Internship Report

Report submitted in fulfilment for

Summer Internship Programme

by

Ramakrishna Reddy Palle
21EE01026

Role: **AI Engineer Intern**
Flasho Tech Consultancy Services

AI Assisted Bidding – Detection of small objects in large image with occlusion and clutter

---

Contents

1. Introduction – Problem Statement

1.1 Problem Statement

In today's digital landscape, accurate and efficient small object detection in large images is critical across industries such as defense, healthcare, and manufacturing. Whether identifying symbols in architectural floor plans, detecting objects in defense imagery, or diagnosing diseases through scans, the challenge of locating small, often occluded objects within cluttered environments remains. Traditional methods struggle with **graphical variability**, **occlusion**, and **low intra-class similarity**, particularly in complex, real-world data.

In the analysis of digital architectural floor plans, detecting small symbols representing structural components, utilities, and furnishings is difficult due to overlapping elements, clutter, and varying styles. Manually counting and classifying these objects during fieldwork is impractical. Traditional symbol spotting methods, though effective for simpler layouts, fall short when dealing with such complexity. Accurate detection of these symbols, even in the presence of occlusion and clutter, is crucial for automating architectural analysis.

Recent advances in deep learning, particularly object detection frameworks like You Only Look Once (YOLO), have addressed these challenges. During my internship, I tried to developed a solution inspired by YOLOv8 for small object detection in both architectural plans and generalized applications. A key innovation was a tile-based training strategy, enabling the model to handle issues like aspect ratio differences and symbol complexity in large images.

1.2 My Research and Development

My work initially focused on symbol spotting in digital architectural floor plans, where the challenge involved detecting small, often occluded symbols in cluttered environments. However, through ongoing research and development, I tried to make a new model with extended capabilities to perform more generalized small object detection tasks. This includes applications in defence, where identifying small objects in high-resolution, cluttered imagery is essential, and in healthcare, where early diagnosis often relies on detecting subtle features in medical scans.

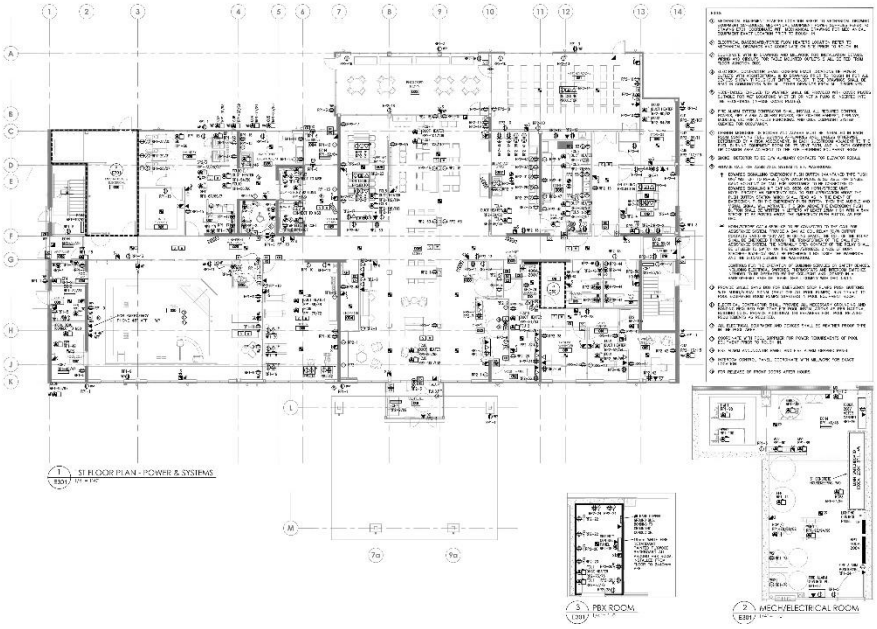
The indigenously developed model is currently under active development, undergoing testing and refinement. The aim is to enhance its adaptability across diverse scenarios and datasets, ensuring strong performance in fields like architectural analysis, defence, and medical diagnosis. This model integrates advanced deep learning techniques, leveraging the YOLOv8 architecture with tailored enhancements, majorly in the feature extraction backbone, for improved small object detection in large images. While it has shown promising results in preliminary tests, further optimization is ongoing to expand its use across different domains.

This report outlines the problem of small object detection, beginning with the architectural symbol spotting solution and expanding to broader applications. It highlights the challenges and innovative methods developed, culminating in a general-purpose model that, though still in development, shows great potential across multiple industries.

2. Key aspects of Floor plan legend detection

2.1 Dataset

In digital architecture, acquiring large, annotated datasets is difficult due to proprietary restrictions. Architectural plans are often **intellectual property**, making public access rare. Furthermore, the process of **manual annotation**, especially for small and complex symbols (legends), is labor-intensive and lacks automated tools. Typically, a few large floor plans provided by clients contain hundreds of symbols across various classes. These symbols, representing structural components and utilities, are often densely packed and subject to occlusion, complicating the detection task.



To address these challenges, I employed **tiling**, a method that splits large floor plans into smaller, more manageable sections. This preserves small object details and optimizes the model's ability to detect objects within large, complex images. Additionally, I used data augmentation techniques such as **CutMix**, **Gaussian blur**, and **random rotations** to increase the dataset's diversity. These methods helped improve the generalization of the model when training on limited, annotated floor plans. Detailed discussions on impact of these strategies on detection performance are covered in Section 4.

2.2 YOLOv8 with EfficientNet Backbone

For the symbol detection task, I utilized **YOLOv8**, a state-of-the-art object detection model known for its real-time performance and high accuracy. However, I replaced the default YOLOv8 backbone with **EfficientNet-B6** to enhance the model's feature extraction capabilities, especially for small objects. EfficientNet-B6 is optimized for balancing depth, width, and resolution, making it highly effective for detecting fine-grained details within cluttered floor plans.

This hybrid architecture leverages YOLOv8's robust detection framework while benefiting from EfficientNet-B6's superior feature representation. This modification enhances detection accuracy, especially for small, occluded symbols, without compromising real-time performance. The detailed architecture and functionality of YOLOv8 and EfficientNet will be covered in Sections 3 and 4.

3. All about YOLO (v1 to v10)

3.1 YOLO: A Unified Approach to Object Detection

You Only Look Once (YOLO) is a groundbreaking approach to object detection that treats the task as a **regression problem** instead of a classification problem. Unlike traditional multi-stage detectors (e.g., R-CNN, Fast R-CNN), YOLO predicts the bounding boxes and associated class probabilities in a single pass, making it a fast and efficient algorithm. YOLO's innovation lies in its ability to globally reason about an image, considering both the object's appearance and the contextual information surrounding it in one go.

YOLO's Core Concept: Classification as Regression

At its heart, YOLO reframes the object detection problem as a single regression task. This is a departure from traditional methods like **R-CNN**, which involve multiple stages: generating region proposals, extracting features, and classifying them. YOLO eliminates the need for region proposals by dividing the image into a fixed grid and predicting bounding boxes and class probabilities for each grid cell simultaneously.

The grid-based approach allows the model to predict **SxS** grid cells for an input image, with each grid predicting **B** bounding boxes and confidence scores, along with **C** class probabilities. The confidence score reflects the likelihood that a box contains an object, and the class probability represents what type of object is present. Each bounding box prediction includes:

- Center coordinates (**x, y**)
- Width and height (**w, h**)
- Confidence score for the object
- Class probabilities

The output for a 7x7 grid with 2 bounding boxes per cell and 20 classes would be a **7x7x30** tensor, which encodes bounding box coordinates, objectness scores, and class probabilities.

Mathematical Approach: SSD vs. Multi-Stage Detectors

YOLO differs fundamentally from multi-stage detectors like **Fast/Faster R-CNN** and **Deformable Parts Model (DPM)**. These detectors rely on **region proposal networks (RPN)** to generate multiple candidate boxes and then apply classification. Instead, YOLO uses **Single Shot Detection (SSD)**: a fully convolutional network that predicts bounding boxes and class probabilities in a single pass over the image.

In YOLO:

- **Bounding Box Regression**: Each grid cell predicts the **x, y** coordinates (normalized to fall between 0 and 1), width, and height of objects. The loss function penalizes errors in these coordinates using a custom weighting factor (**λ_coord**) to ensure localization is prioritized.
- **Confidence Score**: Each predicted box has an associated confidence score representing the Intersection over Union (IoU) between the predicted box and the ground truth.
- **Class Prediction**: For each box, class probabilities are predicted. The model is trained using a **multinomial logistic regression** to handle the multi-class problem.

The loss function combines these components:

- **Localization Loss (Bounding box coordinates)**: Sum-squared error is used for the predicted coordinates (**x, y, w, h**).
- **Confidence Loss**: Measures how confident the model is about the presence of an object.
- **Classification Loss**: For the class prediction using softmax cross-entropy.

3.2 YOLO Versions Overview

YOLO has undergone numerous iterations, with each version introducing architectural improvements and optimization techniques aimed at improving both speed and accuracy. Below is a summary of the key advancements in each YOLO version.

3.2.1 YOLOv1 (2016)

Architecture and Aim: YOLOv1 was the first version to introduce the unified detection approach. The backbone used was Darknet, a custom CNN architecture. The key aim was to create a fast detector by eliminating region proposals and performing detection directly on the full image.

Algorithm:

- Image divided into a 7x7 grid.
- For each grid, 2 bounding boxes and 20 class probabilities are predicted.
- The output is a 7x7x30 tensor.

Loss Function:

- Sum-squared error loss is used for both classification and bounding box regression, which, while easy to optimize, led to instability due to the large number of empty grid cells.
- Custom weighting parameters λ_coord (for coordinates) and λ_noobj (for cells without objects) were introduced to stabilize training.

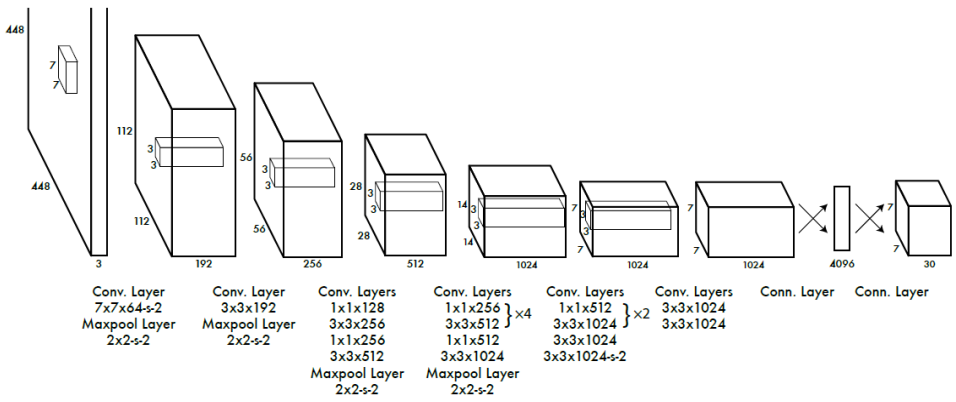


Figure 3: The Architecture. Our detection network has 24 convolutional layers followed by 2 fully connected layers. Alternating 1×1 convolutional layers reduce the features space from preceding layers. We pretrain the convolutional layers on the ImageNet classification task at half the resolution (224×224 input image) and then double the resolution for detection.

3.2.2 YOLOv2 / YOLO9000 (2017)

Aim: YOLOv2 focused on improving accuracy without sacrificing speed, primarily by incorporating techniques like **Batch Normalization** and higher-resolution images during training.

Architecture:

- Introduced the **Darknet-19** backbone for better feature extraction.
- Use of **anchor boxes** similar to Faster R-CNN for better bounding box predictions.
- Multi-scale training for running the model at varying sizes
- Proposed a Joint-Training method with mix of detection dataset (COCO) and classification dataset (ImageNet)
- Reaches 9000 class capacity

Algorithm:

- Higher input resolution of **416x416 pixels**.
- Introduced **fine-grained features** for better small object detection.
- Removal of fully connected layers, making the model fully convolutional.
- Joint training on object detection and classification.
- Multi-scale training method for flexibility in speed and accuracy tradeoffs.

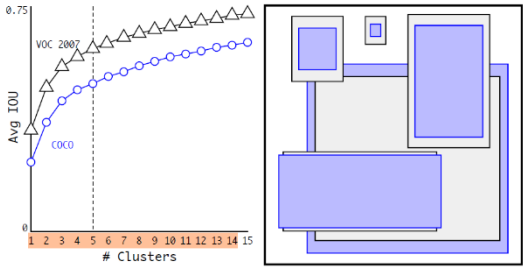


Figure 2: Clustering box dimensions on VOC and COCO. We

- Using hierarchical view of object classification to combine distinct datasets.
- Joint training algorithm to train on both detection and classification data.
- Introduction of Batch Normalization to improve convergence.
- High-resolution classifier fine-tuning for better accuracy.
- Use of k-means clustering for automatic anchor box prior assignment.
- Direct location prediction using logistic activation.
- Fine-grained feature extraction via passthrough layer.
- Multi-scale training to enhance robustness to different image sizes.
- Proposal of Darknet-19 as the classification model.

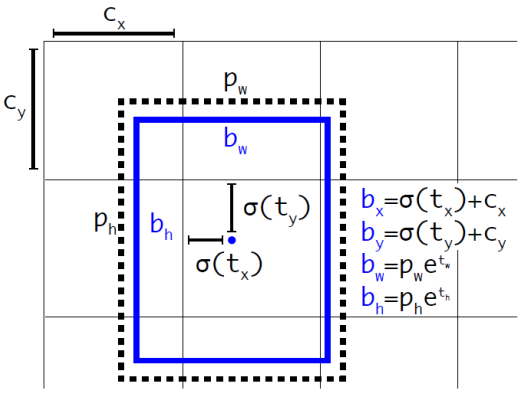


Figure 3: Bounding boxes with dimension priors and location prediction. We predict the width and height of the box as offsets

Loss Function:

- Improved with the introduction of anchor boxes and custom IoU-based loss metrics for better localization.

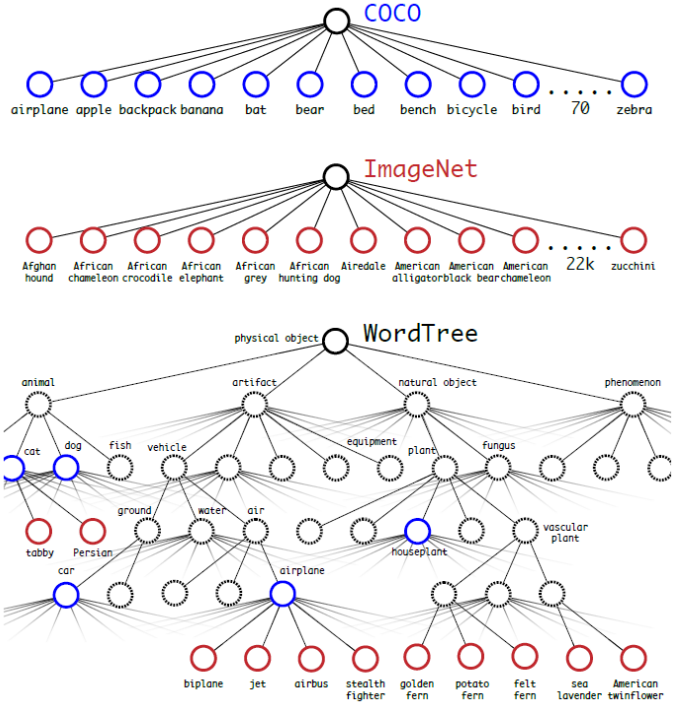


Figure 6: Combining datasets using WordTree hierarchy. Us-

Type	Filters	Size/Stride	Output
Convolutional	32	3×3	224×224
Maxpool		$2 \times 2/2$	112×112
Convolutional	64	3×3	112×112
Maxpool		$2 \times 2/2$	56×56
Convolutional	128	3×3	56×56
Convolutional	64	1×1	56×56
Convolutional	128	3×3	56×56
Maxpool		$2 \times 2/2$	28×28
Convolutional	256	3×3	28×28
Convolutional	128	1×1	28×28
Convolutional	256	3×3	28×28
Maxpool		$2 \times 2/2$	14×14
Convolutional	512	3×3	14×14
Convolutional	256	1×1	14×14
Convolutional	512	3×3	14×14
Convolutional	256	1×1	14×14
Convolutional	512	3×3	14×14
Maxpool		$2 \times 2/2$	7×7
Convolutional	1024	3×3	7×7
Convolutional	512	1×1	7×7
Convolutional	1024	3×3	7×7
Convolutional	512	1×1	7×7
Convolutional	1024	3×3	7×7
Convolutional	1000	1×1	7×7
Avgpool		Global	1000
Softmax			

Table 6: Darknet-19.

3.2.3 YOLOv3 (2018)

Aim: YOLOv3 aimed at boosting both small object detection and overall performance. It introduced a more sophisticated architecture for feature extraction, borrowing from **ResNet**-style multi-scale prediction.

Architecture:

- Backbone: **Darknet-53**, leveraging residual blocks.
- Multi-scale predictions: YOLOv3 predicts bounding boxes at three different scales for better detection of both large and small objects.
- Successive 3x3 and 1x1 convolutional layers with shortcut connections.
- Significantly larger network with 53 convolutional layers.
- Inspired by residual networks, integrated for improved performance and robustness

	Type	Filters	Size	Output
	Convolutional	32	3 □ 3	256 □ 256
	Convolutional	64	3 □ 3 / 2	128 □ 128
1□	Convolutional	32	1 □ 1	
	Convolutional	64	3 □ 3	
	Residual			128 □ 128
	Convolutional	128	3 □ 3 / 2	64 □ 64
2□	Convolutional	64	1 □ 1	
	Convolutional	128	3 □ 3	
	Residual			64 □ 64
	Convolutional	256	3 □ 3 / 2	32 □ 32
8□	Convolutional	128	1 □ 1	
	Convolutional	256	3 □ 3	
	Residual			32 □ 32
	Convolutional	512	3 □ 3 / 2	16 □ 16
8□	Convolutional	256	1 □ 1	
	Convolutional	512	3 □ 3	
	Residual			16 □ 16
	Convolutional	1024	3 □ 3 / 2	8 □ 8
4□	Convolutional	512	1 □ 1	
	Convolutional	1024	3 □ 3	
	Residual			8 □ 8
	Avgpool		Global	
	Connected		1000	
	Softmax			

Table 1. **Darknet-53.**

Algorithm:

- Introduced **Focal Loss** to handle class imbalance.
- Bounding boxes predicted at multiple feature map layers for small, medium, and large objects.
- Bounding boxes predicted using dimension clusters as anchor boxes with 4 coordinates for each bounding box.
- Sum of squared error loss for training.
- Logistic regression for predicting objectness scores with a threshold of 0.5.
- Independent logistic classifiers for class predictions using binary cross-entropy loss.
- Multilabel approach for handling overlapping labels in datasets.
- Predictions at three different scales using a concept similar to feature pyramid networks.
- Bounding box priors determined using k-means clustering with 9 clusters distributed across three scales

3.2.4 YOLOv4 (2020)

Aim: YOLOv4 introduced more optimizations, focusing on both accuracy and inference speed.

Architecture:

- **CSPDarknet-53** as the backbone, improving gradient flow and computational efficiency.
- Integration of **Path Aggregation Network (PANet)** for better localization.

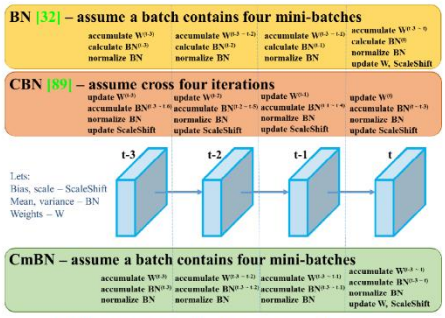


Figure 4: Cross mini-Batch Normalization.

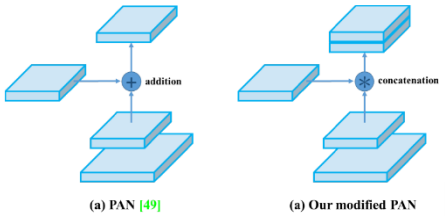


Figure 6: Modified PAN.

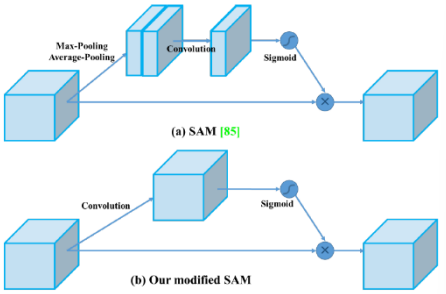


Figure 5: Modified SAM.

- **Input Layer:** Input Image (e.g., 640x640 pixels)**.**
- **Backbone:** CSPDarknet53 (chosen for feature extraction).
- **Neck:**
  - FPN (Feature Pyramid Network)
  - PANet (Path Aggregation Network)
  - BiFPN (Bi-directional Feature Pyramid Network)
- **Intermediate Processing:**
  - SPP (Spatial Pyramid Pooling)
  - ASPP (Atrous Spatial Pyramid Pooling)
  - RFB (Receptive Field Block)
- **Activation Function:** Mish Activation
- **Detection Heads:**
  - YOLO Head (anchor-based)
- **Loss Functions:**
  - CIoU Loss (for bounding box regression)
- **Output Layer:**
  - Bounding Box Predictions
  - Class Scores
  - Objectness Scores
- **Post-processing:**
  - Non-Maximum Suppression (NMS)

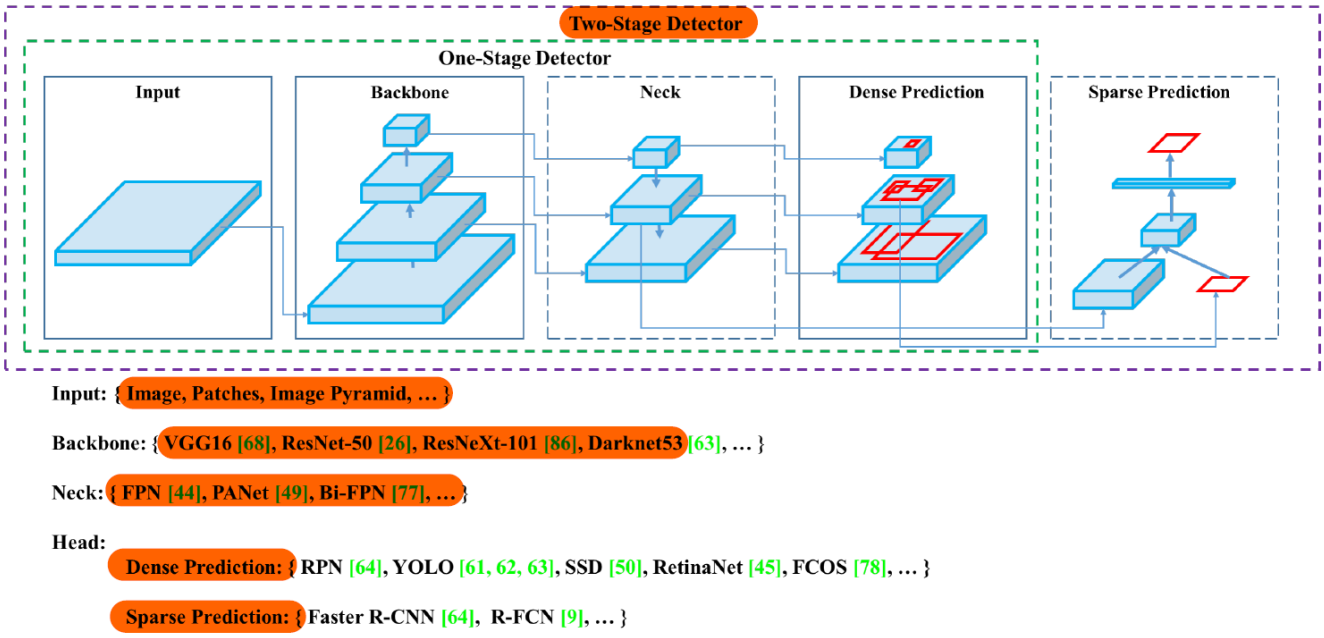


Figure 2: Object detector.

Summary:

- Input Image → Backbone (CSPDarknet53) → Neck (FPN + PANet + BiFPN) → Intermediate Processing (SPP + ASPP + RFB) → Activation (Mish) → Detection Heads (YOLO) → Loss Functions (CIoU Loss) → Output Predictions (Bounding Boxes, Class Scores, Objectness Scores) → Post-processing (NMS)

3.2.5 YOLOv5 (2020)

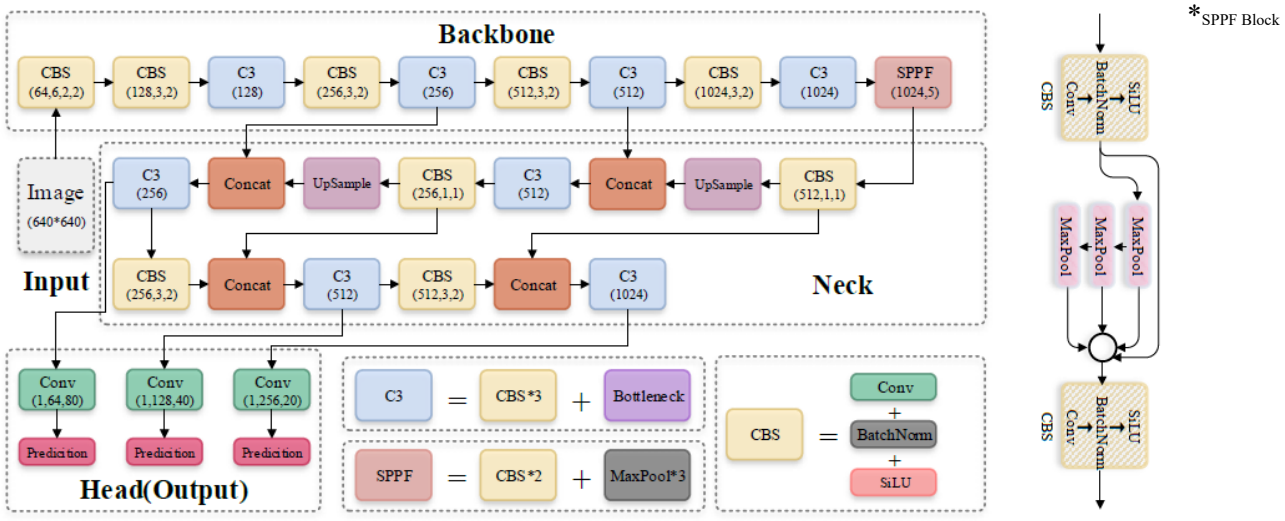
Aim: YOLOv5 aimed at making the framework easier to use and more scalable. It introduced auto-anchor generation and hyperparameter tuning.

Architecture:

- Improved backbone and detection head based on YOLOv4 with auto-anchor optimization.

Loss Function:

- Continued the use of CIoU loss for bounding box prediction.



3.2.6 YOLOv6 (2022)

Aim: YOLOv6 was developed to focus on achieving superior **accuracy in real-time scenarios**, particularly on **edge devices** and in resource-constrained environments. It emphasizes **deployment efficiency**, while maintaining high detection speed and precision.

Architecture:

- **Anchor-Free Design**: YOLOv6 moved towards an **anchor-free architecture** to reduce the overhead caused by anchor box generation and matching, which is computationally expensive. This design simplifies the model and improves efficiency.
- **RepVGG Backbone**: YOLOv6 adopted **RepVGG**, a highly efficient backbone designed for fast inference and deployment, providing significant speedup while maintaining high accuracy.
- **Efficient Feature Pyramid**: YOLOv6 uses an improved **BiFPN (Bidirectional Feature Pyramid Network)** for multi-scale feature fusion, enhancing the detection of objects at various scales.

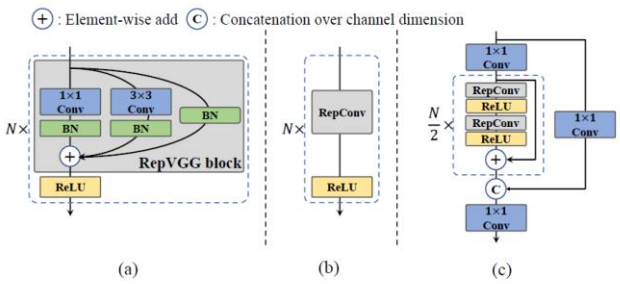


Figure 3: (a) RepBlock is composed of a stack of RepVGG blocks with ReLU activations at training. (b) During inference time, RepVGG block is converted to RepConv. (c) CSPStackRep Block comprises three 1×1 convolutional layers and a stack of sub-blocks of double RepConvs following the ReLU activations with a residual connection.

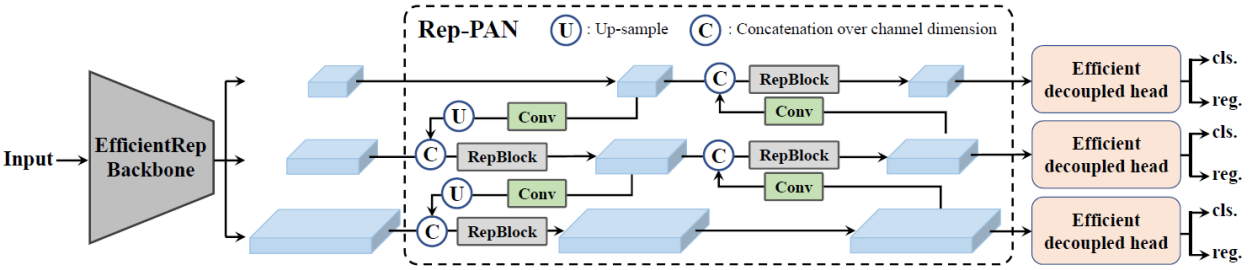


Figure 2: The YOLOv6 framework (N and S are shown). Note for M/L, RepBlocks is replaced with CSPStackRep.

Algorithm:

- **Efficient Training Techniques**: YOLOv6 incorporated advanced training strategies such as **mixed precision training**, **Knowledge Distillation (self)**, **gradient accumulation**, and **label smoothing** to boost training speed and convergence without sacrificing performance.
- **Post-Processing Optimization**: Introduced **Efficient NMS (Non-Maximum Suppression)**, Quatization (PTQ & QAT), reducing inference time by eliminating redundant detections more efficiently.

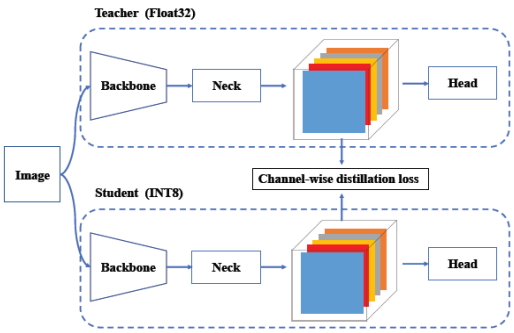


Figure 5: Schematic of YOLOv6 channel-wise distillation in QAT.

Loss Function:

- **Decoupled Head**: YOLOv6 decoupled classification and regression heads, using independent loss functions to optimize the bounding box coordinates and class predictions separately, leading to better convergence and stability.

3.2.7 YOLOv7 (2022)

Aim: YOLOv7 was designed to optimize the accuracy-to-speed trade-off, incorporating modern deep learning techniques for feature extraction, training strategies, and architectural enhancements, aiming for real-time detection with high precision.

Architecture:

1. **Extended Efficient Layer Aggregation Networks (E-ELAN):**
    - **Uses group convolutions to enhance learning without destroying gradient paths.**
    - **Key Techniques:**
        - **Expand: Increase channel and cardinality of computational blocks.**
        - **Shuffle: Group and concatenate feature maps.**
        - **Merge Cardinality: Combine feature maps from multiple groups.**

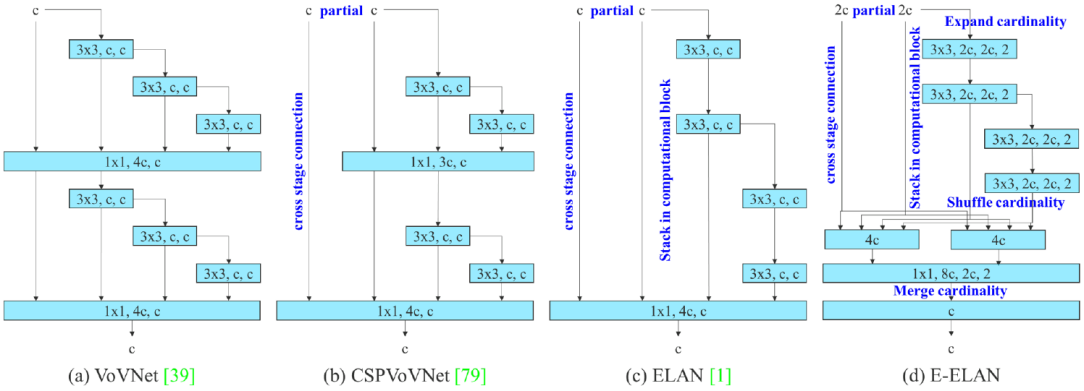


Figure 2: Extended efficient layer aggregation networks. The proposed extended ELAN (E-ELAN) does not change the gradient transmission path of the original architecture at all, but use group convolution to increase the cardinality of the added features, and combine the features of different groups in a shuffle and merge cardinality manner. This way of operation can enhance the features learned by different feature maps and improve the use of parameters and calculations.

2. **Model Scaling for Concatenation-Based Models:**
    - **Width and Depth Scaling: Maintains optimal structure by adjusting transition layers accordingly.**

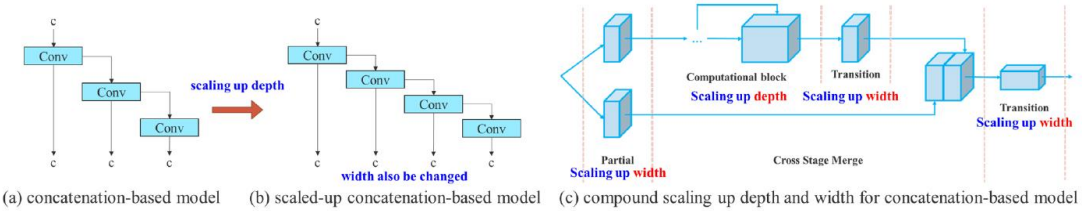


Figure 3: Model scaling for concatenation-based models. From (a) to (b), we observe that when depth scaling is performed on concatenation-based models, the output width of a computational block also increases. This phenomenon will cause the input width of the subsequent transmission layer to increase. Therefore, we propose (c), that is, when performing model scaling on concatenation-based models, only the depth in a computational block needs to be scaled, and the remaining of transmission layer is performed with corresponding width scaling.

Trainable Bag-of-Freebies:

1. **Planned Reparameterized Convolution (RepConvN):**
    - **Replaces traditional RepConv layers to avoid identity connections, enhancing performance in residual or concatenation-based models.**

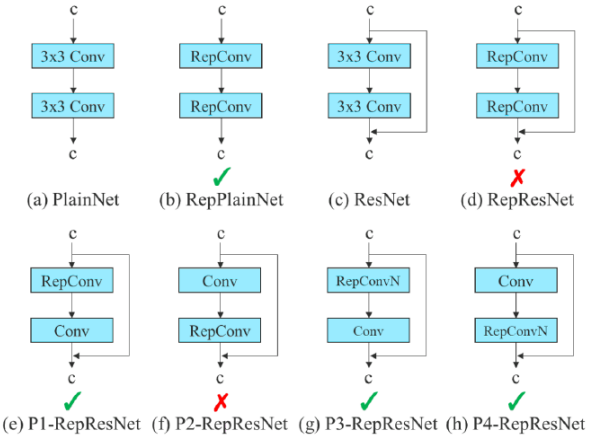


Figure 4: Planned re-parameterized model. In the proposed planned re-parameterized model, we found that a layer with residual or concatenation connections, its RepConv should not have identity connection. Under these circumstances, it can be replaced by RepConvN that contains no identity connections.

2. **Coarse-to-Fine Label Assignment:**
    - **Coarse Labels: Relax constraints to increase recall in auxiliary heads.**
    - **Fine Labels: More accurate labels guided by lead head predictions.**

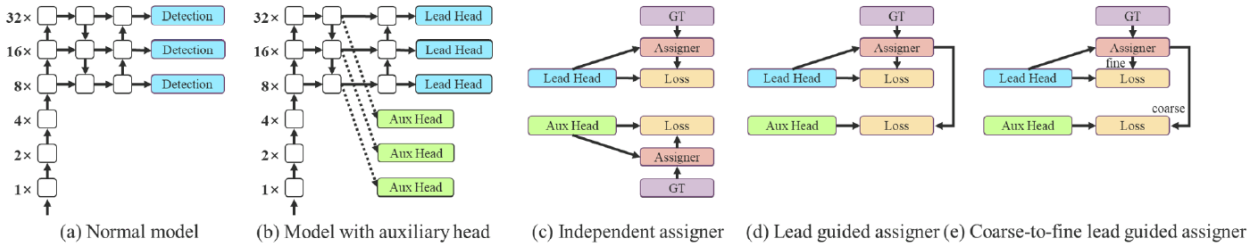


Figure 5: Coarse for auxiliary and fine for lead head label assigner. Compare with normal model (a), the schema in (b) has auxiliary head. Different from the usual independent label assigner (c), we propose (d) lead head guided label assigner and (e) coarse-to-fine lead head guided label assigner. The proposed label assigner is optimized by lead head prediction and the ground truth to get the labels of training lead head and auxiliary head at the same time. The detailed coarse-to-fine implementation method and constraint design details will be elaborated in Apendix.

3. **Additional Techniques:**
    - **Batch Normalization Integration: Merges normalization parameters with convolutional weights.**
    - **Implicit Knowledge Utilization: Simplifies and integrates knowledge from YOLOR.**
    - **EMA Model: Uses Exponential Moving Average for final inference.**

Algorithm:

- **Model Scaling**: YOLOv7 scales across width and depth to balance the accuracy and performance on different hardware devices, offering variants like YOLOv7-tiny for real-time applications.
- **Task-Specific Heads**: YOLOv7 features separate heads for object detection, classification, and instance segmentation, optimizing for various tasks.

Loss Function:

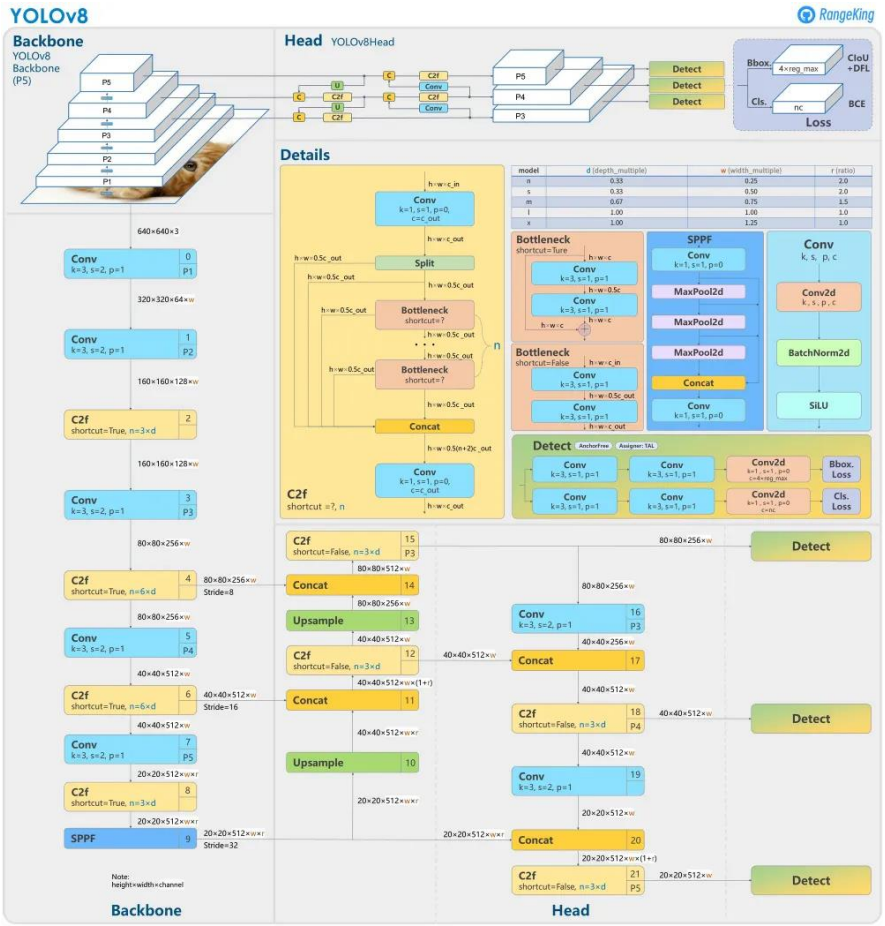
- Continuation of **CIoU (Complete Intersection over Union) Loss** for bounding box regression to ensure more accurate localization.

3.2.8 YOLOv8 (2023)

Aim: YOLOv8 introduced a more **modular architecture**, allowing users to switch between different backbones and necks for better flexibility in specific use cases, further enhancing the ease of implementation.

Architecture:

- **Backbone Modularization**: YOLOv8 introduced the flexibility to use different backbones such as **EfficientNet** or **CSPNet**.
- **Neck Upgrades**: Enhanced neck modules, including **PANet (Path Aggregation Network)** and **BiFPN (Bidirectional Feature Pyramid Network)**, allow better multi-scale feature fusion.



Algorithm:

- **Dynamic Anchor Boxes**: YOLOv8 can dynamically adjust anchor boxes based on the data it encounters during training, leading to better bounding box predictions for varied datasets.
- **Multi-Scale Predictions**: Predicts at multiple scales for improved detection of objects of varying sizes.

Loss Function:

- Incorporates **EIoU (Enhanced Intersection over Union) Loss**, further refining the balance between object localization and prediction stability.

4. Dataset, Training Strategy & Results

4.1 Dataset

In this project, we utilized a **pretrained YOLOv8 model API**. Given the proprietary nature of digital architecture floor plan datasets, obtaining a wide range of floor plans was challenging. Additionally, in the construction and contracting industry, processes like **takeoff**, **estimation**, and **responding to Requests for Proposals (RFPs)** are labor-intensive and prone to human error. These tasks require skilled labor and consume considerable time. Our solution aims to **automate** and streamline these processes, improving both accuracy and operational efficiency.

With only a small set of **floor plans** provided by the client, each with a resolution of approximately **5000x4000 pixels** and containing **electrical legends** of around **120x120 pixels** (e.g., fire extinguishers, alarms, cameras), we manually annotated these images using a tool called **LabelImg**.

Annotation with LabelImg

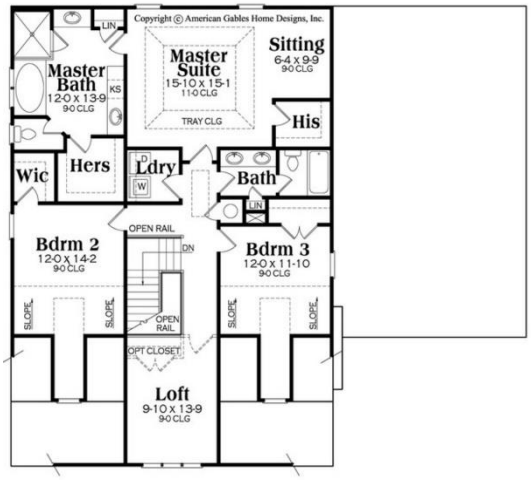
We used **LabelImg**, an open-source annotation tool, to label various electrical components across **71 classes** manually. Some examples of labeled objects include **fire alarms**, **CCTV cameras**, **fire extinguishers**, etc. This process required precision and attention to detail, ensuring that the legends were accurately marked for training.

However, we encountered several challenges:

- **Limited data** availability.
- Existing feature extraction backbones struggle with detecting small objects in such large images.
- **Aspect ratio inconsistencies** across floor plans.
- **Cluttered scenes** with occlusions, varying orientations, and small-sized objects.

4.2 Data Preprocessing Strategy

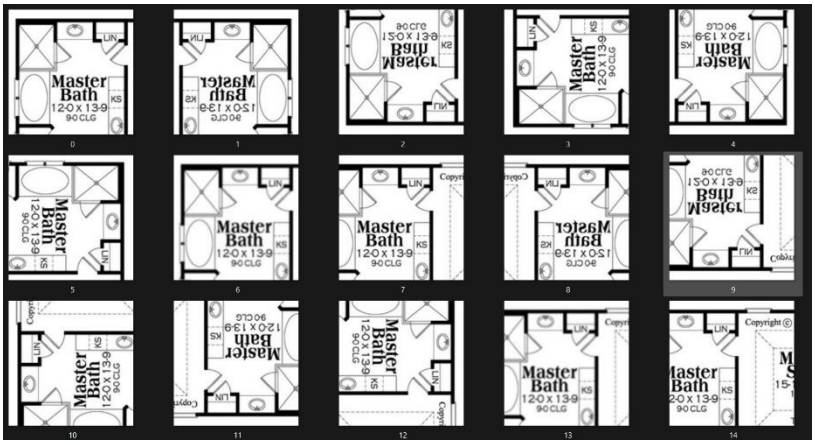
Given the challenges posed by the dataset, we developed a **sophisticated data preprocessing pipeline**. This included a **tiling algorithm** and **data augmentation** to ensure that the model could learn effectively from the available images.



Tiling Algorithm

We divided each floor plan into smaller tiles, ensuring that the model could handle large images while maintaining the integrity of small object detection. The process was as follows:

1. The image was broken into **square tiles** of **250 to 500 pixels** starting from the top-left corner.
2. A **sliding window** technique was used, with a **stride** of **50 or 100 pixels** to avoid information loss between consecutive tiles.
3. The algorithm accounted for **edge cases** where the last tile didn't fit exactly into the image width or height, dynamically adjusting the stride for the final tiles.
4. This process resulted in **thousands of tiles**, each paired with its corresponding annotation.

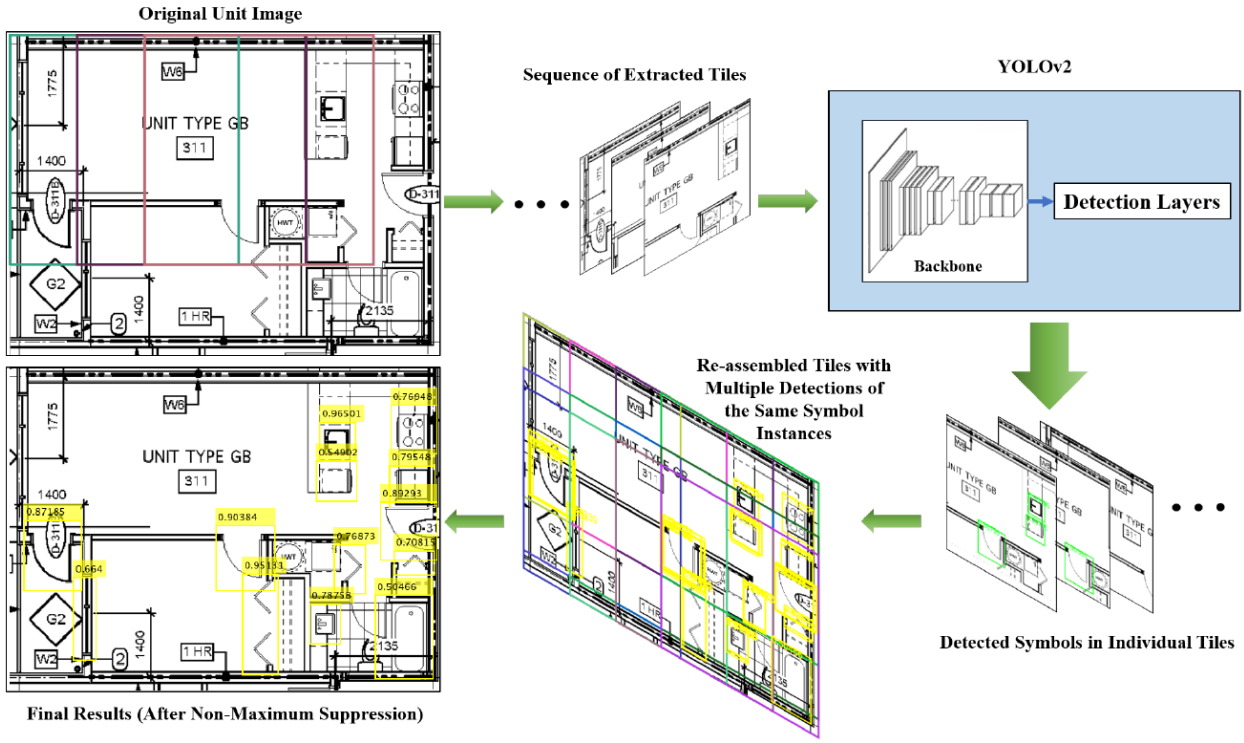


Handling Overlapping Objects

A key challenge was that objects frequently appeared in multiple tiles due to the overlap. This led to duplicate detections, where the model would predict the same object multiple times across neighbouring tiles. To resolve this issue, we implemented **Non-Maximum Suppression (NMS)**.

Non-Maximum Suppression (NMS)

NMS helps eliminate redundant detections by selecting the tile with the **highest confidence score** and suppressing others. This technique is crucial in maintaining accuracy when the same object is detected across multiple tiles. The **threshold value** for suppression was experimentally determined based on the characteristics of our dataset, ensuring optimal performance.



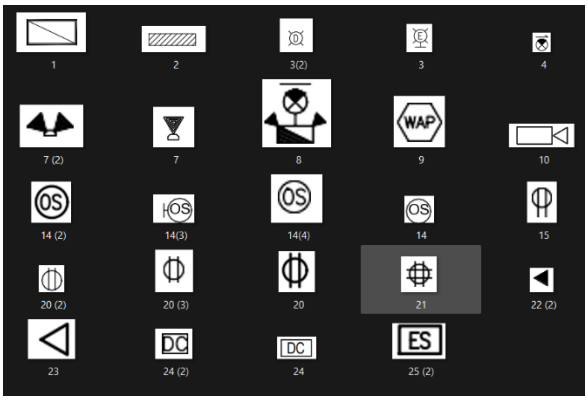
Data Augmentation

To further enhance the robustness of the dataset, we applied various augmentation techniques:

- **Gaussian Blur** to simulate real-world distortions.
- **Random Flips** and **Rotations** to account for varying orientations of objects.
- **Scale Transformations** and **CutMix** for improving generalization and handling occlusions.

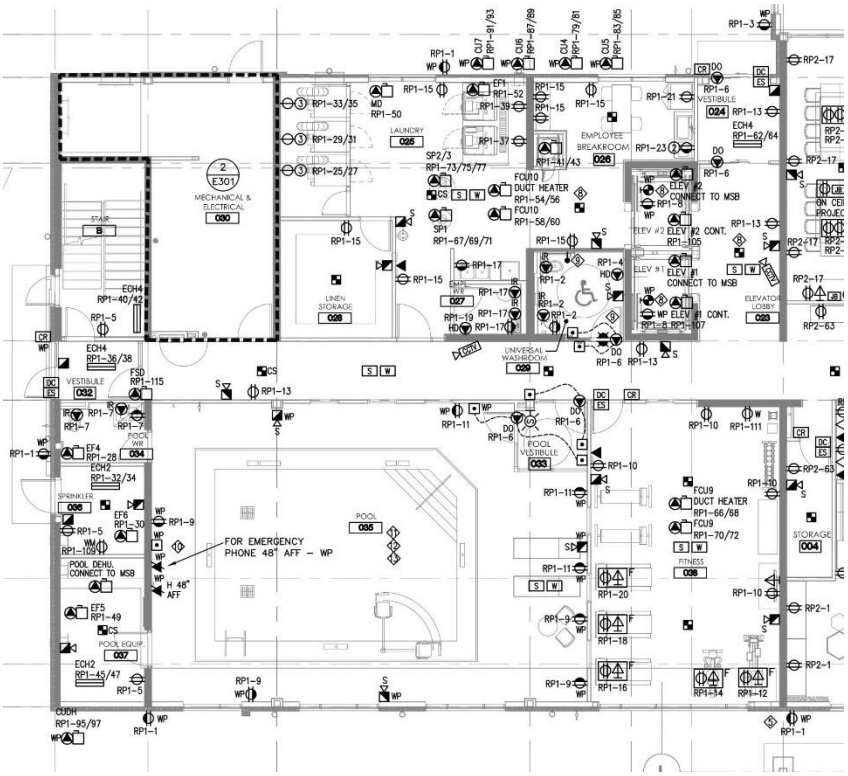
4.3 Training Strategy

The training was carried out in two stages:



1. **Initial Training on Resized Legends**:
   We resized all the annotated legend symbols to a uniform size (maximum being **240x240 pixels**) using padding, then trained the model for **100 epochs**. This provided the model with a foundational understanding of the object classes.

2. **Fine-Tuning with Tiled Dataset**:
   After preparing the tiled dataset with augmentations, we trained a custom YOLOv8 model with an **EfficientNet-B6** backbone. EfficientNet-B6 was chosen due to its **MBConv blocks**, which leverage bottleneck layers, inverted residuals, depthwise convolutions, and **SE (Squeeze-and-Excitation) blocks**. These features enabled efficient processing of our high-resolution tiles, capturing the small and intricate details of the electrical legends.



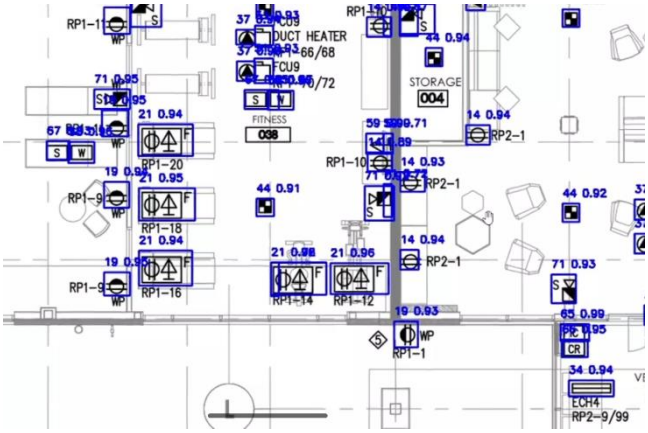
Model Training Environment

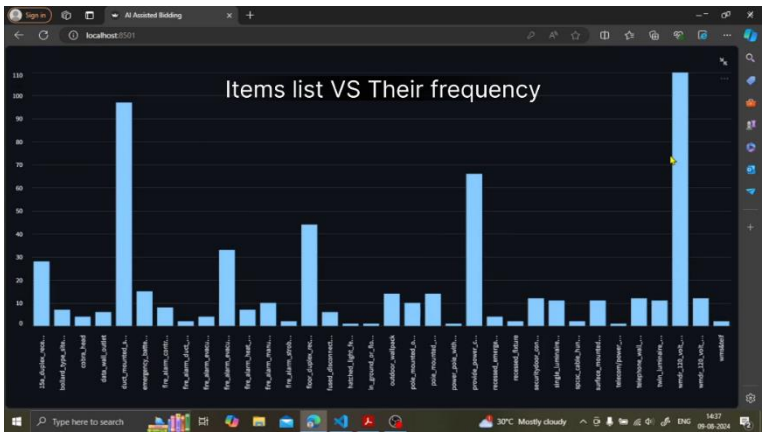
The training was conducted on an **AWS EC2 Spot Instance**, ensuring high computational power while keeping costs low. The model was evaluated using **YOLOv8's standard evaluation metrics**, focusing on both precision and recall to ensure accurate detection across different classes.

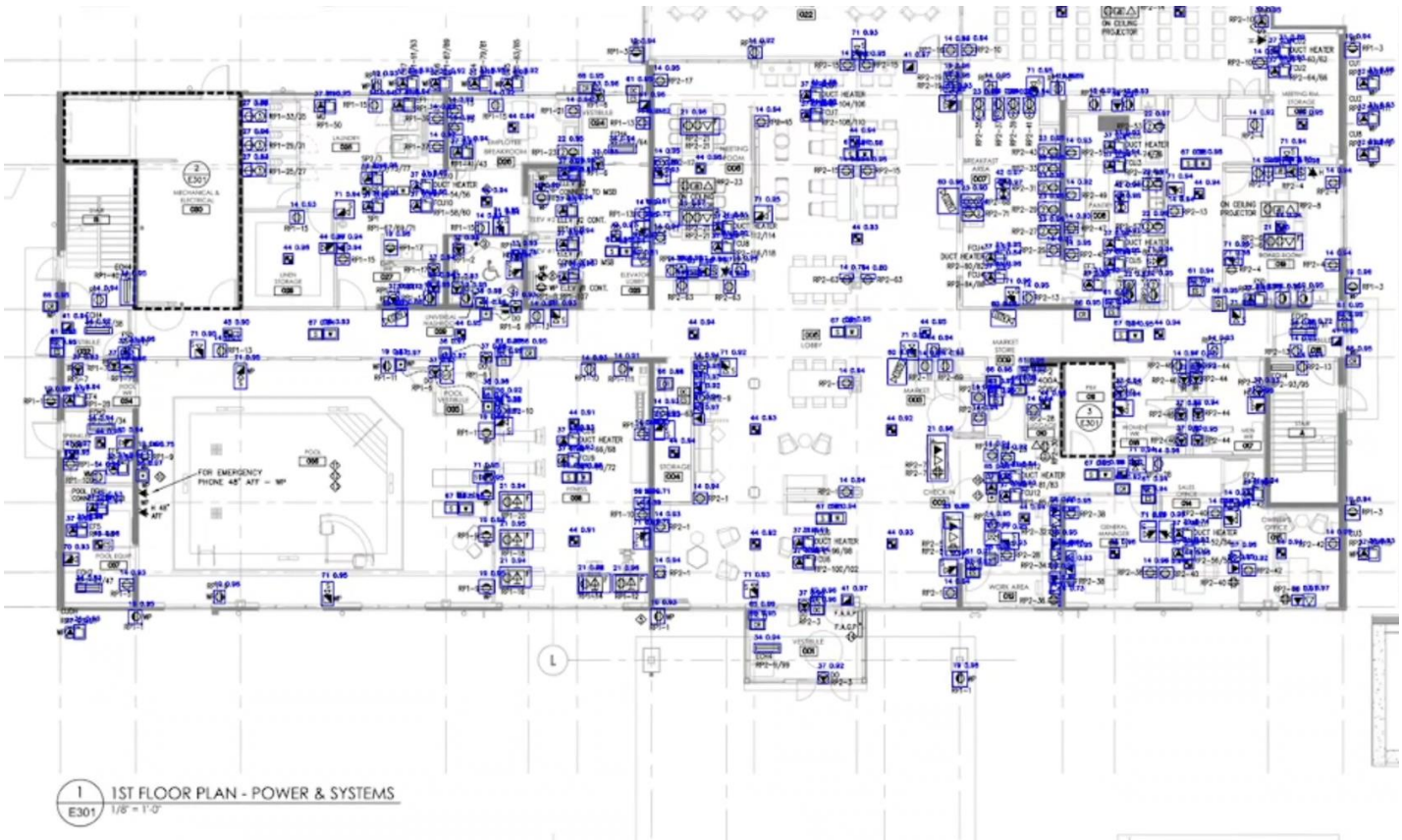
4.4 Output and Evaluation

The final model produced two types of output:

- **Annotated Floor Plans**: Each detected object was marked with a bounding box, class label, and confidence score. These annotated outputs provided a clear visual representation of the detection results.
- **CSV Reports**: A list of all detected objects, along with their counts, was generated as a **CSV file** for easy review and further analysis by the client.







5. Streamlit App & Product Deployment

The web application was developed using **Streamlit**, a framework for building interactive web applications, and was designed according to client requirements. The primary functionality of the app integrates the trained object detection model with a seamless user interface, leveraging **Amazon Web Services (AWS)** for back-end storage and processing.

5.1 Backend Integration with AWS and GitHub

The model deployment workflow relies on integrating the trained YOLOv8 model with **AWS S3** for storing user-uploaded images and fetching model outputs. The following key technologies and services were used for the backend:

- **GitHub Repository Integration**: The app fetches the model code and relevant scripts directly from a **GitHub repository**. Using a **CI/CD pipeline**, the app ensures that the latest version of the model and other dependencies are always available during deployment.
- **AWS S3 Bucket**: All floor plan images uploaded by users are stored in an S3 bucket. The **Boto3** library was used to interact with AWS services, handling image uploads and downloads seamlessly.
- **AWS IAM Role & Access Keys**: To securely access the S3 bucket and execute the model stored on AWS, an **IAM role** was created with the necessary permissions. The app uses **AWS access keys** and **secret keys** to authenticate and connect the backend to the S3 bucket, ensuring secure interaction between AWS services and GitHub code.

5.2 App Flow and Features

The app contains the following key components:

1. **Login Page**: Users are required to log in with credentials to ensure secure access. Once logged in, they are redirected to the main dashboard.
2. **Image Upload**: The app allows users to upload floor plan images in specific formats (JPG, PNG, PDF). Upon clicking the upload button, the image is directly uploaded to the **S3 bucket**. After a successful upload, the image is sent to the model for further processing.
3. **Model Inference**: Once the image is uploaded to the S3 bucket, the backend retrieves it and passes it to the YOLOv8 object detection model hosted on AWS. Using **object keys** (unique identifiers assigned to each file in S3), the app fetches and processes the image. The model is run using the **AWS Lambda service** with the code integrated from GitHub, ensuring real-time inference.
4. **Annotated Output**: The processed output, which includes the original image with bounding boxes drawn around detected objects (electrical legends), is displayed back on the web app. This image gives users a clear view of the detected objects on the floor plan.
5. **Bar Chart Display**: A bar chart is generated that provides a detailed count of each detected item (e.g., fire alarms, CCTV cameras). This gives users a statistical overview of the detected objects.
6. **Download Options**: Users are provided with options to download:
    - The **annotated image** with detected objects.
    - A **CSV file** containing a detailed list of detected items, including class names and counts.

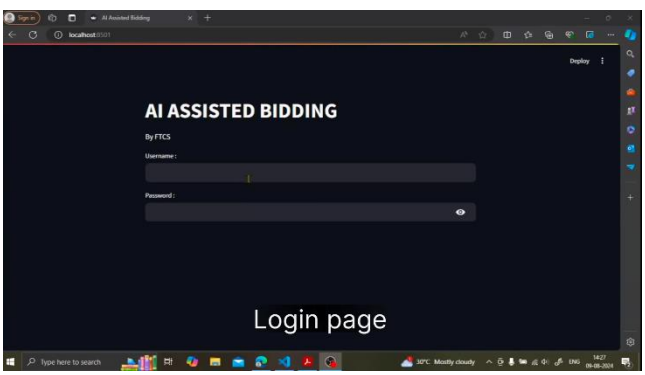
7. **Estimator Feature**: The app includes an estimator tool designed to assist builders or clients in estimating the total cost of equipment based on the detected objects. By considering the local circumstances, such as installation costs and the price of each item, the estimator calculates a comprehensive cost report.

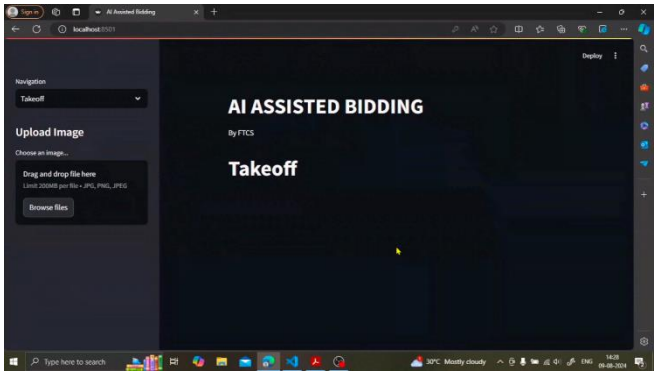
5.3 AWS Architecture Overview

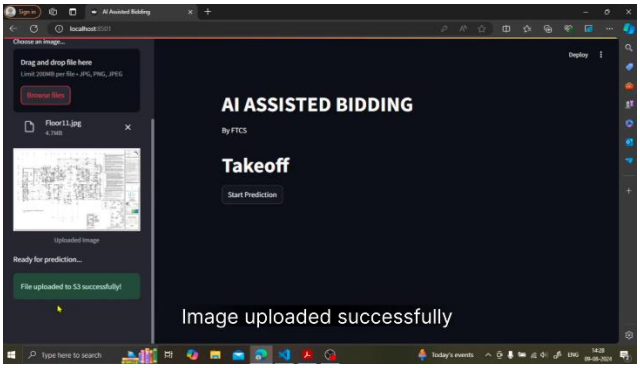
The deployment involves a smooth integration of multiple AWS services:

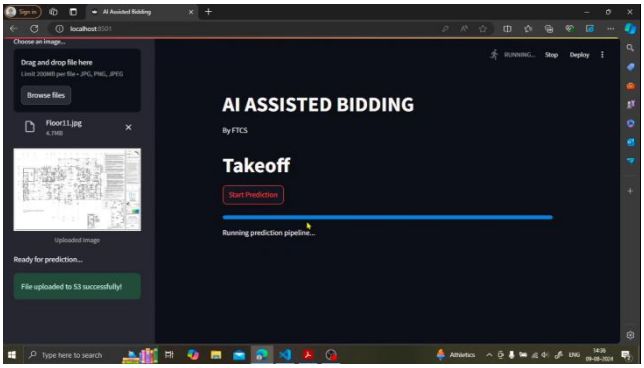
- **S3 for Storage**: Storing both input images and processed results.
- **AWS Lambda**: Triggered for model inference, where the uploaded images are passed to the model for prediction.
- **API Gateway**: Facilitates interaction between the web app and backend services, allowing the model inference to be called via an API.
- **IAM Roles & Access Keys**: Ensure secure access to AWS services, while limiting access to only authorized resources.

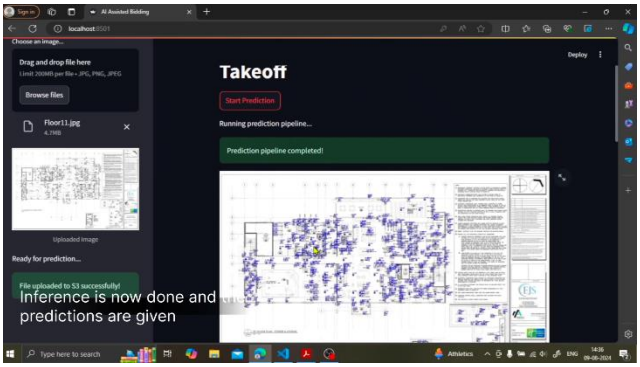
The use of **object keys** ensures that each uploaded image in the S3 bucket is uniquely identified, allowing efficient retrieval and processing. By linking the app's back-end model inference with the key, the application ensures precise detection for each uploaded floor plan.

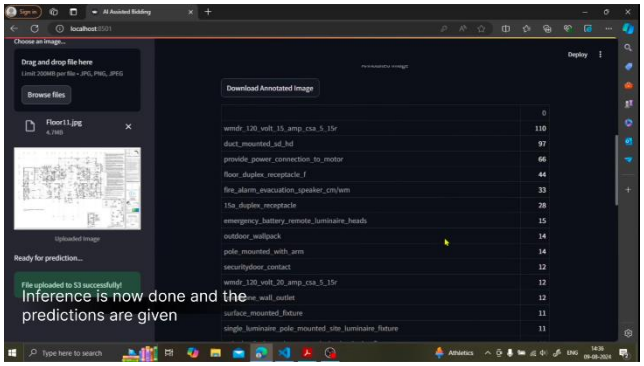


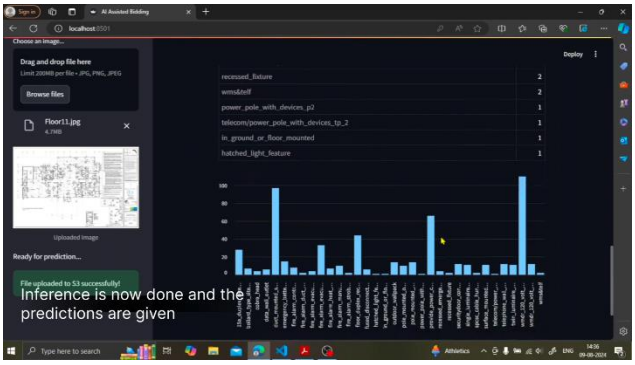


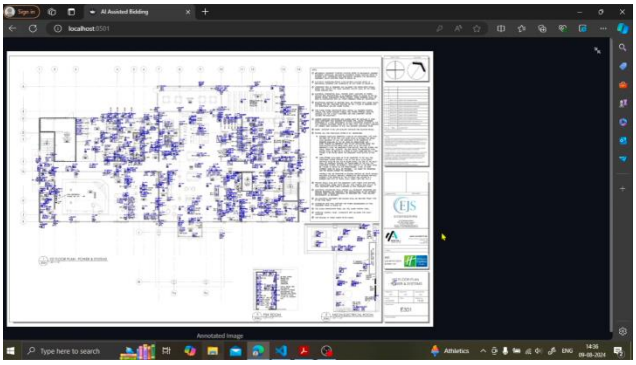


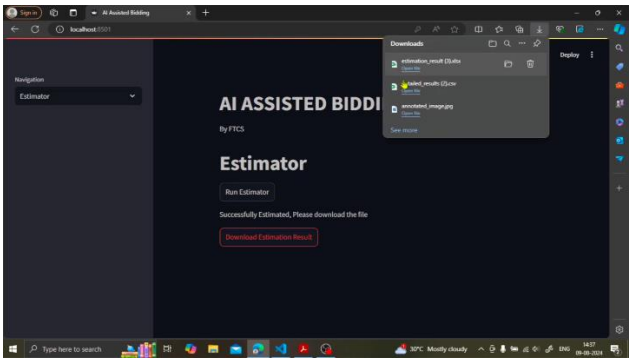


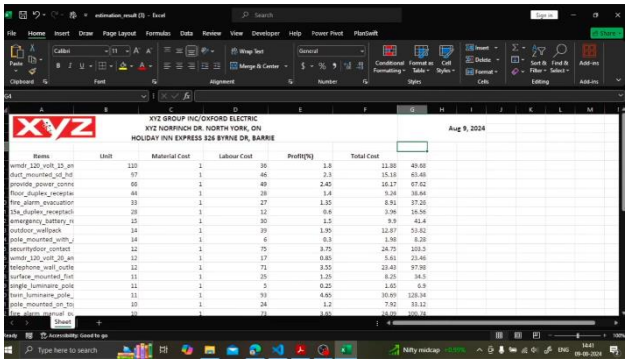












6. Current Research & Development

After successfully completing the product for floor plan legend detection, I became increasingly motivated to dive deeper into **computer vision** and explore cutting-edge techniques in **object detection**. My goal is to design and develop a novel architecture that competes with state-of-the-art models, particularly for **small object detection** with improved accuracy.

6.1 Literature Review

To build a strong foundation, I began by thoroughly studying some of the most important papers and architectures in the field of computer vision. Below is a list of key research papers I have read, along with a brief overview of each:

1. **YOLOv1** – Introduced the idea of a single-stage object detection model, pioneering real-time object detection.
2. **YOLOv2** – Improved accuracy with multi-scale training and better anchor box design.
3. **YOLOv3** – Enhanced detection with feature pyramid networks for better detection at different scales.
4. **YOLOv4** – Improved efficiency and accuracy with the introduction of CSPNet and optimized backbones.
5. **YOLOv5** – Further refined the architecture and introduced various training optimizations.
6. **YOLOv6** – Focused on lightweight model design for mobile and edge devices.

7. **YOLOv7** – Introduced **Extended ELAN (E-ELAN)** to enhance multi-scale feature extraction.
8. **YOLOv8** – Latest iteration with further improvements in detection performance and backbone optimization.
9. **EfficientNet v1** – Introduced a family of models that scale network width, depth, and resolution efficiently.
10. **EfficientNet v2** – Refined model scaling with faster training and improved accuracy.
11. **EfficientNet v3** – Extended scaling strategies for higher accuracy while maintaining efficiency.
12. **Depthwise Convolutions** – Efficient operations used in models like MobileNet for reducing computational cost.
13. **MobileNet v1** – Introduced depthwise separable convolutions for mobile applications.
14. **MobileNet v2** – Added inverted residuals and linear bottlenecks for enhanced efficiency.
15. **MobileNet v3** – A highly efficient model designed for mobile devices, combining depthwise convolutions and attention mechanisms.
16. **RCNN** – A region-based convolutional neural network that led to significant improvements in object detection.
17. **Fast-RCNN** – Optimized version of RCNN that reduces computation and speeds up training.
18. **Mask-RCNN** – Extended Fast-RCNN to include instance segmentation alongside object detection.
19. **FPN (Feature Pyramid Network)** – Aggregates multi-scale features to improve object detection across different object sizes.
20. **RPN (Region Proposal Network) & ROI Pooling** – Efficient object region proposal mechanism for detection models.
21. **Attention Is All You Need** – Introduced the transformer architecture that revolutionized deep learning, particularly in vision tasks.
22. **Quantization** – Reducing model precision to optimize speed and memory usage, essential for deployment.
23. **CSPNet (Cross-Stage Partial Network)** – Efficiently splits the feature map for better gradient flow and optimized computation.
24. **PANNet (Path Aggregation Network)** – Aggregates features from different levels in both bottom-up and top-down pathways.
25. **Dynamic Label Assignment** – Adaptive label assignment during training for better detection results.
26. **Progressive Training Quantization** – Training strategies focused on gradually introducing quantization to improve model performance.
27. **Auto-NAS (Neural Architecture Search)** – Automated architecture search that optimizes network design.
28. **Vision Transformer (ViT)** – Applies transformer models to image classification tasks, focusing on self-attention mechanisms.

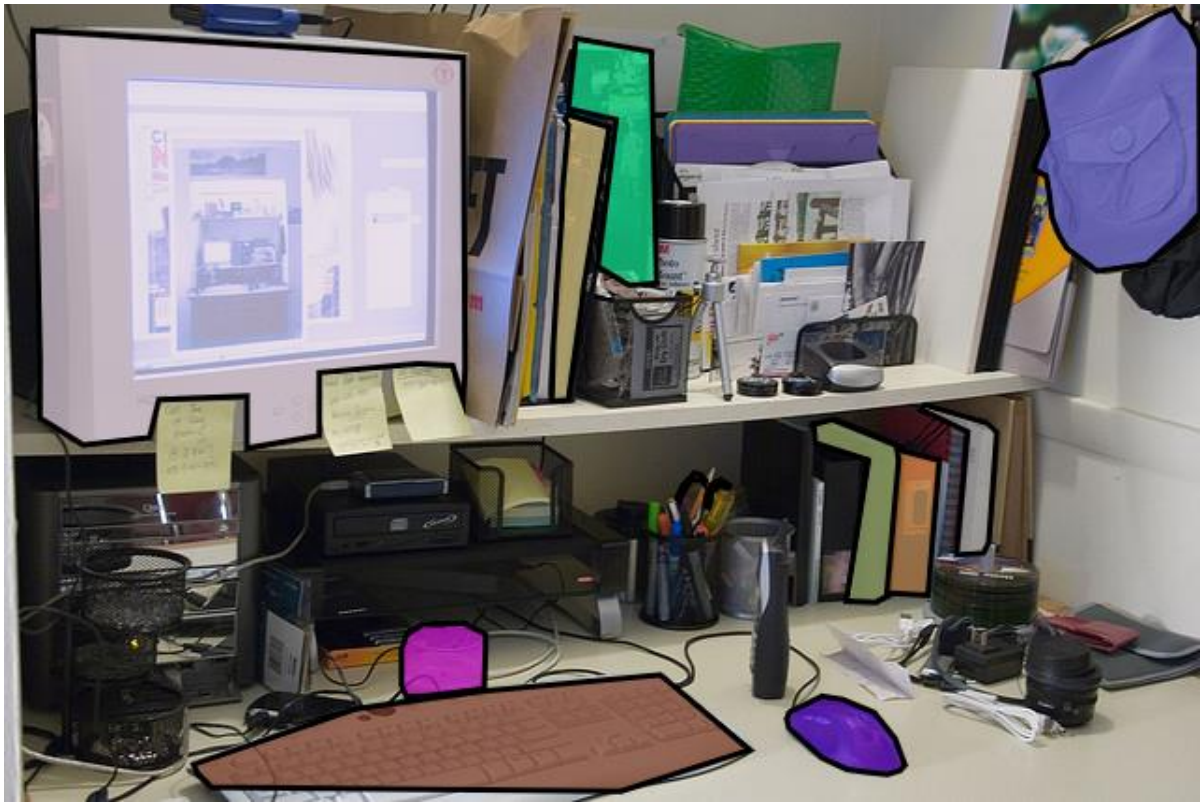
6.2 Activation Map Visualization

To gain a more intuitive understanding of how these architectures perform, I conducted a detailed analysis of their **activation maps**. Using **TensorFlow** and **Keras**, I passed a custom image of very high resolution (3000x3000 pixels), with small objects and noise, through various backbones such as **CSPDarknet**, **YOLOv8**, and **ViT**. The goal was to evaluate how well each model performed in extracting small object details under different conditions like occlusion.

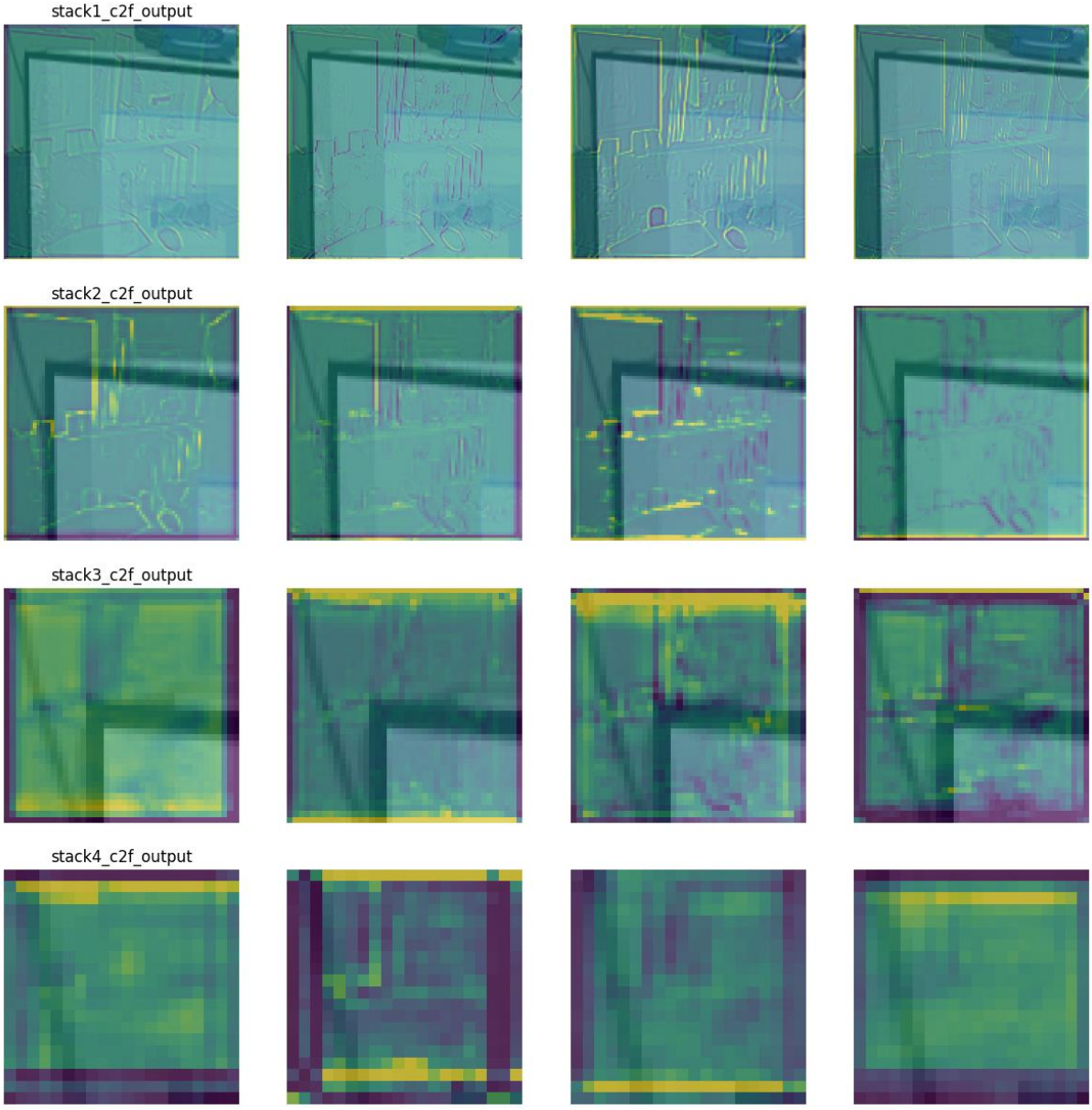
- **YOLOv8**: Struggled to capture fine-grained edge details, especially for small objects.
- **Vision Transformer (ViT)**: Showed the weakest performance in feature extraction for small objects, likely due to its global attention mechanism.
- **EfficientNet**: Performed better than YOLOv8 in terms of fine-grained details but still missed some crucial features.
- **CSPDarknet**: Outperformed other backbones with superior small object detection, showcasing strong feature retention from early layers.

This analysis revealed that the **cross-stage partial connections** in **CSPDarknet** significantly enhanced feature retention, especially for small object detection. This understanding laid the foundation for my custom architecture, which I will discuss in the next section.

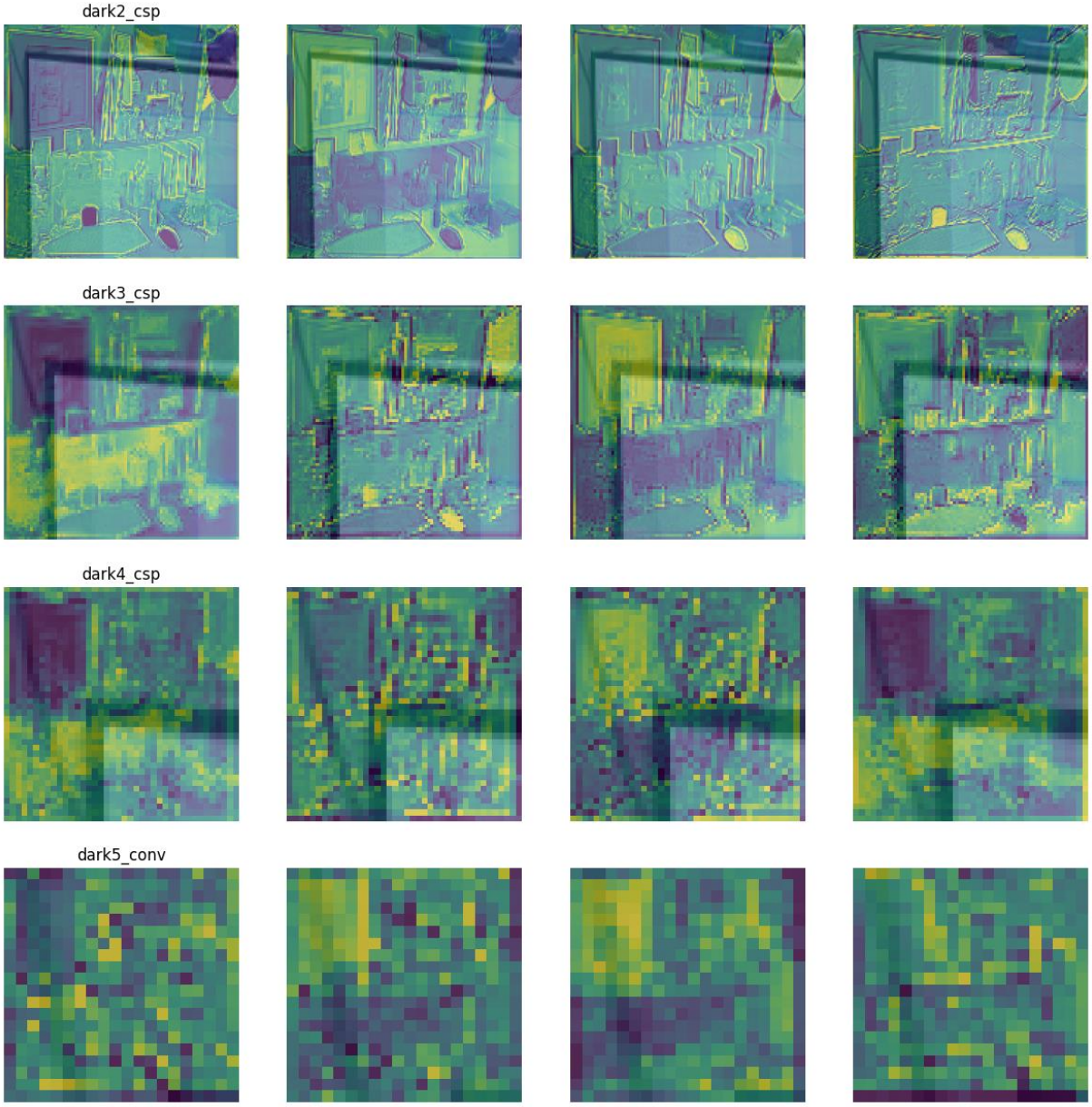
Input Image 1: (From COCO 2017 Dataset) – 640x640 pixel square



a) YOLOv8 backbone (Output across various depths/stages of backbone)

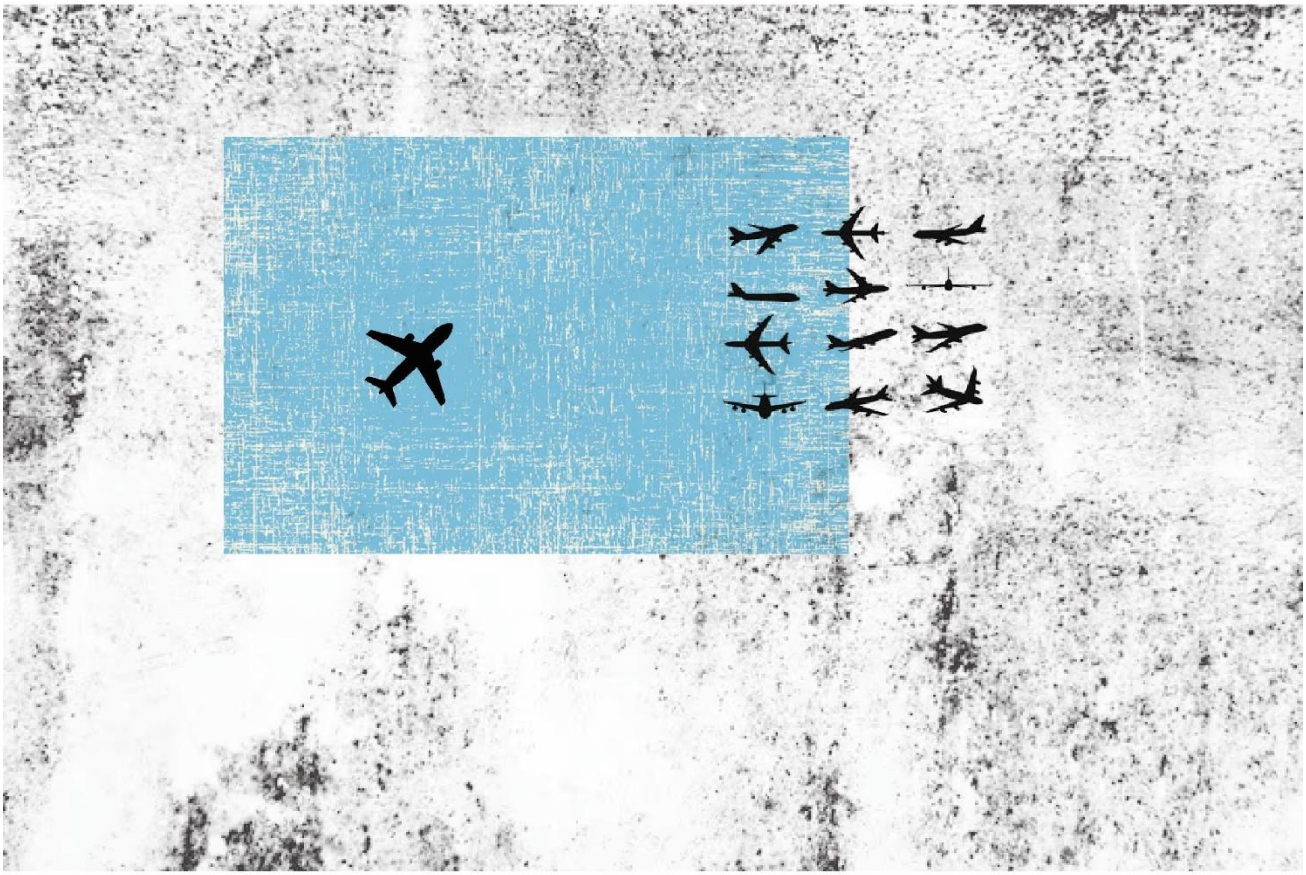


b) CSPDarknet53



*It is clearly evident that the retainment feature of the Cross stage partial connection is helping hold the fine grain features deep in the backbone.

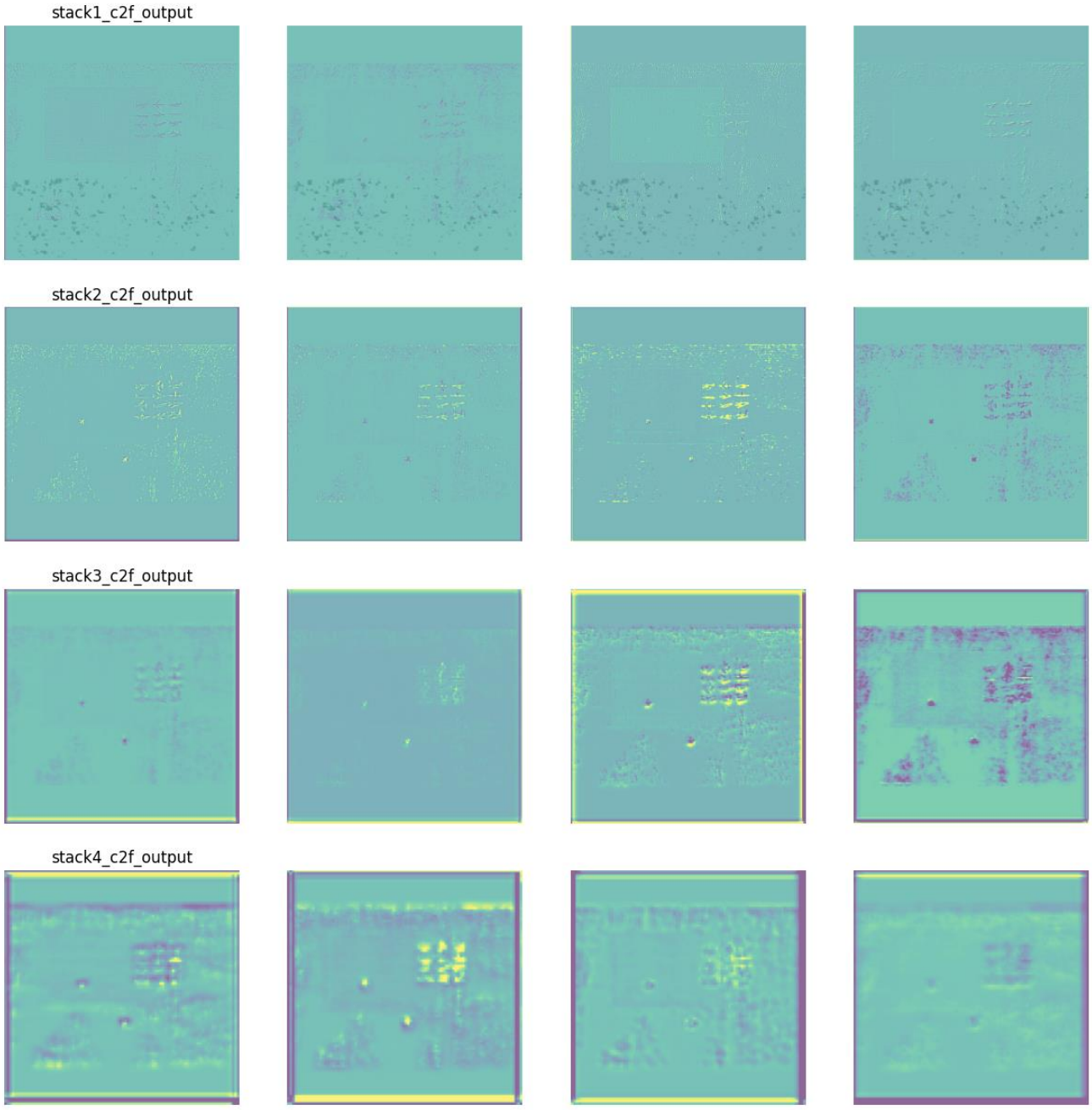
<u>Input Image 2:</u> (Custom made using Adobe Illustrator) – 3000x3000 pixel square



*Different Noisy textures to recreate occlusion.

*Icon sizes averages to 120x120 pixel square

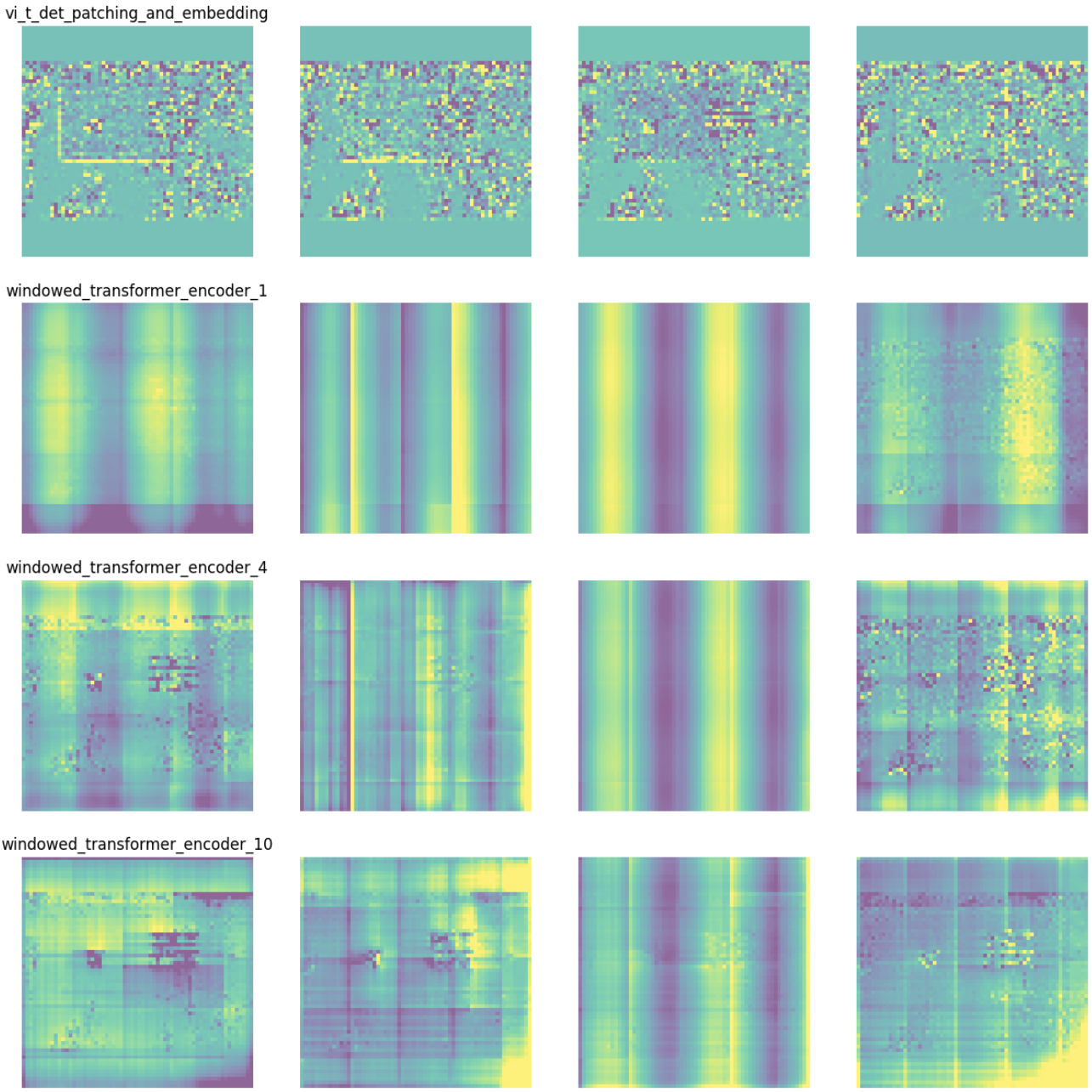
a) YOLOv8 backbone:



*Clearly visible that there is very less feature retainment from the initial stage itself.

* This proves why small object detection become less accurate using YOLOv8 as it focuses more towards "Real-Time" object detection

b) Vision Transformer backbone:

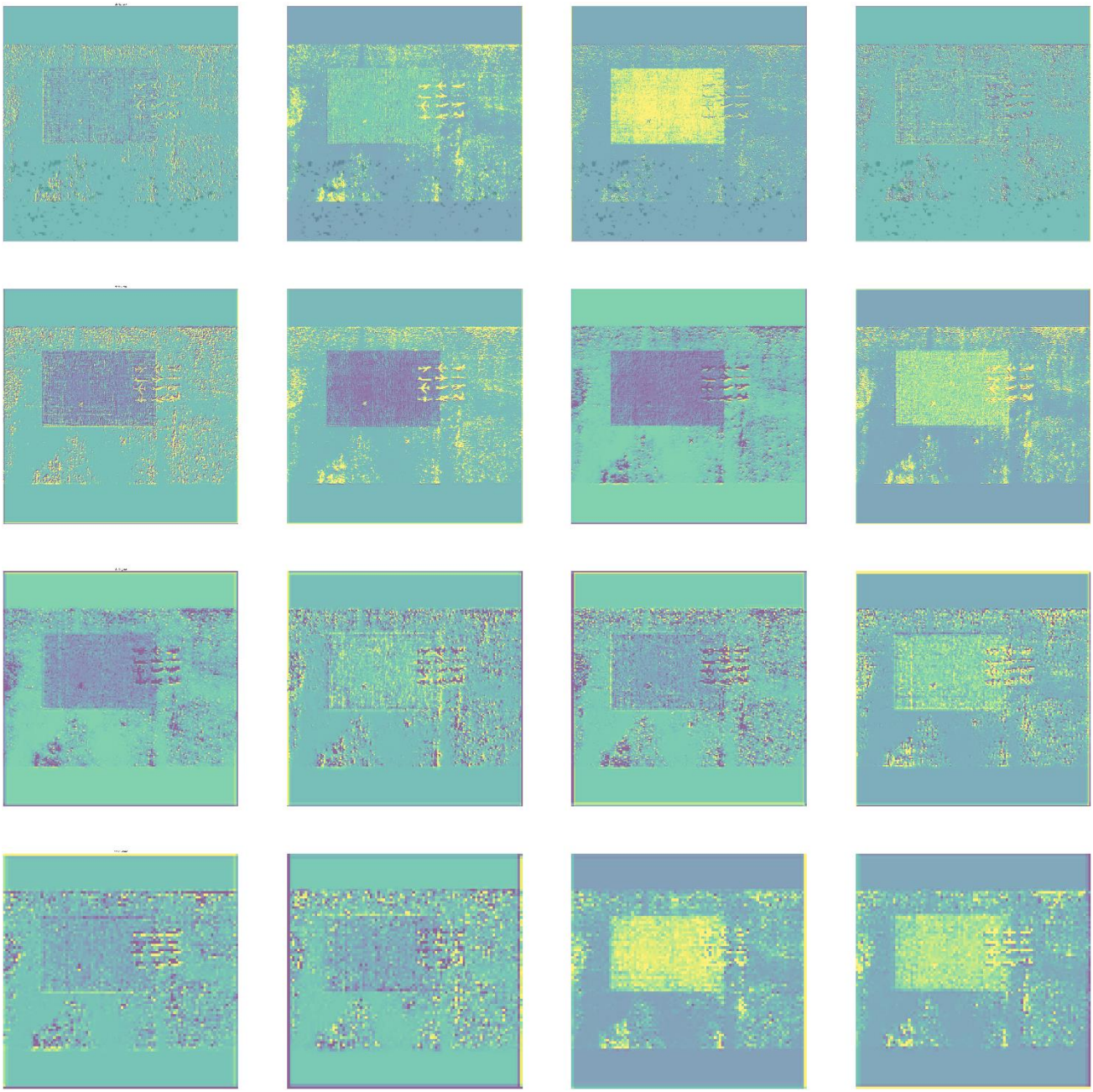


*Evident that this model is completely struggling with heavy resolution images

c) Mix-Transformer backbone:

*Similar to Vision Transformer model

d) CSPDarkNet53 backbone: (Cross-stage partial connections)



*This proves that the features are being retained in the depths of backbones with decent amount of extraction of relevant features.

*Zoom in to see the fine grain features.

7. Indigenously Developed Model Prototype

7.1 Designing My Own Backbone

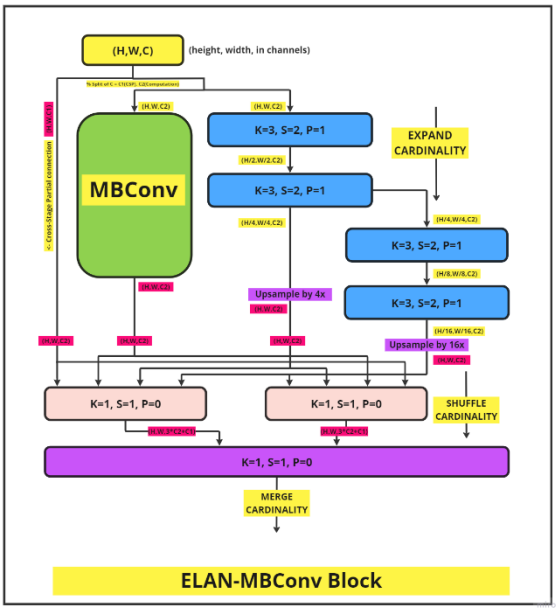
After analyzing the **CSPDarknet** structure, I was inspired to design a novel backbone architecture, combining elements from **YOLOv8**, **CSPDarknet**, **Extended-ELAN**, and **MBConv** from EfficientNet. My design focuses on optimizing feature extraction while maintaining computational efficiency.

The key building block in my architecture is the **ELAN_MBConv Block**, which is designed as follows:

- **ELAN Configuration**: Leverages **Cross-Stage Partial (CSP) connections** to retain information from a portion of the input channels, while the remaining channels are passed through MBConv blocks for more complex computations.
- **MBConv Block**: Ensures efficient computation with **inverted residuals**, improving both feature extraction and backpropagation efficiency.
- **Shuffle Cardinality**: Introduces an information shuffling mechanism to blend processed and retained features, enhancing the model's generalization capabilities.

Key features of this block:

1. **Information Retention**: A portion of the input channels is retained to prevent loss of crucial features.
2. **Efficient Backpropagation**: MBConv's inverted residual structure facilitates an efficient backpropagation path.
3. **Complex Computation**: Ensures the model can handle high-resolution images with fine-grained details.
4. **Shuffle Cardinality**: Promotes a robust mixing of retained and processed information, leading to superior feature aggregation.

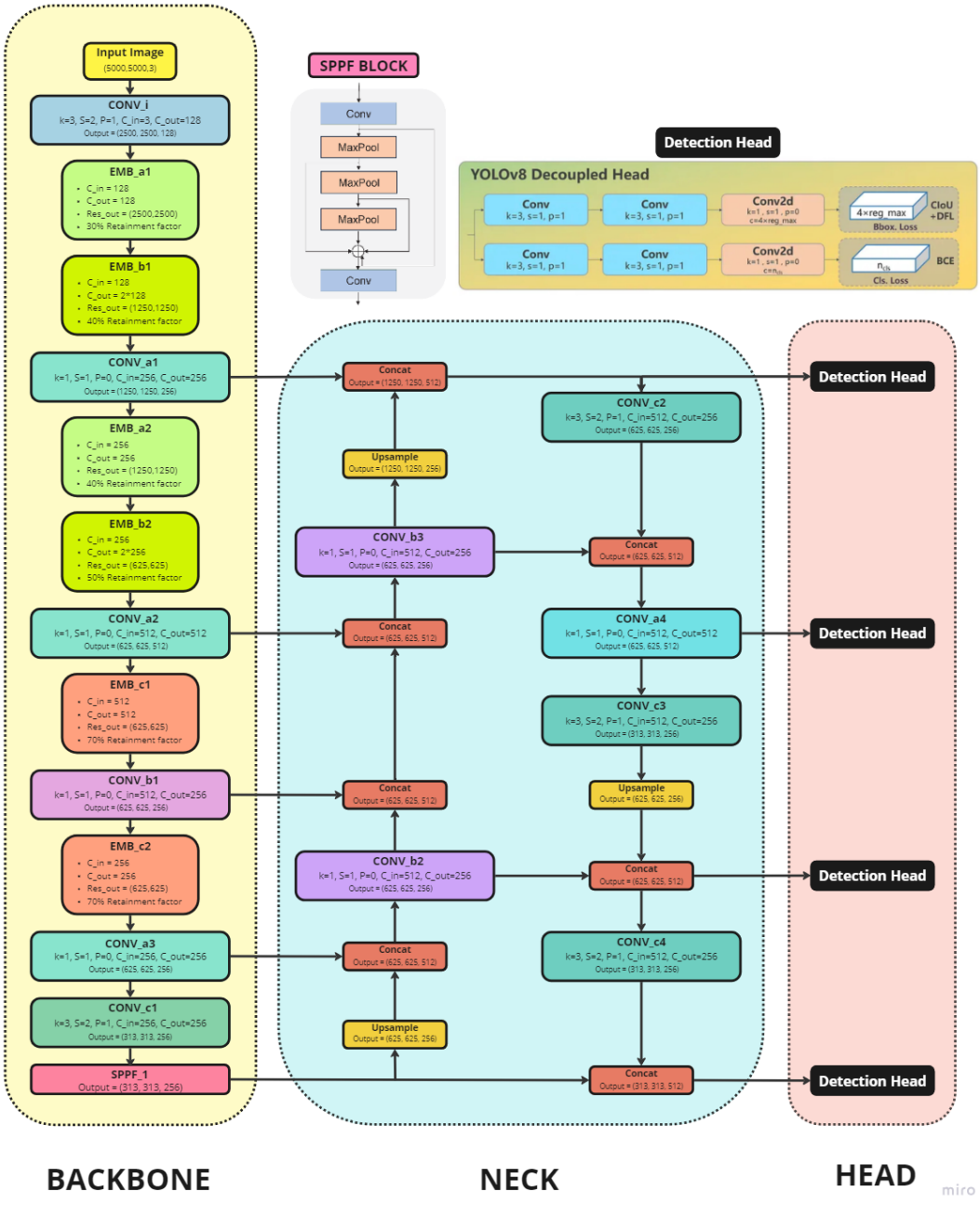


7.2 Proposed Model Architecture

Inspired by the **YOLOv8** structure (backbone, neck, and head), I replaced the backbone with my newly designed **ELAN_MBConv Block**. The full architecture is as follows:

- **Backbone**: Based on the ELAN_MBConv Block, with transition layers and a **Spatial Pyramid Pooling (SPP) layer** for multi-scale feature aggregation.
- **Neck**: Combines **PANNet** and **YOLOv8** features for bottom-up and top-down feature aggregation, improving small object detection across different scales.
- **Decoupled Head**: Implements separate heads for **detection** and **classification**, optimizing the output for bounding box regression and object classification.

The architecture is under active development, incorporating advanced training techniques like **reparameterization** and **quantization** for improved inference speed and accuracy.



8. Results & Conclusion

8.1 Activation Map Analysis

The detailed analysis of activation maps yielded important insights:

1. **YOLOv8**: While effective for larger objects, it struggled with fine-grained details and edges, impacting small object detection.
2. **Vision Transformer (ViT)**: Its global attention mechanism was ineffective for small object features, resulting in poor performance in object detection tasks.
3. **EfficientNet**: Provided a balanced trade-off between efficiency and accuracy but fell short in capturing intricate details.
4. **CSPDarknet**: Outperformed other backbones with its deep feature extraction, maintaining small object details from early layers.

8.2 Proposed Model Performance

The new **ELAN_MBConv backbone** has shown promising results during early-stage testing. The architecture's ability to retain crucial information through **CSP connections**, combined with the efficient processing power of **MBConv**, allows for:

- Better detection of small objects.
- Improved handling of complex, high-resolution images.
- Faster convergence during training due to the efficient backpropagation paths provided by **inverted residuals**.

8.3 Conclusion

My research has led to the development of a novel architecture prototype that combines **CSPDarknet**'s information retention capabilities with the efficient computation of **MBConv, E-ELAN** and the feature aggregation power of **PANNet**. The resulting model is designed to achieve higher accuracy in **small object detection** while maintaining computational efficiency.

The next phase will involve extensive testing and optimization, incorporating **quantization techniques** and **progressive training strategies** to further enhance the model's performance.

9. References

1. Redmon, J., Divvala, S., Girshick, R., & Farhadi, A. (2016). *You Only Look Once: Unified, Real-Time Object Detection*. In Proceedings of the IEEE conference on computer vision and pattern recognition (pp. 779-788).
2. Redmon, J., & Farhadi, A. (2017). *YOLO9000: Better, Faster, Stronger*. In Proceedings of the IEEE conference on computer vision and pattern recognition (pp. 7263-7271).
3. Redmon, J., & Farhadi, A. (2018). *YOLOv3: An Incremental Improvement*. arXiv preprint arXiv:1804.02767.
4. Bochkovskiy, A., Wang, C.-Y., & Liao, H.-Y. M. (2020). *YOLOv4: Optimal Speed and Accuracy of Object Detection*. arXiv preprint arXiv:2004.10934.
5. Jocher, G., et al. (2020). *YOLOv5: Release v1.0*. GitHub. https://github.com/ultralytics/yolov5
6. Meituan. (2022). *YOLOv6: A Single-Stage Object Detection Framework for Industrial Applications*. arXiv preprint arXiv:2209.02976.
7. Wang, C.-Y., Bochkovskiy, A., & Liao, H.-Y. M. (2022). *YOLOv7: Trainable Bag-of-Freebies Sets New State-of-the-Art for Real-Time Object Detectors*. arXiv preprint arXiv:2207.02696.
8. Ultralytics. (2023). *YOLOv8: Ultralytics YOLOv8 in PyTorch*. GitHub. https://github.com/ultralytics/ultralytics
9. Tan, M., & Le, Q. (2019). *EfficientNet: Rethinking Model Scaling for Convolutional Neural Networks*. In Proceedings of the 36th International Conference on Machine Learning (pp. 6105-6114).
10. Tan, M., & Le, Q. (2021). *EfficientNetV2: Smaller Models and Faster Training*. In International Conference on Machine Learning (pp. 10096-10106).
11. Tan, M., & Le, Q. (2021). *Scaling Up EfficientNet Architectures with Efficient Scaling Strategies*. arXiv preprint arXiv:2104.00298.
12. Howard, A. G., Zhu, M., Chen, B., Kalenichenko, D., Wang, W., Weyand, T., ... & Adam, H. (2017). *MobileNets: Efficient Convolutional Neural Networks for Mobile Vision Applications*. arXiv preprint arXiv:1704.04861.
13. Sandler, M., Howard, A., Zhu, M., Zhmoginov, A., & Chen, L.-C. (2018). *MobileNetV2: Inverted Residuals and Linear Bottlenecks*. In Proceedings of the IEEE conference on computer vision and pattern recognition (pp. 4510-4520).
14. Howard, A., Wang, W., Chen, B., Kalenichenko, D., Weyand, T., Andreetto, M., & Adam, H. (2019). *Searching for MobileNetV3*. In Proceedings of the IEEE/CVF international conference on computer vision (pp. 1314-1324).
15. Girshick, R. (2015). *Fast R-CNN*. In Proceedings of the IEEE international conference on computer vision (pp. 1440-1448).
16. He, K., Gkioxari, G., Dollár, P., & Girshick, R. (2017). *Mask R-CNN*. In Proceedings of the IEEE international conference on computer vision (pp. 2961-2969).

17. Lin, T.-Y., Dollar, P., Girshick, R., He, K., Hariharan, B., & Belongie, S. (2017). *Feature Pyramid Networks for Object Detection*. In Proceedings of the IEEE conference on computer vision and pattern recognition (pp. 2117-2125).

18. Ren, S., He, K., Girshick, R., & Sun, J. (2015). *Faster R-CNN: Towards Real-Time Object Detection with Region Proposal Networks*. In Advances in neural information processing systems (pp. 91-99).

19. Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., ... & Polosukhin, I. (2017). *Attention is All You Need*. In Advances in neural information processing systems (pp. 5998-6008).

20. Jacob, B., Kligys, S., Chen, B., Zhu, M., Tang, M., Howard, A., ... & Adam, H. (2018). *Quantization and Training of Neural Networks for Efficient Integer-Arithmetic-Only Inference*. In Proceedings of the IEEE conference on computer vision and pattern recognition (pp. 2704-2713).

21. Wang, C.-Y., Liao, H.-Y. M., Wu, Y.-H., Chen, P.-Y., Hsieh, J.-W., & Yeh, I.-H. (2020). *CSPNet: A New Backbone That Can Enhance Learning Capability of CNN*. In Proceedings of the IEEE/CVF conference on computer vision and pattern recognition workshops (pp. 390-391).

22. Liu, S., Qi, L., Qin, H., Shi, J., & Jia, J. (2018). *Path Aggregation Network for Instance Segmentation*. In Proceedings of the IEEE conference on computer vision and pattern recognition (pp. 8759-8768).

23. Ghiasi, G., Lin, T.-Y., & Le, Q. V. (2021). *NAS-FPN: Learning Scalable Feature Pyramid Architecture for Object Detection*. In Proceedings of the IEEE conference on computer vision and pattern recognition (pp. 7036-7045).

24. Dosovitskiy, A., Beyer, L., Kolesnikov, A., Weissenborn, D., Zhai, X., Unterthiner, T., ... & Houlsby, N. (2020). *An Image is Worth 16x16 Words: Transformers for Image Recognition at Scale*. arXiv preprint arXiv:2010.11929.

25. Mei, Y., et al. (2021). *Symbol Spotting on Digital Architectural Floor Plans Using a Deep Learning-Based Framework*. In Proceedings of the IEEE/CVF International Conference on Computer Vision Workshops (pp. 1001-1007).

26. Zhao, H., et al. (2019). *Dynamic Label Assignment for Object Detection*. In Advances in Neural Information Processing Systems.

27. Han, S., Pool, J., Tran, J., & Dally, W. J. (2015). *Learning Both Weights and Connections for Efficient Neural Networks*. In Advances in Neural Information Processing Systems (pp. 1135-1143).

10. Certificate

I would like to express my gratitude to **FTCS** **(Flasho Tech Consultancy Services)** for providing me the opportunity to work on such an exciting and challenging subject. FTCS is a forward-thinking tech consultancy, specializing in AI, machine learning, software development, and cloud infrastructure, helping businesses innovate and scale through cutting-edge solutions.

FTCS offered me full flexibility, resources, and support, allowing me to deeply explore advanced topics in machine learning and object detection. This freedom, combined with their commitment to fostering innovation, has been instrumental in my research. I'm grateful for their trust and encouragement, which has significantly enriched my professional growth and enabled me to contribute meaningfully to both academic and practical applications.