Penguins vs Turtles,

Image Detection and Classification

**Group Name:**

**Group Member:**

*Abstract*—In this task, we use Yolo and KNN to process the Images. We used the dataset from Kaggle’s Competition to conduct the traning. After the 对于A，我们使用了什么，对于KNN，我们使用了什么。最后我们选择的评估参数是 ，最后得到的结果分别是，我们得到的结论是，并且给出了合理的解释和分析。

Keywords: Yolo, collaborative filtering algorithm, KNN

# **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.

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. 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.

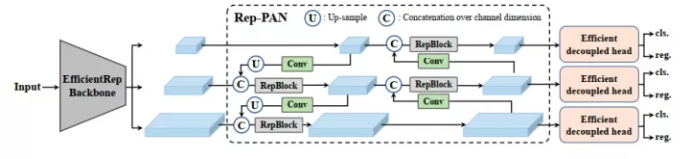


Figure1 The YOLO v6 framework (N and S are shown).

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

1. 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.

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. 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.

# **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. 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.

## 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:

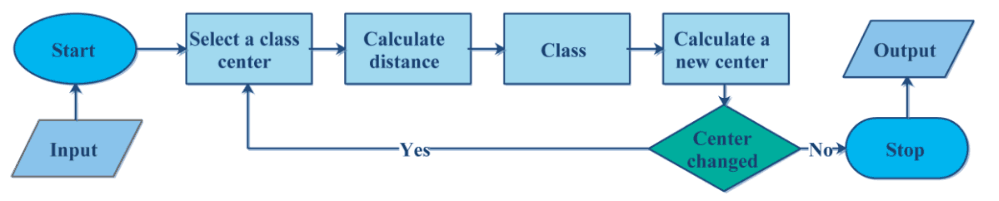


Figure2 The Steps of KNN

# 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.

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.

# **Experimental Results**

## Experimental Setup

# Yolov8x：

1.The infrastructure used comprises a mixture of personal hardware and server-side resources. The local computing infrastructure utilized was a Lenovo R7000P, equipped with an AMD Ryzen 5 4800H processor and an NVIDIA RTX 2060 GPU with 6GB of video memory.

2.For more intensive computations, we made use of a GPU server provided by AutoDL. This server was powered by an AMD EPYC 7T83 64-Core Processor, providing a substantial number of virtual CPUs for parallel computation. The server was outfitted with a powerful NVIDIA RTX 4090 GPU, boasting 24GB of video memory, capable of handling heavy-duty tasks and large-scale image processing jobs. Furthermore, the server had 90GB of RAM.

3.On the software side, we utilized Jupyter Lab as our primary development environment due to its user-friendly interface and robust support for data science workflows. Our image recognition models were developed using TensorFlow 2.9.0, a leading deep learning framework, which offers high flexibility and performance. Our system was running on Python 3.8, taking advantage of its extensive library support and simplicity for machine learning tasks. Moreover, we used CUDA 11.2 to allow 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:

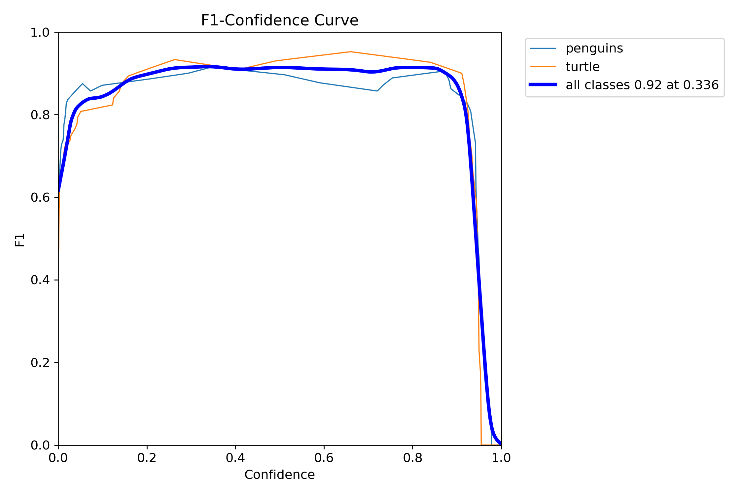
The dataset used in the experiment was reformatted from Roboflow and composed of a total of 572 images divided into 450 for training, 50 for validation, and 72 for testing. Data augmentation techniques applied include saturation (between -25% and +25%) and blur (up to 2.5px). This augmentation strategy helped to ensure the diversity and robustness of the data, improving the model's generalization capabilities.

# KNN：

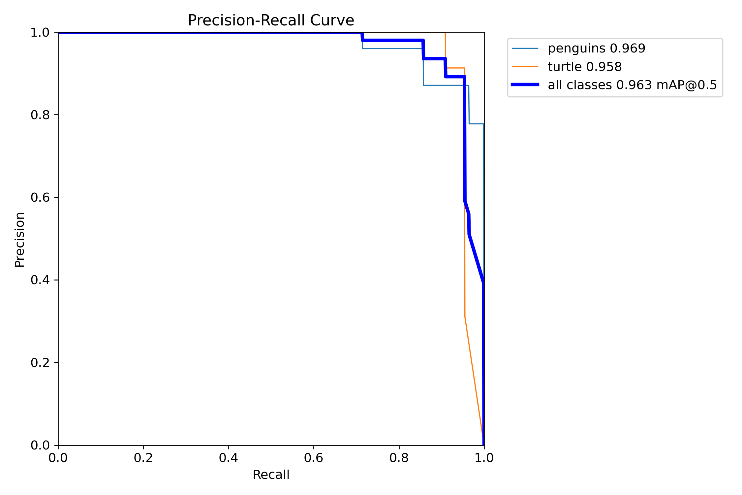
The training set consists of 71 images, which are trained based on images cropped from their original backgrounds. There are two classes, namely turtles and penguins. The RGBT channels of the images are compressed into one-dimensional arrays to calculate the cosine similarity, where the T channel represents the transparency channel.

## Evaluation Metrics:

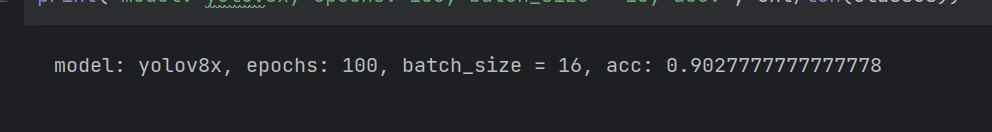
## Yolov8x:



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

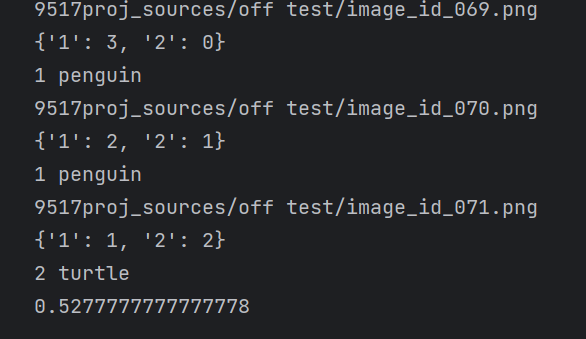


It is obvious that the AUC is almost close to 1, which indicates a good performance of it.



Accuracy is approximately 90.27%.

# KNN：



The model achieved an accuracy of approximately 52.78%.

## Model Architecture:

## Yolo:

The model used for the experiment was YOLOv8x, which includes 365 layers and approximately 68.23 million parameters. It uses a unique architecture that comprises several convolutional (Conv), up-sampling (Upsample), and concatenate (Concat) layers, among others. The core modules utilized include Conv, which is a convolutional layer that applies a specified number of filters to the input, and block.C2f, a custom block module in the Ultralytics framework.

# KNN：

The crux of the algorithm is centered around the prediction phase. During this stage, the algorithm seeks the k instances within the training set that are most similar to the new sample. The label for the new sample is subsequently determined by leveraging a voting procedure that takes into account the labels of these closely associated samples.

## Training Process:

## Yolov8x:

The model configuration was set with 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 a memory of 24,217 MiB. The training configuration included 100 epochs with a batch size of 16, an image size of 640, and a patience of 50. Automatic Mixed Precision (AMP) was enabled to speed up training while maintaining the accuracy of the model.

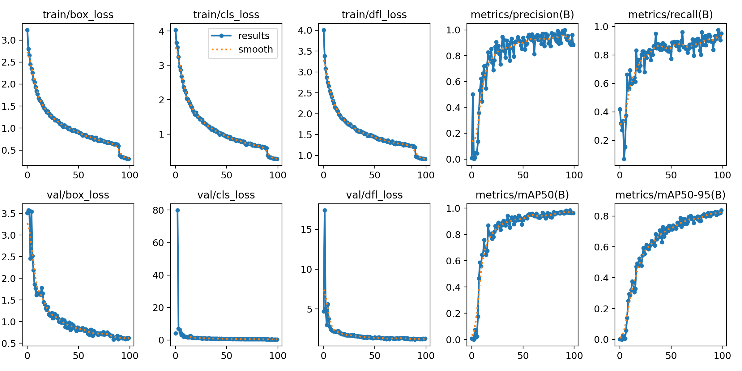
According to the situation that the model hesitated to predict the outcome (the probability of a penguin and the probability of a turtle appearing in a picture at the same time), we chose the one with the highest probability as the outcome output.

# KNN：

Storing training data.

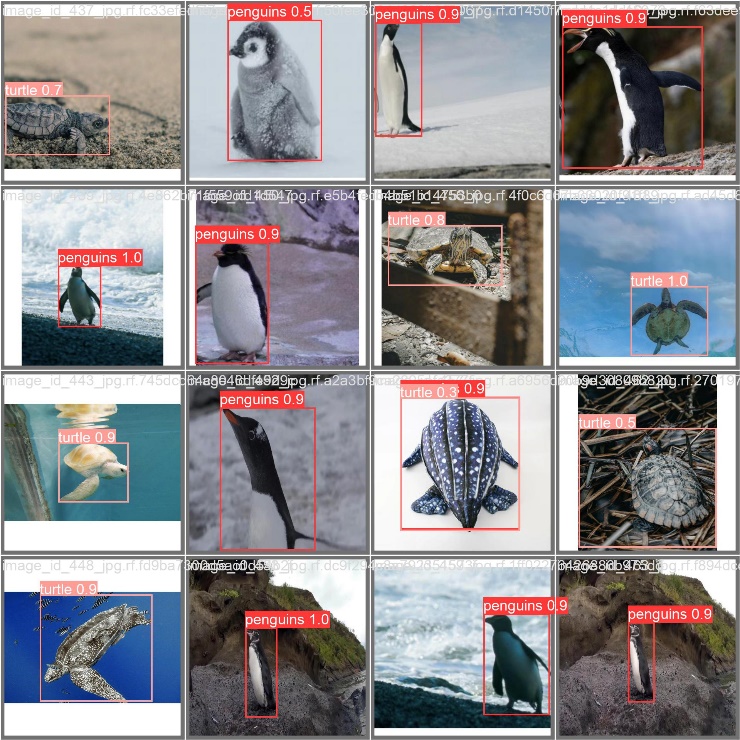
## Experimental Results:

## Yolov8x:



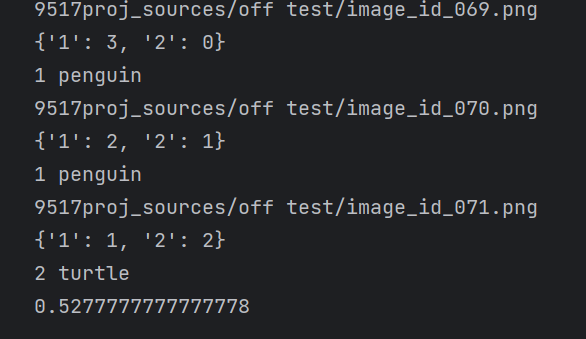
The sequence of five diagrams in the first row of the aforementioned graph clearly shows the model's strong performance during its training phase. There is a consistent downward trend in the error rates associated with bounding box detection and categorization, eventually reaching minimal levels. Concurrently,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



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:



In our research, we experimented with two different segmentation models. The initial model we 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 with nearly equal distribution in predicting 'penguins' and 'turtles'. However, the relatively low accuracy indicates room for improvement. The issues encountered may be attributed to loss of precision during the resizing process and the computation of cosine similarity.

In particular, limitations may arise from the insufficient representation of all possible variations of 'penguin' and 'turtle' images in the training set. Furthermore, the model might not handle well with variations in image factors such as illumination and angle.

Potential steps for future work to improve the model's performance may include exploring more complex models like deep learning models, which are capable of learning more sophisticated features from raw image data, altering the feature representation, or other feature engineering tactics. Increasing the size of the training set to capture a wider range of sample variations may also be beneficial.

## Yolov8x:

Our k-NN model demonstrated mediocre performance with an average accuracy of 53%, irrespective of the number of neighbors (k) chosen for the classification. This relatively low accuracy might be attributed to the limitations of the model in handling the complexity of image data, and the potential loss of precision during the image resizing process and the computation of cosine similarity. Furthermore, the k-NN model's performance is heavily influenced by the size and quality of the training data, which might not have fully captured the variability in 'penguin' and 'turtle' images.

On the other hand, YOLOv8x, a deep learning-based model, is more capable of extracting higher-level features from raw image data due to its deep architecture and sophisticated learning algorithms. This makes it highly effective for object detection tasks, including image classification. Its superior performance can be attributed to its ability to learn and generalize from a broad range of data, thus better accommodating variations in image characteristics such as illumination, orientation, and scale.

# **Conclusion**

Although our handcrafted k-NN model exhibits balanced predictions between the two classes, it falls short in comparison to the performance and efficiency of the YOLOv8x model. The k-NN model's simplicity and interpretability do not compensate for its lack of accuracy and efficiency in image classification tasks. These findings suggest that for complex tasks like image classification, more advanced models such as YOLOv8x that leverage deep learning algorithms might be more suitable. Moreover, YOLOv8x is designed to perform real-time object detection, making it highly efficient in processing large volumes of images.

# **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.