Annual Rport

Contents

[*Abstract* 2](#_Toc189125692)

[Introduction 2](#_Toc189125693)

[Related Works 4](#_Toc189125694)

[1. Line detector : 5](#_Toc189125695)

[1.1 Proposed Method 5](#_Toc189125696)

[1.1.1 Data Preperation : 7](#_Toc189125697)

[1.1.1.1 Megatile: 8](#_Toc189125698)

[1.1.1.2 Physical cell : 10](#_Toc189125699)

[1.1.1.3 Sampling : 12](#_Toc189125700)

[1.1.1.4 Augmentation 16](#_Toc189125701)

[1.2 Detection model ( light weight ) 17](#_Toc189125702)

[1.2.1 Model Evaluation 20](#_Toc189125703)

[1.3 Alignment 24](#_Toc189125704)

[1.4 Image Restoration : 25](#_Toc189125705)

[1.4.1 Unet : 25](#_Toc189125706)

[1.4.1.2 AST 26](#_Toc189125707)

[1.4.1.2.1 Training Setup and Performance Evaluation : 26](#_Toc189125708)

[2. Block detector : 29](#_Toc189125709)

[2.1 Data prepration 29](#_Toc189125710)

[2.2 model evaluation 31](#_Toc189125711)

[Future work 34](#_Toc189125712)

[References 34](#_Toc189125713)

### *Abstract*

**Line segments play a crucial role in understanding the structure of human-made environments, serving as a compact and efficient representation of scene geometry. Despite their importance, robust and accurate line detection remains a challenge, particularly in real-time applications such as autonomous driving, SLAM, and 3D reconstruction. Traditional methods, while computationally efficient, often lack robustness in challenging conditions, whereas deep learning-based approaches, though more robust, struggle with computational efficiency and precise endpoint localization.**

**In this work, we propose a novel approach that combines the strengths of deep learning and traditional methods to achieve robust and accurate line segment detection. We leverage a labeled dataset, Aerialytic, consisting of aerial images of houses, with annotations including keypoints and lines defining roof structures. To address the challenges of processing large-scale aerial imagery, we design a customized F11-based model, incorporating a dataset class for efficient sampling from megatiles and a training pipeline optimized for high precision. To further enhance the quality of detected lines, we integrate advanced restoration networks, including Attention U-Net and AST (Adaptive Sparse Transformer with Attentive Feature Refinement), which refine features and focus on spatially relevant details.**

**Beyond line detection, our work also focuses on keypoint pose estimation, where we aim to identify keypoints that define the edges of houses and the spatial boundaries of their structures. Additionally, we introduce a system capable of distinguishing between four distinct block diagram types for each house, enabling further understanding of the architectural layout.**

**This hybrid approach combines the efficiency of F11 model with the refinement capabilities of restoration networks and the precision of keypoint pose estimation, resulting in a robust and accurate line detection system suitable for real-time applications.**

Keywords—  Line Segment Detection, Attention U-Net, Adaptive Sparse Transformer (AST), Aerial Imagery, Real-Time Applications, Block detector.

# Introduction

Line segments are fundamental elements in human-made environments, encapsulating the underlying structure of a scene in a compact and efficient manner. Due to their spatial extent and presence even in textureless regions, line features serve as a valuable complement to point-based features in various computer vision tasks. These tasks include 3D reconstruction, Structure-from-Motion (SfM) [1][2], Simultaneous Localization and Mapping (SLAM)[3][4], visual localization [5], object tracking [6], and vanishing point estimation [7]. The robustness and accuracy of line segment detection are critical for these applications, as they directly impact the quality of the extracted features and the success of downstream tasks.

Traditionally, line segments have been detected using handcrafted methods such as the Line Segment Detector (LSD) [8], which relies on image gradients and low-level details. While these methods are computationally efficient and accurate under ideal conditions, they often struggle in challenging scenarios, such as low illumination or noisy environments, where the image gradient becomes unreliable. Additionally, traditional methods lack global contextual understanding, leading to the detection of irrelevant or noisy lines that do not contribute meaningfully to the scene structure.

Recent advancements in deep learning have opened new avenues for addressing these limitations [9], [10]. Deep learning-based line detection methods, such as deep wireframe parsing techniques[11] [12], have demonstrated the ability to infer structural lines in indoor scenes[13],[14]. More generic deep line segment [15] and joint line detectors with descriptors [16], [17] have also been proposed[18]. These methods leverage large receptive fields to encode image context, enabling them to distinguish between meaningful and noisy lines. However, most of these approaches are fully supervised and rely on limited datasets, such as the Wireframe dataset[9], which is biased toward indoor structural lines and lacks diversity for training generic line detectors. Furthermore, while deep learning methods show promise in handling challenging conditions, they often fall short in terms of accuracy and endpoint localization compared to traditional methods, particularly in straightforward scenarios.

The growing demand for autonomous robots and vision-based systems has further emphasized the need for robust and efficient line detection algorithms. Tasks such as scene recognition[19], SLAM [20], autonomous driving[21], and 3D reconstruction rely heavily on accurate feature matching and mapping across different views of a scene. While interest points with descriptors[22], [23] and descriptive lines [16] [24] are commonly used for these tasks, the performance of line detection remains a bottleneck for real-time robotic applications. Traditional methods like LSD [8] and Hough Transform [25] with Canny edge detection [26] are computationally efficient but sensitive to illumination changes and require manual parameter tuning, leading to inconsistent results. Although some recent traditional methods [27] have improved upon LSD, they still lack the perceptual ability and adaptability required for dynamic environments.

On the other hand, learning-based methods, such as LCNN [12] and LETR [28], offer greater robustness but are often computationally expensive, making them unsuitable for real-time applications. For instance, LCNN detects line junctions and connects lines from points, achieving high accuracy at the cost of slow inference speeds. Similarly, LETR, while faster, still struggles to meet real-time requirements. Other approaches, such as[29], focus on infinitely long semantic line detection using Hough Transform and deep learning, but they are not suitable for detecting finite line segments due to the inherent limitations of Hough Transform.

In recent years, architectures like YOLO (You Only Look Once)[30], U-Net [31], and Attention U-Net [32]have gained significant attention for their effectiveness in various vision tasks. YOLO, known for its real-time object detection capabilities, has been adapted for line detection tasks due to its efficiency and ability to process images in a single forward pass. U-Net, originally designed for biomedical image segmentation, has been widely adopted for its encoder-decoder structure, which captures both local and global features effectively. Attention U-Net extends this by incorporating attention mechanisms, allowing the model to focus on relevant regions of the image, which is particularly useful for detecting fine details like line segments. These architectures have shown promise in improving the accuracy and robustness of line detection, especially in complex and noisy environments.

Based on this assessment, we propose in this work to keep the best of both worlds: use deep learning to process the image and discard unnecessary details, then use handcrafted methods to detect the line segments. We thus retain the benefits of deep learning, namely, to abstract the image and gain more robustness to illumination and noise, while at the same time retaining the accuracy of classical methods.

To achieve this, we first work with a labeled dataset, Aerialytic, which consists of aerial images of houses, where the labels include keypoints and lines defining the roof structures. We designed a customized F11 model for this task, which includes a dataset class for sampling images from large megatiles and a training pipeline optimized for high precision. To further improve the model's output, we integrated restoration networks such as Attention U-Net and AST (Adaptive Sparse Transformer with Attentive Feature Refinement)[33]. These networks enhance the quality of detected lines by refining features and focusing on relevant spatial details. Our approach combines the efficiency of F11 with the refinement capabilities of advanced restoration networks, resulting in a robust and accurate line detection system suitable for real-time applications.

# Related Works

Line segment detector (LSD)[8], a classic hand-crafted line detector, remains the first preference for most line detection tasks in the robotics field. It relies on a region-growing strategy and runs extremely fast even without GPU to achieve real-time detection. In LSD, pixels with similar image gradients are grouped together to form a region and then converted to line segments by an approximated rectangle. As a step forward, Cho et al. [34]not only utilized gradient but also other information such as brightness and gradient intensity to further improve the performance. Some other variants like PLSD [27] developed line merging strategies but sometimes tend to over-segment. An obvious limitation of the handcrafted approaches is that the detection is based on low-level information like image gradient and brightness, leading to the unawareness of higher-level semantic information.

A classic learning-based algorithm is LCNN[12], which detects endpoints and learns to connect them and form line segments, making no use of edges and relying heavily on endpoint detection. The learned network pairs the detected endpoints to form line segments when the score exceeds a certain threshold. LCNN has achieved high-quality line segment detection on widely-used datasets such as Wireframe [9] and YorkUrban[35]. However, if a large number of endpoints exist, the inference time increases significantly, making stable real-time detection not possible. HAWP[36] optimizes efficiency based on LCNN, achieving similar results in less time, but is still too computationally expensive to support real-time detection. Unlike previous methods, LETR detects lines as entities and represents them similar to the diagonal line of a bounding box produced by a vectorized line segment predictor [37], converted from the box predictor in DETR [38]. This method has achieved state-of-the-art results on the Wireframe dataset under certain metrics, enabled end to-end training of line segment detection, and reduced runtime variance compared to previous methods

Zhao et al. [29] proposed a hybrid pipeline to achieve semantic line detection with Hough Transform. By voting edge-like information, a line is determined by the highestvoted angle and position. A differentiable Hough Transform is used to make the pipeline learnable. Although Hough Transform works well for a few straight lines, it offers a view too global for dense line segment detection. There are also learnable versions of LSD like LSDNet [39] improving LSD by introducing learned line heat maps and orientation maps, but it is based on roughly predicted orientation instead of only edges. To solve the problems above, AirLine achieves stabler short-line detection by limiting the voting region to segments of continuous edge and accurate orientation detection by hand-crafted orientation detectors on edges.

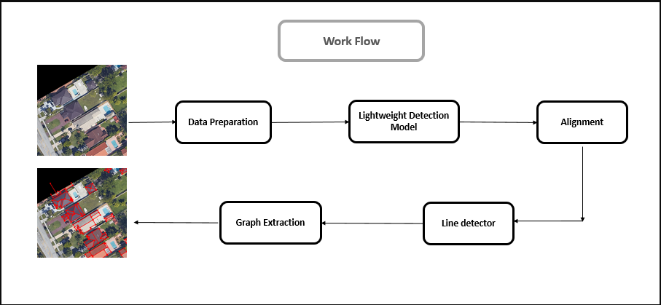
# Line detector :

# 1.1 Proposed Method

YOLO has revolutionized the field of real-time object detection with its ability to process images in a single forward pass, achieving remarkable speed and accuracy. Building on this legacy, F11 object detection model represents an advanced iteration of the F11 object detection model architecture, specifically designed to handle more complex tasks such as object detection and keypoint pose estimation simultaneously. Unlike traditional models that focus solely on bounding box predictions, F11 object detection model is customized to predict both the location of objects and their structural keypoints, making it a powerful tool for applications requiring detailed spatial understanding.

This customization allows F11 object detection model to excel in scenarios where precision and efficiency are paramount, such as autonomous navigation, 3D reconstruction, and robotic manipulation. By integrating advanced features like real-time visualization, distributed processing, and efficient batch management, F11 object detection model sets a new standard for performance in object detection and keypoint estimation tasks.

The F11 object detection model has been customized to address both object detection and keypoint pose estimation, tailored to meet the specific demands of each task. For object detection, the model is fine-tuned to accurately detect block-like structures, ensuring precise bounding boxes around each identified object. This capability is critical for applications requiring detailed spatial understanding, such as structural analysis or autonomous navigation. For keypoint pose estimation, the model is further adapted to predict not only the object’s location but also 22 distinct

Figure 1. Work Flow of F11 line Detector model

keypoints per object. Each keypoint is represented by its coordinates and visibility status, enabling a highly detailed structural understanding of the detected blocks. This dual functionality makes F11 object detection model a versatile tool for tasks requiring both object localization and fine-grained structural analysis.

To support these capabilities, our setup includes custom training and validation components designed for efficient data handling, distributed processing, and advanced batch management. These components ensure that the model can process large-scale datasets effectively while maintaining high accuracy. Additionally, real-time visualization functions have been integrated to provide immediate feedback during training and validation.

These functions save annotated images at various stages, allowing for continuous

monitoring and adjustment of the model’s performance. Together, these enhancements enable F11 object detection model to deliver precise object detection and detailed keypoint predictions, establishing it as an optimal solution for complex, structured datasets.

The workflow of this project is illustrated in Figure 1 and consists of several key stages. The process begins with Data Preparation, where megatiles and physical cells are created to organize and preprocess the input data. Sampling is then conducted to generate representative samples from the physical cells, ensuring that the dataset is both diverse and manageable. The next stage is Model Customization, where customized trainers and validators are developed to meet the specific requirements of the task. This stage ensures that the model is optimized for both object detection and keypoint pose estimation.

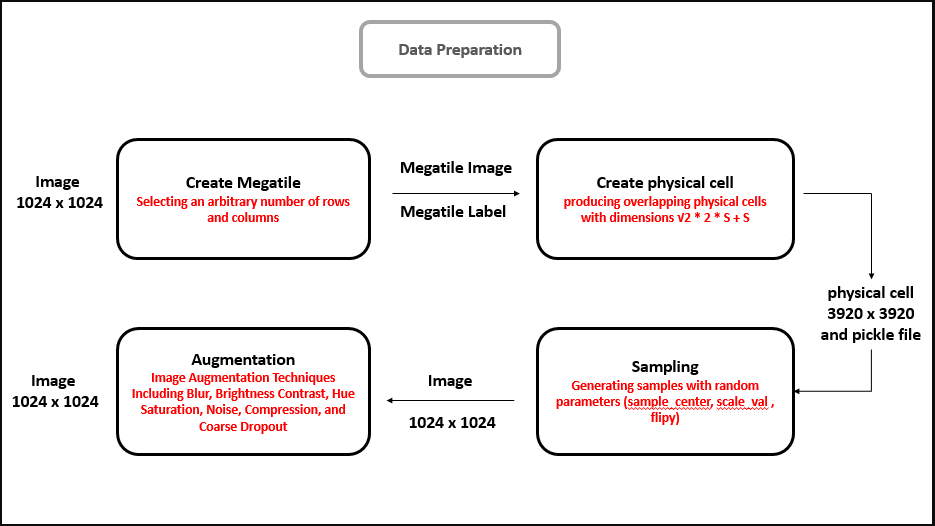
Following customization, the Model Training stage involves configuring training parameters to fine-tune the model’s performance. This step is critical for achieving high accuracy and robustness. Finally, Tensor Matching is performed using a U-Net model to refine the output. U-Net’s encoder-decoder architecture is particularly effective for this task, as it captures both local and global features, enhancing the precision of the detected objects and keypoints.

To further improve the quality of the output, we integrate advanced restoration networks, including Attention U-Net and AST (Adaptive Sparse Transformer with Attentive Feature Refinement). Attention U-Net enhances the model’s ability to focus on relevant regions of the image, improving the detection of fine details such as keypoints and edges. AST, on the other hand, leverages sparse attention mechanisms to refine features adaptively, ensuring that the model can handle complex and noisy environments effectively. These restoration networks work in tandem with F11 object detection model to deliver a robust and accurate solution for object detection and keypoint pose estimation.

By combining the efficiency of F11 object detection model with the refinement capabilities of Attention U-Net and AST, our approach achieves state-of-the-art performance in both object detection and structural understanding. This makes it a powerful tool for applications ranging from autonomous navigation to 3D reconstruction, where precision and robustness are paramount.

## Data Preperation :

Data preparation is a crucial step in training accurate and robust models for line segment detection, particularly when dealing with complex aerial imagery. To begin, aerial images are divided into megatiles, which represent large sections of the scene. Each megatile captures spatial information about rooftops and surrounding structures, enabling efficient sampling from a broader dataset. These megatiles serve as foundational units for labeling, where annotations, such as line segments and keypoints, define the geometric features of roofs. The purpose of creating megatiles is to manage the vast scale of aerial imagery and ensure that the dataset captures a wide variety of building layouts, rooflines, and spatial configurations.

Figure 2. Data Prepreation Diagram

From these megatiles, physical cells are generated. These smaller subdivisions break down the megatiles into localized sections, each containing detailed spatial features like roof edges, ridges, and junctions between building components. The creation of these physical cells ensures that the dataset can represent diverse roof geometries and structures across different building environments. Finally, sampling from these physical cells is performed, extracting various samples that include annotated line segments and keypoints. This step is essential for capturing variations in roof shapes, lighting conditions, and architectural features, which helps to create a comprehensive and diverse dataset suitable for training models capable of accurately detecting line segments in real-world scenarios.

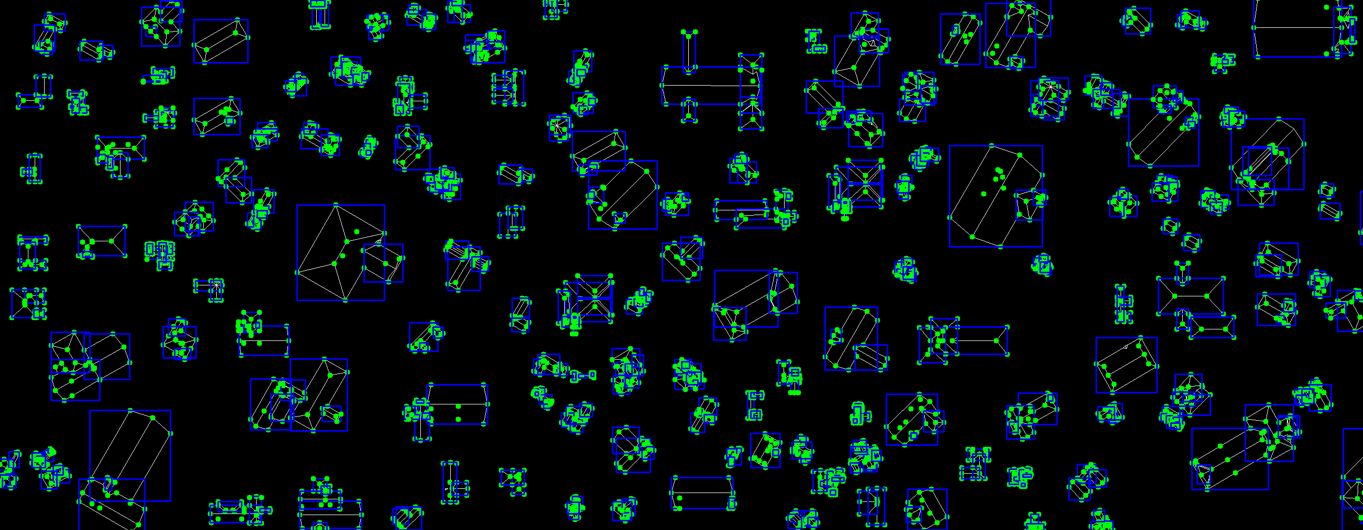
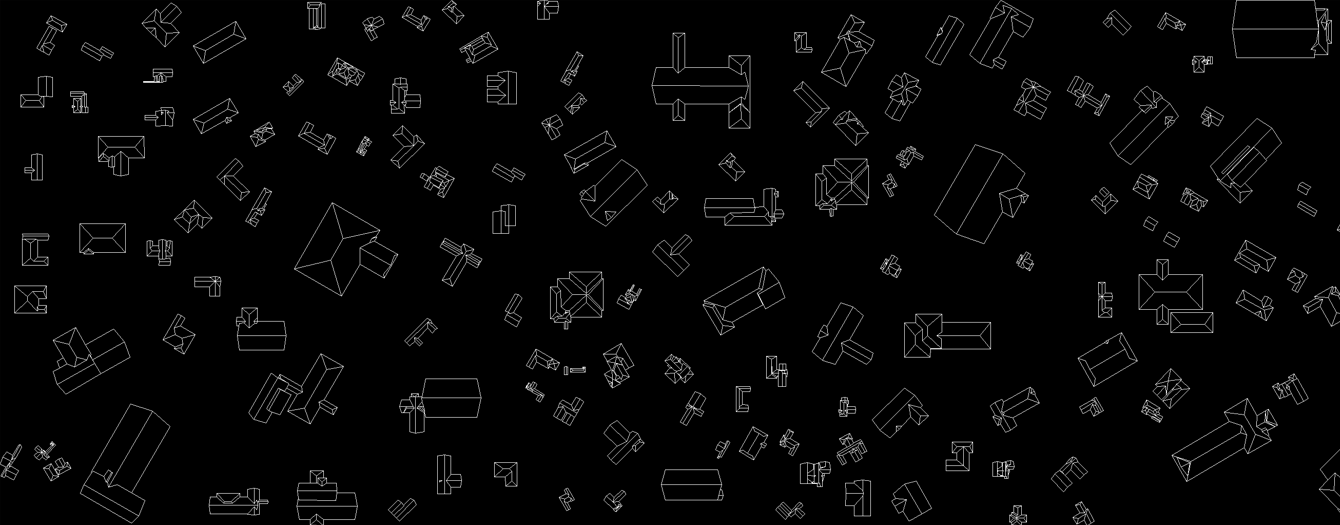
## 1.1.1.1 Megatile:

Images may initially be available as a single large image (megatile) or as smaller, separate images that need to be merged into a megatile for analysis or model training. The formation of megatiles is a crucial step in processing aerial or large-scale imagery, as it allows for efficient management and analysis of spatial data. This approach is particularly useful in computer vision tasks such as object detection, pose estimation, and 3D reconstruction, where integrating multiple images into a unified view enhances the accuracy and robustness of the model.

The process begins by selecting individual images, which are typically available as 1024x1024 pixel segments, and arranging them together to form a grid. Images are carefully aligned to ensure precise spatial relationships, maintaining the relative positions necessary for capturing features like object locations or keypoints. Once arranged, these images are pasted onto a blank canvas that serves as the base for constructing the megatile. This grid layout can vary depending on the task requirements, whether it’s a 3x3, 4x4, or larger grid. The result is a cohesive image that merges the selected segments into a complete view, forming a high-resolution megatile.

|  |
| --- |
| Pseudocode for Megatile Generation |
| Start  1. Initialize Parameters:   * Define the number of columns and rows for the megatile grid.   2. Select Images and Labels:   * Load all image and label files . * Check if the number of images is sufficient for the grid size. * Randomly select the required number of images and corresponding labels.   3. Create Megatile Canvas:   * Get the dimensions of each image (assume all are the same size). * Create a blank canvas with width = num columns × image width and   height = num rows × image height.  4. Assemble Megatile Image:   * for each image in the selected images and Calculate the row and column positions in the grid. * Save the megatile image to the output directory.   5. Create Megatile Labels:   * Initialize an empty list for megatile labels. * Calculate the grid position (row, column) of the corresponding image. * Adjust the bounding box coordinates to fit the megatile’s global coordinate system * Append the adjusted label to the megatile labels list.   6. Save the megatile labels to a text file in the output directory.  End |

In parallel, label files associated with each image are adjusted to reflect the new spatial arrangement. Labels containing bounding box coordinates are modified to account for the positioning within the megatile grid, ensuring that annotations remain accurate and proportionate to the new image dimensions. Once assembled, the resulting megatile image, along with its adjusted labels, provides a unified and comprehensive representation of the scene, ready for use in various computer vision applications.



**Figure 3. Megatile sample with label (include keypoints and bounding box)**

## 1.1.1.2 Physical cell :

The process of creating physical cells from aerial imagery involves multiple stages, starting with the preparation of images, dividing them into smaller sections, and generating meaningful label data for each cell. Initially, the target image is loaded using the Pillow library, which ensures that large images are handled efficiently by setting configurations that prevent truncated image loading and bypass pixel restrictions. After loading, the image is converted from the standard RGB format to BGR, preparing it for subsequent processing. Once the image is prepared, it is divided into smaller grid cells, each measuring 1024x1024 pixels or as specified by the task requirements.

To ensure that the image is divided into even and manageable sections, padding is added if necessary to match the defined grid size. Each grid cell represents a distinct portion of the image with non-overlapping regions. The pixel coordinates for each grid cell are calculated based on the grid’s height and width. The grid cells are isolated and saved individually as separate image files. This segmentation approach ensures that spatial information is maintained and critical features are not lost.

In addition to standard grid cells, physical cells are generated to focus on specific areas of interest within the grid. These physical cells are centered on the midpoint of each grid cell and extend into a square region that encapsulates features of interest. The dimensions of each physical cell are calculated based on the grid cell’s height and width using the following approach:  
The width and height of the grid cells are initially defined, such as 1024x1024 pixels. Using this information, the physical cell dimensions are calculated by determining the center of the grid cell and expanding to cover a square region that extends beyond this center. The size of each physical cell can be derived from a formula that considers the grid cell dimensions (S being the width of the grid cell):

Here, S represents the width (or height) of the grid cell (1024 pixels). This calculation guarantees that the physical cell area is sufficiently large to encapsulate relevant features, while maintaining a consistent square shape.

Each physical cell is isolated by centering on the midpoint of its corresponding grid cell. The cell is padded as necessary to ensure uniform dimensions. These physical cells are then saved individually, enabling focused and detailed analysis of specific areas within the grid.

Once the physical cells are created, label data associated with each cell is generated. Label generation begins by loading annotations from a predefined label file in JSON format. These labels typically contain shapes, lines, and keypoints marked by their coordinates within the original image. For each physical cell, a polygonal boundary is defined based on the cell’s dimensions. Any line or shape from the label file that intersects with this boundary is extracted and processed to ensure the coordinates remain accurate relative to the cell’s center. The coordinates are adjusted by shifting them based on the cell’s top-left corner, ensuring precise alignment and proportional placement within the cell. These adjusted labels are then saved as serialized files, storing only the relevant lines and shapes that intersect with each physical cell.

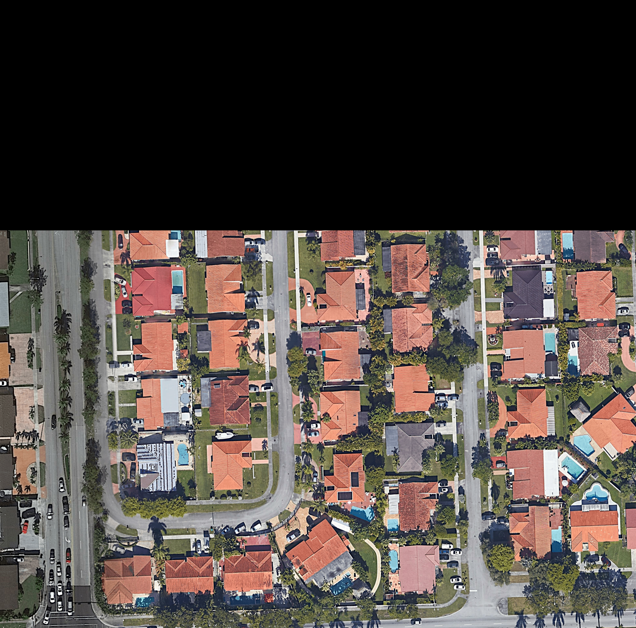
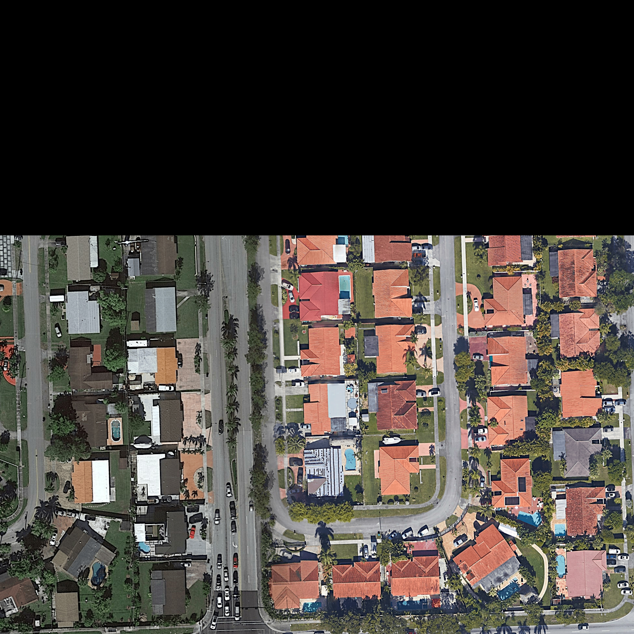
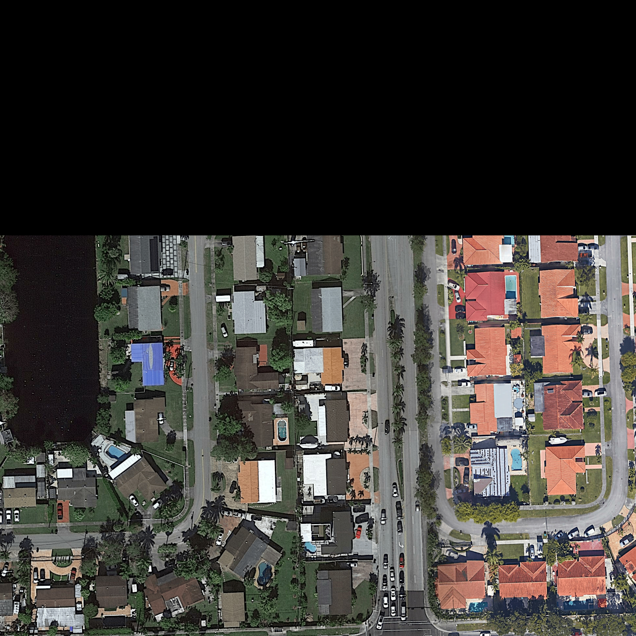


Figure 4. Sample of physical cell

The physical cells cover square areas centered at the midpoint of each grid cell, with dimensions calculated using the grid cell parameters. This approach provides sufficient padding around the center of the grid cell, enabling focused analysis while retaining spatial context from adjacent areas. The generated physical cells and their corresponding label data allow for detailed spatial feature extraction, which is critical for computer vision tasks such as object detection, pose estimation, and 3D reconstruction.

|  |
| --- |
| Pseudocode for PhysicalCell Generation |
| Start  1. Load Image and Labels:   * Load the image and label data (in JSON format)   2. Define Parameters:   * Set grid height and width (e.g., 1024) and calculate the physical size of each cell .   3. Create Physical Cells:   * for each grid cell starting from top-left corner (y, x) to the image width and height .   3.1 Define Bounding Box for the Physical Cell:   * Compute the top-left and bottom-right coordinates of the bounding box. * Ensure that the bounding box does not extend beyond the image boundaries.   1. Extract Physical Cell: * Extract the region of the image defined by the bounding box and pad the extracted region to the desired physical size.   3.3 Save the Physical Cell:   * Save the extracted physical cell as an image file (e.g., PNG format).   3.4 Check for Lines in the Physical Cell:   * For each line shape in the labels data:   if The line intersects the physical cell then  Adjust Points to New Coordinates:  End |

This structured methodology efficiently handles large images by breaking them into smaller sections, each labeled accurately based on its content and spatial location. The physical cells ensure that detailed spatial features are captured and preserved, optimizing image analysis and model training tasks. Through this process, the spatial accuracy and contextual information of individual regions within the image are maintained, enabling better feature extraction and representation for various vision-based applications.

## Sampling :

The process of sampling from physical cells within an image begins with the calculation of the dimensions for each sample. The dimensions are designed to cover a defined square area, ensuring consistency in the output samples. Using the width and height of the physical cell, the center point of the cell is calculated, which serves as the origin for the sample generation. A fixed size, such as 1024x1024 pixels, is assigned to each sample, ensuring uniformity. The height and width of the physical cell, denoted as H and W, are used to determine the sample dimensions through the following calculations. Here, H represents the height of the physical cell, and W represents the width. The fixed size ensures that the extracted samples have a consistent area for processing.

Once the dimensions are established, parameters for the sample generation are set. These parameters introduce variability to the samples, allowing them to cover different regions within the physical cell. A random offset is applied to the cell's center to define the center of each sample, with the offset ranging within a certain limit, typically up to 512 pixels from the center. The center of each sample X\_s, Y\_s(can be defined as:

Where (Xc,Ycis the center of the physical cell, and ΔX, ΔY represent the random offsets. These offsets ensure that samples are positioned differently within the cell while remaining close to the main cell area.

In addition to positional shifts, samples are subjected to various transformations, such as rotation, scaling, and flipping. The rotation of each sample is introduced through a random angle, typically ranging from 0 to 360 degrees. The rotation is represented mathematically using a 2D rotation matrix:

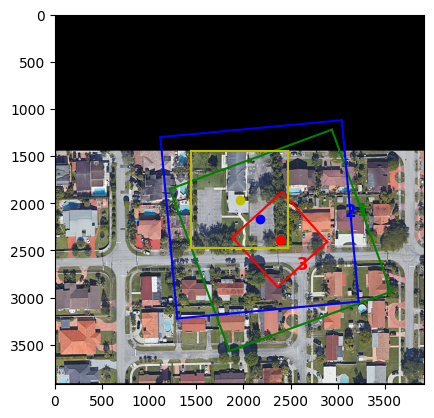
Here, θ is the random rotation angle applied to each sample. This rotation helps introduce diversity in the orientation of the samples, which is crucial for training models that require rotational invariance.

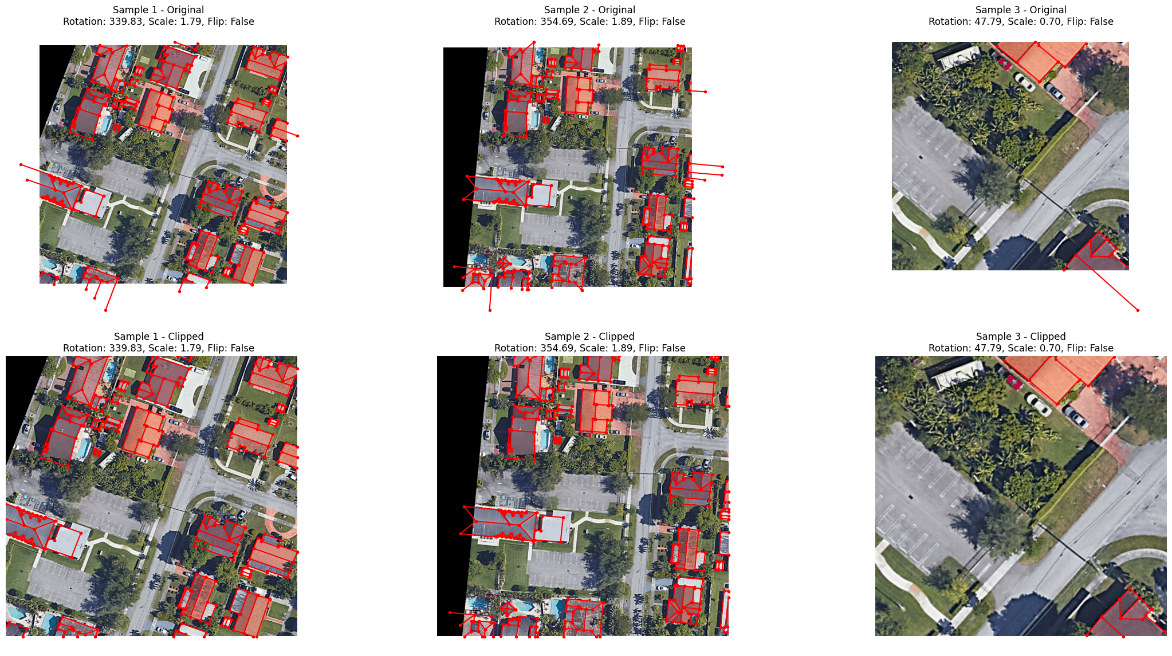
Scaling is another critical transformation, applied by a random scale factor s, where s can vary between 0.5 and 2.

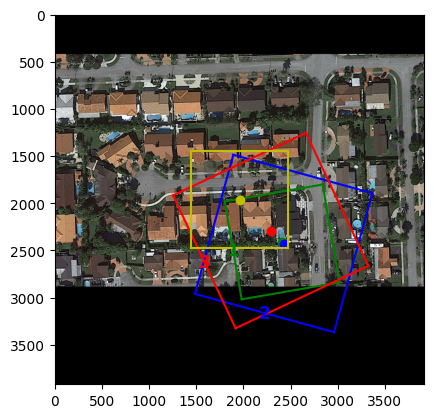
This variety allows the samples to represent different features of the physical cell effectively, making them suitable for training models that need to be robust against varying scales and orientations. Furthermore, flipping the sample horizontally is occasionally performed, which increases the dataset’s variety by mirroring content and enhancing tasks that require symmetry or invariance to orientation.

Once the samples are generated, their bounding boxes are transformed to align with the calculated center, rotation, and scale. The bounding box defines the spatial boundaries of each sample and undergoes scaling and rotation to match the transformed sample area. This process ensures that each sample’s area accurately reflects the intended size and position within the physical cell. Following this, keypoints within the sample area are identified and clipped to ensure they remain within the visible boundaries of the sample. Using a polygon formed around the bounding box, lines are checked to see if they intersect with the polygon. For each line that intersects, the corresponding keypoints are extracted, and their coordinates are transformed to match the perspective of the sample. Keypoints that fall outside the sample boundaries are clipped, preventing them from extending beyond the visible area and maintaining the accuracy of the sampled data.

This sampling process is crucial for covering the entire megatile area effectively. By generating samples with diverse transformations—ranging from rotation and scaling to flipping—the dataset becomes more comprehensive, enriching the data available for machine learning tasks. Clipping keypoints within each sample is vital, as it ensures that the extracted features remain within the visible region, preventing any data from being misaligned or lost. This method not only optimizes the coverage of the megatile but also enhances the quality of the dataset for tasks such as object detection, feature extraction, and model validation. Through careful calculation of sample dimensions and the use of various transformation parameters, this approach ensures that every part of the physical cell is adequately represented, making it an essential step in preparing the data for advanced image analysis and machine learning applications.







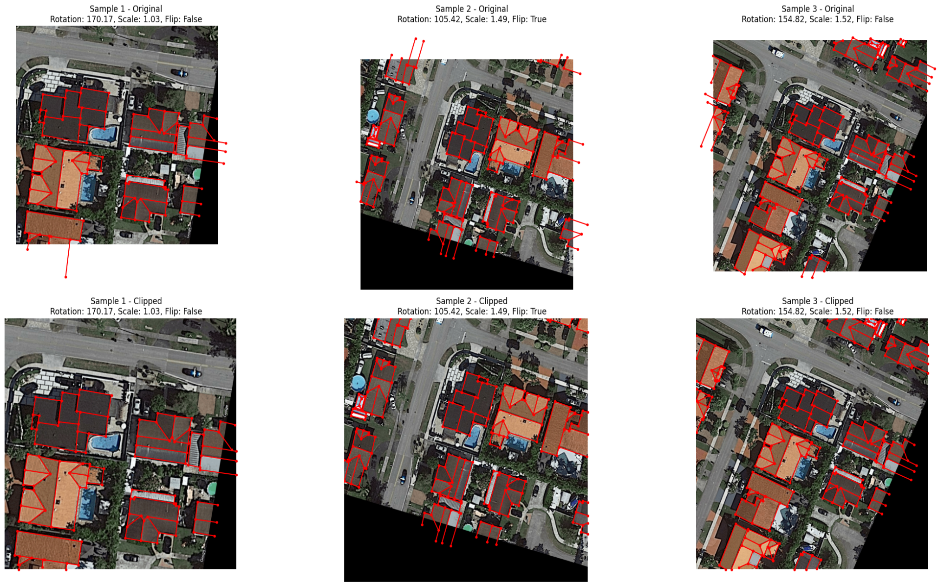


Figure 5. sampling of Physical cell

|  |
| --- |
| Pseudocode for Sample Generation |
| Start  1. Load the image and data:   * Load the cell image and corresponding pickle data containing keypoints information.   2. Compute the center and size of the cell:   * Retrieve the image dimensions (height and width) and calculate the center of the image as the reference point.   3. Generate random samples and apply transformations:   * For each sample: Select a random center point around the cell’s center. Then assign a random rotation angle (between 0 and 365 degrees).Assign a random scale factor (between 0.5 and 2).Randomly decide whether to flip the image vertically (flipy).Generate a bounding box around the random center with fixed size (1024x1024) considering the scale, rotation, and flip.   4. Compute the bounding box for the cell:   * Define a bounding box centered around the cell’s center with the same size (1024x1024).   5. Apply perspective transformation (cutting the image):   * For each rotated bounding box, apply a perspective transform to crop the image according to the bounding box, applying rotation, scaling,and optional vertical flip.   6. Find keypoints that intersect with the bounding box:   * For each bounding box: Check which keypoints from the data intersect with the boundingbox (with a small tolerance).Store these keypoints for further processing.   7. Transform the keypoints:   * Apply the inverse of the transformations (rotation, scale, flip) to the keypoints. * Adjust the keypoints to the transformed image dimensions.   8. Clip the keypoints to image boundaries:   * Ensure that the transformed keypoints are within the image boundsby clipping any that fall outside.   End |

## Augmentation

As the final stage of data preparation, data augmentation is applied to the extracted samples to enhance the diversity and quality of the dataset. This critical step ensures that the model becomes robust and capable of handling a wide range of real-world variations, reducing the risk of overfitting and improving generalization. By introducing controlled distortions and variations to the data, augmentation creates a richer training dataset without the need for additional manual data collection.

The augmentation process involves several carefully selected transformations. First, blur effects are applied to simulate out-of-focus conditions. These include both simple blurring and median blurring, which smooth the image and mimic common scenarios like motion blur or poor focus.

Next, the brightness and color adjustments ensure robustness to different lighting and environmental conditions. Random brightness and contrast variations, along with hue and saturation shifts, help the model adapt to diverse scenarios. Additionally, grayscale conversion is applied to some samples to simulate monochromatic inputs and ensure flexibility in such cases.

To prepare the model for noisy and low-quality images, Gaussian noise is added, mimicking imperfections such as sensor noise or compression artifacts. Similarly, image compression is used to degrade image quality, ensuring the model can handle variations caused by lossy data storage or transmission.

Advanced techniques like Gaussian blur introduce additional smoothing, while coarse dropout randomly masks parts of the image, simulating occlusions or missing regions. These augmentations force the model to rely on contextual information and improve its ability to generalize.

By applying these transformations systematically, data augmentation serves as a powerful tool to prepare samples for training. It not only increases the dataset’s variety but also ensures that the model performs well under diverse and unpredictable conditions, making it an indispensable step in modern machine learning pipelines.

# Detection model ( light weight )

Object detection is a critical area in computer vision that involves identifying and locating objects within an image. This task is distinct from image classification, as it not only classifies objects but also predicts their positions in the form of bounding boxes. Object detection models are typically categorized into two main types: single-stage and two-stage models. Two-stage models, like Faster R-CNN, first generate region proposals and then classify and refine these proposals, often achieving high accuracy at the expense of speed. Single-stage models, such as YOLO (You Only Look Once) and SSD (Single Shot Multibox Detector), handle detection and classification in a single step, prioritizing speed and efficiency. YOLO, in particular, has gained prominence for its ability to process images in real time, making it suitable for applications that require immediate feedback, such as autonomous driving and video surveillance.

The base model is made up of two key components: the feature extractor and the detector. The feature extractor, which is based on a convolutional neural network (CNN), processes the input image and creates feature maps at three different resolutions: 32×32, 16×16, and 8×8. These feature maps are passed to the detector, which creates output grids containing object scores, bounding boxes, and class confidences for each object. Each resolution divides the image into grid cells, with each grid cell containing multiple anchor boxes to capture objects of different dimensions. The architecture of the F11 model, including its feature maps at three resolutions (32×32, 16×16, and 8×8) and the multi-scale detection framework, is illustrated in Fig. 5**.**

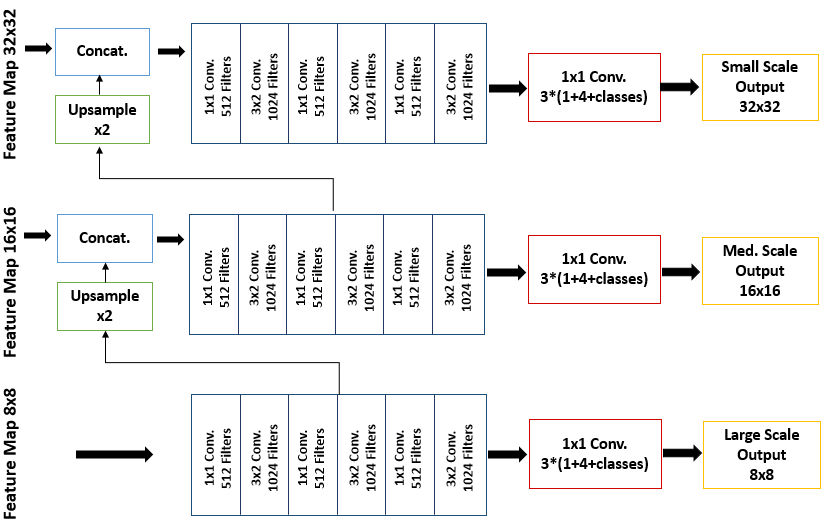


Figure 6. The F11 detector network including multiscale predictions.

The final output of the F11 detection model is displayed at three different resolutions: an 8×8 grid with 16,384 cells, a 16×16 grid with 4,096 cells, and a 32×32 grid with 1,024 cells, making a total of 21,504 grid cells. Each cell holds crucial details, such as the coordinates of the bounding box (x, y, width, height) and the object scores for each anchor box. Specifically, the dimensions for the prediction bounding boxes and target bounding boxes are (4, 21504, 4), where the first dimension (4) corresponds to the batch size, and 21,504 corresponds to the grid cells across the different resolutions. The final four values in each entry correspond to the bounding box dimensions (x, y, width, height). For prediction score and target score, the output dimensions are (4, 21504, 4), with the last 4 values corresponding to the 4 class of our problem. After generating the outputs, the F11 model’s predictions are processed to align with ground truth annotations, ensuring compatibility with the Attention U-Net and AST model.

Before the image restoration can process the data, outputs from the F11 model undergo a series of preprocessing steps to ensure compatibility and improve segmentation accuracy. In the first place, the bounding box information and the class scores produced by F11 detection part are concatenated to matrix form is a divided single into matrix three of parts size in (4, line 21504, with 8). This grid sizes applied in F11, namely 8 x 8, 16 x 16 and 32 x 32. All the grids are then resized to certain sizes based on the number of grid cells; the 8x8 grid becomes a 4x128x128x8 tensor, the 16x16 grid becomes a 4x64x64x8 tensor and the 32x32 grid becomes a 4x32x32x8 tensor. To standardize the input size, all grids are upscaled to a resolution of 128×128 pixels, with the 16×16 grid expanded to (4, 128, 128, 8) and the 32×32 grid similarly resized. These three grids are then merged into a unified matrix with a final shape of (4, 128, 128, 24), which is subsequently fed into the restoration network for further refinement.

In this project, the F11 object detection model was chosen to detect and classify lines within images due to its efficiency and speed. Unlike traditional approaches that scan images region by region, F11 object detection model treats the entire image as a single regression problem. It predicts bounding boxes and class probabilities simultaneously, ensuring faster and more accurate detections. This property makes F11 object detection model well-suited for the specific requirements of this project, where real-time performance and precision are essential. Lines were treated as individual objects, and their classification was based on their orientation, calculated using the slope between two endpoints. This setup allowed the model to handle both detection and classification tasks effectively.

To simplify the classification process, the lines were grouped into either two or four distinct categories based on their slopes. In the two-class approach, horizontal lines with zero or near-zero slopes were categorized as one class, while vertical or positively sloped lines formed the second class. For more detailed classification, the lines were divided into four categories: horizontal, vertical,positive slope, and negative slope. This granularity allowed the model to detect.

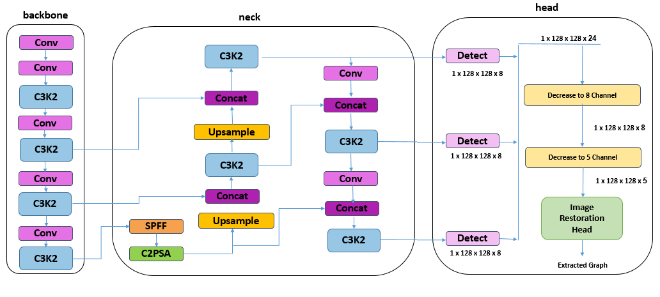
To support this line detection and classification, a custom dataset class was developed. This dataset class was designed to handle the unique requirements of object detection for lines, including processing annotations for bounding boxes and keypoints. Each line was represented by a pair of keypoints, with bounding boxes calculated based on these points. Various augmentations, such as rotation, scaling, and flipping, were applied to create diverse training samples, improving the model’s ability to generalize. Additionally, the dataset class ensured that all keypoints and bounding boxes were correctly aligned with image boundaries, leveraging a clipping mechanism to handle out-of-bounds cases.

Figure 7. F11 Detection Model diagram ( include back bone, neck and head )

The dataset preparation involved several key steps. First, each image and its corresponding annotations were paired to

create structured label data. The bounding boxes and keypoints were then sorted and transformed to ensure consistency across samples. For added flexibility, a grid-based orientation system was introduced, enabling the model to simulate various rotations and alignments. This feature was particularly useful in handling datasets with predefined orientations or random angles, enhancing the model’s adaptability to different scenarios. Each sample generated by the dataset class included a transformed image, normalized bounding boxes, line class labels, and metadata such as the original image size and transformation details.

The training pipeline was implemented using a custom trainer framework tailored to the specific requirements of line detection. This framework extended the F11 object detection model architecture by integrating custom validation and visualization components. During training, a custom visualization callback was used to generate annotated images that showcased the model's predictions alongside the ground truth. These visualizations included bounding boxes, keypoints, and line classifications, offering an intuitive view of the model's progress. The annotated images were organized into directories by epoch and batch, providing an efficient way to review training samples and evaluate model performance over time.

Validation played a crucial role in ensuring the model's robustness and accuracy. A custom validator was implemented to evaluate the model's predictions against ground truth annotations. This validator extended the F11 object detection model validation framework with advanced visualization capabilities, including class-specific line annotations and confidence-based color mapping. Each line was annotated with its orientation and classification, and the confidence score was represented using a gradient color scheme. These visualizations provided detailed insights into the model's performance, highlighting areas where improvements were needed.

To further enhance interpretability, the validation framework included a plotting function that generated grids of annotated images. These grids displayed the predictions and ground truth annotations side by side, making it easier to assess the model’s accuracy visually. The images were aligned on a standardized canvas, ensuring consistency in appearance and facilitating comparison across validation batches. The annotated outputs were saved in a structured directory, enabling seamless navigation and review of validation results.

# 1.2.1 Model Evaluation

In this training experiment, the F11 object detection model was configured with a robust set of hyperparameters to ensure optimal performance for the line detection and classification task. The training was conducted on a high-performance NVIDIA A100 GPU, leveraging its computational capabilities to handle large datasets and high-resolution images efficiently.

The model was trained for a total of 2500 epochs, with each epoch consisting of multiple iterations of forward and backward passes to optimize the weights. The batch size was set to 8, balancing memory usage and training stability, while the input image size was configured to 1024x1024 pixels to retain sufficient detail for accurate line detection. The choice of a relatively large image size ensured that even subtle line features were adequately captured, enabling precise orientation classification.

The Stochastic Gradient Descent (SGD) optimizer was employed to update the model’s weights. SGD is particularly well-suited for large datasets due to its efficiency in handling high-dimensional data and its inherent ability to generalize well, reducing the risk of overfitting. The learning rate schedule was designed to facilitate a smooth and stable training process. The initial learning rate (lr0) was set to 0.002, allowing for gradual weight updates during the early stages of training to ensure stability. This was followed by a decay mechanism, gradually reducing the learning rate to a final value (lrf) of 0.01 as the model approached convergence. This strategy prevented overshooting the optimal solution, fostering fine-tuning of the model parameters in later epochs.

To accommodate potential long periods of plateauing performance, the patience parameter was set to 2000 epochs. This meant that the training process would not terminate prematurely, allowing the model sufficient time to explore the parameter space and identify improvements even after extended periods of stable performance. This high patience value reflects the complexity of the task and the need for iterative refinements in detecting and classifying line orientations.

One noteworthy aspect of the training configuration was the choice of the Intersection over Union (IoU) threshold, which was set to 0.0. By not penalizing

localization errors, the training process focused primarily on optimizing classification performance. This decision aligned with the project's goals, as accurate orientation classification was prioritized over precise bounding box localization. This approach is particularly advantageous in scenarios where localization precision is secondary to correct categorization of objects.

Table 1. Train Config parameters

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Sever | Problem | Image size | Batch size | 100-epoch-time | Epochs | Workers | Iou | optimizer |
| A100 | 4- class | 1024 | 8 | 15h | 2350/2500 | 8 | 0.9 | SGD |

The training process on the NVIDIA A100 GPU leveraged its advanced hardware features, including Tensor Cores, to accelerate matrix operations and support mixed precision training. These capabilities significantly reduced training time while maintaining high computational accuracy. The A100’s large memory capacity also facilitated efficient handling of the high-resolution images and large batch sizes used in this experiment. On average, the time required to complete 100 epochs of training was approximately 15 hours, emphasizing the computational demand of the task while highlighting the A100’s efficiency in handling such workloads.

In summary, the training experiment employed a carefully designed configuration

tailored to the specific requirements of line detection and classification. The choice of hyperparameters, optimizer, learning rate strategy, and patience settings, combined with the computational power of the NVIDIA A100 GPU, contributed to a robust and efficient training process. This setup ensured that the model achieved high classification performance, addressing the challenges inherent in this specialized task.

|  |  |
| --- | --- |
| GT | Result |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |

Table 2. Result of F10 model object detection

|  |  |
| --- | --- |
|  |  |
|  |  |
| **https://lh7-rt.googleusercontent.com/docsz/AD_4nXfZPIGO2E3nStM4wrrLqcTElsyXrI0jWCzIlMi5JWYUA2s7Us05kNKMvk-zSU0tAA12MXNcejUhJb4S1fJyOrMeBYXWNLXk9nY-5dEdfKIDC5cTO_WPK9V5J2yyh9T9Hk5kTsOCT3GL6RvUHsvDqaWEUBk?key=Yv3FGweKfgyi2B9MO644pWYT** | **https://lh7-rt.googleusercontent.com/docsz/AD_4nXd70bWqbV_UjD4c-6idiDybYobZ2uoc77RqYy_cRm6Uqo61xE_Rd9_-lIHiFpdXF2DNl_7eOcSm4Ld1TvEDkoiEhmgYPKoIZTuTa1wthkkmspP9v2JMZ0xT2m5Fjyfb3ghZtqbsB1sto4rKhEnyG_aryec?key=Yv3FGweKfgyi2B9MO644pWYT** |
| **https://lh7-rt.googleusercontent.com/docsz/AD_4nXebmV09jIhomuRRn3h9y1z2LiymrvlOvLWDGU2wsk8iRe8uWjj6sM8ahwwENgxWMFbuoCvfZgpDuL53JuE4TbgrujrT3wcvKRAveL7mF9ZyS3PlqOYkS2AHaNxl7ev_OnJOkom6HBSHWPwfo1sDnerFhPZ0?key=Yv3FGweKfgyi2B9MO644pWYT** | **https://lh7-rt.googleusercontent.com/docsz/AD_4nXek5N_DTfjDHPRIYzWBKKnHZk22gokM7OVuU3Ba-ujZwCBdpUG5guEBheUmzhaXUDqy64qOJuH5EOWBGZpH58ikTcCcUZ5a0vufZiWyMTkvSE9pUjS9N8jNqkuO_GxnX7KD4kCbHRbV_UkjkpktkKr5gsyN?key=Yv3FGweKfgyi2B9MO644pWYT** |

# 1.3 Alignment

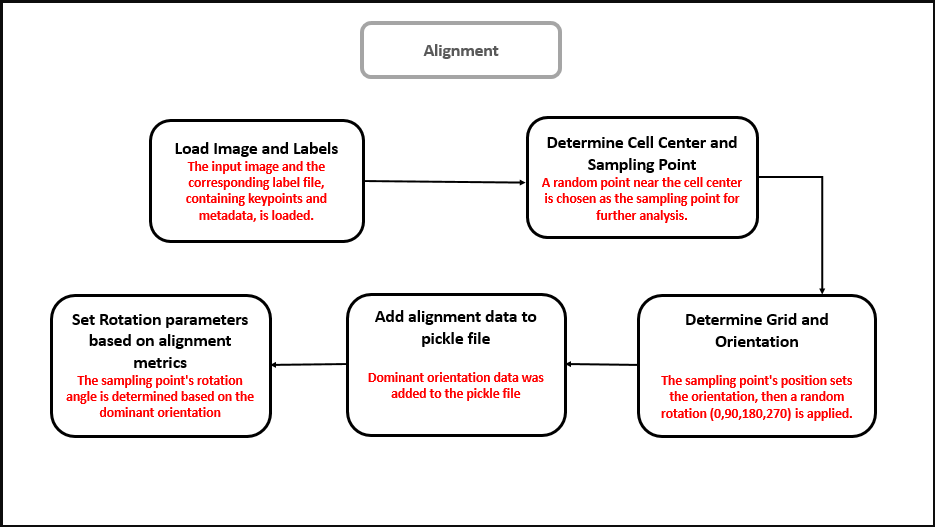
In machine learning and computer vision tasks, ensuring that input images are properly aligned is a critical step in data preparation. Alignment helps standardize images, making them more suitable for model training and reducing variability that could negatively impact performance. Without a structured alignment process, models may struggle with inconsistencies in orientation, leading to poor generalization.

Figure 8. Alignement diagram (including stepts to obtain final rotation )

A key aspect of alignment is determininga sampling point within the image. Instead f relying on arbitrary positions, a point near the cell center is selected. This ensures that analysis focuses on a representative area rather than being affected by irrelevant background noise or edge artifacts. Selecting a sampling point close to the center enhances stability and consistency in image processing.

Once the sampling point is established, the next step is to define a grid and orientation. The orientation of the sampling point dictates how the image is aligned, and to introduce controlled variations, a random rotation (0, 90, 180, or 270 degrees) is applied. This method prevents the model from being biased toward a specific directional pattern, making it more adaptable to real-world scenarios where images may not always be captured in the same orientation.

To ensure reproducibility, the alignment data is stored in a pickle file. This file contains key metadata, such as the dominant orientation, allowing for systematic processing in later stages. Storing alignment information ensures that any transformations applied to the images can be traced and re-applied if necessary, maintaining consistency throughout the pipeline.

The final aspect of alignment is setting rotation parameters based on dominant orientation metrics. By analyzing the primary direction of the image content, an appropriate rotation angle is assigned to the sampling point. This structured approach to rotation eliminates arbitrary transformations and ensures that images are aligned in a way that enhances model learning.

Alignment plays a crucial role in preparing data for machine learning models. It ensures that images are consistently oriented, minimizes unnecessary variations, and enhances model robustness. By systematically selecting a sampling point, setting orientation, applying controlled rotations, and storing alignment metadata, this process helps create a well-structured dataset that improves overall model performance.

# 1.4 Image Restoration :

Image restoration is a fundamental task in computer vision aimed at reconstructing a high-quality image from a degraded version, counteracting distortions such as noise, blur, or missing details. This field plays a critical role in applications like medical imaging, satellite analysis, and computational photography, where image quality directly impacts downstream tasks and decisions. Image restoration often requires advanced models to accurately recover missing or corrupted information, especially when dealing with real-world degradation scenarios.

In this experiment, the primary goal is to process the raw output of the F11 object detection model and apply subsequent refinements to enhance image quality. The F11 object detection model output serves as the initial input, which undergoes a series of pre-processing steps to align it with the restoration requirements. Following this, advanced architectures such as UNet and Adaptive Sparse Transformer (AST) are employed as the final image restoration networks. These models ensure that both global structure and fine details of the image are accurately reconstructed, producing high-quality outputs suitable for further use.

## Unet :

UNet is a convolutional neural network architecture originally designed for biomedical image segmentation but widely adopted for image restoration tasks. Its symmetric encoder-decoder structure enables it to capture multi-scale features while preserving spatial details, making it particularly suitable for tasks like denoising, deblurring, and inpainting.

The encoder extracts hierarchical features from the input through convolutional and pooling operations, progressively reducing the spatial resolution while enriching the semantic representation. The decoder, in turn, reconstructs the high-resolution image by upsampling these features and integrating them with corresponding high-resolution features from the encoder via skip connections. These skip connections bridge the semantic gap between the encoder and decoder, allowing the network to retain crucial spatial information lost during downsampling.

In this pipeline, UNet processes the pre-processed F11 object detection model output, refining the coarse features and ensuring the reconstructed image aligns with the desired quality standards. Its ability to balance global context with local details makes it highly effective in scenarios requiring fine-grained restoration. Furthermore, its relatively lightweight architecture ensures efficient training and inference, making it an excellent choice for iterative refinement of F11-derived data.

## AST

The Adaptive Sparse Transformer (AST) represents a state-of-the-art approach to image restoration, leveraging the strengths of transformer-based architectures to handle complex and high-resolution data. AST is specifically designed to overcome the limitations of convolutional models by capturing long-range dependencies and focusing computational resources on the most relevant regions through its sparse attention mechanism.

The adaptive sparse attention module in AST selectively identifies and processes the most important areas of the image, reducing computational overhead while retaining essential features. This mechanism ensures that the model remains efficient even when handling high-resolution F11 object detection model outputs. Moreover, AST incorporates an attentive feature refinement module, which iteratively enhances multi-scale features by integrating global context with local details. This enables AST to excel in reconstructing intricate textures and resolving severe distortions.

In this framework, AST takes the pre-processed F11 object detection model output as input and performs sophisticated restorations to produce high-quality images. Its transformer backbone allows it to model dependencies across large spatial regions, making it particularly well-suited for restoration tasks involving significant noise, occlusions, or complex degradation patterns. While computationally more intensive than UNet, AST's ability to deliver superior restoration quality justifies its application in tasks where precision is paramount.

## Training Setup and Performance Evaluation :

The training framework was meticulously designed to achieve high-performance results in matrix reconstruction tasks. The configuration utilized a batch size of 1 with an input patch size of 159, ensuring precise processing of individual data samples. The training process spanned a total of 1000 epochs, reflecting the model's emphasis on thorough learning over extended periods. An initial learning rate of 0.0002 was employed, which facilitated stable gradient updates during the early stages of training, avoiding abrupt changes in the optimization process.

To optimize the model’s parameters, the AdamW optimizer was utilized. AdamW, known for its adaptive learning rate capabilities and decoupled weight decay, is particularly well-suited for large-scale deep learning tasks. This choice of optimizer effectively balanced convergence speed and generalization performance, critical for complex reconstruction problems.

The computational demands of training were efficiently managed through GPU acceleration (GPU 0). Leveraging high-performance hardware significantly reduced the runtime per epoch and ensured that the model could process large datasets with intricate details. The learning rate scheduler further enhanced the training dynamics by employing a dual-phase approach: an initial warmup phase allowed the model to stabilize during the early epochs, while a CosineAnnealing strategy gradually reduced the learning rate as training progressed, facilitating convergence to an optimal solution.

To expedite the training process, pretrained weights were loaded at the beginning of the retraining phase. This initialization leveraged prior knowledge, enabling the model to converge more efficiently while reducing the number of iterations required to achieve high performance.

The training process was computationally intensive, with each epoch requiring approximately 1.5 hours to complete. This substantial time investment underscores the complexity of the task and the computational rigor required for effective model optimization. Despite the lengthy training time, the framework successfully utilized these resources to achieve significant performance gains.

During epoch 1000, key performance metrics were evaluated to assess the model’s progress. The Peak Signal-to-Noise Ratio (PSNR), a critical metric for measuring reconstruction quality, was calculated as 32.86 dB between the input and ground truth matrix. This value reflects a substantial improvement in matrix reconstruction accuracy and highlights the model's capability to recover fine-grained details from the input data.

The results of this training setup demonstrate the effectiveness of the chosen strategies, including the use of AdamW, learning rate scheduling, and pretrained weights, in addressing computationally demanding reconstruction tasks. The model's ability to achieve high PSNR values at an intermediate epoch underscores its potential for further improvement with continued training and optimization. This study highlights the critical interplay between advanced training techniques and computational resources in achieving state-of-the-art performance in matrix reconstruction tasks.

|  |  |  |
| --- | --- | --- |
| input | target | result |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |

Table 3. Result of AST model

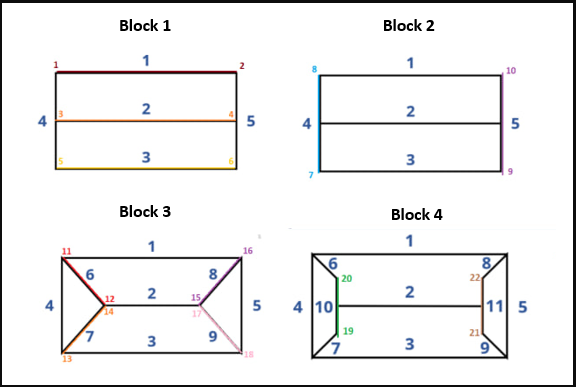
|  |  |  |
| --- | --- | --- |
|  |  |  |
|  |  |  |

## Block detector :

Keypoint pose estimation plays an essential role in block detection and recognition in, especially in aerialytic dataset. In this section, we propose a dataset and methodology focused on keypoint-based object detection, targeting a scenario where four distinct blocks each contain 22 keypoints. These keypoints represent critical landmarks that enable accurate alignment and object identification, providing essential structural information for pose estimation and object localization.

# 2.1 Data prepration

The dataset consists of images paired with labeled annotations. Each annotation file includes bounding boxes (BBoxes), which encapsulate the objects detected in the images, class labels that define the type of each object, and keypoints that specify crucial spatial landmarks. Each block in the dataset is annotated with 22 keypoints, each associated with a visibility score, allowing for precise representation of the object’s pose. The visibility information of the keypoints—where a score of 2 indicates fully visible points—ensures reliable estimation of the object’s spatial configuration.

Figure 9. four type of block diagram with line and keypoint visualization

The data processing pipeline begins with loading the images along with their respective keypoint annotations. Following this, keypoints are verified and adjusted to ensure they adhere to a consistent order across all images. This process guarantees that the keypoints are reliably positioned to represent the object’s structure. Bounding boxes are then dynamically generated based on the visible keypoints. These bounding boxes are determined by calculating the min-max range of the visible keypoints, providing the necessary parameters such as center coordinates, width, and height. This dynamic approach allows the bounding boxes to adapt to variations in object pose and scale.

To further enhance the dataset's robustness and generalization capability, various augmentation techniques are applied. These transformations include random rotation, where objects are rotated within a 0 to 360-degree range, scaling variations, which modify the object’s size between 30% and 50% of its original dimensions, and flipping operations that randomly mirror images along the vertical axis. These augmentations ensure that the model trained on this dataset can handle a diverse set of real-world scenarios.

A dedicated mechanism is incorporated to ensure the consistency of the keypoint order throughout the dataset. The system enforces specific spatial relationships, such as ensuring that certain keypoints remain to the right or left of their counterparts, and that keypoints related to the top and bottom of the object maintain the correct hierarchy. This correction process guarantees the spatial integrity of the keypoints, allowing for more accurate pose estimation.

In conclusion, this dataset and preprocessing pipeline offer a structured and systematic approach to keypoint pose estimation in a multi-block house. By enforcing consistent keypoint ordering, dynamically calculating bounding boxes, and employing robust data augmentation techniques, the methodology enhances object detection performance. The four-block structure with 22 keypoints per block provides a flexible and scalable foundation for training deep learning models for keypoint-based pose estimation and object detection tasks, ensuring the model's effectiveness in various real-world applications.

# 2.2 model evaluation

The Block Detector model, utilizing a Tiny version of F11- pose estimation, was trained to perform pose estimation on a dataset of block images. The configuration parameters for the training process are summarized in Table 1, which outlines the server, image size, batch size, epoch time, and other key details.

The training was conducted on an A100 server with a batch size of 8 and an image size of 1024x1024. Each epoch took approximately 18 minutes, and the model was trained for 1000 epochs out of a total of 1000 epochs. The optimizer used was SGD, and the Intersection over Union (IoU) threshold was set to 0.9. The model's objective was to detect keypoints and estimate poses of objects in the dataset, which required effective training and fine-tuning of its parameters.

Regarding the model's performance over the epochs, the pose\_loss was closely monitored as a metric for keypoint prediction accuracy. At epoch 1, the pose\_loss was found to be approximately 11. This high value indicated that the model was still in the early stages of learning and had room for significant improvement in its pose estimation capabilities.

However, by epoch 1000, the model demonstrated notable progress, with the pose\_loss decreasing to 0.6. This improvement reflects the model’s better understanding of the keypoint positions and its ability to make more accurate pose estimations. The reduction in loss is a positive indicator that the model is effectively optimizing its weights and learning to predict keypoint locations more precisely over time.

Table 4. Train Config parameters for block detector

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **Sever** | **Problem** | **Image size** | **Batch size** | **1-epoch-time** | **Epochs** | **model** | **Iou** | **optimizer** |
| **A100** | **Block detector** | **1024** | **8** | **18 min** | **1000/1000** | **tiny** | **0.9** | **SGD** |

Table 5. Block detector result

|  |  |
| --- | --- |
| GT | Result |
| **https://lh7-rt.googleusercontent.com/docsz/AD_4nXfzKmzGIx1YdMYqgajPbETkBPJ3MFTPKnK06KTcyEJa3HGeHmu8fzdBP_P73D4TH4wHY0zv0xVNtDDJruefNfmwVxmg-MJwbaQKg2wP3Ut30Q8NZ23pmC8IklDWn3uf-0TjenyMJg?key=T1D0B9_ILh_nNOdcc5Umtwwl** | **https://lh7-rt.googleusercontent.com/docsz/AD_4nXe0CloZ67_WNLkYxnEDM6DN0uT9YIsLystZdgv3hGFBxUJYmFlaWhTs2R-wAOO5RAz_4x6U2eLgwkA2N7p4htKdIdBN0_aqcG1Ys8yNPEi2bS2LAmn-DHXUgZxOi9v6p3iDVmJazw?key=T1D0B9_ILh_nNOdcc5Umtwwl** |
| **https://lh7-rt.googleusercontent.com/docsz/AD_4nXcPJ8nOTeCk-UOM3Z_Isp9wvm1TSZ4peIhAz8aAxCWagqWO7FQ05_qvHfgnhI5XLtUzL1-vyKMBiS8ICZuhPAbXnfMrrIvPAz7QVvdCMwdGGj7OlhJVnw_NFsmZLvonUjcWYEXTJg?key=T1D0B9_ILh_nNOdcc5Umtwwl** | **https://lh7-rt.googleusercontent.com/docsz/AD_4nXchSTD_3TPpyq2N-HpO0MggA9Z3BSvrYfEnxD0tnOdXxG1fG22aTSFjT8w2eMzRsGdVPagbKVwZIgpM23E4PaAiWTH79tqjVDmvaQU4pxx4SbMK5XjZf-Sqz7_rdzDfu914TYG1?key=T1D0B9_ILh_nNOdcc5Umtwwl** |
| **https://lh7-rt.googleusercontent.com/docsz/AD_4nXe1hBJF50Wc3AdrkwT2pquMeN2Ybi98eYgzudn1YmI5r5-ykDIuemXRuxtb0WDqIj6X0wqqcwtFrQ5v7w3sjKIZf66CnpM-JG9eyy-u_1_b6_0BZ_KS6wwqHcEeoOngjROtl_kzXg?key=T1D0B9_ILh_nNOdcc5Umtwwl** | **https://lh7-rt.googleusercontent.com/docsz/AD_4nXeVgJF-5Wiu9-G0vq-czOxE0ILKmqL6FK2P7F5VN4Xnk2oiJ0zHeO-KwJoeKi2WyPcwxO4XwjK12tZX1uMHnmC5qMEh2ExKByJEHIci0P1iqldrJJ1J_b-7YkNdjLcIrvofO7nM?key=T1D0B9_ILh_nNOdcc5Umtwwl** |

|  |  |
| --- | --- |
| **https://lh7-rt.googleusercontent.com/docsz/AD_4nXe7v0hl1MbNFNTRwkSqlpIQUiUlRzrARGKtFJ4kQu-5UQZzZnG4GguibGFTcz9g0COQWLcttmY8toy9ObzJ7ZuT82nsvZaBZ-G_mZOgwySNRtJv_yFlMzsc7zeAbCq47QjCQbyV?key=T1D0B9_ILh_nNOdcc5Umtwwl** | **https://lh7-rt.googleusercontent.com/docsz/AD_4nXcVygakAox51INPsNr5TCcwX7WVz50c4DytqiTunsur39uNtbtYR4qvNI5MQxt1NmdFQ4zqbsTVLHXMtM8a_Kn9zQiPZ8Tfmy14rRrrJ1MeiiCWZPuW5L3417LSLZpBMMRgNIsb_A?key=T1D0B9_ILh_nNOdcc5Umtwwl** |
| **https://lh7-rt.googleusercontent.com/docsz/AD_4nXeZviB10dIkhYo7FFov4dhxqUOGia-0b2YJFkJ42rzdZhfLDtpSKODCWqnWZteqE2J7LLUeaZtrkZx_fZUHKAybVPYGh7MXkaQBAm1a5k_umt-Sdd0L_HQPZWUtGehww-UMq-gjHQ?key=T1D0B9_ILh_nNOdcc5Umtwwl** | **https://lh7-rt.googleusercontent.com/docsz/AD_4nXdT4_ax1G48_ITn6_bZouFX-qfY-Vd4JVh2Vg1GWukgynhIDSkaJgVzGzoHj1f-1lcy1DxUBQ0HInyKsb2uk6TviX6raJiEDjjeiL12IwxBhSg9K9jd9aPq8kG6F3M22YhR56jOaA?key=T1D0B9_ILh_nNOdcc5Umtwwl** |
| **https://lh7-rt.googleusercontent.com/docsz/AD_4nXcdDlLPEloWQNvrukmNHtHShsx1lblvDnaKq0z4nphZEOwOJ4mu7as1MLe_R5IN4_MIAUnFW3GmO-STz8nm2ukf_cMUn2tHw9_TdVrYe-UuJuFrr-3RX8lfEilRtmcBh16YN30YjA?key=T1D0B9_ILh_nNOdcc5Umtwwl** | **https://lh7-rt.googleusercontent.com/docsz/AD_4nXeaDFeZGEdKCh2ciHjuSjwpgoG9WCVsb1s4h2ZU0PTr0htK4UdjBV4deeRZKUHCcxVWQuPmfvyEUsqruhIQOavn3rWSOHhqSgZalyCDnQnMtPGOqFM1LVX2vMBAy4bcNh-DTCbw?key=T1D0B9_ILh_nNOdcc5Umtwwl** |
| **https://lh7-rt.googleusercontent.com/docsz/AD_4nXdOUMgEqp97TQePI5ebYbAZNDgGU-1c_1ivFBAwjJaIoVMi3pgVUd1GLGR8mA10DoPxrqJnA6KGZvOgOACE2az9WhAzYY7Q9P5w48rbBpPPXSwjGCRJmSlJclOu9tXAoaxz_tJsXw?key=T1D0B9_ILh_nNOdcc5Umtwwl** | **https://lh7-rt.googleusercontent.com/docsz/AD_4nXcuqRkRyef8-4w2wHqQLaJ5-aLcZJcr4r5q6FE-ENRfMY16yF_q82bf-XuyivvW3QwojkEvhsz4eHbMV1RScXKLiDoAnJTNKMScRb3wjdXu4GJ9a9e-q6gRqbn7k3zaXwGjokVpCg?key=T1D0B9_ILh_nNOdcc5Umtwwl** |

Overall, the training process showcased that the Block Detector model was able to significantly improve its pose estimation accuracy, as reflected by the reduction in pose\_loss from 11 in the first epoch to 0.6 by epoch 1000.

# Future work

In future work, we aim to enhance the performance of the line detector by improving the image restoration process. By optimizing image restoration techniques, we expect to provide higher-quality input data for the detection model, ultimately leading to more accurate line detection results. Specifically, we plan to refine the Unet and AST-based restoration methods to generate clearer and more structured images that improve the downstream detection tasks.

Another key improvement involves redefining the Intersection over Union (IoU) calculation for line detection. Previous models computed IoU based on bounding boxes, which is not an optimal approach for line-based structures due to their inherent geometric differences. We propose a novel IoU computation method specifically tailored for line detection, ensuring a more accurate evaluation metric that aligns better with the nature of the task.

Furthermore, we aim to reduce the computational load of our model by decreasing the number of input channels from 24 to 5. This reduction is expected to significantly enhance efficiency while maintaining detection performance. By optimizing the input representation, we can achieve faster inference times and reduce the memory footprint, making the model more suitable for real-time applications.

Overall, these enhancements will contribute to a more efficient and accurate line detection system, paving the way for further advancements in image processing and computer vision applications.

# References

[1] M. Hofer, M. Maurer, and H. Bischof, “Efficient 3D scene abstraction using line segments,” *Computer Vision and Image Understanding*, vol. 157, pp. 167–178, Apr. 2017, doi: 10.1016/j.cviu.2016.03.017.

[2] A. Mateus, O. Tahri, A. P. Aguiar, P. U. Lima, and P. Miraldo, “On Incremental Structure-from-Motion using Lines,” *IEEE Trans. Robot.*, vol. 38, no. 1, pp. 391–406, Feb. 2022, doi: 10.1109/TRO.2021.3085487.

[3] Q. Fu *et al.*, “PL-VINS: Real-Time Monocular Visual-Inertial SLAM with Point and Line Features,” Apr. 15, 2022, *arXiv*: arXiv:2009.07462. doi: 10.48550/arXiv.2009.07462.

[4] “LVO: Line only stereo Visual Odometry.” Accessed: Jan. 14, 2025. [Online]. Available: https://ieeexplore.ieee.org/document/8911808

[5] X. Zuo, X. Xie, Y. Liu, and G. Huang, “Robust visual SLAM with point and line features,” in *2017 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, Sep. 2017, pp. 1775–1782. doi: 10.1109/IROS.2017.8205991.

[6] M. Quan, Z. Chai, and X. Liu, “LOF: Structure-Aware Line Tracking based on Optical Flow,” Sep. 17, 2021, *arXiv*: arXiv:2109.08466. doi: 10.48550/arXiv.2109.08466.

[7] “Non-iterative approach for fast and accurate vanishing point detection.” Accessed: Jan. 14, 2025. [Online]. Available: https://ieeexplore.ieee.org/document/5459328

[8] R. Grompone von Gioi, J. Jakubowicz, J.-M. Morel, and G. Randall, “LSD: A Fast Line Segment Detector with a False Detection Control,” *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 32, no. 4, pp. 722–732, Apr. 2010, doi: 10.1109/TPAMI.2008.300.

[9] K. Huang, Y. Wang, Z. Zhou, T. Ding, S. Gao, and Y. Ma, “Learning to Parse Wireframes in Images of Man-Made Environments,” Jul. 15, 2020, *arXiv*: arXiv:2007.07527. doi: 10.48550/arXiv.2007.07527.

[10] Y. Lin, S. L. Pintea, and J. C. van Gemert, “Deep Hough-Transform Line Priors,” Jul. 18, 2020, *arXiv*: arXiv:2007.09493. doi: 10.48550/arXiv.2007.09493.

[11] N. Xue, S. Bai, F. Wang, G.-S. Xia, T. Wu, and L. Zhang, “Learning Attraction Field Representation for Robust Line Segment Detection,” Mar. 05, 2019, *arXiv*: arXiv:1812.02122. doi: 10.48550/arXiv.1812.02122.

[12] Y. Zhou, H. Qi, and Y. Ma, “End-to-End Wireframe Parsing,” presented at the Proceedings of the IEEE/CVF International Conference on Computer Vision, 2019, pp. 962–971. Accessed: Jan. 14, 2025. [Online]. Available: https://openaccess.thecvf.com/content\_ICCV\_2019/html/Zhou\_End-to-End\_Wireframe\_Parsing\_ICCV\_2019\_paper.html

[13] X. Dai, H. Gong, S. Wu, X. Yuan, and Y. Ma, “Fully Convolutional Line Parsing,” Dec. 29, 2022, *arXiv*: arXiv:2104.11207. doi: 10.48550/arXiv.2104.11207.

[14] G. Gu, B. Ko, S. Go, S.-H. Lee, J. Lee, and M. Shin, “Towards Light-Weight and Real-Time Line Segment Detection,” *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 36, no. 1, Art. no. 1, Jun. 2022, doi: 10.1609/aaai.v36i1.19953.

[15] S. Huang, F. Qin, P. Xiong, N. Ding, Y. He, and X. Liu, “TP-LSD: Tri-Points Based Line Segment Detector,” Sep. 11, 2020, *arXiv*: arXiv:2009.05505. doi: 10.48550/arXiv.2009.05505.

[16] “L2D2: Learnable Line Detector and Descriptor | IEEE Conference Publication | IEEE Xplore.” Accessed: Jan. 14, 2025. [Online]. Available: https://ieeexplore.ieee.org/document/9665961

[17] R. Pautrat, J.-T. Lin, V. Larsson, M. R. Oswald, and M. Pollefeys, “SOLD2: Self-supervised Occlusion-aware Line Description and Detection,” Apr. 09, 2021, *arXiv*: arXiv:2104.03362. doi: 10.48550/arXiv.2104.03362.

[18] H. Zhang, Y. Luo, F. Qin, Y. He, and X. Liu, “ELSD: Efficient Line Segment Detector and Descriptor,” Apr. 29, 2021, *arXiv*: arXiv:2104.14205. doi: 10.48550/arXiv.2104.14205.

[19] C. Wang, W. Wang, Y. Qiu, Y. Hu, and S. Scherer, “Visual Memorability for Robotic Interestingness via Unsupervised Online Learning,” 2020, pp. 52–68. doi: 10.1007/978-3-030-58536-5\_4.

[20] Y. Qiu, C. Wang, W. Wang, M. Henein, and S. Scherer, “AirDOS: Dynamic SLAM benefits from Articulated Objects,” May 2022, pp. 8047–8053. doi: 10.1109/ICRA46639.2022.9811667.

[21] H. Caesar *et al.*, “nuScenes: A multimodal dataset for autonomous driving,” May 05, 2020, *arXiv*: arXiv:1903.11027. doi: 10.48550/arXiv.1903.11027.

[22] P.-E. Sarlin, D. DeTone, T. Malisiewicz, and A. Rabinovich, “SuperGlue: Learning Feature Matching With Graph Neural Networks,” presented at the Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition, 2020, pp. 4938–4947. Accessed: Jan. 14, 2025. [Online]. Available: https://openaccess.thecvf.com/content\_CVPR\_2020/html/Sarlin\_SuperGlue\_Learning\_Feature\_Matching\_With\_Graph\_Neural\_Networks\_CVPR\_2020\_paper.html

[23] “SuperPoint: Self-Supervised Interest Point Detection and Description | Request PDF,” in *ResearchGate*, doi: 10.1109/CVPRW.2018.00060.

[24] V. N. Nguyen, R. Jenssen, and D. Roverso, “LS-Net: Fast Single-Shot Line-Segment Detector,” Jan. 24, 2020, *arXiv*: arXiv:1912.09532. doi: 10.48550/arXiv.1912.09532.

[25] R. O. Duda and P. E. Hart, “Use of the Hough transformation to detect lines and curves in pictures,” *Commun. ACM*, vol. 15, no. 1, pp. 11–15, Jan. 1972, doi: 10.1145/361237.361242.

[26] J. Canny, “A Computational Approach to Edge Detection,” *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. PAMI-8, no. 6, pp. 679–698, Nov. 1986, doi: 10.1109/TPAMI.1986.4767851.

[27] “(PDF) PLSD: A Perceptually Accurate Line Segment Detection Approach,” *ResearchGate*, Dec. 2024, doi: 10.1109/ACCESS.2020.2977119.

[28] P. Bernet, J. Chazalon, E. Carlinet, A. Bourquelot, and E. Puybareau, “Linear Object Detection in Document Images Using Multiple Object Tracking,” in *Document Analysis and Recognition - ICDAR 2023: 17th International Conference, San José, CA, USA, August 21–26, 2023, Proceedings, Part V*, Berlin, Heidelberg: Springer-Verlag, Aug. 2023, pp. 454–471. doi: 10.1007/978-3-031-41734-4\_28.

[29] H. Li, Y. Ma, H. Bao, and Y. Zhang, “Probabilistic Hough Transform for Rectifying Industrial Nameplate Images: A Novel Strategy for Improved Text Detection and Precision in Difficult Environments,” *Applied Sciences*, vol. 13, no. 7, Art. no. 7, Jan. 2023, doi: 10.3390/app13074533.

[30] J. Redmon, S. Divvala, R. Girshick, and A. Farhadi, “You Only Look Once: Unified, Real-Time Object Detection,” May 09, 2016, *arXiv*: arXiv:1506.02640. doi: 10.48550/arXiv.1506.02640.

[31] CHOOSEHAPPY, “Digital Pathology Segmentation using Pytorch + Unet,” Andrew Janowczyk. Accessed: Mar. 02, 2024. [Online]. Available: http://www.andrewjanowczyk.com/pytorch-unet-for-digital-pathology-segmentation/

[32] O. Oktay *et al.*, “Attention U-Net: Learning Where to Look for the Pancreas,” May 20, 2018, *arXiv*: arXiv:1804.03999. doi: 10.48550/arXiv.1804.03999.

[33] S. Zhou, D. Chen, J. Pan, J. Shi, and J. Yang, “Adapt or Perish: Adaptive Sparse Transformer with Attentive Feature Refinement for Image Restoration,” in *2024 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, Jun. 2024, pp. 2952–2963. doi: 10.1109/CVPR52733.2024.00285.

[34] X. Zhang, C. Hu, H. Liu, R. Du, X. Zhou, and L. Wang, “A Line Segment Detector for Space Target Images Robust to Complex Illumination,” *Aerospace*, vol. 10, no. 2, Art. no. 2, Feb. 2023, doi: 10.3390/aerospace10020195.

[35] E. Tretyak, O. Barinova, P. Kohli, and V. Lempitsky, “Geometric Image Parsing in Man-Made Environments,” *Int J Comput Vis*, vol. 97, no. 3, pp. 305–321, May 2012, doi: 10.1007/s11263-011-0488-1.

[36] A. Sindel, T. Klinke, A. Maier, and V. Christlein, “ChainLineNet: Deep-Learning-Based Segmentation and Parameterization of Chain Lines in Historical Prints,” *Journal of Imaging*, vol. 7, no. 7, Art. no. 7, Jul. 2021, doi: 10.3390/jimaging7070120.

[37] J. Yang *et al.*, “MLNet: An multi-scale line detector and descriptor network for 3D reconstruction,” *Know.-Based Syst.*, vol. 289, no. C, Jun. 2024, doi: 10.1016/j.knosys.2024.111476.

[38] N. Carion, F. Massa, G. Synnaeve, N. Usunier, A. Kirillov, and S. Zagoruyko, “End-to-End Object Detection with Transformers,” May 28, 2020, *arXiv*: arXiv:2005.12872. doi: 10.48550/arXiv.2005.12872.

[39] L. Teplyakov, L. Erlygin, and E. Shvets, “LSDNet: Trainable Modification of LSD Algorithm for Real-Time Line Segment Detection,” *IEEE Access*, vol. 10, pp. 45256–45265, 2022, doi: 10.1109/ACCESS.2022.3169177.