Object Detection and Depth Estimation for Drone Camera Images

RBE 577: Machine Learning for Robotics

Nate Dixon

Part 1: Object Detection

Objective:

- Detect cars in drone-captured images using bounding boxes.
- Fine-tune the YOLO v11 object detection model on the SynDrone dataset.

Dataset:

- **SynDrone (Town01 data)**:
  - RGB images captured from a drone in urban scenarios.
  - Annotated with bounding boxes for each object.
  - Focused only on vehicle detection (Cars and Trucks).

Methodology:

- **Model:**
  1. YOLO v11
  2. Optimized for real-time performance and high accuracy.
- **Steps:**
  1. Preprocessing the dataset.
  2. Creating a custom training dataloader for YOLO v11.
  3. Fine-tuning the model on SynDrone data.

YOLO v11

YOLO stands for *You Only Look Once* – a convolutional neural network designed for fast and efficient object detection model.

Purpose: Detects objects in images by drawing bounding boxes around them and classifying what they are.

How it works:

- Splits an image into a grid.
- Each grid cell predicts if an object is present, its location, and its type.

Why YOLO?

- **Speed**: Processes images in real-time.
- **Simplicity**: Detects multiple objects in a single pass through the model.
- **Accuracy**: Performs well even on complex scenes.

Ideal for tasks like detecting vehicles in drone images.

Dataset Preparation

Dataset: The project uses the SynDrone Town01 dataset, containing RGB images, semantic maps, depth maps, and camera metadata.

Bounding Box Conversion:

- 3D bounding box annotations in the SynDrone dataset are converted to YOLO format:
  - [class, x_center, y_center, width, height]
  - The conversion process includes:
    1. Shifting bounding box coordinates by camera offsets.
    2. Rotating based on camera orientation using Euler angles.
    3. Projecting 3D bounding boxes into 2D using the camera intrinsic matrix.
    4. Filtering out invalid boxes.

Annotation Files:

- Converted YOLO bounding boxes are saved in .txt files, with filenames matching the corresponding image files.

Data Preparation:

- Images and annotation files are organized into folders.
- The dataset is structured to comply with YOLO model input requirements.

Training

Model Loading:

- Pre-trained YOLO model is used as the base for fine-tuning on the SynDrone dataset.

Dataset Input:

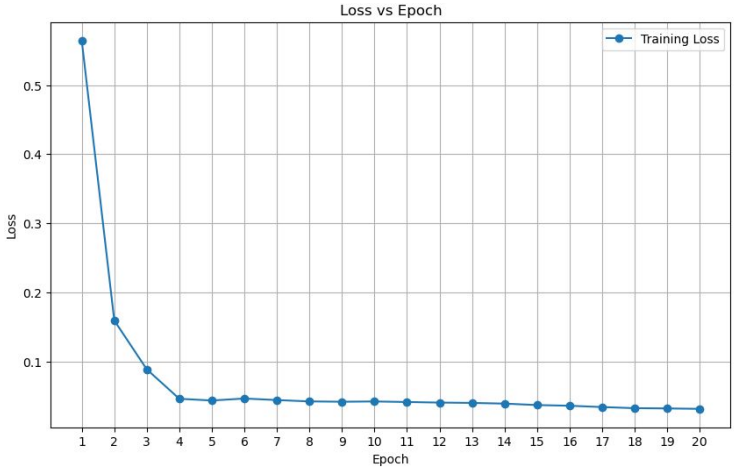
- The training and validation datasets, along with class labels, are defined in a YAML configuration file.

Training Process:

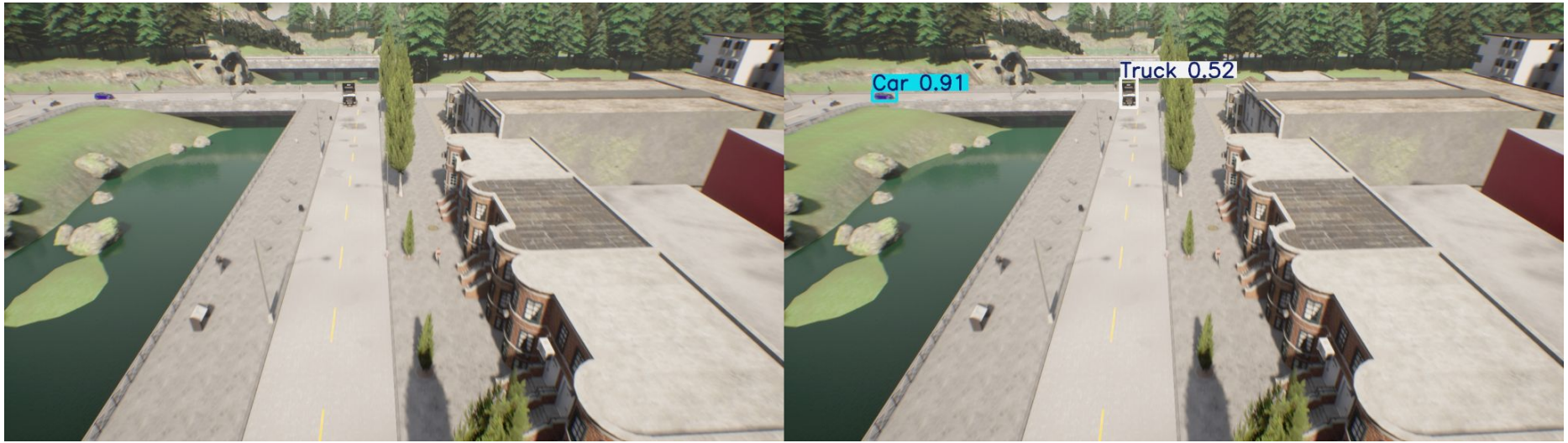
- The model learns to detect objects by adjusting its weights over multiple epochs.
- Key techniques include learning rate adjustment, weight decay, and data augmentation to improve accuracy and prevent overfitting.

Validation and Export:

- The model's performance is evaluated on a validation set after training.



Results



Results

Results

Results

Challenges

Key Challenge:

- **Bounding Box Conversion**:
  - The SynDrone dataset uses a different bounding box format compared to YOLO.
  - Required precise transformation to YOLO's normalized center-coordinate format:
    - (x_center, y_center, width, height).

Why It Was Challenging:

- **Format Differences**:
  - SynDrone bounding boxes were defined with pixel-based coordinates.
  - YOLO requires normalized values relative to image dimensions.
- **Data Validation**:
  - Ensuring no data corruption or mismatches during conversion.

Part 2: Depth Estimation

Supervised Depth Estimation:

- Based on the method proposed by R. Ranftl et al., *"Vision Transformers for Dense Prediction" (2021)*.
- Vision Transformer (ViT) backbone excels in capturing long-range dependencies in the image

Key Steps:

1. **Dataset Preparation**:
    - SynDrone depth maps were processed to ensure compatibility with the Vision Transformer model.
    - Depth maps normalized to appropriate scales for training.
2. **Training**:
    - Fine-tuned the Vision Transformer model on the SynDrone dataset.
    - Used supervised loss functions to train on ground-truth depth maps.
3. **Evaluation**:
    - Compared predicted depth maps to ground-truth depth maps.
    - Visualized results and quantified performance using loss curves.

Data Loader

Dataset Loading:

- Reads RGB images and corresponding depth maps from specified folders.
- Uses a split file to separate training and testing data.

Preprocessing:

- Resizes RGB images and depth maps to a fixed resolution (384x384).
- Normalizes RGB images and converts depth maps to inverse depth format.

Model Initialization:

- Loads a pre-trained Vision Transformer (ViT) based Depth Estimation model.

Dataloader Setup:

- Creates batches of preprocessed RGB images and depth maps for training and evaluation.

Training

Setup:

- Initializes the pre-trained DPT depth model and dataset loaders.

Training Process:

- The model is fine-tuned over multiple epochs using the training data.
- For each batch:
  - The model makes predictions.
  - A loss function calculates the error between predictions and ground truth.
  - The model updates its weights to minimize this error.

Evaluation:

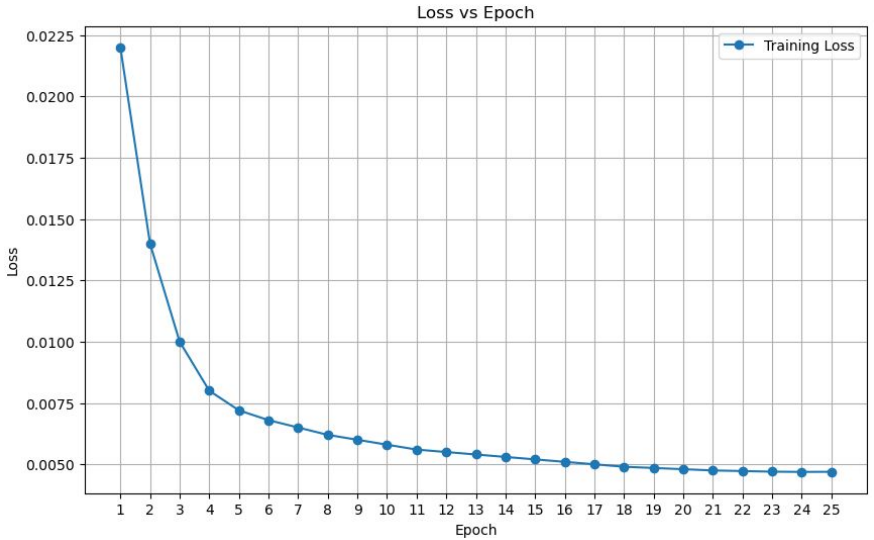
- After each epoch, the model's performance is tested on validation data to monitor its accuracy.

Learning Rate Adjustment:

- Gradually reduces the learning rate to fine-tune the model’s performance over time.

Model Saving:

- Saves the model weights at each epoch and finalizes training by saving the best model.



Results

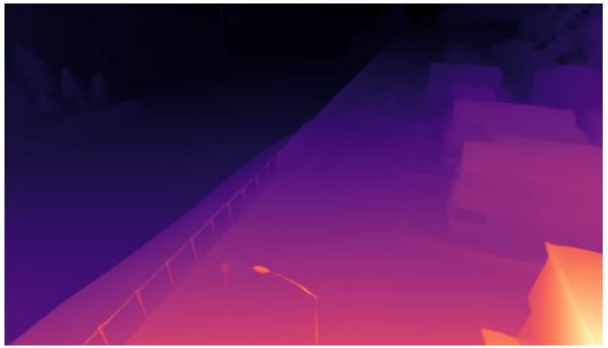
Original Image

Predicted Depth Map

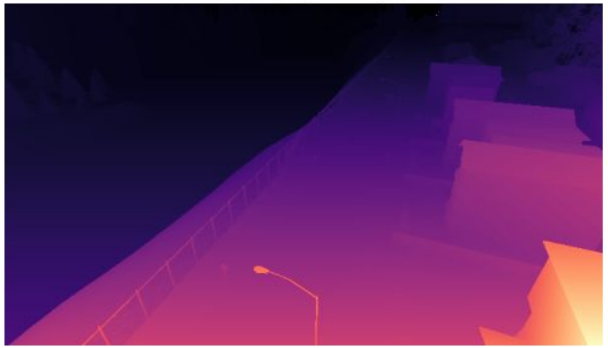
Truth Depth Map

Original Image

Predicted Depth Map



Truth Depth Map

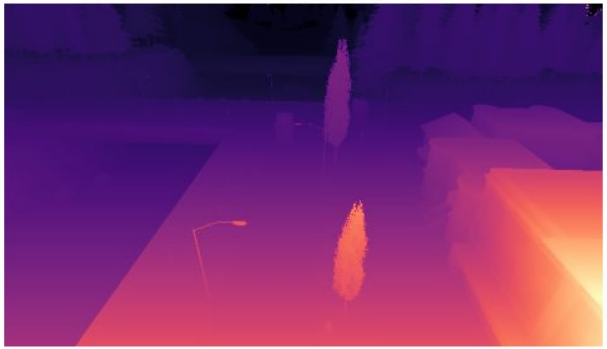


Results

Original Image

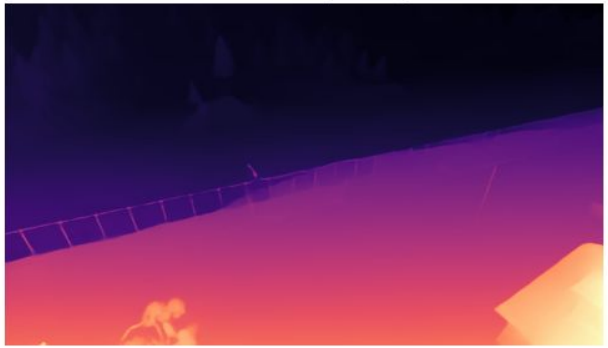
Predicted Depth Map

Truth Depth Map

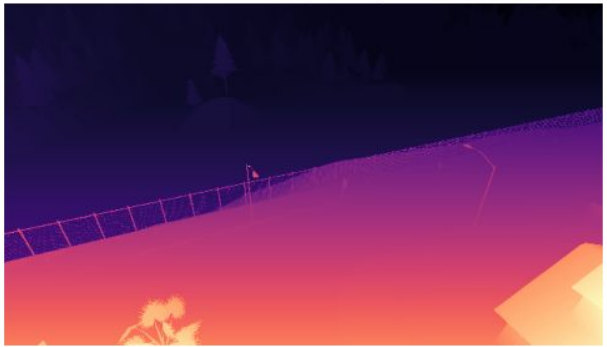


Original Image

Predicted Depth Map



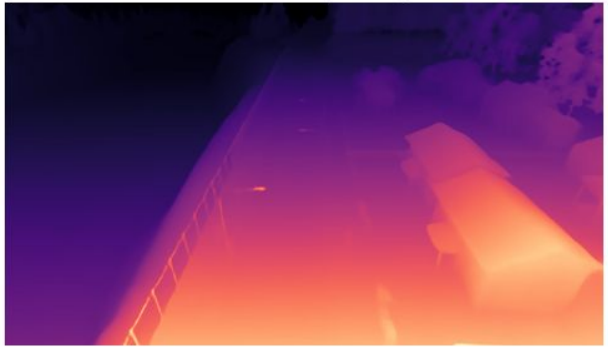
Truth Depth Map



Results

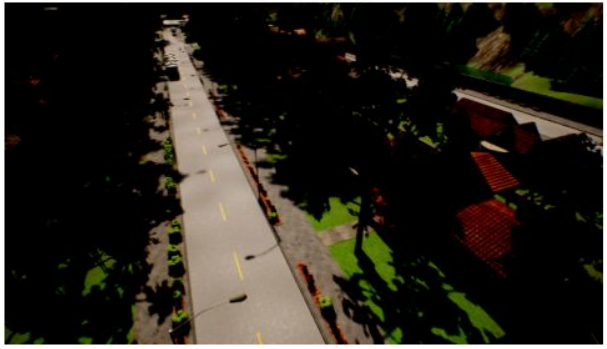
Original Image

Predicted Depth Map



Truth Depth Map

Original Image



Predicted Depth Map

Truth Depth Map

Challenges

No Training Script Provided

- *Challenge*: The provided Vision Transformer-based model lacked a training script, requiring significant effort to implement one from scratch.
- *Solution*: Developed a custom training pipeline, including a dataset loader, loss functions, and optimization routines, tailored to the SynDrone dataset.

Different Image Dimensions

- *Challenge*: Images and depth maps had mismatched resolutions, leading to alignment issues during training.
- *Solution*: Applied consistent resizing and preprocessing techniques, ensuring both images and depth maps matched the input size required by the model.

Slow Training on CPU

- *Challenge*: Limited computational resources significantly slowed down training, especially for Vision Transformer models with large input sizes.
- *Solution*: Used techniques like smaller batch sizes, learning rate scheduling, and efficient data loading to optimize training time. Considered downsizing model input dimensions when necessary.

Conclusion

Overall Success

- The project successfully achieved its objectives of object detection and depth estimation using the SynDrone dataset.

Object Detection

- The YOLO model demonstrated high accuracy, correctly detecting objects in the majority of test cases.

Depth Estimation

- The depth estimation model performed well but showed potential for improvement in certain scenarios.

Future Work

- Continuous training and fine-tuning of both models to enhance accuracy and robustness.
- Exploring additional techniques or architectures to improve depth estimation further.
- Optimizing computational resources for faster and more efficient training.