MISTY GROUP

Mohamed Abdelaziz	k12137202
Lydia Mayer	k11904969
Ivan Drinovac	k12104744

Birds Audio Classification

Machine Learning & Pattern Classification [UE]

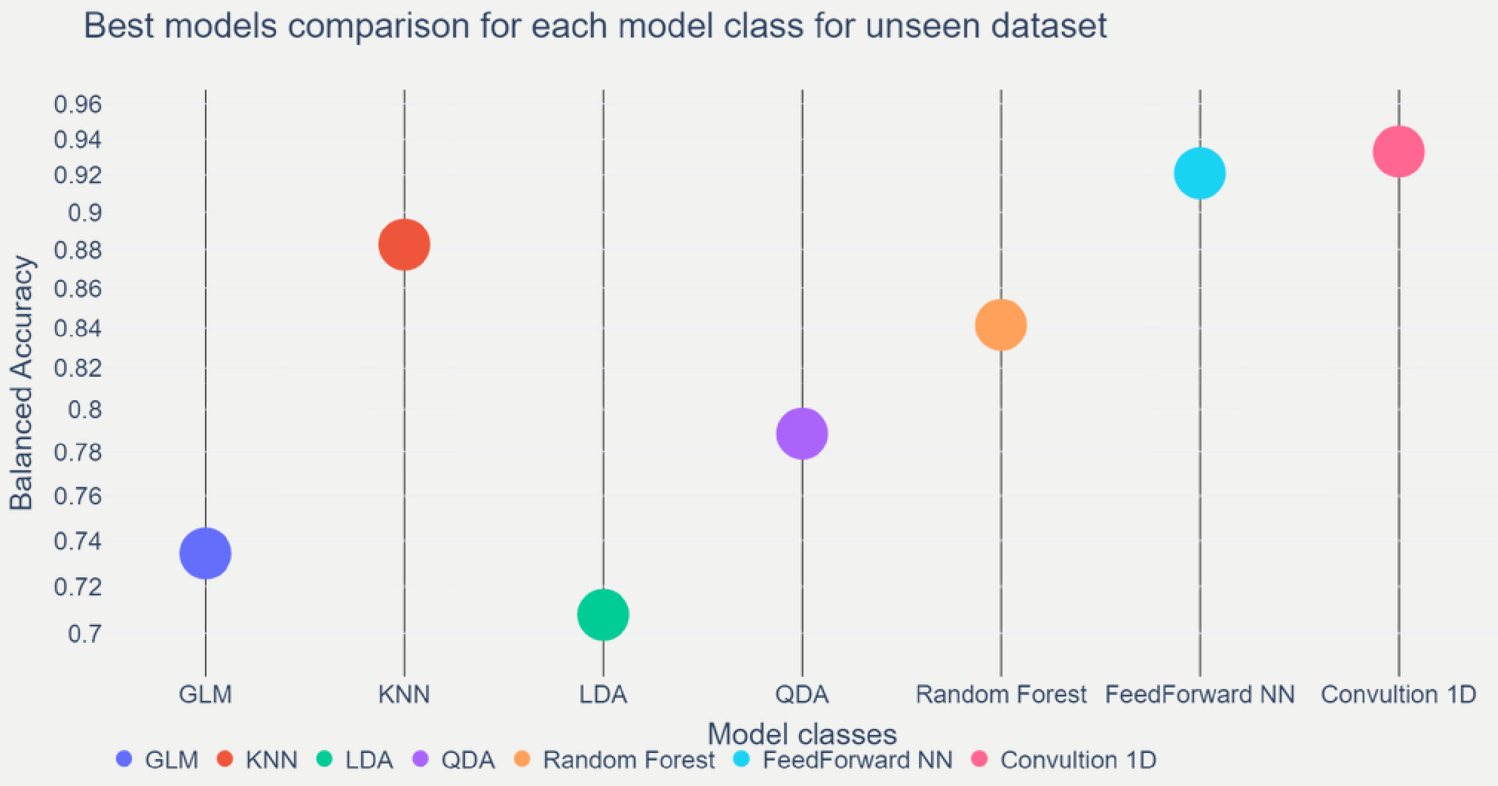
CHALLENGE TASK

Table of Contents

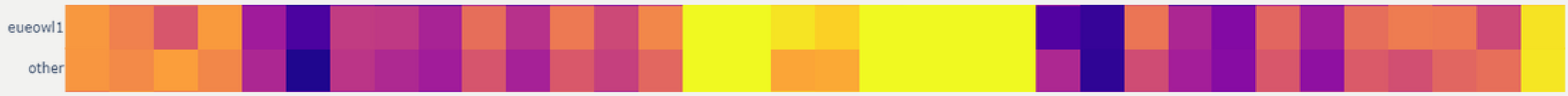
1. Introduction

Dataset Characteristics:

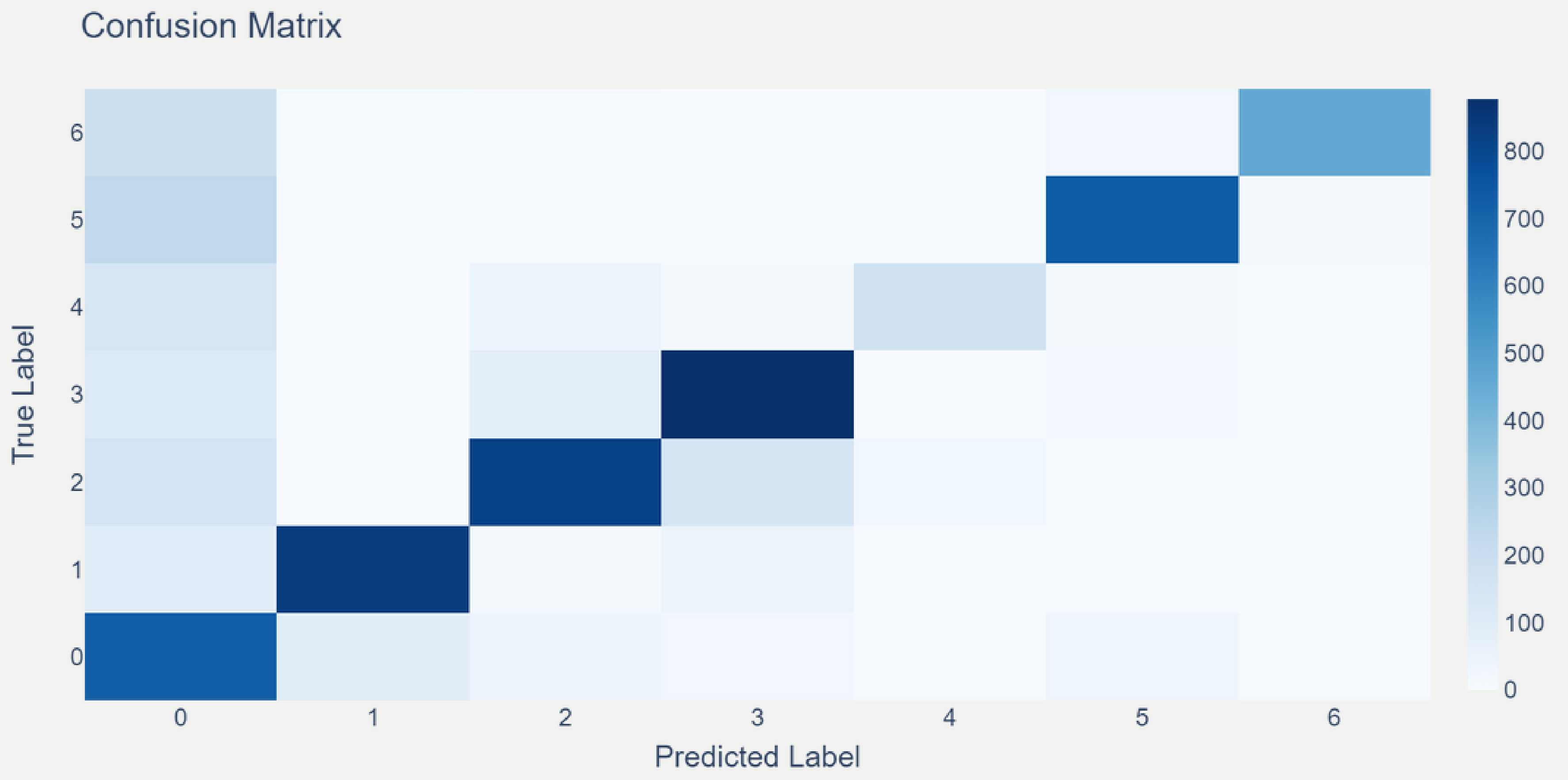
- High inter-correlation between different features, within the same feature-set, was observed, which lead to the assumption of a superior performance for model classes which utilize the inter-correlations between different features, e.g. Convolutional Neural Network, Long-Short-Term Memory, .. etc.
    - Previously submitted experiments supported this assumption shown in the graph.



- As mentioned in the previous submissions, we encountered some classes, "other" and "eueowl1", which were very similar in their feature properties, thus it was hypothesized that it would affect the classification between these classes.



- This was also illustrated in the confusion matrix submitted previously which shows that different model classes struggled with distinguishing these particular classes from each other.



1. Introduction

General Procedure:

- **Individual Models:**

We tried to increase the performance of each trained model individually by:

  - Enhancing the feature selection to mitigate the observed problems during the experiments.
  - Trying different hyperparameters, e.g. kernel sizes, padding, schedulers. and optimizing algorithms, to further decrease the time of the training or boost the models' capacity.
  - Different Sampling techniques to alter the distribution of the classes as a possible solution to the skewness of the dataset towards the "other" class in connection with the rest of the classes especially "eueowl1" that was heavily under represented.
- **Ensemble Methods:**
  - We experimented with different ensemble methods.
    - Majority voting.
    - Multi-stage predictions.
- **Smoothing:**
    - Bird call smoothing.
    - Heuristic guided smoothing.

1. Introduction

Evaluation Metrics:

- **Negative Log Likelihood:**
    - In most of the experiments we modified the negative log likelihood to accommodate the skewness of the dataset, by elementwise multiplication of the loss function with predefined weights, which is based mainly on the distribution of the classes represented in the following $W_c = \frac{1}{\sum_0^n Y_i}$ such that Wc:= the weight for each class, n := number of elements in the class. Yi := Normalized annotation frequency.
        - This method produced small weights for classes with higher distribution in the dataset and higher weights for classes with smaller distribution, which helped the loss function to penalize the "other" class less than penalizing the rest of the classes, particularly the ones with the lowest distribution.
        - we only weighted the log likelihood of the training data and not for the validation or the test datasets, to only influence the training process
    - Furthermore we experimented with multiplying the negative log likelihood with the given gain matrix, yet these experiments failed because some of the losses calculated had negative values, which caused numerical instability during the calculation of the direction in the descent algorithms.
- **Average Recall / Confusion matrix:**
    - In most of the experiments we relied on the Average recall combined with insights obtained from the confusion matrix, because of the different weights of the negative log likelihood of the training and validation, we also observed the area under the recall-precision curve to get a clearer overview of the performance of each model.
- **Total Savings:**
    - We calculated the total savings by utilizing the gain matrix as mentioned in the task description to gain some information about the performance on the challenge server.

2. Models & Experiments

General Information:

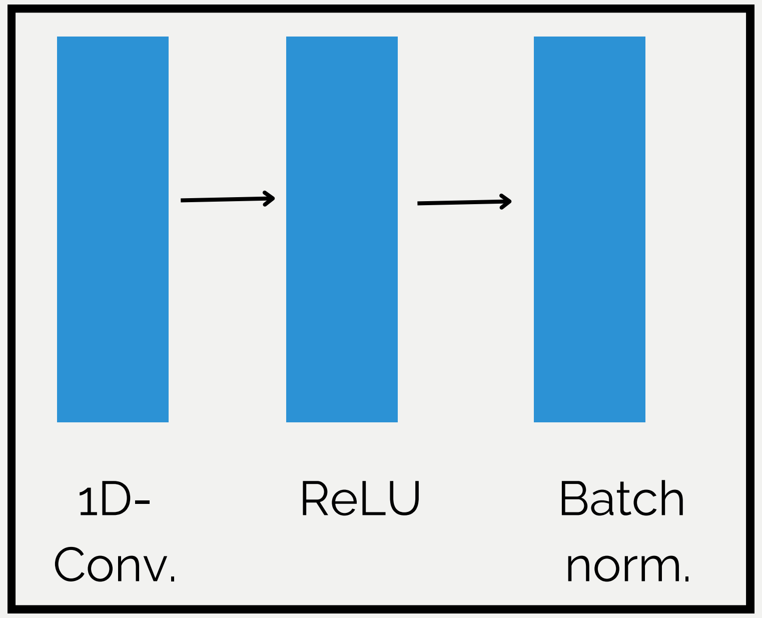
- **Convolutional Neural Networks:** Guided by the previous assumptions mentioned in the introduction about the temporal / spatial inter-correlation of the feature sets, e.g. "60 mel spectrogram values / sample represent temporal information", and our previous experiments, we decided to further study of the convolutional neural networks for this task.
- **Feature Selection:** In order to alleviate the issues observed in the confusion matrix, of misclassifying some of the classes, we expanded the feature sets selection used in our earlier implementations to include features that have high variance for the misclassified classes and lower variance in the rest, the feature sets selected were:
  - ["raw_melspect_mean", "cln_melspect_mean", "raw_mfcc_mean", "raw_mfcc_std", "centroid_mean", "yin", "cln_melspect_std",]
- **Optimizer:** We tried utilizing different optimizers with different hyper parameters and found that, in general, the Stochastic Gradient Descent with a learning rate of 0.01 outperformed all the others including different variants of Adam and RMS.
- **Cross Validation:** Due to the limited time and computational power, we limited the number of folds to 5 folds, using KFold cross validation, in contrast to the previously tested stratified Kfolds, as we wanted to experiment further on the effect of different sampling techniques.
- **Sampling:** Due to the imbalance of the dataset we experimented with different sampling techniques, performed on the training set only and kept the validation and test sets with the original distribution.
  - Over Sampling: For the under represented classes, we randomly duplicated samples with a confidence rate of the annotators >= 0.5, to end up with an average of 30000 samples per class.
  - Under Sampling: For the "other" class, the most over represented one, we randomly selected different samples using the same technique to end up with the same number of samples an average of 30000 samples matching all other samples.
- **Augmentation:** Since our data represents audio files, which is already preprocessed, we could not perform any augmentations except for adding gaussian noise to all samples with a maximum change of the normalized values of 1e-5 to preserve the original dataset. yet this method did not yield any improvement to the validation or the test sets, in fact results were generally worse. so we refrained from using this method.

2. Models & Experiments

Main Elements:

Convolutional Block:

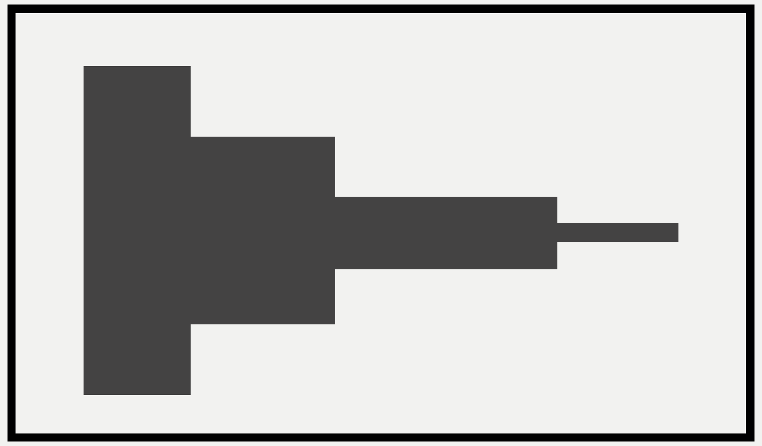
- In all of our experiments we used a Convolutional Block shown in the figure as the Basic unit of our networks, The block consisted of:
  - 1-Dimensional Convolutional Layer
  - Rectified-Linear Unit
  - 1-Dimensional Batch Normalization Layer as a regularization technique to help in overcoming the overfitting problem.



Convolutional Block

Convolutional Encoder:

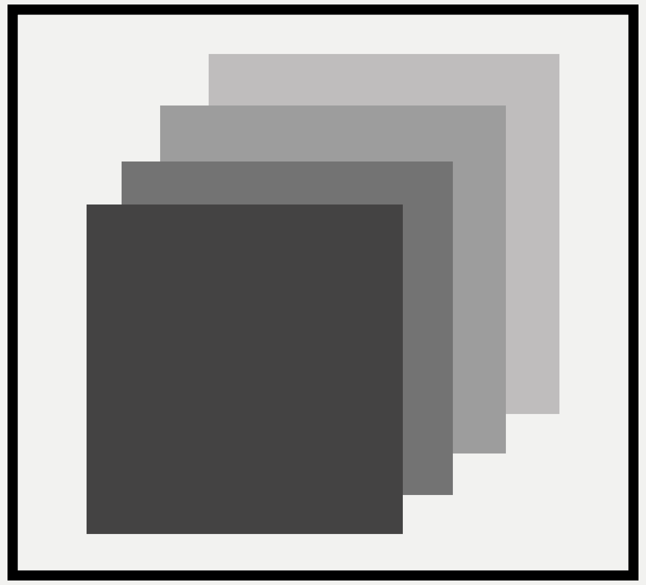
- Using the Convolutional Blocks, we constructed an encoder block to serve as a feature extractor, that consisted of:
  - 7 Convolutional Blocks with an increasing number of filters as follows: 1, 8, 16, 32, 64, 128 , 256, 512
  - The kernel sizes used varied between 7, 5, 3 with paddings of 3, 2, 1 respectively.



Convolutional Encoder

Convolutional Bottleneck:

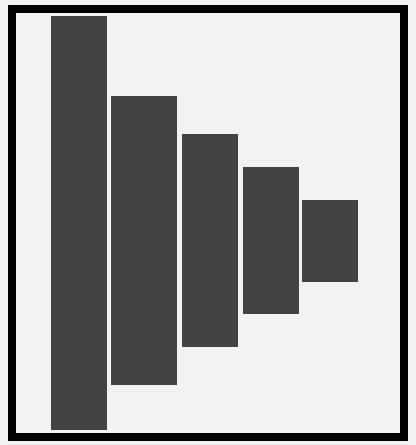
- Utilizing the convolution blocks constructed earlier, we built a latent space with dimension usually smaller than the input, to force the model to learn a more compressed representation.
- These latent spaces were usually a consecutive convolutional blocks with the same number of filters and kernel sizes as each other.



Convolutional Bottleneck

Linear Decoder:

- A Feature decoder element, was also used in different experiments were we used other 7 Linear Layers each followed by a ReLU activation with a decreasing number of features as follows: 512, 256, 128, 64, 32, 16, 8, 7



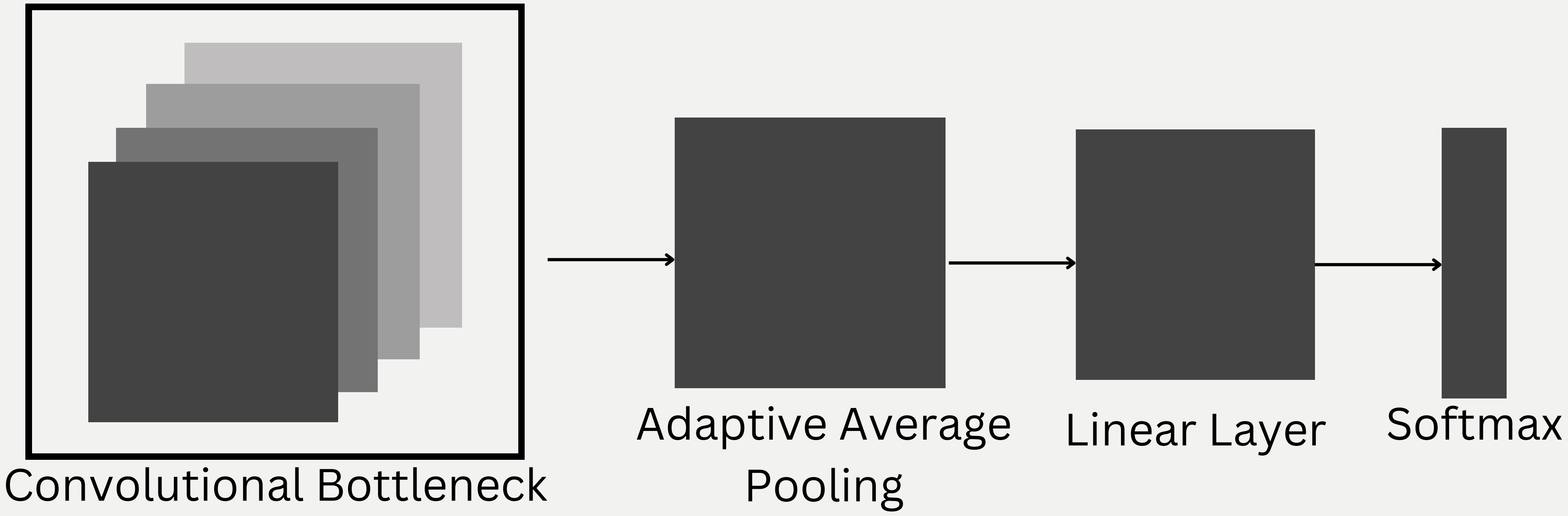
Linear Decoder

2. Models & Experiments

Model (A)

Convolutional Bottleneck:

- Initially we tested the performance of the convolutional bottleneck without adding any extra components except for Adaptive Average pooling, a linear layer followed by a softmax function, as shown in the figure.
- **Parameters:** The best model parameters for this architecture on the selected feature set was found to be 8 convolutional blocks with 128 filters. for each block.

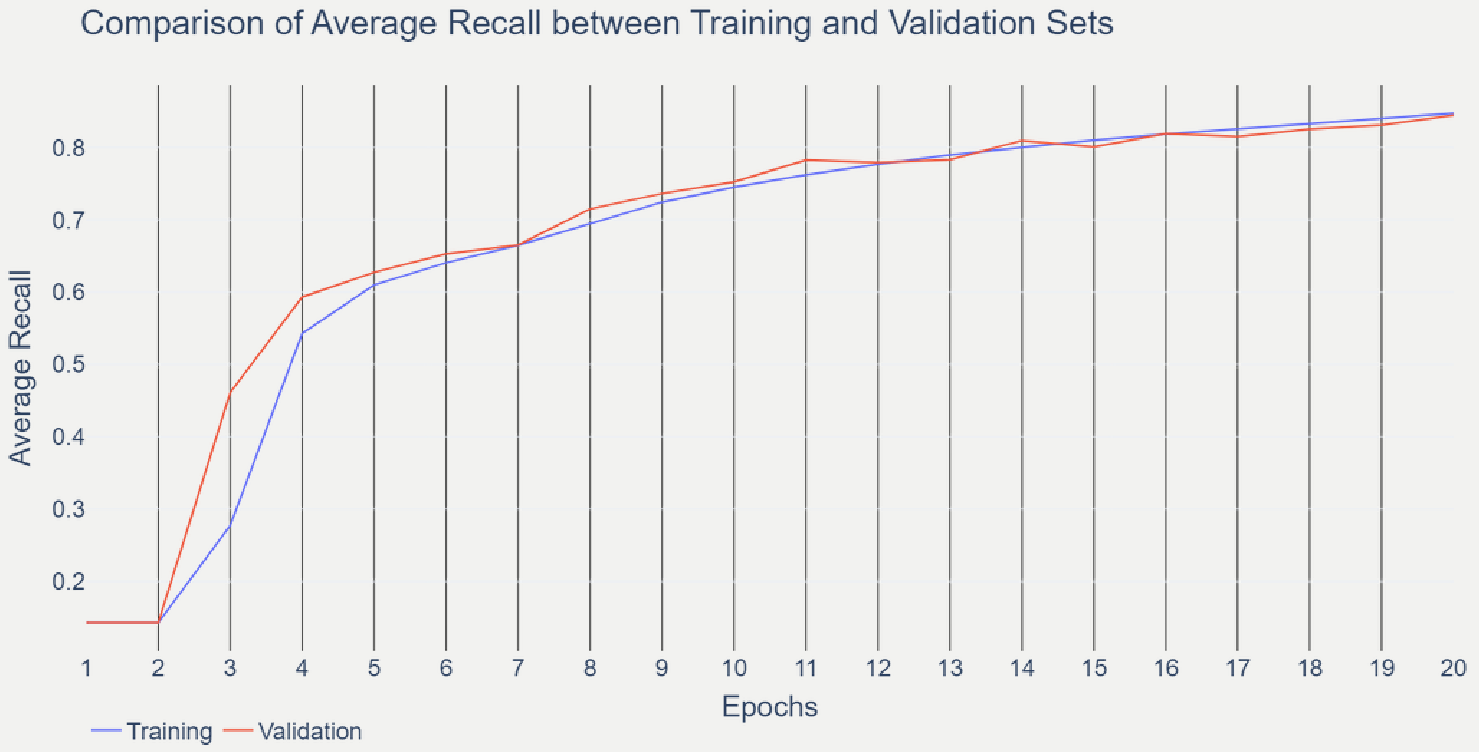


Convolutional Bottleneck

Adaptive Average Pooling

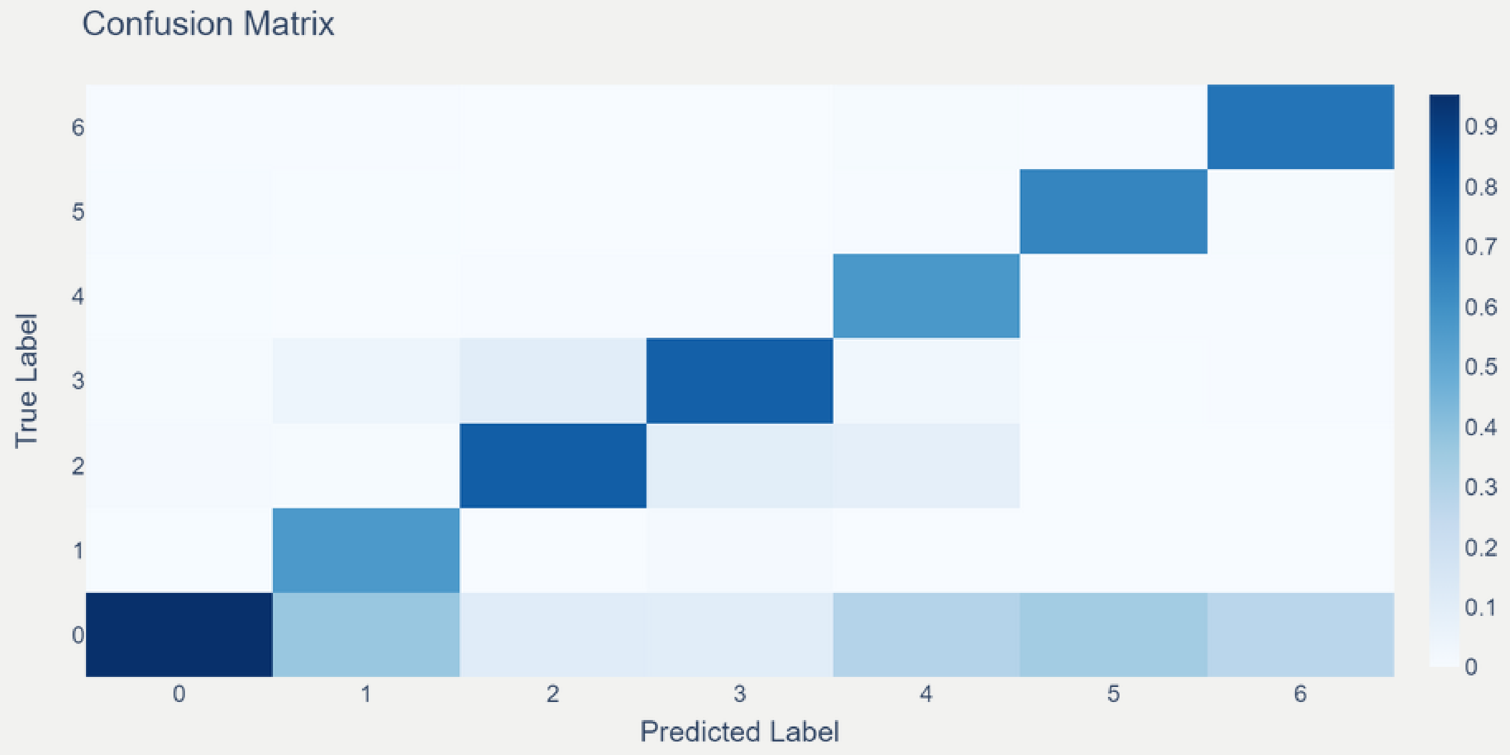
Linear Layer

Softmax



Comparison of Average Recall between Training and Validation Sets

- As Shown in the figure, the Model Performance showed signs of slight underfitting converging at around 85% Average recall.



Confusion Matrix

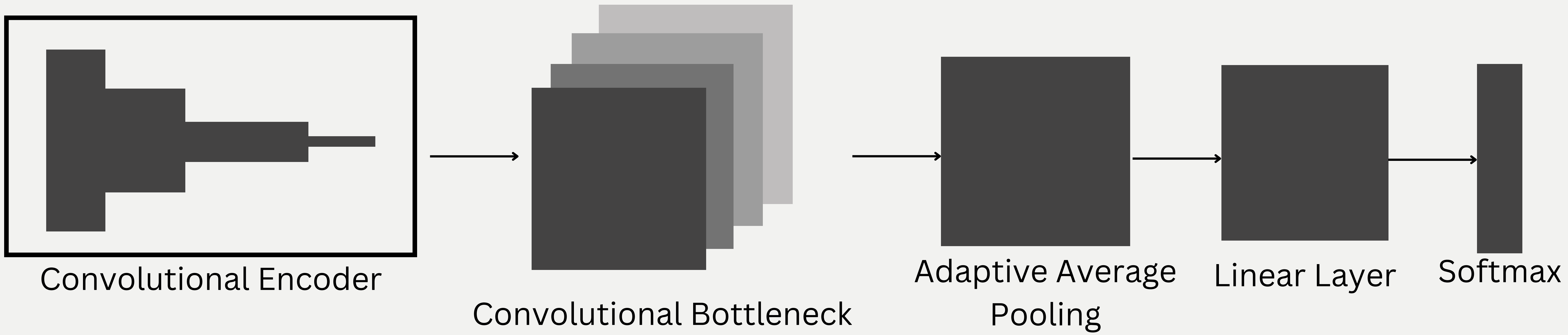
- The Model also suffered from misclassification of the other class as different classes, as shown in the confusion matrix.

2. Models & Experiments

Model (B)

Convolutional Encoder + Bottleneck:

- In the Second model we added Convolutional Encoder to the previous model in order to enhance its capabilities and possibly increase its performance overcoming the underfitting issue observed earlier.
- **Parameters:** The best model parameters for this architecture on the selected feature set was found to be:
  - Encoder: 7 Blocks with 1, 8, 16, 32, 64, 128, 256, 512 filters with kernel sizes of 5, 3, 3, ... 3 Filters respectively.
  - Bottleneck: 4 Blocks with 512 filters and kernel size of 3.

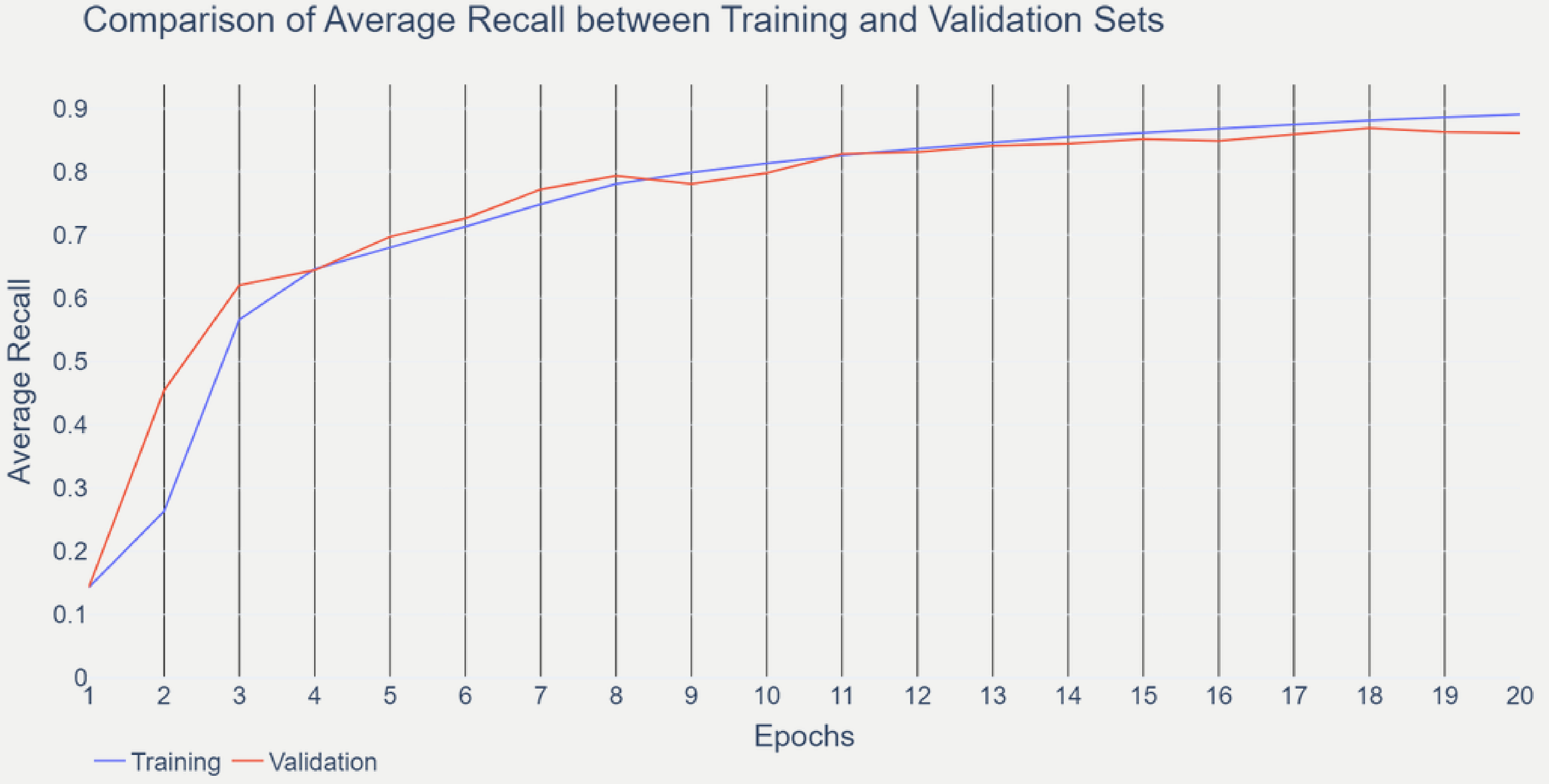


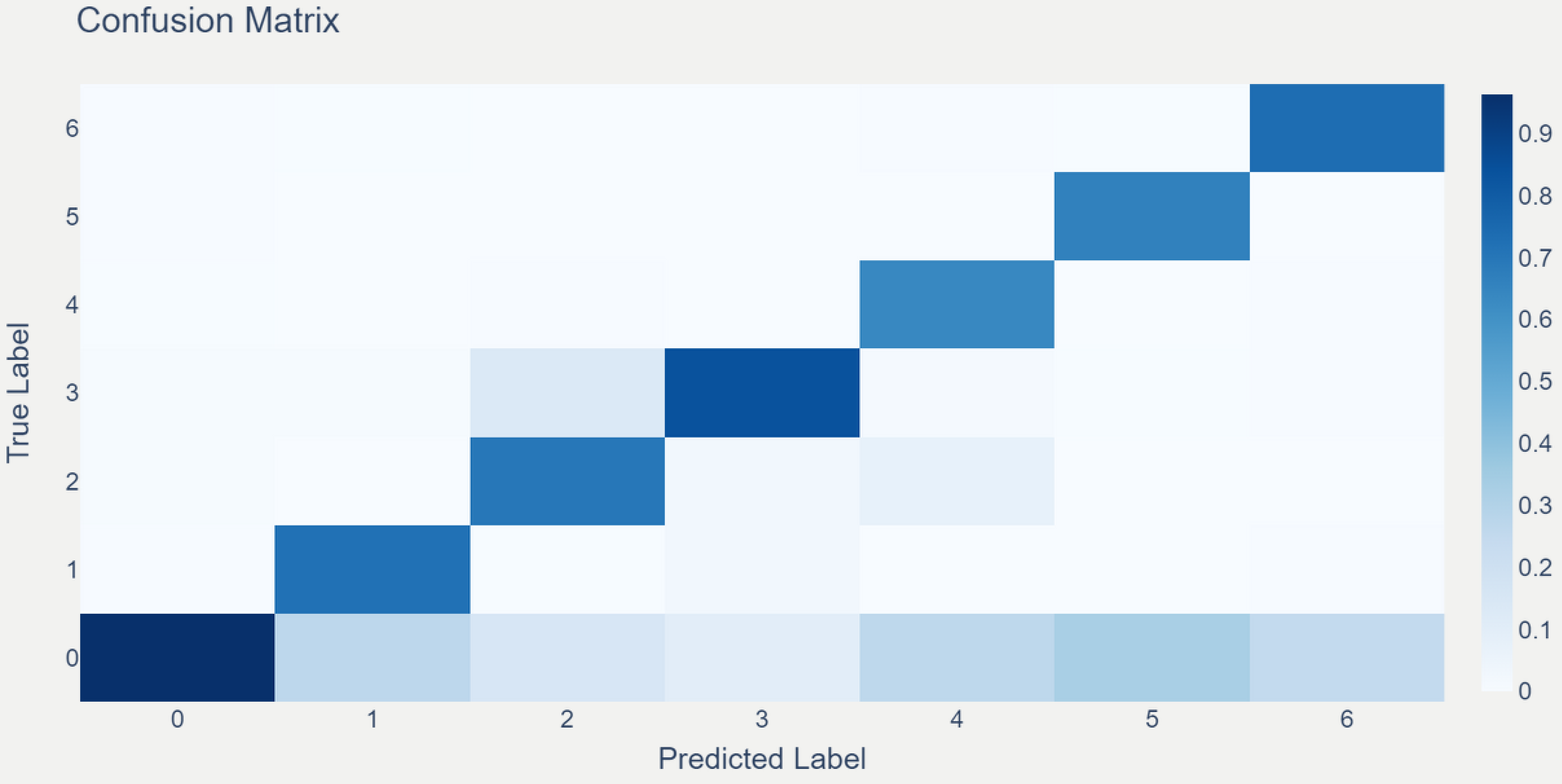
2. Models & Experiments

Model (B)

Convolutional Encoder + Bottleneck:

- The model started to show signs of overfitting, at an average recall of 87%, yet the models' capabilities was slightly enhanced in separating the birds themselves, yet it in the same time it showed underfitting properties converging at around 89% for the training set.
- It's also noticeable that the model still suffered from misclassifying the "other" class, instead it assigned values to that class. thus reducing the precision of the model as shown in the confusion matrix of the validation set.


Comparison of Average Recall between Training and Validation Sets

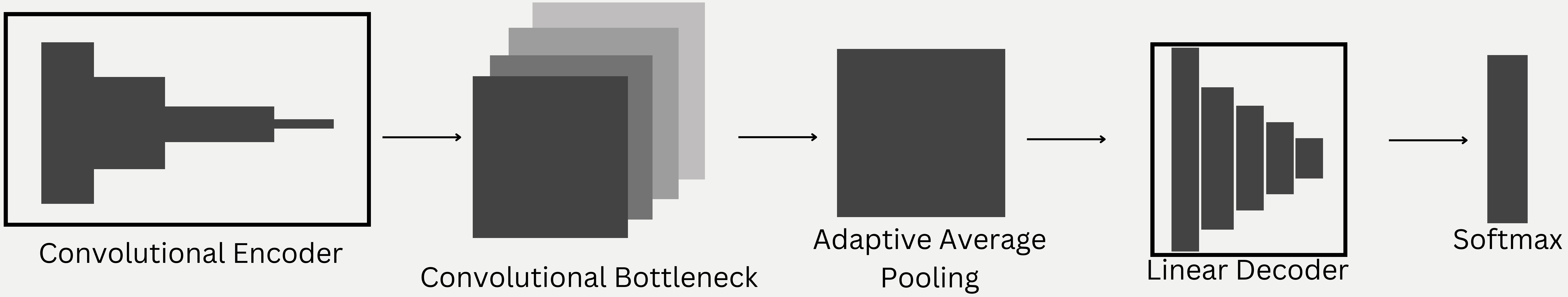

Confusion Matrix

1. Models & Experiments

Model (C)

Convolutional AutoEncoder:

- In the Third model, we added Linear Decoder to the previous model in order to enhance its capabilities and possibly increase its performance overcoming the underfitting issue observed earlier.
- **Parameters:** The best model parameters for this architecture on the selected feature set was found to be:
  - Encoder: 7 Blocks with 1, 8, 16, 32, 64, 128, 256, 512 filters with kernel sizes of 5, 3, 3, ... 3 Filters respectively.
  - Bottleneck: 4 Blocks with 512 filters and kernel size of 3.
  - Decoder: 7 Linear Layers with decreasing number of outputs as follows 512, 256, 128, 64, 32, 16, 8, 7 accompanied by a ReLU activation for each layer.



Convolutional Encoder

Convolutional Bottleneck

Adaptive Average Pooling

Linear Decoder

Softmax

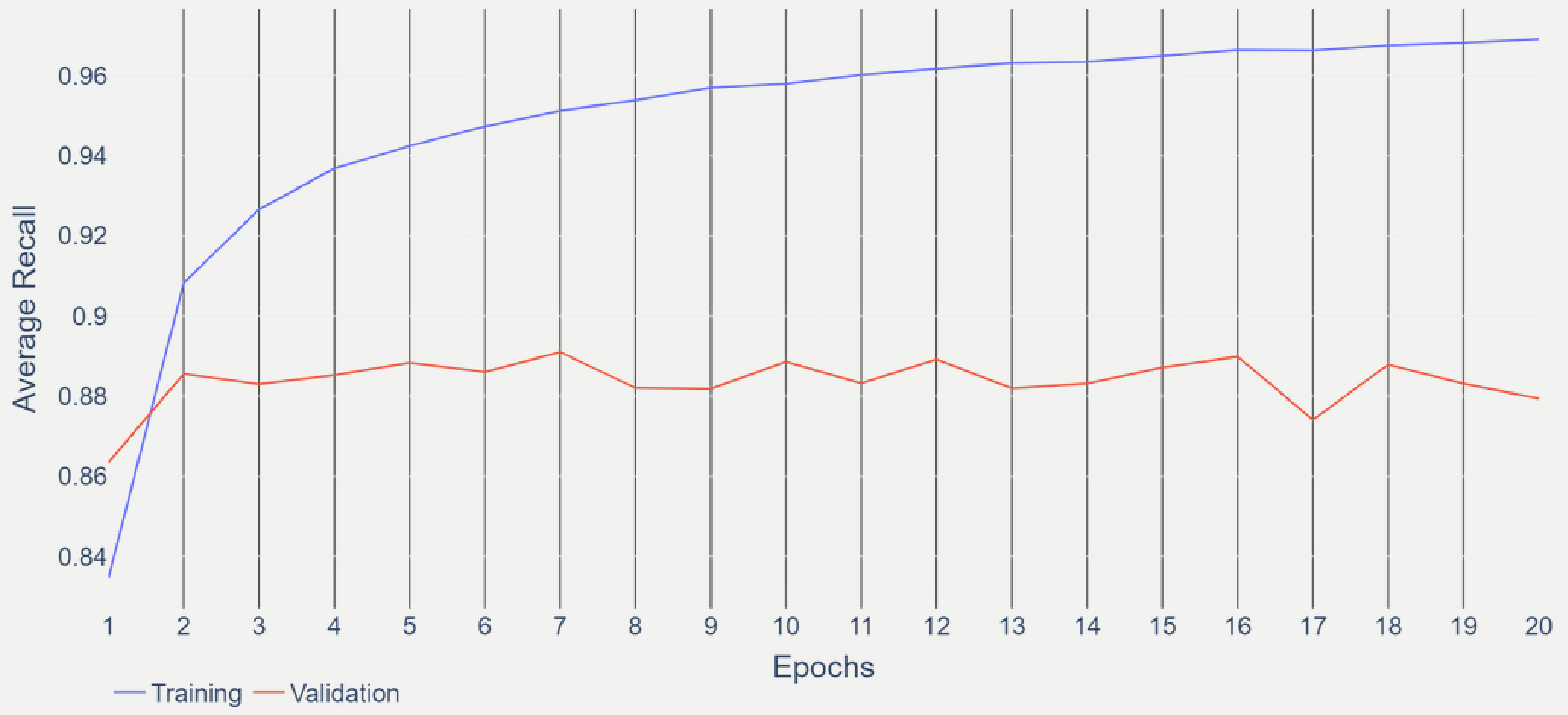
2. Models & Experiments

Model (C)

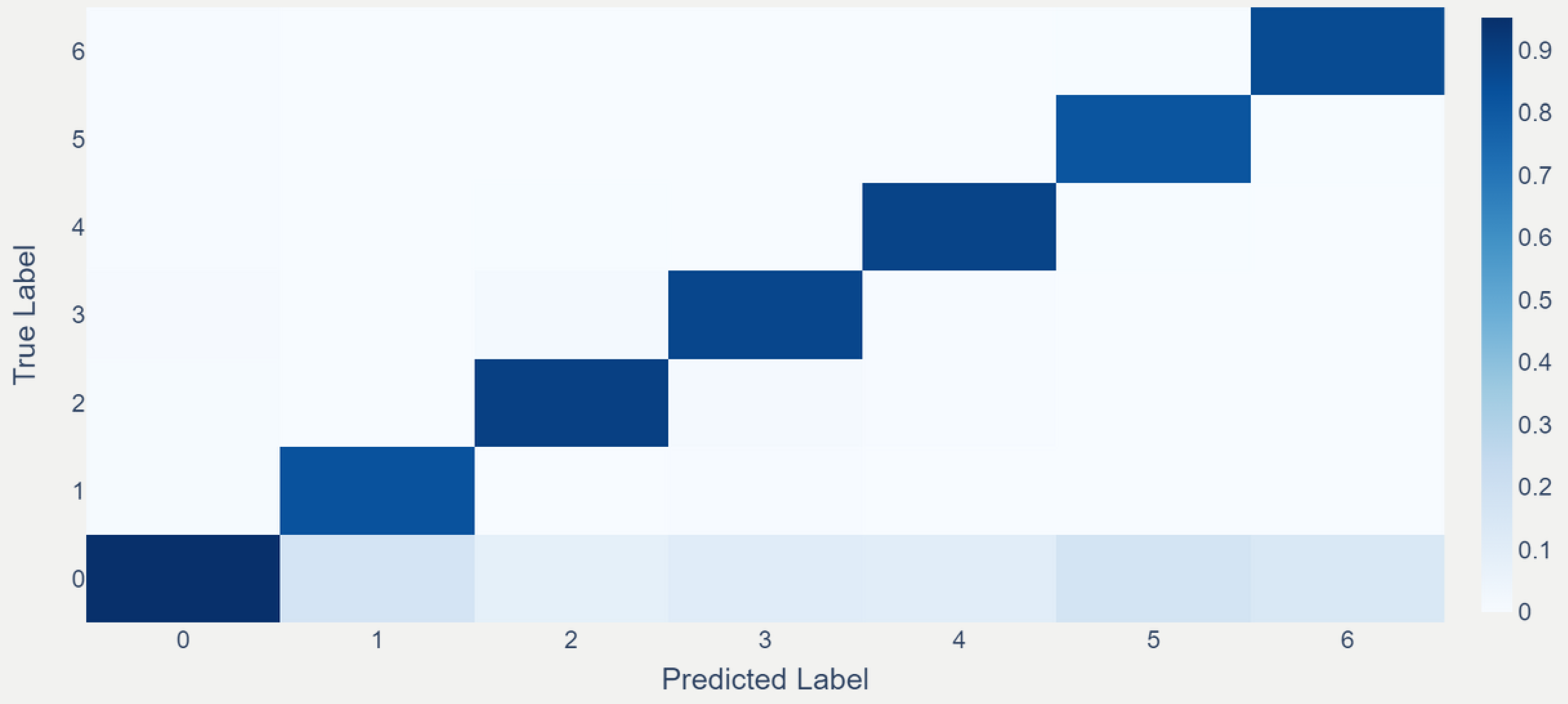
Convolutional AutoEncoder:

- The goal of this experiment was met as shown in the figure as it has inceased the learning capabilities of the model to converge at 96% on the training set.
- It's also very noticeable that the model started to show aggressive signs of overfitting, as in the validation set the average recall was roughly unchanged around 88%.
- In the confusion matrix of the validation set, it is clear that the model was almost perfectly distinguishing the birds from each other yet it was still incapable of distinguishing the "other" class

Comparison of Average Recall between Training and Validation Sets



Confusion Matrix



2. Models & Experiments

Model (C)

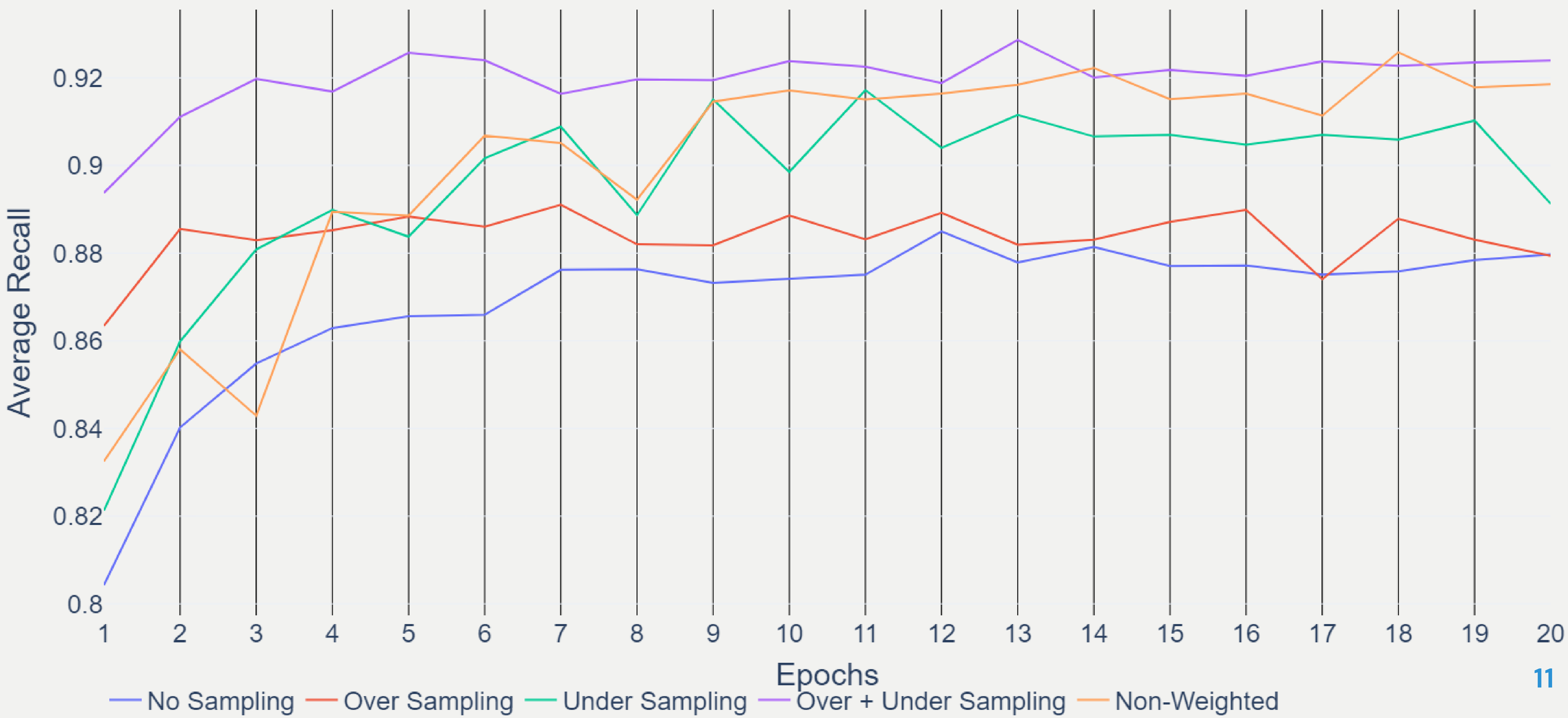
Convolutional AutoEncoder:

- By Applying different sampling techniques to the dataset:
  - Over Sampling: by randomly duplicating the samples of each class, the average number of samples reached around 30000 samples
  - Under Sampling; by randomly selecting the samples of the "other class" to reach around 30000 samples.

- Overall, the Combination between the Under and Over Sampling techniques showed the highest result in terms of average recall, as it reduced the effect of over fitting to some extent to converge on the validation set at around 92%.

- Also it is worth noting that weighting the log likelihood with the same weights as earlier was still viable on this dataset as it outperformed the non-weighted one.

Comparison of Average Recall between Different Sampling Techniques

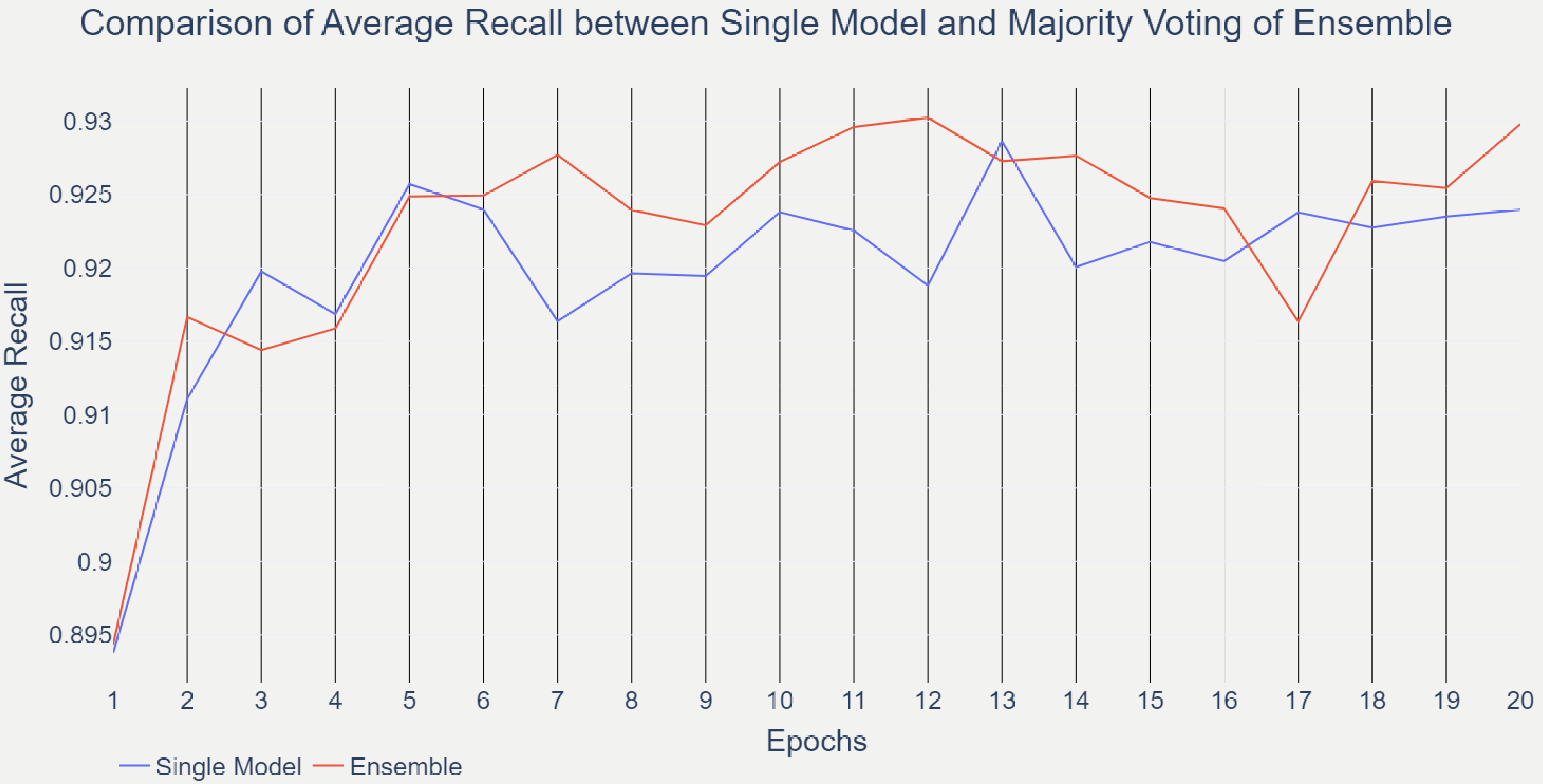


3. Ensemble

Model (C) - Variations

Majority Voting:

- Using the same earlier sampling and weighting techniques, and for the same model class, we trained 3 different models with different parameters varying in maximum number of filters and kernel sizes, and trained on different feature sets to be used in an ensemble of models.
- The results were used in a majority voting algorithm to choose the most common class from the outputs of the models, acting as a mixture of experts, in order to boost the results of the model. Yet the difference on the validation set was not that noticeable, as it only increased from 92% to 93% average recall.

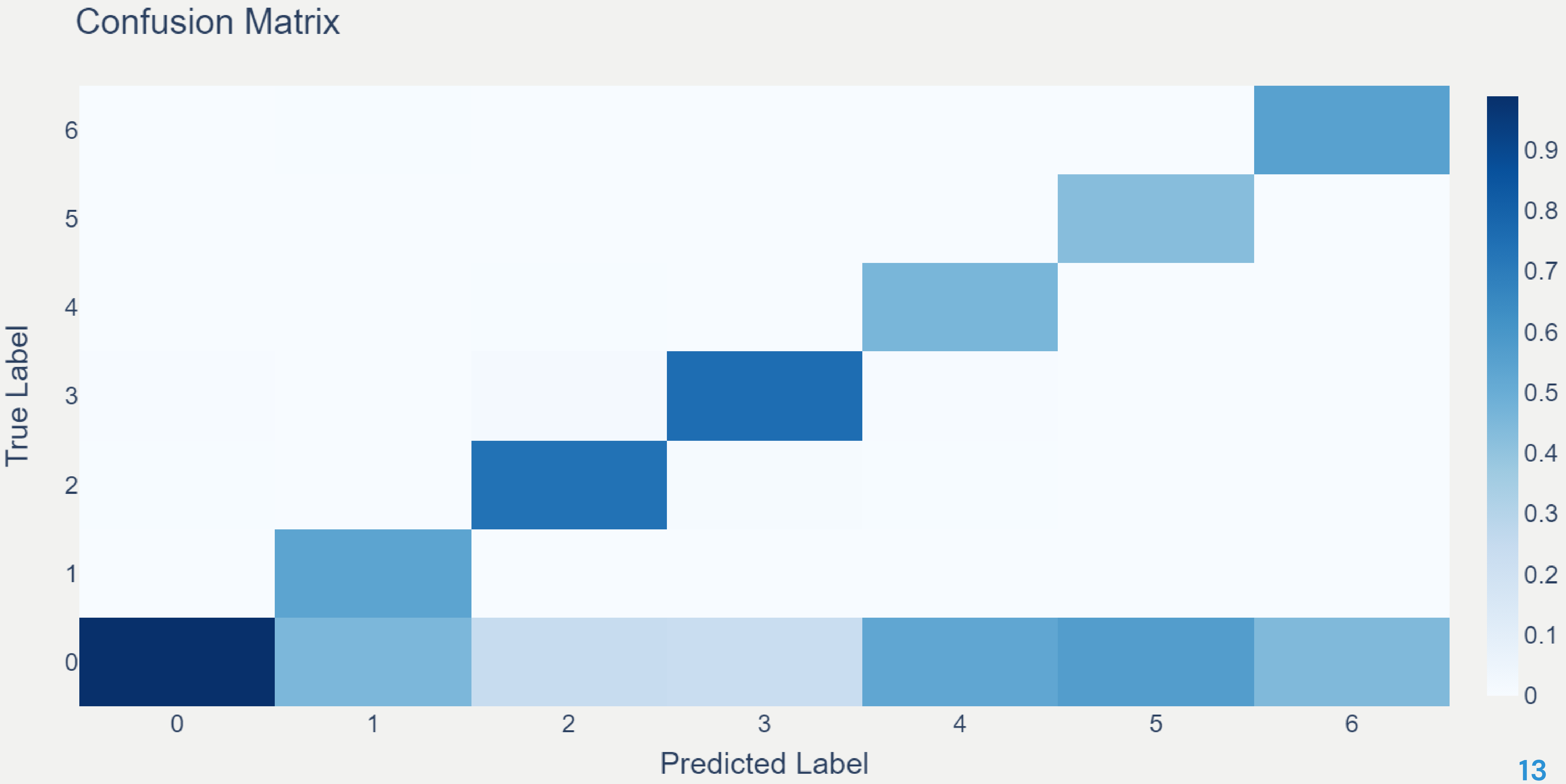


3. Ensemble

Model (C) - 1D + Binary

Multi-Stage:

- Since our attempts so far had a common problem, which is identifying the "other" class from the rest, we tended to create a separate binary model, as the next stage to enhance our predictions.
- The same model architecture was used but instead of 7 classes as outputs we only used 2 classes "bird" or "no bird"
- Similarly we adopted the oversampling technique, to mitigate the imbalance of the classes.
- Lastly we only used this model to predict the samples which was previously predicted as birds, since we had a problem in identifying the no birds.
- Yet, Not only that the problem still persisted, but it reduced the overall performance as it has increased the number of birds mistakenly predicted as "other", thus reducing the precision of the model, as shown in the confusion matrix.



3. Ensemble

Model (C) - 2D + Binary

Multi-Stage:

- In another attempt, we tried to exploit the temporal relationship between the samples, [multiple samples form a bird call] with the followings approach:
  - We grouped consecutive samples belonging to the same class with the same order as they appeared in the dataset, to form a 2-dimensional representation of bird calls, each was regarded as a sample.
  - We then chose a specific window length [Number of rows per bird call] for each experiment we performed.
  - For The label of the 2D samples, we created a one-hot-encoding of the bird class present in these samples.
  - By using a 2-Dimensional Convolutional Neural Network with the same architecture and features as before, we trained different models and evaluated them independently, yet the only difference between the previous experiments and this one was replacing the 1-D Layers with 2-D Layers, to accommodate for the new input shapes.
- Since our task is to predict single samples, we used the binary models trained earlier to predict the location of the bird calls, then predicted the bird calls themselves.
- Although this method was not successful for small window lengths, "produced total savings with negative values and very low average recall", it is worth mentioning that the models trained with large window lengths, window length = 100 to predict a single label for each file, performed reasonably well, reaching an average recall of 97% on the validation set, yet that was not our main objective of the task.

4. Smoothing

- **Bird Call Majority Voting:** As per the task description, each call consisted from several consecutive samples, with the same label, so
  - we selected each consecutive predictions, and regarded them as a single call.
  - Then we applied majority voting for these classes. in order to smooth the output.
  - e.g. [000133332200] -> [000333333300]

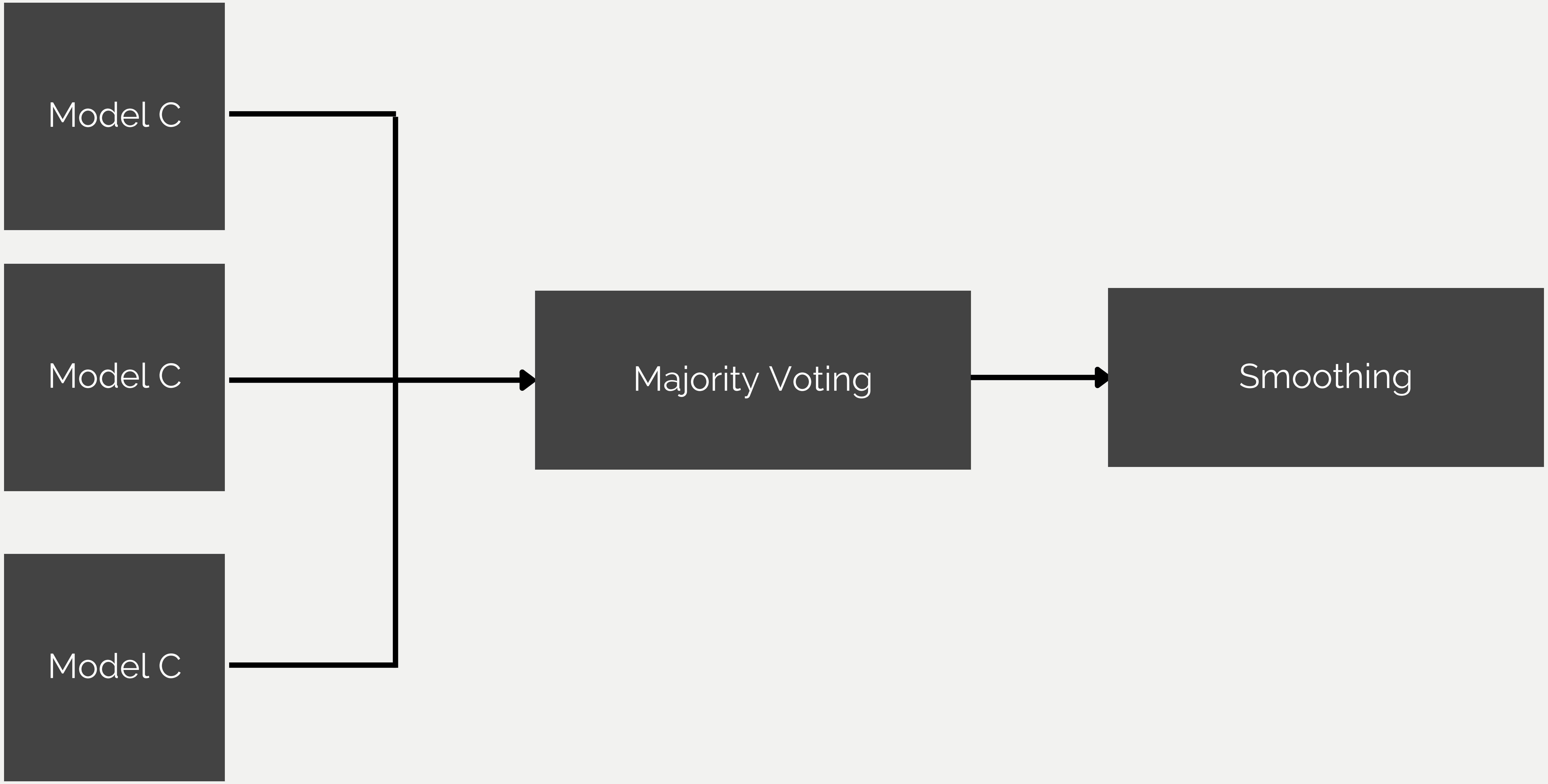
This method resulted in slightly increasing the result on the test server.

- **Guided Heuristics Smoothing:** From the task description we observed that the samples before and after each call were disregarded during the calculation of the total savings, so we exploited this fact by:
  - Selecting the calls with length = 1 preceded or followed by a call with length > 1 and created pairs of these calls e.g. **[001011]**
  - From those pairs we disregarded the pairs which had different predicted classes **[002011] x**
  - Further more we disregarded any calls where the space between them was more than 2. **[00100011] x**
  - we then flipped the space between them to the same prediction class. **[0010110] -> [0011110]**

This method slightly decreased the result on the test server.

5. Conclussion

Proposed Classifier:

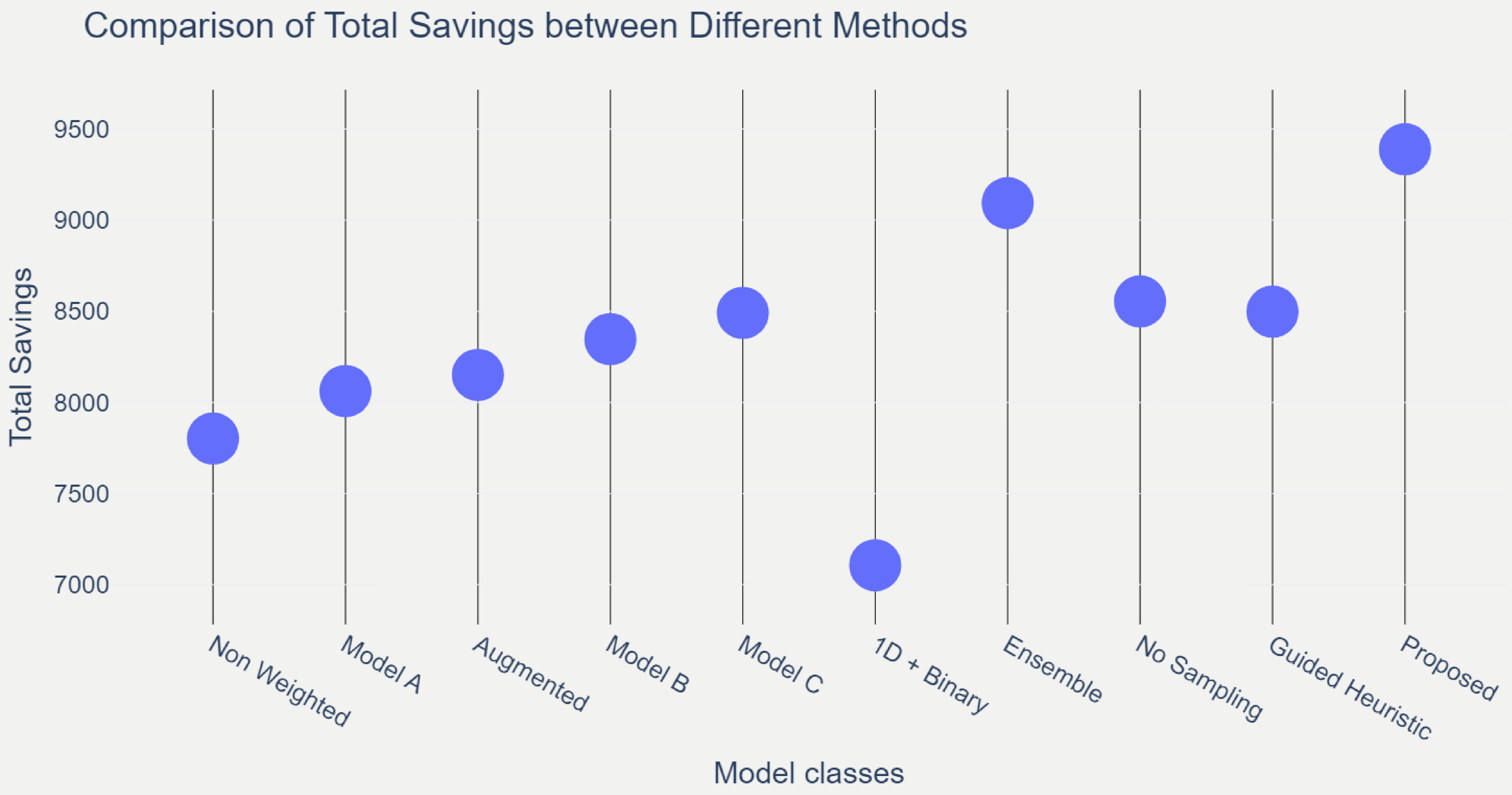


5. Conclussion

Key Findings:

- **Sampling:** Over and Under sampling techniques, boosted the model performance further.
- **Weighted Loss:** Weighted negative log likelihood outperformed the original log likelihood.
- **Feature Modification:** Modifying the feature sets in accordance to the behavior of the models, helped in increasing the average recall.
- **Majority Voting Ensemble:** 3 Different Models using the same architecture as Model C, trained on different feature sets, with different parameters, e.g. Kernel sizes, padding, .. etc.., slightly yielded better results than utilizing a single model.
- **Bird Call Smoothing:** Reduced the errors within the bird calls, and in general achieved higher score in the challenge server.
- **Data Augmentation:** Introducing gaussian noise to the input data worsened the results.
- **Multi-Stage [1D]:** Experiments using the binary classification on 1D inputs failed to enhance the results, thus neglected in the final classifier architecture.
- **Multi-Stage [2D]:**
    - Small window length produced very poor results, yet when predicting a single label for samples with window length, more than 70, the problem of overfitting vanished, but this was not the task.
    - Yet when combining it with the 1D Binary Classifier it performed very poorly, thus refrained from using it.

5. Conclussion


Comparison of Total Savings between Different Methods
Total Savings
Non Weighted
Model A
Augmented
Model B
Model C
1D + Binary
Ensemble
No Sampling
Guided Heuristic
Proposed
Model classes