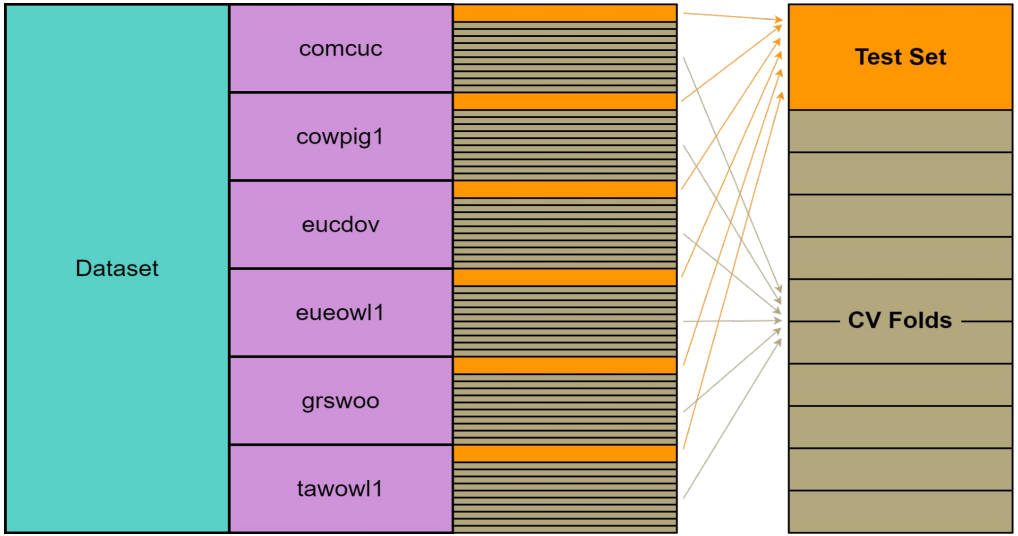
Dora the Model Explorer

Team Ugly

Leon Alexander Akkad, Dominik Baron, Matthias Guzmits, Aditya Kumar Tomar

Data split

- In the very beginning, we randomly split the data (shape = 120, 100, 548) on the first dimension into 80% training and 20% test set. The split was done on the first dimension to ensure that similar samples from the second dimension (same recording) do not get split to avoid contamination of the test set. We selected the labels based on the majority vote, since the annotator agreement was generally quite high.

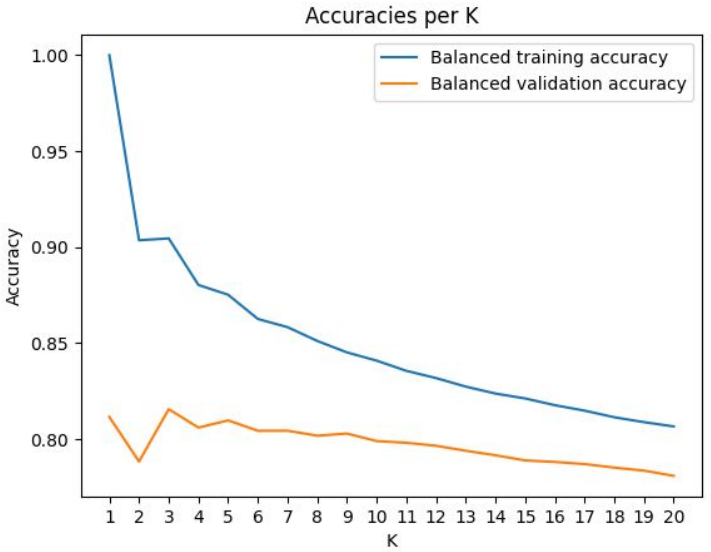


Preprocessing & Training Pipeline

1. Highly correlated features were removed (threshold 0.9 based on Bravais Pearson Correlation Coefficient) from the training and then removed those same features also from the test set.
2. In order to balance the data, we removed samples from the training set, thus balancing the classes. Basically, we ended up having roughly 7 times the number of samples of the class with least occurrence in the training set. The idea was to provide a balanced dataset to optimize for the balanced accuracy instead of the normal accuracy. Furthermore, this led to a much faster training time in both approaches. All classifiers, except the FNN and Attention classifier used this approach. Nonetheless, these two classifiers also employed some form of data balancing which is explained in the respective slides.
3. The two aforementioned classifiers utilized Cross Validation following the approach in a., while all others used approach b.
    a. In the first approach we split the samples of each bird into 10 folds, then combining the folds of each bird, such that each fold has the same amount of samples per bird.
    b. In the second approach CV was achieved by splitting the training set into 10 folds using StratifiedKfold from sklearn to ensure that the same balanced class distribution occurs within each fold. Keeping the class distribution yielded better results for balanced accuracy
4. For each iteration of the CV, the training folds (9/10) get normalized using StandardScaler from sklearn, and this normalization was applied on the validation fold (10th fold) as well.
5. For each model, we tried a varying number of hyperparameter settings using CV and then finally chose the one with the best balanced validation accuracy. We used the balanced validation accuracy as our primary evaluation metric as it captures the true generalization capability of the respective classifier due to the unbalanced nature of the dataset (class “other” dominates the other classes as seen in the data exploration task). The baseline accuracy of the dataset is around 71% (ZeroR) which is basically proportional to the ‘other’ class (no bird). All our classifiers were able to achieve better accuracy (balanced) than this. The baseline balanced accuracy should be approximately □ (≈ 0.143) and based on the performance of our classifiers the best accuracy that can be achieved on the test set could be roughly around 90% ∓ 2%.
6. We used the best hyperparameter setting (based on the mean validation accuracy over all CV iterations) to finally train our model on the whole training dataset (not just the 9/10 folds) and used the normalization used on the training dataset for normalizing the test set.
7. Finally, each classifier was evaluated on the test set and a confusion matrix (normalized over row) was plotted to see how well the model predicts each class. The diagonal entries will show the percentage of correct predictions for that class and off-diagonal entries show how much the true class of that row was confused with other classes in that row.

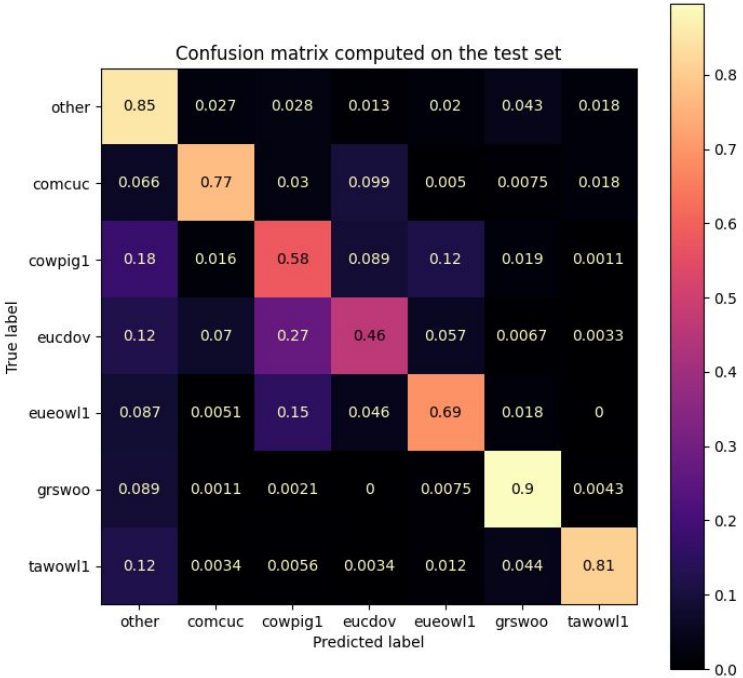
K-Nearest Neighbour (1)

- For finding the best parameters, different values for K, the leaf size and the power parameter P for the minkowski metric were tried.
- The model severely overfits, as expected, with K=1. With K getting larger, the model starts to underfit, indicating that larger values than 20 should not be considered.
- Changing the leaf size did not have much impact on the performance of the classifier. Setting P to 1 for the manhattan distance, on the other hand, resulted in worse performance compared to the default value P=2 across the board.



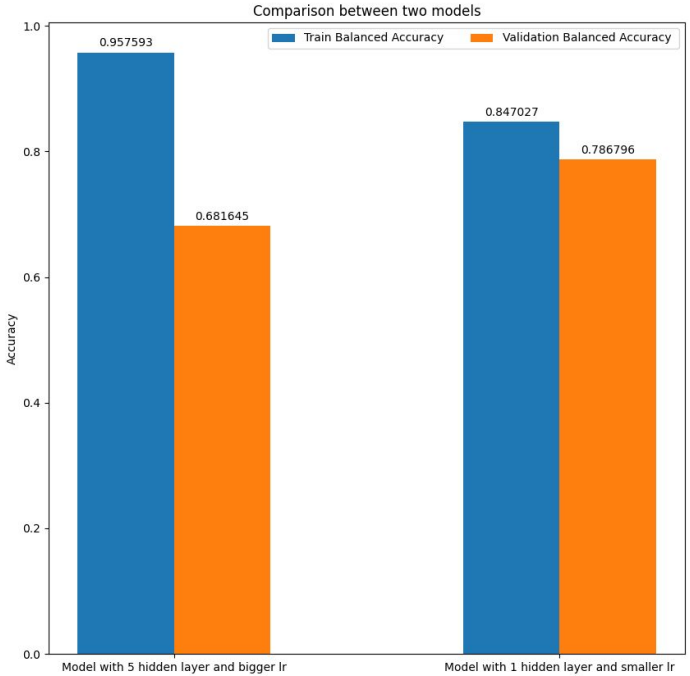
K-Nearest Neighbour (2)

- The best performing classifier utilized 3 nearest neighbours, hence K=3.
- A test balanced accuracy of **0.7241** was achieved.
- The confusion matrix indicates that the classes “cowpig1” and “eucdov” seem to be the hardest to distinguish.



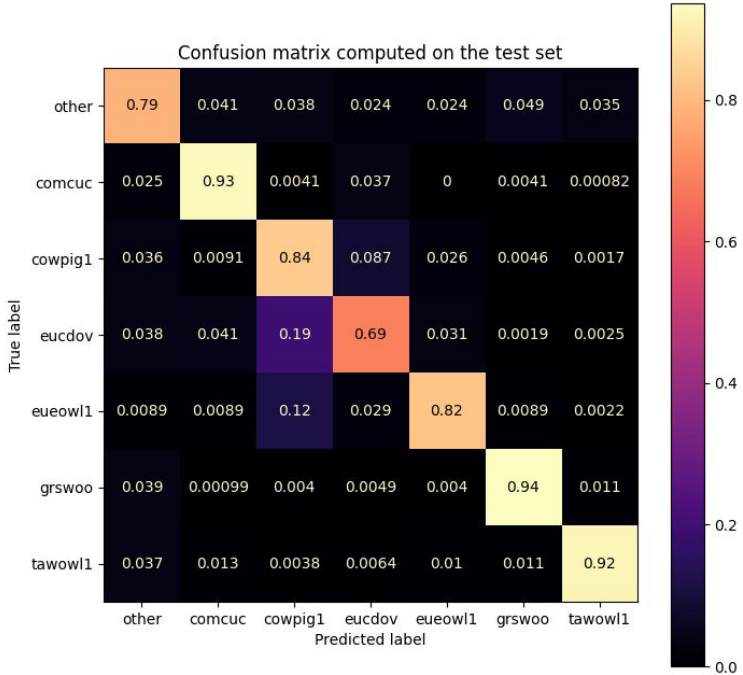
Feedforward Neural Network (1)

- In order to balance the data, a torch dataset with fixed epoch size was created, which provides a balanced number samples for each class per epoch, going through the samples of each class round robin style.
- Overfitting does occur for networks with more than 1 hidden layer and without any regularization methods, when trained for many epochs. This impacts the performance on the validation set as it is around 25% lower than the one on the training set.
- Combining a high learning rate with many layers causes the network to overshoot and stop learning at all.
- We primarily used ReLU or Leaky ReLU to not run into vanishing gradients.
- Using ReLU as activation function poses an issue as it can set weights to 0 (creating a so called “Dead Neuron”) if the gradient gets too large. This problem is known as “Dying ReLU”, resulting in only predicting one of the classes.
- We utilized the Adamax optimizer with the CrossEntropyLoss with a learning rate (lr) between 1e-3 and 1e-4. Adam optimizer also lead to sufficient results, nevertheless, Adamax performed slightly better.



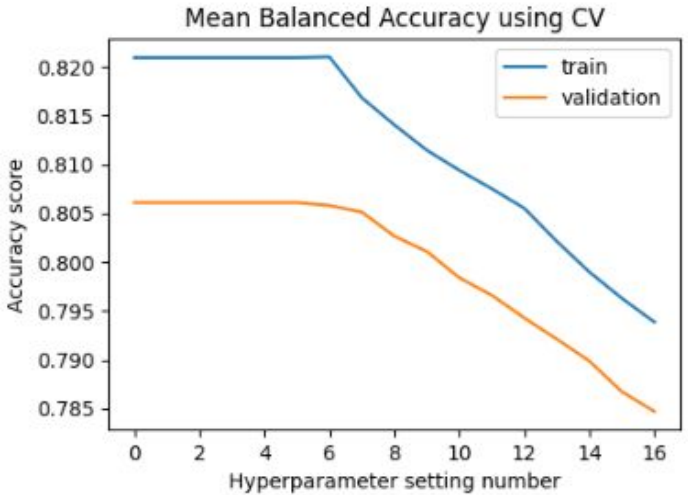
Feedforward Neural Network (2)

- The so far best model has only a single hidden layer and utilizes a dropout rate of 0.4 for regularization purposes. The used learning rate is 8e-4 and the model was trained for 400 epochs on the whole training data.
- A test balanced accuracy of **0.8464** was achieved.
- The confusion matrix indicates that the classes “cowpig1” and “eucdov” seem to be the hardest to distinguish.



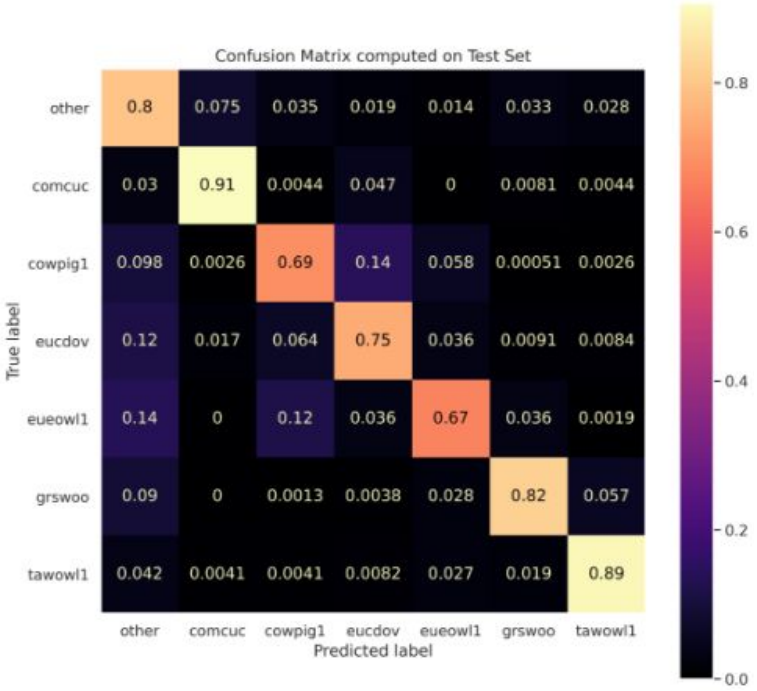
Linear Discriminant Analysis (1)

- 17 Hyperparameters were explored using ParameterGrid from sklearn on the following [{'solver': ['svd'], 'n_components': [1, 2, 3, 4, 5, 6]}, {'solver': ['lsqr'], 'shrinkage': list(np.linspace(0, 0.2, 11))}].
- The main idea was to see how the model behaves with varying shrinkage (for lsqr) and n_components (for svd).
- The plot indicates that the accuracy stays same (the constant part) when varying the components while it slowly goes down when the shrinkage is increased hinting that shrinkage can be thought of as a potential regularization parameter.
- To conclude, we see that the model doesn't really overfit but starts to underfit with increasing shrinkage.



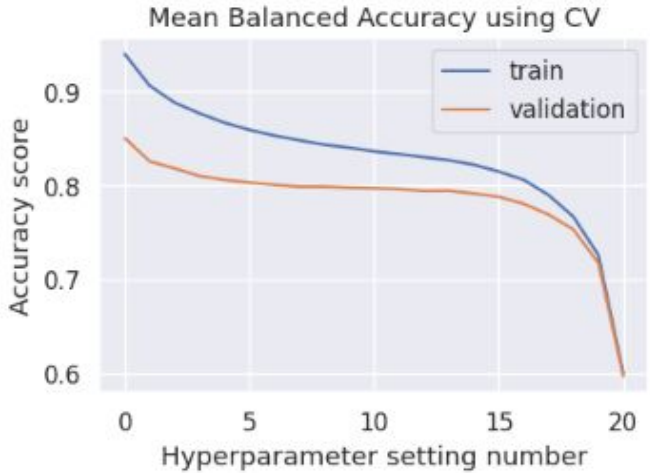
Linear Discriminant Analysis (2)

- Finally, the best hyperparameter ({'n_components': 1, 'solver': 'svd'}) was selected the way it was mentioned in the Training Pipeline slide.
- A balanced test set accuracy of **0.791** was achieved.
- The confusion matrix shows that the model mostly confused ‘cowpig1’ with ‘eucdov’ and ‘eueowl1’ with ‘cowpig1’.



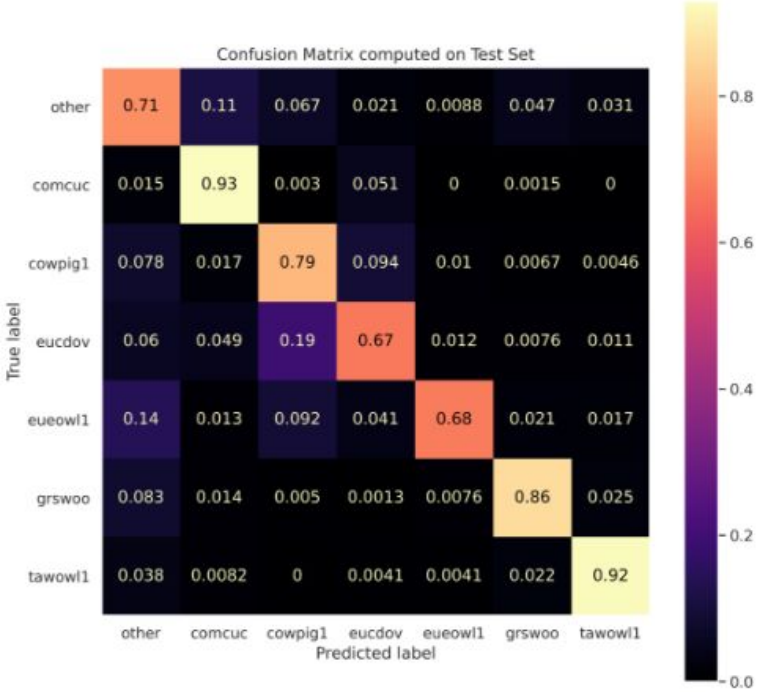
Quadratic Discriminant Analysis (1)

- 21 Hyperparameters were explored using ParameterGrid from sklearn on the following [{'reg_param' : list(np.linspace(0,1,21))}].
- The main idea was to see how the model behaves with varying reg_param which should hopefully control the overfitting/underfitting behaviour of the model.
- The plot indicates that the accuracy decreases with the increase of reg_param value which can be thought of as a potential regularization parameter.
- To conclude, we see that the model doesn't really overfit as the difference between train and validation accuracy is not much but starts to underfit with high reg_param values.



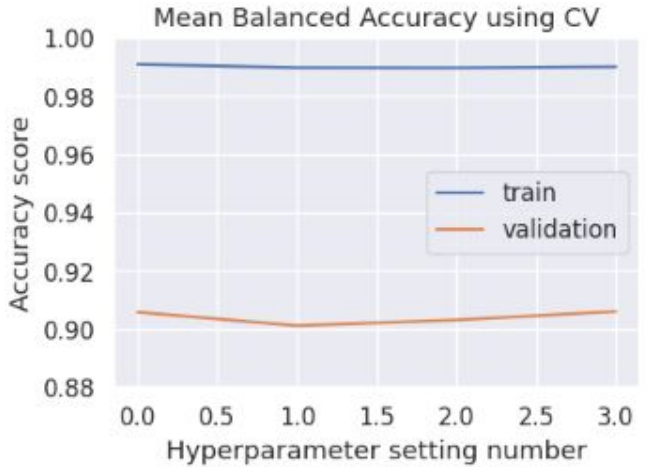
Quadratic Discriminant Analysis (2)

- Finally, the best hyperparameter ({'reg_param': 0.0}) was selected the way it was mentioned in the Training Pipeline slide.
- A balanced test set accuracy of **0.795** was achieved.
- The confusion matrix shows that the model mostly confused ‘cowpig1’ with ‘eucdov’ and vice-versa. Also it confused ‘eueowl1’ with ‘other’ (no bird).
- Overall, it has somewhat similar performance to linear discriminant analysis with some differences in prediction as shown in the confusion matrix.



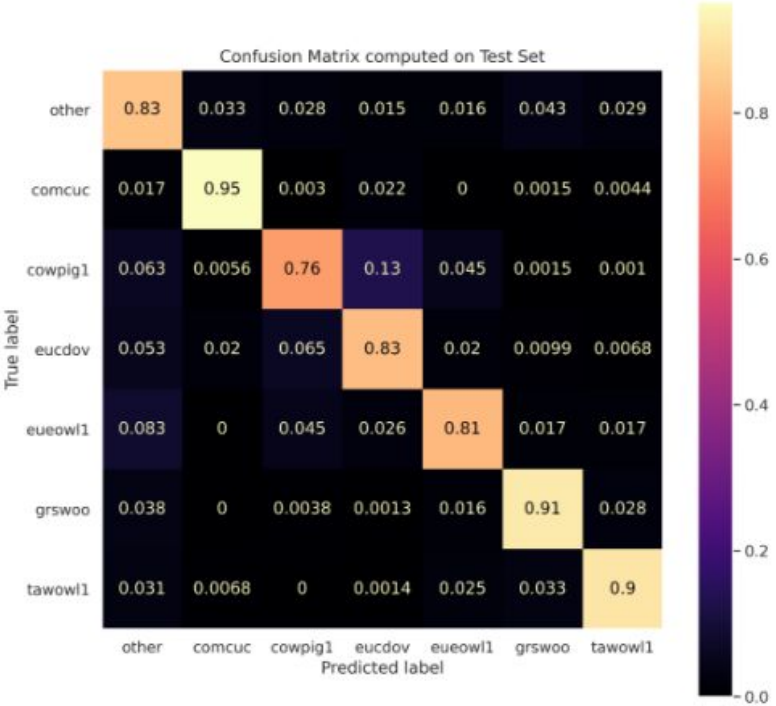
Histogram-based Gradient Boosting Classification Tree (1)

- We chose this sklearn model over normal Gradient Boosting Classifier since it was mentioned that this estimator is much faster (still relatively slow) for big datasets (n_samples >= 10 000) .
- Since this was quite slow to train on our machine, we only considered 4 hyperparameter settings. [{'loss': 'log_loss'},{'learning_rate': 0.2}, {'max_leaf_nodes': 20}, {'l2_regularization': 0.2}].
- The main idea behind these hyperparameter settings was to identify those which can probably lead to some regularization and/or strong overfitting.
- The plot indicates that the model was able to fit (potentially overfit) to the data very well (99% train accuracy) in all 4 settings. The validation accuracy was also above 90% which is quite good. None of the chosen settings seem to reduce (regularize) the training accuracy suggesting either the model is complex enough for the data already or more settings need to be explored.
- To conclude, this model had one of the best performance on the validation set and on the test set as we'll see in the next slide, making it a good candidate for the challenge as well.



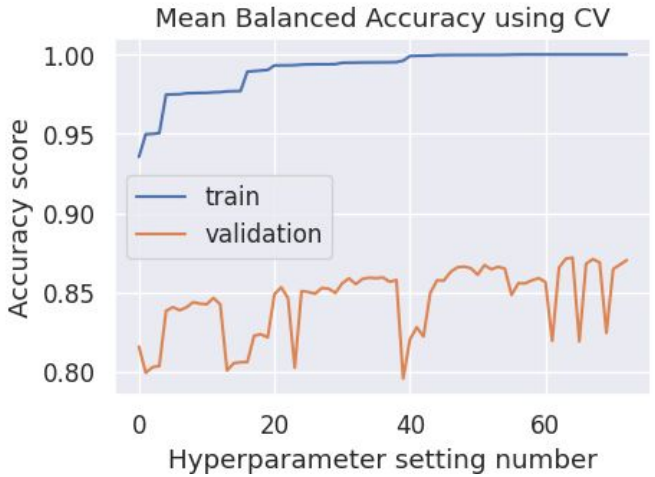
Histogram-based Gradient Boosting Classification Tree (2)

- Finally, the best hyperparameter ({'l2_regularization': 0.2}) was selected the way it was mentioned in the Training Pipeline slide.
- A balanced test set accuracy of **0.8566** was achieved.
- The confusion matrix shows that the model performs fairly well in classifying all the classes. The only weak point could be confusing ‘cowpig1’ with ‘eucdov’.
- Overall, it has one of the best performances among all the classifiers we used.



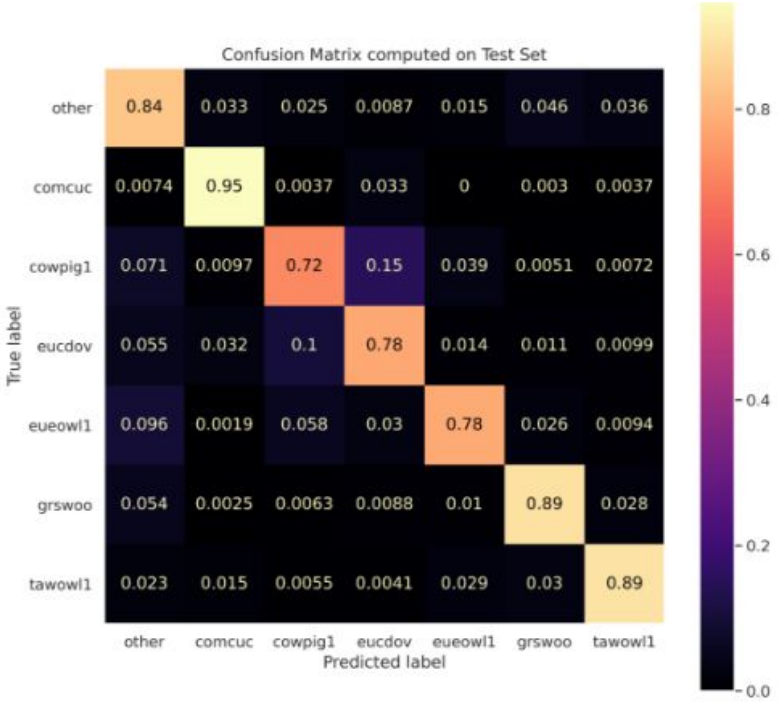
Random Forest Classifier (1)

- 73 Hyperparameters were explored using ParameterGrid from sklearn on the following
[{'n_estimators': [10, 100, 200, 300], 'min_samples_leaf': [1, 3, 5], 'min_samples_split': [2, 4, 6], 'n_jobs': [-1], 'bootstrap': [True, False]}, {'max_depth': [10], 'n_jobs': [-1]}].
- The main idea behind the chosen parameters was that they seem to have an impact on the underfitting/overfitting behaviour.
- From the plot, it was inferred that fewer estimators (no. of trees) and more min_samples_leaf (the minimum number of samples required to be at a leaf node) with bootstrap set to True generally lead to low performance on training and validation set. This corresponds to the beginning of the plot.
- Strong overfitting (98-100% train but ≈80% validation accuracy corresponding to sharp downward peaks in the plot) was observed for 10 estimators with min_samples_leaf=1 which makes sense as it was also mentioned that more estimators (trees) and/or making leaf nodes with more than 1 sample can help reduce overfitting by decreasing variance/increasing bias.
- To conclude, we see that sometimes random forest overfits quite well on the training data and this behaviour is mostly influenced by n_estimators and min_samples_leaf.



Random Forest Classifier (2)

- Finally, the best hyperparameter ({'bootstrap': False, 'min_samples_leaf': 1, 'min_samples_split': 2, 'n_estimators': 300, 'n_jobs': -1}) was selected the way it was mentioned in the Training Pipeline slide.
- A balanced test set accuracy of **0.8345** was achieved.
- The confusion matrix shows that the model performs decently in classifying all the classes. However it sometimes confuses ‘cowpig1’ with ‘eucdov’ and vice-versa.
- Overall, it also had quite good performance among all classifiers.



SVM (1)

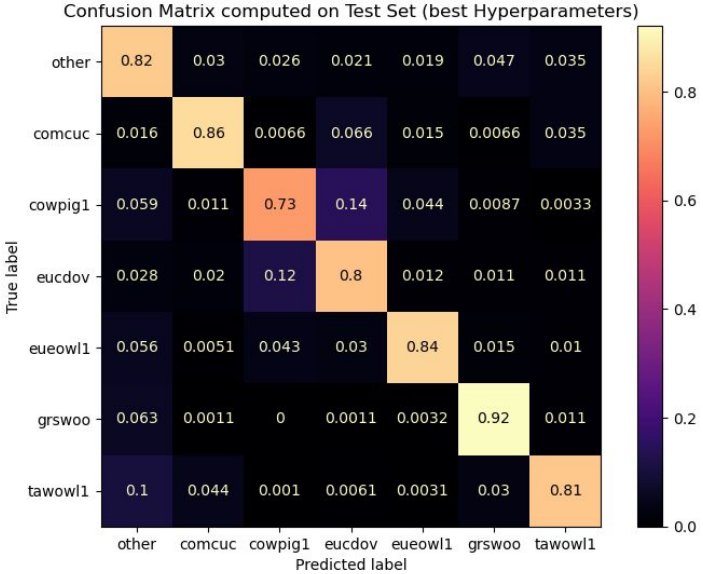
- 1350 Hyperparameter combinations were originally tested (ParameterGrid from sklearn).
- The result of this testing was, that the gamma value had no significant influence on the results (27 gamma values tested: np.linspace(1e-3, 1, 25) + [‘scale’, ‘auto’]). In later testing gamma was omitted and set to the default value of ‘auto’ or ‘scale’ since these performed the best. ‘auto’ and ‘scale’ give the same value for gamma since the training data is normalized such that mean=0 and var=1. ‘auto’ uses the formula 1/n_features and ‘scale’ 1/(n_features * X.var()).
- C has more impact on the result but differences are also very slow for higher C values. First grid search: np.linspace(1e-1, 100, 25), later on np.linspace(1, 20, 20).
- ‘rbf’ and ‘poly’ kernels were tested and ‘rbf’ performed significantly better without introducing more hyperparameters to tune. C values below 1 actually worsen the result on the training set as well as on the validation set.
- The best combination of the first search: [{'C': 12, 'gamma': 'auto', 'kernel': 'rbf'}].
- For further testing the ParameterGrid was reduced to only using ‘rbf’ kernel with gamma value ‘scale’ and C=np.linspace(1, 20, 20).
- The idea was to observe how different hyperparameter settings influence the performance of the classification.
- Overfitting didn’t occur during testing of the mentioned parameters. When the validation accuracy dropped also the test accuracy was worse.

SVM (2)

- The last experiment conducted was to check how the accuracy changes with a different number of features.
- As expected the more features the better the predictions. The difference between having all features and having all with a correlation coefficient of more than 0.9 removed is so tiny that the speed improvement is more preferable.
- For this experiment the gamma value was fixed to ‘auto’. In the table on the right the results are shown. Only Odd C values are visible.
- The first column in the table shows the threshold value which was used to remove features by the correlation coefficient.
  - 0.5: only 79 features left
  - 0.6: only 95 features left
  - 0.7: only 123 features left
  - 0.8: only 160 features left
  - 0.9: only 268 features left
  - 1.0: 548 features left
- On the reduced features, the C value 20 yielded the best result with a balanced accuracy of 0.9118 on the validation set and **0.8264** on the test set.

Balanced Accuarcy with different C values and thresholds (validation set)

	C=1	C=3	C=5	C=7	C=9	C=11	C=13	C=15	C=17	C=19	Best BAcc
0.5	0.8592	0.8778	0.8748	0.8711	0.8726	0.8741	0.8733	0.8696	0.8652	0.8645	0.8778
0.6	0.8726	0.8933	0.8948	0.8934	0.8934	0.8904	0.8904	0.8926	0.8911	0.8911	0.8948
0.7	0.8844	0.8978	0.9	0.8956	0.8948	0.8948	0.8941	0.8948	0.8956	0.8971	0.9
0.8	0.8948	0.9111	0.9067	0.9082	0.9052	0.9022	0.9008	0.9022	0.9015	0.9008	0.9111
0.9	0.8903	0.9015	0.9059	0.9074	0.9082	0.9096	0.9089	0.9096	0.9104	0.9111	0.9111
1.0	0.8866	0.9022	0.9111	0.9133	0.9163	0.9126	0.9104	0.9104	0.9096	0.9089	0.9163

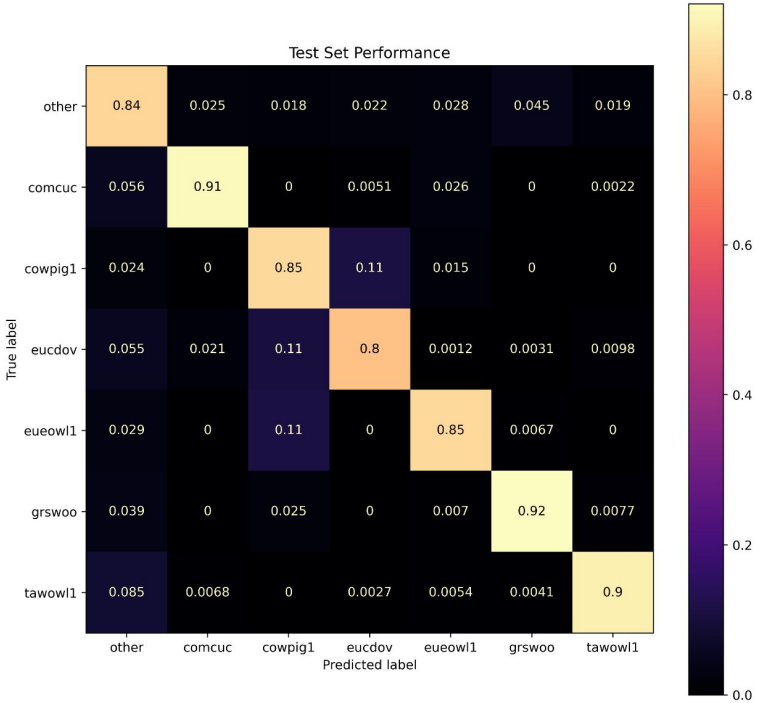


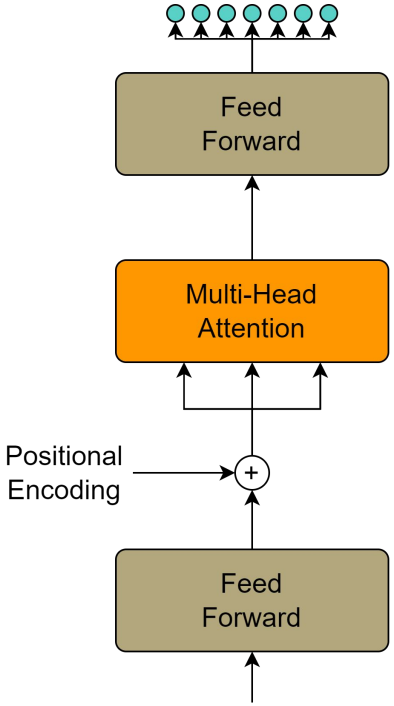
Transformer

- We tried using pytorch's Transformer to make use of the information of all timesteps in the sequence to predict the class of each timestep. However, despite trying many different hyper-parameters, the Transformer never achieved a training accuracy higher than baseline (bacc ≈ 0.14). We hypothesize that the Transformer is too complex to be properly trained by the data that we have.
- Since the Transformer was not made for our use-case (n to n classification) and did not seem to perform well, we created a custom neural network architecture which we called **Attention Classifier**.

Attention Classifier (1)

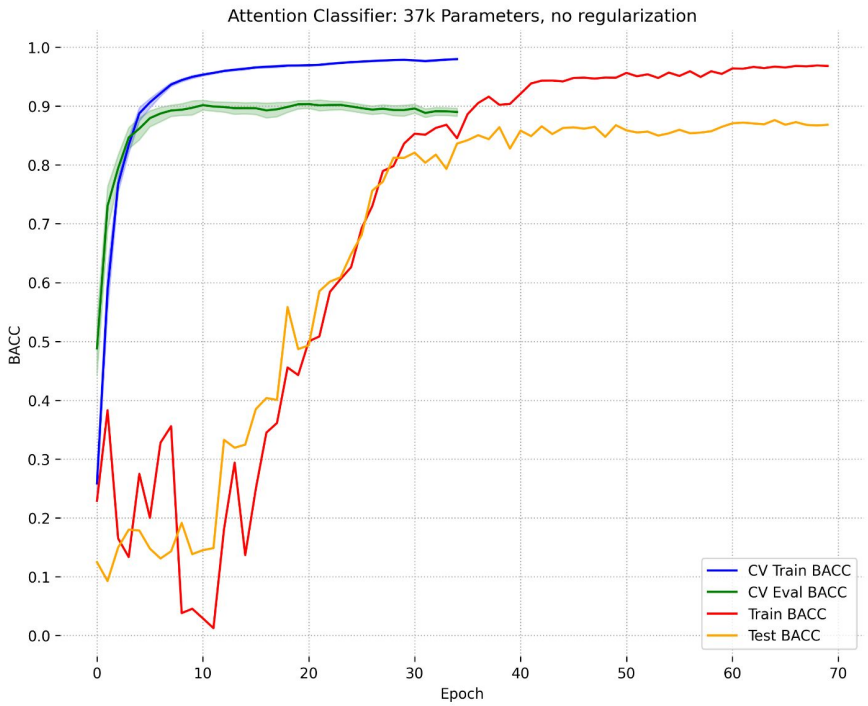
- The dataset itself was not balanced, but loss weights were applied to the CrossEntropyLoss in order to mitigate the unbalanced nature of the data.
- Overfitting occurred when training the model for over 10 epochs in CV and for over 40 epochs when training on the whole train data.
- We employed the Adamax optimizer.
- The best performance was achieved with a particularly low parameter count (≈37k).
- Test set bacc of **0.868** (best among all) was achieved.





Attention Classifier (2)

- Up until now an unexplainable phenomenon is that the BACC progression through the epochs looks very different when training on the CV training folds compared to when training on the whole training data, despite the whole training data containing only 11% more data-samples than each of the CV training folds.



Conclusion

- All classifiers performed decently but FNN, Histogram Based Gradient Boosting and the Attention Classifier were the best.
- Some classifiers struggled in recognizing ‘cowpig1’ and/or ‘eucdov’. Furthermore, most classifiers confused ‘cowpig1’ and ‘eucdov’ with each other.
- For the classification challenge, we figured, combining either the best classifiers or first training a binary classifier that distinguishes between the ‘other’ class and all the birds and then using a classifier that only classifies the birds.
- Balancing the dataset was crucial to obtaining reasonable predictions for all the different classes.