Introduction

Team assignment from: Ilya Malkerov and Jaden Mannes

This is a project about classifying the species of penguins with a neural network based on their measurements using a highly accurate neural network classifier. This dataset contains observations about three species of palmer penguins: Adeline, Chinstrap, and Gentoo.

The columns are:

- Species (species). The variable to predict. There are three classes.
- Which island are they from? (island) It is a categorical variable.
- Lengths and heights of their bills and the length of their flippers (bill_length_mm, bill_depth_mm, flipper_length_mm)
- Body mass (body_mass_g)
- Sex (sex)
- The year in which that penguin was observed (year)

Note that due to how these columns are named, there is no ambiguity about the units of measurements used. The only slight ambiguity was the “year” column. Is it the specimen’s birth year or the year in which that specimen was observed? However, we found the source of this dataset and confirmed that the year column refers to the year in which the specimen was observed.

Note: to make results reproducible, a fixed value of 42 was used for all random seeds, and TensorFlow’s deterministic operations were enabled.

Preprocessing steps

Rows with any missing columns are dropped (df.dropna()) (done directly in the load_data function). This is because we decided that we wanted to consider all columns for our model to yield the most accurate results, and because there are only very few rows with missing values.

The main preprocessing function (make_preprocessor) separates the ‘species’ column from the rest of the columns. Then, numerical columns are identified using ‘select_dtypes(include=np.number)’ and categorical columns are identified by considering all non-numerical columns except for ‘species’ to be categorical.

Numerical columns are normalized using the StandardScaler; categorical columns are encoded into numerical columns using the OneHotEncoder.

80% of the data gets split off into training data, while the remaining 20% is used as the testing dataset. The training dataset is then once again split into a 75% chunk used for training, and a 25% chunk for internal validation. Stratified splitting is used to maintain an even amount of every penguin species between all split chunks of the dataset.

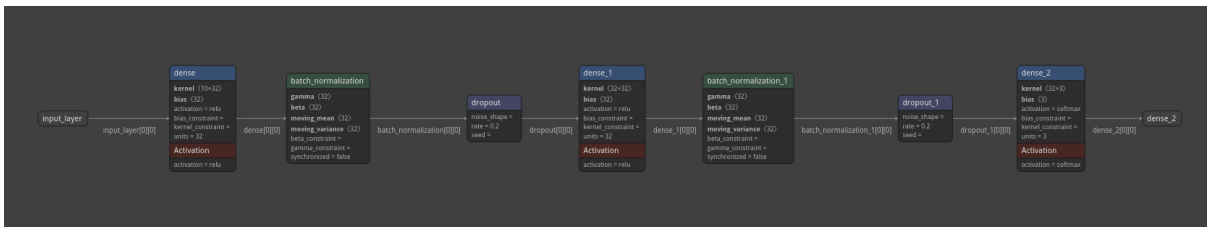
Model architecture and training

The classification model was implemented as a fully connected feed-forward neural network built with TensorFlow / Keras.

The network architecture consists of one to two hidden layers, each containing 16–64 neurons depending on the hyperparameter configuration being tested. Every hidden layer uses the ReLU activation function followed by batch normalization and, when specified, dropout regularization to reduce overfitting. The output layer contains three soft-max neurons, corresponding to the three penguin species in the *Palmer Penguins* dataset. L2-regularization is applied to the dense layers when the hyperparameter $l2 > 0$.

Training was performed with a categorical-cross-entropy loss function and accuracy as the secondary evaluation metric. To improve convergence and stability, early stopping and a learning-rate-reduction callback were used. Each run used a maximum of 200 epochs with a patience of 20 epochs for early stopping. Class imbalance was handled by computing class-specific weights through ‘compute_class_weight’ so that all species contributed equally to the loss.

At a high level, the model looks like this (visualization generated using netron.app). The model contains 2 hidden layers with 16 neurons each.



Hyperparameter tuning

Hyperparameter optimization was conducted automatically through a nested cross-validation procedure to avoid optimistic bias. The outer loop (5 folds) estimated generalization performance, while the inner loop (3 folds) searched for the best hyperparameter combination. Within each inner loop, several optimizers (SGD, Adam, AdamW) and a grid of model-size and regularization parameters were tested. The

primary selection metric was macro-F1 score, with accuracy used as a secondary criterion for tie breaking.

Hyperparameter	Ranges tested	Notes
`opt`	{ `sgd`, `adam`, `adamw` }	three optimizers compared
`lr`	1e-4 – 1e-2 (log-spaced)	learning-rate grid
`momentum`	0.0 / 0.9	used only with SGD
`n_hidden`	1 – 2 layers	network depth
`width`	16, 32, 64 neurons	hidden-layer width
`l2`	0, 1e-6, 1e-5, 1e-4, 1e-3	L2 regularization strength
`dropout`	0.0, 0.2, 0.5	dropout probability

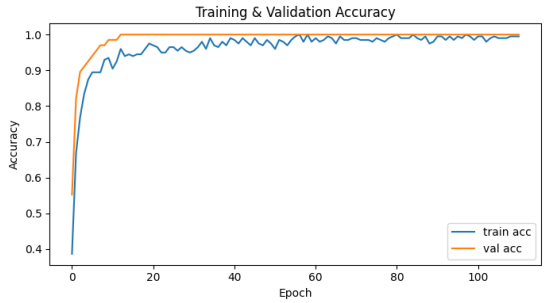
In total, up to 24 randomly sampled configurations were evaluated per inner loop to keep runtime manageable. For every candidate's set of hyperparameters, the inner cross-validation mean macro-F1 was computed; the configuration with the highest mean F1 was chosen for that outer-fold model. After completing all outer folds, a consensus set of hyperparameters or, the most frequently selected combination across folds was identified and retrained once on the full training + validation data to produce the final deployable model.

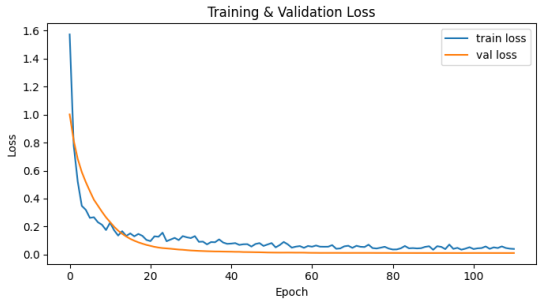
The submitted Python file has the best set of hyperparameters found through these methods.

Setup and outcomes of the evaluation experiment

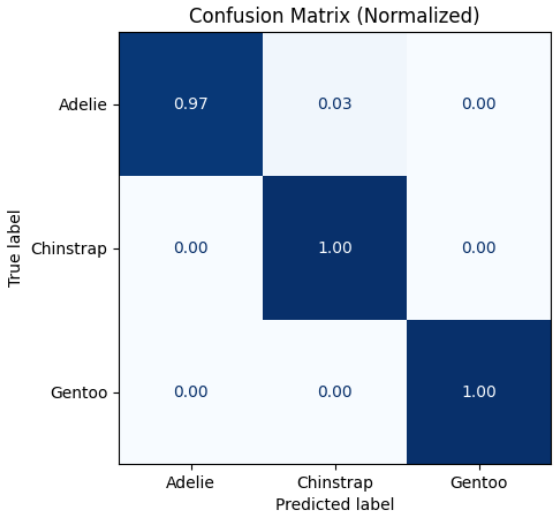
The model achieved a test accuracy of 0.985 and a macro-F1 score of 0.983.

In addition, three graphs were generated to visualize the model's accuracy.





The learning-curve plots, plotting training/validation accuracy and loss over epochs, show early plateauing of both training and validation accuracy, indicating good generalization without overfitting.



The normalized confusion matrix confirms that all three species were classified correctly with minimal confusion.

Additionally, in a shuffle label sanity check, the model's accuracy dropped to about 48%, once again confirming that it produces good predictions.

Conclusions

Overall, this dataset was very clean and easy to work with, which made it easy to create a neural network that could accurately predict the species of palmer penguins based on the provided data. This resulted in a very high model accuracy without any signs of overfitting.

References

Based on knowledge from

Course materials (PowerPoints)

Lectures

Cawley, G. C., & Talbot, N. L. C. (2010). On over-fitting in model selection and subsequent selection bias in performance evaluation. Journal of Machine Learning Research, 11(Jul), 2079-2107.

Varma, S., & Simon, R. (2006). Bias in error estimation when using cross-validation for model selection. BMC Bioinformatics, 7(1), 91. https://doi.org/10.1186/1471-2105-7-91

Powers, D. M. W. (2011). Evaluation: From precision, recall and F-measure to ROC, informedness, markedness & correlation. *Journal of Machine Learning Technologies, 2*(1), 37-63.

Hand, D. J., & Christen, P. (2018). A review of metrics for evaluating performance of machine learning algorithms. *Wiley Interdisciplinary Reviews: Data Mining and Knowledge Discovery, 8*(4), e1255. https://doi.org/10.1002/widm.1255

Domingos, P. (2012). A few useful things to know about machine learning. *Communications of the ACM, 55*(10), 78-87. https://doi.org/10.1145/2347736.2347755

Feurer, M., & Hutter, F. (2019). Hyperparameter Optimization. In F. Hutter, L. Kotthoff, & J. Vanschoren (Eds.), *Automated Machine Learning: Methods, Systems, Challenges* (pp. 3-33). Springer.