DATA612 Final Write-Up

Dennis Reyes, Cole Bromfield

# Problem Statement

Wild animals can present a distinct threat to safety and security, whether of individuals, livestock, or assets. As such, early detection is crucial. A Convolutional Neural Network (CNN) model can provide helpful aid in this endeavor, detecting the presence of a wild animal, in addition to whether it presents a danger. The differentiation of dangerous animals and harmless ones is left up to the user and may differ by his or her goals for the technology, as certain species may present a higher level of danger to human life, while others may pose a greater threat to livestock or assets.

# Deep Learning Model

Our chosen model is YOLO (You Only Look Once). Unlike traditional object detection methods that repurpose classifiers or localizers to perform detection, YOLO frames object detection as a single regression problem, straight from image pixels to bounding box coordinates and class probabilities. This approach allows YOLO to achieve high accuracy and speed, making it suitable for real-time applications. We trained it using a custom dataset, which includes images of a handful of different animals and their corresponding labels (Dangerous or Not Dangerous). For the purpose of this experiment, 7 animals were included: buffalo, elephant, rhino, zebra, lion, ostrich, turtle. The “dangerous” label was conferred upon the buffalo, rhino, lion, and ostrich classes.

## Modifying Model

### Modifying Animals

Additional animal image sets can be added from external sources such as https://universe.roboflow.com/search?q=lion (modify query as needed). Add each new animal class (along with index and label) to config.yaml in wildlife\_detector.zip, ensuring that the class index (the first value in the label .txt file for each instance) is set accordingly. The images should be added to the images folder within the correct category directory (train, test, or valid), and the label .txt file to the labels folder in the same directory.

### Modifying Labels

The labels are set in config.yaml within wildlife\_detector.zip, and can be reconfigured depending on the goals of the user.

## Model Process

The YOLO model divides the input image into a grid of cells. Each cell is responsible for predicting a fixed number of bounding boxes and their corresponding confidence scores. These confidence scores reflect the probability that a bounding box contains an object and the accuracy of the bounding box itself. Additionally, each bounding box prediction includes class probabilities, indicating the likelihood of the object belonging to a particular class.

One of the key innovations of YOLO is its ability to predict multiple bounding boxes and class probabilities simultaneously, which significantly speeds up the detection process. The model uses a single neural network to process the entire image, making it more efficient compared to traditional methods that require multiple passes over the image. YOLO’s architecture typically consists of convolutional layers followed by fully connected layers, enabling it to capture both spatial and contextual information. Overall, YOLO’s unique approach to object detection has made it a popular choice for various applications, including autonomous driving, surveillance, and robotics. Its balance of speed and accuracy continues to drive advancements in the field of computer vision.

Training a YOLO model involves several key steps. First, a large dataset of labeled images is required, where each image contains annotations for the objects present, including their bounding boxes and class labels, which are used to detect the animals. The model is then initialized with random weights and trained using a combination of convolutional and fully connected layers. During training, the input animal images are divided into a grid, and each grid cell predicts bounding boxes, confidence scores, and class probabilities.

The loss function used in YOLO training is a combination of localization loss to measure the accuracy of the predicted bounding boxes, confidence loss to measure the accuracy of the confidence scores, and classification loss to measure the accuracy of the predicted class probabilities. The model is trained using backpropagation and gradient descent, iteratively updating the weights to minimize the loss function. Data augmentation techniques, such as random cropping, scaling, and flipping, are often used to improve the model’s robustness and generalization.

# Results

As a result of this project, we observed that YOLO offers several advantages, including speed, a unified architecture, high accuracy, and flexibility. However, the model also has some limitations, such as trade-offs between speed and accuracy, localization errors, and a complex training process.

## Advantages

YOLO’s speed can be attributed to its use of a single neural network to process the entire image in one go, unlike traditional object detection methods that rely on a pipeline of multiple stages (e.g., region proposal, feature extraction, classification). This unified architecture approach eliminates the need for multiple passes over the image, significantly accelerating the detection process. The model achieves high accuracy by leveraging this single neural network to process the entire image, capturing both spatial and contextual information. An impressive feature is its flexibility to detect multiple objects of different classes in a single image, making it versatile for various applications.

## Limitations

Based on this project, we observed that when YOLO makes predictions for certain videos, there are localization errors where it struggles to accurately localize small animals or animals that are too close to each other, resulting in lower precision metrics (for example, in the demo, we see the model classify a group of turtles as elephants). This can be mitigated by increasing the number of epochs, at the cost of necessitating high computational resources to achieve a perfect balance of speed and accuracy. Unfortunately, with a significant increase in the number of epochs, the code was no longer able to run to completion (even when using HPC allocation).

Lastly, YOLO requires a large labeled dataset, which can be very labor-intensive when it comes to annotating bounding boxes for a large set of images, as well as indexing new data.

## Issues Overcome

### Image Annotation

The team initially planned to annotate a set of images using a data annotation tool to create label files with bounding boxes. However, we quickly realized this approach would be very time-consuming and labor-intensive. To address this challenge, we used Roboflow to download multiple pre-labeled images and labels, which served as our custom training dataset.

### Installing Dependencies

We first encountered an encoding locale error with this cell:

!pip install ultralytics PyYAML patool

To ensure a correct UTF-8 encoding, we added these blocks to run at the top:

!apt-get update  
!apt-get install -y locales

!locale-gen en\_US.UTF-8  
!update-locale LANG=en\_US.UTF-8

import os  
os.environ['LC\_ALL'] = 'C.UTF-8'  
os.environ['LANG'] = 'C.UTF-8'

After doing so, the installation was able to proceed without error.

# Individual Contributions

### Dennis

* Sourcing the initial set of animal images
* Formatting the input data according to YOLO’s requirements in a zip file
* Drafting the Jupyter notebook code
  + Data preprocessing
  + Model training
  + Testing model on videos
* Contributing to write-up

### Cole

* Finalizing the Jupyter notebook code
  + Optimizing file reading methods
  + Ensuring consistency of external dependencies
* Expanding the dataset by adding additional animal classes
* Adapting the notebook for use with the HPC allocation
* Contributing to write-up
* Creating presentation