrum as a sequence of intensities. This sequential perspective was meant to capture broader patterns and dependencies within the spectra without explicitly isolating and categorizing individual peaks. By treating the spectra as sequences, we aimed to exploit the inherent structure and continuity of the data for more nuanced analysis and interpretation.

By taking these contrasting approaches, we aimed to evaluate and compare the effectiveness of feature-based versus sequence-based methods in our analysis of the mass spectrometry data. With this dual approach, we hoped to gain valuable insights into the optimal strategies for handling and utilizing mass spectrometry data within our specific analytical framework.

2.1 Data Pre-processing

Each sample in the dataset has two parts: the peptide sequence and its corresponding mass spectrometer readings (m/z ratios and corresponding intensities).

The first part of data pre-processing involved encoding the 3-mer peptide occurrences within a peptide sequence using an 8000-bit vector. We began by listing all possible 3-mer peptides formed from the 20 amino acids, resulting in 8000 unique combinations (e.g., AAA, AAC, AAD, up to VVV). Next, we applied a sliding window of size 3 across each peptide sequence. At every position, we identified the specific 3-mer peptide formed by the amino acids in the window and set the corresponding bit in the 8000-bit vector. If a particular 3-mer peptide was present at a position, we set the corresponding bit to 1; otherwise, it remained 0. This process was repeated for the entire length of each peptide sequence, generating an 8000-bit vector for each sequence. Each bit in the vector represented a unique 3-mer peptide, indicating its presence (1) or absence (0) within the sequence (Figure 1). This encoding method was applied consistently across all peptide sequences in our dataset for subsequent analysis and comparison.

In the second phase of data pre-processing, we focused on the mass spectrometer readings. Our goal was to transform the readings from each sample into a feature vector for our initial method, treating the detected peaks as individual features.

To achieve this, we employed a sliding window approach with a unit size of 0.25 m/z, spanning the range from 0 to 1000 m/z. This sliding window resulted in a matrix of size N x 4000 array, where N represents the number of samples, and each column of the matrix corresponds to a specific 0.25 m/z

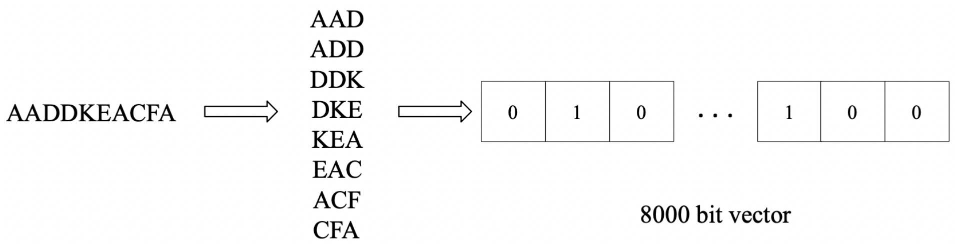

Figure 1: Encoding 3-mer peptide occurrences using bit vectors.

window within the range. The values in each column were calculated as the sum of intensities of peaks falling within the corresponding m/z window for each sample.

Then, we applied min-max normalization to each sample's feature vector. This normalization procedure rescaled the peak intensity values within each sample's feature vector to a common range (typically 0 to 1), preserving the relative relationships and distributions of peak intensities across different samples. The final outcome of this process was a normalized N x 4000 feature array, where each row represented a sample and each column represented a specific m/z window with its corresponding normalized peak intensity sum.

For our second approach, however, we adopted a different strategy. Here, we chose not to perform additional pre-processing on the mass spectra. Instead, we directly treated the spectra as sequential data, leveraging the inherent structure of the spectra without explicitly extracting individual peak features.

2.2 Method 1

The model takes as input a normalized feature array of size N x 4000 and consists of an Encoder followed by a Decoder.

The Encoder starts with an Embedding layer, comprising a single 1-D convolution layer, Batch Normalization, and ReLU activation. This is followed by a Max Pooling layer, which serves to downsample the features and potentially enhance translation invariance.

The Encoder then includes multiple Residual Network Blocks. These blocks are the same as used in the ResNet34 architecture (He et al. (2015)). Each block contains two consecutive 1-D convolution layers with Batch Normalization and ReLU activation, connected by a residual connection. The number of channels in these convolutional layers is increased to capture finer details. We experimented with varying the number of Residual blocks and channels to determine the optimal depth and breadth for performance enhancement.

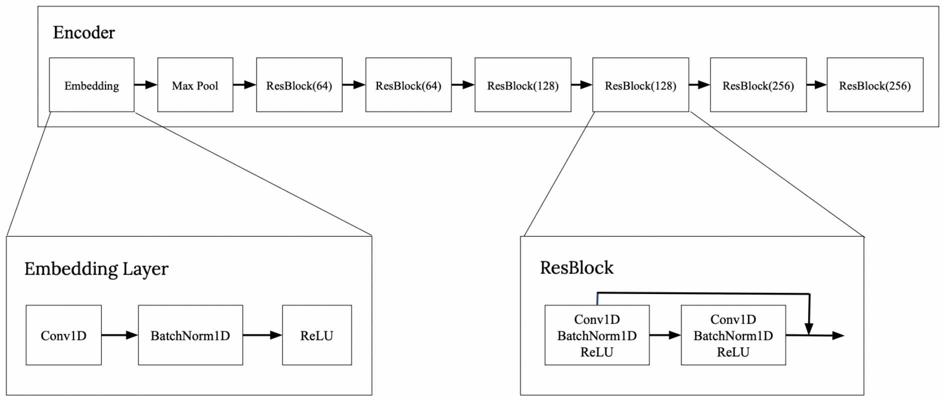

Figure 2: Encoder architecture when peaks are treated as features.

We passed the encoder output to a decoder (Figure 3), which was designed with multiple sequential blocks to process and classify the encoded input spectrum effectively. Each block consisted of a batch normalization layer, followed by a linear layer, GELU activation layer, and dropout layer. We used batch normalization to stabilize and accelerate the training process by normalizing the input layer, ensuring that the model learned robust features from the input data, thereby improving generalization and reducing overfitting. After batch normalization, we applied a linear layer to perform linear transformations on the normalized input, helping to map the input features to a higher-dimensional space where complex relationships could be captured and learned by subsequent layers. Next, we applied the GELU (Gaussian Error Linear Unit) activation function to introduce non-linearity into the model. To further enhance model robustness and prevent overfitting, we employed dropout layers after the activation functions. The final block in the decoder culminated with a linear layer responsible for the ultimate task of multi-label classification. This layer outputted probabilities for each bit in the 8000-bit vector, assigning labels to the spectrum based on learned representations from the preceding layers.

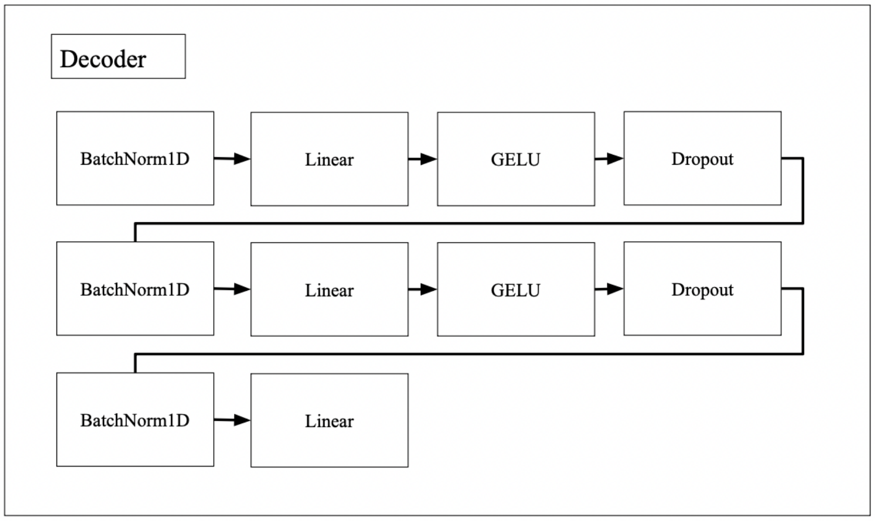

Figure 3: Decoder Architecture.

2.3 Method 2

The input to the model consisted of mass spectrometry spectra presented in sequential format, with each data point containing the m/z ratio and corresponding intensity. The model architecture comprised an Encoder followed by a Decoder.

In the Encoder (Figure 4), the input spectra were processed through an embedding layer followed by multiple RNN (Recurrent Neural Network) layers. The embedding layer included two 1-D convolution blocks with subsequent ReLU activation blocks. This setup was designed to transform the input spectra into a structured format that preserves sequential information, crucial for capturing patterns within the spectral data. The output from the embedding layer was then fed into the RNN layers, allowing the model to learn and encode the sequential patterns inherent in the input spectra.

Subsequently, the output from the Encoder was passed to the Decoder, which mirrored the architecture used in a previous method (refer to Figure 3).

2.4 Model Training

For all models, we used the AdamW optimizer with a weight decay of 0.1, and the Reduce LR on Plateau learning rate scheduler. We found these settings to work well after some initial ablation studies. Additionally, all models used the BinaryCrossEntropy loss function, with most utilizing a weighting function. We chose to use a weighted loss due to our imbalanced data. For any 3-mer, there

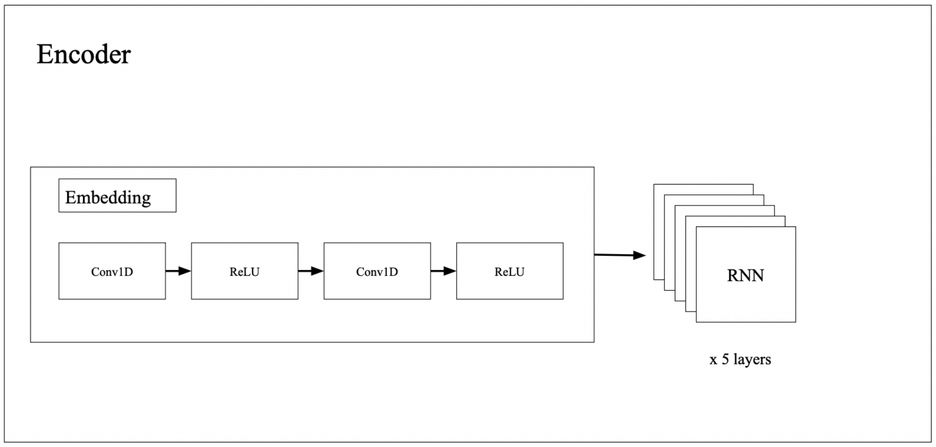

Figure 4: Encoder architecture when spectra are treated as sequential data.

are many more examples of negative spectra than positive spectra, so an unweighted loss function could encourage the model to always predict 0. To avoid this, we used the loss function to penalize the model more for misclassifying a positive spectra. The loss value would be multiplied by the following factor in such a case: $\frac{total\ \#\ of\ spectra}{\#\ of\ positive\ spectra\ for\ that\ threemer}$. This encourages the model to make more positive predictions.

2.5 Baseline

Although we could not find any reference models in the literature working on our specific task or identifying peptide fragments, there have been several recent works employing deep learning to directly predict entire peptide sequences (Yilmaz et al. (2024); Tran et al. (2019); Litsa et al. (2023); Qiao et al. (2021)). We chose to use Casanovo, which utilizes a encoder-decoder transformer architecture, as our baseline model as it is a very well performing model which was also compatible with our data and easy to deploy with publicly available weights and code (Yilmaz et al. (2024)). The model has been trained on its own cross-species mass spectra dataset, but its results have been shown to generalize to other datasets as well (Yilmaz et al. (2024)). Casanovo expects spectra in the mgf file format as inputs and outputs its own file format known mztab, which contains predicted sequences in addition to some other auxiliary information. We wrote our own script to convert our msp files to mgf files, and were successfully able to utilize the publicly available weights to perform inference on our training and validation data. We chose not to train the baseline on our own data, as we anticipated doing so would take a considerable amount of time/computational power, especially considering the model was based on a transformer.

To convert the outputs of Casanovo to be comparable to our model, we performed the same process described in Figure 2. For each predicted sequence, we split it up into 3-mers and then encoded the collection of 3-mers into a single 8000 bit vector. We were then able to directly compare these predicted bit vectors against the vectors produced by our model.

One important caveat of this model is that it makes predictions with modified amino acids within its vocabulary. In our data, we did not have any modified amino acids, but we still found Casanovo to predict sequences containing modified amino acids from our spectra. When processing these outputs, we considered modified amino acids to be equivalent to their unmodified versions, but this assumption is poor. It would be better to force the model to only predict unmodified amino acids, but doing so would require us to retrain the model. Additionally, the model utilizes a beam search algorithm to produce its predictions. We used the default beam width of 1 when performing inference. but a higher beam width would be more ideal as it would produce better results. Unfortunately, even small increases in beam width resulted in significantly longer inference times and memory requirements, so we were not able to run inference in time.

Besides this baseline, we also tested a simple MLP baseline of our own to assess whether model depth or weighted loss was necessary at all for our task. The description of this model can be found in Table 1.

3 Results

Model Name	Model Architecture
MLP Baseline	1-Layer of 8192 with unweighted loss
Model 1	5-layer birectional RNN, encoding dim = 128, hidden size = 64 with 2-layer decoder
Model 2	5-layer birectional LSTM, encoding dim = 128, hidden size = 64 with 2-layer decoder
Model 3	6-block ResNet (64, 64, 64, 128, 128, 256), with 2-layer decoder
Model 4	2-layer Decoder Only: (1024, 2048)
Model 5	Wider, 4-Layer Decoder Only: (8192, 4096, 2048, 2048)
Model 6	ResNet: Same 2-layer decoder, 6-block encoder, Encoding dim = 256
Model 7	ResNet: Same 2-layer decoder, 6-block encoder, Encoding dim = 512
Model 8	ResNet: Same 2-layer decoder, 3-block encoder (64, 64, 64), Encoding dim = 256
Model 9	ResNet: Same 2-layer decoder, 8-block encoder (64, 64, 64, 128, 128, 256, 256, 512), Encoding dim = 256
Model 10	ResNet: Encoding dim = 512, same 8-block encoder, 4-layer Decoder (1024, 2048, 2048, 2048)

Table 1: Model architectures. All models were trained for 30 epochs and employ the weighted loss described in the Methods section unless otherwise specified. RNN and LSTM were the only models to use the sequences of (peak mass, intensity) as features. All other model used the sliding window approach described in Data Pre-processing. Encoding dim indicates the dimension of the vector outputted by the encoder, which is then fed into the decoder.

Model	Acc_T	Acc_V	Prec_T	Prec_V	Recall_T	Recall_V	Spec_T	Spec_V	F1_T	F1_V
Casanovo Baseline	0.9969	0.9969	0.00803	**0.00559**	0.00549	0.00559	0.9985	0.9985	0.00564	0.00569
MLP Baseline	**0.998**	**0.998**	0.0005	0.0004	0.0004	0.0002	**1**	**1**	0.0005	0.0003
Model 1	0.9826	0.5506	0	0.0008	0.0042	0.0440	0.9397	0.5670	0	0.0015
Model 2	0.9824	0.5406	0	0.0009	0.0002	0.0468	0.9911	0.5312	0	0.0016
Model 3	0.8822	0.8112	0.0104	0.0047	0.1507	0.0650	0.8821	0.8113	0.0185	**0.0084**
Model 4	0.6205	0.5402	0.0028	0.0025	0.1205	**0.0702**	0.6203	0.5397	0.0054	0.0047
Model 5	0.5651	0.4647	0.0025	0.0023	0.1169	0.0693	0.5649	0.4643	0.0049	0.0043
Model 6	0.9135	0.8097	**0.0154**	0.0044	**0.1566**	0.0616	0.9134	0.8097	**0.0258**	0.0078
Model 7	0.8731	0.8013	0.0095	0.0045	0.1483	0.0664	0.8731	0.8013	0.0170	0.0080
Model 8	0.9706	0.5444	0	0.0008	0.0011	0.0449	0.9723	0.5445	0	0.0016
Model 9	0.9817	0.5445	0	0.0008	0.0005	0.0443	0.9834	0.5446	0	0.0015
Model 10	0.8354	0.8910	0.0071	0.0030	0.1453	0.0313	0.8353	0.8918	0.0130	0.0048

Table 2: Model training metrics. Acc indicates accuracy, Prec indicates precision, Spec indicates specificity and F1 indicates F1 score. The T and V subscripts indicate training and validation, respectively.

We performed several ablation studies by varying model architectures, as can be seen in Table 1 and Table 2. Due to time constraints, we were only able to train most models for 30 epochs, which is relatively small, but we still found our results to be meaningful.

3.1 Metrics

We used the following metrics to evaluate our models' performances. These are all meant to account for the unequal distribution of our data. Below, TP indicates true positives, TN indicates true negatives, FP indicates false positives and FN indicates false negatives.

$$Accuracy = \frac{TP + TN}{TP + TN + FP + FN} \qquad Precision = \frac{TP}{TP + FP}$$

$$Recall/Sensitivity = \frac{TP}{TP + FN} \qquad Specificity = \frac{TN}{TN + FP} \qquad F1 = \frac{2 * Precision * Recall}{Precision + Recall}$$

From these equations, we can see that accuracy measures overall correctness, precision measures the likelihood a positive prediction is correct, recall measures the likelihood a positive is actually predicted as positive, specificity measures the likelihood a negative prediction is correct and F1 score is similar to an average of precision and recall.

3.2 Analysis

Looking at the performance of our two baselines, we can see that both struggle on our task. The high accuracies and specificities but very low precision, recall and F1 scores reveal these models are making very few true positive predictions. This means the models are predicting 0 very often. The MLP baseline doesn't use a weighted loss function, so this makes sense, and likely explains its very high accuracies and specificities. The Casanovo baseline performs better, but still suffers. It achieves similar performances between the training and validation data, which makes sense, since it wasn't actually trained on the training data, and appears to generalize equally well to both datasets. While it seems to produce more true positives than the MLP baseline, it still isn't doing so very often. This is potentially due to its use of modified amino acids and the limited beam search as mentioned earlier, or perhaps due to some other hyperparameter setting we were not aware of. However, these results are encouraging, as they indicate our approach improves upon these baselines.

First, comparing models 1, 2, 3, we can see that the ResNet architecture performs significantly better than the RNN or LSTM. The training F1 scores for the RNN and LSTM are very low, because of their low precision. This indicates these models are unable to accurately make true positive predictions. Additionally, all validation metrics are higher for the ResNet architecture, so this model can not only perform better on the training data, but also seems to generalize much better. Thus, the ResNet encoder looks like it can create more meaningful encodings.

We next decided to test whether an encoder was necessary at all. In model 4, we removed the encoder entirely, to see if this degraded performance. To make a fairer comparison, we also tested a larger decoder in model 5, to ensure a potential degradation of model performance wasn't just due to a decrease in model size. Looking at the results from models 3-5, we can see that an encoder is very helpful. Pretty much all metrics, except for validation recall, are significantly worse for the decoder-only models. This indicates the encoder is useful in producing a meaningful representation early on, as the decoders seem unable to produce accurate predictions. The higher recall of these two models means these models are producing fewer false negatives, but that likely indicates they are predicting 1 too often, which is also a bad thing. Also, comparing the MLP baseline performance to models 4 and 5 shows how much the weighted loss helps in forcing the model to predict 1 more often, which improves performance.

From these results, we were now more confident that our encoder-decoder architecture with a ResNet encoder performed well on this task. The model beat the Casanovo baseline by most metrics and evidently produces meaningful representations that generalize well. We next decided to vary the architecture of this model, to see if increasing depth or size would help us.

The first thing we tested was increasing the size of the encoded vector. In models 3, 6 and 7, we used the same encoder and decoder, and only changed the size of the intermediate encoding vector. Our assumption was that a larger encoding vector would be able to incorporate more information from the spectra, and thus ideally produce better results. We found that increasing the encoding vector size from 128 to 256 did seem to improve model performance, but beyond that was not that helpful. Model 6 is able to achieve higher training precision, recall and F1 score than model 3. This means the model is learning to make true positive predictions more often, which is very encouraging. However, the validation metrics for model 6 are approximately the same as model 3. For model 7, it seems to perform the same as model 3. Overall, it seems like increasing the encoding dimension a little does allow the model to better learn the training data. Increasing the dimension too much seems to degrade performance, perhaps due to inefficient training or due to the inclusion of too much extra noise. Overall, it seems all three models struggle with generalizing well. Although we employ both dropout and weight decay as regularization techniques, potentially a better regularization technique might be able to address this. Overall, it seems like increasing the encoding dimension a little helps with training performance, but does not significantly affect validation performance.

After testing increasing the encoding dimension, we next tried to assess how changing the depth of our model would affect performance. We couldn't increase the depth too significantly, to avoid increasing training times too much, so we decided to test just the encoder depth at first. From comparing models 1-3, we know the advantage of our model is in its ability to create meaningful encodings, so we decided to just vary the depth of our encoder, since it seemed to be the best performing component. Ideally, a larger depth would create a more meaningful representation that would improve performance. We decided to test a shallower encoder in model 8 and a deeper encoder in model 9. We used an encoding dimension of 256 since it was able to achieve the best performance on our training data at that point. Thus, we compared models 8 and 9 to model 6. From our results, we can see that both decreasing and increasing the encoder depth seemed to hurt performance. The precision scores for models 8 and 9 are significantly worse, which indicates the models were making very few true positive predictions. For model 8, this would make sense, since the shallower depth would likely produce less meaningful representations of the spectra, and thus degrade performance. However, the poor performance for model 9 was very surprising. Ideally, greater depth should produce more hierarchical features which should result in more meaningful representation. If anything, the performance should be the same as model 6, not worse. This is likely due to our limited training time. Training for longer could have potentially improved performance. There's a chance that this model also just ended up getting stuck in a poor local minimum, and resetting the seed or changing some other hyperparameters could have improved performance. Overall, it seems the depth we chose initially for models 3 and 6 were ideal, and varying it hurts performance.

Finally, in our last test, we wanted to see if increasing the size of the model significantly and training for a very long time would improve our performance. In model 10, we used the largest depth encoder, a deeper and wider decoder, and also a greater width encoding vector of 512, and trained for 50 epochs. Our hope was that using a larger model and training for longer would allow for better performance, since this seems to be the solution for most deep learning models. Unfortunately, the performance of model 10 isn't better than models 3 or 6, and greater model size does not seem to be that helpful. However, there is also a good chance this is due to hyperparameter settings. In most scenarios, greater depth does improve model performance, so there is a chance our training procedure is what is causing issues. We didn't have a chance to really optimize and vary the training process, like trying different weight decays or optimizers or learning rate schedulers. All of these things could potentially improve the performance of our model.

In conclusion, we were able to improve upon our baseline's performance on this task by employing an encoder-decoder architecture. Using an encoder allows us to create meaningful representations of spectral data quickly that generalize as well. We found a ResNet encoder to function better than an RNN and LSTM encoder, and our weighted loss function allowed us to produce models which predict something besides 0. However, overall, we still aren't able to achieve the best performance. The precision scores for our model are still low, and much lower than the recall score, indicating that there are too many false positive predictions. Using a different weighting for our loss function might improve this, though we didn't try testing this. There is also some room to improve how much our models generalize, as our validation metrics are still lower than our training metrics. Future directions could include different regularization techniques, or using better data. A lot of our mass spectra are noisy, and potentially using feature engineering or data augmentation to produce better spectra could help. Finally, we found our ResNet encoder to function well, but there is a lot of space to explore different encoder architectures to perform better, like a transformer or even testing the ResBlock structure used in ResNet50 (He et al. (2015)).

References

Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. 2015. Deep residual learning for image recognition. *CoRR*, abs/1512.03385.

Eleni E. Litsa, Vijil Chenthamarakshan, Payel Das, and Lydia E. Kavraki. 2023. An end-to-end deep learning framework for translating mass spectra to de-novo molecules. *Communications Chemistry*, 6(1):1–12.

Rui Qiao, Ngoc Hieu Tran, Lei Xin, Xin Chen, Ming Li, Baozhen Shan, and Ali Ghodsi. 2021. Computationally instrument-resolution-independent de novo peptide sequencing for high-resolution devices. *Nature Machine Intelligence*, 3(5):420–425.

Ngoc Hieu Tran, Rui Qiao, Lei Xin, Xin Chen, Chuyi Liu, Xianglilan Zhang, Baozhen Shan, Ali Ghodsi, and Ming Li. 2019. Deep learning enables de novo peptide sequencing from data-independent-acquisition mass spectrometry. *Nature Methods*, 16(1):63–66.

Melih Yilmaz, William E. Fondrie, Wout Bittremieux, Carlo F. Melendez, Rowan Nelson, Varun Ananth, Sewoong Oh, and William Stafford Noble. 2024. Sequence-to-sequence translation from mass spectra to peptides with a transformer model. *bioRxiv*.