Penguins vs Turtles,

Image Detection and Classification

**Group Name:  StableEmotion**

**Group Member:**

**z5433687 Rui Zhu**

**z5434264 Yuguan Lu**

**z5434640 Weiming Wu**

**z5437960 Chen Yang**

**z5437995 Sizhuo Long**

*Abstract— In this project, three different models, YOLOv8x, k-NN, and Deeplabv3 with a ResNet50 backbone, were implemented and evaluated for processing and classifying images of 'turtles' and 'penguins' sourced from Kaggle. For YOLOv8x and Deeplabv3, deep learning architectures were used. For the k-NN model, the Python language was used, utilizing libraries such as 'os', 'cv2', and 'tqdm'. YOLOv8x showed a high degree of accuracy (nearly 90.27%) while k-NN and Deeplabv3 lagged behind, with the k-NN accuracy at 52.78% and deeplabv3 outputting unsatisfactory results. The Deeplabv3 model produced jagged images, which may be due to an insufficient training set. The analysis highlights the superiority of the YOLOv8x model in extracting advanced features from raw image data and infers the limitations of the k-NN and Deeplabv3 models in tasks that require the processing of complex image data.*

Keywords: Deep Learning, YOLOv8x, k-NN, Deeplabv3, Image Classification, Object Detection, Python, TensorFlow, Accuracy.

# **Introduction**

Object detection is a crucial task in the field of computer vision, where various machine learning (ML) and deep learning (DL) models are employed to enhance performance. In the past, two-stage object detectors were widely popular and effective. However, with recent advancements in single-stage object detection and related algorithms, they have shown significant improvements over most two-stage object detectors.

In this task, we need to deal with the image of penguins and turtles. In our group project, we aim to explore and address the object detection and classification tasks on the "Penguins versus Turtles" dataset. This dataset consists of 500 training images and 72 validation images, each containing either a penguin or a turtle, with their positions annotated in the corresponding annotation files.

Furthermore, the emergence of YOLOs (You Only Look Once) has led to remarkable applications of YOLOs in object detection and recognition, outperforming their two-stage counterparts. This motivation drives us to use YOLOs for object detection. And due to the dataset is a small one, we determine to use KNN as an extra method to complete classification task.

# **Literature Review**

## YOLO

YOLO algorithm is a real-time object detection algorithm based on deep learning. Compared to traditional object detection algorithms, YOLO has the key advantage of being highly efficient and fast, enabling simultaneous prediction of object classes and spatial locations in a single forward pass.

The development of Yolo is as follows:YOLO v1: The YOLO algorithm was first proposed in 2015. It divides the input image into a grid and makes predictions within each grid cell, enabling real-time object detection. However, it has certain limitations in detecting small objects and accurately localizing complex-shaped objects.

YOLO v2: YOLO v2 was released in 2016 as an improvement over the original version. It introduced the concept of anchor boxes to enhance object detection for different scales and aspect ratios. This version improved feature extraction by adding more convolutional layers and achieved better localization performance.

YOLO v3: YOLO v3 was further improved upon the previous version and released in 2018. It introduced the Feature Pyramid Network (FPN) to fuse multi-scale features for better detection of objects at different scales. YOLO v3 also increased the number of anchor boxes and incorporated techniques such as multi-label prediction and dynamic anchor box assignment[4].

YOLO v4: YOLO v4, released in 2020, brought significant breakthroughs to the algorithm. It introduced techniques such as the CSPDarknet53 backbone network, PANet, and various optimization strategies[4]. YOLO v4 achieved the best performance in terms of accuracy, small object detection, and processing speed.

YOLO v5: YOLO v5, released in 2020, focused on improving the algorithm’s speed and accuracy. Compared to the previous version, it introduced a lightweight architecture and adopted technologies such as the EfficientDet-D7 backbone network, focal loss, and various data augmentation methods. YOLO v5 maintains high performance while being suitable for deployment on resource-limited devices.

YOLO v6：YOLO v6 supports the complete industrial application requirements of model training, inference, and multi-platform deployment. It has undergone several improvements and optimizations at the algorithmic level, including network architecture and training strategies. On the COCO dataset, YOLO v6 outperforms other algorithms of similar scale in terms of both accuracy and speed.

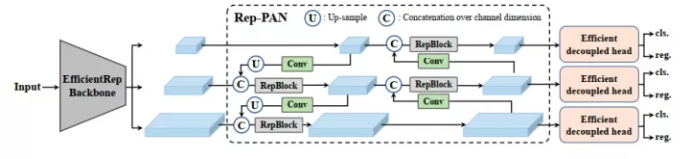


Figure1 The YOLO v6 framework (N and S are shown).

YOLO v7：YOLO v7 was released in 2022.The official version of YOLO v7 achieves higher accuracy and a 120% faster speed (in terms of FPS) compared to YOLO v5 with the same model size. It is 180% faster than YOLOX, 1200% faster than Dual-Swin-T, 550% faster than ConvNext, and 500% faster than SWIN-L. YOLO v7 surpasses all known detectors in terms of both speed and accuracy within the range of 5FPS to 160FPS. Tested on GPU V100, a model with an accuracy of 56.8% AP can achieve a detection rate of over 30 FPS (batch=1). Currently, it is the only detector capable of exceeding 30 FPS at such high accuracy.

YOLO v8：According to the official description, YOLO v8 is described as a state-of-the-art model that builds upon the success of previous YOLO versions and introduces new features and improvements to enhance performance and flexibility further. The specific innovations include a new backbone network, a new anchor-free detection head, and a new loss function. YOLO v8 is designed to run on various hardware platforms, ranging from CPUs to GPUs. However, Ultralytics has not directly named the open-source library as YOLO v8, but instead, it is referred to as “Ultralytics”. This is because Ultralytics positions the library as an algorithm framework rather than a specific algorithm. One notable characteristic is its scalability. The goal of the library is not only to support YOLO series models but also a wide range of tasks such as non-YOLO models, classification, segmentation, and pose estimation.

Two main advantages of the Ultralytics open-source library are:

1.Integration of Current SOTA Technologies

2.Support for Other YOLO Series and Beyond

Since its introduction, the YOLO algorithm has continuously evolved and become a foundational component for real-time object detection. Researchers and developers continue to explore advanced techniques and modifications to enhance the robustness, accuracy, and efficiency of the YOLO algorithm.

## KNN

Nearest Neighbors (KNN) algorithm is a commonly used supervised learning algorithm for classification and regression problems. Its basic principle involves calculating the distances between a test sample and the samples in the training set. It identifies the K nearest training samples and then determines the class or prediction value of the test sample based on the labels of these K samples through voting or averaging[6].

KNN has many advantages.

Simple and easy to understand: The KNN algorithm is relatively simple and does not require many assumptions or tuning.

Non-parametric method: KNN does not make any assumptions about the data distribution, making it suitable for various types of data.

Lazy learning: KNN is a lazy learning method, which means it stores all the training samples in memory and performs calculations only during prediction, without an explicit training process.

Suitable for multi-class problems: KNN can handle multi-class classification problems by using a voting mechanism to determine the final class.

However, KNN also has some drawbacks.

Sensitivity to outliers: KNN is sensitive to outliers because it makes judgments based on the nearest neighbor samples[7].. If there are outliers among the neighbors, it may result in incorrect predictions.

High computational complexity: KNN has a relatively high computational cost on large datasets as it requires calculating distances between samples.

## DeepLabv3

DeepLabV3 is a deep learning-based algorithm for image semantic segmentation, aiming to assign each pixel in the input image to a specific semantic class, achieving pixel-level image segmentation and understanding.

DeepLabV3 employs various techniques at the pixel-level classification to improve performance. Firstly, it utilizes Dilated Convolution to enlarge the effective size of the receptive field, capturing more contextual information. This technique increases the model’s effective receptive field without introducing additional computational parameters, leading to a better understanding of the semantic information of surrounding pixels[9].

Secondly, DeepLabV3 incorporates Multi-Scale Atrous Pooling into the network architecture. This technique leverages dilated convolutions at different scales to obtain multi-scale contextual information and fuses these features using pyramid pooling, effectively capturing details and contextual information in the image.

Additionally, DeepLabV3 employs the Atrous Spatial Pyramid Pooling (ASPP) module, which combines features with different receptive field sizes and global average pooling to capture semantic information at different scales.

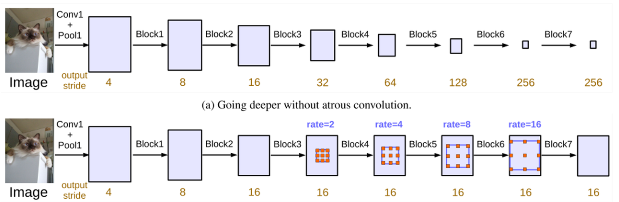


Figure12 DCNN framework

DeeplabV3 has many features.

High-precision semantic segmentation: DeepLabV3 achieves high-precision image semantic segmentation at the pixel level. It accurately assigns each pixel to a specific semantic class, resulting in excellent segmentation results in terms of details and boundaries.

Multi-scale contextual information: DeepLabV3 combines various techniques to capture multi-scale contextual information in images. By using dilated convolutions and multi-scale atrous pooling, DeepLabV3 captures context information at different receptive fields and scales, thereby better understanding the details and semantic information in the image segmentation task.

Efficient model design: DeepLabV3 incorporates a series of effective techniques, such as dilated convolutions and pyramid pooling, to improve performance while reducing computational complexity. This enables DeepLabV3 to maintain high accuracy while having higher computational efficiency.

Strong performance and robustness: DeepLabV3 exhibits excellent performance and robustness on multiple public datasets. It can handle images with different backgrounds, lighting conditions, and scales, accurately segmenting semantic information from them[10].

# **Methods**

## Yolo

The YOLO algorithm transforms the object detection problem into a regression problem by partitioning the entire image into a grid and utilizing a convolutional neural network (CNN) for feature extraction. Each grid cell is responsible for predicting the presence of objects in the image. For each grid cell, the algorithm predicts the bounding box coordinates and class label of the object, along with a corresponding confidence score[2]. Finally, non-maximum suppression (NMS) is applied to filter out the most reliable detection results.

The basic steps of the YOLO algorithm are as follows:

1.Input Processing: The input image is divided into fixed-sized grids. Each grid is responsible for detecting objects in the image. Additionally, the image is often resized to a size that can be handled by the model.

2.Feature Extraction: A convolutional neural network (CNN) is used to extract features from the image. Typically, a pre-trained CNN model such as Darknet is used to extract feature maps at different scales of the image.

3.Object Detection: In each grid cell, objects are detected by predicting the position and class of bounding boxes. Each bounding box consists of the object’s location (x, y coordinates) and dimensions (width and height), along with a confidence score indicating the presence of the object.

4.Non-Maximum Suppression (NMS): To reduce redundant detections and improve detection quality, a non-maximum suppression technique is employed to filter and prune different bounding boxes, retaining only the most confident detections.

5.Output Prediction: Finally, the algorithm outputs the detected objects along with their corresponding confidence scores, typically presented in the form of bounding boxes and class labels.

The characteristics of Yolo are as follows:

Real-time Processing: The design philosophy of the YOLO algorithm aims to achieve real-time processing speed while maintaining relatively high detection accuracy. By simplifying the object detection task as a regression problem and performing detection in a single forward pass, YOLO significantly improves detection efficiency.

Global Context: YOLO performs feature extraction and object detection at the whole image level, capturing global contextual information that helps enhance detection accuracy. Compared to traditional methods that detect objects at different scales and locations, YOLO exhibits better global perception capability.

Multi-scale Detection: YOLO utilizes feature maps of different scales for object detection, effectively handling objects of various sizes and predicting different bounding boxes at different scales. This enables better handling of objects with significant scale variations[3].

## KNN

# Basic steps：

1.Distance Calculation: Calculate the distances between the test sample and each sample in the training set. Common distance metrics used include Euclidean distance, Manhattan distance, Minkowski distance, and others.

2.K Selection: Determine the number K of nearest neighbors to be used for voting or averaging. Choosing an appropriate value for K is important and often requires cross-validation.

3.Neighbor Determination: Select the K samples with the shortest distances as the neighbors of the test sample.

4.Voting or Averaging: For classification problems, the labels of the K neighbors are used for voting to determine the predicted class of the test sample. For regression problems, the average of the labels of the K neighbors is taken as the predicted value of the test sample.

The workflow of KNN is as follows:

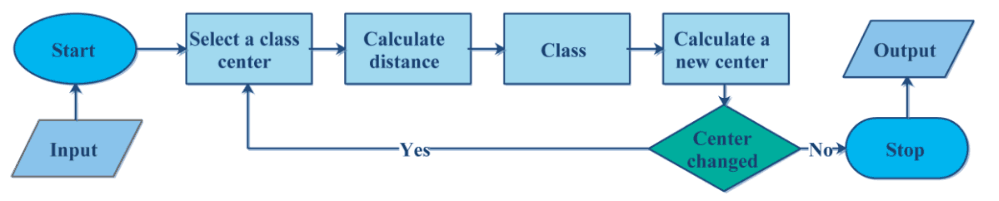


Figure2 The Steps of KNN

Cosine similarity:

In the KNN algorithm, cosine similarity is a commonly used measure of similarity that evaluates the similarity between two vectors. Cosine similarity measures the cosine value of the angle between two vectors in a multidimensional space. It can be used to compute similarity in domains such as text, images, and more.

The calculation of cosine similarity between two vectors can be done through the following steps:

1.Normalize the two vectors: To eliminate the influence of vector lengths in calculations, it is common to normalize the two vectors to unit vectors. This is achieved by dividing each vector by its length, making them both length.

2.Compute the dot product of the vectors: Multiply the normalized vectors element-wise and sum the results to obtain their dot product.

3.Calculate the cosine similarity: Divide the dot product by the product of the lengths of the vectors[11].

.

In the KNN algorithm, cosine similarity can be used to measure the similarity between samples and determine their neighbors. By calculating the cosine similarity between an unlabeled sample and the training samples, the K most similar samples can be identified. The class labels of these neighbors can then be used for voting, possibly with weighting, to determine the class label of the unlabeled sample.

## DeepLabv3

Medical image segmentation: DeepLabV3 has extensive applications in medical image segmentation tasks. It can be used to segment different tissue structures, lesions, or organs from medical images, such as tumor segmentation, vascular segmentation, etc. This is crucial for medical analysis, assisting diagnosis, and treatment decision-making.

Autonomous driving: DeepLabV3 can be applied in scene understanding and object recognition for autonomous driving. By accurately segmenting road scenes, pedestrians, vehicles, and obstacles, it provides a deeper and more detailed understanding of the environment, aiding autonomous vehicles in making accurate decisions and planning.

Geographical information systems (GIS) and land cover classification: DeepLabV3 can be employed in land cover classification and GIS applications for semantic segmentation of satellite images, aerial imagery, or satellite remote sensing data. It helps extract and analyze land surface features such as roads, buildings, water bodies, forests, etc., which are useful in urban planning, environmental monitoring, and resource management.

Visual scene understanding and intelligent video analysis: DeepLabV3 can be used in visual scene understanding and intelligent video analysis tasks. By performing real-time semantic segmentation on video frames, it aids in understanding different objects and scenes in videos, making it applicable in video surveillance, behavior recognition, human-computer interaction, and related fields[5].

Apart from these domains, DeepLabV3 can also be applied in various other tasks involving image segmentation, such as virtual and augmented reality, image editing, and human pose estimation. Its flexibility and accuracy make it highly valuable in the fields of computer vision and image processing.

# **Experimental Results**

## Experimental Setup

# Yolov8x：

The infrastructure used includes a mix of personal hardware and server-side resources. The local computing facility is a Lenovo R7000P with an AMD Ryzen 5 4800H processor and an NVIDIA RTX 2060 GPU with 6GB of video memory.

For more intensive computations, a GPU server supplied by AutoDL was used. Powered by an AMD EPYC 7T83 64-core processor with an NVIDIA RTX 4090 GPU, it has 24GB of video memory and is capable of handling heavy-duty tasks and large-scale image processing jobs. In addition, the server has 90GB of RAM.

On the software side, Jupyter Lab was used as the main development environment. The image recognition model was developed using TensorFlow 2.9.0. The system runs on Python 3.8. In addition, CUDA 11.2 was used for GPU-accelerated processing.

# KNN：

The k-NN algorithm is implemented using the Python programming language, utilizing libraries such as 'os', 'cv2', and 'tqdm'.

## dataset:

## Yolov8x:

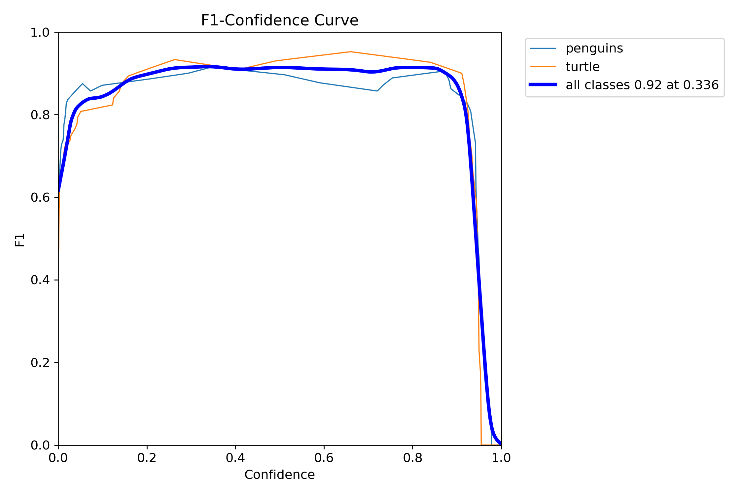
The dataset used was reformatted from Roboflow and contains a total of 572 images, of which 450 were used for training, 50 for validation and 72 for testing. The data enhancement techniques applied include saturation (between -25% and +25%) and blurring (up to 2.5 pixels).

# KNN：

The training set contains 71 images which are trained based on images cropped from the original background. There are two categories i.e. turtles and penguins. The RGBT channels of the images are compressed into a one-dimensional array to compute the cosine similarity, where the T channel represents the transparency channel.

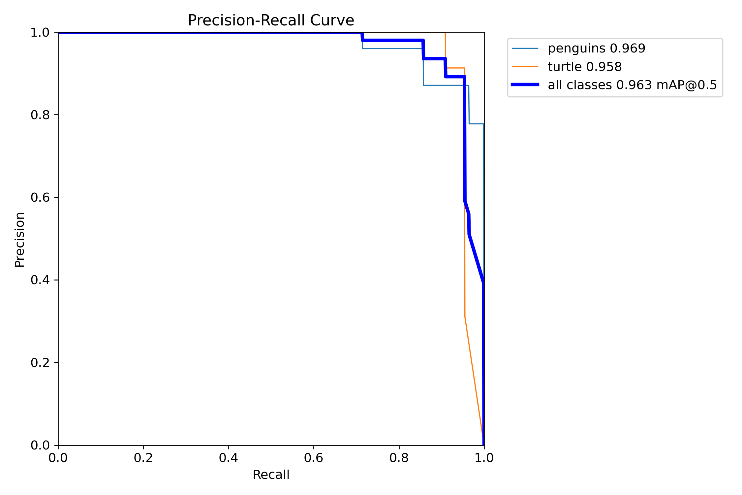
## Evaluation Metrics:

## Yolov8x：



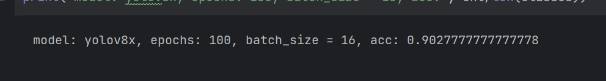
## Figure3 F1

The graph indicates that, across the entire confidence interval, the F1 score remains above 0.8, maintaining a high level.



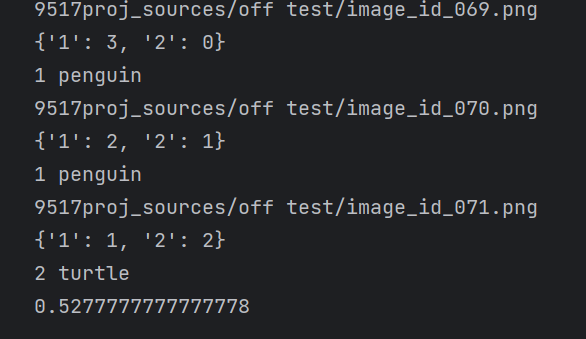
## Figure4 Precision and Recall

It is obvious that the AUC is almost close to 1, which indicates a good performance of it.



Accuracy is approximately 90.27%.

# KNN：



The model achieved an accuracy of approximately 52.78%.

## Model Architecture:

## Yolo:

The model used for the experiments is YOLOv8x, which contains 365 layers with approximately 68.23 million parameters. It uses a unique architecture that contains several layers of convolution (Conv), upsampling (Upsample), and merging (Concat). Among the core modules utilized are Conv, which is a convolutional layer that applies a specified number of filters to the input, and block.C2f, which is a custom module in the Ultralytics framework.

# KNN：

The core of the algorithm revolves around the prediction phase. In this phase, the algorithm looks for the k instances in the training set that are most similar to the new sample. The label of the new sample is then determined by utilizing a voting procedure that takes into account the labels of these closely related samples.

## Training Process:

# Yolov8x:

The model configuration was Ultralytics YOLOv8.0.143 running on Python-3.8.10 and torch-2.0.1+cu117. The model was trained on an NVIDIA GeForce RTX 4090 GPU with 24,217 MiB of memory. The training configuration consisted of 100 cycles with a batch size of 16, an image size of 640, and a patience of 50. automatic mixing precision (AMP) was enabled to speed up training while maintaining the accuracy of the model.

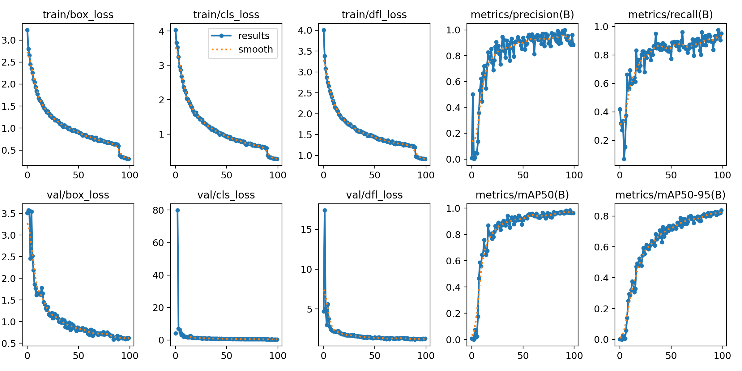
Based on the model's hesitation in predicting the outcome (probability of having both penguins and turtles in the image), the highest probability was selected as the outcome output.

# KNN：

Storing training data.

## Experimental Results:

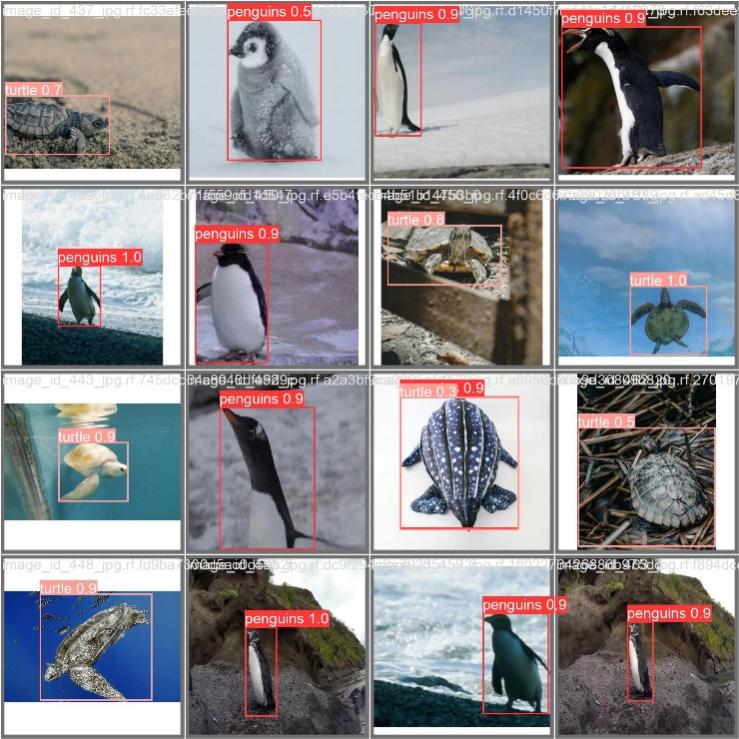
## Yolov8x:



## Figure5 Precision and Recall

The sequence of five diagrams in the first row of the aforementioned graph clearly shows the model's strong performance during its training phase. There is a consistent downward trend in the error rates associated with bounding box detection and categorization, eventually reaching minimal levels. Concurrently, the model's predictive accuracy demonstrated an increasing trajectory.

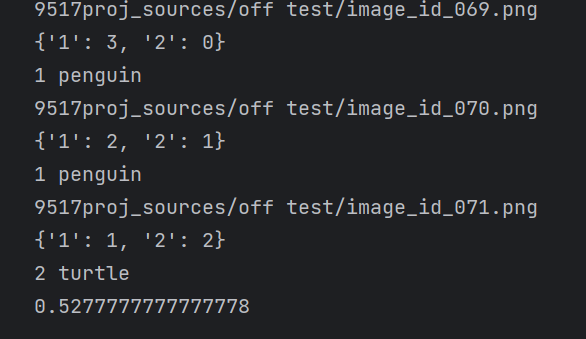
In the second row of diagrams, the model's performance on the validation set is displayed. The trend in error rates for bounding box detection continues to descend, while the classification error rate remains impressively low from the outset. The predictive accuracy of the model during the validation phase is consistent with that observed during training.



## Figure7 results after traning

This pertains to a subset of the validation dataset. It is evident that the accuracy for the majority of the instances is exceeding 0.9. Nonetheless, for a few images that pose higher recognition difficulty, the accuracy dips to around 0.5.

# KNN：



In the set of 72 test images, the model predicted 38 images to be 'penguins' (category 1) and 34 images to be 'turtles' (category 2).

## Analysis of Failure Cases

## Deeplabv3\_resnet50:



## Figure8 results after traning

The initial model employed was Deeplabv3, equipped with ResNet50 as a backbone. However, the performance of this model fell short of expectations. The output consisted of images exhibiting jagged characteristics, which were deemed unsatisfactory for our purposes. This subpar performance may be attributable to an inadequacy in the training set.

# **Discussion**

# KNN：

The model shows balanced predictions, predicting almost equal distributions of "penguins" and "turtles". The problems encountered may be attributed to the loss of accuracy during the resizing process and the calculation of cosine similarity.

In particular, the limited performance may stem from the inadequate representation of all possible variations of the "penguin" and "turtle" images in the training set. In addition, the model may not be able to handle changes in image factors such as illumination and angle well.

## Yolov8x:

YOLOv8x, a deep learning-based model, is more capable of extracting advanced features from raw image data due to its deep architecture and sophisticated learning algorithms. This makes it very effective for target detection tasks, including image classification. Its superior performance can be attributed to its ability to learn and generalize from a wide range of data to better adapt to changes in image characteristics such as illumination, orientation and scale.

# **Conclusion**

The project used datasets consisting of "turtle" and "penguin" images to perform a detailed comparison and analysis of YOLOv8x, k-NN, and Deeplabv3 models equipped with the ResNet50 backbone. Each model is implemented using a unique infrastructure with different optimizations, reflecting the uniqueness of each approach.

The k-NN model, while simple and intuitive, performed mediocrely with an average accuracy of 52.78%. This study reveals the limitations of the model, pointing to possible deficiencies in dealing with the complexity of image data and the diversity of training sets. This emphasizes the need for more representative training sets and possibly further improvements in model design or feature engineering strategies.

Deeplabv3 equipped with the ResNet50 backbone was another model tested. However, it performed poorly with jagged output images. This poor performance could be attributed to an insufficient training set, demonstrating the importance of a diverse dataset needed to train the full range of complex models.

In contrast, the YOLOv8x model shows significantly superior performance with an accuracy of nearly 90.27%. Despite initial challenges during the implementation phase, the model's performance was significantly improved through modifications and optimizations. Due to its deep architecture and sophisticated learning algorithms, it effectively extracted high-level features from the raw image data, thus proving its proficiency in target detection tasks, including image classification.

Overall, while simpler models may provide baseline performance, when implemented and fine-tuned correctly, more complex architectures, such as deep learning models, can provide superior performance on a variety of tasks and datasets.

# **References**

1. Kaggle. Penguins Versus Turtles: Image Classification With Bounding Boxes. 2023. <https://www.kaggle.com/datasets/abbymorgan/penguins-vs-turtles>
2. Redmon, Joseph, et al. “You Only Look Once: Unified, Real-Time Object Detection.” Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR). 2016.
3. Redmon, Joseph, and Ali Farhadi. “YOLO9000: better, faster, stronger.” Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR). 2017.
4. Redmon, Joseph, and Ali Farhadi. “YOLOv3: An Incremental Improvement.” arXiv preprint arXiv:1804.02767 (2018).
5. Bochkovskiy, Alexey, et al. “YOLOv4: Optimal Speed and Accuracy of Object Detection.” arXiv preprint arXiv:2004.10934 (2020).
6. Zhang, T. (2004). Solving large scale linear prediction problems using stochastic gradient descent algorithms. In Proceedings of the Twenty-First International Conference on Machine Learning (ICML) (pp. 116-123).
7. Wilson, D. R.& Martinez, T. R. (1997). Improved heterogeneous distance functions. Journal of Artificial Intelligence Research, 6, 1-34.
8. Shiliang Sun.&Rongqing Huang.(2010).An Adaptive k-Nearest Neighbor Algorithm.East China Normal University.
9. Chen, L. C.Papandreou, G. Schroff, F.& Adam, H. (2018). Rethinking atrous convolution for semantic image segmentation. arXiv preprint arXiv:1706.05587.
10. Chen, L. C.Zhu, Y.& Yuille, A. L. (2018). Deeplab: Semantic image segmentation with deep convolutional nets, atrous convolution, and fully connected crfs (vol 32, pg 178-373, 2018). IEEE transactions on pattern analysis and machine intelligence, 41(2), 498-499.
11. Zhou, Z.Siddiquee, M. M. R., Tajbakhsh, N.& Liang, J. (2017). Unet++: A nested u-net architecture for medical image segmentation. In Deep Learning in Medical Image Analysis and Multimodal Learning for Clinical Decision Support (pp. 3-11). Springer.
12. Yang, J.Price, B., Cohen, S.& Lee, H. (2018). Effective deep learning-based multi-view ensemble for breast cancer diagnosis. In International Conference on Medical Image Computing and Computer-Assisted Intervention (pp. 484-492). Springer.