BIRMINGHAM CITY University

"Deep Learning Algorithms to Identify and Detect Endangered Species of Animals"

Author: Shila Pradhan
Student ID: 24152372
Dataset - https://www.kaggle.com/datasets/brsdincer/danger-of-extinction-animal-image-set
Confidential: YES – DEPARTMENT ONLY
Submission date: 4th june, 2025

Contents

Tables of Figures:

Acknowledgement

I would like to express my deepest gratitude to Sunway College of Kathmandu and BCU for giving me this golden opportunity to present my report on the topic of “**Animal Extinction**” for the module “**Artificial and Machine Learning**”. The journey of completing this module came with its challenges, but thanks to the unwavering support of my teachers, I was able to overcome each obstacles with ease. Their guidance was invaluable, and I am sincerely grateful for their assistance.

Lastly, I would like to acknowledge the various online technologies like kaggle for the data set and google scholar to learn more deep about this topic and that facilitated my research and helped me successfully complete their coursework. Without these resources, this report would not have been possible.

Abstract

Many types of mammals such as the African Elephant, Chimpanzee, Lion, and others are vanishing due to habitat modification, illegal hunting, invasive species, emerging diseases, pollution, and climate change. Monitoring endangered species is essential for conservation actions to mitigate these threats. The IUCN Red List of Threatened Species provides a globally recognized framework for classifying species based on extinction risk, offering a standardized approach to identify and prioritize those in need of protection (IUCN Red List Categories and Criteria, 2001).

This project aims to classify endangered animal species using image data and deep learning techniques. The dataset, sourced from Kaggle, contains 6,484 labeled images across 11 animal classes, many of which are listed as endangered by the IUCN. Accurate identification of these species is crucial for ecological monitoring and wildlife conservation. To improve generalization, the images were preprocessed using resizing, normalization, and data augmentation.

We experimented with three pre-trained models: VGG16, ResNet50V2, and EfficientNetB0, each fine-tuned with optimized hyperparameters. VGG16 achieved 90.6% accuracy, ResNet50V2 reached 93.6%, and EfficientNetB0 attained 95.9% accuracy on the test set. These results demonstrate that transfer learning with pre-trained models outperforms traditional training approaches. A stratified k-fold cross-validation framework was employed to ensure robust evaluation, using accuracy and validation loss as key metrics. The findings confirm that deep learning especially with transfer learning is an effective tool for endangered species classification and a promising foundation for future AI-assisted conservation efforts.

Keywords: VGG16, EfficientNetB0, ResNet50V2, Transfer Learning, Endangered Species, IUCN Red List, Deep Learning Classification.

1. Introduction

1.1 Understanding Animal Extinction

Animal Extinction is a critical global issue characterized by the increasing number of animal species facing high risk of disappearing from the wild or becoming completely extinct. Once extinct, a species is lost for forever. Nearly 500 animal species have gone extinct in the past decade alone. Animal Extinction is a critical global issue, with unprecedented rates driven mainly by human activites. Scientists estimate that species loss ranges from 200 to 10,000 per year globally. There are a lot of reasons why animals are extinction due to Habitat destruction, Overexploitation, Pollution, Climate change and many more (*IUCN red list categories and Criteria* 2001).

The loss of one often leads to co-extinctions, where other species that rely on it also vanish, and disrupting entire ecosystem. According to the IUCN Red List (2001), thousands of species are classified as Vulnerable, Endangered, or Critically Endangered (RODRIGUES et al., 2006). This highlights the urgent need for conservation efforts and robust monitoring tools to prevent further biodiversity loss.

1.2 IUCN Red List and Its categories

The IUCN Red List is the most comprehensive resource for global conservation status of species. Its main goal is to provide a consistent and scientifically grounded way to classify species into categories based on how threatened they are. The Red List is widely regarded as the most credible and influential system for assessing species conservation status due to its clear, measurable criteria related to population decline, geographic range, and threats (*The history of emotions: An introduction* 2015). These include animals that are Vulnerable, Endangered, or Critically Endangered. Species are classified into categories such as (*IUCN red list categories and Criteria* 2001):

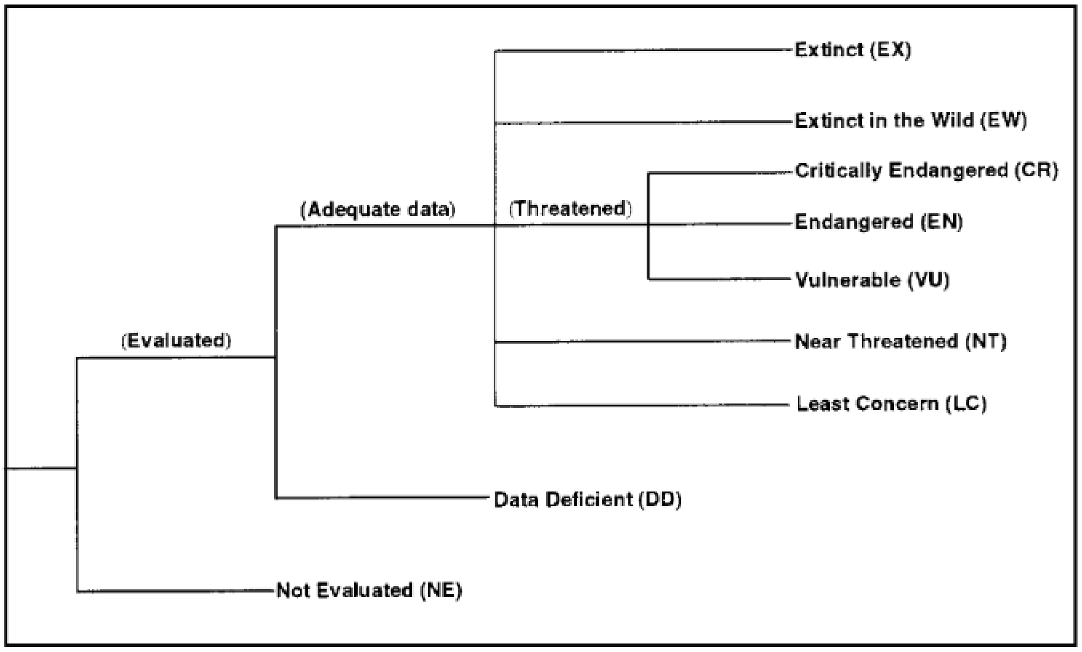


Figure 1. Structure of the categories.

Figure 1 Structure of categories

1.3 Problem Statement

Global extinction rates are accelerating, driven by habitat destruction, introduced species, and human activity. Traditional way of conservation method is insufficient and actually not effective (*Science and the endangered species act* 1995). This creates uncertainty in extinction status designations and risks both over and underestimation of true extinction rates (Fisher & Humphreys, *Evidence for modern extinction in plants and animals* 2024). Climate change, logging and urban expansion, is accelerating at an alarming rate, particularly in tropical and biodiversity-rich regions continue to intensify this crisis, resulting in the loss of hundreds of animal species (Tarkik-Gautam-Ranjan, 2022). There is a critical need for integrating data-driven tools including machine learning and statistical model to improve the objectivity, transparency, and frequency of extinction assessments.

1.4 Project Objectives

The primary objective of this project is to develop a deep learning-based image classification system that can automatically identify endangered animal species and assess their extinction risk based on IUCN Red List categories. The system aims to support wildlife conservation by leveraging computer vision.

The specific goals are:

1. To curate and preprocess a labeled image dataset of 11 endangered animal species, including species like the Amur Leopard and African Elephant.
2. To train and evaluate deep learning models **VGG16**, **ResNet50V2**, and **EfficientNetB0** — using image augmentation, normalization, and **stratified k-fold cross-validation** for robust evaluation.
3. To implement a **stacked ensemble model** using the outputs of the base models and a meta-classifier to improve overall classification accuracy.
4. To apply **Grad-CAM visualization** to explain and interpret model predictions for selected test images.
5. To map the predicted species to their respective **IUCN Red List categories** and visualize the predicted conservation status across test data samples.

2. Literature Review

2.1 Overview of Extinction Monitoring Techniques

Traditional field surveys and local sightings have long been the backbone of monitoring threatened animal species (Tuia et al., 2022). However, these human-based methods are often slow, costly, and can produce incomplete or biased data due to limited coverage and observer error. The IUCN Red List assesses extinction risk using such data, for eg: criteria include population size, the rate of decline, and the species geographic range (*IUCN red list* 2025). In recent years, newer techniques have emerged to support these efforts such as, camera traps that automatically photograph wildlife, and AI-based image recognition can identify the species in those images. These modern methods complement on-the-ground surveys by covering larger areas, gathering data faster, and often detecting elusive species making the monitoring of extinction risk more efficient and effective (Libretexts, 2022).

2.2 Deep Learning in Wildlife Conservation

Deep learning is increasingly used in wildlife conservation to automatically identify endangered animals from images, making it easier to monitor threatened species (Subek Sharma Department of Electronics and Computer Engineering et al.). Convolutional Neural Networks(CNNs) including modern architectures like ResNet, VGG and EfficientNet have been applied to classify animal species in photo with high accuracy (Supritha R, 2024). This AI driven approach greatly reduces the manual efforts, save time by analyzing of large dataset, and error in sorting camera trap images. In practice, such models enable conservationists to monitor wildlife populations over time and even detect poaching activities in real time, providing data to guide conservation planning (Ijibadejo Oluwasegun William, 2021). By linking each identified animal to its IUCN Red List category, deep learning tools help focus attention on the species most at risk, aiding more effective protection efforts.

2.3 Use of Models

We explored three pre-trained convolutional neural network (CNN) models in this project. Each model brings a different balance of complexity, parameter count, and prior training, and here we explain each and why it was chosen for the task.

2.3.1 ResNet50V2

2.3.2 EfficientNetB0

2.3.3 VGG16

3. Dataset Description

3.1 Dataset Overview

The dataset used in this project, "Danger of Extinction - Animal Image Set", is sourced from kaggle which is based on the IUCN Red List of Threatened species. It is compiled to support machine learning application in wildlife conservation which includes 6,484 images across 11 species, with each image labeled by species for classification tasks.

The 11 classes are:

1. Panda
2. Lion
3. Rhino
4. Jaguars
5. Panthers
6. Amur Leopard
7. Orangutan
8. African
9. Elephant
10. Chimpanzee
11. Arctic Fox

Preprocessing involved resizing all images to 224x224 pixels and converting them to RGB format to ensure consistency, as no missing values were present, though image quality varies and almost 260 duplicate images. This dataset aligns with the projects goal of classifying endangered species using deep learning, using diverse set of images for model training.

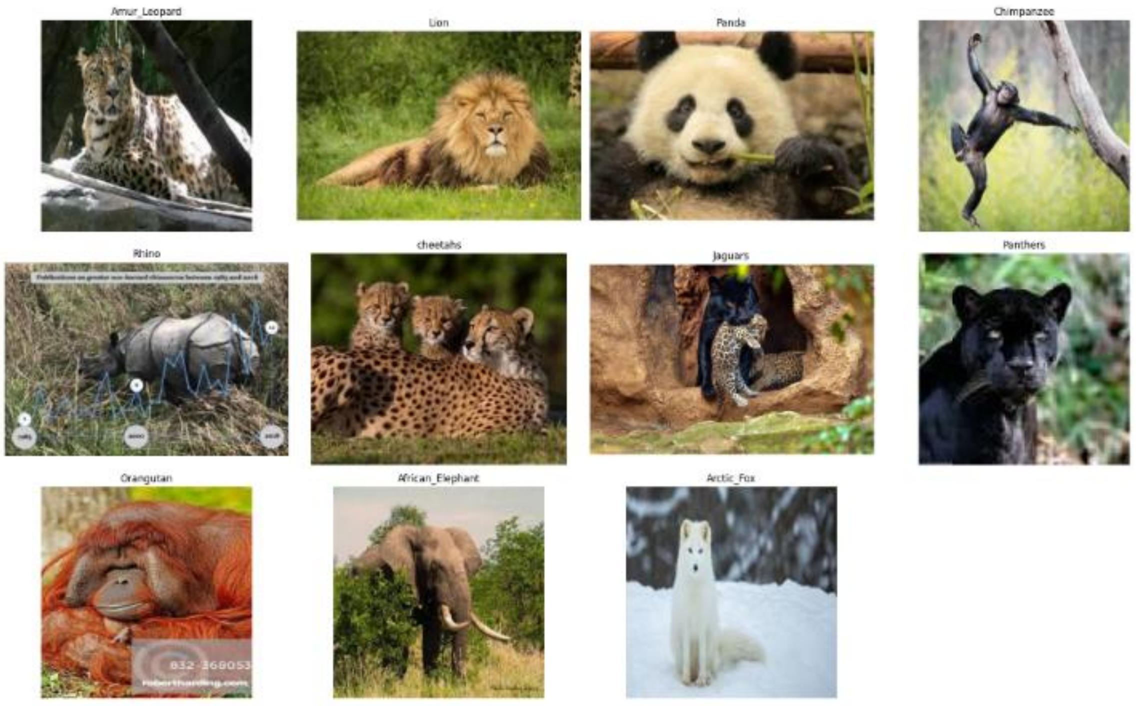


Figure 2 Images of animal Categories

However, limitations such as class imbalance and inconsistent image quality may effect performance and will be addressed in the methodology. Overall, the dataset is well-suited for endangered species classification and supports the integration of IUCN Red List categories in later stages of the project.

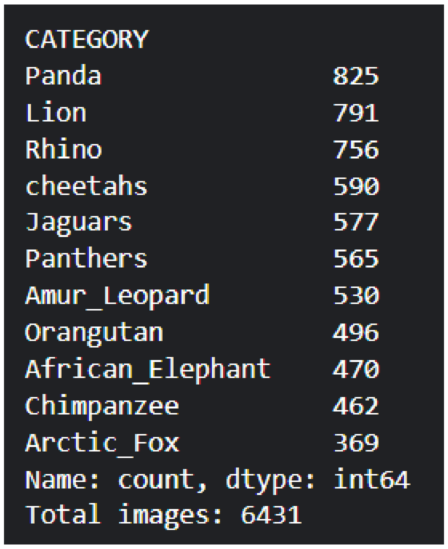
3.2 Identification of Supervised Learning Task

This project addresses a multi-class classification task within the supervised learning framework. The model outputs are probability distribution over these 11 classes, with the highest probability indicating the predicted species. This task aligns with the projects goal of developing an automated system to identify endangered species, supporting conservation efforts by enabling efficient tracking of at-risk population, as informed by the IUCN Red List. Each image is labeled with one of the 11 species categories. Supervised learning is appropriate here due to the availability of labeled data, which enables the model to learn from input-output pairs.

4. Exploratory Data Analysis (EDA)

Exploratory Data Analysis (EDA) is the process of examining and visualizing the dataset to understand its structure, identify patterns, and gain insights that guide further preprocessing and model development. In this project, EDA was used to explore the distribution of animal species classes, check for class imbalance, inspect image quality and resolution and ensure the integrity of labels.

4.1 Class imbalance Visualization



```
CATEGORY
Panda               825
Lion                791
Rhino               756
cheetahs            590
Jaguars             577
Panthers            565
Amur_Leopard        530
Orangutan           496
African_Elephant    470
Chimpanzee          462
Arctic_Fox          369
Name: count, dtype: int64
Total images: 6431
```

Figure 3 Categories and Total Images

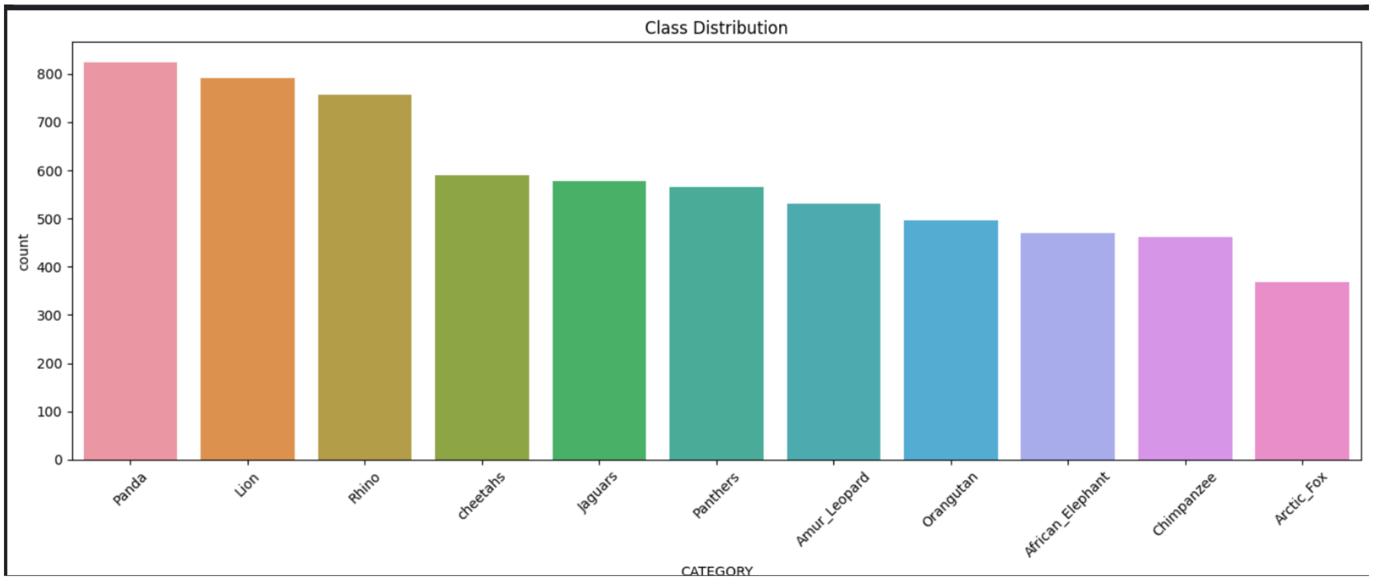


Figure 4 Class distribution

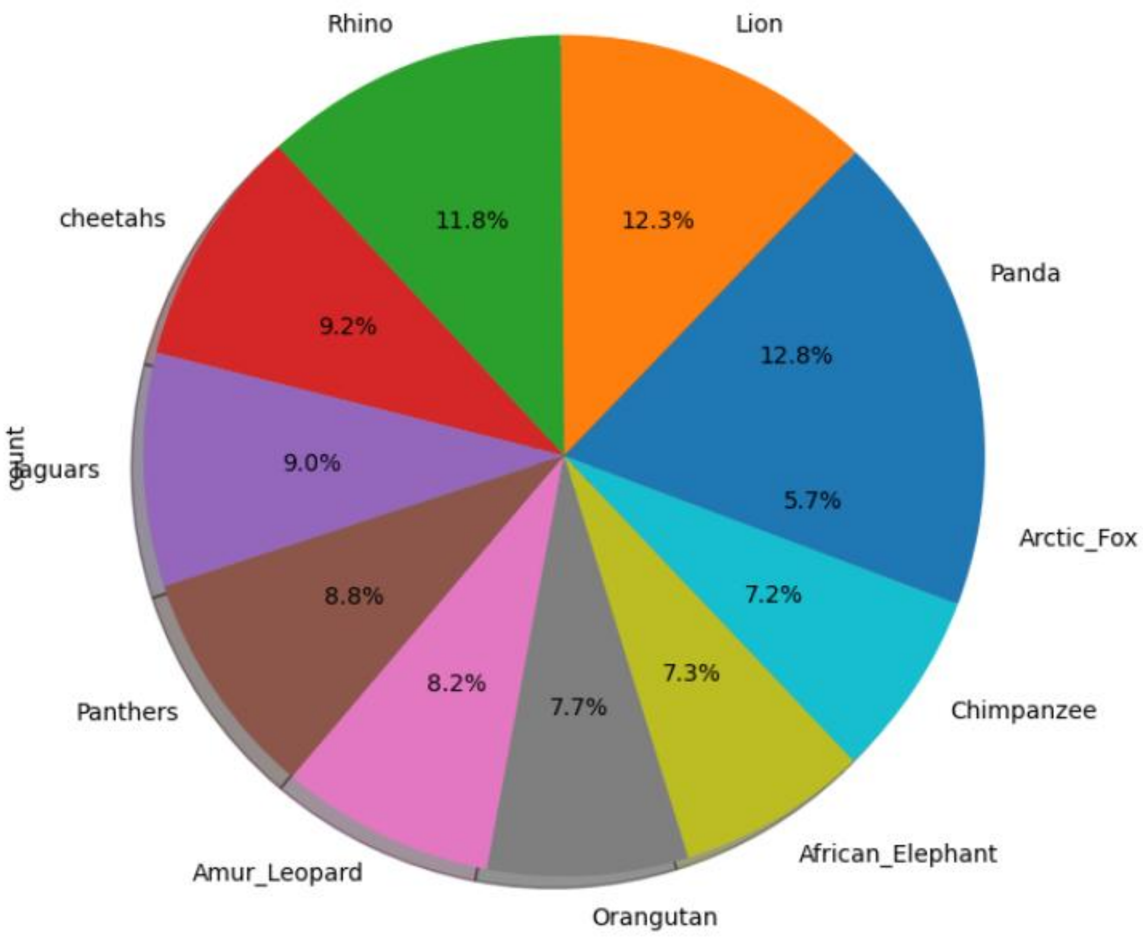


Figure 5 Class Distribution Rate

The bar chart and pie chart above illustrate the distribution of image sample across the 11 animal species. It is evident that the dataset is imbalanced, with classes like Panda, Lion and Rhino have more images compared to Arctic Fox, Chimpanzee and African Elephant. This imbalance highlights the need for strategies like data augmentation or class weighting during training to ensure fair model performance for all classes.

4.2 Image Dimension and Mode Analysis

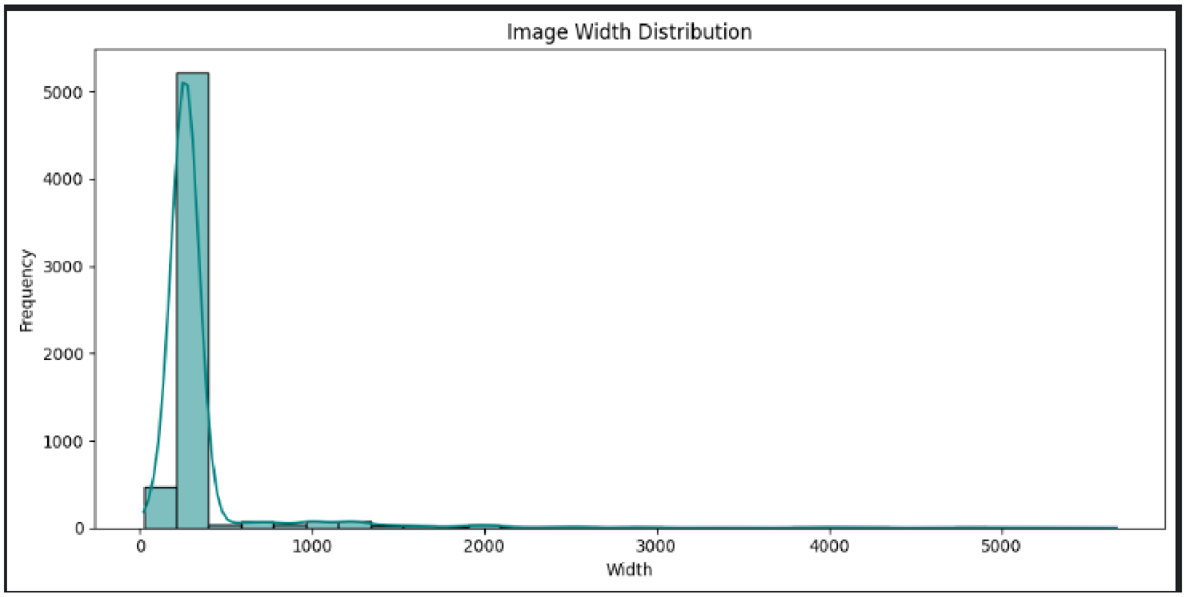


Figure 6 Image Width Distribution

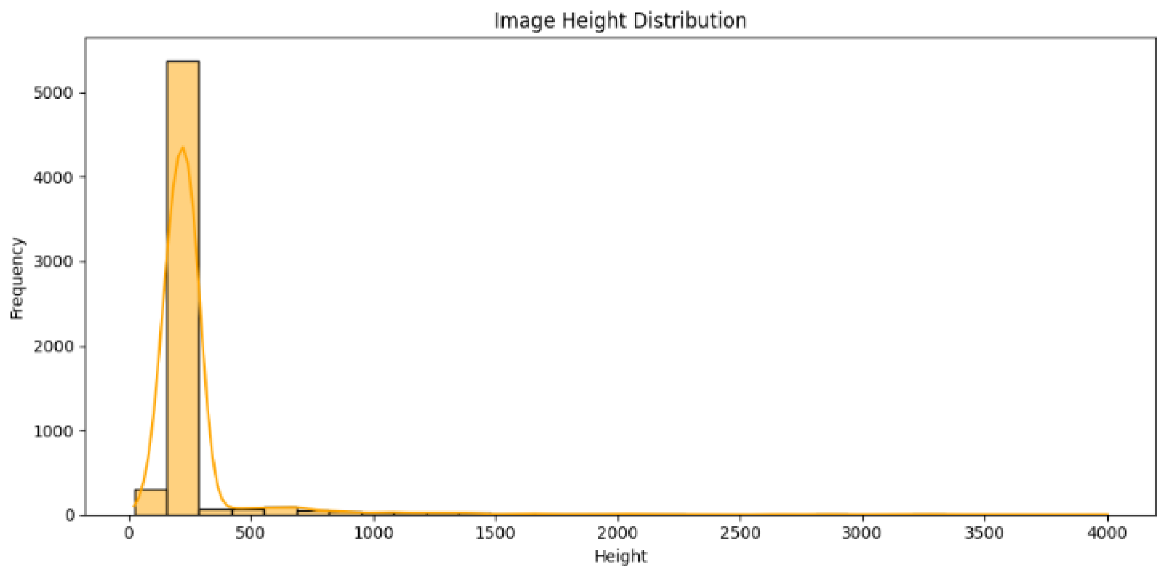


Figure 7 Image Height Distribution

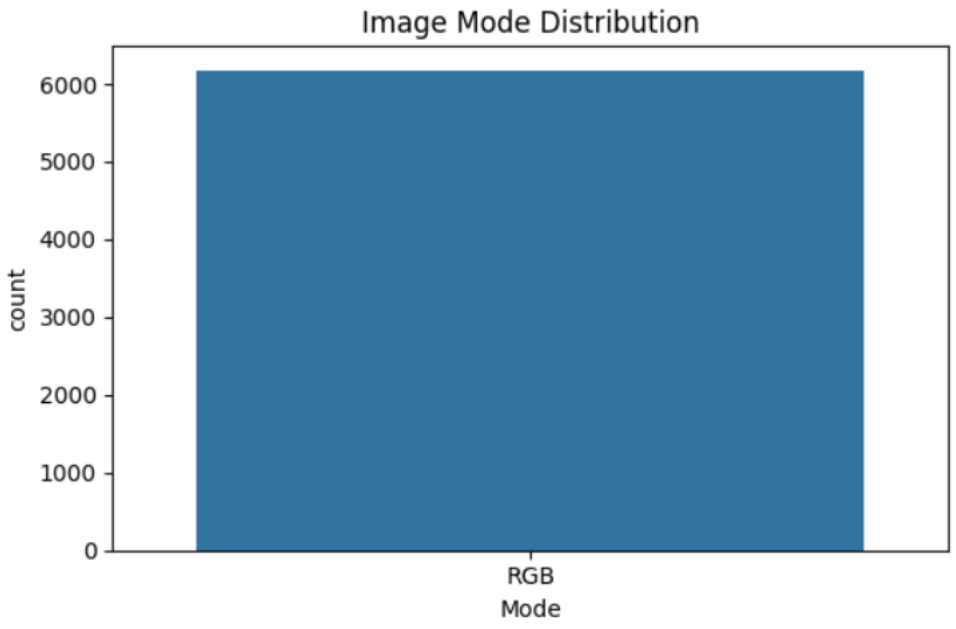


Figure 8 Image Mode Distribution

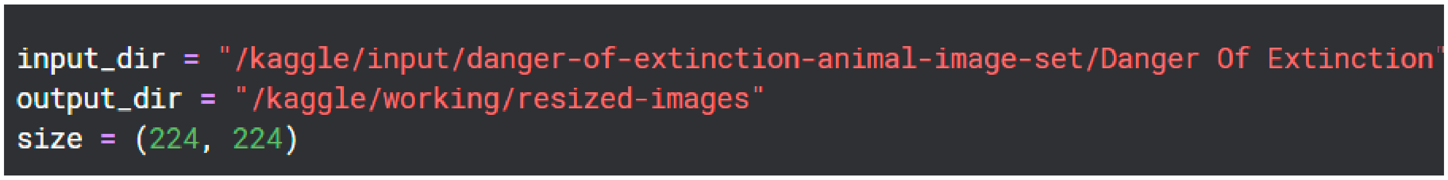
The histograms of image width and height reveal that most images have dimensions clustered between 200 and 500 pixels, though a few outliers exist with much larger sizes. To standardize input for model training all images were resized to a fixed resolution. The mode distribution confirms that all images are in RGB format, meaning they contain three color channels.

5. Data Preprocessing

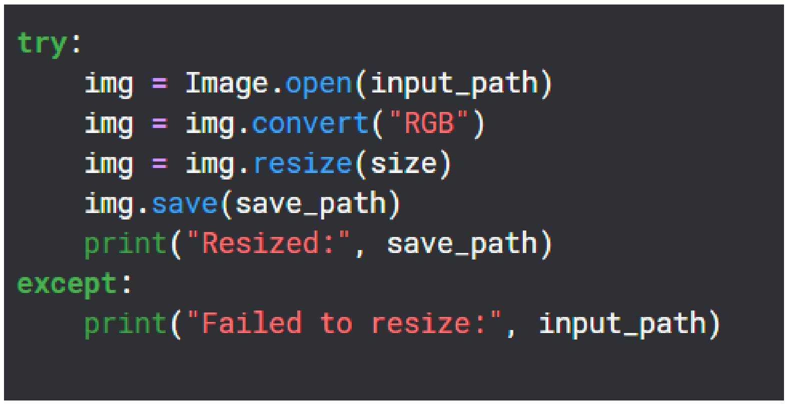
In this section, we outline steps taken to prepare the raw dataset of endangered animal species images for model training. The preprocessing ensures consistency, improve model convergence and addresses dataset-specific issues.

5.1 Image Resizing

All images were resized to a uniform dimension of 224x224 pixel. This step ensures that the input data is consistent across the dataset, as the original images varied in size and resolution, which could otherwise affect model performance.



```
input_dir = "/kaggle/input/danger-of-extinction-animal-image-set/Danger Of Extinction"
output_dir = "/kaggle/working/resized-images"
size = (224, 224)
```



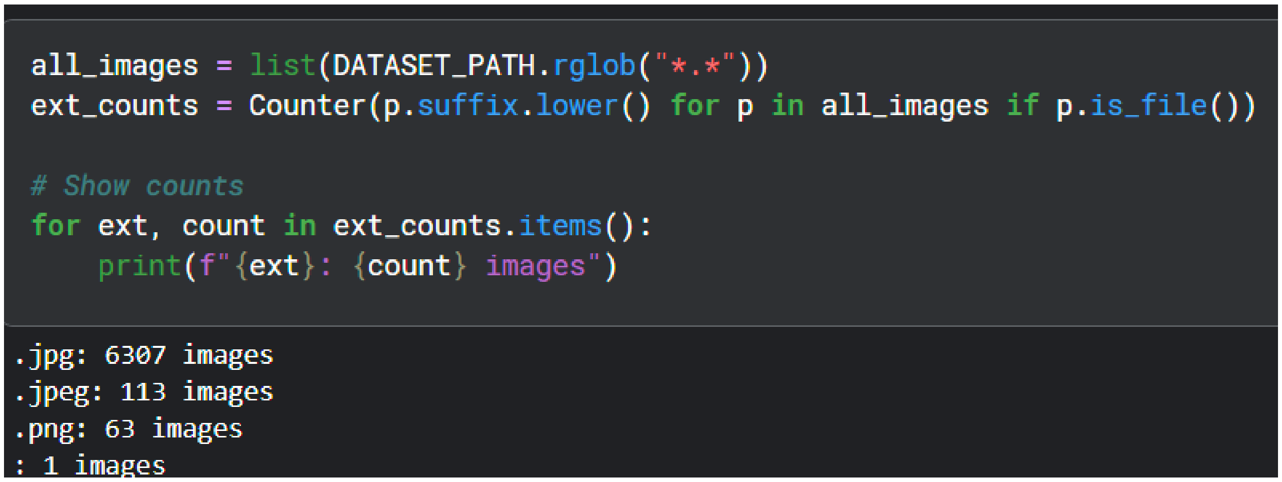
```python
try:
    img = Image.open(input_path)
    img = img.convert("RGB")
    img = img.resize(size)
    img.save(save_path)
    print("Resized:", save_path)
except:
    print("Failed to resize:", input_path)
```

```
Image size: (224, 224)
```

Figure 9 Sample of Resized Image

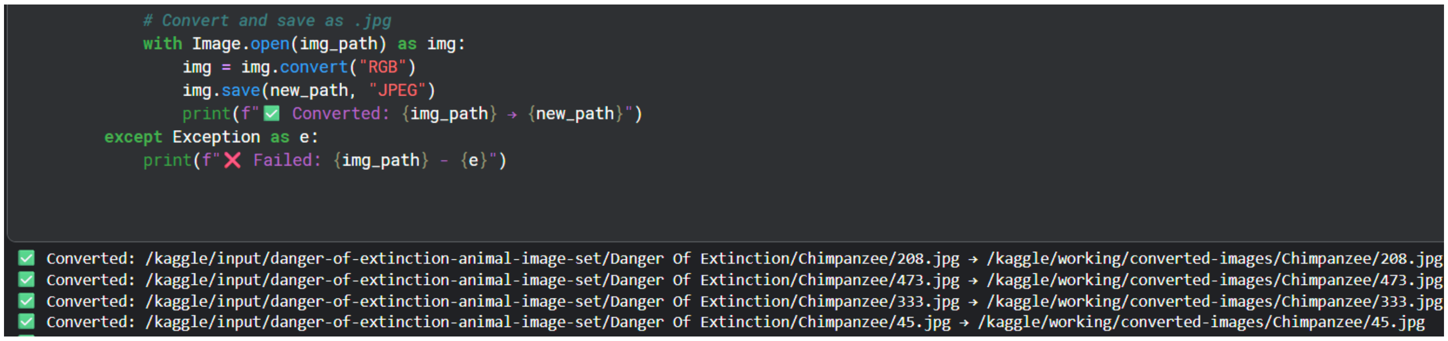
5.2 Converting Image into JPG

The original dataset contained images in multiple formats including .jpg, .jpeg, .png. To ensure uniformity and simplify preprocessing, all images were converted to jpg. This standardization is essential for consistent loading and compatibility with image preprocessing pipelines in deep learning.



```
all_images = list(DATASET_PATH.rglob("*.*"))
ext_counts = Counter(p.suffix.lower() for p in all_images if p.is_file())

# Show counts
for ext, count in ext_counts.items():
    print(f"{ext}: {count} images")
```

```
.jpg: 6307 images
.jpeg: 113 images
.png: 63 images
: 1 images
```

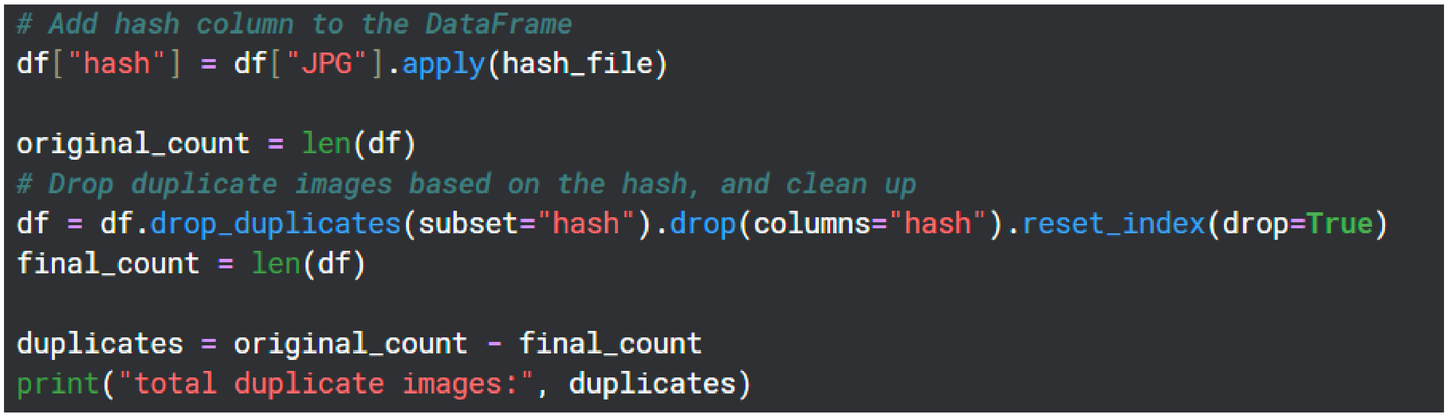


```
            # Convert and save as .jpg
            with Image.open(img_path) as img:
                img = img.convert("RGB")
                img.save(new_path, "JPEG")
                print(f"✅ Converted: {img_path} → {new_path}")
        except Exception as e:
            print(f"❌ Failed: {img_path} - {e}")
```

```
✅ Converted: /kaggle/input/danger-of-extinction-animal-image-set/Danger Of Extinction/Chimpanzee/208.jpg → /kaggle/working/converted-images/Chimpanzee/208.jpg
✅ Converted: /kaggle/input/danger-of-extinction-animal-image-set/Danger Of Extinction/Chimpanzee/473.jpg → /kaggle/working/converted-images/Chimpanzee/473.jpg
✅ Converted: /kaggle/input/danger-of-extinction-animal-image-set/Danger Of Extinction/Chimpanzee/333.jpg → /kaggle/working/converted-images/Chimpanzee/333.jpg
✅ Converted: /kaggle/input/danger-of-extinction-animal-image-set/Danger Of Extinction/Chimpanzee/45.jpg → /kaggle/working/converted-images/Chimpanzee/45.jpg
```

5.3 Removing Duplicate Images

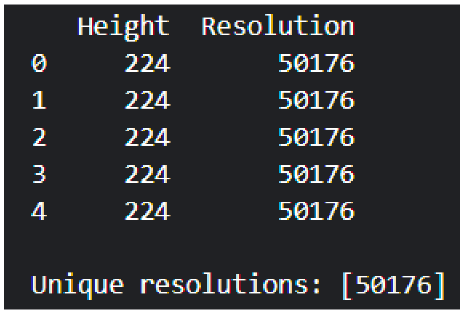
To improve dataset quality and avoid training bias, duplicate images were identified and removed. A hashing function was applied to each image to generate a unique fingerprint. This process reduced the total number of images from original_count to final_count and effectively removing X duplicate images. More than 250 images were delated.



```
# Add hash column to the DataFrame
df["hash"] = df["JPG"].apply(hash_file)

original_count = len(df)
# Drop duplicate images based on the hash, and clean up
df = df.drop_duplicates(subset="hash").drop(columns="hash").reset_index(drop=True)
final_count = len(df)

duplicates = original_count - final_count
print("total duplicate images:", duplicates)
```

5.4 Resolution Standardization

To ensure consistency across the dataset and compatibility with deep learning model input requirements, all images were resized to a resolution of 224x224 pixels. After resizing, the entire dataset had a single unique resolution of 50,176 pixels, confirming successful normalization of image dimensions.



```
   Height  Resolution
0     224       50176
1     224       50176
2     224       50176
3     224       50176
4     224       50176

Unique resolutions: [50176]
```

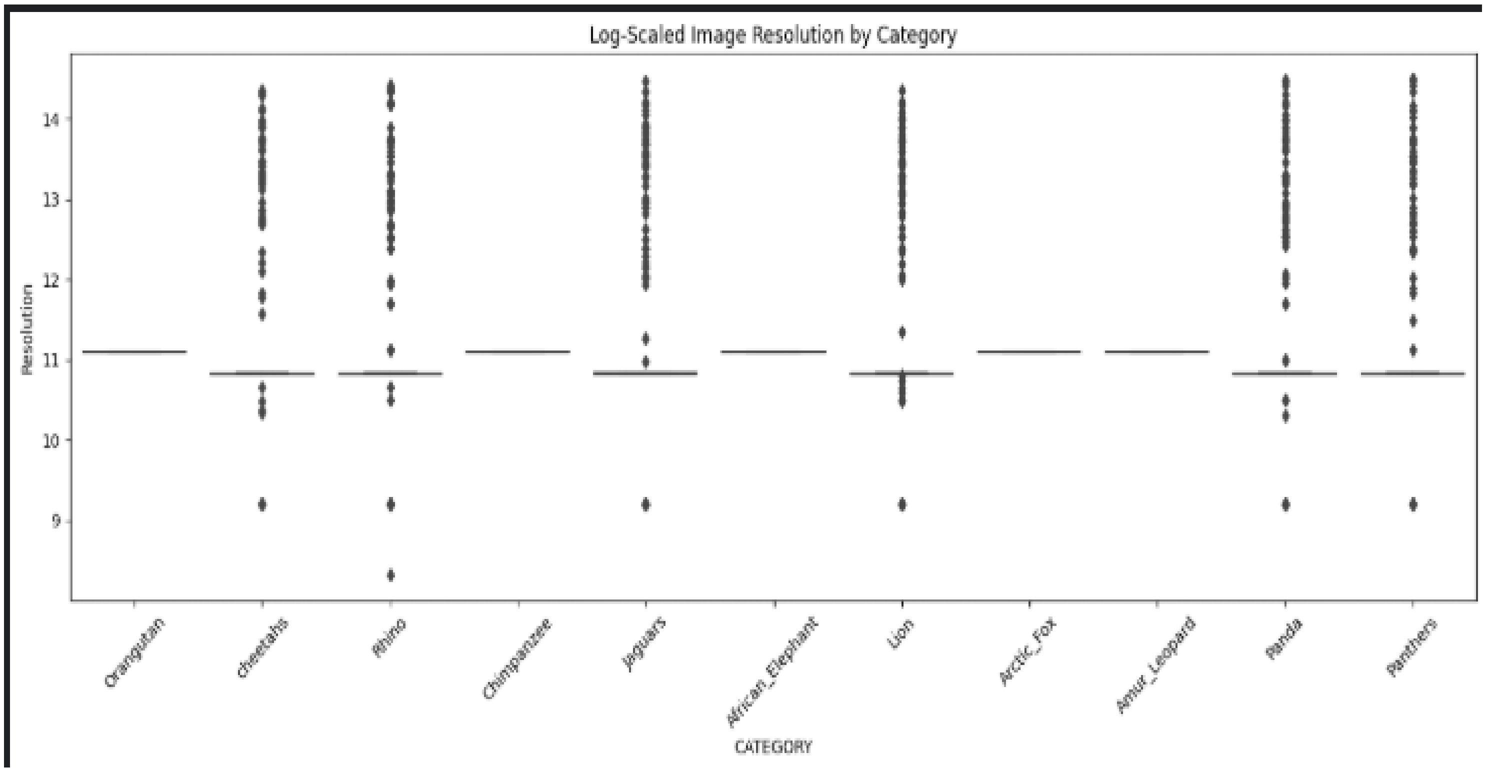


Figure 10 Image Resolution by Category before resized

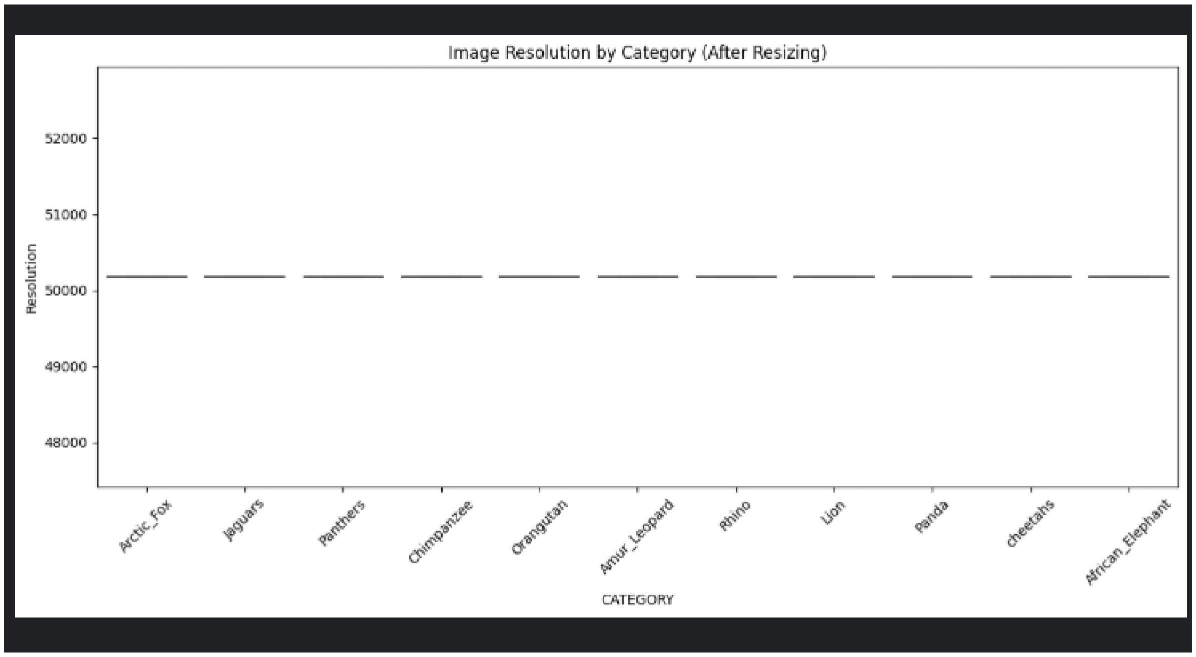


Figure 11 Image resolution by Category (after Resizing)

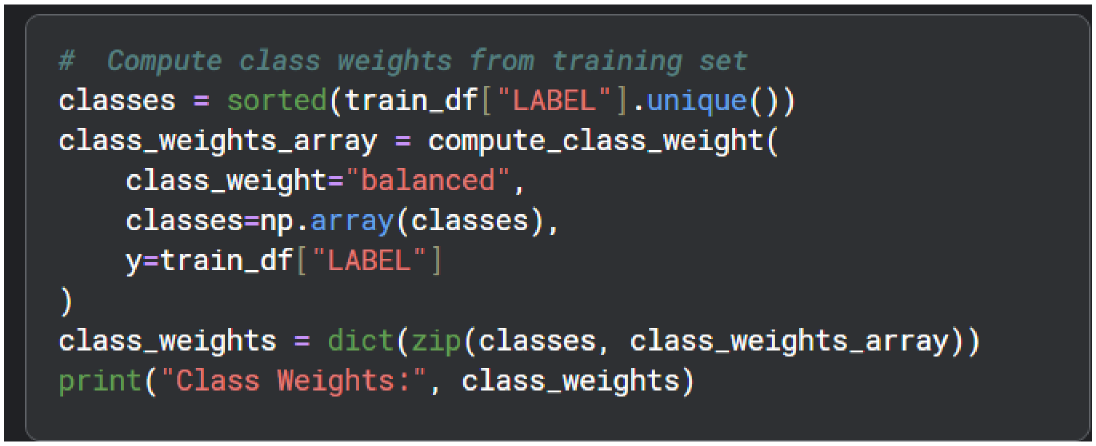
5.5 Class Weights and Data Augmentation

To address class imbalance in the dataset, two techniques were applied during model training.

5.5.1 Class weights:

This method calculates weights inversely proportional to class frequencies, ensuring that underrepresented classes contribute more significantly to model's loss function. The weights were passed during training to encourage the model to treat all classes more equally.

```
Class Weights: {0: 1.2212349054454317, 1: 1.082828282828283, 2: 1.551823972
2061378, 3: 1.239879713162156, 4: 0.9985096870342772, 5: 0.760175861579917
8, 6: 1.1574174044482832, 7: 0.7251082251082251, 8: 1.0215361158757386, 9:
0.804080408040804, 10: 0.9764984514483512}
```



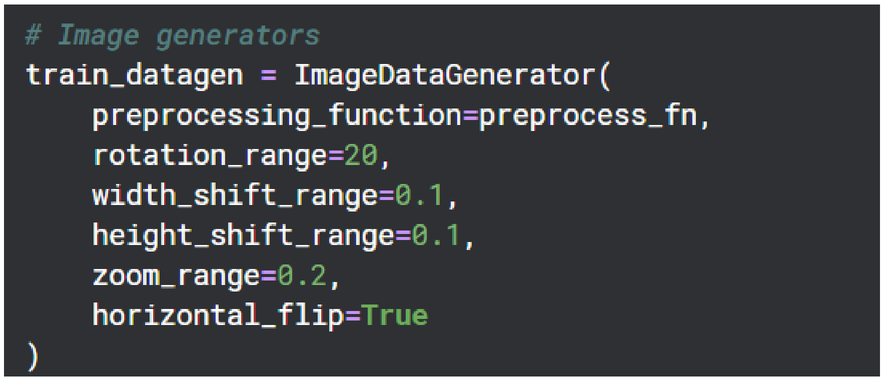
```python
#  Compute class weights from training set
classes = sorted(train_df["LABEL"].unique())
class_weights_array = compute_class_weight(
    class_weight="balanced",
    classes=np.array(classes),
    y=train_df["LABEL"]
)
class_weights = dict(zip(classes, class_weights_array))
print("Class Weights:", class_weights)
```

5.5.2 Data Augmentation

Data Augmentation was implemented using Kera's ImageDataGenerator to artificially expand the training dataset and reduce overfitting. The following augmentations were applied:

- Rotation: upto ±20 degrees
- Width and height shift: Up to 10% of the image dimensions
- Zoom: Up to 20% in/out
- Horizontal flipping: Random horizontal flip

These transformations were applied in real-time during training.

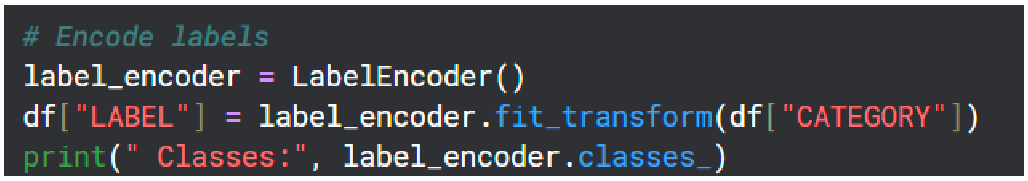


```
# Image generators
train_datagen = ImageDataGenerator(
    preprocessing_function=preprocess_fn,
    rotation_range=20,
    width_shift_range=0.1,
    height_shift_range=0.1,
    zoom_range=0.2,
    horizontal_flip=True
)
```

5.6 Label Encoding

To enable the model to process categorical labels, each animal class name was converted into a numerical label using label encoding. This transformation maps each of the 11 animal species to a unique integer value ranging from 0 to 10. The label encoded are-:

1. Panda – 0
2. Lion – 1
3. Rhino – 2
4. Cheetahs – 3
5. Jaguars – 4
6. Panthers - 5
7. Amur_leopard – 6
8. Orangutan – 7
9. African_Elephant – 8
10. Chimpanzee – 9
11. Arctic_Fox - 10



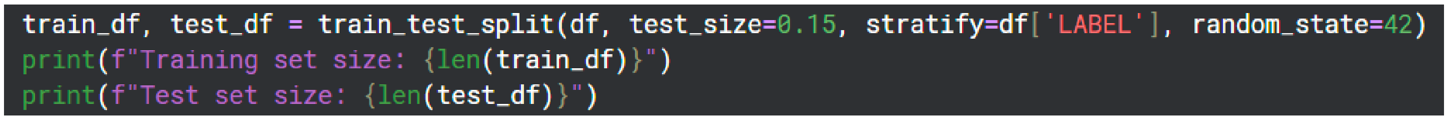
```
# Encode labels
label_encoder = LabelEncoder()
df["LABEL"] = label_encoder.fit_transform(df["CATEGORY"])
print(" Classes:", label_encoder.classes_)
```

The numeric representation is required for model training, especially in supervised classification task.

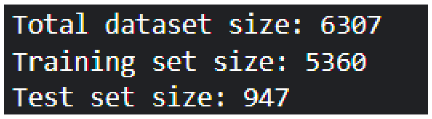
5.7 Dataset Splitting

To evaluate model performance effectively, the dataset was initially split into two parts using stratified sampling:

- Training set: 85% of the data
- Test set: 15% of the data



```
train_df, test_df = train_test_split(df, test_size=0.15, stratify=df['LABEL'], random_state=42)
print(f"Training set size: {len(train_df)}")
print(f"Test set size: {len(test_df)}")
```



```
Total dataset size: 6307
Training set size: 5360
Test set size: 947
```

The split was performed using the train_test_split() function from sklearn.model_selection with stratify set to the label column and random_state = 42 to ensure reproducibility.

6. Model Architecture and Implementation

In this project, three deep learning models were selected for training and evaluation VGG16, ResNet50V2 and EfficientNetB0. These models were chosen based on their proven performance in image classification task. These models are widely used for image classification and were chosen due to their diverse architectural designs and proven performance on large-scale image datasets like ImageNet. Leveraging transfer learning, each model was fine-tuned on the endangered animal species dataset to classify images into 11 categories aligned with the IUCN Red list.

6.1 VGG16 Implementation

VGG16 is a deep convolutional neural network consisting of 13 convolutional layers followed by 3 fully connected layers. Its simple and uniform architecture (using 3×3 filters) makes it an excellent baseline for image classification tasks. In this project:

- Pre-trained ImageNet weights were used.
- The top layers were replaced with a custom classifier (Dense, Dropout, Softmax).
- The convolutional base was frozen during initial training and later unfrozen for fine-tuning.
- After hyperparameter tuning, the entire model was fine-tuned with a reduced learning rate to adapt to the specific task.

```
def build_vgg(input_shape, num_classes):
    input_layer = Input(shape=input_shape)
    x = vgg16.preprocess_input(input_layer)
    base = VGG16(include_top=False, weights="imagenet", input_tensor=x)
    for layer in base.layers[:10]:
        layer.trainable = False
    x = GlobalAveragePooling2D()(base.output)
    x = Flatten()(x)
    x = Dense(128, activation="relu")(x)
    x = Dropout(0.3)(x)
    x = Dense(64, activation="relu")(x)
    x = Dropout(0.3)(x)
    output = Dense(num_classes, activation="softmax")(x)
    return Model(inputs=input_layer, outputs=output, name="VGG16")
```

```
# VGG16
acc_vgg, hist_vgg = train_model_with_kfold(
    VGG16,
    train_df,
    save_path="/kaggle/working/vgg16"
)
```

6.2 ResNet50V2 Implementation

ResNet50V2 introduces **residual connections**, allowing the network to train deeper architectures without degradation. This model was chosen for its ability to extract deep hierarchical features while maintaining stability during training. In this project:

- The base model was initialized with ImageNet weights.
- Custom top layers were added for classification.
- A few final residual blocks were unfrozen during fine-tuning.

```
# === Model Builders ===
def build_resnet(input_shape, num_classes):
    input_layer = Input(shape=input_shape)
    x = resnet_v2.preprocess_input(input_layer)
    base = ResNet50V2(include_top=False, weights="imagenet", input_tensor=x)
    base.trainable = False
    x = GlobalAveragePooling2D()(base.output)
    x = Dropout(0.3)(x)
    output = Dense(num_classes, activation="softmax")(x)
    return Model(inputs=input_layer, outputs=output, name="ResNet50V2")
```

```
# EfficientNetB0
acc_effnet, hist_effnet = train_model_with_kfold(
    EfficientNetB0,
    train_df,
    epochs=10,
    save_path="/kaggle/working/efficientnet"
)
```

6.3 EfficientNetB0 Implementation

EfficientNetB0 uses a compound scaling method to efficiently balance network depth, width, and resolution. It was selected for its optimal trade-off between speed and accuracy. In this project:

- ImageNet weights were used as a base.
- Top layers were replaced with a Dropout + Dense classifier.
- The final few layers were unfrozen for additional fine-tuning.

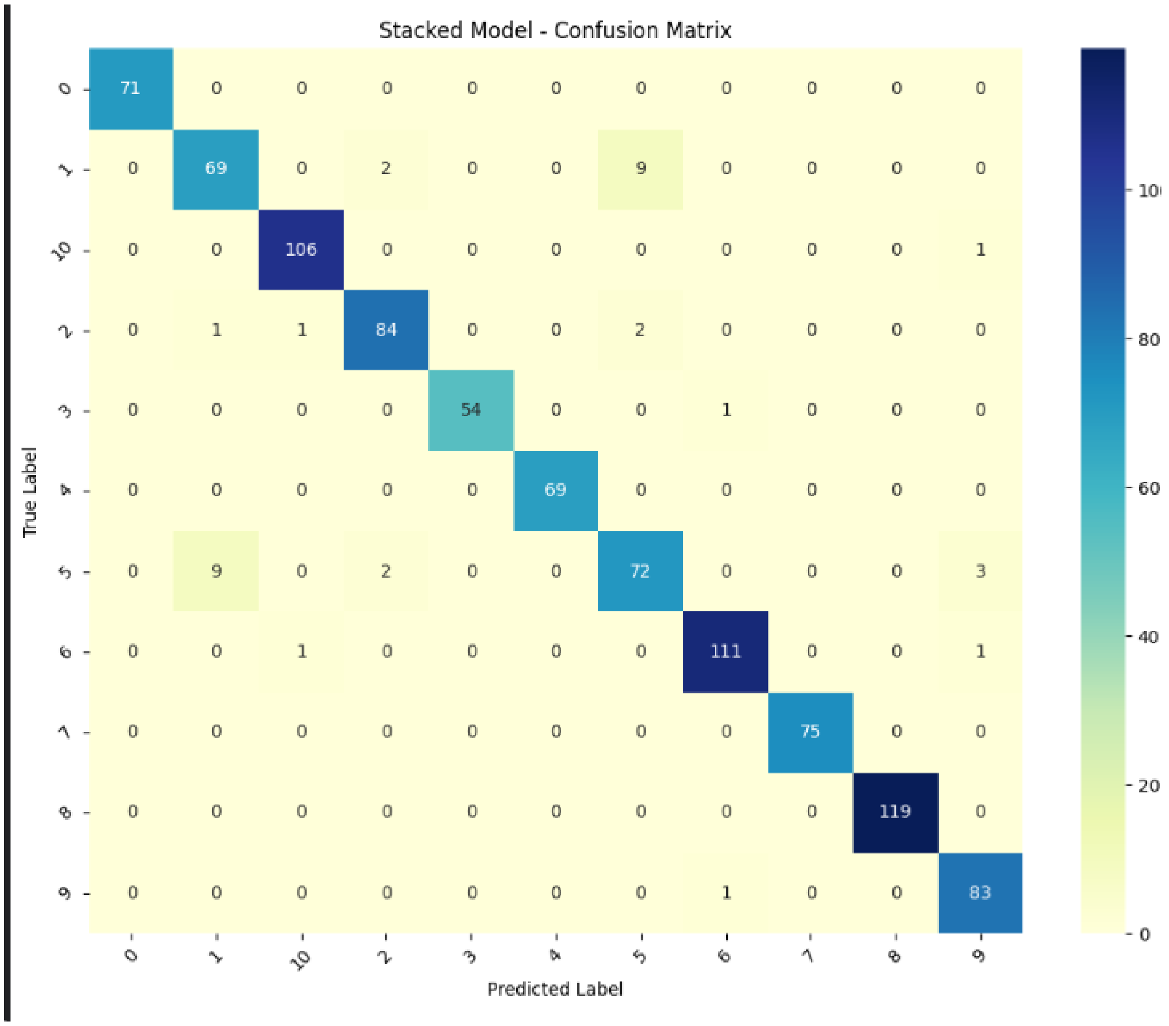
```
def build_effnet(input_shape, num_classes):
    input_layer = Input(shape=input_shape)
    x = efficientnet.preprocess_input(input_layer)
    base = EfficientNetB0(include_top=False, weights="imagenet", input_tensor=x)
    base.trainable = False
    x = GlobalAveragePooling2D()(base.output)
    x = Dropout(0.3)(x)
    output = Dense(num_classes, activation="softmax")(x)
    return Model(inputs=input_layer, outputs=output, name="EfficientNetB0")
```

```
# ResNet50V2
acc_resnet, hist_resnet = train_model_with_kfold(
    ResNet50V2,
    train_df,
    epochs=10,
    save_path="/kaggle/working/resnet"
)
```

6.4 Stacking Ensemble Approach (Logistic Regression as meta learner)

To improve predictive performance, a **stacking ensemble** technique used. Predictions from the three base models (ResNet50V2, EfficientNetB0, and VGG16) were collected and each of these model produced probability prediction for each image.

```
Getting predictions from VGG16
Found 947 validated image filenames belonging to 11 classes.
Getting predictions from EfficientNetB0
Found 947 validated image filenames belonging to 11 classes.
Getting predictions from ResNet50V2
Found 947 validated image filenames belonging to 11 classes.

 Stacked Model Accuracy: 0.9641
```

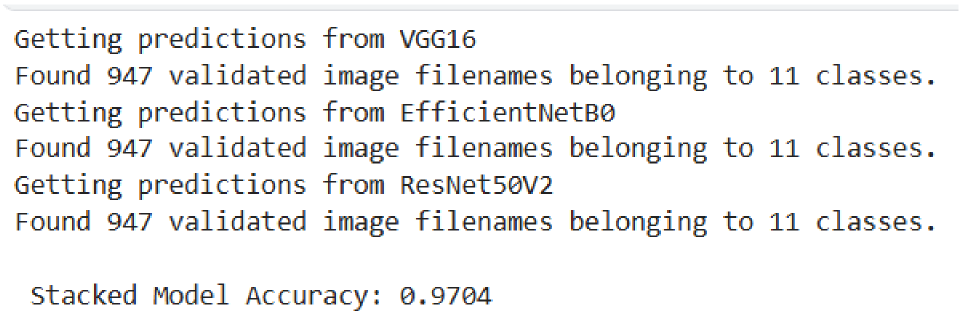


Meta model construction

The outputs from all three base model after retraining models were concatenated to form a new feature set. This formed the input X_meta which was trained to get best combine base model predictions. A logistic regression model was chosen as the meta-classifier due to its simplicity and effectiveness for classification task.

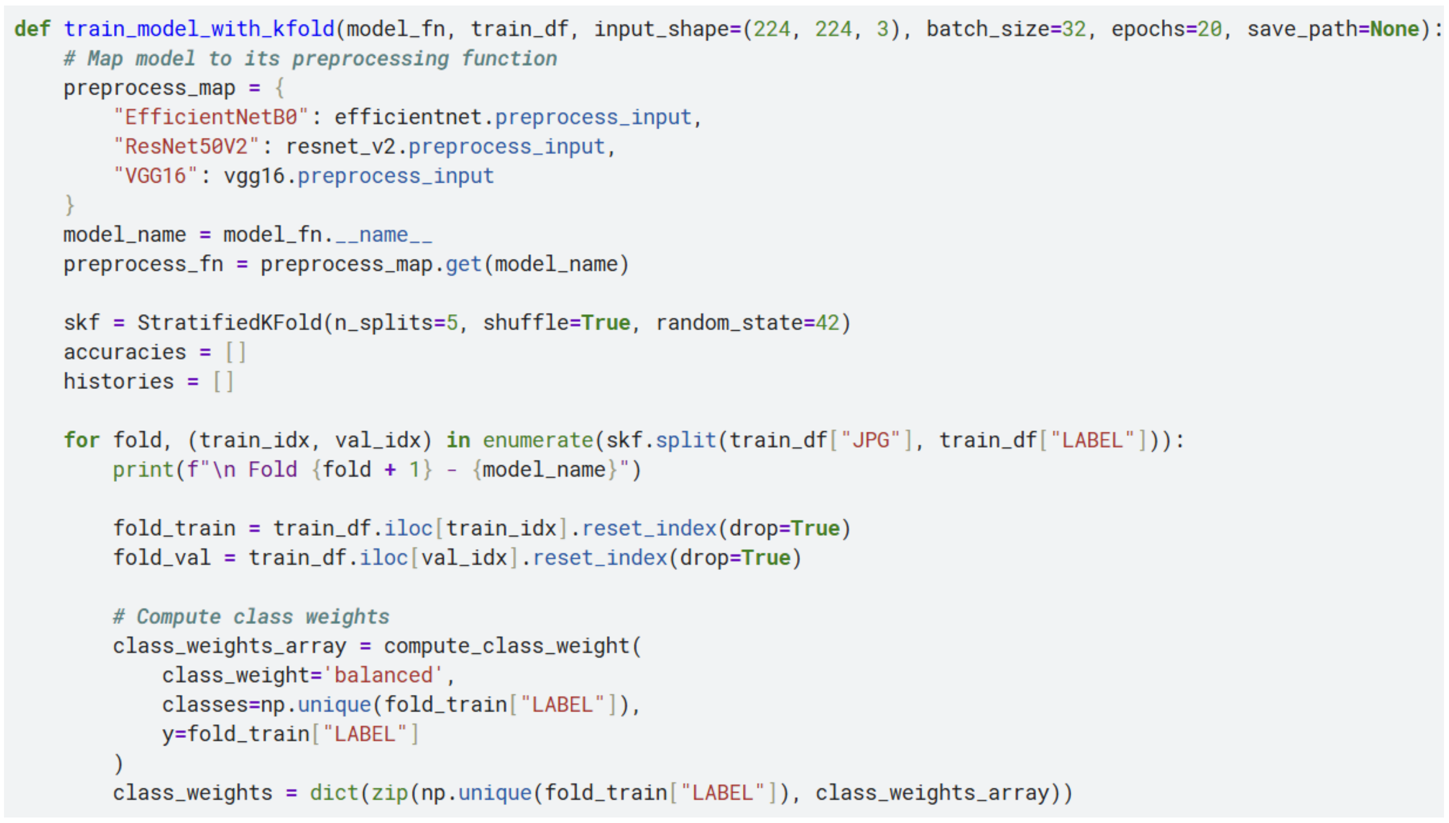
```python
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import accuracy_score

test_df["LABEL"] = test_df["LABEL"].astype(str)
label_encoder = LabelEncoder().fit(sorted(test_df["LABEL"].unique()))
# Create predictions for meta-model
X_meta = []
y_true = None

for name, (model, preprocess_fn) in models.items():
    print(f"Getting predictions from {name}")
    test_gen = ImageDataGenerator(preprocessing_function=preprocess_fn)
    test_flow = test_gen.flow_from_dataframe(
        test_df,
        x_col="JPG", y_col="LABEL",
        target_size=(224, 224), batch_size=32,
        class_mode="categorical", shuffle=False
    )

    preds = model.predict(test_flow, verbose=0)
    X_meta.append(preds)

    if y_true is None:
        y_true = test_flow.classes

# Stack predictions (shape: [samples, 3 * num_classes])
X_meta = np.concatenate(X_meta, axis=1)
y_true_encoded = label_encoder.transform(y_true)
```

```python
# Train meta-classifier
meta_clf = LogisticRegression(max_iter=1000)
meta_clf.fit(X_meta, y_true_encoded)

# Evaluate stacking model
y_pred_stack = meta_clf.predict(X_meta)
stack_acc = accuracy_score(y_true_encoded, y_pred_stack)
print(f"\n Stacked Model Accuracy: {stack_acc:.4f}")
```



```
Getting predictions from VGG16
Found 947 validated image filenames belonging to 11 classes.
Getting predictions from EfficientNetB0
Found 947 validated image filenames belonging to 11 classes.
Getting predictions from ResNet50V2
Found 947 validated image filenames belonging to 11 classes.

 Stacked Model Accuracy: 0.9704
```

6.5 Training Configuration



```python
def train_model_with_kfold(model_fn, train_df, input_shape=(224, 224, 3), batch_size=32, epochs=20, save_path=None):
    # Map model to its preprocessing function
    preprocess_map = {
        "EfficientNetB0": efficientnet.preprocess_input,
        "ResNet50V2": resnet_v2.preprocess_input,
        "VGG16": vgg16.preprocess_input
    }
    model_name = model_fn.__name__
    preprocess_fn = preprocess_map.get(model_name)

    skf = StratifiedKFold(n_splits=5, shuffle=True, random_state=42)
    accuracies = []
    histories = []

    for fold, (train_idx, val_idx) in enumerate(skf.split(train_df["JPG"], train_df["LABEL"])):
        print(f"\n Fold {fold + 1} - {model_name}")

        fold_train = train_df.iloc[train_idx].reset_index(drop=True)
        fold_val = train_df.iloc[val_idx].reset_index(drop=True)

        # Compute class weights
        class_weights_array = compute_class_weight(
            class_weight='balanced',
            classes=np.unique(fold_train["LABEL"]),
            y=fold_train["LABEL"]
        )
        class_weights = dict(zip(np.unique(fold_train["LABEL"]), class_weights_array))
    train_datagen = ImageDataGenerator(
        preprocessing_function=preprocess_fn,
        rotation_range=20,
        width_shift_range=0.1,
        height_shift_range=0.1,
        zoom_range=0.2,
        horizontal_flip=True
    )
    val_datagen = ImageDataGenerator(preprocessing_function=preprocess_fn)

    train_flow = train_datagen.flow_from_dataframe(
        dataframe=fold_train,
        x_col="JPG",
        y_col="CATEGORY",
        target_size=input_shape[:2],
        batch_size=batch_size,
        class_mode="categorical",
        shuffle=True
    )
    val_flow = val_datagen.flow_from_dataframe(
        dataframe=fold_val,
        x_col="JPG",
        y_col="CATEGORY",
        target_size=input_shape[:2],
        batch_size=batch_size,
        class_mode="categorical",
        shuffle=False
    )
```

All models were trained using the following settings:

- **Input Size**: 224 × 224 pixels
- **Batch Size**: [Specify, 32]
- **Epochs**: [Specify total epochs, 30–50]
- **Optimizer**: Adam (with tuned learning rate)
- **Loss Function**: Categorical Cross-Entropy
- **Evaluation**: Stratified 5-Fold Cross-Validation
- **Callbacks**: EarlyStopping, ReduceLROnPlateau, ModelCheckpoint

6.6 Libraries and Tools Used

- **TensorFlow & Keras**: Model building and training
- **NumPy & pandas**: Data manipulation
- **Matplotlib & Seaborn**: Visualization
- **scikit-learn**: Cross-validation, metrics, class weights
- **OpenCV & PIL**: Image processing and format conversion

7 Experimental Results

This section evaluates the performance of three deep learning models using stratified k-fold validation, hyperparameter tuning with Optuna, and final test accuracies. The primary metric used is classification accuracy:

7.1 Cross-validation Performance

The models were assessed using stratified 5-fold cross-validation on the pre training dataset. The accuracies for each fold are shown below:

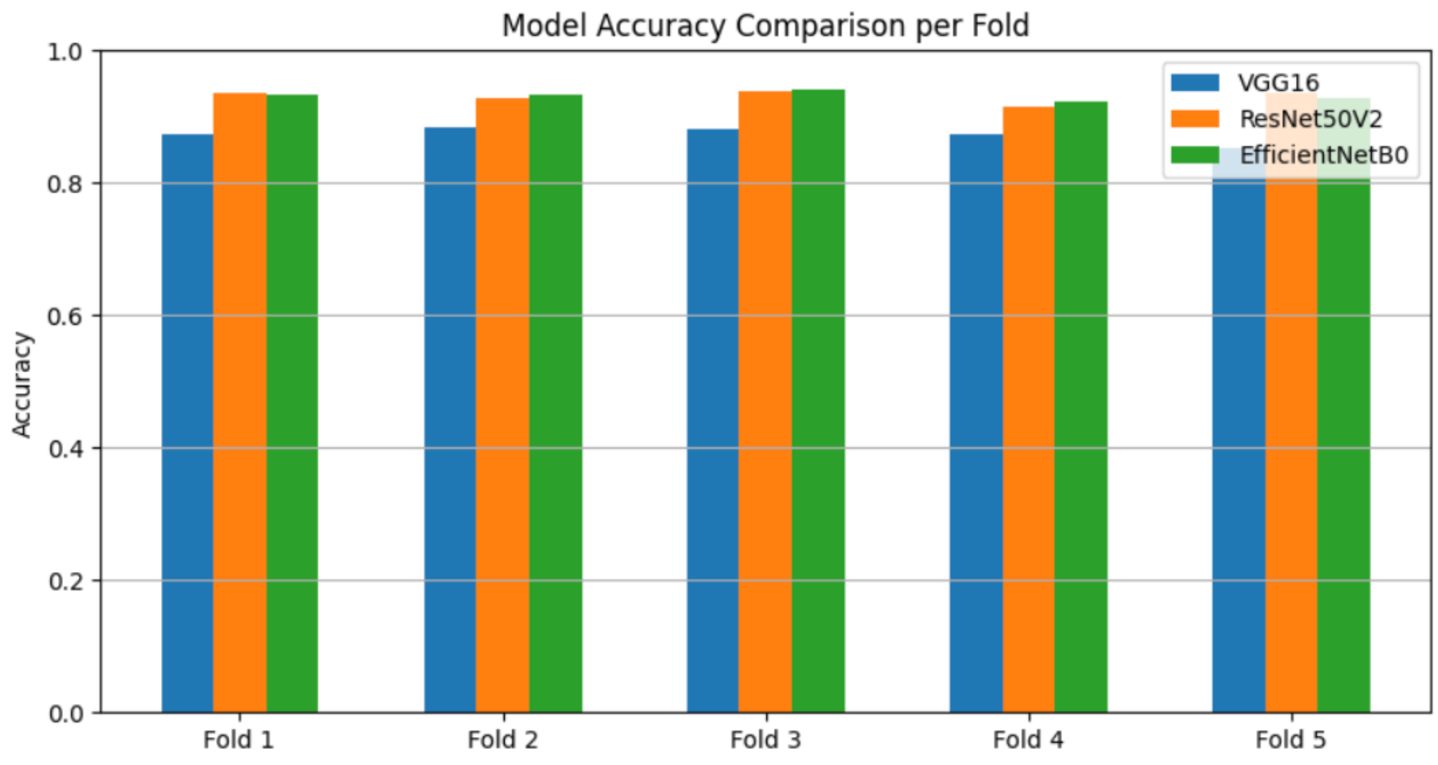


Figure 12 Model Accuracy Comparison per Fold

- .VGG16: Average accuracy if 87.31%, with stable performance across folds.
- EfficientNetB0: Average accuracy of 93.17%, showing strong generalization.
- ResNet50V2: Average accuracy of 93.08%.

7.2 Hyper parameter Tuning

To enhanve model performance and ensure optimal generalization, Hyper parameter optimization was conducted using Optuna optimization framework. This automated search process was applied to all three deep learning models.

Tuning parameters:

- Learning Rate(lr) : ranging between 1e-5 and 1e-3.
- Dropout Rate: varied between 0.2 and 0.5 to prevent overfitting
- Dense Units: Tested Values between 64 and 256 to balance model complexity

The best results from the trials are:

- **EfficientNetB0**:
  - Best Validation Accuracy: **0.9522**
  - Parameters:
    - Learning rate: 0.000416
    - Dropout rate: 0.307
    - Dense units: 434

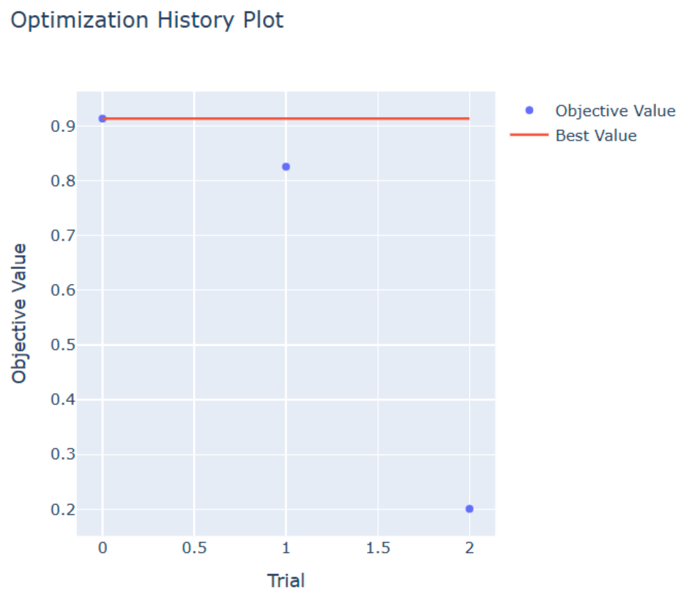


Figure 13 optimization History plot

- **ResNet50V2**:
  - Best Validation Accuracy: **0.9401**
  - Parameters:
    - Learning rate: 0.000198
    - Dropout rate: 0.221
    - Dense units: 298

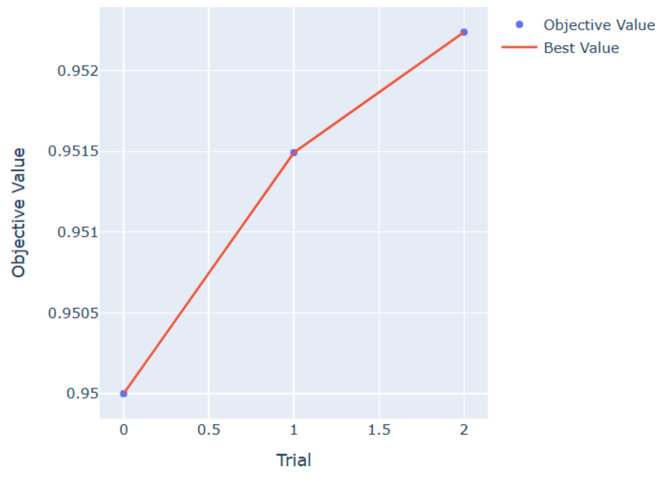


Figure 14 ResNet 50 optimization history

- **VGG16**:
  - Best Validation Accuracy: **0.9132**
  - Parameters:
    - Learning rate: 0.000132
    - Dropout rate: 0.389
    - Dense units: 319

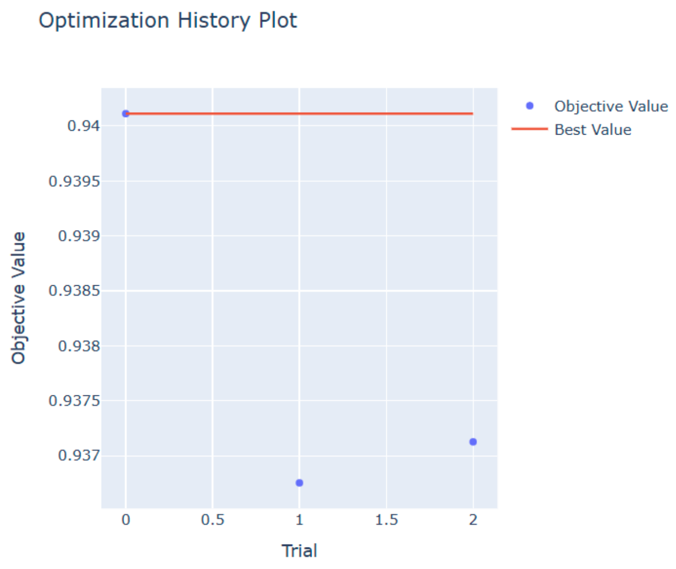
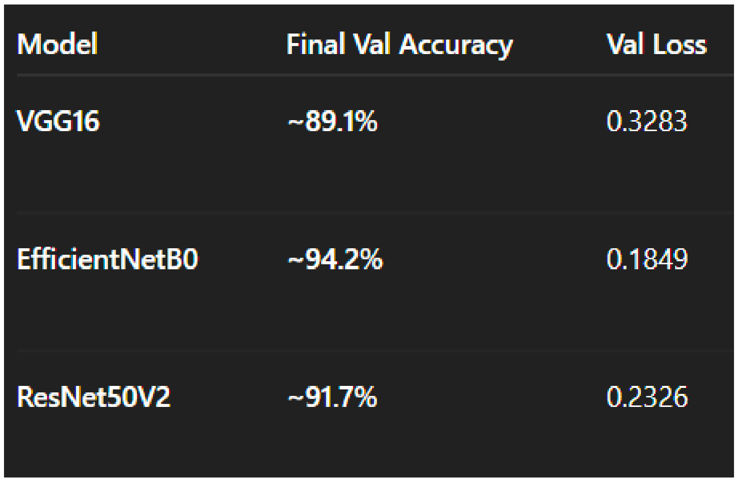


Figure 15 VGG16 optimization history plot

These tuned parameters improved the validation accuracies compared to the cross-validation averages.

7.3 Retrain Model

To improve model performance, after hyper parameter tuning again retrained three models using tuned hyperparameter obtained via Optuna optimization. Each model trained for up to 12 epochs with early stopping and learning rate reduction strategies applied.



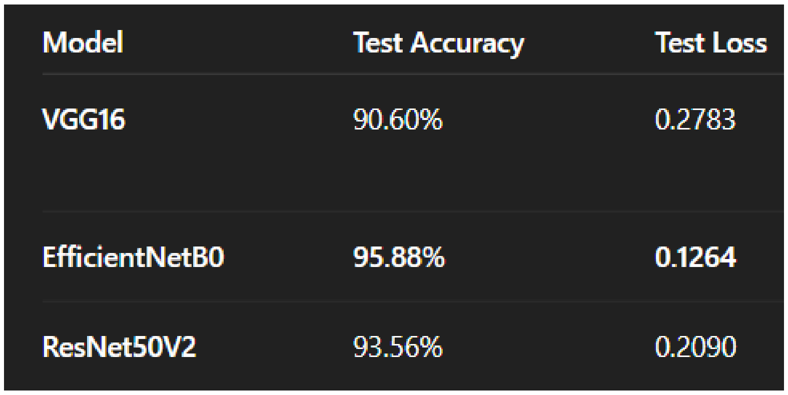
Model	Final Val Accuracy	Val Loss
VGG16	~89.1%	0.3283
EfficientNetB0	~94.2%	0.1849
ResNet50V2	~91.7%	0.2326

Observations

- **EfficientNetB0** consistently showed faster convergence and higher validation performance.
- **VGG16** started slow but improved drastically after a few epochs, indicating good learning behavior.
- **ResNet50V2** plateaued slightly in later epochs, suggesting it may benefit from unfreezing more layers or fine-tuning further.

7.4 Testing

After tuning each model using the best hyperparameters obtained during cross-validation and Optuna-based optimization, the final evaluation was performed was performed on the test dataset, which include 15% of data with 947 images split with 11 endangered animal classes.



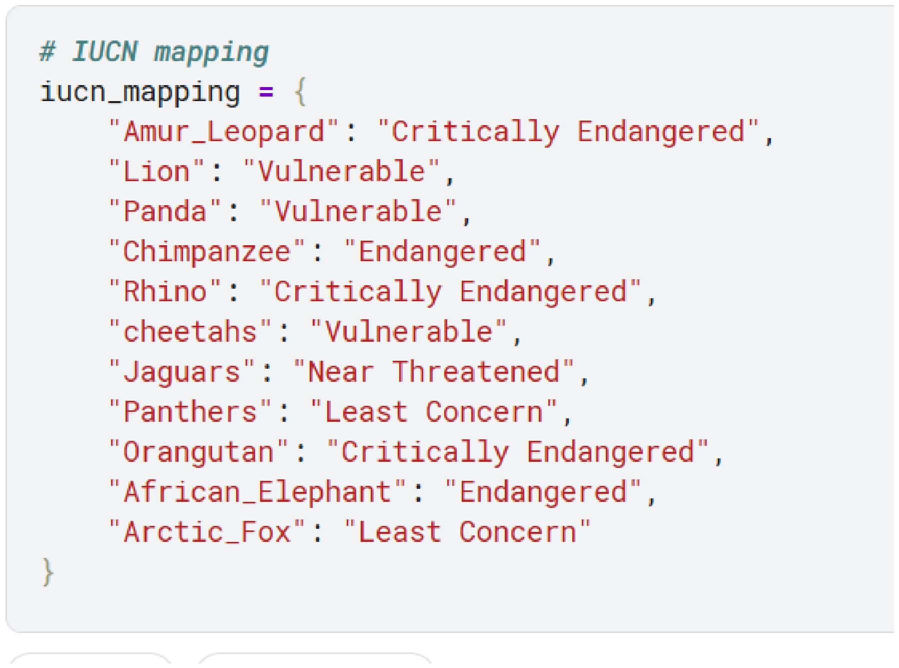
Model	Test Accuracy	Test Loss
VGG16	90.60%	0.2783
EfficientNetB0	**95.88%**	**0.1264**
ResNet50V2	93.56%	0.2090

Observations

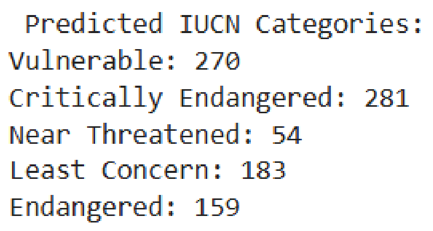
- **EfficientNetB0,** is the best performed model comparing to other model. This model achieved highest accuracy and demonstrate excellent generalization
- **VGG16**, while not the top performer, still achieved over 90% test accuracy, indicating effective learning after tuning.
- **ResNet50V2** showed competitive results with a high test accuracy and relatively low inference time.

7.5 Mapping predictions to IUCN Red List Categories

After the species of animal were classified using the trained models, the next step was to determine the conservation status of each predicted species. This was achieved by mapping the predicted class labels to IUCN Red List categories as defined by the IUCN Red List of Threatened species.



```
# IUCN mapping
iucn_mapping = {
    "Amur_Leopard": "Critically Endangered",
    "Lion": "Vulnerable",
    "Panda": "Vulnerable",
    "Chimpanzee": "Endangered",
    "Rhino": "Critically Endangered",
    "cheetahs": "Vulnerable",
    "Jaguars": "Near Threatened",
    "Panthers": "Least Concern",
    "Orangutan": "Critically Endangered",
    "African_Elephant": "Endangered",
    "Arctic_Fox": "Least Concern"
}
```



```
 Predicted IUCN Categories:
Vulnerable: 270
Critically Endangered: 281
Near Threatened: 54
Least Concern: 183
Endangered: 159
```

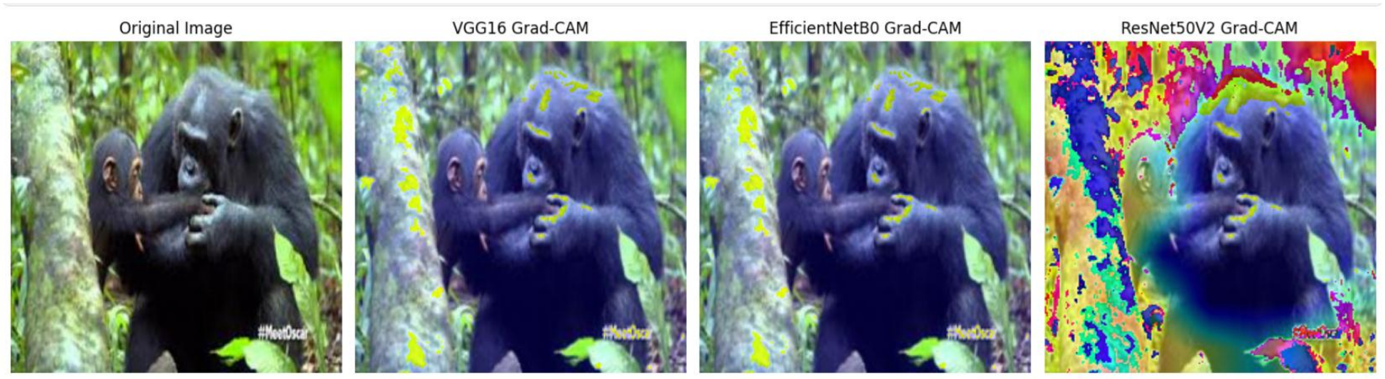
A dictionary was created to link 11 species to their IUCN categories, such as Critically Endangered, Endangered, and Vulnerable. The final predictions were mapped to their corresponding IUCN categories, allowing the project to classify animals and interpret conservation significance. The output provided insight into predicted extinction risk distribution across test data and generated a count of predicted categories, highlighting the model's practical relevance in biodiversity monitoring and conservation prioritization.

7.5 Model Explainability using Grad-Cam

To enhance the interpretability of model predictions, **Grad-CAM (Gradient-weighted Class Activation Mapping)** was applied to visualize which regions of an image influenced the model’s decision the most. This technique generates heatmaps that highlight the most discriminative areas in an image for a specific class prediction.

Using Grad-CAM, it was observed that the models (particularly EfficientNetB0 and ResNet50V2) correctly

focused on key visual features such as the animal's face, body texture, and unique patterns.

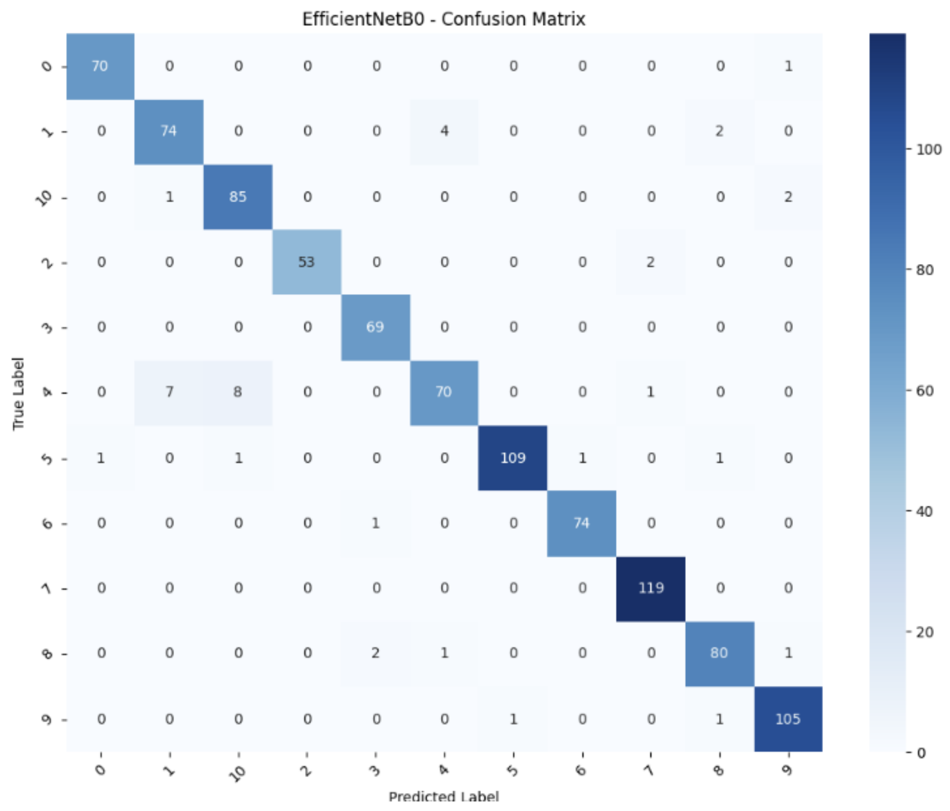


This confirmed that the models were not relying on irrelevant background information.

The Grad-CAM results not only support the reliability of predictions but also provide transparency, which is critical when applying AI systems in conservation and ecological monitoring.

8 Evaluation Metrics

1. Confusion matrix



	0	1	10	2	3	4	5	6	7	8	9
0	70	0	0	0	0	0	0	0	0	0	1
1	0	74	0	0	0	4	0	0	0	2	0
10	0	1	85	0	0	0	0	0	0	0	2
2	0	0	0	53	0	0	0	0	2	0	0
3	0	0	0	0	69	0	0	0	0	0	0
4	0	7	8	0	0	70	0	0	1	0	0
5	1	0	1	0	0	0	109	1	0	1	0
6	0	0	0	0	1	0	0	74	0	0	0
7	0	0	0	0	0	0	0	0	119	0	0
8	0	0	0	0	2	1	0	0	0	80	1
9	0	0	0	0	0	0	1	0	0	1	105

Figure 16 EfficientNetBO Confusion Matrix

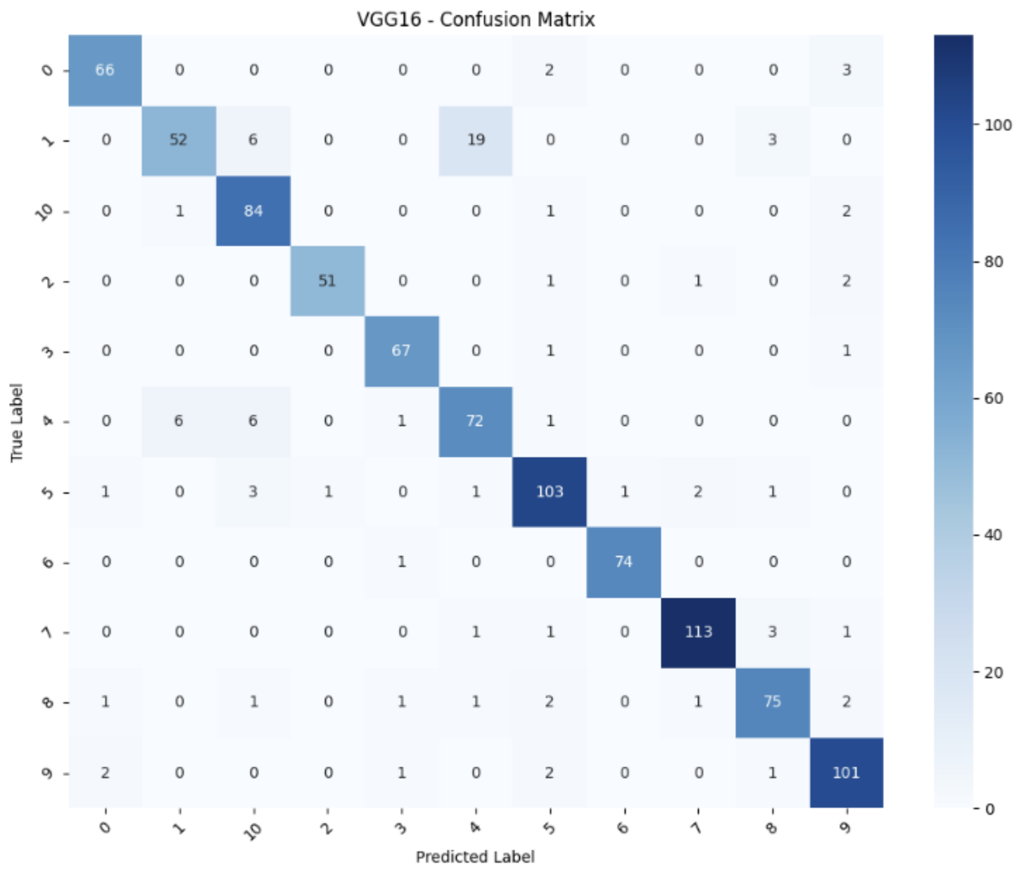


Figure 17 VGG16 - Confusion Matix

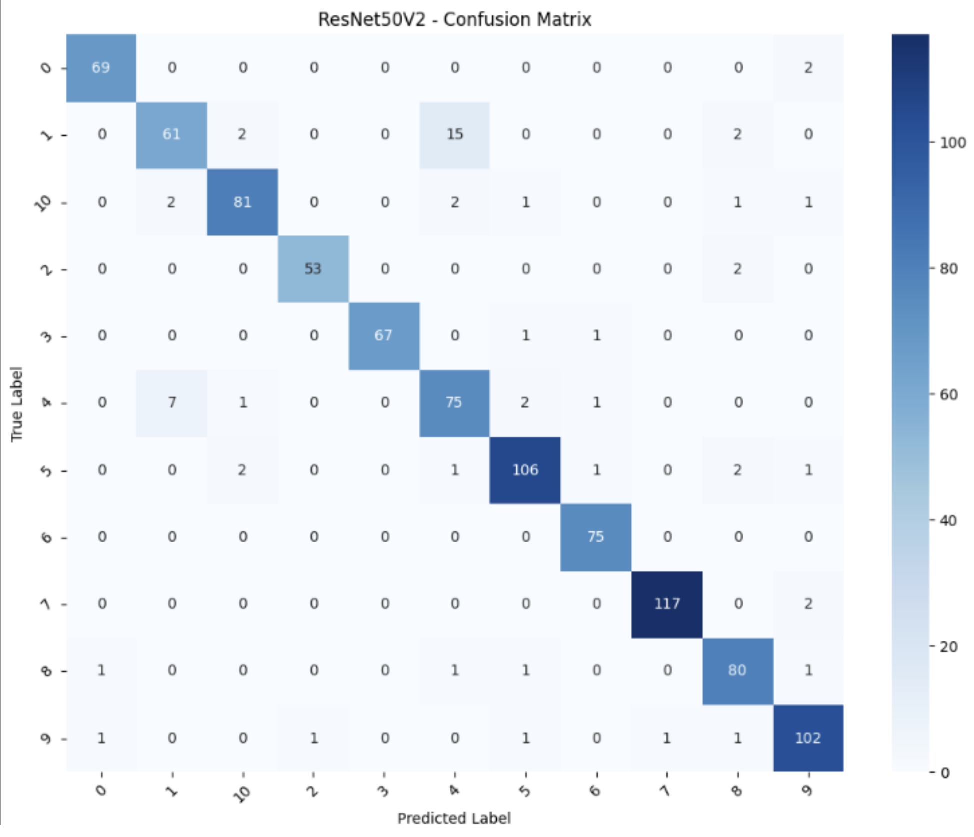


Figure 18 ResNet50V2 - Confusion Matrix

2. Train vs test performance analysis

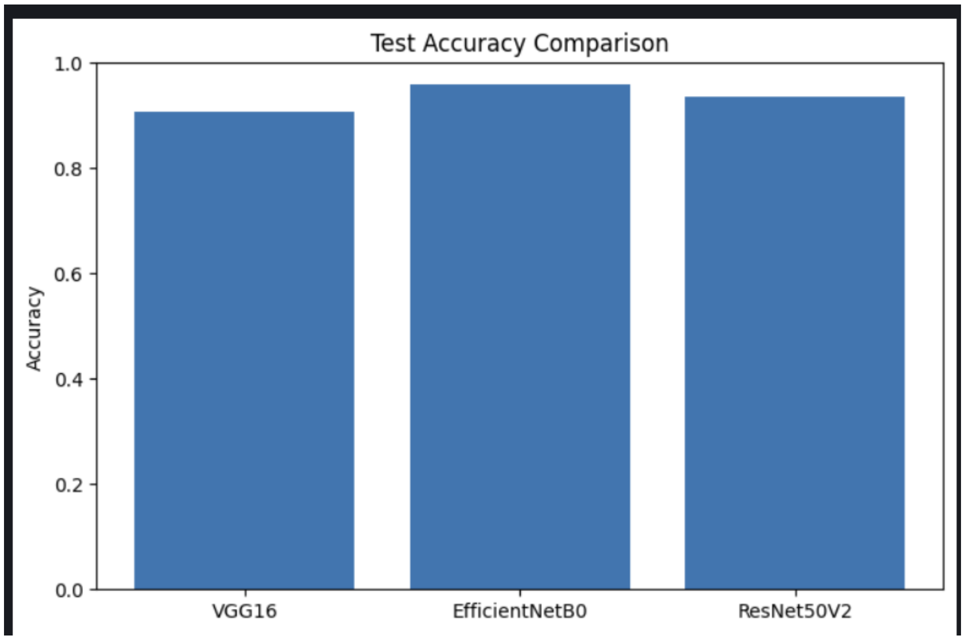
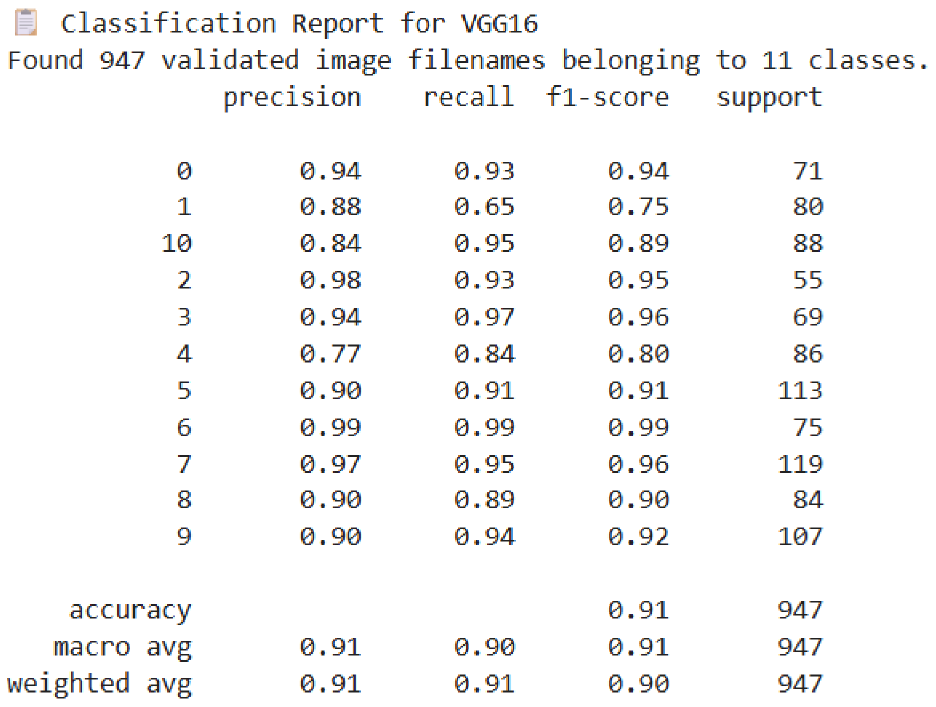


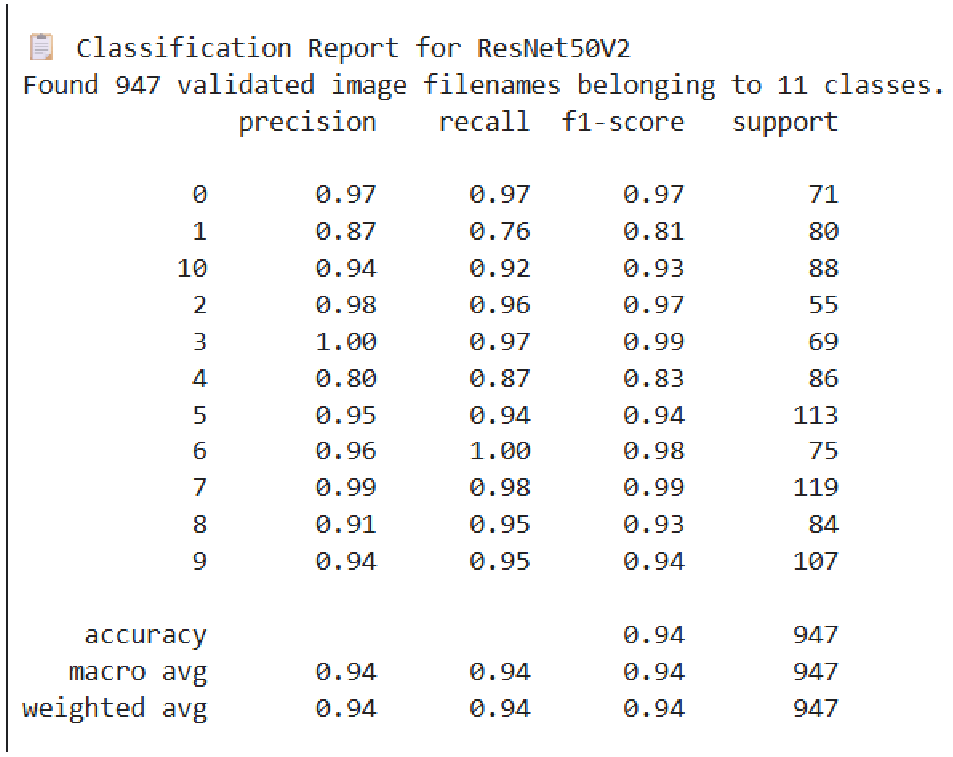
Figure 19 Test Accuracy Comparision

3. Classification Report



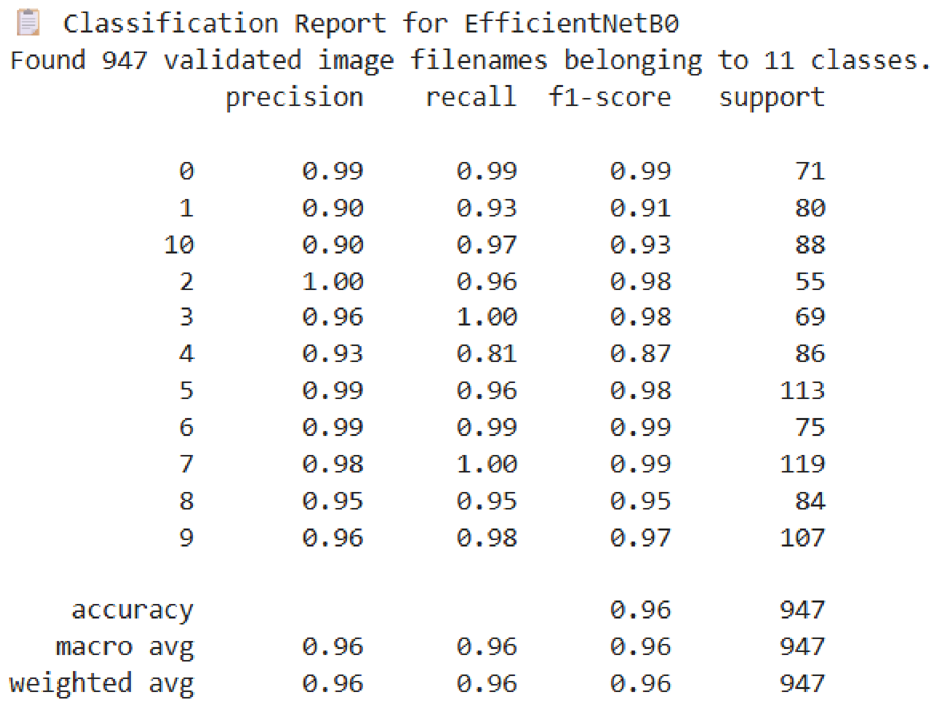
```
📋 Classification Report for VGG16
Found 947 validated image filenames belonging to 11 classes.
              precision    recall  f1-score   support

           0       0.94      0.93      0.94        71
           1       0.88      0.65      0.75        80
          10       0.84      0.95      0.89        88
           2       0.98      0.93      0.95        55
           3       0.94      0.97      0.96        69
           4       0.77      0.84      0.80        86
           5       0.90      0.91      0.91       113
           6       0.99      0.99      0.99        75
           7       0.97      0.95      0.96       119
           8       0.90      0.89      0.90        84
           9       0.90      0.94      0.92       107

    accuracy                           0.91       947
   macro avg       0.91      0.90      0.91       947
weighted avg       0.91      0.91      0.90       947
```

Figure 20 VGG16 - Classification Report



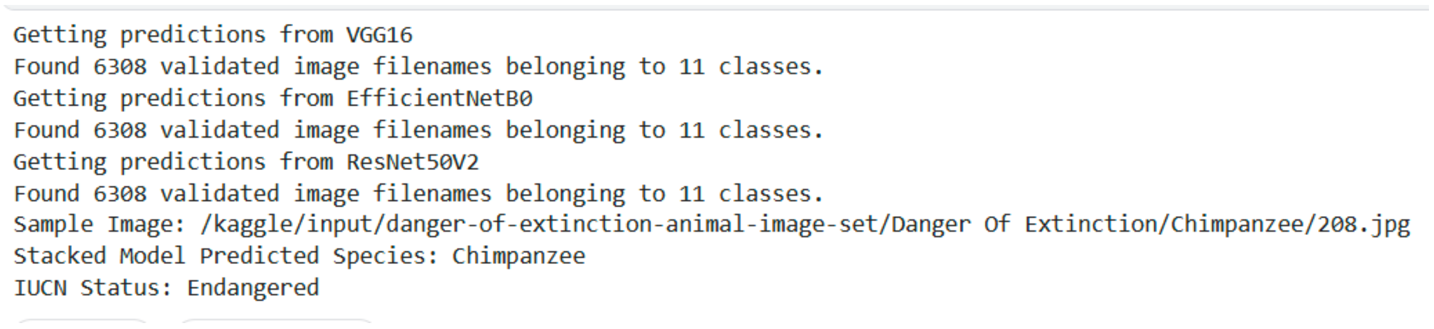
```
📋 Classification Report for ResNet50V2
Found 947 validated image filenames belonging to 11 classes.
              precision    recall  f1-score   support

           0       0.97      0.97      0.97        71
           1       0.87      0.76      0.81        80
          10       0.94      0.92      0.93        88
           2       0.98      0.96      0.97        55
           3       1.00      0.97      0.99        69
           4       0.80      0.87      0.83        86
           5       0.95      0.94      0.94       113
           6       0.96      1.00      0.98        75
           7       0.99      0.98      0.99       119
           8       0.91      0.95      0.93        84
           9       0.94      0.95      0.94       107

    accuracy                           0.94       947
   macro avg       0.94      0.94      0.94       947
weighted avg       0.94      0.94      0.94       947
```

Figure 21 ResNet50V2 - Classification Report



```
📋 Classification Report for EfficientNetB0
Found 947 validated image filenames belonging to 11 classes.
              precision    recall  f1-score   support

           0       0.99      0.99      0.99        71
           1       0.90      0.93      0.91        80
          10       0.90      0.97      0.93        88
           2       1.00      0.96      0.98        55
           3       0.96      1.00      0.98        69
           4       0.93      0.81      0.87        86
           5       0.99      0.96      0.98       113
           6       0.99      0.99      0.99        75
           7       0.98      1.00      0.99       119
           8       0.95      0.95      0.95        84
           9       0.96      0.98      0.97       107

    accuracy                           0.96       947
   macro avg       0.96      0.96      0.96       947
weighted avg       0.96      0.96      0.96       947
```

Figure 22 EfficientNetB0 - Classification Report



```
Getting predictions from VGG16
Found 6308 validated image filenames belonging to 11 classes.
Getting predictions from EfficientNetB0
Found 6308 validated image filenames belonging to 11 classes.
Getting predictions from ResNet50V2
Found 6308 validated image filenames belonging to 11 classes.
Sample Image: /kaggle/input/danger-of-extinction-animal-image-set/Danger Of Extinction/Chimpanzee/208.jpg
Stacked Model Predicted Species: Chimpanzee
IUCN Status: Endangered
```

Final output of using Grad-CAM in all three model. First the model correctly identified the species and mapped it to the correct IUCN category, validating both the classification accuracy and the integration with conservation data.

9 Conclusion

In this project, a deep learning is used for classification of the images and to identify endangered animal species and associate them with IUCN Red list categories. Evaluated with 3 pre-trained modes VGG16, EfficientNetB0, and ResNet50V2 using cross-validation and Optuna hyper parameter tuning, followed by stacking with logistic regression meta-classifier. Among the individual models, EfficientNetB0 performed best with 95.88% accuracy, while the stacked model achieved the highest accuracy of 97.04% on the test set. To better interpret model decisions, we also implemented Grad-CAM visualizations, which helped validate that the models focused on relevant features in the animal images. Key challenges included class imbalance and ensuring consistent label mapping for IUCN integration. In future work, the system could be enhanced by incorporating additional contextual data or expanding the model to support more species and deploying the model as a conservation tool in the field.

Limitations and Challenges Faced

Despite the encouraging results, several limitations and challenges were encountered during the project:

- **Limited Data and Class Imbalance:** The dataset, while covering 11 classes, is relatively small by deep learning standards (6484 images in total). Some classes had very few examples (e.g., fewer than 400 Arctic Fox images). This limited data regime makes it hard for models to learn all variations of the species and increases the risk of overfitting.
- **Image Quality and Variability:** The images varied in quality – some were high resolution, well-lit photographs, while others were low resolution or contained the animal at a small scale in the frame. Such variability can confuse models.
- **Overfitting Concerns:** Training powerful models on a small dataset runs the risk of overfitting – memorizing the training images rather than learning generalizable features.

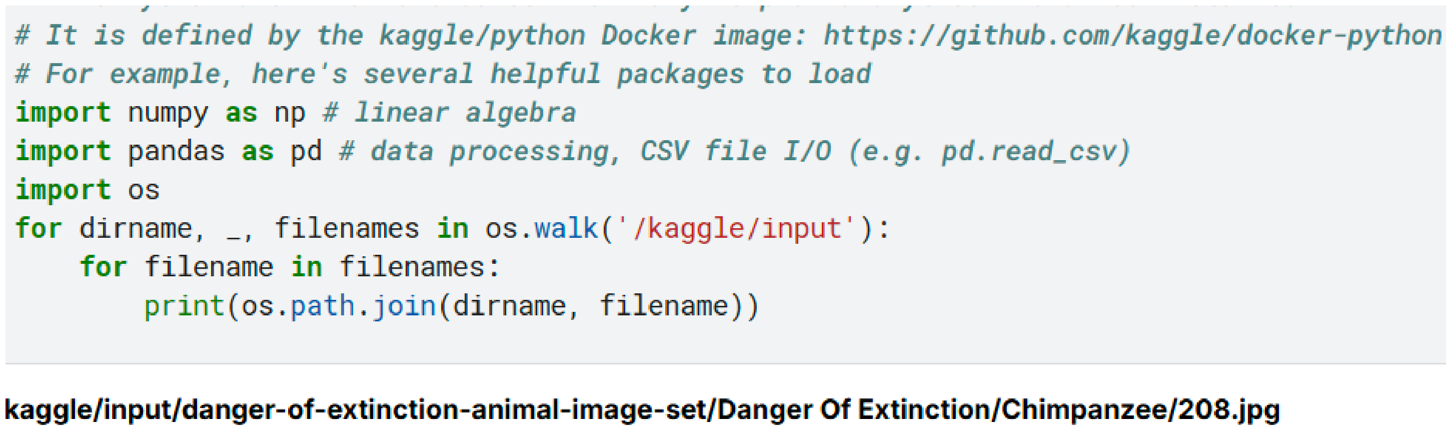
Summary of Key Results:

- EfficientNetB0 achieved the highest classification accuracy of 95%, followed by ResNet50V2 (94%) and VGG16 (89%) after hyperparameter tuning, demonstrating the effectiveness of transfer learning.
- The stacking ensemble further improved prediction consistency, and mapping the predictions to IUCN Red List categories enabled identification of extinction risk levels.
- Stratified K-Fold Cross-Validation and Grad-CAM provided robust evaluation and interpretability of the models.

References

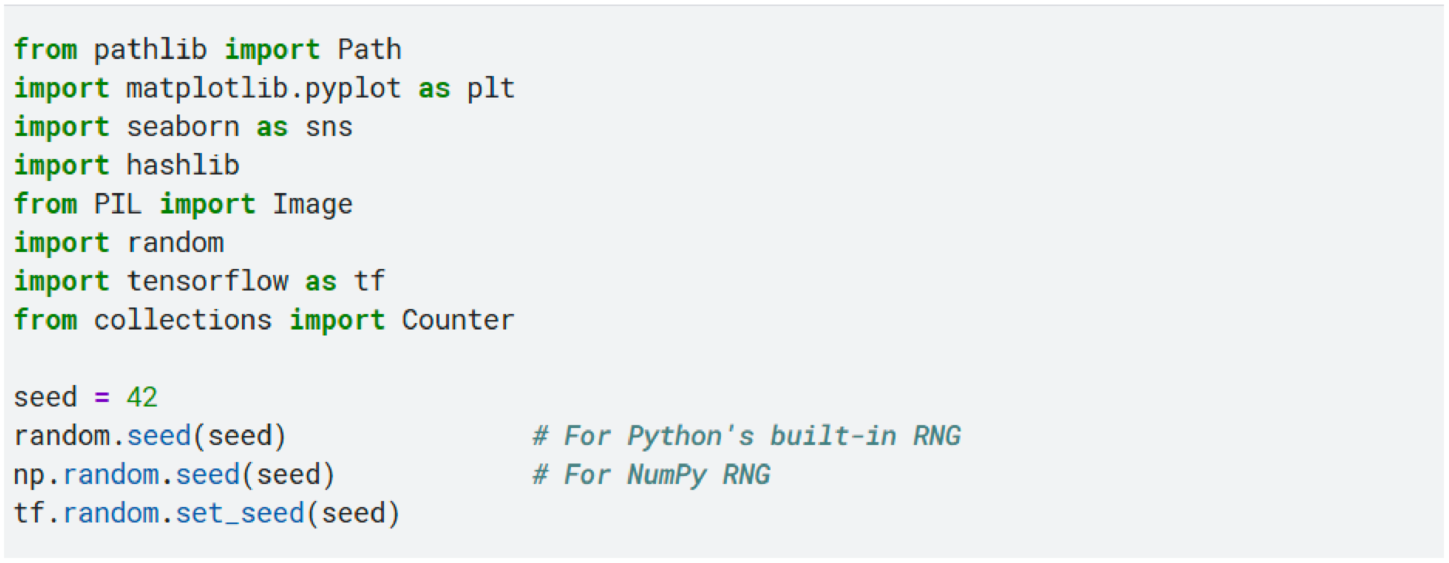
- *IUCN red list categories and Criteria* (2001). Gland, Switzerland: IUCN--The World Conservation Union.
- *What is agricultural biodiversity and what are ecosystem services?* (no date) *Plant Production and Protection Division: What are biodiversity and ecosystem services*. Available at: https://www.fao.org/agriculture/crops/thematic-sitemap/theme/spi/scpi-home/managing-ecosystems/biodiversity-and-ecosystem-services/what1/en/ (Accessed: 23 May 2025).
- 'The history of emotions: An introduction' (2015) *Choice Reviews Online*, 53(02). doi:10.5860/choice.192178.
- RODRIGUES, A. *et al.* (2006) 'The value of the IUCN Red List for Conservation', *Trends in Ecology &amp; Evolution*, 21(2), pp. 71–76. doi:10.1016/j.tree.2005.10.010.
- *Science and the endangered species act* (1995). Washington, DC: National Academy Press.
- Tarkik-Gautam-Ranjan (2022) 'Global deforestation and its relation to animal extinction', *World Journal of Advanced Research and Reviews*, 15(1), pp. 499–511. doi:10.30574/wjarr.2022.15.1.0749.
- Fisher, D.O. and Humphreys, A.M. (2024) 'Evidence for modern extinction in plants and animals', *Biological Conservation*, 298, p. 110772. doi:10.1016/j.biocon.2024.110772.
- Nazir, S. and Kaleem, M. (2024) 'Object classification and visualization with Edge Artificial Intelligence for a customized camera trap platform', *Ecological Informatics*, 79, p. 102453. doi:10.1016/j.ecoinf.2023.102453.
- McMahon, K. (2025) *How ai is reshaping wildlife conservation - for better or worse*, *The Verge*. Available at: https://www.theverge.com/ai-artificial-intelligence/653322/ai-wildlife-conservation (Accessed: 24 May 2025).
- Tuia, D. *et al.* (2022) 'Perspectives in Machine Learning for Wildlife Conservation', *Nature Communications*, 13(1). doi:10.1038/s41467-022-27980-y.
- *IUCN red list* (2025) *Wikipedia*. Available at: https://en.wikipedia.org/wiki/IUCN_Red_List#:~:text=Species%20are%20classified%20by%20the,6%20%5B%2025 (Accessed: 24 May 2025).
- Libretexts (2022) *9.2: Estimating extinction risk*, *Biology LibreTexts*. Available at: https://bio.libretexts.org/Bookshelves/Ecology/Conservation_Biology_in_Sub-Saharan_Africa_(Wilson_and_Primack)/09%3A_Applied_Population_Biology/9.02%3A_Estimating_Extinction_Risk#:~:text=tools%20for%20making%20such%20predictions,example%2C%20modify%20harvesting%20regulations%2C%20perform (Accessed: 24 May 2025).
- Subek Sharma Department of Electronics and Computer Engineering *et al.* (no date) *Evaluating transfer learning in deep learning models for classification on a custom wildlife dataset: Can yolov8 surpass other architectures?* Available at: https://arxiv.org/html/2408.00002v1#:~:text=However%2C%20poaching%20and%20unintentional%20human,of%20different%20architectures%20like%20DenseNet (Accessed: 24 May 2025).
- 'Wildlife species classification from camera trap images using fine-tuning EFFICIENTNETV2' (2024) *International Journal of Intelligent Engineering and Systems*, 17(6), pp. 624–638. doi:10.22266/ijies2024.1231.48.
- (2021) *Review for 'camera trapping and spatially explicit capture–recapture for the monitoring and conservation management of Lions: Insights from a globally important population in Tanzania'* [Preprint]. doi:10.1002/2688-8319.12129/v1/review1.

Appendices



```python
# It is defined by the kaggle/python Docker image: https://github.com/kaggle/docker-python
# For example, here's several helpful packages to load
import numpy as np # linear algebra
import pandas as pd # data processing, CSV file I/O (e.g. pd.read_csv)
import os
for dirname, _, filenames in os.walk('/kaggle/input'):
    for filename in filenames:
        print(os.path.join(dirname, filename))
```

```
kaggle/input/danger-of-extinction-animal-image-set/Danger Of Extinction/Chimpanzee/208.jpg
```



```python
from pathlib import Path
import matplotlib.pyplot as plt
import seaborn as sns
import hashlib
from PIL import Image
import random
import tensorflow as tf
from collections import Counter

seed = 42
random.seed(seed)                # For Python's built-in RNG
np.random.seed(seed)             # For NumPy RNG
tf.random.set_seed(seed)
```

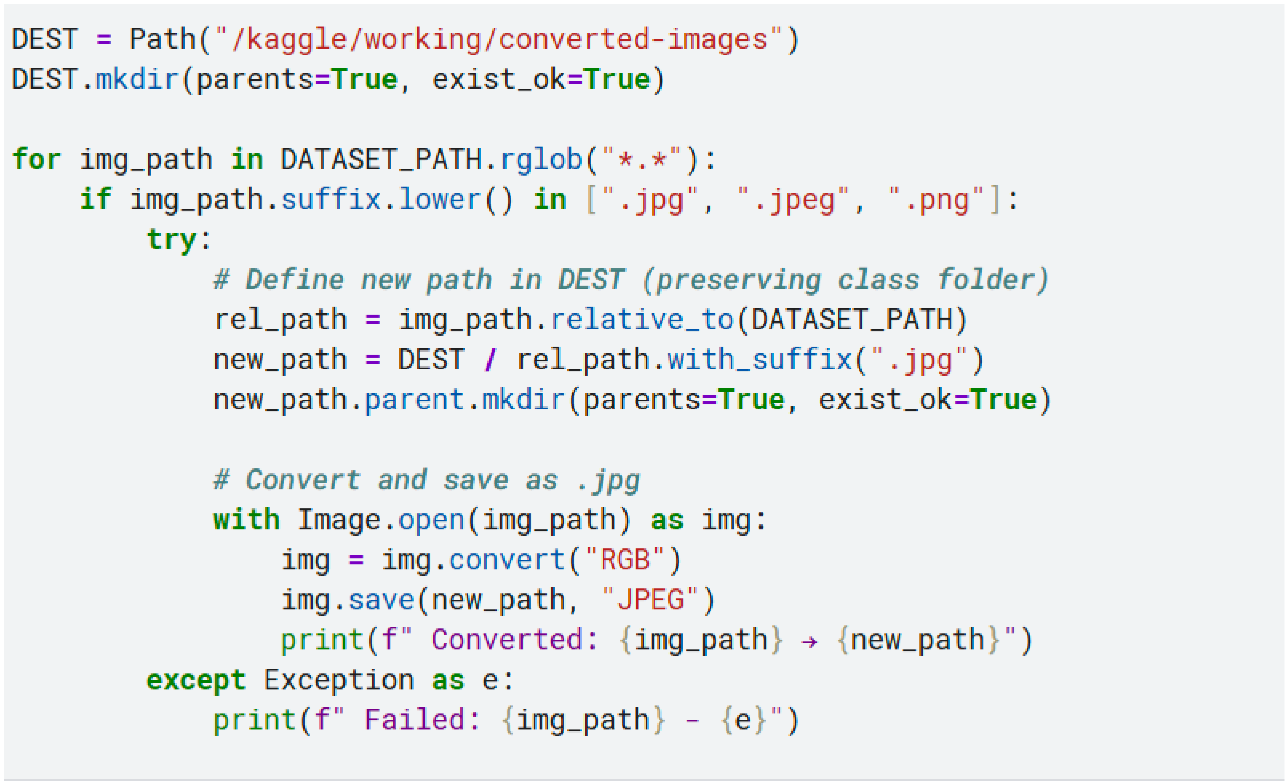
```python
DATASET_PATH = Path("/kaggle/input/danger-of-extinction-animal-image-set/Danger Of Extinction")
```

Counting the number of image files by their extensions(JPG, PNG) in the dataset folder and prints the count for each type.

```python
all_images = list(DATASET_PATH.rglob("*.*"))
ext_counts = Counter(p.suffix.lower() for p in all_images if p.is_file())

# Show counts
for ext, count in ext_counts.items():
    print(f"{ext}: {count} images")
```

```
.jpg: 6307 images
.jpeg: 113 images
.png: 63 images
: 1 images
```

```python
DEST = Path("/kaggle/working/converted-images")
DEST.mkdir(parents=True, exist_ok=True)
```



```python
DEST = Path("/kaggle/working/converted-images")
DEST.mkdir(parents=True, exist_ok=True)

for img_path in DATASET_PATH.rglob("*.*"):
    if img_path.suffix.lower() in [".jpg", ".jpeg", ".png"]:
        try:
            # Define new path in DEST (preserving class folder)
            rel_path = img_path.relative_to(DATASET_PATH)
            new_path = DEST / rel_path.with_suffix(".jpg")
            new_path.parent.mkdir(parents=True, exist_ok=True)

            # Convert and save as .jpg
            with Image.open(img_path) as img:
                img = img.convert("RGB")
                img.save(new_path, "JPEG")
                print(f" Converted: {img_path} → {new_path}")
        except Exception as e:
            print(f" Failed: {img_path} - {e}")
```

```
# Get all .jpg images (including subfolders)
image_paths = list(DEST.rglob("*.jpg"))

# Build the DataFrame
df = pd.DataFrame({
    "JPG": [str(p) for p in image_paths],
    "CATEGORY": [p.parent.name for p in image_paths]
})
print(df.head())
print(df.tail())
```

```
df = df.sample(frac=1, random_state=seed).reset_index(drop=True)
```

```
plt.figure(figsize=(14,6))
sns.countplot(data = df, x='CATEGORY', order=df['CATEGORY'].value_counts().index)
plt.xticks(rotation = 45)
plt.title("Class Distribution")
plt.tight_layout()
plt.show()
```

```
# Function to generate a hash from image file
def hash_file(file_path):
    with open(file_path, "rb") as f:
        return hashlib.md5(f.read()).hexdigest()

# Add hash column to the DataFrame
df["hash"] = df["JPG"].apply(hash_file)

original_count = len(df)
# Drop duplicate images based on the hash, and clean up
df = df.drop_duplicates(subset="hash").drop(columns="hash").reset_index(drop=True)
final_count = len(df)

duplicates = original_count - final_count
print("total duplicate images:", duplicates)
```

```python
def get_image_info(path):
    try:
        with Image.open(path) as img:
            return img.size[0], img.size[1], img.mode  # width, height, mode
    except Exception as e:
        print(f"Error reading {path}: {e}")
        return None, None, None

# Apply function to each image path
df[["Width", "Height", "Mode"]] = df["JPG"].apply(
    lambda p: pd.Series(get_image_info(p))
)

# Check the result
print(df)
```

```python
# === Visualize Image Size Distributions === #
plt.figure(figsize=(10, 5))
sns.histplot(df['Width'], bins=30, kde=True, color='teal')
plt.title('Image Width Distribution')
plt.xlabel('Width')
plt.ylabel('Frequency')
plt.tight_layout()
plt.show()

plt.figure(figsize=(10, 5))
sns.histplot(df['Height'], bins=30, kde=True, color='orange')
plt.title('Image Height Distribution')
plt.xlabel('Height')
plt.ylabel('Frequency')
plt.tight_layout()
plt.show()

plt.figure(figsize=(6, 4))
sns.countplot(data=df, x='Mode')
plt.title('Image Mode Distribution')
plt.tight_layout()
plt.show()
```

```python
# Select 11 animal categories from df
animals = [
    "Amur_Leopard", "Lion", "Panda", "Chimpanzee", "Rhino",
    "cheetahs", "Jaguars", "Panthers", "Orangutan",
    "African_Elephant", "Arctic_Fox"
]

# Collect one sample image (index 80 if available) from each category
animal_list = []
labels = []

for category in animals:
    subset = df[df["CATEGORY"] == category].reset_index(drop=True)

    # Check if index 80 exists, otherwise take last available
    idx = 80 if len(subset) > 80 else len(subset) - 1

    img_path = subset.loc[idx, "JPG"]
    img = cv2.imread(img_path)

    if img is not None:
        img = cv2.cvtColor(img, cv2.COLOR_BGR2RGB)
        animal_list.append(img)
        labels.append(category)
    else:
        print(f"Could not load image for {category}")
print("Dataset folders in /kaggle/input:")
print(os.listdir("/kaggle/input"))
```

```
Dataset folders in /kaggle/input:
['danger-of-extinction-animal-image-set']
```

+ Code + Markdown

```

input_dir = "/kaggle/input/danger-of-extinction-animal-image-set/Danger Of Extinction"
output_dir = "/kaggle/working/resized-images"
size = (224, 224)

os.makedirs(output_dir, exist_ok=True)

for root, _, files in os.walk(input_dir):
    for file in files:
        if file.lower().endswith(('.jpg', '.jpeg', '.png')):
            input_path = os.path.join(root, file)
            relative_path = os.path.relpath(input_path, input_dir)
            save_path = os.path.join(output_dir, relative_path)

            os.makedirs(os.path.dirname(save_path), exist_ok=True)
```

```
# Point to resized image folder
resized_path = Path("/kaggle/working/resized-images")
data = []
for img_path in resized_path.rglob("*.jpg"):
    try:
        with Image.open(img_path) as img:
            width, height = img.size
            resolution = width * height
            label = img_path.parent.name  # category = folder name
            data.append((str(img_path), label, width, height, resolution))
    except:
        print("Failed:", img_path)

df = pd.DataFrame(data, columns=["JPG", "CATEGORY", "Width", "Height", "Resolution"])
print("Sample rows:\n", df.head())
print("\nUnique resolutions:", df["Resolution"].unique())
print("\nImages per category:\n", df["CATEGORY"].value_counts())
```

```python
# Encode labels
label_encoder = LabelEncoder()
df["LABEL"] = label_encoder.fit_transform(df["CATEGORY"])

print(f"Total dataset size: {len(df)}")
train_df, test_df = train_test_split(df, test_size=0.15, stratify=df['LABEL'], random_state=42)
print(f"Training set size: {len(train_df)}")
print(f"Test set size: {len(test_df)}")
```

```
Total dataset size: 6307
Training set size: 5360
Test set size: 947
```

```python
#  Compute class weights from training set
classes = sorted(train_df["LABEL"].unique())
class_weights_array = compute_class_weight(
    class_weight="balanced",
    classes=np.array(classes),
    y=train_df["LABEL"]
)
class_weights = dict(zip(classes, class_weights_array))
print("Class Weights:", class_weights)
```

```python
plt.figure(figsize=(12, 6))
sns.boxplot(data=df, x="CATEGORY", y="Resolution")
plt.xticks(rotation=45)
plt.title("Image Resolution by Category (After Resizing)")
plt.tight_layout()
plt.show()
```

```python
def build_resnet(input_shape, num_classes):
    input_layer = Input(shape=input_shape)
    x = resnet_v2.preprocess_input(input_layer)
    base = ResNet50V2(include_top=False, weights="imagenet", input_tensor=x)
    base.trainable = False
    x = GlobalAveragePooling2D()(base.output)
    x = Dropout(0.3)(x)
    output = Dense(num_classes, activation="softmax")(x)
    return Model(inputs=input_layer, outputs=output, name="ResNet50V2")

def build_vgg(input_shape, num_classes):
    input_layer = Input(shape=input_shape)
    x = vgg16.preprocess_input(input_layer)
    base = VGG16(include_top=False, weights="imagenet", input_tensor=x)
    for layer in base.layers[:10]:
        layer.trainable = False
    x = GlobalAveragePooling2D()(base.output)
    x = Flatten()(x)
    x = Dense(128, activation="relu")(x)
    x = Dropout(0.3)(x)
    x = Dense(64, activation="relu")(x)
    x = Dropout(0.3)(x)
    output = Dense(num_classes, activation="softmax")(x)
    return Model(inputs=input_layer, outputs=output, name="VGG16")

def build_effnet(input_shape, num_classes):
    input_layer = Input(shape=input_shape)
    x = efficientnet.preprocess_input(input_layer)
    base = EfficientNetB0(include_top=False, weights="imagenet", input_tensor=x)
```

```python
# Define Grad-CAM function
def compute_gradcam(model, img_array, conv_layer_name, class_idx):
    grad_model = Model(inputs=model.inputs, outputs=[model.get_layer(conv_layer_name).output, model.output])

    with tf.GradientTape() as tape:
        conv_outputs, predictions = grad_model(img_array)
        class_score = predictions[:, class_idx]

    grads = tape.gradient(class_score, conv_outputs)
    pooled_grads = tf.reduce_mean(grads, axis=(0, 1, 2))
    conv_outputs = conv_outputs[0]
    heatmap = tf.reduce_mean(conv_outputs * pooled_grads, axis=-1)
    heatmap = np.maximum(heatmap, 0)
    heatmap /= np.max(heatmap) + 1e-10  # Avoid division by zero
    return heatmap  # Return heatmap as NumPy array
```

```python
X_meta = []
y_true = test_flow.classes  # Store true labels
for name, (model, preprocess_fn) in models.items():
    print(f"Getting predictions from {name}")
    test_gen = ImageDataGenerator(preprocessing_function=preprocess_fn)
    test_flow = test_gen.flow_from_dataframe(
        test_df,
        x_col="JPG", y_col="LABEL",
        target_size=(224, 224), batch_size=32,
        class_mode="categorical", shuffle=False
    )
    preds = model.predict(test_flow, verbose=0)
    X_meta.append(preds)
```