Karunya INSTITUTE OF TECHNOLOGY AND SCIENCES
(Declared as Deemed to be University under Sec.3 of the UGC Act, 1956)
MoE, UGC & AICTE Approved
NAAC A++ Accredited

Division of Electronics and Communication Engineering
2023-2024 (EVEN SEM)

III IA EVALUATION REPORT

for

DIGITAL SIGNAL PROCESSING-PROJECT BASED COURSE

Title of the project:

IMAGE PROCESSING USING YOLOv8

A report submitted by

Name of the Student	***ALDRIN G***
Register Number	***URK22EC1019***
Subject Name	***DIGITAL SIGNAL PROCESSING***
Subject Code	***18EC2015***
Date of Report submission	***04/04/24***

Project Rubrics for Evaluation

First Review: Project title selection - PPT should have four slides (Title page, Introduction, Circuit/Block Diagram, and Description of Project).

Second Review: PPT should have three slides (Description ofConcept, implementation, outputs, results and discussion)

Rubrics for project (III IA - 40 Marks):
Content - 4 marks (based on Project)
Clarity - 3 marks (based on viva during presentation)
Feasibility - 3 marks (based on project)
Presentation - 10 marks
Project Report - 10 marks
On-time submission - 5 marks (before the due date)
Online submission-GCR - 5 marks

Total marks:_____/ 40 Marks

Signature of Faculty with date:

TABLE OF CONTENTS

CHAPTER 1

INTRODUCTION

Image processing, a fundamental aspect of computer vision, involves manipulating digital images to extract meaningful information or enhance visual quality. Digital Signal Processing (DSP) plays a crucial role in image processing, offering a diverse toolkit of mathematical operations and algorithms for analyzing and transforming digital signals, including images. DSP techniques, such as filtering, convolution, Fourier transforms, and correlation, are employed to preprocess, enhance, or analyze digital images.

The integration of DSP techniques with advanced neural network architectures like YOLOv8 has opened up new possibilities for improving the efficiency and accuracy of image processing tasks, especially in object detection. YOLOv8, a cutting-edge Convolutional Neural Network (CNN) architecture designed for object detection, offers real-time efficiency and high accuracy.

In this integration, DSP methods are utilized to preprocess images before they are fed into the YOLOv8 network. These preprocessing steps typically involve operations such as noise reduction, edge enhancement, contrast adjustment, or other techniques aimed at improving the quality of input images and enhancing the discriminative features relevant for object detection.

Overall, the integration of DSP techniques with YOLOv8 enhances the capabilities of object detection systems by improving the quality of input data and augmenting the network's ability to detect and localize objects accurately within images.

CHAPTER 2

DESCRIPTION OF THE PROJECT

Object Detection using YOLOv8:

Object detection is a critical task in computer vision, enabling machines to identify and locate objects within images or videos. YOLOv8, a state-of-the-art Convolutional Neural Network (CNN) architecture, is renowned for its real-time efficiency and high accuracy in object detection. This project aims to develop and implement an object detection system using YOLOv8, catering to diverse applications such as surveillance, autonomous vehicles, and industrial automation.

1) **Data Collection & Dataset preparation:**

The project begins with data collection and preparation, where a comprehensive dataset containing annotated images of various objects is curated. This dataset serves as the foundation for training and evaluating the YOLOv8 model. The annotations include bounding boxes delineating the location and class label of each object within the images, facilitating supervised learning for object detection.

2) **Training the Dataset:**

Once the dataset is prepared, the YOLOv8 model architecture is configured and trained using the collected data. During training, the model learns to detect objects by iteratively optimizing its parameters based on a predefined loss function. The training process involves multiple epochs, with the model gradually improving its ability to accurately localize and classify objects within images.

3) **Optimization:**

To enhance the performance of the YOLOv8 model, various optimization techniques are employed. These techniques include data augmentation, regularization, and hyperparameter tuning, aimed at mitigating overfitting, improving generalization, and enhancing the robustness of the model to diverse environmental conditions.

4) **Analysis & Evaluation:**

After training, the performance of the YOLOv8 model is evaluated using a separate validation dataset. Evaluation metrics such as mean Average Precision (mAP) are computed to assess the model's accuracy and effectiveness in detecting objects across different classes. The evaluation results provide insights into the strengths and limitations of the trained model, guiding further optimization efforts.

5) **Deployment:**

Once the YOLOv8 model demonstrates satisfactory performance during evaluation, it is deployed for real-world object detection tasks. The deployed model processes input images or video streams in real-time, accurately detecting and localizing objects within the scene. The object detection system offers versatility and scalability, capable of adapting to dynamic environments and accommodating various use cases.

6) **Continuous Updation:**

Throughout the project lifecycle, rigorous testing and validation procedures are conducted to ensure the reliability and effectiveness of the object detection system. Continuous monitoring and refinement of the model contribute to its ongoing improvement and adaptation to evolving requirements and challenges.

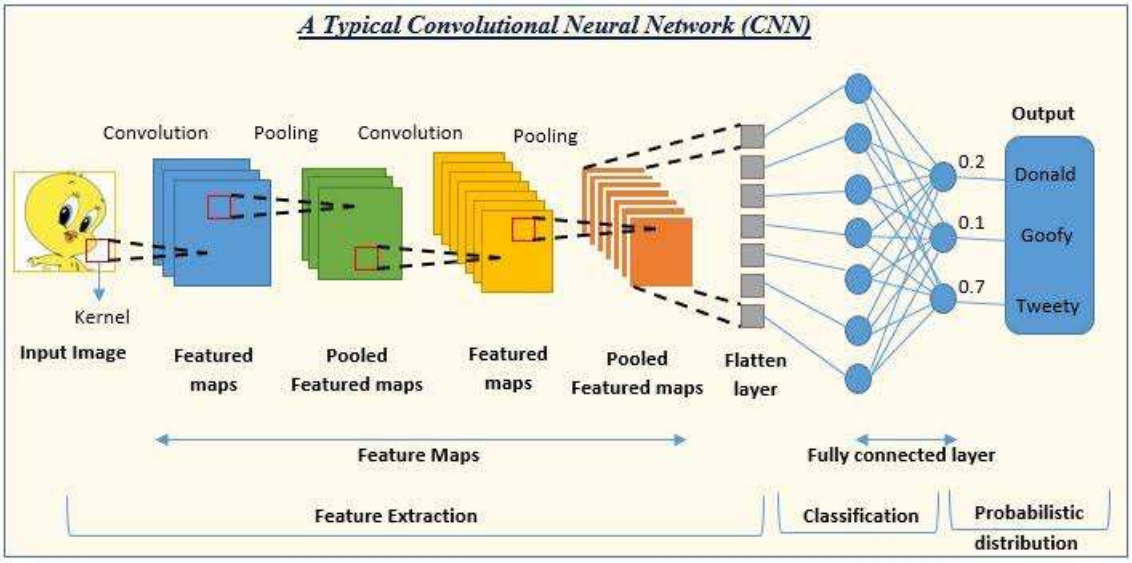
CHAPTER 3

CONCEPT INVOLVED

Convolution Neural Networks (CNN):

Convolutional Neural Networks (CNNs) are a class of deep neural networks that are particularly effective for tasks involving images, such as image classification, object detection, and image segmentation. They are inspired by the organization and functionality of the visual cortex in animals, where neurons are arranged in layers and respond to stimuli in a hierarchical manner.

Here's an overview of the key components and concepts of Convolutional Neural Networks:



1. **Convolutional Layers:** The core building blocks of CNNs are convolutional layers. These layers apply convolution operations to input data using learnable filters (also known as kernels or feature detectors). Convolution involves sliding the filters across the input data and computing the dot product at each position. This process extracts features from the input, such as edges, textures, or shapes.

2. **Activation Functions:** After convolution, an activation function is applied to introduce non-linearity into the network. Common activation functions include ReLU (Rectified Linear Unit), which sets negative values to zero and preserves positive values, and sigmoid or tanh functions.

3. **Pooling Layers:** Pooling layers are used to downsample the feature maps obtained from convolutional layers. Max pooling and average pooling are commonly used techniques, where the maximum or average value within a region of the feature map is retained, respectively. Pooling helps reduce the spatial dimensions of the feature maps while preserving important features, aiding in computational efficiency and reducing overfitting.

4. **Fully Connected Layers:** After several convolutional and pooling layers, the feature maps are flattened into a vector and passed through one or more fully connected layers. These layers connect every neuron in one layer to every neuron in the next layer, enabling the network to learn complex patterns and relationships in the data. Fully connected layers are typically followed by a softmax activation function in classification tasks to produce probability scores for each class.

5. **Loss Function:** CNNs are trained using supervised learning, where the difference between the predicted output and the ground truth labels is minimized. The loss function quantifies this difference, and backpropagation is used to update the network's parameters (weights and biases) to minimize the loss during training. Common loss functions include cross-entropy loss for classification tasks and mean squared error for regression tasks.

6. **Training:** CNNs are trained using large datasets with labeled examples. The training process involves feeding input data through the network, computing the loss, and adjusting the network's parameters using optimization algorithms such as stochastic gradient descent (SGD), Adam, or RMSprop. The model's performance is evaluated on a separate validation set, and training continues until convergence or a predefined stopping criterion is met.

CNNs have revolutionized various fields such as computer vision, natural language processing, and speech recognition, achieving state-of-the-art performance in many tasks. Their ability to automatically learn hierarchical features from raw data makes them powerful tools for extracting patterns and making predictions from complex inputs like images.

YOLOv8:

YOLO (You Only Look Once) is a state-of-the-art object detection algorithm known for its speed and accuracy. It was introduced by Joseph Redmon et al. in 2016 and has since undergone several iterations, with YOLOv4 and YOLOv5 being the latest versions. The fundamental idea behind YOLO is to frame object detection as a regression problem, where a single neural network predicts bounding boxes and class probabilities for all objects in an image in a single pass.

Here's a breakdown of the key components and concepts of the YOLO algorithm:

1. **Single Neural Network:** Unlike traditional object detection methods that rely on region proposal techniques followed by classification, YOLO performs both tasks simultaneously within a single neural network. This end-to-end approach makes YOLO faster and more efficient.

2. **Grid Cell Partitioning:** YOLO divides the input image into a grid of cells, typically, say, 7x7 or 13x13 grid cells, depending on the network architecture. Each grid cell is responsible for predicting bounding boxes and class probabilities for objects whose centers fall within that cell.

3. **Bounding Box Prediction:** For each grid cell, YOLO predicts a fixed number of bounding boxes (often predetermined). These bounding boxes are represented by parameters such as center coordinates, width, height, and confidence scores. The confidence score indicates the likelihood that the bounding box contains an object and how accurate the bounding box's localization is.

4. **Class Prediction:** In addition to predicting bounding boxes, YOLO also predicts the probability distribution over all classes for each bounding box. This allows the algorithm to determine the most likely class(es) for each detected object.

5. **Anchor Boxes:** YOLO uses anchor boxes (or priors) to improve the accuracy of bounding box predictions. Anchor boxes are pre-defined shapes of various sizes and aspect ratios. During training, the network learns to adjust these anchor boxes to better fit the shapes of objects in the dataset.

6. **Loss Function:** YOLO uses a composite loss function that combines localization loss, confidence loss, and classification loss. The localization loss penalizes inaccurate bounding box predictions, the confidence loss penalizes incorrect objectness predictions, and the classification loss penalizes misclassification of object classes. These components are weighted and summed to compute the total loss, which the network aims to minimize during training.

7. **Non-Maximum Suppression (NMS):** After predicting bounding boxes and class probabilities for all objects in the image, YOLO applies non-maximum suppression to remove redundant and overlapping bounding boxes. This ensures that each object is represented by only one bounding box with the highest confidence score.

YOLO's architecture and design principles make it well-suited for real-time object detection applications where speed and accuracy are paramount. It has been widely adopted in various domains, including surveillance, autonomous vehicles, and robotics, for tasks such as pedestrian detection, vehicle detection, and general object recognition. Despite its effectiveness, YOLO continues to be improved and refined with each iteration to achieve even better performance.

CHAPTER 4

TOOLS

Python:

Python is widely used in object detection and training AI models due to its simplicity, flexibility, and extensive ecosystem of libraries and frameworks. Here's how Python is utilized in these domains:

1. **Data Preprocessing:** Python is used for data preprocessing tasks, such as loading, cleaning, and transforming datasets. Libraries like NumPy and Pandas are commonly employed for data manipulation and analysis, allowing developers to preprocess raw data into a suitable format for training AI models.

2. **Image Processing:** In object detection tasks, images are often the primary source of data. Python's libraries such as OpenCV and Pillow provide powerful tools for image manipulation, including resizing, cropping, and applying various transformations. These libraries enable developers to preprocess images and extract features that are essential for training AI models.

3. **Model Training:** Python is the language of choice for training AI models, including those used in object detection tasks. Frameworks like TensorFlow and PyTorch offer high-level APIs for building and training deep learning models. Developers can define the architecture of the neural network, specify loss functions, and configure optimization algorithms using Python code. Additionally, specialized libraries like TensorFlow Object Detection API and Detectron2 provide pre-built models and training pipelines specifically tailored for object detection tasks.

4. **Evaluation and Validation:** Python is used for evaluating and validating the performance of trained AI models. Developers can use libraries such as scikit-learn and TensorFlow/Keras to compute evaluation metrics like accuracy, precision, recall, and F1 score. These metrics help assess the model's ability to accurately detect objects in unseen data and identify areas for improvement.

5. **Deployment:** Once trained, AI models need to be deployed in production environments. Python frameworks like Flask and Django are commonly used for

building web-based applications that serve predictions generated by AI models. Additionally, deployment platforms like TensorFlow Serving and TensorFlow Lite enable seamless integration of trained models into production systems, allowing for real-time object detection in various applications.

6. **Visualization and Reporting:** Python libraries such as Matplotlib and Seaborn are used for visualizing training progress, model predictions, and evaluation results. These visualization tools help developers gain insights into the performance of AI models and communicate findings effectively to stakeholders.

Overall, Python's versatility and rich ecosystem of libraries and frameworks make it an ideal choice for object detection and training AI models. Its simplicity and readability enable developers to build and deploy sophisticated AI systems for a wide range of applications, from autonomous vehicles and surveillance systems to medical imaging and industrial automation.

Visual Studio Code:

Visual Studio Code (VS Code) is a popular, free, and open-source code editor developed by Microsoft. It is designed to be lightweight, customizable, and highly extensible, making it suitable for a wide range of programming languages and development tasks. Here's an overview of the key features and functionalities of Visual Studio Code:

1. **Cross-Platform:** Visual Studio Code is available for Windows, macOS, and Linux, making it a versatile choice for developers across different operating systems. This ensures consistency and compatibility regardless of the platform used.

2. **Intuitive User Interface:** VS Code features a clean and intuitive user interface that focuses on simplicity and productivity. It includes a customizable layout with a sidebar for file navigation, a code editor window with syntax highlighting and IntelliSense (code completion), and a status bar for displaying information and shortcuts.

3. **Extensions:** One of the standout features of VS Code is its extensive ecosystem of extensions. These extensions add functionality and support for various programming

languages, frameworks, and tools. Developers can easily install extensions from the Visual Studio Code Marketplace to customize their editor according to their needs.

4. **Terminal Integration:** Visual Studio Code includes an integrated terminal that allows developers to run shell commands and scripts directly within the editor. This eliminates the need to switch between the code editor and external terminal applications, streamlining the development workflow.

5. **Language Support and IntelliSense:** Visual Studio Code provides rich language support and IntelliSense features for a wide range of programming languages, including JavaScript, TypeScript, Python, Java, and more. IntelliSense offers context-aware code completion, parameter hints, and quick documentation, enhancing productivity and reducing coding errors.

6. **Customization and Theming:** Visual Studio Code is highly customizable, allowing developers to personalize their editor with themes, color schemes, and custom keybindings. Users can choose from a variety of built-in themes or create their own to tailor the editor's appearance to their preferences.

7. **Community and Support:** Visual Studio Code has a large and active community of developers who contribute to its ongoing development and improvement. The official documentation, forums, and community resources provide support and guidance to users, helping them get the most out of the editor.

Overall, Visual Studio Code is a powerful and versatile code editor that offers a rich set of features and extensions to support modern software development workflows. Its intuitive interface, extensive customization options, and robust ecosystem make it a popular choice among developers for a wide range of projects and programming languages.

CHAPTER 5

IMPLEMENTATION

Training a Custom Dataset:

1.**Prepare Dataset:** Organize data into YOLO format (images + 1. annotation files).

2.**Set Up Colab:** Create a new Python 3 notebook and connect to GPU runtime.

3.**GPU:** Connect the runtime to GPU.

4. **Install Dependencies:** Run !pip install -U -r yolov8/requirements.txt to install necessary packages.

5. **Prepare Custom Dataset:** Upload or mount dataset in Colab, and modify YOLOv8 configuration file.

6. **Training:** Execute training script with appropriate arguments, specifying paths to data, configuration, and pre-trained weights.

!yolo task=detect mode=train model=yolov8l.pt data=xxxx/xxxx/xxxx epochs=30 imgsz=640

7. **Evaluation:** Evaluate model performance using validation or test dataset, measuring metrics like mAP.

8. **Inference:** Use trained model for object detection in new images or videos.

Implementing The Custom Trained Model:

1. **Exporting the Trained Model:** Once the YOLO model is trained, you need to export it to a format compatible with inference. This typically involves converting the trained model to a format such as TensorFlow SavedModel or ONNX (Open Neural Network Exchange) format for deployment.

2. **Setting Up Inference Environment:** Next, you need to set up the inference environment where you'll run the YOLO model to perform object detection on new images or videos. This involves installing the necessary dependencies, including the deep learning framework (e.g., TensorFlow, PyTorch), any additional libraries or packages required for inference, and the trained YOLO model.

3. **Performing Object Detection:** With the inference environment set up, you can now perform object detection using the custom trained YOLO model. This typically involves loading the trained model into memory, processing input images or videos, and running inference to detect objects. You may need to post-process the output to filter detections, remove duplicate detections, or visualize the results.

4. **Integration with Application:** Finally, you can integrate the object detection functionality into your application or system. This might involve developing a user interface for interacting with the detection results, integrating the detection pipeline into an existing software stack, or deploying the model to a production environment for real-time inference.

Custom Trained Models:

Code of Models Trained to detect Personal Protective Equipment in Laborers:

```
from ultralytics import YOLO
import cv2
import cvzone
import math

model = YOLO("ppe.pt")

classNames = ['Hardhat', 'Mask', 'NO-Hardhat', 'NO-Mask', 'NO-Safety Vest', 'Person', 'Safety
Cone',
              'Safety Vest', 'machinery', 'vehicle']

cap = cv2.VideoCapture("video1.mp4")  # For Video

myColor = (0, 0, 255)
while True:
    success, img = cap.read()
    results = model(img, stream=True)
    for r in results:
        boxes = r.boxes
        for box in boxes:
            # Bounding Box
            x1, y1, x2, y2 = box.xyxy[0]
            x1, y1, x2, y2 = int(x1), int(y1), int(x2), int(y2)
            # cv2.rectangle(img,(x1,y1),(x2,y2),(255,0,255),3)
            w, h = x2 - x1, y2 - y1
            # cvzone.cornerRect(img, (x1, y1, w, h))

            # Confidence
            conf = math.ceil((box.conf[0] * 100)) / 100
            # Class Name
            cls = int(box.cls[0])
            currentClass = classNames[cls]
            print(currentClass)
            if conf>0.5:
                if currentClass =='NO-Hardhat' or currentClass =='NO-Safety Vest' or currentClass
== "NO-Mask":
                    myColor = (0, 0,255)
                elif currentClass =='Hardhat' or currentClass =='Safety Vest' or currentClass ==
"Mask":
                    myColor =(0,255,0)
```

```
            else:
                myColor = (255, 0, 0)

            cvzone.putTextRect(img, f'{classNames[cls]} {conf}',
                               (max(0, x1), max(35, y1)), scale=1, thickness=1,colorB=myColor,
                               colorT=(255,255,255),colorR=myColor, offset=5)
            cv2.rectangle(img, (x1, y1), (x2, y2), myColor, 3)

    cv2.imshow("Image", img)
    cv2.waitKey(1)
```

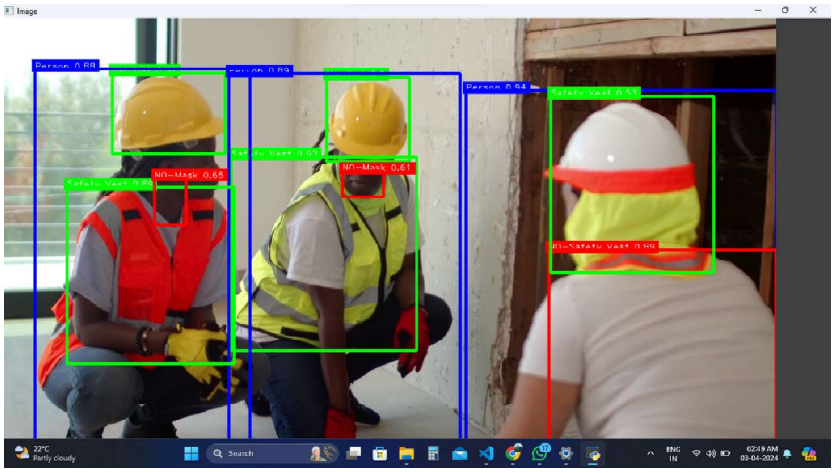
We can modify few lines of this code to implement other custom trained models as per our needs.

I have also modified this code to implement other custom trained models like to detect presence of helmets and number plates of motorcyclists in road, and also another model to detect fruits and vegetables

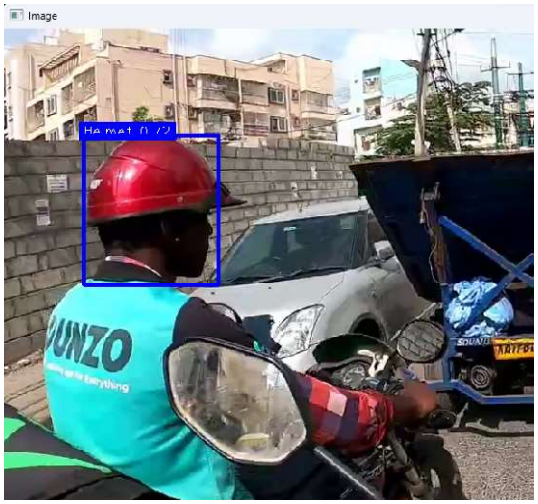
CHAPTER 6

RESULTS WITH GRAPH/SIMULATION

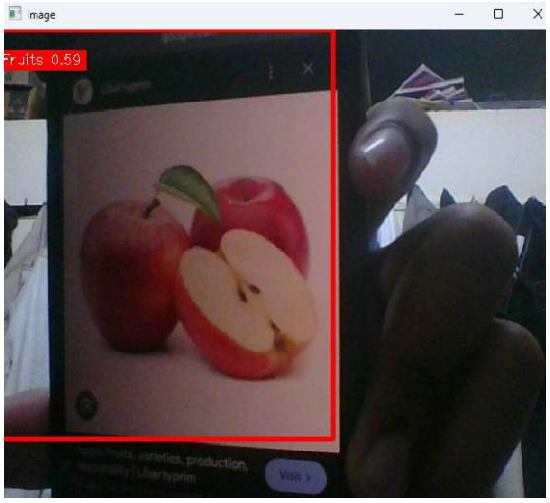
Custom Trained YOLOv8 Models for Object Detection



a) To detect PPE



b) To detect Helmets and Licence Plates in Motorcyclists



c) To detect Fruits and Vegetables using WebCam

CHAPTER 7

INFERENCES

The object detection project using YOLOv8 exemplifies the fusion of cutting-edge deep learning techniques with practical applications in computer vision. By leveraging the capabilities of YOLOv8, the project delivers an efficient and accurate object detection solution, empowering diverse industries with enhanced automation, surveillance, and decision-making capabilities.

Inferences from a custom trained YOLOv8 model for object detection can provide insights into accuracy, speed, generalization, and error analysis. Key distinctions between YOLOv8 variants include:

1. **YOLOv8:** Optimized for real-time detection with a balance between speed and accuracy.
2. **YOLOv8-Tiny:** Lightweight variant for faster inference on resource-constrained devices.
3. **YOLOv8-Small:** Balances accuracy and speed, offering improved performance over YOLOv8-Tiny.
4. **YOLOv8-Large:** Maximizes detection accuracy with deeper networks, sacrificing speed for precision.

Each variant offers trade-offs between speed and accuracy, catering to different application requirements.

CHAPTER 8

CONCLUSION

The project of training a custom YOLOv8 model and deploying it and involvement of DSP for object detection yields the following conclusions:

1. **Enhanced Model Performance:** DSP techniques improve object detection accuracy by optimizing feature extraction and mitigating environmental variability.
2. **Addressing Training Challenges:** DSP aids in addressing annotation challenges, enhancing convergence, and fine-tuning model parameters for improved performance.
3. **Optimized Deployment Considerations:** DSP optimization ensures efficient inference speed and minimizes hardware requirements for deployment.
4. **Tailored Application Specificity:** DSP allows customization of the model and deployment strategy to meet specific application needs and performance trade-offs.
5. **Continuous Improvement:** DSP-driven monitoring, refinement, and adaptation ensure ongoing optimization and adaptation to evolving requirements, enhancing the model's effectiveness over time.