Max Lifespan Research Log

Wyatt McCarthy

0.1 Introduction

This semester's research builds on the foundational work conducted during the Spring, detailed in a report you can view here. The original study investigated a hypothesis proposed by David Sabitini, suggesting that fundamental components of DNA could influence a species' maximum lifespan. By analyzing orthologous gene sequences across 453 mammalian species, we explored evolutionary conservation and its implications for longevity.

The initial phase of the project involved extensive data cleaning and analysis of DNA sequences from the Zoonomia Project, focusing on conserved orthologs and transcription factor binding sites. Statistical approaches such as chi-square tests and clustering techniques allowed us to identify significant associations between genetic variation and lifespan. These insights informed the development of a predictive modeling framework, which leveraged species-aware DNA embeddings and machine learning architectures like PerceiverIO. However, early results indicated challenges in model generalization, highlighting the need for refined training data and alternative approaches.

Over the summer, collaborators extended this work by deepening statistical analysis and experimenting with additional model architectures. This semester's efforts aim to refine the data pipeline, address noise in the training datasets, and conduct various modeling experiments.

This log chronicles the steps undertaken, challenges encountered, and progress made in advancing the project throughout the semester.

Weeks of: September 16th - September 30th

The focus of this period was to get re-situated with the project and caught up on the work that was done over the summer. For a quick recap, when I finished my research in the spring, we had just gotten to modeling, following a lengthy EDA period where we identified potentially "good" and "bad" data.

Quick EDA Recap:

- We have 51 million DNA sequences orthologous to the human genome from 453 different mammalian genomes
- Added `lifespan` variable to data (according to the species which a sequence was from)
- Organized data into sets categorized by the gene from which a sequence was extracted
- For each gene:
  - used DNABERT-S embedding model to embed sequences (embeddings of dim 1 x 768)
  - reduced embedding dimensionality to 1 x 3 using PCA
  - k-means clustered reduced embeddings
    - idea here is that clustered embeddings represent similar DNA sequences
  - computed lifespan statistics (mean, median, std deviation, z score) for each cluster to glean whether there is association between DNA similarity and lifespan in any clusters
    - if there is no association between DNA similarity and lifespan, data is definitely bad; if there is association, data has potential
  - Glimpse of the data we've collected:

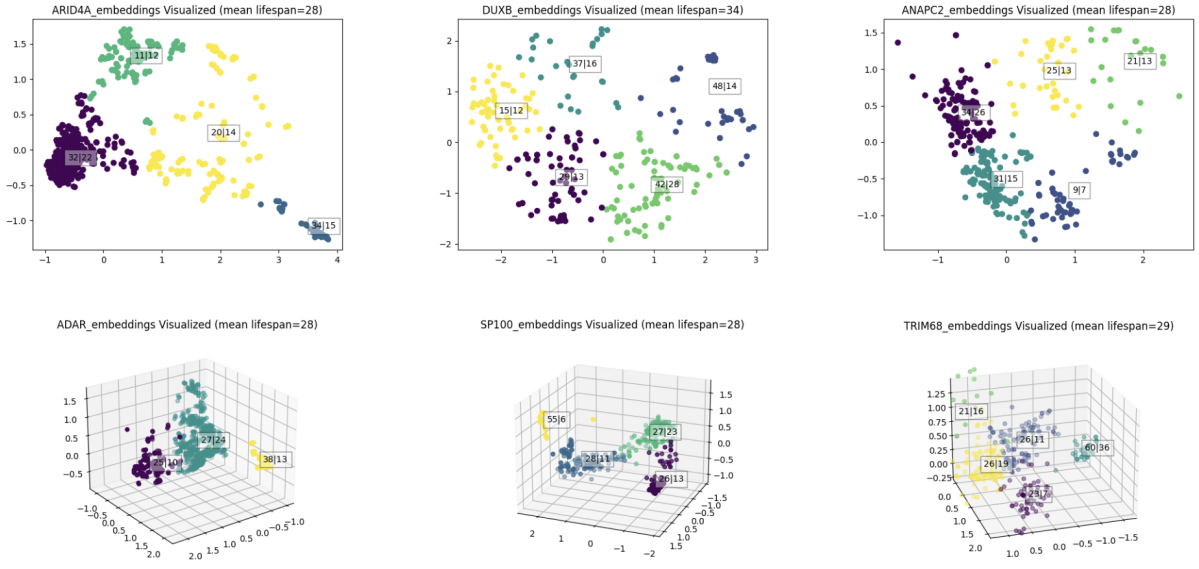
gene type	max seq len	min seq len	mean seq len	median seq len	wmc similarity score	species represented
E2F2	1374	1308	1327.8	1323	0.2	418
ARID4B	4131	3669	3819.2	3900	0.2	418
NFKBIE	1626	1083	1295.0	1101	0.1	415

gene type	set mean lifespan	set std dev	set iqr bounds	cluster	cluster mean lifespan	cluster std dev	cluster iqr	cluster iqr bounds	#points in cluster	#species in cluster	modified z-score	statistical significance
PFN1	27.4	21.1	15-35	cluster0	23.8	14	17.6	14-32	315	181	-0.1	0.5
PFN1	27.4	21.1	15-35	cluster1	46.1	37	41.0	20-60	59	42	1.2	0.1
PFN1	27.4	21.1	15-35	cluster2	30.8	11	5.2	27-32	8	6	0.5	0.3

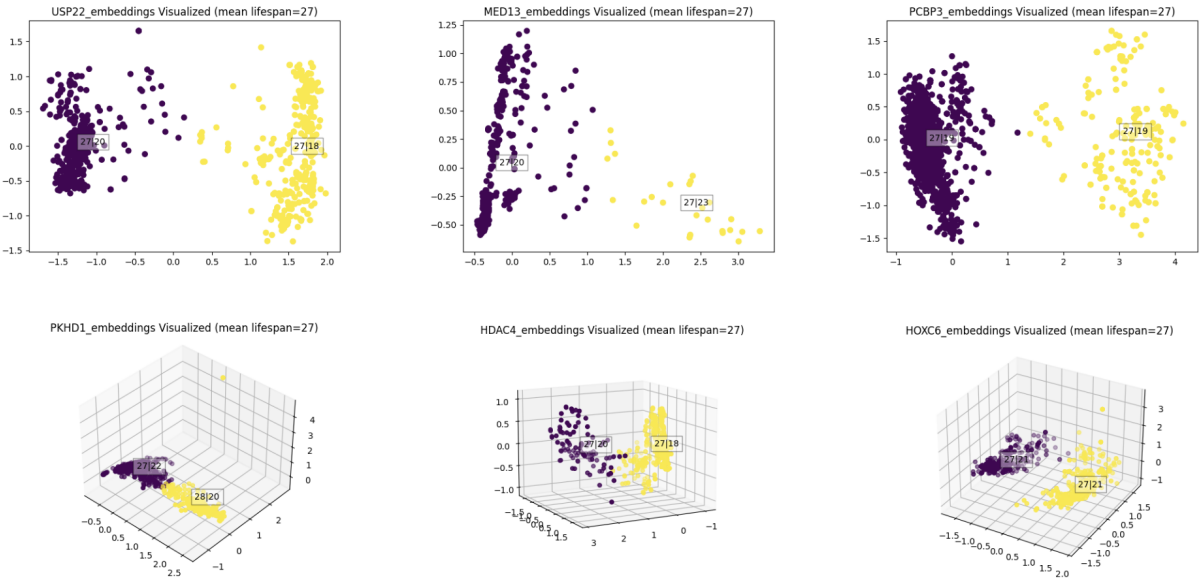
gene	avg cluster stat sig	most stat sig cluster	#species in most sig cluster	least stat sig cluster	avg # species per cluster
PFN1	0.306	0.124	42	0.474	76.333
ZNF804A	0.393	0.324	104	0.433	64.750
IRAK3	0.442	0.340	61	0.494	68.500

- And some examples of clustering results (and how they vary according to z-score)

Embedding visualizations on sets for which there are high z-scoring clusters



Embedding visualizations on sets for which there are low z-scoring clusters



Quick Modeling Recap:

- The main purpose of the EDA described above was to identify and compile training data; the idea from there was that if we could train a model to accurately predict lifespan when given a DNA sequence from an "outlying" species, we were on to something
  - in this context, by "outlying" species we mean a species whose lifespan is uncharacteristically large relative to genetically similar species
  - to identify such species, we performed further analysis, briefly shown in the tables and plots below

family	median lifespan	IQR	outlying species	outlying max lifespan
Procyonidae	24.0	5.40	Potos_flavus	38.4
Lemuridae	36.2	1.80	Prolemur_simus	17.6
Hominidae	59.0	1.27	Pan_paniscus	55.0
Cervidae	22.0	4.70	Cervus_elaphus	31.5
Cervidae	22.0	4.70	Muntiacus_crinifrons	11.0

genus	median lifespan	IQR	outlying species	outlying max lifespan
Microtus	4.80	0.83	Microtus_oeconomus	2
Macaca	38.05	2.92	Macaca_mulatta	30
Pteropus	20.60	7.27	Pteropus_giganteus	44

order	median lifespan	IQR	outlying species	outlying max lifespan
Rodentia	7.3	8.00	Hystrix_cristata	28.0
Rodentia	7.3	8.00	Heterocephalus_glaber	31.0
Rodentia	7.3	8.00	Coendou_prehensilis	26.6
Ruminantia	22.0	8.60	Giraffa_camelopardalis	39.5
Whippomorpha	49.5	38.67	Balaena_mysticetus	211.0

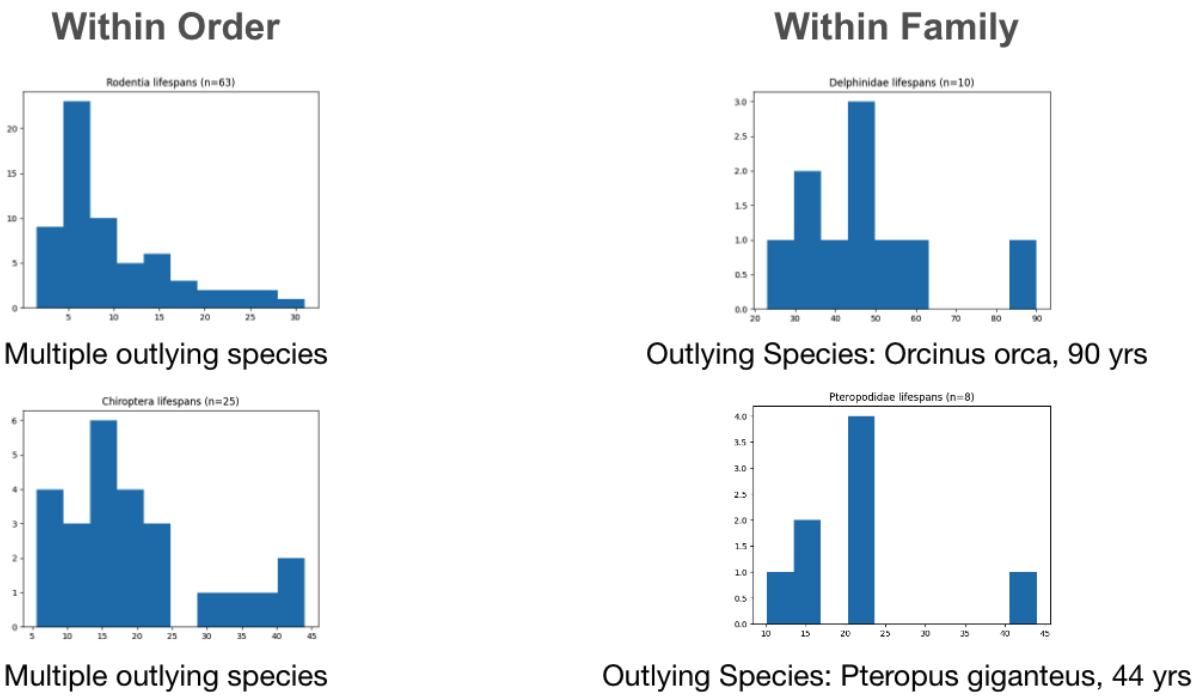


Figure 1: Species Lifespan Histograms

- With the context gleaned from this EDA, we experimented with various modeling approaches:
  - The Perceiver Model
    - unsuccessful, model did not appear to learn well regardless of the configuration of training data tested; some results shown below

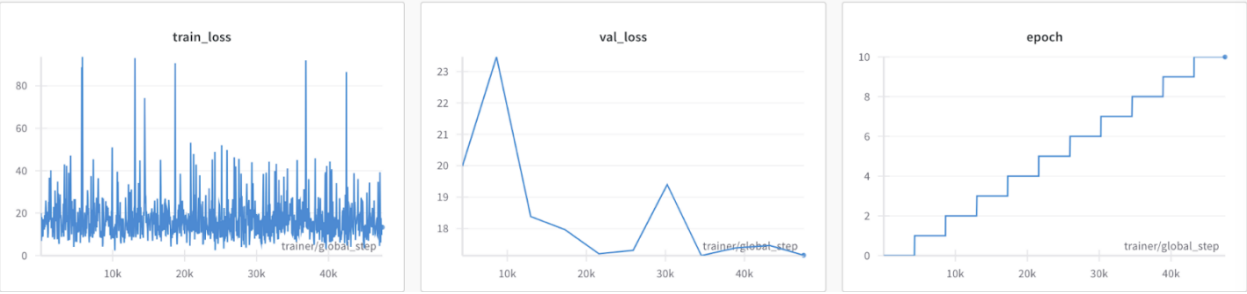


Figure 2: Perceiver Training Results

- The Enformer Model
  - some successful runs; able to get low validation loss on two sets of training data; one with 1000 entries selected from the top 6 genes (according to statistically significant clusters) and another with 100 arbitrarily selected entries across all data
  - positive results here inspired us to investigate training on isolated gene datasets; finding a relationship between DNA composition and lifespan within a certain gene would be particularly interesting
  - when trying to train on larger datasets, we ran into memory issues due to how we were loading in data. That is, we were trying to read in a 20gb dataset rather than processing in smaller chunks at a time (this explains why per-gene analysis cut-off after a very small subset of genes)

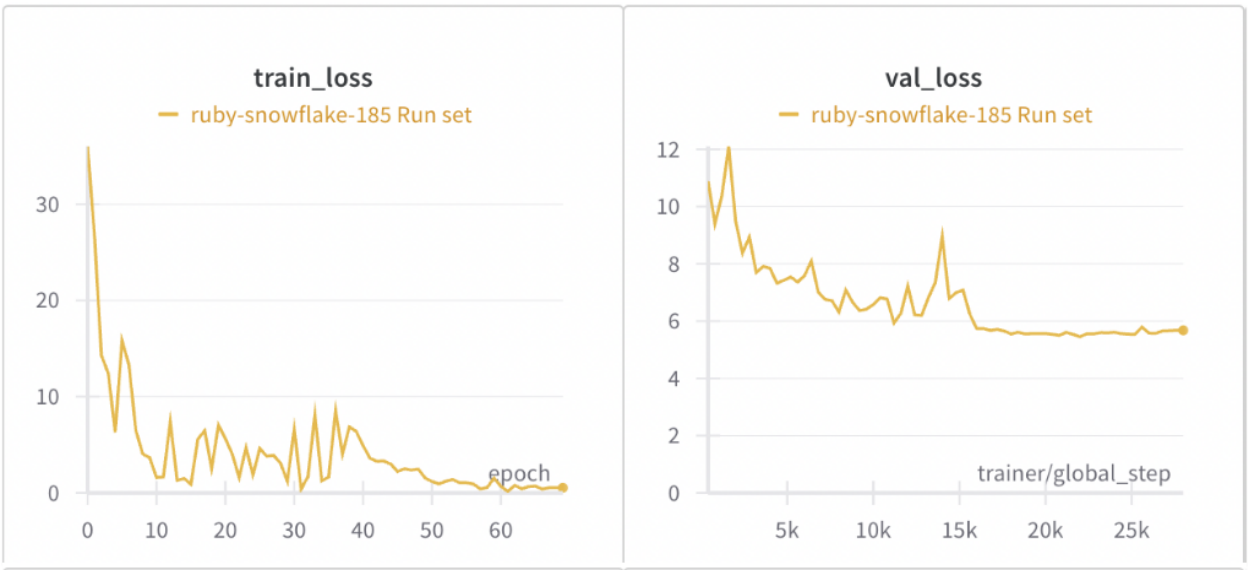


Figure 3: Enformer Training Results on "Good" Data

Figure 4: Enformer Training Results on Arbitrary Data

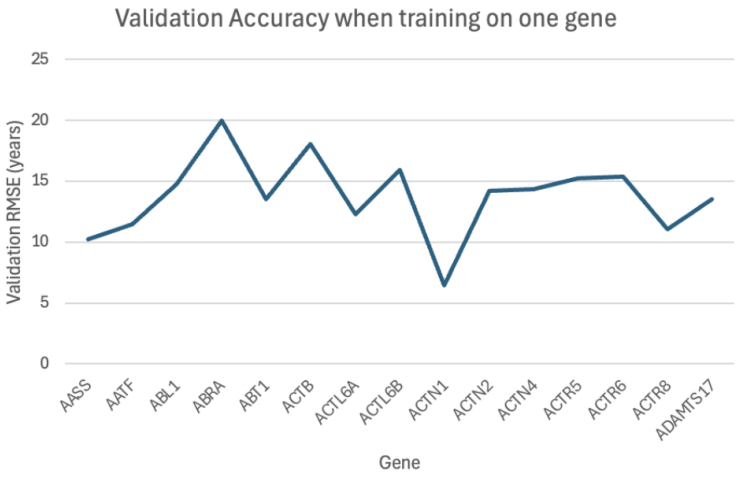
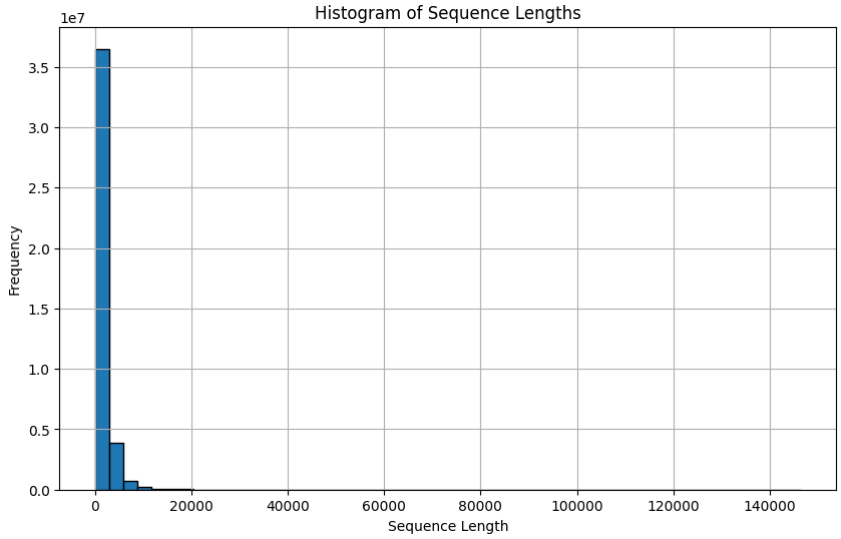


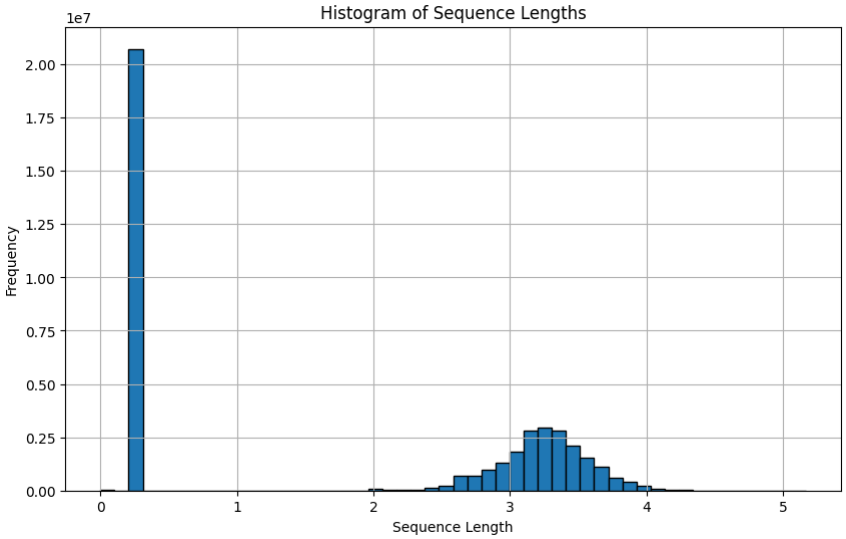
Figure 5: Enformer Training Per-Gene Results

Ultimately, the Enformer model has shown the best performance, so we will use it moving forward. Given our failures to analyze larger quantities of data, however, we have to re-evaluate our data pipeline and approach to training.

Weeks of: October 1st - October 25th

The focus of this period was to retrace our data collection (from months and months ago) to ensure it's validity, recompile and trim data where necessary and devise a modeling approach in which we could effectively train on these smaller, trimmed datasets. Our first task was to trim our cumulative data. Given the memory issues we encountered earlier, as well as the performance issues incurred by pre-processing extremely long DNA sequences before training a model, we wanted to trim data with outlying sequence lengths. To define a valid range of sequence lengths, we collected metrics on the distribution of sequence lengths across all gene data:





Given the massive range of sequence lengths, we decided an appropriate range of sequence lengths to incorporate in analysis was between 100 (inverse log10(2)) and 31622 (inverse log10(4.5)), where the vast majority of the distribution falls. With that, we decided to trim data such only we only kept sequences with lengths in this range and that were annotated as "one2one" (given our gene annotations come from TOGA, a ML classifier, we want sequences that are most likely to be orthologous). Below you can see the code used for one of our trimming methods (this chunk also outputs some stats on the trimmed data).

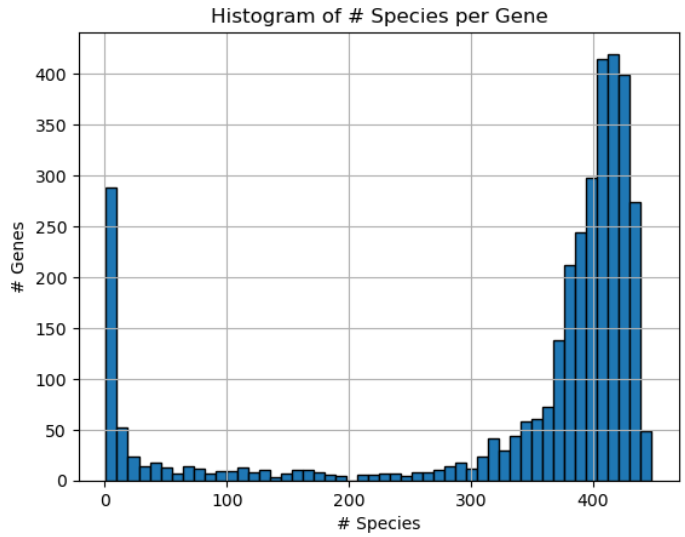
```
# Demo Script for trimming data
"""
method to trim a given gene's csv file path
```

```
    (expected format:
    ['organism','max_lifespan', 'gene_id',
    'orthologType','chromosome','start','end',
    'direction (+/-)','intactness','sequence'])
    generates new file only including one2one sequences btwn lengths of 104 and 31620
    computes stats of # of sequences and # of species represented per gene
    """
    def create_one2one_gene_sets():
        gene_stats = {} #dict to hold gene:(num_data_entries, num_species) pairs
        num_genes = 0
        for file in os.listdir(gene_datasets_path):
            file_path = gene_datasets_path + file
            new_file_path =
              "/".join(file_path.split("/")[:-2]) +
              "/regulatory_one2one/" + "".join(file.split(".")[:-1]) +
              "_one2one.csv"
            #only execute on most up-to-date 'trimmed' gene files
            if file_path[-21:] != "orthologs_trimmed.csv": continue
            num_genes += 1
            cur_gene = file.split("_")[0]
            num_data = 0
            organisms = set()

            with open(new_file_path, "w") as write_to:
                writer = csv.writer(write_to)
                writer.writerow(
                  ['organism','max_lifespan', 'gene_id',
                  'orthologType','chromosome','start','end',
                  'direction (+/-)','intactness','sequence']
                )
                with open(file_path) as read_from:
                    for line in read_from:
                        line = line.split(",")
                        if len(line) < 10: continue
                        seq = line[-1].strip()
                        length = len(seq)
                        if length < 104 or length > 31620:
                          continue #only include seqs of length in this range
                        ortholog_type = line[3]
                        if ortholog_type != "one2one":
                          continue #only include one2one seqs
                        num_data += 1
                        print(line[0])
                        organisms.add(line[0])
                        writer.writerow(line)

            #fill dict entry for current gene
            gene_stats[cur_gene] = (num_data, len(organisms))

            write_to.close()
            os.system(f'rm -rf {file_path}')

        # after iterating thru all genes s.t we've generated one2one, trimmed, gene sets
```

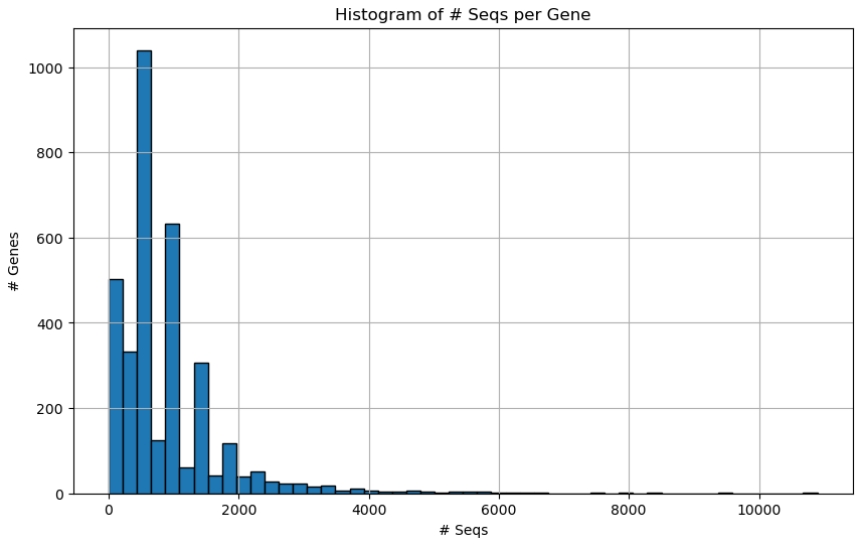
```
    # create new file that outputs stats per gene
    # (csv of format [gene | num_entries | num_species])
    # we can use this file to generate histograms thereafter

    with open(
      "/data/rbg/users/wmccrthy/chemistry/Everything/ +
      "EDA/regulatory_one2one_sets_metadata.csv",
      "w") as write_to:
        writer = csv.writer(write_to)
        writer.writerow(["gene", "# seqs", "# species"])
        for gene in gene_stats:
            num_seqs, num_species = gene_stats[gene]
            writer.writerow([gene, num_seqs, num_species])
    write_to.close()
```

For the time being, we want to experiment with training on individual genes (which can be individually pre-processed and tokenized) to avoid the memory issues we were encountering earlier. Thereafter, we will circle back to training on larger datasets. Once our data was trimmed according to the above constraints, we collected the following to inform this training:

- distribution of number of species represented in each gene dataset



- distribution of number of sequences in each gene dataset



Given the distribution of species in each gene dataset, we decided it makes sense to train on gene sets with at least 400 species represented.

Weeks of: October 28th - December 1st

The focus of this period was to implement, debug, and run scripts to train Enformer model on each of our gene datasets, evaluate results, and move forward accordingly.

First Training Run Model Overview (tag: default)

- We input tokenized sequences (all padded to length 31620)
- Pass tokenized sequences to pre-trained Enformer and pull out sequence embeddings
- Pass embeddings through two linear layers (no dropout)
- Output lifespan prediction; evaluate loss accordingly
- 10 epochs per gene
- batch size = 1

For this initial training run, we trained on only genes with >=400 species represented in their set. For a sample of initial results, see below.

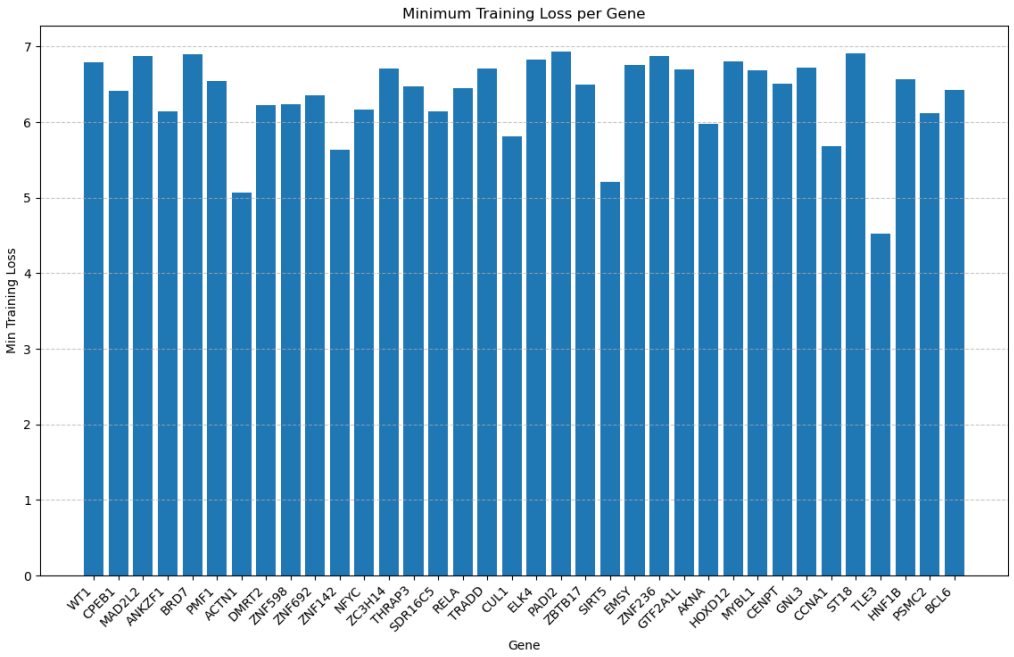


Figure 6: Minimum Training Loss by Gene

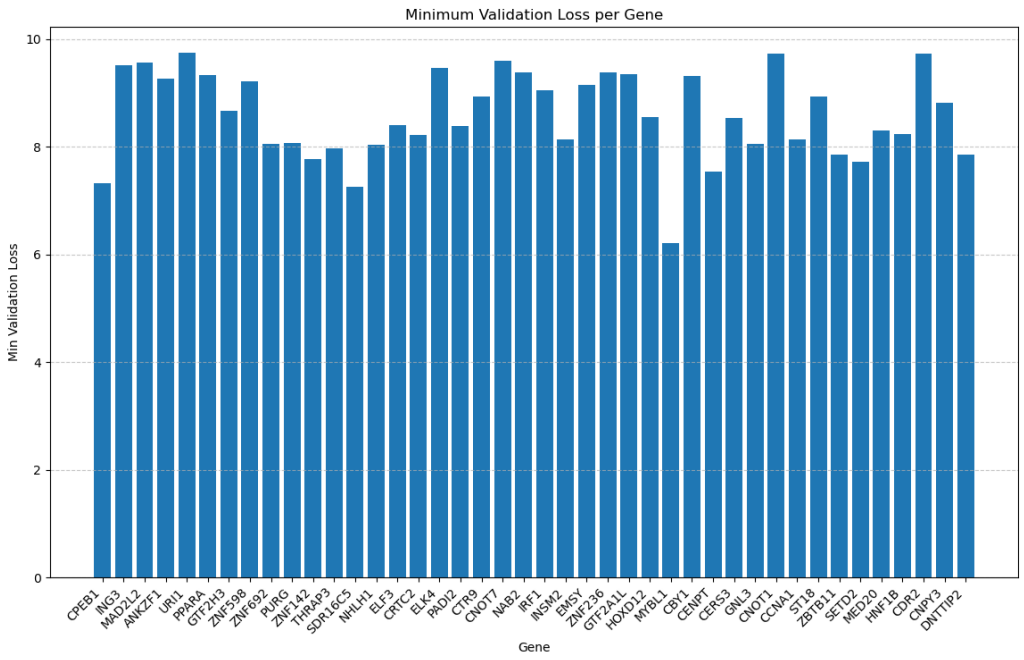


Figure 7: Minimum Validation Loss by Gene

The results above are limited to the "best" performing genes; that is, the genes for which the minimum validation loss was < 10. Given we were only able to train on a sample of our gene datasets, it is encouraging to see that the model learns (relative to our prior results) somewhat well on many of these datasets. In some cases, like on MYBL1, the model exhibits some of the best performance we've seen so far. Given how MYBL1 stands out as the best performing gene in this initial set of results, we will use it to benchmark the performance of other model configurations.

Second Training Run Model Overview (tag: frozen_enformer)

- We input tokenized sequences (all padded to length 31620)
- Pass tokenized sequences to pre-trained Enformer and pull out sequence embeddings
- Pass embeddings through two linear layers (dropout with p = 0.1 on each linear layer)
- Output lifespan prediction; evaluate loss accordingly
- For this run, we froze the enformer layer (it's weights won't update), added early stopping (if val loss plateaus for 3 epochs), added dropout to the final two linear layers and tweaked LR (1e-3 instead of 1e-4)
- 10 epochs per gene
- batch size = 1

The key difference in this second run is that we froze the enformer layer and added dropout; based off the preliminary results, this seems to hurt the model's performance quite significantly. In some cases, it helps to avoid overfitting but at the cost of the model's ability to learn (generally higher loss in training and validation sets). We observe a few examples of this below, where for one gene, SIRT5, we see improved validation loss with slightly higher training loss, but for another gene, CERS3, we see much worse validation and training loss:

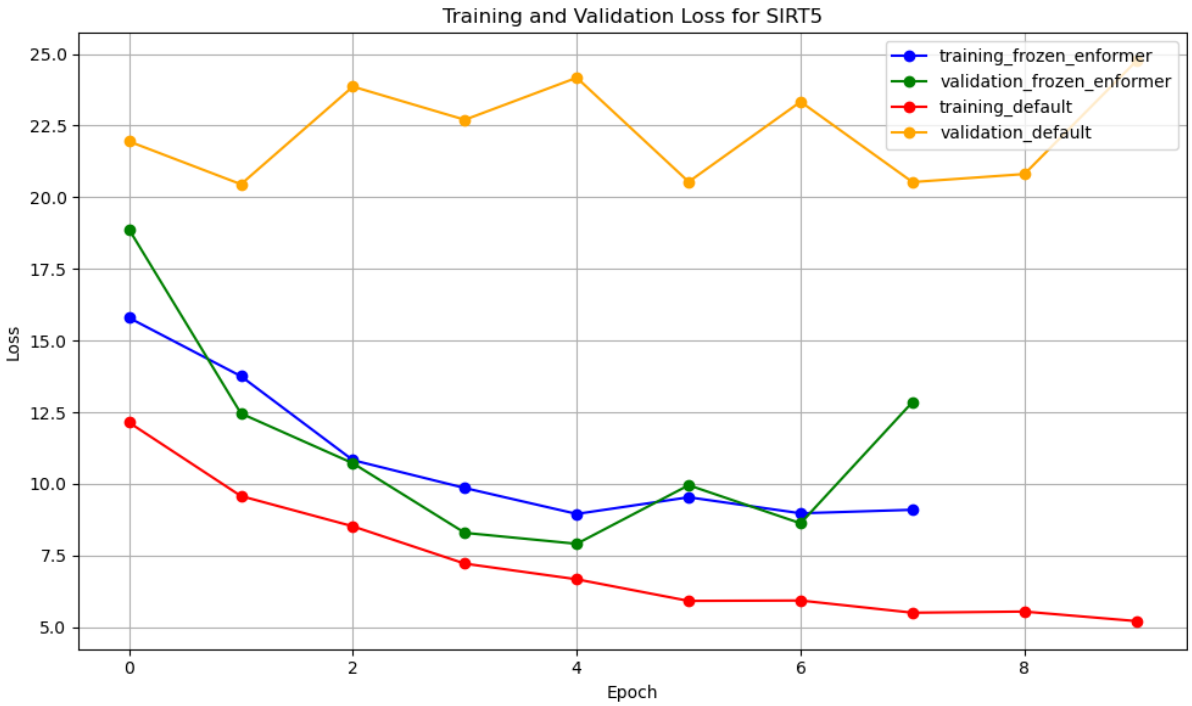


Figure 8: SIRT5 Performance Comparison

Figure 9: CERS3 Performance Comparison

Given how poorly the model was performing in it's initial training, we stopped the run before it could train on what is meant to be our benchmark gene, MYBL1. In any case, back to the drawing board...

Third Training Run Model Overview (tag: pt2)

- We input tokenized sequences (all padded to length 31620)
- Pass tokenized sequences to pre-trained Enformer and pull out sequence embeddings
- Pass embeddings through two linear layers (dropout with p = 0.1 on each linear layer)
- Output lifespan prediction; evaluate loss accordingly
- For this run, we unfroze the Enformer layer and kept everything else the same as prev run
- 10 epochs per gene
- batch size = 1

This run had the worst performance yet. This suggests that the freezing of the Enformer layer was not the primary cause of worse performance. Considering the other differences between this model configuration and that of our best model so far (the first run), the addition of dropout stands out as a potential concern. Generally, dropout should help with generalization and model performance, but in our case, I'm concerned that it might be causing problems since we apply at the final two linear layers (from which the output is directly derived); as such, the model might not be able to correct for errors induced by dropout before returning a final output.

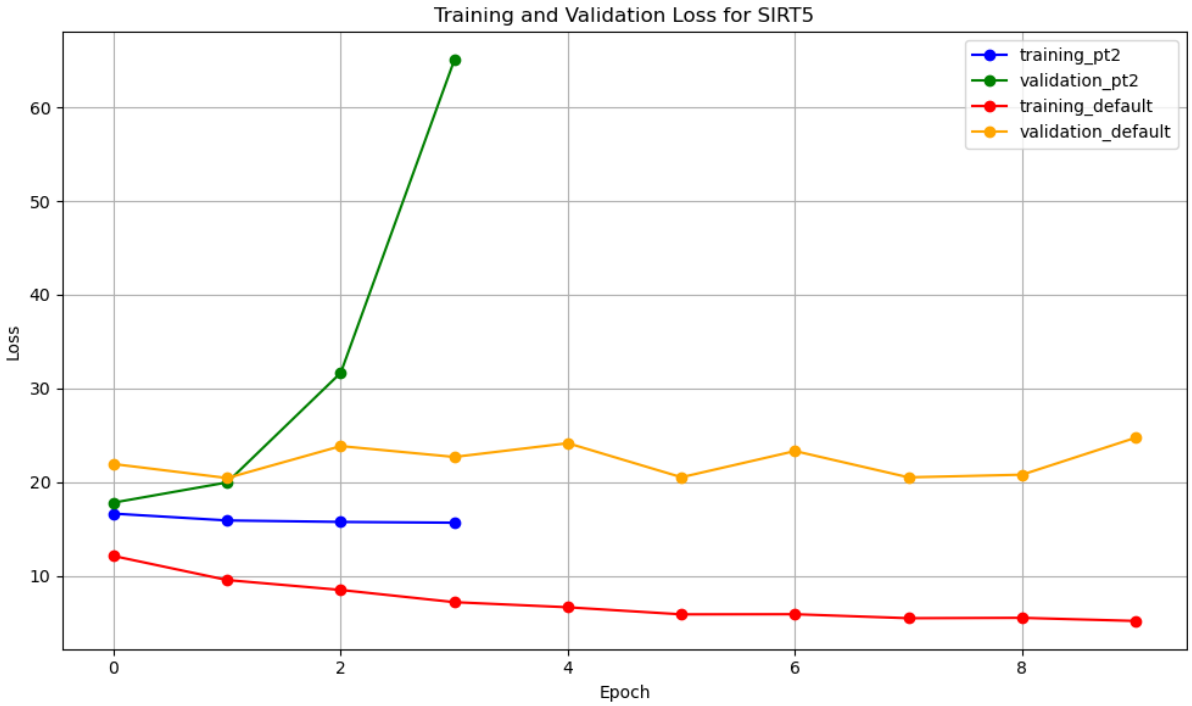


Figure 10: SIRT5 Performance Comparison

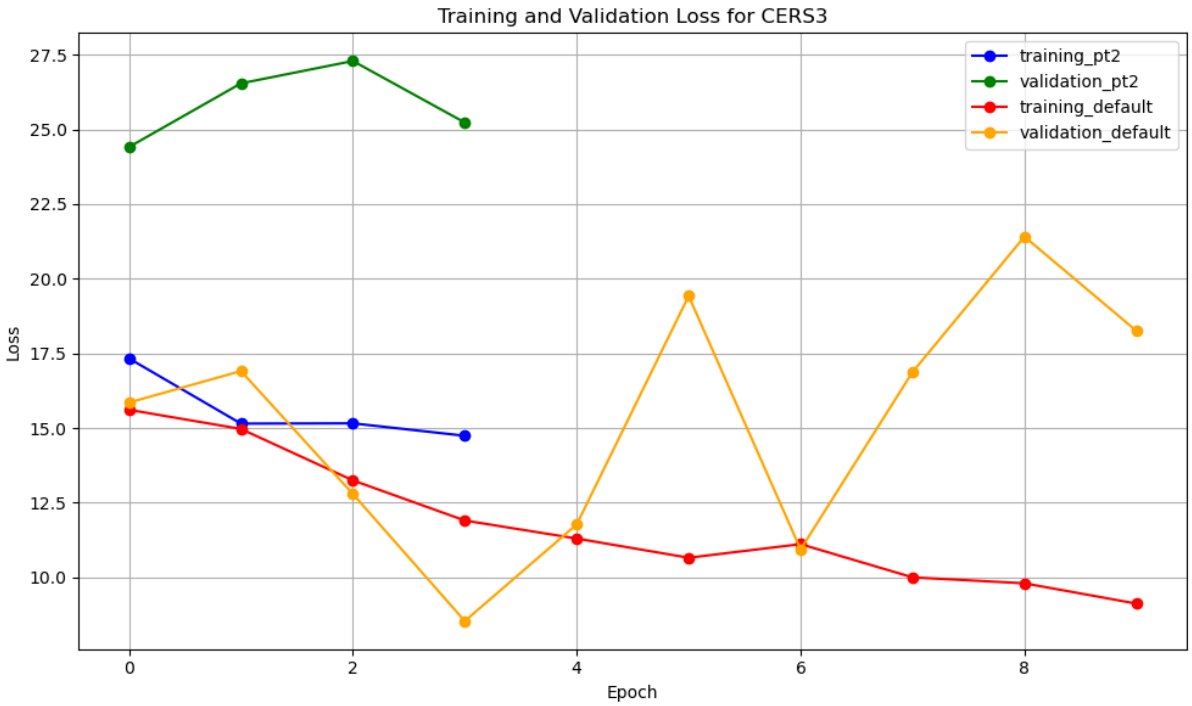


Figure 11: CERS3 Performance Comparison

For many genes during this run, training stopped prematurely as val loss failed to improve with additional epochs. Similarly with the previous run, performance was generally so poor that we stopped the run before it could train on most genes (i.e. MYBL1).

Fourth Training Run Model Overview (tag: pt3)

- We input tokenized sequences (all padded to length 31620)
- Pass tokenized sequences to pre-trained Enformer and pull out sequence embeddings
- Pass embeddings through one linear layer (no dropout) and one MLP layer (implementation below)

```python
import torch
from torch import nn
# MLP Layer
class MLP(nn.Module):
    def __init__(self, input_dim, hidden_dim, output_dim, dropout_prob=0.1):
        super(MLP, self).__init__()
        self.fc1 = nn.Linear(input_dim, hidden_dim)
        self.dropout = nn.Dropout(dropout_prob)
        self.fc2 = nn.Linear(hidden_dim, output_dim)

    def forward(self, x):
        x = F.relu(self.fc1(x))
        x = self.dropout(x)
        x = self.fc2(x)
        return x
```

- Output lifespan prediction; evaluate loss accordingly
- Enformer unfrozen, lr starts at 1e-4,
- 10 epochs per gene

- batch size = 1

This models performance is very similar to the first variant we tried (denoted as default in most comparisons). There is some variation in this approaches performance, where some genes have higher loss than in the first variant, while others have lower loss, as shown in the results below.

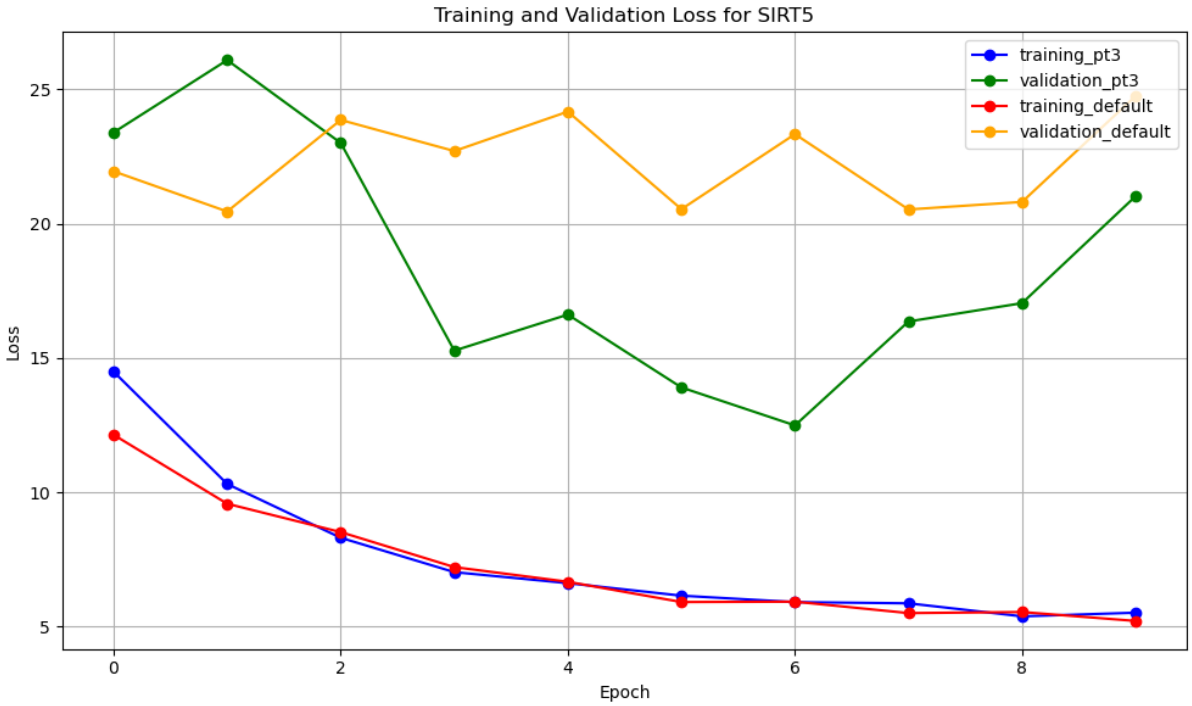


Figure 12: SIRT5 Performance Comparison

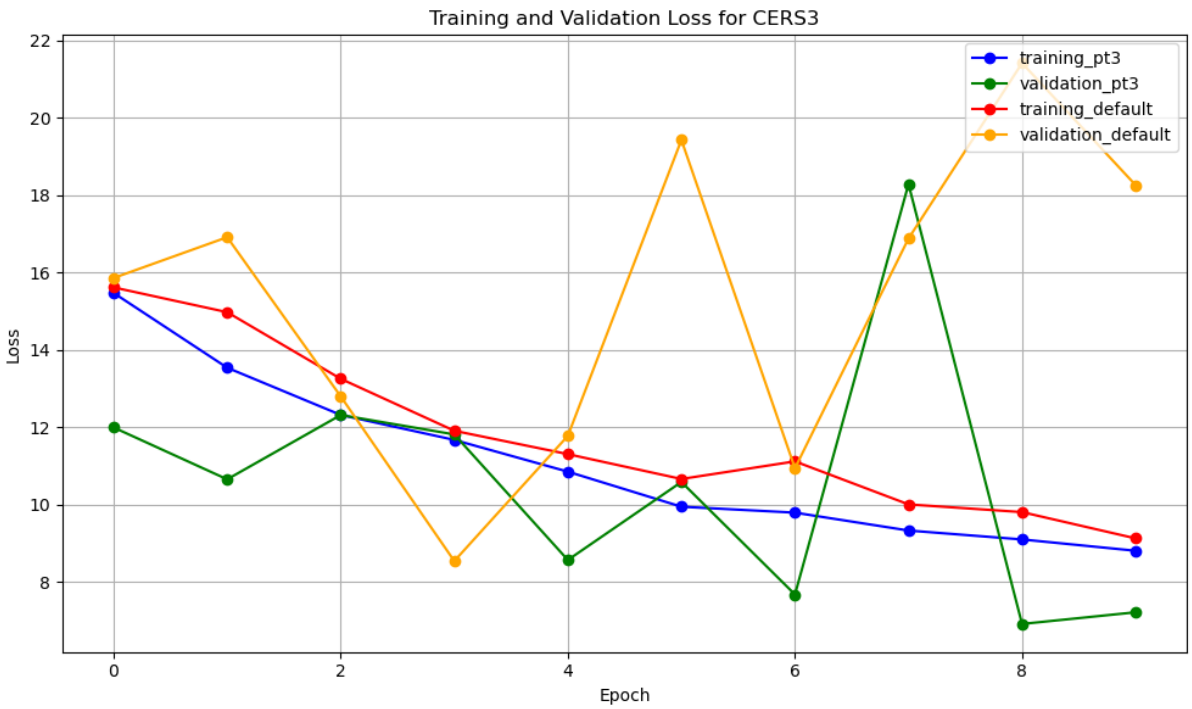


Figure 13: CERS3 Performance Comparison

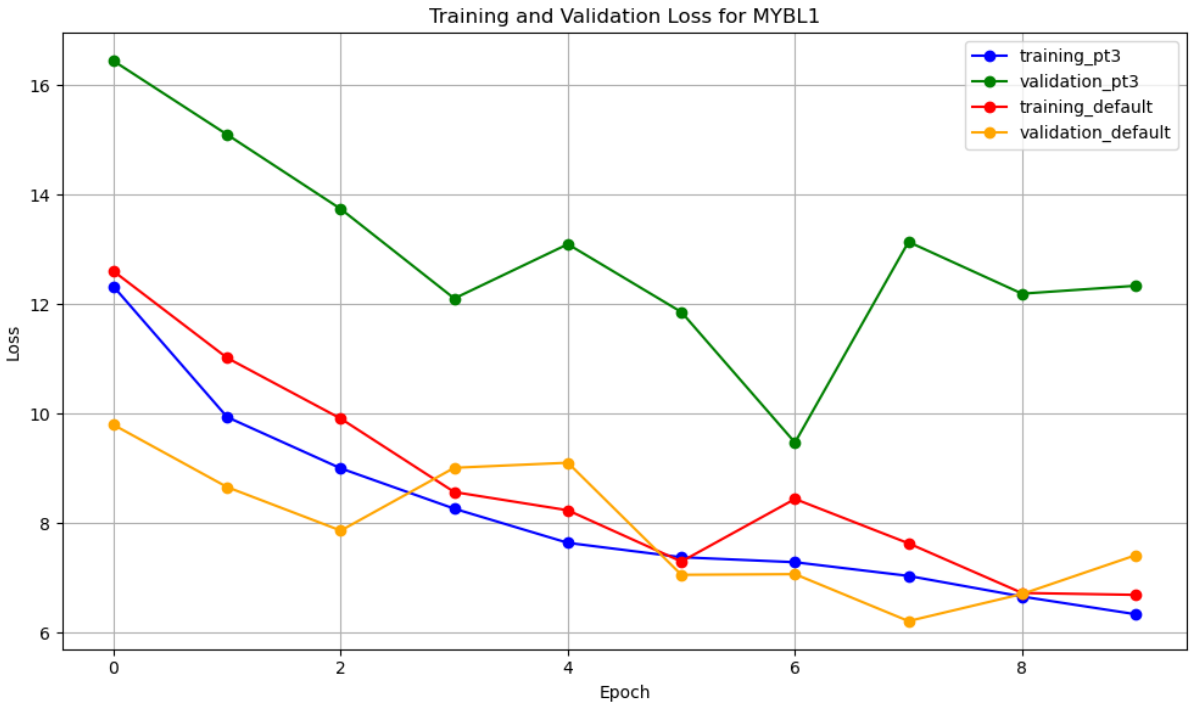


Figure 14: MYBL1 Performance Comparison

Training loss appears to be very similar between these two best performing model configurations (this variant, denoted pt3, and the first variant, denoted default), while validation loss is generally more variant. In any case, the model is clearly overfitting so our next revision seeks to improve in that regard.

Fifth Training Run Model Overview (tag: pt4)

- We input tokenized sequences (all padded to length 31620)
- Pass tokenized sequences to pre-trained Enformer and pull out sequence embeddings
- Pass embeddings through one linear layer (dropout w/ p = 0.1) and one MLP layer (implementation changes shown below)
  - adding additional hidden/linear layer to our MLP

```python
import torch
from torch import nn
# MLP Layer
class MLP(nn.Module):
    def __init__(
        self, input_dim, hidden_dim1, hidden_dim2, output_dim, dropout_prob=0.1
    ):
        super(MLP, self).__init__()
        self.fc1 = nn.Linear(input_dim, hidden_dim1)
        self.dropout1 = nn.Dropout(dropout_prob)
        self.fc2 = nn.Linear(hidden_dim1, hidden_dim2)
        self.dropout2 = nn.Dropout(dropout_prob)
        self.fc3 = nn.Linear(hidden_dim2, output_dim)

    def forward(self, x):
        x = F.relu(self.fc1(x))  # First hidden layer
```

```
        x = self.dropout1(x)
        x = F.relu(self.fc2(x))  # Second hidden layer
        x = self.dropout2(x)
        x = self.fc3(x)  # Output layer
        return x
```

- Output lifespan prediction; evaluate loss accordingly
- Enformer unfrozen, lr starts at 1e-4,
- 10 epochs per gene
- batch size = 1

This model configuration exhibits generally similar performance in terms of best validation loss/training loss as the default and pt3 variations; notably however, this variant performed well on different genes. That is, the best performing genes with this model had similarly low loss as the best performing genes with other configurations (pt3, default), but the two sets of genes were very different. An example of this is MYBL1, which performed poorly on this run.

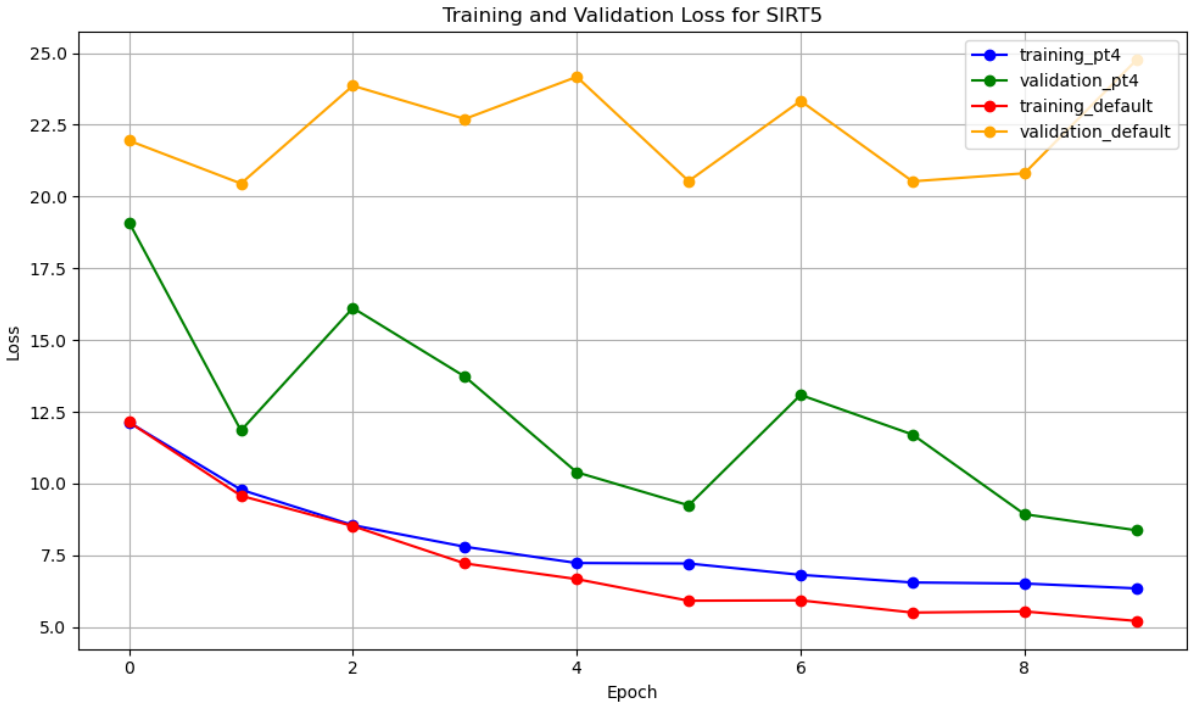


Figure 15: SIRT5 Performance Comparison

Figure 16: CERS3 Performance Comparison

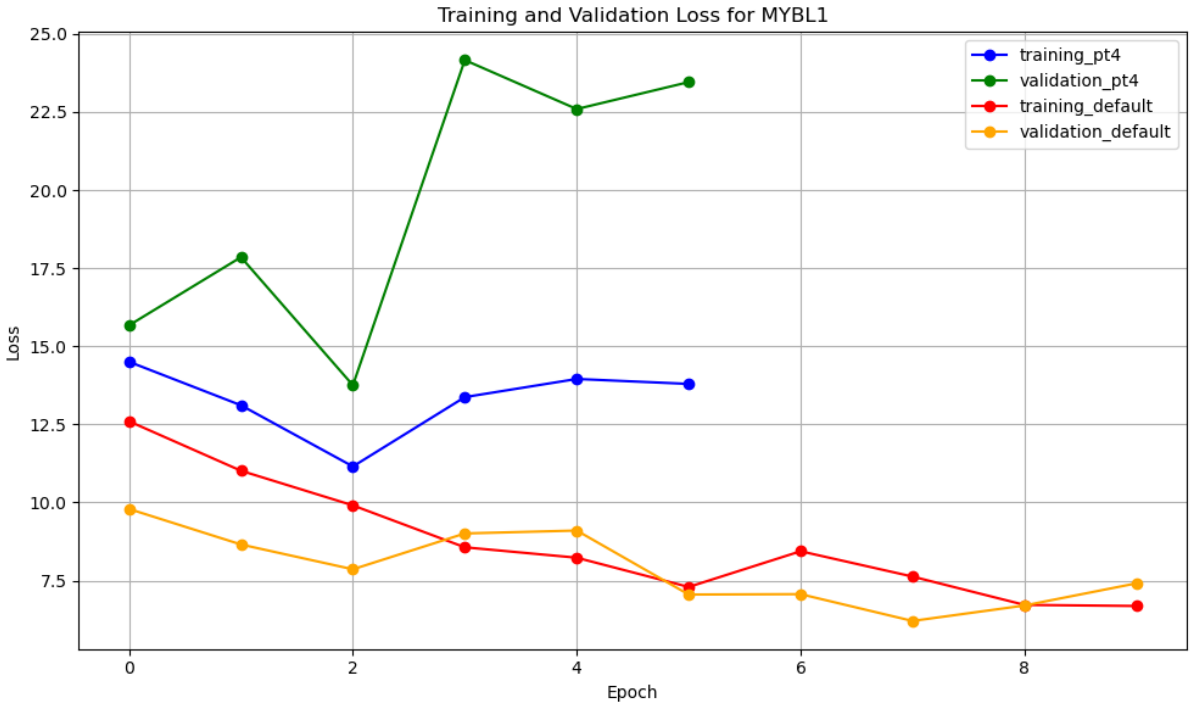


Figure 17: MYBL1 Performance Comparison

Sixth Training Run Model Overview (tag: pt5)

- We input tokenized sequences (all padded to length 31620)
- Pass tokenized sequences to pre-trained Enformer and pull out sequence embeddings

- Pass embeddings through one linear layer (modified dropout to p = 0.05) and one MLP layer
  - modifying dropout in MLP layer to p = 0.05 instead of 0.1
- Output lifespan prediction; evaluate loss accordingly
- Enformer unfrozen, lr starts at 1e-3 **(increased from 1e-4 on prev runs)**
- 10 epochs per gene
- batch size = 1

The model performs very poorly! Seems that any time we using learning rate >= 1e-3, the model's learning suffers. It is not displaying line plots of gene performance given how poor the results were, so please see the top 3 performing genes' stats below for context.

gene	avg_train_loss	min_train_loss	avg_val_loss	min_val_loss
LBH	15.15	14.61	18.90	14.03
ZNF618	15.52	15.52	14.21	14.21
GTF2H3	15.41	14.90	17.41	14.29

Seventh Training Run Model Overview (tag: pt6)

- We input tokenized sequences (all padded to length 31620)
- Pass tokenized sequences to pre-trained Enformer and pull out sequence embeddings
- Pass embeddings through one linear layer (dropout p = 0.05) and one MLP layer (dropout p = 0.05)
- Output lifespan prediction; evaluate loss accordingly
- Enformer unfrozen, lr starts at 1e-4 and can decrease to 1e-6
- 10 epochs per gene
- batch size = 1
- changed early stopping s.t we stop after 4 epochs of plateau

This model configuration performed similarly well to the default (baseline) and pt4 model variants, our two best performing prior to this run. In some cases, this model was not able to fit as well to genes as the other two variants, but it generally seems to overfit less and has produced the lowest average validation loss across genes so far. See below for examples that contrast this model's performance on two of our benchmark genes (SIRT5 and CERS3) and a couple other interesting cases.

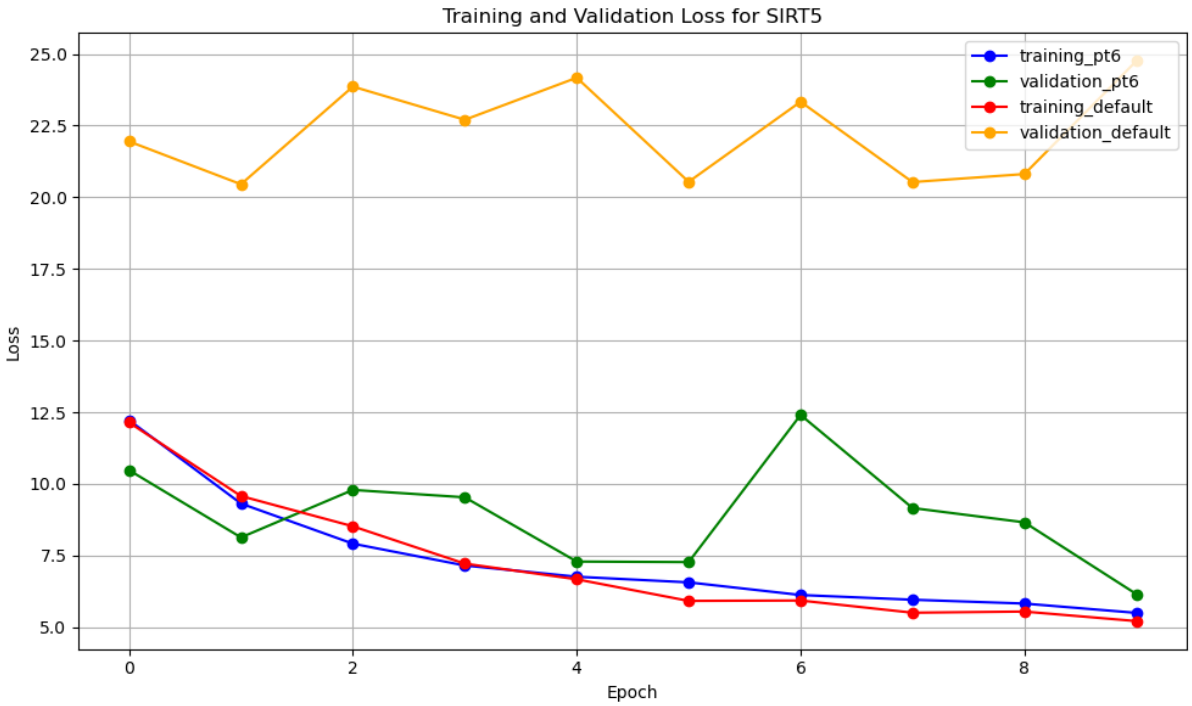


Figure 18: SIRT5 Performance Comparison

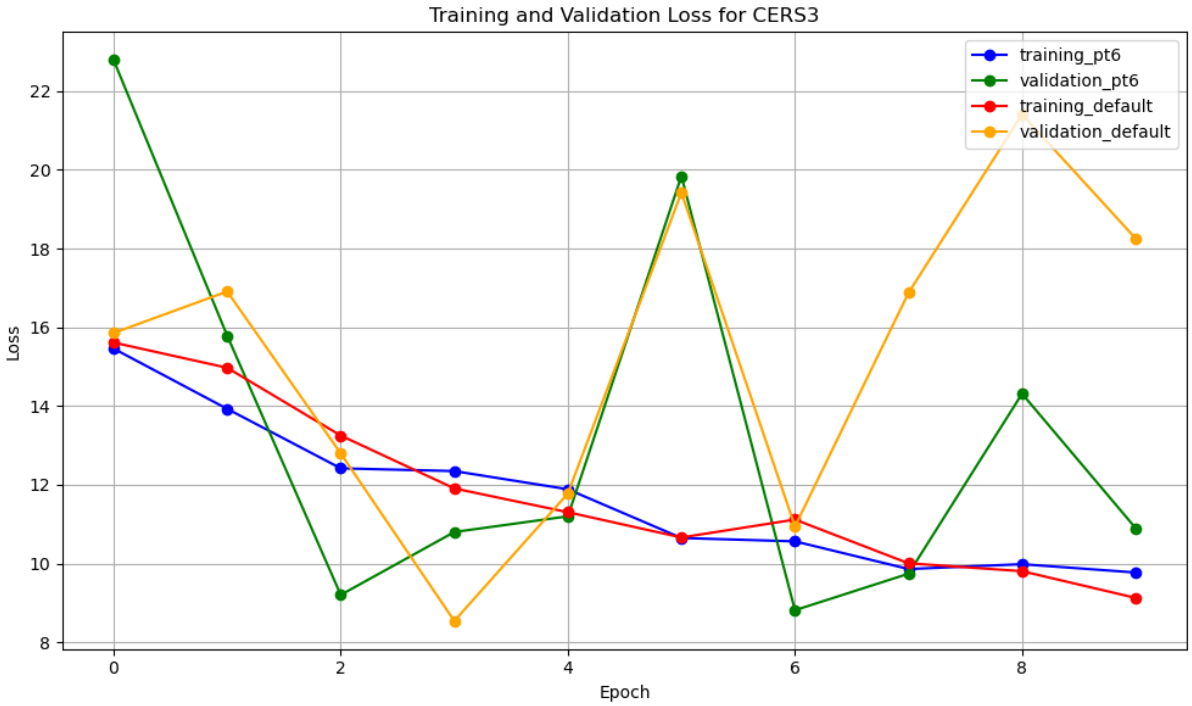


Figure 19: CERS3 Performance Comparison

On SIRT5 and CERS3 (which we have used to benchmark each model's performance), our pt6 variant out-learns the default (previously best performing) variant.

On MYBL1, another gene we've used for benchmarking, this model configuration has similar training loss and slightly worse validation loss than the default configuration.

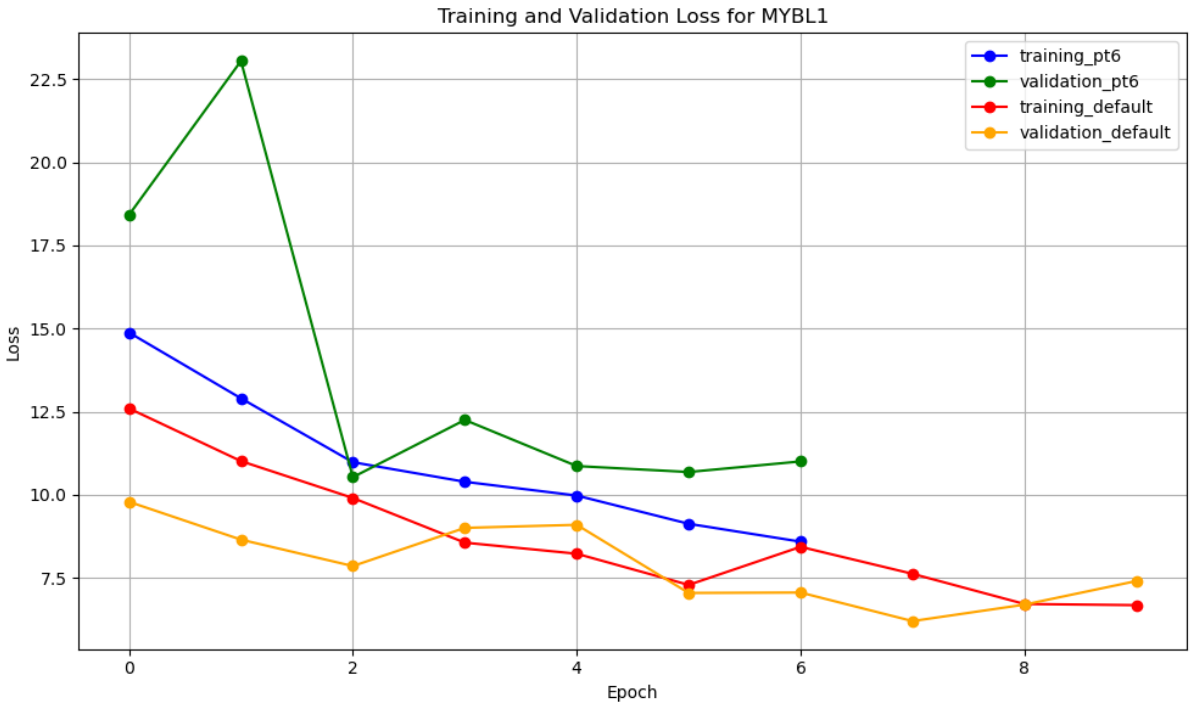


Figure 20: MYBL1 Performance Comparison

The best performing gene on this run was LPIN3, for which validation loss was very similar to training loss (indicating the model is not overfitting as badly) and minimum validation loss was near 5.

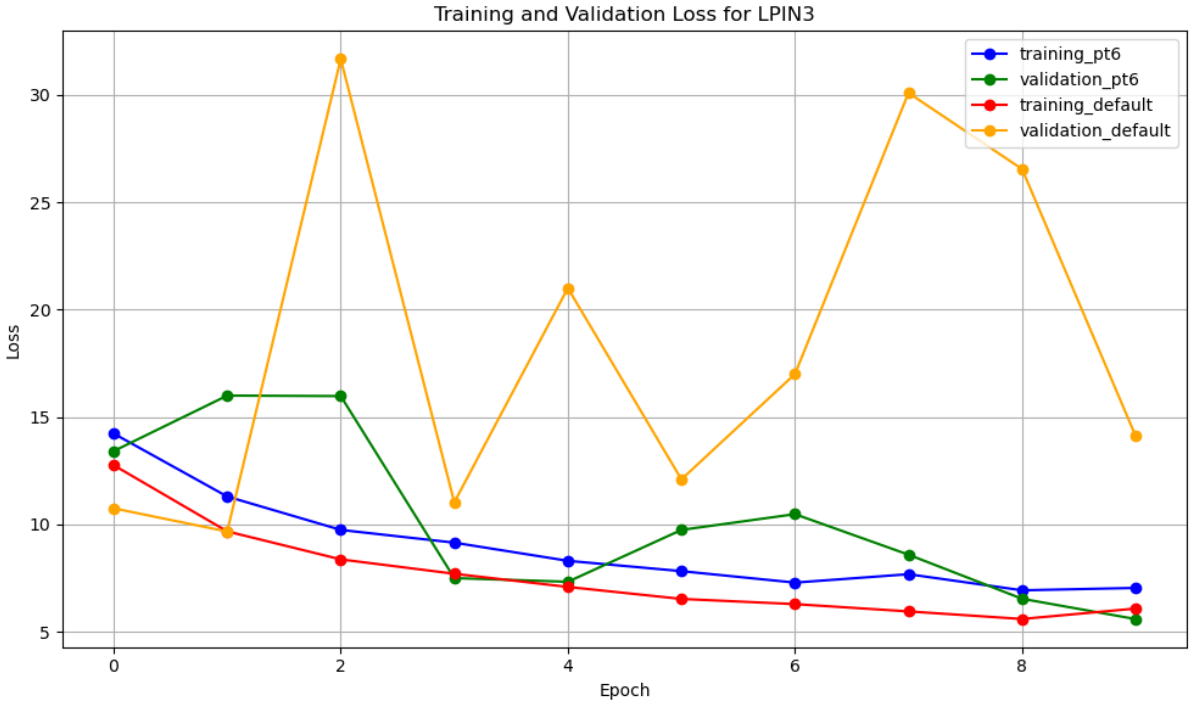


Figure 21: LPIN3 Performance Comparison

Summary of Various Models

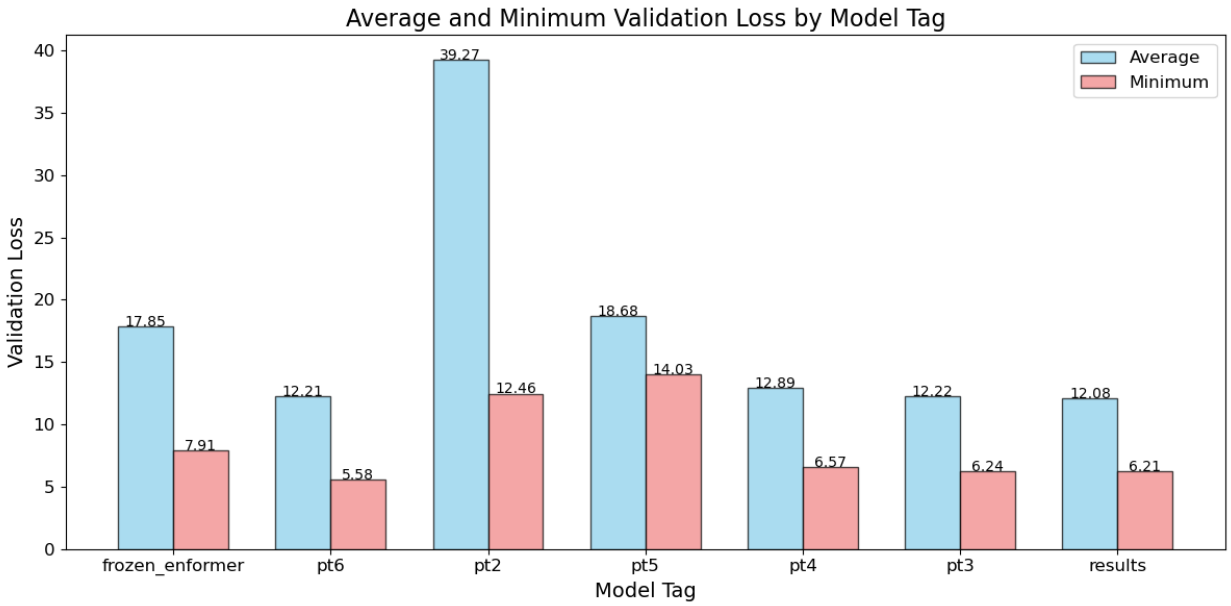


Figure 22: Model Performance Comparison (note: `results` corresponds to the `default` model config)

Some Key Observations

1. `pt2`:
   - Exhibits the **highest average validation loss** at **39.27**, suggesting poor overall performance and instability.
   - However, its minimum loss of **12.46** indicates occasional good results.
2. `pt6`:
   - Achieves the **lowest minimum validation loss** of **5.58**, making it the best-performing model for minimum error.
   - Its average loss of **12.21** demonstrates consistent performance across runs, essentially matching the other strong performing models.
3. `pt4`, `pt3`, **and** `results/default`:
   - These models perform very similarly, with average losses between **12.08** and **12.89**, and minimum losses around **6.2** to **6.57**.
   - This indicates these models are both consistent and competitive in performance.
4. `frozen_enformer`:
   - While its **average loss** is higher at **17.85**, it achieves a decent minimum loss of **7.91**.
   - This suggests occasional good runs but overall less stability compared to models like `pt6`.
5. `pt5`:
   - Shows an average loss of **18.68**, which is higher compared to other models except for `pt2`.
   - Its minimum loss of **14.03** indicates relatively poorer performance compared to the best models.

Summatively:

- `pt6` is the best-performing model based on its lowest minimum validation loss.
- `pt2` struggles significantly with stability, as evidenced by its high average loss.
- Models like `pt6`, `pt4`, `pt3`, and `results/default` provide a strong balance between low average loss and minimum loss, making them reliable options for consistent validation performance. Given that `pt6` essentially matches the best average validation loss and has the lowest minimum validation loss, this is the configuration we will use for the time being as we proceed.

Weeks of: December 1st - Next Steps

As we approach the semester's end, it is time to evaluate potential next steps.

Recall, the longer term objective is to perform what we've denoted as the "outlying species test". As touched on earlier in this log, this test will involve evaluating the model on DNA from a species whose lifespan is uncharacteristically high compared to their genus/family (taxonomic classifications that indicate genetic similarity). If our model is able to predict somewhat accurately under these conditions, we are hopefully onto something.

In any case, to arrive at this goal, we want to continue to tune and test the model. Recall that thus far, most of our training has been on DNA from individual genes. As we've seen, the strongest model is able to perform somewhat well, achieving a minimum validation loss <6, for a plethora of these genes. To move towards our goal of the outlying species test, however, we now want to train the model on DNA from a collection of genes rather than individual genes. While the model has shown strong performance on single genes, there may be underlying patterns which it is able to pick up when learning from DNA that belongs to a collection of different genes; hopefully, these could further enhance performance.

Under this approach, we want to amalgamate data from various genes, then test the model on data from a single gene that is left out from the training data. We think this will be a good "stress" test for the model, and will allow us to move forward accordingly. See below for a demo method that implements this procedure (and for the rest of the code, check out the project GitHub).

```
def train_amalgamation_of_genes(train_genes, test_genes):
# what we could do is go through genes with decent val loss in prior tests
# sample random set of k of those genes for training
# sample one random for testing
    results_path = training_dir + "per_gene_results_pt6.csv"

    to_sample = []
    with open(results_path) as read_from:
        for line in read_from:
            line = line.split(",")
            if line[0] == "gene": continue #skip first line
            avg_val, min_val = line[2:]
            gene = line[0].strip()
            avg_val, min_val = int(avg_val), int(min_val)
            if avg_val < 11 and min_val < 7.5: to_sample.append(gene)

    train_genes = random.sample(to_sample, 10)
    test_gene = random.split(to_sample, 1)
    training_inps, training_labels = [], []
    # iterate thru and combine datasets into single list
    for gene in train_genes:
        gene_path = gene_one2one_datasets_path +
                    gene +
                    "_orthologs_trimmed_one2one.csv"
        gene_inps, gene_labels = prepare4Enformer(gene_path, UNIVERSAL_MAX_LEN)
        training_inps.extend(gene_inps)
        training_labels.extend(gene_labels)
    # shuffle training dataset
    indices = np.arange(len(training_labels))
    np.random.shuffle(indices)
    shuffled_training_inps = [training_inps[i] for i in indices]
    shuffled_training_labels = [training_labels[i] for i in indices]
    # create validation set
```

```
    val_gene_path = gene_one2one_datasets_path +
                    test_gene[0] +
                    "_orthologs_trimmed_one2one.csv"

    val_inps, val_labels = prepare4Enformer(val_gene_path, UNIVERSAL_MAX_LEN)

    # call train on the combined data
    train(
        shuffled_training_inps,
        shuffled_training_labels,
        "gene_amalgamation",
        -1,
        UNIVERSAL_MAX_LEN,
        "_".join(train_genes) + f'_{test_gene[0]}',
        (val_inps, val_labels)
    )
```

Ideally, we can scale this approach to train the model on an amalgamation of data from all our genes before we test it with the outlying species test. With that being said, we must first test the approach on smaller sets of genes and tune the model accordingly. Additionally, we still have some concerns with memory, given how dense a of data from *all our genes* would be. As such, iteratively scaling this approach should allow us to simultaneously debug memory issues and tune the model. Stay tuned!