2nd Floor Street Images Classification

Kabir Bagai
kb3343
Columbia University

Riki Shimizu
rs4613
Columbia University

Arsalan Firoozi
af3410
Columbia University

I. Introduction

This project uses a dataset of images of street traffic captured at the Amsterdam Ave. and 120th Street intersection. The goal is to build a model that can effectively identify rare classes and situations that have the potential to disrupt traffic flow or cause harm to pedestrians. This task is more challenging than typical image classification tasks due to the complexity of the images and the limited dataset size. By leveraging a pre-trained model Recognize Anything, a vision-language model, we achieved a classification performance of 0.716 on the test set, the highest score on the Kaggle in-class competition.

II. Approach and Methodology

A. Motivation and Dataset Challenges

There are a few features from the data that we need to consider:

- **Small Sample Size:** The biggest problem is we have 100 images with just labels (not bounding boxes). With such data, training is not possible. So, we could either do fine-tuning or use it as a validation set to evaluate different models. Either possible way is all about inference, using models to find out objects in the image.
- **Class Imbalance:** The classes are highly imbalanced. Barricade: 97, Ambulance: 3, Moving Truck: 1, Construction Car: 5, Construction Truck: 8, Construction Worker: 3, Excavator: 16, Firecar: 4, Offroad Parking: 3, Police Car: 5, Scaffold: 99, Traffic Barrel: 95, Traffic Cone: 20, Trailer Truck: 11. Categories like scaffold and traffic barrel are saturated in the data, which is not good. Among the 100 images in the training set, some classes, such as ”Barricade,” ”Traffic Barrel,” and ”Scaffold,” appeared more than or equal to 95 times, while other classes (e.g., ”Car Moving Truck,” ”Offroad Parking”) appeared less than 5 times.
- **High-Resolution Images:** An image in the dataset has a resolution of 3840 by 2160. Although it is a good thing about the data, representing the whole image leads to a loss of information due to resizing. This dimension is significantly larger than images used for training Recognize Anything. Therefore, we manually introduced slicing to the original images, breaking them down into a collection of 98 sliced images. For each class, we took the maximum probability across sliced images as our final probability output before thresholding.

B. Initial Approaches

- *YoloV11 Sahi Tiled Inference*: We started with SAHI Tiled Inference (Link). In this package, the image is split into overlapping windows, and an object detection model is applied to each tile. I set the window size to 256×256 pixels with a 0.2 overlap. For the object detection model, we experimented with several options: YOLOv11x-seg, YOLOv8x-worldv2, RT-DETR-x, and YOLOv11x.
  As shown in figure 1, the predefined 80 labels in YOLO do not include ”traffic barrel.” One workaround was to repurpose the ”fire hydrant” class to represent a traffic barrel, and to use the combination of ”person” and ”stop sign” to represent a construction worker. Similarly, a label like ”off-road parking” was used for a car located in the bottom-left corner of the image.
  However, this approach had several issues. First, the model’s accuracy was poor. Second, some critical objects—such as scaffolding—were not detected at all. Additionally, SAHI’s bounding box merging function, which aggregates results from the tiled windows into a final image prediction, made it nearly impossible to apply modifications manually.
  Tuning the object detection confidence threshold also proved problematic: increasing the threshold reduced false positives but caused important objects, like traffic barrels, to be missed. In the end, I couldn’t find a robust way to use SAHI tiled inference effectively with any of the object detection models we tested.
- *EfficientNet*: We experimented with fine-tuning a pre-trained EfficientNet, a purely convolutional model [2]. We leveraged the dataset of 100 labeled training images along with an augmented duplicate set to enhance diversity. To address class imbalance, we employed focal loss, aiming to increase the weight given to rare classes [3]. While this approach yielded a relatively high average balanced accuracy of 77% on the training set, it struggled to generalize, achieving only 51% balanced accuracy on the test data.
  We knew that the training data samples were not sufficient for proper training, as we needed to classify objects that were not present in any of the existing models. Therefore, we decided to use zero-shot object detection models, which allow us to detect previously unseen objects. We ultimately chose the Recognize Anything model [1] because its performance was promising during our search

for zero-shot image and object classification solutions.

C. Recognize Anything Model (RAM)

Recognize Anything [1] is a powerful vision-language model trained on large-scale image-text pairs, which enables it to generalize effectively across diverse visual concepts. Uniquely, it accepts open-ended text descriptions as input, allowing for flexible and fine-grained recognition without being constrained to a fixed set of categories. It is designed to recognize a diverse range of objects across various domains without requiring extensive fine-tuning.

Installing the package took a few days! However, it performed relatively well for detecting large objects.

We first defined 14 objects using the GPT API and stored them in a JSON file (Link). I tested the model on this dataset, but as expected, due to the high resolution of the images, the results were not satisfactory.

To address this, we sliced the images into windows of size 512×512, with a 50% overlap between adjacent windows. However, we needed a method to merge the results from these smaller windows.

To achieve this, I modified the model's method. Specifically, I added a function to return the probabilities of all classes instead of only the final tags (Link, function: `generate_tag_openset_prob` located in `RAM_path/ram/models`).

Next, I extracted the probabilities for all windows and took the maximum probability of each class across all subwindows of an image. As a result, each image is represented by 14 probability values (Code: Link), which are saved in Link.

The only remaining task is to determine the significant probability threshold for each class. For this, we used the training data, as described in the main notebook section: *"Extract threshold of significance for RAM"*.

In summary, we were able to generate predictions without any fine-tuning on the model, simply by providing specific descriptions for our labels, from which the model could generate text embeddings. After the model made predictions on the training data, we used F1 scoring to determine optimal thresholds for classifying each label.

III. RAM Implementation

In this section, we outline the key technical components of our system. First, we describe the foundational *Recognize Anything* model, which underpins our architecture for open-vocabulary object detection and classification. Then, we detail the specific pipeline we employed, including how we slice the images and apply class-dependent thresholding.

A. Recognize Anything Plus

a) Model Overview.: We build upon the *Recognize Anything* model (RAM) [1], a vision-language model that computes alignment scores between image representations and text representations. Let $\mathcal{C} = \{c_1, c_2, \ldots, c_{14}\}$ be the set of 14 classes under consideration (e.g., `Barricade`, `Construction Worker`, `Offroad Parking`, etc.). We denote the model's *image encoder* by

$$\mathbf{h}_\psi : \mathcal{X} \to R^d,$$

where $\mathcal{X}$ is the input image space and d is the dimensionality of the latent space. Similarly, we denote the *text encoder* by

$$\mathbf{g}_\phi : \mathcal{T} \to R^d,$$

where $\mathcal{T}$ is the space of textual inputs, and ϕ is the learned parameter set for the text encoder.

The fundamental output of the model for an image I and textual description t is an alignment score, which we denote as

$$A(I, t) \;=\; \mathrm{sim}\big(\mathbf{h}_\psi(I), \mathbf{g}_\phi(t)\big),$$

where $\mathrm{sim}(\cdot, \cdot)$ is a similarity function. After appropriate calibration (e.g., via a logistic or softmax transformation), we interpret this alignment score as a probability $\hat{p}(I, t)$ of the text t matching the image I.

b) Augmenting Textual Descriptions.: To improve recognition of underrepresented or nuanced classes, we generate diverse textual descriptions for each class $c_i \in \mathcal{C}$. Specifically, we query GPT-3.5-turbo (via the OpenAI API) to produce multiple detailed statements about each class. Formally, for each class c_i, we obtain a set of descriptions

$$D(c_i) = \{t_{i,1}, t_{i,2}, \ldots, t_{i,k}\},$$

where $k = 50$ in our experiments. Each $t_{i,j}$ is a unique natural language description (e.g., for `Barricade`: *"A barricade is typically a temporary structure made of metal, wood, or other materials..."*).

We feed each description $t_{i,j}$ through the text encoder $\mathbf{g}_\phi$ to obtain embeddings $\mathbf{g}_\phi(t_{i,j})$. This augmented textual representation helps the model account for broad variations of each concept, improving recall for rare classes.

c) Combining Class Probabilities.: Once we have the textual descriptions for a class c_i and the image encoding $\mathbf{h}_\psi(I)$, the model computes alignment scores

$$A(I, t_{i,j}) = \mathrm{sim}\Big(\mathbf{h}_\psi(I), \mathbf{g}_\phi\big(t_{i,j}\big)\Big), \quad \text{for all } j = 1, 2, \ldots, k.$$

In practice, we interpret these as class probabilities via a calibration layer (e.g., a logistic function):

$$\hat{p}\big(I, t_{i,j}\big) = \sigma\Big(A\big(I, t_{i,j}\big)\Big),$$

where σ is the softmax transformation. To derive a single probability for class c_i, we assign different weights to outputs across all textual variants:

$$\hat{p}(I, c_i) \;=\; \sum_{j=1}^{k} w_j \hat{p}\big(I, t_{i,j}\big).$$

B. Detection Pipeline

a) 1. Image Slicing.: One unique aspect of our dataset is the large resolution of each image (3840×2160). To handle this effectively within the RAM model's computational constraints, we subdivide each raw image I_{raw} into a set of smaller tiles:

$$\mathcal{S}(I_{\text{raw}}) = \{I_1, I_2, \dots, I_m\},$$

where $m = 96$ slices, each of dimension 512×512. We then resize each slice to 384×384 for input to the RAM model. Let $I_s \in \mathcal{S}(I_{\text{raw}})$ denote the s-th sliced image.

b) 2. Class Probability Aggregation.: For each slice I_s and each class c_i, we compute the probability $\hat{p}(I_s, c_i)$ using the procedure described above. Then, we aggregate these probabilities over all slices by taking:

$$\hat{p}(I_{\text{raw}}, c_i) \;=\; \max_{1 \le s \le m} \; \hat{p}(I_s, c_i).$$

The intuition here is that if *any* slice of the large original image strongly exhibits class c_i, we want our final classification for that class to be high.

c) 3. Class-Dependent Thresholding.: Due to the imbalance in class frequencies and the varying difficulty levels across classes, we apply a separate threshold τ_i for each class c_i. Formally, the predicted label $\hat{y}_i$ for class c_i is computed via

$$\hat{y}_i = \begin{cases} 1, & \text{if } \hat{p}(I_{\text{raw}}, c_i) \;\ge\; \tau_i, \\ 0, & \text{otherwise.} \end{cases}$$

To determine τ_i for each class, we perform a grid search over $\{0, 0.01, 0.02, \dots, 1.0\}$ using the training set. Specifically, we define

$$\tau_i^* = \arg \max_{\tau \in \{0, 0.01, \dots, 1\}} \text{F1-score}\big(\tau; c_i\big),$$

where $\text{F1-score}(\tau; c_i)$ evaluates the precision and recall for class c_i on the training data when threshold τ is applied. This class-dependent thresholding directly addresses the high imbalance and prevents overly frequent false negatives or false positives in rare classes.

d) 4. Final Inference.: At inference time, for each test image:

1) We slice and resize the image into $m = 96$ tiles of 384×384.
2) For each tile I_s and each class c_i, we compute the weighted probability across the set of textual prompts $D(c_i)$.
3) We aggregate probabilities over slices using a max operation to obtain $\hat{p}(I_{\text{raw}}, c_i)$.
4) We compare $\hat{p}(I_{\text{raw}}, c_i)$ to τ_i^* for final classification.

This pipeline yields a multi-label prediction vector $\hat{\mathbf{y}} \in \{0,1\}^{|\mathcal{C}|}$, indicating the presence or absence of each class in the raw image.

IV. Results

A. Sahi - Sliced Inference

You can see object detection result on a sample image from data with Sahdi YoloV11 in Figure 1.

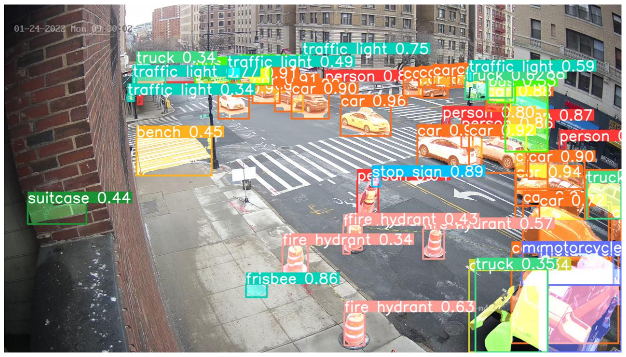

Fig. 1. A sample output of Sahi with YoloV11 - As you can see 1. not all objects are detected (Scaffold) 2. Too many false positives 3. No clear significant threshold to consider for filtering the objects.

B. EfficientNet, RAM

As these models do not provide bounding boxes, we measured performance by computing the balanced accuracies of each model for different categories on both the training data and the average balanced accuracy on the test data (Table 1).

Class Name	**RAM+**	**EfficientNet**
Barricade	0.88	0.52
Traffic cone	0.65	0.72
Traffic barrel	0.99	0.52
Scaffold	0.99	0.52
Trailer truck	0.72	0.68
Police car	0.60	0.64
Ambulance	0.83	0.99
Firecar	1.00	0.99
Excavator	0.94	0.65
Construction truck	0.60	0.65
Car moving truck	1.00	0.99
Offroad parking	0.52	0.99
Construction car	0.55	0.89
Construction worker	0.58	0.99
Mean Accuracy - Train	**0.77**	0.77
Mean Accuracy - Test	**0.72**	0.51

TABLE I
Balanced Accuracies for Each Class on Data

V. Discussion

A. Performance

While both models demonstrate similar average performance on the training set (0.77), RAM+ significantly outperforms EfficientNet on the test set (0.72 vs. 0.51), suggesting that RAM+ generalizes better to unseen data.

Class-wise, RAM+ shows particularly strong performance on construction-related categories such as ”Traffic barrel” (0.99), ”Scaffold” (0.99), and ”Excavator” (0.94), whereas EfficientNet underperforms on these same classes, scoring around 0.52 to 0.65. In contrast, EfficientNet achieves high accuracy on categories like ”Ambulance” (0.99), ”Firecar” (0.99), and ”Construction worker” (0.99), where RAM+ scores more modestly. These differences suggest that while EfficientNet may memorize frequent, visually distinct objects well, RAM+ is more robust across a broader range of object types, especially those less represented or more visually ambiguous in the training set.

B. Future Work

As the next steps of the study, similar to positional encoding of the tokens in vision transformer based models, we can consider positional embedding of each subwindow for deciding objects in the frame. This can be so helpful since the camera is fixed and some of the objects can only be found in specific locations of the scene (like offroad parking). Another thing possibly worth of doing is to add images of other intersections (similar to the data we have) to make the dataset better and more balanced. These issues can be addressed in further studies.

VI. Individual Contributions

All of us were involved in different parts of the study and helped each other for the aim of the project.

UNI	**Contributions**
kb3343	Equal contribution to EfficientNet, RAM, and Report.
rs4613	Equal contribution to EfficientNet, RAM, and Report.
af3410	Equal contribution to EfficientNet, RAM, and Report.

References

[1] Huang, Xinyu, Yi-Jie Huang, Youcai Zhang, Weiwei Tian, Rui Feng, Yuejie Zhang, Yanchun Xie, Yaqian Li, and Lei Zhang. *Open-Set Image Tagging with Multi-Grained Text Supervision*, arXiv preprint arXiv:2310.15200. (2023).

[2] Tan, Mingxing and Quoc V. Le. "EfficientNet: Rethinking Model Scaling for Convolutional Neural Networks." ArXiv abs/1905.11946 (2019): n. pag.

[3] Lin, Tsung-Yi et al. "Focal Loss for Dense Object Detection." IEEE Transactions on Pattern Analysis and Machine Intelligence 42 (2017): 318-327.