Individual Final Report

1. Introduction

The goal of the final project is to develop a neural network model capable of performing multi-label classification of bird species based on environment audio recordings. This challenge was inspired by a Kaggle competition where participants were tasked with identifying bird calls in complex soundscape recordings. Each audio clip could contain one or multiple bird species, making this a multi-label classification problem that required careful handling of overlapping class signals.

Our team collaborated on key aspects of the project, including understanding the dataset and class distributions, training and evaluating models, visualizing results, and compiling the final report. We adopted a CNN-based model, EfficientNet-B0, trained on mel spectrograms derived from .ogg audio files and evaluated its performance using metrics such as AUC.

To support usability and result demonstration, we also worked on designing an interactive Streamlit application. This app allows users to upload audio files and view model predictions in a user-friendly format. The Streamlit interface was developed as part of our effort to make the model more accessible and interpretable for users without requiring them to run code manually.

To simulate a realistic test environment, we implemented a custom validation setup that split the original training data into training and test subsets while preserving label distributions. This approach allowed us to validate model performance on unseen data without relying on Kaggle's hidden test set.

This report details my individual contributions to the project, the model evaluation results, and the rationale behind the technical and design choices I made throughout the process.

2. Background

Because the Kaggle test set was only available for final submission and not accessible during model development, we needed to create our own local validation set to properly evaluate model performance. To accomplish this, I developed a script named create_test_soundscapes.py that replicates the structure of the Kaggle test environment.

The script begins by loading the original train.csv file, which contains metadata for all training audio files. In order for the label distribution to be preserved in both training and validation sets, the script performs a stratified split based on the primary_label column. Specifically, it divides the data into 80% for training (train_split.csv) and 20% for validation (valid_split.csv).

Next, the script identifies the corresponding audio files in the train_audio/ directory and copies the validation files into a new directory named test_soundscapes/. During this step, each file is renamed following the Kaggle convention using the format soundscape_XXXXXX.ogg, where XXXXXX is a zero-padded index. This renaming ensures compatibility with our test script and mimics how Kaggle handles its test set internally.

Finally, the script creates a new metadata file called test_metadata.csv, which logs the new filenames and their corresponding primary labels. This file serves as the reference ground truth for evaluating model predictions during local testing.

3. Description of Individual Work

Since I joined the project after the dataset and proposal had been finalized by my teammates, I first took time to familiarize myself with the data. I downloaded the dataset from Kaggle and explored its structure, paying attention to the class distributions and metadata to ensure I understood the problem scope and the multi-label classification task.

To improve modularity and comply with the professor's required directory structure, I refactored the code by separating functionality into distinct Python scripts. I created class definitions and utility functions in their respective .py files under the src/ directory. This organization enhanced maintainability and made collaboration easier for my teammates.

Next, I reviewed the baseline code from a Kaggle notebook referenced by the team. After ensuring the code worked locally, I restructured it into two scripts: train_to_do.py for model training and test_to_do.py for evaluation. This setup allowed my teammates to run training and inference independently on their local machines.

During testing, I discovered that the test_soundscapes folder was initially empty because the original Kaggle notebook used hidden test data for submission. I wrote a script to simulate the Kaggle test setup by randomly selecting 700 samples from the training data and populating the test_soundscapes folder accordingly. Upon evaluation, this approach yielded an AUC of 0.9806 (see Figure 1), which was suspiciously high. After discussing with the team, we concluded that the model had likely already seen those samples during training.

To address this issue, I revised my approach by splitting train.csv into two subsets: train_split.csv (80%) for model training and valid_split.csv (20%) for validation. I modified both the training and testing scripts to correctly read from these new splits. This ensured a proper evaluation protocol and prevented data leakage.

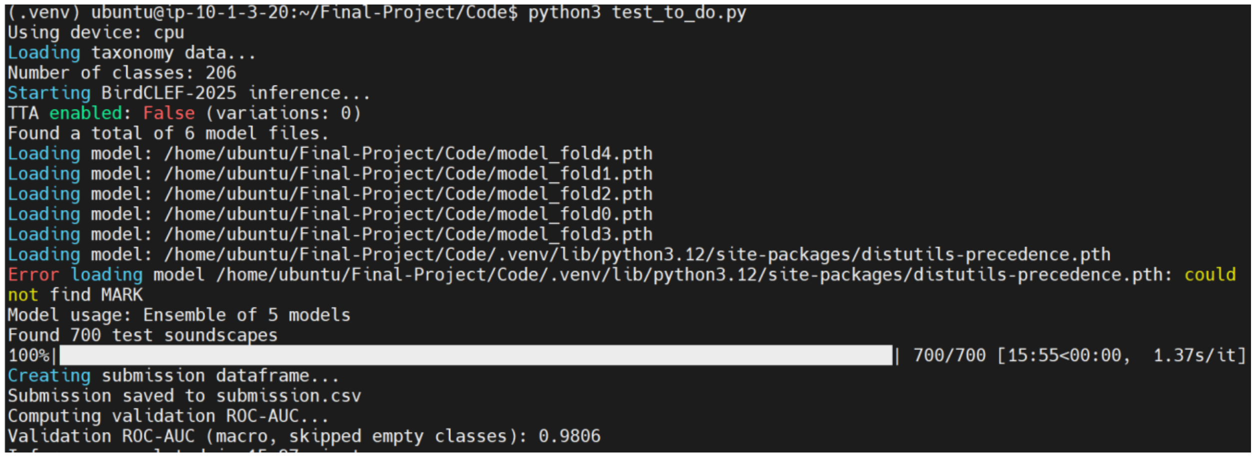
Additionally, I implemented model evaluation visualizations such as ROC curves and metric plots to assess performance. I also took the lead in drafting and organizing the majority of the final project report.

4. Results

After restructuring the data pipeline and establishing proper training/validation splits, I retrained the model on train_split.csv and validated it on valid_split.csv. The model performance was significantly more realistic compared to the earlier test with reused samples.

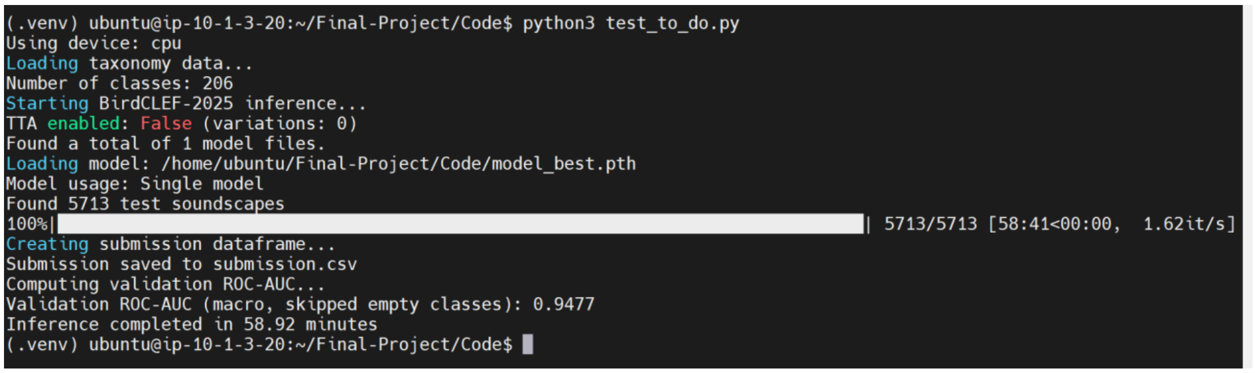
As shown in Figure 1, the initial model evaluation on reused data yielded an AUC of 0.9806. However, this result was inflated due to data leakage. After correcting the evaluation protocol, the model achieved a more reasonable validation AUC score (0.9477) as seen in Figure 2.

Figure 1. The Initial Model Evaluation



```
(.venv) ubuntu@ip-10-1-3-20:~/Final-Project/Code$ python3 test_to_do.py
Using device: cpu
Loading taxonomy data...
Number of classes: 206
Starting BirdCLEF-2025 inference...
TTA enabled: False (variations: 0)
Found a total of 6 model files.
Loading model: /home/ubuntu/Final-Project/Code/model_fold4.pth
Loading model: /home/ubuntu/Final-Project/Code/model_fold1.pth
Loading model: /home/ubuntu/Final-Project/Code/model_fold2.pth
Loading model: /home/ubuntu/Final-Project/Code/model_fold0.pth
Loading model: /home/ubuntu/Final-Project/Code/model_fold3.pth
Loading model: /home/ubuntu/Final-Project/Code/.venv/lib/python3.12/site-packages/distutils-precedence.pth
Error loading model /home/ubuntu/Final-Project/Code/.venv/lib/python3.12/site-packages/distutils-precedence.pth: could
not find MARK
Model usage: Ensemble of 5 models
Found 700 test soundscapes
100%|████████████████████████████████████████████████████████████████| 700/700 [15:55<00:00,  1.37s/it]
Creating submission dataframe...
Submission saved to submission.csv
Computing validation ROC-AUC...
Validation ROC-AUC (macro, skipped empty classes): 0.9806
```

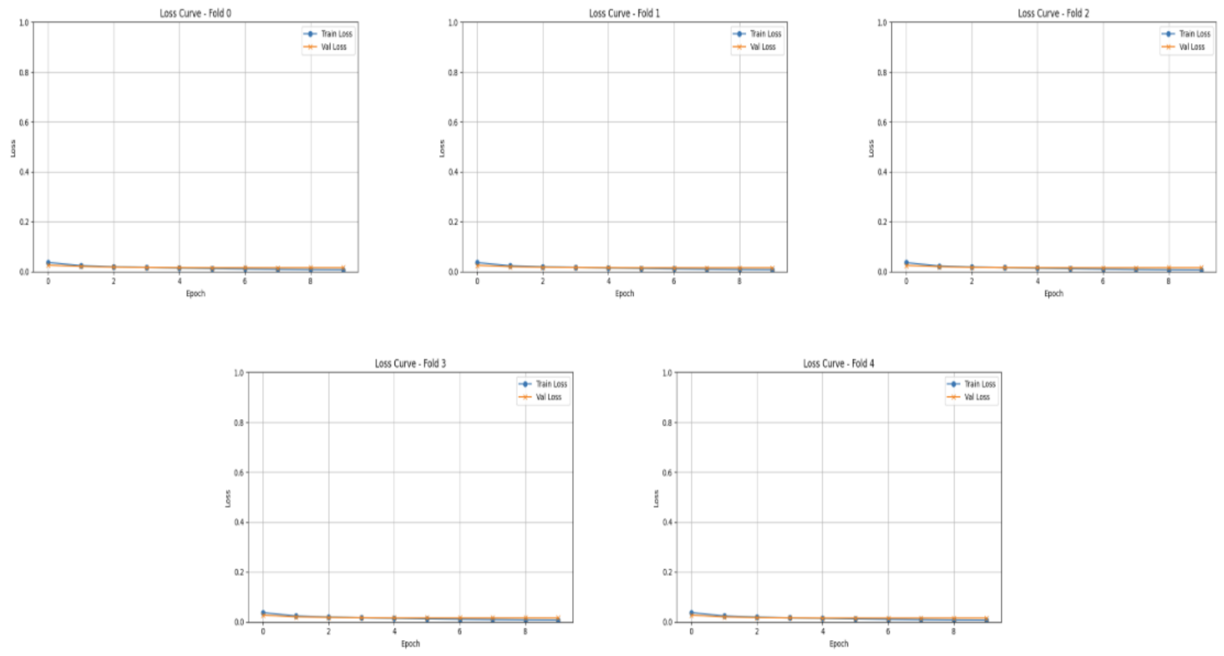
Figure 2. The Model Evaluation After Modification



```
(.venv) ubuntu@ip-10-1-3-20:~/Final-Project/Code$ python3 test_to_do.py
Using device: cpu
Loading taxonomy data...
Number of classes: 206
Starting BirdCLEF-2025 inference...
TTA enabled: False (variations: 0)
Found a total of 1 model files.
Loading model: /home/ubuntu/Final-Project/Code/model_best.pth
Model usage: Single model
Found 5713 test soundscapes
100%|██████████████████████████████████████████████████████████████| 5713/5713 [58:41<00:00,  1.62it/s]
Creating submission dataframe...
Submission saved to submission.csv
Computing validation ROC-AUC...
Validation ROC-AUC (macro, skipped empty classes): 0.9477
Inference completed in 58.92 minutes
(.venv) ubuntu@ip-10-1-3-20:~/Final-Project/Code$ █
```

In addition to AUC, we visualized model performance using metric such as Loss Curve for each fold. Figure 3 displays the Loss curves for each fold, showing the model stability and not overfitting.

Figure 3. Loss Plots For Each Fold



5. Summary and Conclusions

In this project, our team developed a convolutional neural network model using EfficientNet-B0 to perform multi-label classification of bird species from environmental audio recordings. The task was inspired by a Kaggle competition and involved several real-world challenges, such as overlapping class labels and the absence of accessible test data. To address these, we implemented a robust local validation framework that mimicked the Kaggle testing setup by splitting the dataset while preserving class distributions and simulating test files.

My individual contributions focused on ensuring the codebase was well-organized, testable, and reproducible. I restructured the scripts for training and evaluation, implemented the custom validation split through the create_test_soundscapes.py script, and created visualizations for performance metrics. I also led the writing of the project report and ensured our model evaluation was both fair and informative.

Our experimental results showed that initial testing on seen data gave an inflated AUC of 0.9806. After correcting for data leakage using a proper validation set, the model's AUC score adjusted to a more reliable 0.9477. Additional visualizations such as loss curves and metric breakdowns helped provide deeper insights into the classifier's strengths and weaknesses across different bird species.

In conclusion, the project not only achieved high classification performance on a challenging dataset, but also demonstrated the importance of rigorous evaluation practices. The development of an interactive Streamlit application further enhanced the usability of our model, allowing users to make predictions on their own audio files without needing to interact directly with the codebase. This project reflects our team's strong collaboration and practical application of machine learning techniques to solve a real-world audio classification problem.

6. Calculate The Percentage of Code

The percentage is $\frac{643-16}{643+156} \times 100 = 78.50\%$

7. References

Kaggle. (2025). BirdCLEF 2025 competition data. Retrieved April 30, 2025, from [URL]

Sharma, A. (2024, April 12). EfficientNetB0 architecture: Stem layer. *Image Processing with Python* (Medium). Retrieved April 30, 2025, from [URL]