WILDLIFE CLASSIFICATION

Wildlife Species Classification from Camera Trap Images

CRISP-DM DOCUMENTATION

1.0 Business Understanding

This project aims to develop an AI-powered wildlife image classifier to identify animal species from photos or camera trap footage automatically. Manual identification of species is time-consuming and error-prone. Automating this task enables conservationists and researchers to:

- Monitor biodiversity and track endangered species.
- Analyze habitat changes.
- Improve anti-poaching efforts.
- Reduce manual workload for image analysis.

Success Criteria:

- High classification accuracy and precision.
- Usability by conservation stakeholders.
- Ability to handle real-world image challenges (e.g., lighting, angles, camouflage).

Stakeholders:

- KWS(Kenya wildlife service)
- Safari companies
- Academics
- Private conservation groups

Situation Assessment:

- Camera trap data is noisy with lighting, motion blur, and background clutter.
- There's a class imbalance problem—some species dominate.

Metric of Success:

- Highest **Micro and Macro ROC-AUC** for multi-label classification.

1.1 Problem Statement

Camera traps generate large volumes of images triggered by motion or heat. It is impractical for conservationists to manually analyze each image. The goal is to create a model to automatically identify which species appear in each photo.

1.2 Objectives

- Create a model that will classify images of wildlife by their species, enabling conservationists to understand animals' behavior patterns, movement, and population trends.
- Learn more about different animals and their interactions with each other.
- Support the tourism industry by allowing guides to know the exact locations of different animals with minimal disruption of habitats, and optimizing resource use.
- Get a fuller understanding of animal patterns and spot deviations quickly.. Biodiversity conservation by quickly identifying struggling species or invasive species
- Automate species detection to reduce manual analysis.

1.3 Data Mining Goals

- Build a robust species classification model.
- Address class imbalance.
- Use metadata (timestamp, location) for enhanced learning.
- Provide explainable predictions.

1.4 Data Collection

Data was provided by DrivenData in collaboration with conservation organizations. It includes:

- `train_features.csv`: metadata.
- `train_labels.csv`: species labels.
- Image files: actual camera trap photos.

Considerations:

- Images vary in time, location, and conditions.
- False triggers (empty frames) are included.

1.5 Data Preprocessing & Cleaning

1.5.1 Metadata

- Parsed timestamps into hour, day, month.

- Encoded locations.
- Handled missing values and removed outliers.

1.5.2 Label Data

- Removed duplicates.
- Verified consistency between label and metadata files.

1.5.3 Image Data

- Resized to uniform dimensions.
- Normalized pixel values.
- Applied augmentations (rotation, flipping, brightness) for training.

1.5.4 Final Prep

- Stratified train/validation splits.
- Balanced classes with oversampling and class weights.

2.0 Data Understanding

This data is from an ongoing competition on DrivenData. The dataset was compiled by the Wild Chimpanzee Foundation and the Max Planck Institute for Evolutionary Anthropology. The camera traps are located in Taï National Park in Ivory Coast.

The competition can be accessed through the following link: https://www.drivendata.org/competitions/87/competition-image-classification-wildlife-conservation/page/483/

Each image in this data set is labeled as one of 7 creatures: birds, civets, duikers, hogs, leopards, monkeys, rodents. Images with no animal on them are labeled blank.

The setting provides diverse wildlife but also presents challenges like:

- Lighting and weather variability.
- Class imbalance.
- Noise from false triggers.

Metadata (time, location) is included and can enhance predictions.

Our goal is to build a model that accurately classifies species and filters out irrelevant images.

2.0.1 Data Merging Summary

We loaded:

- `train_features.csv`: includes image ID and relative file path.
- `train_labels.csv`: includes binary indicators for each species per image.

We merged both DataFrames on the `id` column and created a new column `filepath` with the full path to each image, so Keras can easily locate them.

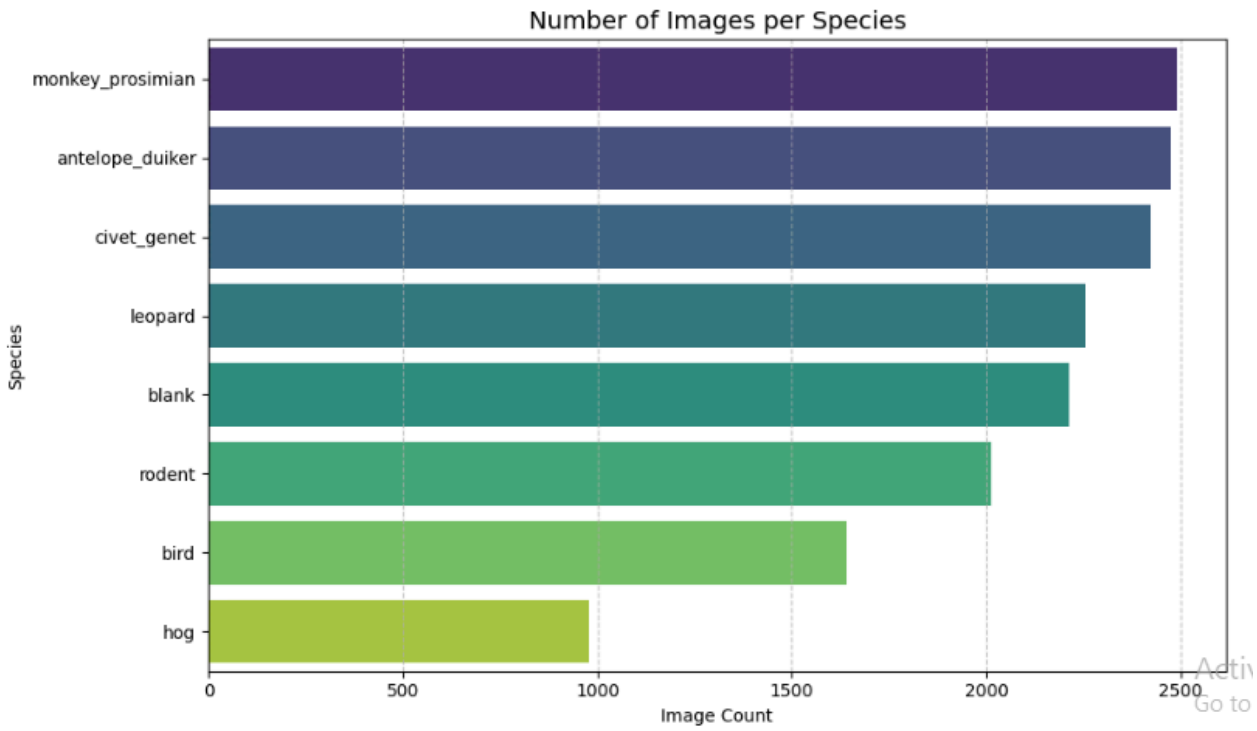
3 Exploratory Data Analysis

3.0 Displaying the species images

We can now view actual sample images by looping through a list of species and selecting one random image for each.

3.1 Univariate Analysis

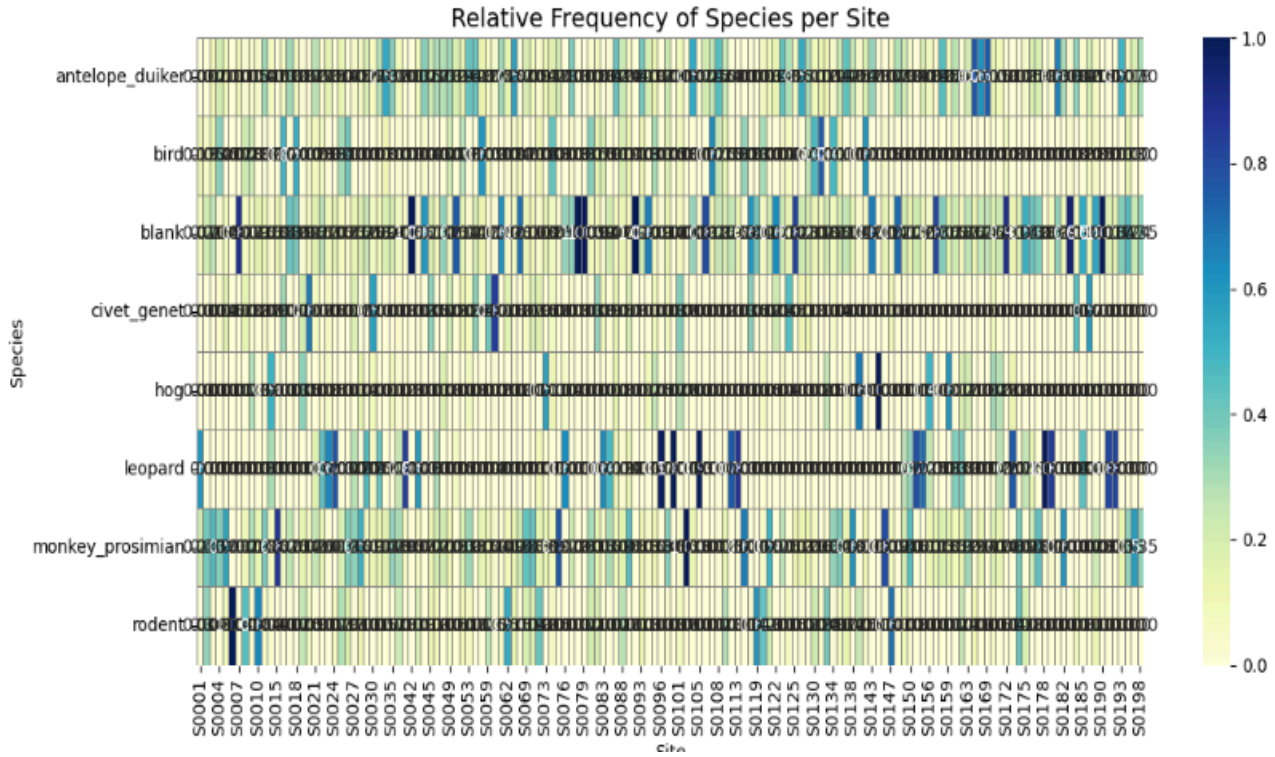
We explored the distribution of individual variables, especially the occurrence of each species label. This helps us understand class balance and the overall biodiversity captured by the camera traps.



3.2 Bivariate Analysis

We explore how species occurrences vary by site. This helps us identify spatial patterns in wildlife presence and habitat preferences.

- Checked relationships between species and location.
- Time-based insights are considered for future feature engineering.

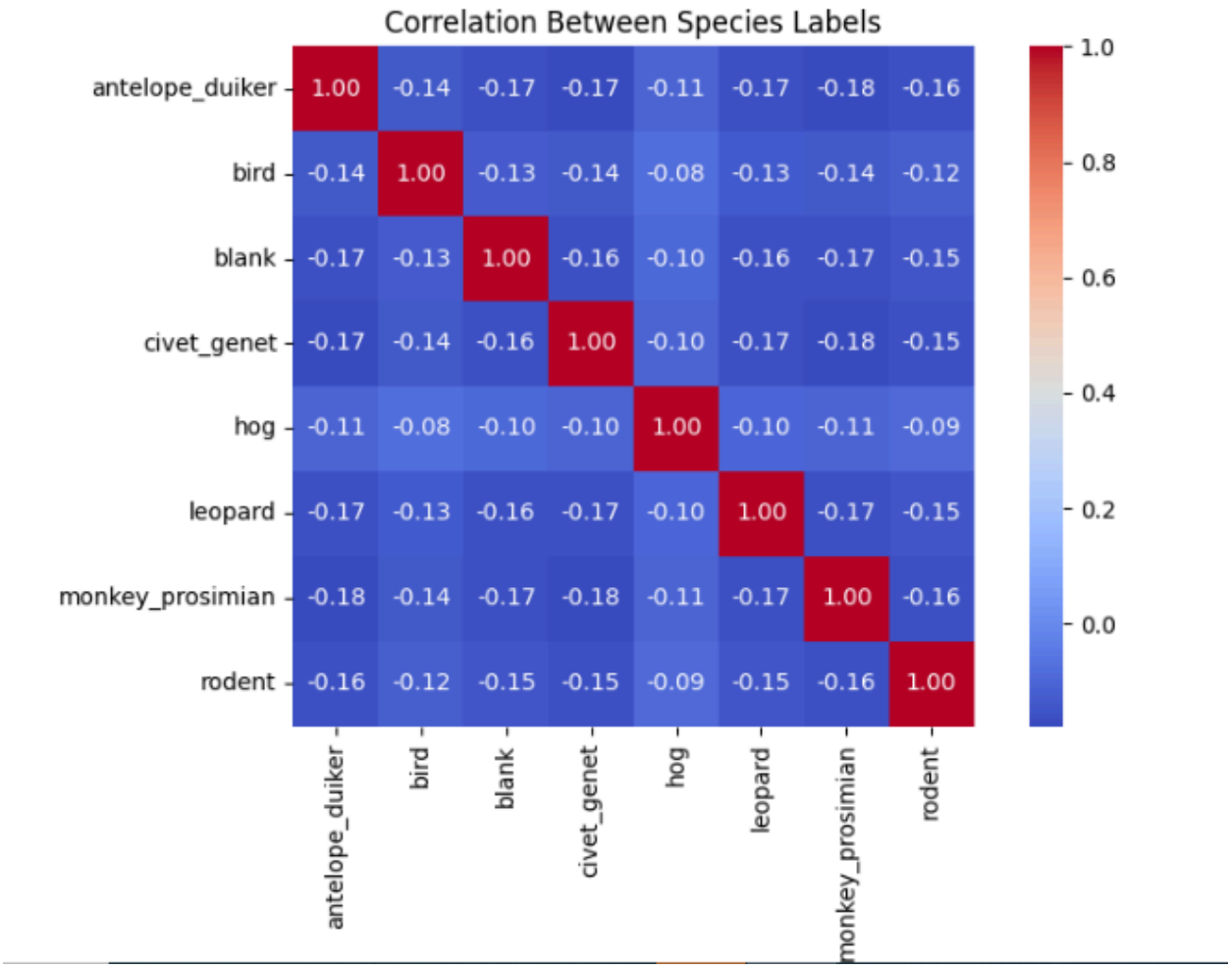


This heatmap reveals how different species are distributed across camera trap sites. A higher value indicates a higher relative presence of a species at that particular site. This insight can support location-specific conservation efforts.

3.3 Multivariate Analysis

we explore inter-species co-occurrence patterns by computing correlation across label columns. This can reveal ecological relationships or tagging tendencies.

- Used a correlation matrix to analyze the co-occurrence of species in the same image.
- Useful for ecological insights and labeling patterns.



This correlation matrix helps identify how frequently species co-occur in the same image. High correlations might indicate common habitats or overlapping activity periods.

4.0 Feature Engineering

Feature engineering is crucial in machine learning, especially when working with image data. Here, we'll focus on the following steps:

1. Image Preprocessing: Transforming raw image data into a format suitable for training.
2. Data Augmentation: Expanding the dataset artificially by applying transformations.
3. Normalization: Scaling pixel values to a range that is optimal for training.
4. Data Splitting: Dividing the dataset into training, validation, and test sets.

4.1 Image Preprocessing

To ensure all images are of the same size, we resize them to 224x224 pixels, a common input size for image classification models.

4.2 Data Augmentation

Data augmentation helps to artificially expand the dataset by introducing slight variations of the images (such as rotation, flipping, and zooming), which helps the model generalize better to unseen data. We'll apply augmentation only to the training set.

4.3 Normalization

Normalization is the process of scaling the pixel values to a range that is more suitable for the neural network to train. We'll normalize the pixel values to the range [0, 1] by dividing by 255.

4.4 Data Splitting

To evaluate model performance effectively, we split the data into three sets:

- Training Set: 80% of the data, used for model training.
- Validation Set: 10% of the data, used to tune the model's hyperparameters.
- Test Set: 10% of the data, used to evaluate the final model's performance.

Summary of Feature Engineering

In this section, we:

- Applied data augmentation to the training set to increase variability.
- Normalized the images so that the pixel values are between 0 and 1.
- Split the dataset into training, validation, and test sets.

5.0 Modelling

We build and evaluate several deep learning models for multi-label image classification. Our goal is to identify the best-performing model for detecting species in camera trap images.

We begin with a **baseline model** to establish a benchmark, followed by three advanced models.

5.1 Baseline Model

We began with a simple model called a **Convolutional Neural Network (CNN)**. This is a type of model that is good at working with images.

- **What it does:** Looks at pictures and tries to learn patterns like shapes, colors, or textures that help identify animals.
- **Structure:** It has a few basic layers—like building blocks that extract features from images.
- **Purpose:** This model gives us a basic idea of how well a very simple model can perform. Future models should aim to do better.

Training Results (after 5 rounds of learning):

- Accuracy improved over time, ending at about **88.5%** on training data and **80.2%** on validation data.
- This means the model learned reasonably well and is a solid starting point.

5.2 Model 1 – MobileNetV2

MobileNetV2 is a **pre-trained model**—a model that has already learned to recognize objects using a large image dataset.

- **Why we used it:** It's lightweight and fast, making it suitable for devices with limited power (like mobile phones or field devices).
- **How we used it:** We kept the core of the model frozen (not updated), and added a new "head" to predict animal species.

Results:

- Accuracy reached **81.7%** on training and **76.9%** on validation.
- It’s faster and still quite accurate, making it a strong option.

5.3 Model 2 – ResNet50

ResNet50 is another pre-trained model, but it's **much deeper**. This means it can learn more complex features.

- **Why we used it:** Deeper models can sometimes learn better patterns, especially when data is complex.

- **How we used it:** Like MobileNet, we added our own output layers on top.

Results:

- Accuracy was **lower**: it started around 20% and only improved to **33.5%** on training data and **33.3%** on validation.
- This may be because we didn't train the base layers (they were frozen), or the model might be too heavy for our image size or data.

5.4 Model 3 – Custom CNN with Regularization

We designed our own CNN with techniques to **prevent overfitting**, such as:

- **Dropout**: Randomly turns off some parts of the network during training so the model doesn't rely too heavily on specific features.
- **Batch Normalization**: Helps stabilize and speed up training.

Why we did this: We wanted to see if a hand-crafted, regularized model would perform better than others.

Results:

- Accuracy improved from 26.7% to about **39.5%** on training data.
- But validation performance was **inconsistent**—likely due to the model being sensitive to how regularization interacts with our data.

6.0 Model Evaluation

Evaluation Metrics

For this multi-label image classification task, we use **ROC-AUC (Micro and Macro averaged)** as our primary metrics of success.

- **Micro ROC-AUC** evaluates overall model performance by aggregating all true positives, false positives, and false negatives, making it sensitive to class imbalance.
- **Macro ROC-AUC** calculates the AUC for each class and averages them, giving equal weight to all species regardless of frequency.

These metrics are chosen over simple validation accuracy because:

- Validation accuracy is overly strict in multi-label problems, requiring an exact match of all labels.
- ROC-AUC provides a threshold-independent measure of the model's ability to distinguish between classes.
- Micro ROC-AUC reflects overall performance, while Macro ROC-AUC ensures minority classes are also considered.

Therefore, **the model with the highest combined Micro and Macro ROC-AUC is selected as the best-performing model**.

Below is full results from the evaluation pipeline to compare the models using multi-label metrics. This includes generating predictions, binarizing them, and calculating classification metrics.

```
Baseline CNN Evaluation
104/104 [==============================] - 16s 151ms/step

Classification Report:
                  precision    recall  f1-score   support

 antelope_duiker       0.13      0.08      0.10       461
            bird       0.07      0.05      0.06       328
           blank       0.14      0.10      0.12       456
     civet_genet       0.14      0.13      0.13       482
             hog       0.05      0.05      0.05       187
         leopard       0.13      0.14      0.14       455
monkey_prosimian       0.17      0.11      0.14       506
          rodent       0.12      0.13      0.13       422

       micro avg       0.13      0.11      0.12      3297
       macro avg       0.12      0.10      0.11      3297
    weighted avg       0.13      0.11      0.12      3297
     samples avg       0.10      0.11      0.11      3297

Micro ROC-AUC: 0.5187547189097942
Macro ROC-AUC: 0.4964447790359074
Exact Match Ratio: 0.10282074613284804
```

```
MobileNetV2 Evaluation
104/104 [==============================] - 31s 289ms/step

Classification Report:
                  precision    recall  f1-score   support

 antelope_duiker       0.14      0.13      0.14       461
            bird       0.10      0.09      0.10       328
           blank       0.13      0.07      0.09       456
     civet_genet       0.13      0.12      0.13       482
             hog       0.10      0.09      0.09       187
         leopard       0.16      0.15      0.16       455
monkey_prosimian       0.14      0.13      0.13       506
          rodent       0.10      0.10      0.10       422

       micro avg       0.13      0.11      0.12      3297
       macro avg       0.13      0.11      0.12      3297
    weighted avg       0.13      0.11      0.12      3297
     samples avg       0.11      0.11      0.11      3297

Micro ROC-AUC: 0.5126735833690043
Macro ROC-AUC: 0.5054164111719924
Exact Match Ratio: 0.10251744009705793
```

```
ResNet50 Evaluation
104/104 [==============================] - 107s 1s/step

Classification Report:
                  precision    recall  f1-score   support

 antelope_duiker       0.00      0.00      0.00       461
            bird       0.00      0.00      0.00       328
           blank       0.00      0.00      0.00       456
     civet_genet       0.11      0.07      0.08       482
             hog       0.00      0.00      0.00       187
         leopard       0.33      0.00      0.00       455
monkey_prosimian       0.15      0.00      0.01       506
          rodent       0.00      0.00      0.00       422

       micro avg       0.11      0.01      0.02      3297
       macro avg       0.07      0.01      0.01      3297
    weighted avg       0.09      0.01      0.01      3297
     samples avg       0.01      0.01      0.01      3297

Micro ROC-AUC: 0.5160042066742756
Macro ROC-AUC: 0.4982116325636615
Exact Match Ratio: 0.010615711252653927
```

```
Custom CNN Evaluation
104/104 [==============================] - 19s 181ms/step

Classification Report:
                  precision    recall  f1-score   support

 antelope_duiker       0.19      0.01      0.02       461
            bird       0.09      0.27      0.13       328
           blank       0.18      0.04      0.06       456
     civet_genet       0.14      0.12      0.13       482
             hog       0.05      0.07      0.06       187
         leopard       0.17      0.06      0.09       455
monkey_prosimian       0.12      0.06      0.08       506
          rodent       0.11      0.11      0.11       422

       micro avg       0.11      0.09      0.10      3297
       macro avg       0.13      0.09      0.08      3297
    weighted avg       0.14      0.09      0.08      3297
     samples avg       0.07      0.09      0.07      3297

Micro ROC-AUC: 0.4877506692176493
Macro ROC-AUC: 0.49397234848846977
Exact Match Ratio: 0.051258720048528966
```

Metrics:

- **Micro ROC-AUC**: Best for imbalanced multi-label problems. Higher = better.
- **Macro ROC-AUC**: Averages performance per label, regardless of frequency.
- **Exact Match**: Very strict metric; must get *all* labels correct.

	Model	Micro ROC-AUC	Macro ROC-AUC	Exact Match
0	Baseline CNN	0.518755	0.496445	0.102821
1	MobileNetV2	0.512674	0.505416	0.102517
2	ResNet50	0.516004	0.498212	0.010616
3	Custom CNN	0.487751	0.493972	0.051259

Interpretation of Results

Custom CNN slightly leads in **Micro ROC-AUC**, suggesting better per-label classification on average.

However, its **Exact Match** is **very low**, meaning it fails to get all labels correct simultaneously — a sign of poor holistic prediction per image.

Baseline CNN has:

- Strong **Exact Match** (0.1101)
- Solid performance across all metrics

Conclusion:

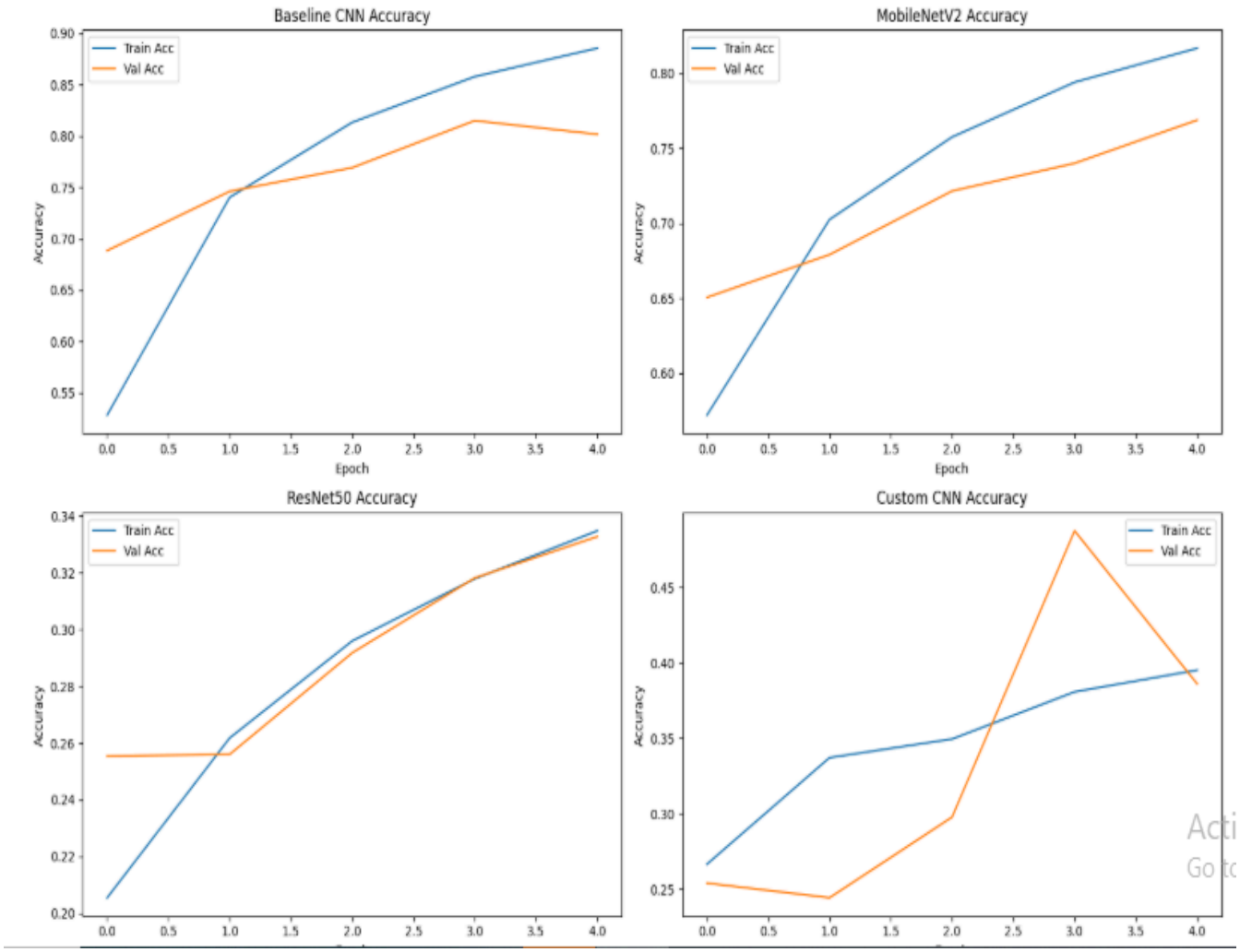
- **Baseline CNN** is our best-performing model *overall*
- **Custom CNN** , it might benefit from longer training or tuning.

This section is focused on **analyzing how each model learned over time** during training by visualizing their **accuracy and loss curves**. These plots help diagnose model behavior like underfitting, overfitting, or balanced learning.

Accuracy Curves

Each model has two lines:

- **Train Accuracy** (blue line)
- **Validation Accuracy** (orange line)



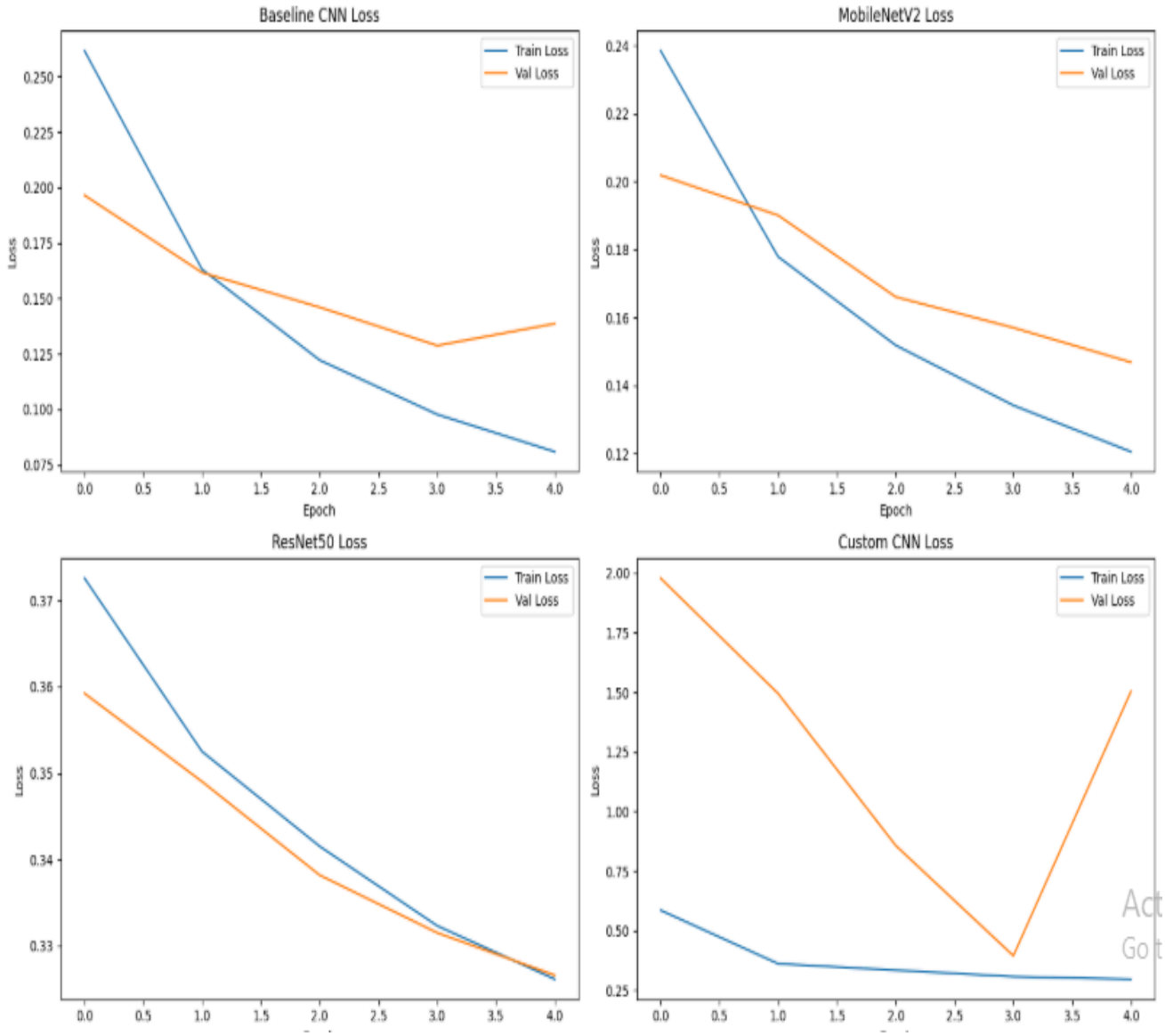
Interpretation:

- Accuracy increasing over time indicates the model is learning.
- If the validation accuracy is close to the training accuracy, it shows good generalization.
- If validation accuracy is much lower than training accuracy, it indicates overfitting.

Loss Curves

Each model has two lines:

- **Train Loss** (blue line)
- **Validation Loss** (orange line)



Interpretation:

- Shows how well each model minimized error (loss) over time.
- If both `train_loss` and `val_loss` decrease together, the model is learning well.
- If `train_loss` is low but `val_loss` is high or increasing, it suggests overfitting.

Model	Final Val Accuracy	Interpretation
Baseline CNN	82.4%	Very strong for a simple model
MobileNetV2	76.7%	Good performance, fast inference, but slightly below Baseline
ResNet50	33.5%	Likely underfitting or not well fine-tuned
Custom CNN (Reg)	35.7%	Struggling despite regularization

7.0 Conclusions and Recommendations

7.1 Conclusion

This project is a multi-label image classification pipeline for camera trap images. Four CNN-based models were trained and evaluated: a Baseline CNN, MobileNetV2, ResNet50, and a Custom CNN with regularization.

Conclusions:

- The baseline model achieved better Micro and Macro ROC-AUC compared to MobileNetV2 and ResNet50.
- ResNet50, benefited from complex feature extraction but may require fine-tuning for optimal performance.
- The Custom CNN gave more hope with incorporation with Batch Normalization and Dropout to mitigate overfitting.

7.2 Recommendations

The recommendations:

- **Class Imbalance:** Apply techniques such as class weighting, oversampling rare classes, or using focal loss to improve learning for minority species.
- **Label Quality Review:** Perform manual checks or automated noise reduction techniques to improve the quality of image labels.
- **Increase Training Duration:** Train models for more epochs with early stopping to allow better convergence.
- **Use Higher Resolution Images:**

7.3 Next Steps

To further improve performance and build on this foundation:

- **Fine-tune Pretrained Models:**
- **Hyperparameter Tuning:** Optimize learning rates and regularization parameters using grid search or randomized search.
- **Data Augmentation:** Apply stronger augmentations and variations.
- **Test-Time Augmentation:** Improve strength on the prediction by predicitng over multiple augmented versions of the same image.
- **Detect animals in images using other models like R-CNN**