Geolocation from Latent Image Features: Through Text Extraction, Object Detection, and colour Analysis with Attention Mechanisms

Noah King[a*]

[a]Department of Software Engineering and Computer Science, University of Canterbury, Christchurch, New Zealand

*Corresponding author: Noah King; nki38@uclive.ac.nz

Abstract

We propose a method for multi-neighbourhood scale geolocation using open-source and open-license machine learning models to generate embeddings from street-level imagery. Our approach involves two networks: one based entirely on signage extracted using fine-tuned object detection models, and another that incorporates text embeddings and detected object frequencies. Both networks generate embeddings from explicitly defined geoinformative features, including text, object frequencies, and colour space information. These embeddings are used to train attention-inclusive neural networks to detect locations based on overall frequencies, as well as the distribution and embedded content of advertising and signage in cityscapes. The signage model achieves useful results, with a location accuracy of within 1 km over a 20 km^2 area, while the main model replicates the input dataset distribution.

1 Overview

This paper proposes a method for determining the geolocation of a street-level image at city scale using text detection, object detection, and color space analysis as inputs to a deep learning model. Ultimately, users will be able to submit an image captured somewhere in Chicago, and the model will produce a tuple for (LAT and LONG) that is within a reasonable distance. Such a model has applications for theft recovery, geotagging aged media or verifying dashcam footage, among other intuitive applications.

2 Introduction

GeoLocalization of imagery is a well-defined domain with intuitive applications including safety and navigation. While geotagged datasets are readily available from online sources, accurately identifying the location of an unseen image remains a challenging problem. The task requires identifying *GeoInformative Features*—distinctive local features that sufficiently differentiate one location from another. However, extracting and effectively leveraging these features poses several challenges, including feature importance ambiguity and cross-location similarities.

At its core, GeoLocalization essentially asks: *What features define a location?* More formally, which image features provide reliable geographic cues? A *GeoInformer* is a feature that offers strong geolocation signals. These can be explicit (i.e. text from signage) or implicit (i.e. environmental color distributions or frequency of objects). Notably, a GeoInformer can be composed of multiple less-informative features. Traditional approaches to geolocalization often rely on learned GeoInformers, but these can be prone to noise, overemphasizing certain features and leading to suboptimal weighting or feature over reliance. MultiModal transformers, which translate images into textual semantics, offer a promising way to bridge these gaps by integrating diverse contextual information but are *as at 2025* memory intensive and too large to run with useful performance on local hardware.Typically models for GeoLocalization are not time invariant and the time of capture of the dataset can be a significant contributing factor to the mdoels *decisions* in geolocation.

The popular online game GeoGuessr gamifies the geolocation process, demonstrating how players recursively narrow down possible locations by analyzing decreasingly verbose image features. This hierarchical reasoning aligns well with machine learning techniques such as attention mechanisms and feature extraction. For instance, textual elements like street signs or globally recognizable landmarks significantly reduce the set of possible locations. Text-based signals—such as business names, phone numbers, and web domains—offer particularly strong geolocational cues due to their specificity. Noisy GeoInformative patterns are trivially dealt with by players weighting GeoInformers.

Advances in machine learning, particularly in pretrained models for tasks such as Optical Character Recognition (OCR) and Object Detection, introduce new, scalable methods for geolocalization without requiring extensive preprocessing or image annotation; notably, these models make determining *explicit* geoinformers comparatively trivial. Similarly, the introduction of Attention[47] in machine learning has somewhat simplified noise reduction and extraction of the overall most important features in some datasets.

Data Procurement and Scope data is collated from open source datasets; primarily, the *Mapillary Dataset* comprising mostly still frame DashCam captures. A combination of Google Street View (Maps service) images are included in the dataset.

Proposed Method We introduce a geolocalization approach that extracts and analyzes **GeoInformative Features**. Our method leverages fine-tuned *off-the-shelf* object detection models to identify key geographic indicators such as signage, crosswalks, and architectural elements in street-level imagery. Additionally, we employ an OCR model to extract text, which is transformed into higher-dimensional embeddings using a pretrained **text embedding** model. To capture implicit environmental cues, we generate **HSV histogram embeddings**, which encode color distributions indicative of geographic location, such as building facades, vegetation, and sky proportions. By integrating these explicit and implicit features, we create a comprehensive embedding that encodes both structured and ambient location characteristics.

Our method performs GeoLocalization at the **multi-neighborhood scale**, trained on approximately *150,000* images spanning a 20 km^2 area. Unlike most models that operate at broader scales, our approach focuses on finer granularity, fitting in the last hop of geolocation. Three distinct machine learning models are employed to geolocate an unseen image:

- **K-means clustering:** Clusters images based on text embeddings and object frequencies, leveraging natural groupings within the dataset.
- **CNN with Attention:** Predicts latitude and longitude by analyzing extracted signage, focusing on text and color-space features.
- **Multi-Dense Network with Attention:** Processes full-image text, color-space features, and object embeddings to refine geolocation predictions.

The output Latitudinal and Longitudinal Coordinates are simply averaged; however, the proposal shows how transfer learning can be used to further enrich the model output.

Each model extracts meaningful spatial information from different feature sets, enabling complementary geolocational reasoning. A **MultiHead Attention** layer [47] is employed to dynamically weight the most informative features, enhancing the model's ability to focus on verbose GeoInformers across text and Object Embeddings. Attention mechanisms allow the model to prioritize certain features over others, effectively filtering out noise and emphasizing the most relevant information for accurate geolocation. This proposal includes a brief ablation study contrasting the efficacy of the attention layers in place of a typical CNN / Multi-Dense network.

Our approach addresses the issue of misalignment and weight management (i.e., overpowering signals) through the introduction of Attention layers and through the *pseudo ensemble* of the three models. The model is trained and replicable on consumer hardware. Off-the-shelf models are used to generate embeddings which are trained using custom architectures.

3 Background And Prior Work

Geolocalization is a complex problem with a well-defined upper bound. However, the intricacy of the problem becomes intractable long before reaching that theoretical limit. Image geolocalization specifically deals with a vast number of street-level locations and a multitude of varying local signals, such as seasonality, luminance variations, viewing angles, traffic conditions, and alterations to geoinformers like building maintenance or road changes; all of which can introduce noise or diminish the usefulness of GeoInformers in GeoLocalizaers.

Approaches employ a mix of terminologoy; we define the following as blanket terms.

- **GeoInformer / GeoInformative Feature**: A local feature in an image that can identify the location or contribute to a location regression. Examples include foliage, text, local architectural features, and road directions.
- **GeoInformative Pattern**: A collection of **GeoInformers**.
- **GeoLocalizer**: A model that uses a **GeoInformative Pattern** to predict the location where an image was captured.
- **Geocell**: An area which contains some number of images for training, or represents a possible output for a classification network

3.1 Dataset Representation and Bias

GeoLocalization is a domain which *inherently* requires significant training information.

Public datasets, such as those used in geoinformers, often exhibit significant biases that can impact model performance. Per Torralba et al[43]; popular datasets frequently overrepresent certain regions while underrepresenting others. This imbalance can lead to models that perform well on specific datasets but struggle to generalize across different or less-documented areas. Torralba et al. highlight that datasets collected with the intention of representing the visual world 'in the wild' can inadvertently become 'closed worlds,' limiting a potential model's ability to accurately interpret diverse real-world scenarios.

Despite advancements in dataset size and quality, the issue of bias remains.. Modern studies continue to cite Torralba et al.'s work, acknowledging that even with larger and more comprehensive datasets, biases persist, including Scene Analysis papers published as recently as *2025* including PadaNet [**PadaNet**] which addresses the issue through *optimal transport and game-theoretic alignment.* Approaches to density management are presented in PlaNet [49] and Pigeon[17] 3.3 3.4

These biases can affect the density and distribution of data, ultimately, if not considered influencing the accuracy and reliability of geoinformers in less-documented regions. Addressing these biases is crucial for developing robust and generalizable models that can perform effectively across varied geographic and visual contexts.

3.2 General Geolocalizers

Several approaches to GeoLocalization have been employed in recent years to varying success. The core principle in GeoLocation is *quality of GeoInformers.* The granularity of GeoInformers is also in ways proportional to the intended precision of the geolocalizer. For example, a roadworks sign that is common in Australia and New Zealand is a telling Country scale GeoLocator, but provides no insight at a smaller scale and would simply introduce noise therein.

Forward advancces in Object Detectors, Scene Analysis and Large Language models with Visual Adapters have seen significant changes to the methods in which GeoLocation is approahced.

3.3 Convolution-based Locators

Convolutional Neural Networks[25] (CNNs) excel in object classification by learning hierarchical features from raw image data (post preprocessing), making them a natural fit for identifying geolocative features like vegetation patterns, architectural styles, and road signage (i.e. Geoinfromers). A CNN based approach to GeoLocalization is **PlaNet** [49], which combines CNNs with Long Short-Term Memory (LSTM[20]) networks to predict image locations. In PlaNet, LSTM networks accumulate observations by sequentially processing multiple images, capturing temporal dependencies and refining location predictions through the integration of spatial and temporal data. This approach improves geolocalization accuracy by 50% over single-image models, leveraging context from previous observations to enhance overall performance. CNNs are also widely used in applications like Google Lens, further demonstrating their effectiveness in production environments.

PlaNet's CNN is trained on millions of geotagged images from public datasets like Flickr and Google Street View, covering diverse visual features. This enables the model to respond to 26,263 multi-scale geocells, creating an interpretable loss function and straightforward output distribution. However, reliance on public datasets can introduce biases, overrepresenting certain regions and underrepresenting others, limiting the model's accuracy in less-documented areas.

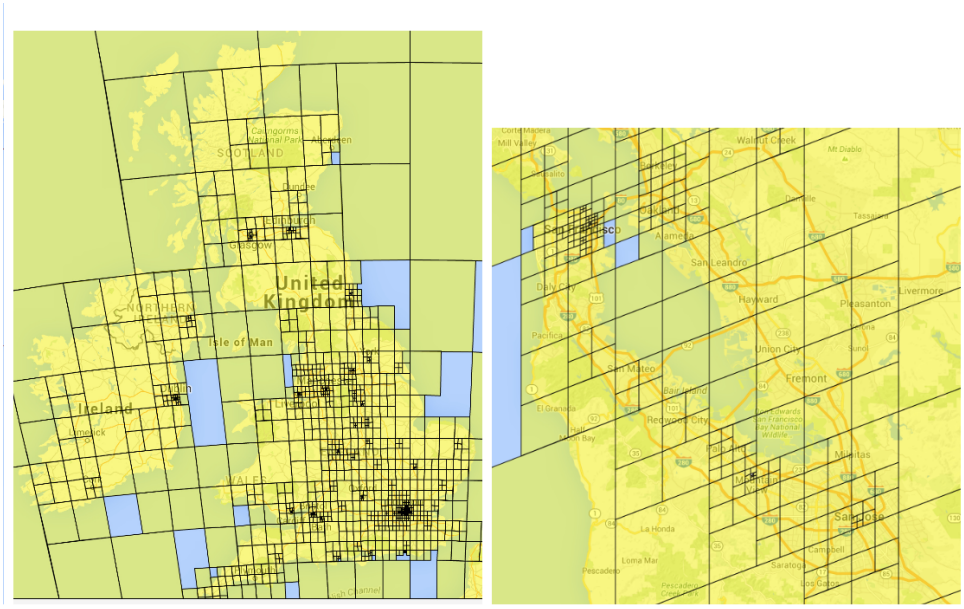


Figure 1: PLANET[49] query cells using S2 partitioning[13]

PlaNet learns geoinformative features through its convolutional layers, capturing broad features like vegetation and architectural styles, as well as specific local features like vehicle types. However, this can lead to challenges such as overfitting to spurious correlations or failing to distinguish between visually similar but geographically distinct features. For example, classic American vehicles might be misidentified as Cuban even if the image is from Australia due to a mismatch in GeoInfromative *strength* of features

Figure 2: Examples of Poorly Performing GeoLocalization in PlaNet [49]

This highlights the limitations of relying solely on visual similarity without additional contextual information. CNN models are challenging to interpret once trained, making it difficult to align weight distributions with actual features across a broad dataset.

To address uneven geotagged image distribution, PlaNet uses **adaptive partitioning** with Google's S2 geometry library [13], creating smaller geocells in dense areas and larger cells in sparse regions. While this mitigates some data density imbalances, it doesn't fully resolve dataset bias and may skew predictions towards areas with higher data density.

PlaNet performs well in diverse scenes but struggles with ambiguous locales. For instance, an old American vehicle in Australia might be misidentified as being in Cuba due to visual similarities. This shows strongly the need for additional modalities like textual metadata or temporal information to disambiguate visually similar but geographically distinct locations,

A key strength of PlaNet is its ability to learn relevant features without manual intervention, enhancing adaptability and generalizability. However, this also introduces interpretability challenges, as the model operates as a *black box*, making it difficult to understand which features contribute most to its predictions.

Additionally, training PlaNet is computationally expensive, having required 2.5 *months* of compute over 200 CPU cotes. This raises concerns about scalability and accessibility and ongoing refinement as geodata is inherently time invariant.

3.4 Transformer Based Locators

3.4.1 CLIP and Bridging the Gap Between Language and Objects

In 2021, OpenAI introduced CLIP[31] (Contrastive Language-Image Pre-training), a model that integrates computer vision and natural language processing. Unlike traditional models that require fine-tuning for specific datasets or tasks, CLIP learns from a vast array of image-text pairs. This enables it to understand and interpret images based on natural language descriptions, allowing it to perform a wide range of visual tasks without being explicitly optimized for any single benchmark.

CLIP allows the implementation of custom classifiers **without** requiring task-specific training data.

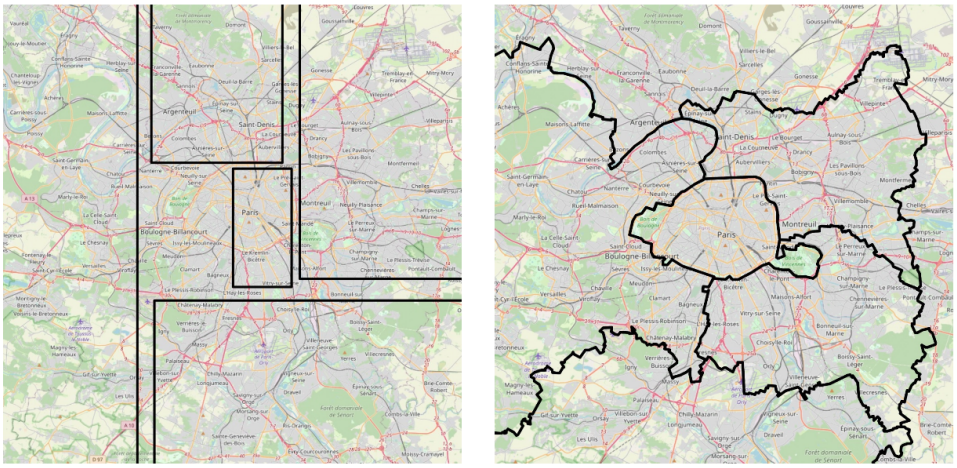
CLIP responds to *natural language* queries, making it different from typical object detection models that rely on predefined classes and annotated data. Pretrained on 400 million images with descriptors, CLIP provides an inverse approach to typical object detectors; the model can respond to natural languagef queries like "*is there a bird in this photo?*" and identify the relevant objects based on the description.

3.5 Transformer Backed Locators

Lukas Haas, Michal Skreta, Silas Alberti, and Chelsea Finn propose PIGEON (Predicting Image Geolocations)[17], a model that leverages CLIP [31] as a backbone for extracting visual features from images. PIGEON extends CLIP through *multi-task contrastive pre-training* and *hierarchical guess refinement* to enable the extraction of local GeoInformative Features.

PIGEON utilizes CLIP's pretrained vision transformer to generate image embeddings. These embeddings are used in conjunction with an OPTICS clustering model to create location cluster representations. During inference, an image embedding is computed and passed through a final linear layer to predict geocells and identify the top geocell candidates1.

Unlike PlaNet[49] which defines *geocell shapes* arbitrarily, PIGEON employes semantic geocell creation, wherein cells are balanced in size (images per cell) and flattens their distribution. PIGEON also employs haversine [39] smoothing in the loss function



(a) With naive, rectangular geocells.

(b) With our semantic geocells.

Figure 3: PLANET query cells using Semantic partitioning

Unlike PlaNet [49], PIGEON uses no convolutional layers (outside of CLIP) and instead relies on multi-modal transformers for feature extraction and processing. This approach allows PIGEON to utilize the rich contextual information captured from CLIP, enabling geolocalization without the need for convolutional layers traditionally used in models like PlaNet which capture their *own* GeoInformers.

Haas et al also employ the Haversine[39] measure for informing their loss function.

$$L_n = -\sum_{g_i \in G} \log(p_{n,i}) \cdot \exp\left(\frac{-\text{Hav}(g_i, x_n) - \text{Hav}(g_n, x_n)}{\tau}\right) \tag{1}$$

Equation 1: Haversine loss function used in PIGEON's geolocation model[17].

Where *HAV* is the *Haversine Distance Function* and is *Temperature*, a parameter which controls the smoothness of the weighting factor. The smoothing process ensures that the model does not overfit to individual geocells and provides a tunable hyperparameter that balances overfitting by adjusting how sharply the weighting factor drops off with distance.

Haas et al. consider geographic coordinates, climate patterns, and directional cues such as the orientation of shadows and sun direction to be useful geoinformers in training their model. Geographic coordinates, naturally, are unavailable in unseen data but are included to help model spatial relationships during training. These features, combined with semantic geocell creation, multi-task contrastive pretraining, and hierarchical guess refinement, enable PIGEON to create a highly accurate and generalized model.

PIGEON conveniently extends the large pre-trained CLIP model, however, due to the size of the model, resources for computing extensions or even the forward pass inference are significantly demanding; making it impractical for real-time capture.

Similarly, the model is dependent on street view imagery and while the model should *discount* features which are products of capturing from street view itself (i.e., the location of the vehicle taking the photo vs. where a pedestrian may have), the versatility of the model is uncertain. PIGEON is intended as a GeoGuessr bot, however, but extending the domain is uncertain

The clustering asymmetry also contributes to a potential loss of features which span clusters (i.e. a long row of hedges). Clustering based on image latent features as well as geolocation may improve this, however this combines a preprocessing step with the training step, meaing a fundamental rewrite of the PIGEON architecture.

3.6 Large Language Models for GeoLocalization

Large language models (LLMs) have fundamentally transformed the landscape of artificial intelligence, enabling, at a cursory level, completely natural human interaction with text-based inputs. These models, in 2025, approach being trained on *the entire internet* and have become powerful tools for *essentially* any domain where word association is central.

Building on this, multi-modal models such as OpenAI's CLIP[31] (Contrastive Language-Image Pre-training) are able to *bridge the gap* between visual data and textual data by learning joint representations of such. These models are backed by large language models and are trained to respond to plain language prompts such as "What is this?" by returning contextually relevant descriptions; the models are interacted with by providing image inputs and text queries, wherein the models to generate coherent and accurate responses that link visual content to textual descriptions.

StreetCLIP[16] leverages the multi-modal capability by extending CLIP's architecture to specifically address street-level imagery. StreetCLIP is trained on a combination of street view images and associated textual metadata, such as street names, business signs, and regional descriptors. This allows the model to learn fine relationships between visual features and local text, **which are critical for accurate geolocalization**. i.e StreetCLIP can recognize that a specific architectural style combined with a particular language on signage is indicative of a certain region or city. This approach significantly improves the model's ability to generalize across diverse and ambiguous locales, making it a powerful tool for geolocalization.

Similarly, GeoReasoner[51] builds on the foundation of LLMs by incorporating spatial reasoning capabilities. GeoReasoner uses a combination of visual and textual inputs to infer geographic relationships, including as the relative positions of landmarks or the directional orientation of street topography.Integrating these into the model enalbes GeoReasoner can to more informed predictions about the location of an image than streetCLIP's base. This is particularly useful in scenarios where the visual and textual cues are subtle or ambiguous, as the model can leverage its understanding of spatial relationships to narrow down the possible locations; essentially, applying an LLM and the reasoning therein to a geoinformative pattern which may otherwise be learnt implicitly by [49] planet or PiGeon[17].

Both StreetCLIP and GeoReasoner demonstrate the large of combining visual and textual data for geolocalization. However, they also highlight the challenges associated with scaling these models for real-time applications, as the computational demands of multi-modal transformers can be significant as at 2025. However they do show that a combinatorial approach to text and object embeddings can form a useful and performant GeoLocalizer.

3.7 Alternate Methods for Defining GeoInformative Features

Haas et al showed that automatically derviing geoinformers is now trivial with convolutional neural networks through PlaNet[49], or that explicitly defining them in terms of object Detection, i.e. PiGeon[17] other approaches can provide useful context. Convolutional layers are computationally expensive and LLM based models are memory intensive, Doersch et al [10] propose that geo-informers can be algorithmically determined for binary classifiers. Et al seeds 25'000 randomized high contrast patches from some 10m images from Parisian streets; their nearest neighbours, that is, most similar in terms of colour descriptor, are considered in weighting each feature, resulting in around 1000 Geo-Informative image sections for Paris. Of note; the resultant model and approach is binary; Is the unseen image Parisian

or Not? Doersch et al adopts a linear SVM for determining similarity; overcoming unwanted similarity metrics in swatchces, such as camera obfuscation or capture artifacts per Shrivastava et al [41] in addition to CV over k fold in both Parisian and negative datasets for 3 iterations to achieve convergence; thusly defining their detectors.

This approach sufficiently defines geo-informative features for a binary moddel where the image domain has sufficient architectural features. We note that Doersch et al also seeded some number of swatches and the overall paper quantifies results in terms of an informal study. Applying this approach to any given locale is likely to perform with poorer accuracy than Paris. However, this defines a valid metric for determing geolocators through a random seed. The approach was applied to northern american homes and Doersch et al note that there is insufficient Geo Diversity in Architectural features to inform a useful binary Model.

3.8 Explicit GeoInformers

3.8.1 Object Detection With Convolutional Neural Networks

Object detection is a fundamental computer vision application that underpins consequential domains such as geolocalization. Object detection comprises two sub-tasks, *Object Localization within images* and *Object Classification* Advances in Deep Learning have allowed the field to progress past traditional approaches such as Haar Cascades or Histogram of Oriented Gradients Methods, which were, for the time effective solutions which required manual feature engineering and were therefore poorly scalaale. For the purpose of our GeoLocator, Object Classification is sufficent.

Convolutional Neural Networks (CNNs) for object detection moved the field forward significantly. CNNs automatically learn hierarchical features from data, eliminating the need for manual feature engineering, i.e. per doersch et al[10] and enabling greater accuracy and scalability.

The YOLO family of deep learning models are well-regarded for their efficiency and effectiveness in real-time object detection tasks, especially in still images. YOLOv1 introduced single-pass detection, a significant shift from previous multi-pass methods, yet it was limited by accuracy and speed constraints [34]. YOLOv2 introduced anchor boxes and batch normalization, addressing localization errors (i.e.errors predicting locations of bounding boxes due to local variation) and improving detection performance by normalizing activations and reducing internal covariate shift [32]. YOLOv3 further refined the architecture, incorporating a deeper feature extraction network (Darknet-53) for better accuracy and multi-scale predictions, which enhanced performance on small objects [33]. YOLOv4 added new components such as *mosaic data augmentation*, which improved the robustness of the model by creating more extensive training data from multiple images, and cross mini-batch normalization, which stabilized training by normalizing across mini-batches [3].

The subsequent YOLO iterations introduced new features. YOLOv5 leveraged CSPDarknet to optimize computational efficiency and gradient flow, making the model more resource-efficient while maintaining high accuracy [23]. YOLOv6 and YOLOv7 continued to push the boundaries with additional architectural improvements and optimization techniques, further enhancing both speed and accuracy. YOLOv8 and YOLOv9 introduced Programmable Gradient Information (PGI) and other refinements to enhance learning dynamics and performance on diverse sets.

Model	**mAP (mean Average Precision)**	**FPS (frames per second)**
YOLOv1	33.4%	45
YOLOv2	48.1%	30
YOLOv3	57.9%	25
YOLOv9	72.5%	20
YOLOv11	73.8%	15

Table 1: Performance of YOLO models on the COCO dataset [26, 34, 32, 33, 3, 23, 2]

As shown in Table 1, the performance of different YOLO models on the COCO dataset varies significantly [34, 32, 33, 3, 23, 2]. YOLOv1 achieved a mAP of 33.4%, while YOLOv11 reached 73.8% on the common baseline COCO dataset [26]

It introduces significant architectural improvements, such as the C3k2 block, SPPF, and C2PSA components, which enhance feature extraction and overall performance. YOLOv11 achieves a higher mean Average Precision (mAP) on the COCO dataset while using 22 percent fewer parameters than YOLOv8m,

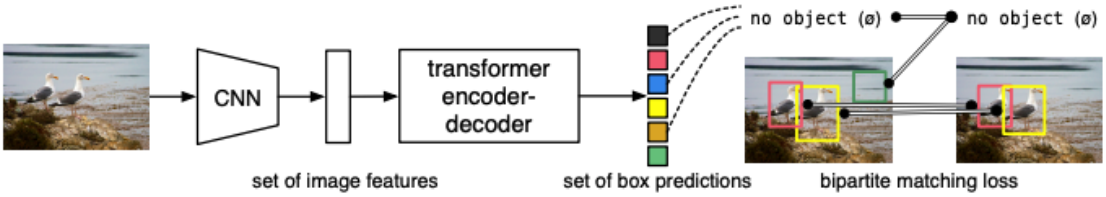


Figure 4: Deter Detection Puipeline

making it computationally efficient without compromising accuracy [2]. YoloV11 is available as an *off-the-shelf* model which is trivially extendable to beyond the 80 classes which the existing model can recognize (the product of training on the COCO dataset[27]) making it a useful *drop in* GeoInformative model

3.9 Object Detection with Transformer Based Models

Mention was made of CLIP[31] in

Transformer-based models, such as DETR (DEtection TRansformer)[5], are increasingly contending with traditional CNN[25]-based approaches like those employed by YOLO[34] detection models in object detection. Unlike CNNs, which rely on convolutional layers to extract local features, transformers use self-attention mechanisms to capture global context and relationships between all pixels in an image. This global perspective allows transformer-based models to simplify the detection pipeline by eliminating the need for post-processing steps like Non-Maximum Suppression (NMS)

Additionally, NVIDIA[4], in early 2025 have shifted their industry leading Frame Interpolation and Real Time Direct X upscaler to Transformer models from CNN based models for hardware which is consumer grade. Optimizations; software and hardware in this space as well as forward advancement have meant that the viability of transformers in edge or client devices in place of CNN's is becoming much geater. While not a direct analogue to Object Classification and detection; it does show the viability of this approach going forward.

3.10 Optical Character Recognition for GeoInformers

According to Pro GeoGuesser players, *script* or *text* represents some of the most informative features for pro *GeoGeussr* players [12] [48] behind factors such as availability of streetview images and near-binary operators like driving direction.

It therefore holds that *OCR* represents an opportunity for an effective GeoInformer, Effective such that Google implemented a method for using deep learning and OCR to to extract difficult street names and business names from Street View images with high accuracy to maintain the accuracy of their maps service. the approach achieves 84.2 percent accuracy on the challenging French Street Name Signs (FSNS) dataset. Much of the challenge faced to combat here were artifacts of streetview imagery, such as blur, shutter distortion, and other distortions.

3.11 Off-the-shelf OCR Models

3.11.1 easyOCR

EasyOCR[22] is a versatile off-the-shelf OCR model designed to recognize text in images across diverse environments, such as outdoor signage. It integrates multiple machine learning techniques to achieve high accuracy and adaptability across various languages, scripts, and challenging conditions. The model's architecture is built on several advanced deep learning approaches, enabling robust text recognition capabilities.

At its core, EasyOCR utilizes **Convolutional Neural Networks (CNNs)**[25] for feature extraction, identifying patterns and structures within text regions. To handle the sequential nature of text, it employs **Recurrent Neural Networks (RNNs)**, particularly Long Short-Term Memory (LSTM)[20] networks, which ensure the model captures the context and order of characters effectively. Additionally, **attention mechanisms** are incorporated to allow the model to focus on relevant parts of the image, enhancing its ability to recognize text in complex or cluttered scenes.

EasyOCR supports over 80 languages and a wide range of character sets, including Latin, Chinese, and Japanese scripts. Its pretraining on a large dataset of image-text pairs enables strong out-of-the-box performance, minimizing the need for extensive fine-tuning. This makes it exceptionally accessible and easy to integrate into existing work. easyOCR

The model's development is grounded in foundational research in OCR and deep learning. Key techniques include **Connectionist Temporal Classification (CTC)** for sequence modeling without pre-segmented data [**graves2006connectionist**], and **Scene Text Detection** methods for recognizing text in natural environments. A notable component is **CRAFT (Character Region Awareness for Text Detection)** [**CRAFT**], which enhances text localization by generating precise bounding boxes around detected text regions. This ensures that the OCR engine receives well-defined inputs, significantly improving recognition accuracy.

3.11.2 Other OCR off-the-shelf mdoels

In comparison to *EasyOCR*[22], Amazon Textract, Google's Cloud Vision, and PaddleOCR each offer unique advantages. Amazon Textract excels in extracting structured data from documents, including tables and forms, with a pay-as-you-go pricing model and cloud-based s insing that minimizes local resource usage in favor of network dependence[40]. Google's Cloud Vision provides high accuracy and extensive language support, also operating on a pay-per-use basis with cloud infrastructure[7]. PaddleOCR, like EasyOCR, is open-source and similarly easy to run locally[30]. PaddleOCR is built on the PaddlePaddle deep learning framework[30], whereas EasyOCR is built on the more ubiquitous *PyTorch*[22]. For the purpose of extracting text from street-level imagery and signs, the local models are more than sufficient.

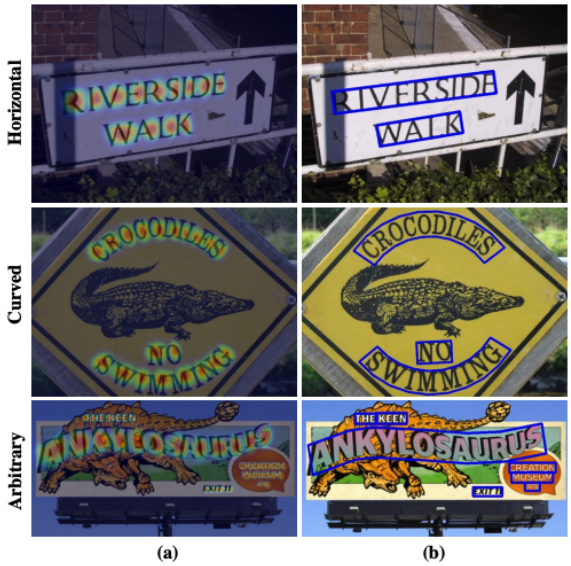


Figure 5: Illustration of CRAFT-based text detection in complex scenes.

3.12 Text Embeddings

Raw text is a useful GeoInformer, but the relationships between words often provide more valuable information than individual words themselves. For example, the word "branch" can have multiple meanings depending on context. Raw text and Traditional word embeddings can struggle to capture such complex relationships, as they often overlook the nuances of how words interact within sentences.

Text Embedding Models address this problem by converting entire sentences into fixed-size embeddings, allowing for better semantic understanding. These embeddings represent the full meaning of a sentence, making it easier to compare and group sentences based on their underlying context rather than their individual words. For building out a **GeoInformative text embeddings** from street signage, these mebeddings can accurately capture and interpret the context of location-based information, ensuring that geographic references are extracted and categorized correctly based on semantic meaning rather than surface-level keywords and reduce the impact of noise in the text.

BERT[9] (Bidirectional Encoder Representations from Transformers) is a pre-trained model that captures context in both directions (left-to-right and right-to-left). While powerful, BERT typically produces word-level embeddings, which are not ideal for tasks that involve comparing entire sentences. To solve this, Sentence-BERT was developed to fine-tune BERT into a model capable of generating high-quality sentence embeddings.

SBERT[35] uses a Siamese network architecture, where two identical BERT models are trained in parallel to generate sentence embeddings. The model then learns to place semantically similar sentences closer together in the embedding space. This makes SBERT highly effective for tasks like semantic textual similarity, clustering, and information retrieval, which require comparing sentence meanings directly.

In the context of geolocalization networks, SBERT can enhance text extraction by accurately capturing the relationships between geographical terms in sentences.

SBERT is available as an *off-the-shelf* encoder model with a small memory footprint, making it practicable and ideal for encoding text.

3.13 Combined Embeddings as a GeoInformative Pattern and Attention

In 2017, Vaswani et al [47] introduced Attention Layers, which can be used in Neural Netowrks to dynamically weigh the improtance of different input features, enabling more sophisticated but also more contextually aware models. Attention layers can benefit deep learning models by reducing convergence time (in some cases) by having the network focus on the most prominent features (reducing 'search' time for global optimums.

An attention layer or multi head attention layer can capture relationships between text and object embeddings, for example.

It is therefore intuitive that an attention layer which highlights the most important geoinformers is a reasonable addition to a Geolocalizer.

3.14 State of GeoLocalization in 2024

The current state of geolocalization models includes diverse approaches with unique strengths and limitations. CNN-based models like PlaNet[49] excel in implicit feature learning but suffer from significant dataset bias and interpretability issues. Transformer-based models such as CLIP and PIGEON integrate visual and textual data, improving accuracy but are computationally intensive. Large Language Model-based models like StreetCLIP and GeoReasoner enhance geolocalization by incorporating spatial reasoning, yet their high computational requirements make them unsuitable for real-time processing.

The performance of LLMs on more affordable hardware has improved significantly with the introduction of models like DeepSeek, which offer high performance and cost efficiency through architectural innovations like Mixture-of-Experts (MoE)[8]. Additionally, the development of smaller, more efficient LLMs has shown that significant advancements can be made without the need for massive computational resources[8]. This rapid evolution suggests that the core dependencies of geolocalization models may develop very quickly, potentially leading to more accessible and scalable solutions.

4 Broad Proposal Overview

We propose a machine learning-based geolocation method that leverages *latent image features* to refine location predictions. Specifically, we extract and analyze:

- **Objects** detected within the scene,

- **Text** present in the scene,
- **A color embedding** representing the image's overall color distribution.

This approach contrasts with prior methods [49], [10], [17], which learn these features implicitly. Instead, our model *explicitly* focuses on **neighborhood-scale geolocation**, avoiding broader regional classification beyond the candidate cell.

Figure 6: Examples of Geo-Informative Street-Level Images. (1) Chicago street with explicit geo-informative text. (2) Chicago intersection with strong contextual clues but no text. (3) A video game scene where objects are identical assets. (4) Wellington waterfront with ambiguous signage.

Objects in an image serve as **geo-informers** if they meaningfully constrain the set of possible locations where the image was taken. **Geo-informative patterns**, such as recurring urban layouts or architectural styles, are also geo-informers in their own right, we consider exclusively those defined above.

Individual objects—such as traffic lights, crosswalks, or streetlamps—may not be geo-informative in isolation, but their **combinations** can contribute to a **geo-informative embedding** that can be combined iwth other features to meaningfully assist localization. A *GeoInformer* is an aggregate of GeoInformers in it's own right; herien this is a *GeoInformative Pattern.*

Among GeoInformers, **signage with text** is particularly valuable, often offering direct geospatial clues such as names of locations, businesses or phone numebers which are regional. Text extraction and recognition play a crucial role in our method, providing a stronger geo-informative signal than more general features like color distributions or indeed the objects embedding vector.

This paper proposes that **object extraction** from street-level imagery forms a valuable **geo-informative pattern**, refining geolocation predictions. In particular, **text-based signage** is an especially strong geo-informative cue and is weighted accordingly in our training process and architecture.

Our geolocation model employs a **three-pronged approach** to feature extraction and learning, detailed in **Section 7**.

5 Object Detection and Constructing An Embedded Representation

We consider **Object Detection** to be a step in image **Preprocessing**. However, it is important to note that the relative performance of these models plays a significant role in the overall performance of the geolocalizer.

This section outlines in isolation, conducting object detection and transfer learning for the purpose of supporting the overall GeoLocalization architecture.

5.1 Explicit Selection of Objects as GeoInformers

Our approach *explicitly* considers combinations of the follliwng a valid GeoInformatve Pattern. Detecting and embedding these is a core part of the overall architecture.

- Clock
- Fire Hydrant
- Bench
- Vase
- Stop Sign
- Class Label
- Traffic Light
- Parking Meter
- Signage
- Crosswalks
- Streetlamps

Figure 7: Image segments from a sample image.

The ideal extraction of objects from Figure 7 includes:

5.1.1 Object Detection Models

To extract objects effectively, we fine-tune the pre-trained YOLOv11 model with a ResNet100 backbone from *Ultralytics* [46] using *Transfer Learning*.

Ultralytics [45] provides a pre-trained YOLOv11 model with a ResNet100 backbone [19]. This model serves as a robust starting point for transfer learning, enabling us to create models for the detection of object classes not originally included in ResNet100, as well as geo-informative features that are already part of its architecture.

Off-the-shelf, the YOLOv11 model achieves a higher mean Average Precision (mAP) on the COCO[27] validation set compared to YOLOv8m, while using 22% fewer parameters. This makes YOLOv11 computationally efficient without sacrificing accuracy, as demonstrated by its performance on benchmark datasets.

5.1.2 Signage as a Separate GeoLocator

To genearte the signage detection model, we conduct this transfer learnign; We use public ally available Datasets [36][37][38] for training the Signage detection model. For all images, the same class is applied; creating an aggregate dataset per 5

Dataset - Signages (combined)	**Total Images**	**Train**	**Test**	**Validation**
Signages[38]	778	622	78	78
Board Detection[36]	640	512	64	64
Billboard Detection[27]	3399	2719	340	340
Total	**4817**	**3853**	**482**	**482**

Table 2: Dataset Splits

5.1.3 Objects as GeoLocators

In the same motion extracting the *signage* objectss within a model; YOLO Model(s) are generated to detect *GeoInfromers* within the image.

The same pre-trained YOLOV11 model[46] with a resnet100[18] backbone is implemented. Usefully and rather conveniently, a subset of our chosen Geoinformers are recognized by the base model. For classes who are not a member of this set, transfer learning is employed using public datasets:

Discrete YOLO models are used in lieu of a single combined model. This does introduce memory overhead, yet mitigates issues with class crossover lacking annotations and ultimately reduces development load.

Classes Filtered From Base Model
Clock
Vase
Traffic Light
Fire Hydrant
Stop Sign
Parking Meter
Bench

Table 3: Classes filtered by Base Model

Class Label
Signage
Crosswalks
Streetlamps

Table 4: Classes with Discrete, Fine-Tuned Models

5.1.4 Transfer Learning for All Classes

We employ transfer learning to generate new detection models for GeoInformers by fine-tuning a base model, specifically the *YOLOv11 with ResNet100 backbone* from Ultralytics[46]. By leveraging the pre-trained model's *knowledge* of low-level features such as edges, textures, and shapes, we adapt the final layers to detect new object classes. This robust foundation allows us to efficiently detect new classes with minimal retraining, importantly, we are able to achieve sufficient detection accuracy and precision with small, publically avalable datasets.

- **Base Model:** The YOLOv11 model with a ResNet100 backbone, pretrained on the COCO dataset, which predicts 80 classes and is filtered according to the classes shown in 3.
- **Custom Datasets:** New object classes, such as **signs**, are sourced from various datasets and grouped into a single class. All datasets used are RoboFlow datasets with appropriate licensing .
- **Discrete Models:** A separate YOLO model is trained for each new object class, minimizing class crossover and simplifying the development process.

For all YOLO models, transfer learning is conducted using a standard data split of 80% for training, 10% for testing, and 10% for validation. This ensures that class imbalance does not significantly impact training, as datasets are focused on learning a super-class (e.g., signage) **50 Epochs is applied across the board**

Dataset	Total Images	Train	Test	Validation
Crosswalks[11]	778	622	78	78
Hydrants[36]	640	512	64	64
Streetlamps[24]	3399	2719	340	340

Table 5: Dataset Splits

We employ the default learning rate of 0.001, which balances the speed and stability of the training process. This value is generally effective for a wide range of tasks and is recommended by Ultralytics to achieve the best results without the need for manual adjustments [46]. Similarly, the Adam optimizer is used for its efficiency and adaptability, as it dynamically adjusts the learning rate as required.

Comprehensive Results for the YOLO model are available in the */results* directory in the repo.

The performance of the StreetLamps detection model is poor, however the shown signage and crosswalks is reasonable (well balanced).

Evaluation Metric	**Crosswalk Detection**	**Streetlight Detection**	**Signage Detection**
Training Time (seconds)	390.05	311.587	354.042
Training Box Loss	0.2635	1.00986	0.76153
Training Classification Loss	0.19382	1.06163	0.77274
Training Distribution Focal Loss	0.96084	1.32338	0.98338
mAP@ 50% IoU (mAP50)	0.98589	0.90639	0.98629
mAP(mAP50-95)	0.8411	0.58943	0.71343
Validation Box Loss	0.76425	1.1804	0.91991
Validation Classification Loss	0.581	0.88113	0.42802
Validation Distribution Focal Loss	1.40819	1.45533	1.02577

Table 6: Comparison of Metrics across Crosswalk, Streetlight, and Signage Detection

The results per Table 6 demonstrate *sufficiently negligible* performance for most of the object detection models, particularly for the Crosswalks and Signage classes, which show high mean Average Precision (mAP50) values close to 0.986. This indicates that the models are highly accurate. However, the Streetlight model, with a lower mAP50 of 0.906, suggests that the detection of certain objects may be *insufficiently negligible* in the context of considering the Object Detection purely preprocessing.

5.1.5 Discrete Models are Not Ideal

While using discrete models is not the most efficient way in terms of memory usage (or, more particularly, in terms of swapping models and loss of parallelization), it saves significant time and effort in development. This deliberate decision allows us to much more easily adapt and fine-tune models for specific tasks without the need for extensive retraining of a single combined model. Additionally, it helps in managing and mitigating issues related to class crossover, where annotations for different classes might interfere with each other if there is unintended overlap within our single-class datasets. And conveniently, we almost fully avoid the risk of *catastrophic forgetting*[2] regarding the original 80 classes.

In our implementation

```
[Image] → [YOLOv11 Base Model], [Crosswalk Model], [Hydrant Model], [Signage Model]
```

Models must be shifted in and out of memory or use memory space otherwise available for data.

In the ideal case

```
[Image] → [Conglomerate Model]
```

Figure 8: Signage Validation Check

Figure 10: Signage Validation Check

Figure 12: Crosswalk Validation

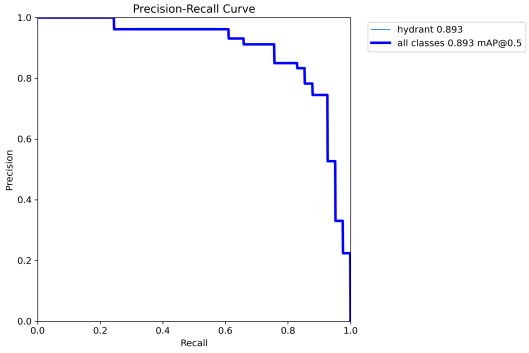


Figure 9: StreetLight Val Predictions (note Incorrect Label String Saved only)

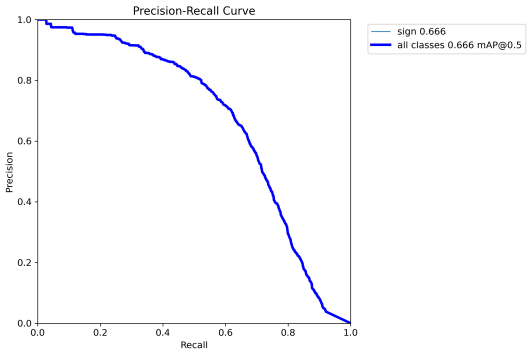


Figure 11: Aggregate sign PR curve

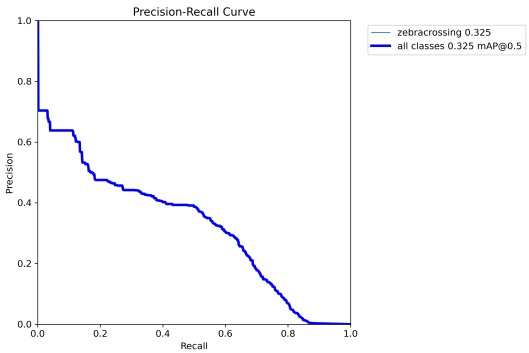


Figure 13: Crosswalk PR Curve

A single model can produce an amalgamation of all classes at no additional size nor computational overhead (same weights counts, same number layers etc). However time constraints play strongly into our decision to use discrete models.

We simply filter detections from the unchanged model YoloV11 with resnet 100 backbone model which are not GeoInformers, there is no performance benefit to retraining to remove these.

5.1.6 Transfer Learning Hyperparameters and Precision

For training all YOLO models we employ the **ADAM** Optimizer and the

YOLO models are commonly used for Image Segmentation and Instance Segmentation; to align with the datasets, we simply return bounding boxes from the YOLO model.

5.1.7 Image Augmentation

The training sets for our YOLO model include image augmentations such as random scale, random rotation, random salt and pepper noise, and random blurring. These augmentations are applied to the dataset with random frequencies to increase the diversity of training data, reduce overfitting, and improve the model's generalization ability on unseen data. By introducing variability in image appearance, the model becomes more robust to real-world variations like object occlusion, scale changes, and noise. The total number of images after augmentation is recorded in 5.

5.1.8 Wrapping YOLO Models

Of the returned values from each pass of the *off-the-shelf* YOLO and the *fine tuned* models, the Bounding Boxes, Confidence scores and class labels are encoded. The ultimate tensors take the form of the table in9.

5.1.9 Image Statistics

The dataset comprises a total of 150,510 images. The detection model's performance is summarized below. Around a third of the locations have some form of GeoInformer detection from the YOLO model aggregate.

MOVE

Total Images:	150,510
Images with Detections:	111,125
Total Number of Detections of Signs:	308,618
Detection Rate:	73.83% of images contain detectable signage

Table 7: Detection Summary

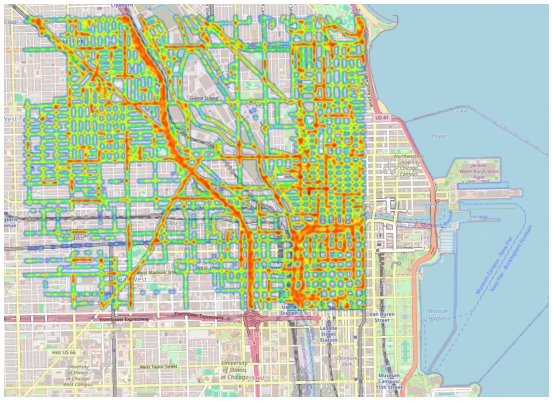


Figure 14: Frequency of detections in Chicago for Signage

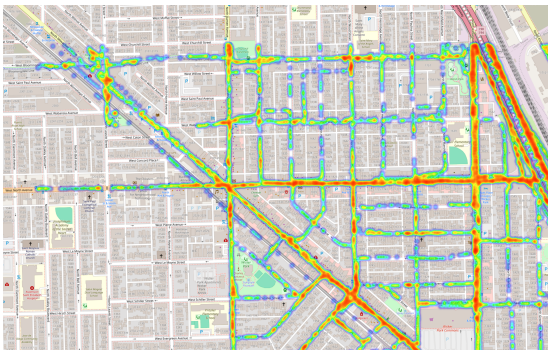


Figure 15: Zoom of Chicago Signage detection frequency; Missing areas etc

Figure 16: Signs Extracted by The YOLO models to form the sign Dataset

5.1.10 Performance of The YOLO Models

Notably; the pool of street level images that are used to train the Machcine Learning and Clustering Models later are not considered in the training of the YOLO models.

The performance of YOLO models for detecting our GeoInformers is evaluated using classification and localization metrics; YOLO models are a combination of classification and regression metrics; as Object detection is considered preprocessing, the depth of results and analysis is particularly shallow and non contrastive.

5.2 Alternate Approaches and Improvements

An insufficiently performant model here essentially adds noise to the downstream models; whilst relative performance on the validation, test (and train) sets is excellent, no member of the *Chicago* dataset is included in the test / train or valid set.The generalizability of the model to unseen data, particularly when the model faces street level datasetes wherein the distribution of features has not been seen in the training set (primarily, the size and scope of the image, most of the images used in transfer learning are swatched) is questioned.

An alternative approach to improve the robustness and generalizability of the model is to leverage CLIP (Contrastive Language–Image Pretraining) for object detection over YOLO models, which rely heavily on bounding boxes for object detection and are thusly dependany heavily on the datasets and the quality of annotation. CLIP is a multimodal model that learns to associate images with natural language descriptions; which allows zero shot inference. However the memory constraints are currently limiting for transformer based models of this kind, yet an object detection pass about a candidate image could forgo the entire preprocesssing / ingest phase per section **??** CLIP will not generate a bounding box, however an approach which simply counts occurances may be sufficient; especially with the derivable context per [16]

6 Data Procurement

Ubiquitous availability of street-level imagerythroughwith platforms like Google Maps [14] and online communities such as Reddit's `r/averagepicturesofcity name` provide extensive repositories of geo-tagged images. While these sources offer broad accessibility, they come with notable limitations.

6.1 Data Source: Google Maps and Social Media

For instance, Google Maps and other Image Hosts imposes API costs and suffers from snapping issues, where images are aligned to predefined points rather than exact locations, as well as indiscriminatly including user-uploaded images of the internals of malls or offices alongside street level imagery. The Google

Maps Service impose a credit to *so 640*640 * 25000 images / m equivalent* which queries additional are charged.

6.2 Data Source: Mapillary

Mapillary as our primary data procurement platform. Mapillary operates under a permissive license, allowing for free access and use of its imagery for research purposes, which significantly reduces costs compared to proprietary services like Google Maps. However, it is important to note that Mapillary's reliance on user-uploaded content introduces some variability in data quality,density and type. While this collaborative model enables global coverage and less distinct time variability than other platforms, it can result in less dense or **unevenly distributed imagery** in certain areas, as well as occasional inconsistencies in reliability. The API is also somewhat rate limited and a simple distributed application is used to download and consolidate images; writing against a truth source database.

6.3 Selection of City

Chicago was selected as our candidate environment due to its reasonable housing density, approximately 1,700 persons per square kilometer [6] and relative density per Tile in terms of available images through the Mapillary API. This density is comparable to cities like Christchurch, New Zealand (1,351 persons/km^2) and Sydney, Australia (2,038 persons/km^2) [50]. Middle and high-density urban areas, such as those found in Chicago, are particularly rich in the latent features identified by our discrete models. Additionally, the implicit infrastructure and diversity of Chicago's neighborhoods make it an ideal setting for our analysis. Conveniently, Chicago is around 92% covered by the Mapillary Dataset [28].

At approximately 20km$\hat{2}$, the candidate are difference in terms of latitude and longitude is particularly precise: at just 0.004- and 0.060- respectively.

6.4 Generalizability to Cities

In contrast; Detroit, MI is the city mose extensively covered by the Mapillary dataset [21]; yet due to various socioeconomic reasons, random samples of street level imagery tend to not show any GeoInformers more often than not.

Figure 17: Detroit Street Level Images[28] from Mapillary are Particularly Sparse in Our Selection of GeoInformers

The proposed model leverages Object Detections, Text Embeddings and Colour Profiles, so the selection of dataset must mirror this with some useful distribution. **The model is inherently not intended to be generalizable to sparse locales in this way.**

6.5 Sample Region and Street Level Imagery

For the purpose of this paper, we consider the following area occupying 20.5 km^2 which produces 151,510 images occupying 42.1 GB of disk space as loose .jpg files, with an additional 6.1MB as the state control database for managing preprocessing, ingest, and download across clients. An average density of 7,391 images per km$^2 is found, although non-indicative per city topography.$

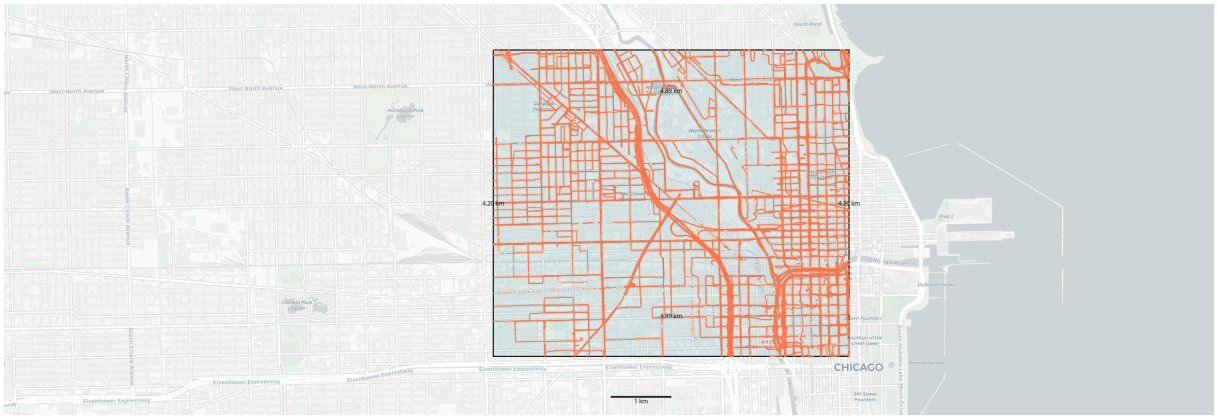


Figure 18: Candidate Area Within Chicago

Figure 19: Samples of street level imagery from mappilary

Each red point in **??** represents an individual street level image. Street level images procured via Mapillary are typically captured from a vehicle and often include dashcam stills.

7 Method and Implementation

This paper proposes a method for GeoLocalization of street level imagery that leverages multiple off-the-shelf discrete neural networks to generate **intermediary embeddings**. These embeddings serve as input to a **three** Machine Leraning Models for **GeoLocalization**.

Figure 20: Street Level Imagery from the Dataset Cicago

Three distinct machine learning models are trained on i these intermediary datasets; The models' predictive outputs—latitude and longitude—are averaged to explicitly determine the geographical coordinates at which a street-level image was captured.

The model is scoped to a multi-neighborhood scale. In the context of a planet-scale GeoLocalization system, this would represent the final step in the localization process. A broader model would first identify a specific "geocell" and then apply this network, either as additional layers within a larger GeoLocalization framework or as a hot-swapped module. Consequently, all predictions are expected to fall within the bounds of the candidate area defined in Section **??** and the model is not intended to handle points outside of this area.

For the purpose of this GeoLocalization application, the models responsible for generating intermediary embeddings are treated as **preprocessing steps only**. The primary focus is on learning GeoLocalization directly from these embeddings, under the assumption that they are sufficiently descriptive to capture the necessary spatial and contextual information.

By leveraging attention mechanisms, the model will dynamically prioritize the most *GeoInformative* features in the source dataset. We define variatons of the models which include Attention and which do not and conduct a brief ablation study.

7.1 Scope

The model only considers the chicago candidate area and is designed to learn from only local image features using a broad range of street level imagery.

The method encompasses two core *Toolboxes*:

- **Generating Embeddings from a Dataset:**
  - Pre-Built Text Embedding
  - Object Detection
  - OCR Models
  - Colour Space Analysis
- **Learning Features from Embeddings:**
  - Convolutional Neural Networks
  - Attention Mechanisms
  - Neural Networks
  - Gradient Management
  - Learnt Embeddings
  - Clustering Analysis

The model is scoped to **Multi-Neighbourhood** scale, yet we will show how increasing the candidate area is plausible and also how the model is insufficient for larger scopes.

7.2 Model Architecture (Overview)

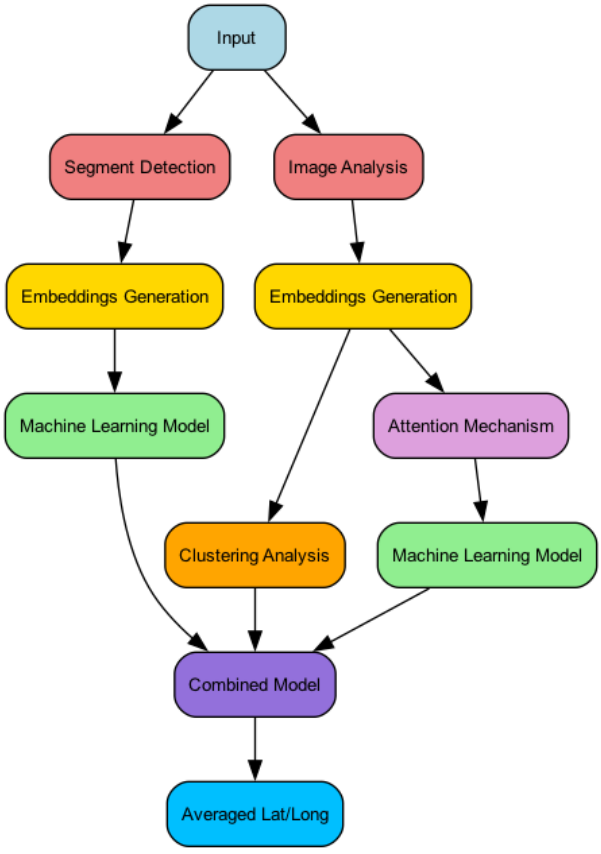


Figure 21: Diagram of the Overall learning process.

7.3 Results Interpretation and Context

The model conducts a regression, unlike [13] [31] which define GeoCells. Broadly; this model would *fit into* a network as a last hop within a geocell. The relative performance metrics are therefore rather intuitive; Loss is simply defined as the Euclidean distance (through MSE). All datapoints contain a Latitude and Longitde value and the entire space occurs 3 dp of those values. The regression is normalized *in and out* of the candidate area.

We expect that averaging of the 3 ML approaches improves accuracy overall; accuracy simply being the euclidean distance to the true observationb and we similarly expect that Attention can better overcome noise to refine precision for Text and Objects within an iamge.

7.4 Overview

We define an overall architecture with three core steps across the board.

- Extracting Local Image features to an embddded Datasets (preliminary Embedding)
- Training Neural Network Mdoels on these datasets to learn patterns
- Analysing performance on unseen data

Our approach infers GeoData by analyzing local image features and their spatial patterns. Specifically, we extract and combine information such as text, objects, and color spaces from street-level imagery to identify GeoInformative patterns. Intuitively, in some cases, features like text embeddings from OCR models can directly pinpoint a location (e.g., a uniquely named business), however a broader pattern of lesser GeoInformative features is usedto inform our GeoLocalization Model.

7.5 GeoInformers and GeoInformative Patterns

7.6 Local Features as GeoImformers

Defining GeoInformers is a core step in our model definition. While some methods [49][**france**] learn these features implicitly, we find that manually selecting intuitive features is sufficient for effective GeoLocalization at the city scale. This approach balances simplicity and precision but does require some familarity with the domain city.

Our GeoInfromers are the following:

- Text within a scene
- Objects and their frequency within a scene
- Object Overalp within a scene
- The Colour Profile (represented as an embedded Colour Histogram) within a scene.

7.7 Object Detections to Build an Informative Vector Embedding

Objects within a scene are also GeoInformers, and finding the relative importance of GeoInformers is a task that can be automated. A trivial approach may be simply detecting *as much as possible* and performing dimensionality reduction on a discrete encoding. For the purpose of defining the **Object Embeddings Vector** per image, we consider the results of the models trained in section **??????**.

The following class names are given as aggregates from **4 discrete YOLO Models**:

- Clock
- Vase
- Traffic Light
- Fire Hydrant
- Parking Meter
- Bench
- Signage
- Crosswalks
- Streetlamps

The embeddings for these form a simple frequency vector whose index corresponds to the class and value corresponds to the count of detections of that feature within an image.

```
3x traffic light, 2x hydrant, 1 x sign  $\rightarrow$ [0,0,3,2,0,1,0,0]
```

The YOLO models are abstracted herein. It is important to acknowledge the performance impacts downstream from the accuracy and precision of the YOLO models, including the accuracy of the bounding boxes and the **confidence score** parameter for detection employed. Herein, this paper assumes that the YOLO models are simply a source of truth. This is rather intuitively not the case. We *abstract away* complications from model modality for detections and precision.

7.8 Signage as a Unique GeoLocator

Signage within images serves as a particularly distinctive geo-informative feature, as it often includes textual information alongside a unique color profile.

However, similar to OCR embeddings extracted from an entire image, signage can sometimes introduce noise into the dataset. Certain types of signage, such as temporary signs or misleading advertisements, may lead to inaccurate geolocation predictions.

Figure 22: Examples of signage that may introduce noise: a time-invariant Samsung advertisement[42], a temporary sign, and misleading text[44].

It is expected that the model will learn to some extent from this noise. For example, the temporary traffic management sign in Figure 22 would still be detected and incorporated into the model despite its transient nature. Similarly, text that may typically serve as a strong geo-informer—such as product-specific advertisements for products on a fixed cycle could lead to incorrect associations. A notable example of this phenomenon is PlaNet[49], which erroneously linked classic cars almost exclusively to *Cuba*.

Advertising that is transient, yet prevalent at the time of capture may also erraneously indicate a location based on, implicitly, the time of capture alone.

Of note, data procured by services such as *Mapillary*[28] or *Google Street View*[14] do state the time of capture, however the nature of *Google Street View* image capture means that streets within a locale are updated in chunks in the sapce of a few days or weeks so advertising such as the *Samsung Advert* from 22 could introduce erroneous noise

Broadly, Each image in the core dataset has some number of signage detections and their aggregate is the finalized value.

7.9 Text Embeddings as GeoLocators and OCR

The EasyOCR model [22]is employed in this branch as a pre-trained, drop-in solution without any fine-tuning or additional training.

EasyOCR leverages CRNNs and CTC with a small memory footprint. For ingestion, it must be loaded into memory alongside the YOLO models, and its small footprint is sufficient.

Input images are normalized before being processed by the model. EasyOCR is chosen for its robustness in detecting and localizing text across diverse image conditions. From an implementation standpoint, the model is straightforward to use on batched images.

Each input image is processed through EasyOCR, which extracts localized text. The extracted text is tokenized, with punctuation removed, and each word is embedded into a high-dimensional space using the `all-MiniLM-L6-v2` model [35]. A tensor of shape $\texttt{BATCH\_SIZE} \times 12 \times 128$ is produced, where an upper limit of 12 words is applied. This embedding model is selected for its efficiency and ability to capture semantic relationships between words, making it well-suited for generating intermediary embeddings that feed into downstream tasks.

Importantly, no training is performed on this branch of the network; both EasyOCR and the embedding model are used as-is and are considered ancillary to the core architecture, serving purely as preprocessing steps. Their relative performance has a similar overall impact on the efficacy of the implementation as the discrete YOLO models **??**.

Additionally, a small *corrections* library is generated using Generative AI to correct common OCR misinterpretations. Some examples include:

```
STOP: S!OP, STOP, STQP, 5TOP, ...
```

Attention[47] is applied to the text embeddings in combination with the learned embeddings and/or one-hot object embeddings. Applying attention to text is a prudent decision, as it can help mitigate the influence of *noise* words (e.g., "for", "the") and better capture relationships between words in the embeddings.

7.10 Machine Learning for GeoLocation

Given resource constraints and the availability of a strong dataset with 151,510images, combining simpler machine learning approaches can be more efficient than fine-tuning complex models. For instance, training a k-means model requires significantly fewer resources compared to training a deep neural network with multiple dense or convolutional layers. However, K-means is potentially less precise. As such, this method proposes aggregating approaches through **ensemble learning** as a valid alternative to extensive hyperparameter tuning.

- **Attention**[47] has been transformative in the space of Machine Learning in the last decade; it follows that some *GeoInfromers* are more *descriptive* than other *GeoInformers* and that *Attention* could naturally be a valuable processor for a Neural Network model for GeoLocalization.
- **Convolutional Neural Networks** (CNNs) on concatenated text and color histogram embeddings is effective because CNNs can extract complex patterns from both types of data simultaneously. Text embeddings capture semantic meaning, while color histograms represent visual information. This combined approach allows a model which understand and interprets data more comprehensively, leading to ultimately more preciuse predictions.
- **Learned Embedding**
- **Clustering** is also a natural fit for the domain; Unlike the two approaches afore, clustering methods such as K-means clustering are trained by recursively moving the centers of clusters based on similarity metrics. As above, text embeddings, Colour space analysis and object embeddings all contribute to this similarity (distanec) metric. However, such an approach makes feature selection particularly nonviable.

This paper proposes a method where all 3 methods are *Ensemble Learnt*, wherein their outputs are simply combined to deliver an average.

7.11 Loss for Neural Nets

At a broad scale, the *Haversine*[39] distance is more appropriate than the simple *Euclidean Distance* metric due to its ability to account for the Earth's curvature. However, for this implementation, we use the *Euclidean* distance to inform our loss function.

Abstracting to a flat plane is sufficient for our purposes. To provide some context, within the 20 km sqyare area defined in our implementation scope, the Earth's curvature would only contribute a delta of approximately 15 meters. This small falloff makes Euclidean distance a practical choice, balancing simplicity and computational efficiency. For larger candidate areas, it would be beneficial to consider the precision tradeoff with the complexity of using the Haversine distance.

The model generates Lat/longitudinal tuple pairs $(\hat{\lambda}, \hat{\phi})$, where $\hat{\lambda}$ and $\hat{\phi}$ are the predicted values respectively, and (λ, ϕ) are the true values.

Intuitively, geographic distance can be used to derive our loss function. The Haversine[39] formula calculates the great-circle distance between two points on the Earth's surface. The loss function is defined as follows:

$$\mathcal{L} = R \cdot 2 \cdot \text{atan2}(a, b)$$

where a,b are the haversine denominator and numerator respectively:

$$a = \sqrt{\sin^2\left(\frac{\Delta\phi}{2}\right) + \cos(\phi) \cdot \cos(\hat{\phi}) \cdot \sin^2\left(\frac{\Delta\lambda}{2}\right)}$$

$$b = \sqrt{1 - \left(\sin^2\left(\frac{\Delta\phi}{2}\right) + \cos(\phi) \cdot \cos(\hat{\phi}) \cdot \sin^2\left(\frac{\Delta\lambda}{2}\right)\right)}$$

where:

$$\Delta\phi = \hat{\phi} - \phi, \quad \Delta\lambda = \hat{\lambda} - \lambda$$

ϕ and λ are the given latitude and longitude, while $\hat{\phi}$ and $\hat{\lambda}$ are the predicted latitude and longitude. R is Earth radius, 6,371 km.

However, given the candidate area, *Euclidean Distance* is sufficient and implemented for all Dense Loss Metrics.

i.e.

$$L_{\text{total}} = L_{\text{Euclidean}} + L_{\text{BatchNorm}} + L_{\text{Attention}}$$

Where the Loss is the aggregate of Euclidean and Other Loss metrics. We concisder the performacnce in terms of the finalized error; the MSE in this case.

7.12 Normalization for Machine Learning

To optimize model performance and retain floating-point precision, given that the candidate area has a difference of around 10^{-4} in latitude and longitude, we normalize the latitude and longitude coordinates to fit within the candidate bounding box.

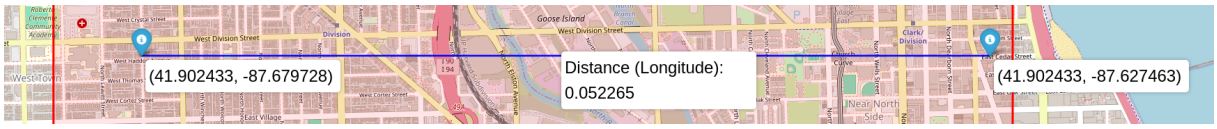


Figure 23: Difference in terms of Longitude for two random points about the Chicago Candidate Area

The normalization of each latitude lat_i and longitude lon_i is computed as:

$$\text{normalized\_lat}_i = \frac{\text{lat}_i - \text{min\_lat}}{\text{max\_lat} - \text{min\_lat}}, \quad \text{normalized\_lon}_i = \frac{\text{lon}_i - \text{min\_lon}}{\text{max\_lon} - \text{min\_lon}}$$

This scales the coordinates into the range $[0, 1]$, ensuring that there is consistent input for machine learning models while maintaining relative spatial relationships. *The values are simply denormalized during prediction.*

Results are therefore readable in terms of a proportional step in the $20\text{km}^2 area$.

7.13 Learning Rates for Machine Learning

Learning rates are determined relative to the precision requirements of the domain. In the case of the *Chicago Candidate Area* we consider the normalized values in lieue of the actual Lat / Lon values.

We consider a non-linear learning rate relative to the number of epochs:

$$\text{lr}(epoch) = \begin{cases} \text{lr} & \text{if } epoch < 25 \\ \text{lr} \cdot e^{-0.1} & \text{if } epoch \geq 25 \end{cases}$$

A decaying learning rate is justified as it allows the model to make larger updates initially, enabling faster convergence. As training progresses, reducing the learning rate helps to fine-tune the model, ensuring stable and precise steps in our gradient descent, as we *ideally* converge to the optimal model. This approach balances speed and precision

We employ the **ADAM** optimizer with a seed of **0.001**. Adam automatically adapts the learning rate for each parameter based on the first and second moments of the gradients; the scheduler alters the base learning rate. The initial rate is chosen based on *sensible stariting points* [29] based on the performance of the *training set* during a few epochs; it is more likely than not that the optimal LR seed is not chosen. This is a deliberate consideration given the time cosntraint and the lage number of GPU hours to usefull train the model(s).

We select **ADAM** due to it's versatility and performance for complex models.

The parameters for the scheduler, the seed, and the optimizer type are hyper-parameters which can be tuned.

We select only the specified seed values due to the scope of the implementation and time / resource constraints.

7.14 Ensemble Learning to improve Net Performance

Ensemble methods improve the limitations of individual models. Using *bagging*, we average the outputs of three distinct models to enhance accuracy and robustness. This helps address performance variability from discrete models.

We create a *psuedo ensemble* - Each of the three discrete models is trained on some different train / text / validation combination, however only singular models of each are used (results are simply averaged)

To process each image, we generate two distinct datasets: one of explicit features like signage and another for broader visual informers. Both paths are trained on the Mapillary dataset and used for prediction. Resource constraints necessitate embedding-based processing rather than using full image data, optimizing performance without sacrificing key information. The *Overall Image* embeddings set informs a **Clustering** model and an **Attention inclusive Machine Learning** Model.[47] The set of *segments* defines all signage extracted from the source images and is used to train a **Machine Learning Model**

7.15 Hyperparameter Tuning

Conducting an exhaustive hyperparameter search is not feasible due to time constraints and the inability to effectively test on a smaller sample of the data due to the required crossover of GeoInformers between images. Instead, a *sensible* set of hyperparameters is selected, and their performance is evaluated based on the loss and validation metrics over a single epoch. This approach provides insight into the training trends and helps identify the most promising configurations.

While this method is not ideal, it is a practical compromise given the limited time available.

For the learning rate (LR) in **??**, we test values ranging from 0.0001 to 0.001 in 5 evenly spaced increments and choose the one that performs best under the specified conditions. Similarly, for the activation functions in the dense layers, we experiment with combinations of 'ReLU,' 'Leaky ReLU,' and 'ELU,' selecting the optimal mix based on the same evaluation criteria.

This streamlined process allows us to efficiently identify a strong starting point for model training, A retrain or retool of this method **should** conduct a hyperparamter search.

Note: A great deal of the hyperparameters, unless not explicitly mentioned are the default reccommendations from *Keras* Documentation [**kerasdocs**]

Hyperparameter	**Value**
Text-Embedding Dimension	128
Text-Embedding Cutoff	13 Observations
OCR Confidence - Text	0.7
OCR - NMS	0.3
Colour HSV Bins	64
Object Embedding Learned Size	128
Batch Size	512
YOLO Confidence	0.25
Dense Activation (funcs)	ReLU
Architecture	**??** 10
Learning Rate initialization α	0.001
Learning Rate Decay Rate	0.99
Optimizer / Seed α	Adam / 42
Epochs	50
Early Stopping	Yes (patience=10)
Normalization	BatchNorm
Initializer	Xavier (default)
Gradient Clipping	5.0
Scheduler	StepLR (step=30, gamma=0.1)
Dropout Rate	0

Table 8: Hyperparameter Table

This is a non comprehensive list of hyper parameters; however it serves to show what *has* been trained and the results therein. The architecture of models **??** 10 used is itself a hyperparameter as are the sizes, embedding sizes, layers and width of all layers therein.

7.16 Data Representation and Generating Embeddings

Resource constraint necessitates that embeddings are used as an intermediary for the images . A lengthy 'ingest process' is defined in 84.

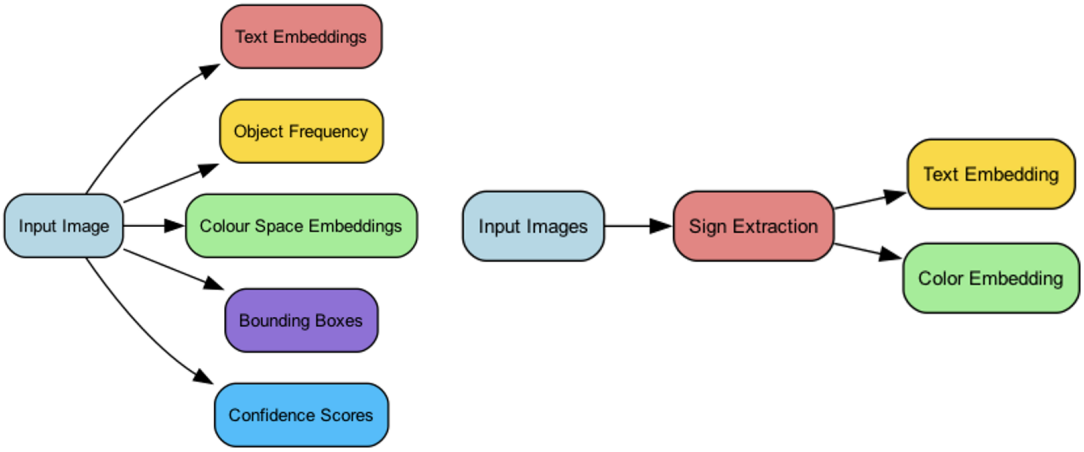


Figure 24: The two embeddings sets created. **Left:** The overall image embeddings totaling 151,510 records. **Right:** The dataset containing signs that have been ingested as signs **381,234** records.

7.17 Colour Histogram Embedding

The overall colourspace of an image is a telling Geolocator which permeates across input variables. The roofline in a given image, for example, is a telling geolocator, yet the sky is consistent but is not a useful feature for our model.

This particular input feature for each of the model branches is essentially preprocessing only; we create colour histograms which represent the distributions of the local colours.

We convert the image to the HSV colour space to better capture colour variations and compute a histogram for each channel (Hue, Saturation, and Value). These histograms are then flattened and normalized to create a the colour vector.

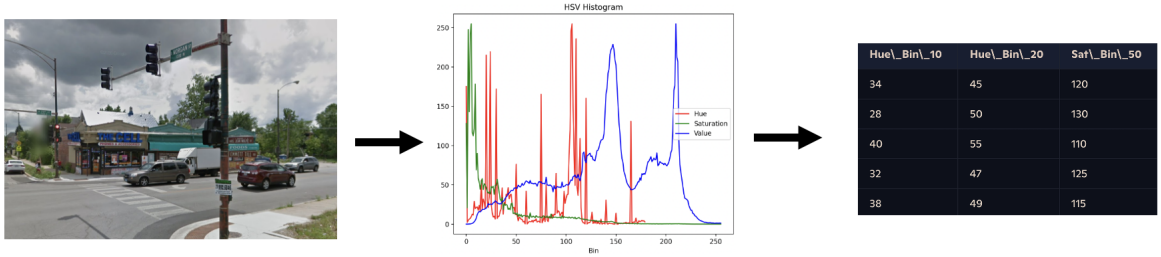


Figure 25: Example of HSV extraction from **??** and a sample embedding (truncated)

The extracted histogram is used as an input feature in a separate branch of the neural network. This branch processes the histogram through some number of dense layers to identify significant colour patterns. The output from this branch is concatenated with the object embeddings and the text embeddings, before the attention layer(s).

The embeddings for the images (as histograms) not learnt in the interest of computability. A bin size of 64 is used for each pass, resulting in an embedding tensor of **Batch_Size** $\times$ **3** $\times$ **256** for **both** the overall image profile and the individualized sign / advert profile.

7.18 Confidences and Bounding Boxes

While these features are extracted and stored, they are ultimately not used in the finalized model. The overlap and relative distance between bounding boxes is a particularly informative GeoFeature, however due to time and performance limitations they are discarded. Overlap and relative distance must be normalized and perspective corrected.

7.19 Data Ingest and Embeddings Generation

Programmatically, each image is loaded, and the corresponding text, color histogram, and object tensors are generated *in addition to the unused Bounding Box and Confidence Tensors*. Tensors are not padded at this stage.

While this approach may seem to disregard the relationships between signs within the same image, it offers significant computational advantages. By isolating sign embeddings, the model can process them in parallel, reducing training time and resource requirements. Furthermore, this separation simplifies the architecture, making it more straightforward to achieve a useful model without a hyperparameter search on limited hardware .

The most prudent step here would be to maintain the relationship between signs, as their co-occurrence within an image could provide valuable contextual information. However, in this implementation, computational efficiency is prioritized over explicitly modeling these relationships.

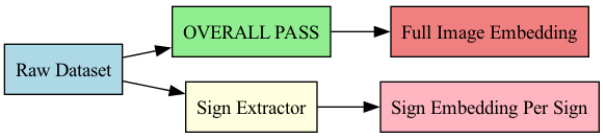


Figure 26: Datasets Embeddings Diagram

7.20 Tensor Size and Shape

Feature	Overall Image Profile Shape	Sign Shape
Latitude	$B \times 1$	$B \times 1$
Longitude	$B \times 1$	$B \times 1$
ID	$B \times 1$	-
Num_detections	$B \times 1$	$B \times 1$
Text_Embeddings	$B \times 12 \times 128$	$B \times N \times 12 \times 128$
Object_Class_Vector	$B \times N \times 13$	-
Colour_Histogram	$B \times 3 \times 256$	$B \times N \times 3 \times 256$
Confidence	$B \times N$	-
Bounding_Boxes	$B \times N \times 4$	-

Table 9: Tensor shapes for Overall Image Profile and Sign datasets. Here, **B** represents the batch size, and **N** represents the number of detections (defined as *Num_detections*).

Num_detections is simply used to formalize the splits for dividing contiguous flattened records into their respective tensors.

Object Class vectors are, where 5 instances of class 2 are detected and a single instance of class 1: [[1 2,2,2,2,2]] with no padding in the storage of the record.

Tensors are padded at load to ensure consistent shapes. CUDA with JIT compilation necessitates padding tensors to consistent sizes for implementation, yet tensorflow for *Metal* does not. Padding creates a platform-safe embedding as the raw **Overall Pass** is of variable size.

7.20.1 Batch Size

We employ a **batch size of 512 records** universally, as this is the maximum batch size supported by the available memory. This decision is primarily driven by the inclusion of Convolutional neural Network Layers and Attention Layers, whose compute can *blow out* memory rather quickly.

One key consideration of using a batch size of **512** is its impact on training efficiency. Larger batches enable more parallel processing, which can significantly reduce the time required for each training iteration. This is especially important given the size of our dataset and the time constraints. By maximizing the batch size within memory limits, we ensure that the training process is as fast as possible, enabling us to explore more model configurations and hyperparameters within the allotted time. However, the model may not *easily* converge to an optimum.

Larger batch sizes tend to produce smoother gradient estimates *batch gradients with smoother gradients*, which can lead to more stable convergence during training. While full convergence to a global optimum is unlikely due to the dataset's size and complexity and *choosing what will fit best* approach to hyperparamter selection and tuning; a batch size of 512 provides a **reasonable** approximation of the true gradient, helping the model reach a useful local optimum.

Smaller batches are often associated with better generalization due to the *noise* they introduce in gradient updates (i.e small batch creates psuedo randomness as a form of psuedo randomization and the benefit is also at odds with convergence speed and training stability....). This effect can be mitigated through techniques such as learning rate scaling and batch normalization. The former has been used successfully by [15], who demonstrate that scaling the learning rate linearly with the batch size allows large-batch training to achieve comparable performance to small-batch training. However, once more for the sake of simplicity, we are choosing fixed size batches. It is simply too time consuming to compute against the entire dataset some pass of differing batch sizes.

An additional consideration is Interoperability between NVIDIA CuDa And Apple Metal backed Machine learning Software and the simplicity of storing records to a predefined batch size as separate records compared to storing them as a *mega record*

7.20.2 Logging for Analysis

Naturally, we track the loss function for each epoch, including values for each training batch, validation batch, and the test set, for later analysis. Since we're not performing a grid search to fine-tune hyperparameters but relying on automatic tuning through the ADAM optimizer, it's beneficial to record how the weights contribute to each layer over time. This information helps us interpret the model's behavior and identify areas for future improvement.

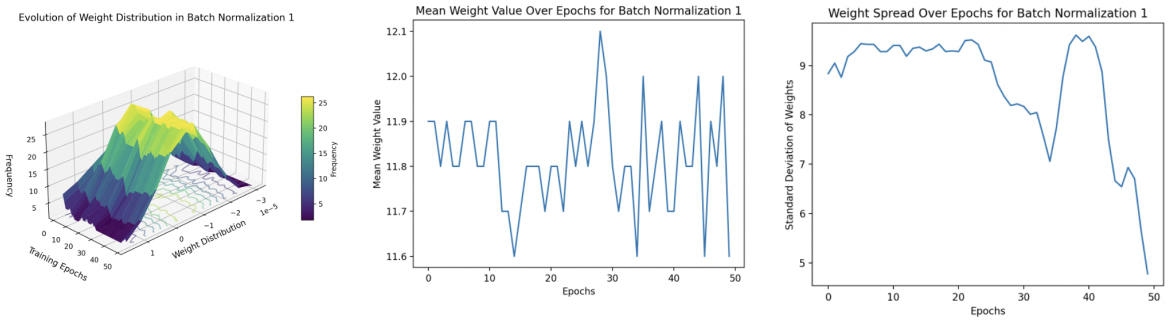


Figure 27: sample of logging for a single layer over all epochs

We record **mean weight values over epochs** for the purpose of modelling how the average weight in each layer evolves over the training process. Tracking the mean provides insight into whether the model is over- or under-adjusting the weights. i.e. if the mean weight increasingly signficantly could indicate that the learning rate is too high, possibly leading to instability or poor convergence.

The SD of the weights over epochs is also recorded (or, rather, derived from the histograms) for the purpose of monitoring the spread of the weights within each layer. A small SD indicates that weights converge to similar value, where the model may fail to learn compelx patterns. A large SD may indicate overfitting by particular ndoes.

The results for all passes and all trainings are simply stored as JSON objects and are available in the code repository under /training Results

7.21 The Signage-Only Neural Network

7.21.1 Architecture

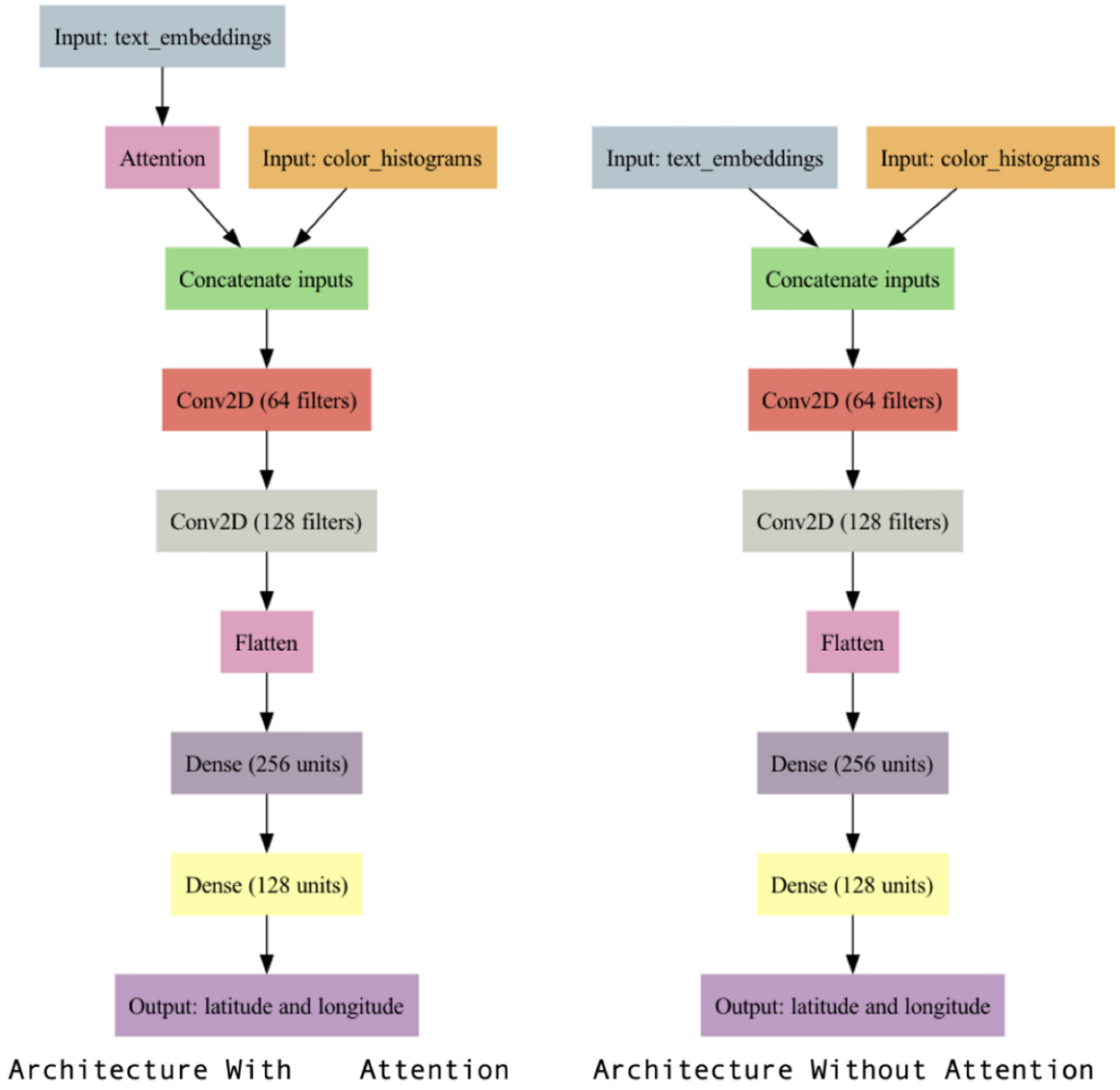


Figure 28: Model Architecture for the signs pass with and without Attention

The Signage Only branch of the overall architecture is given above.

Full trainings are run over each for the purpose of an ablation study. The neural networks are defined in Keras.

7.22 Training the Overall image Branch

The *Left* branch of our proposal uses the full embeddings generated in **??** This branch is given a randomized split of test / train / val as at 10%, 80%, 10% of records

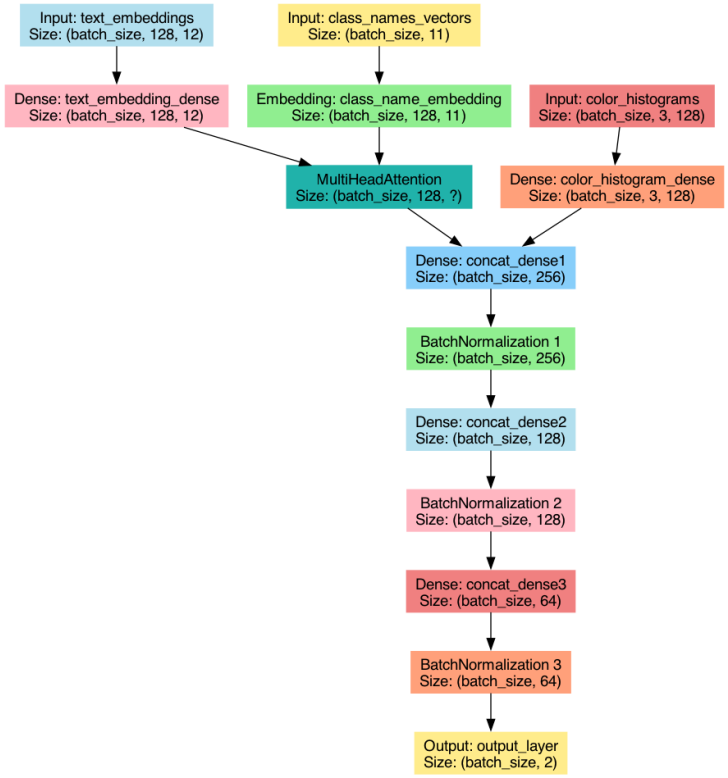


Figure 29: Full Image Model with Multi Head Attention

Attention is a core consideration, we define a second model for the purpose of comparison through an ablation study and to determine if indeed our thoughts are correct regarding it's impact.

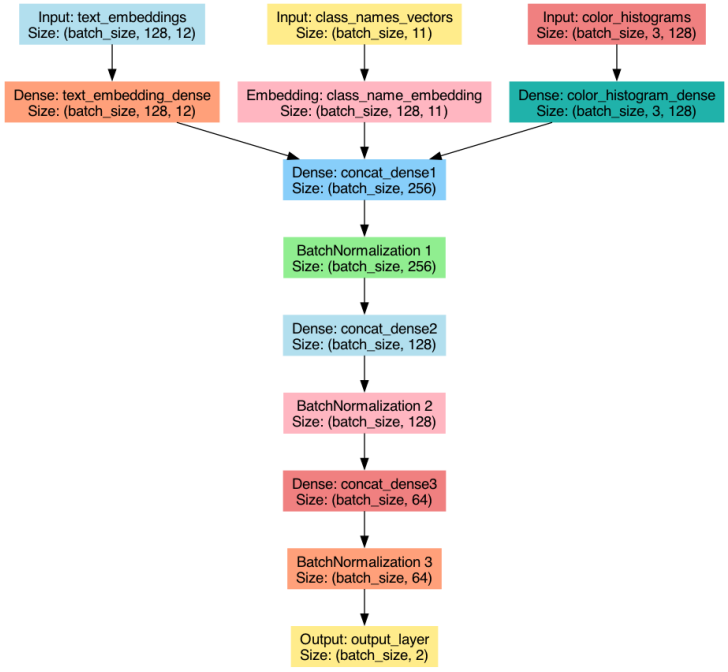


Figure 30: Full Image Model sans Multi Head Attention

Layer	Model with Attention
Input	Text Embeddings → B, 128, 12 Color Histograms → B, 3, 128 Class Names Vectors → B, 11
Embedding	Class Names Embedding → B, 128, 11
Dense	Text Embedding Dense → B, 128, 12 Color Histogram Dense → B, 3, 128
lightgray Attention	Multi-Head Attention → B, 128, ?
Dense	256 → 128 → 64
Output	Output Layer → B, 2
Batch Normalization	256 → 128 → 64

Table 10: Summary of Layers and Tensors for the Model with Attention

7.22.1 Attention in Context

Attention mechanisms [47] are particularly valuable for extracting relevant Geoinformers by focusing on different parts of the input that contribute most to the task at hand. In this framework, we apply multi-head attention to the combined embeddings from the OCR (Optical Character Recognition) pass and the object detection embeddings.

The multi-head attention mechanism enables the model to attend to various features of the input in parallel, allowing it to capture intricate relationships within the concatenated tensor for text extracted from the OCR pass and spatial features detected by the object detection model. This enables the model to more effectively *focus* on key patterns and spatial dependencies within the embedding space that are most relevant for our geolocation prediction, ideally improving its ability to extract the *most meaningful* Geoinformers.

Attention mechanisms are computationally expensive to train because they require significant computational resources to process multiple attention heads and large amounts of data. Each attention head independently processes the input data, which increases the overall computational load. Additionally, the parallel processing of multiple features and the need to capture intricate relationships within the data contribute to the high computational cost.

Both Models include a **4-headed** multi-head attention layer wherein the attention mechanism has four separate heads, each focusing on different parts of the input simultaneously. ideally capturing broader and less verbose patterns than a a single or dual headed attention layer. The sizes of other layers are considered to *fit this one in* given the GPU memory constraints.

The use of multi-head attention can help overcome noise and irrelevant text by focusing on the most relevant parts of the input. However, this approach might conflict with the text embedding, as the attention mechanism could prioritize different features. This is captured in a brief Ablation study.

7.22.2 Learned Embeddings

Whilst an *off-the-shelf* text-embedding model (sBERT) [35] is employed for generating embeddings from text derived from *EasyOCR* [22] and a form of simplified embedding is used to form one-hot vectors for the purpose of clustering analysis, learned embeddings are used for the object frequency embeddings generated by Ingest Yolo.

When concatenated with text embeddings, this approach leverages both visual and textual data. The learned embeddings provide detailed insights into the visual content, while the text embeddings capture the semantic meaning. This combined representation enhances the model's performance in tasks like object detection and classification by offering a more comprehensive understanding of the data.

The core benefit here is that learned embeddings are tailored to the dataset, capturing the unique distribution and frequency of objects. This specificity allows the model to make more informed decisions, improving overall accuracy and effectiveness. *Essentially*, the embedding layer simply weights the importance of each class in a way that complements the text embeddings in relation to the distribution of those detections in the dataset.

7.22.3 Dense Layers and Gradient *Management*

The number of layers and units in each dense layer are chosen as hyperparameters to balance model complexity and performance. We opted for 256, 128, and 64 units progressively to allow the model to capture increasingly refined features whilst mitigating overfitting. A comprehensive hyperparameter search, such as grid search, is deemed non-viable due to time and resource constraints.

Activation functions and regularization techniques like dropout are also key hyperparameters. ReLU is used throughout the layers for its ability to accelerate training and mitigate the vanishing gradient problem, although our model is not *particularly deep*. The final layer uses a linear activation function for regression.

Batch Normalization layers are included to stabilize and potentially accelerate training by reducing internal covariate shift. This helps the model generalize better and achieve faster convergence. These are applied before each dense layer after concatenation

Dropout is included as a hyperparameter; the number of dropout layers and their values (where, of course a value of zero simply means the layer is removed)

7.23 Creating the clustering

This branch is given a randomized split of test / train / val as at 10%, 80%, 10% of records

Generating clusters is computationally very expensive; we use a combination of text embeddings and translate the classnames vectors to one-hot encoded class names to form our feature vectors. The text embeddings are dense vectors that capture semantic information from the text, while the one-hot encoded class names provide categorical information about the data. Similarly to Attention networks in **??** the one hot vector values represent a more heavily weighted similarity measure than the *distributed over the embedding* text at this particular location of capture

While dimensionality reduction techniques like Principal Component Analysis (PCA), t-Distributed Stochastic Neighbor Embedding (t-SNE), or Uniform Manifold Approximation and Projection (UMAP) could be applied to reduce the size of these feature vectors, it's important to note that such reduction might lead to loss of information. Since text embeddings capture nuanced semantic relationships and one-hot encoding provide distinct categorical information, reducing their dimensions could potentially remove valuable information that contributes to the clustering process.

Time Constraints necessitate foregoing these steps. We assume that all contributing columns are *important*. We cannot be certain of the effect of dimensionality reduction and whether it will improve the clustering performance.

to balance the trade-off between computational efficiency and the preservation of meaningful information.

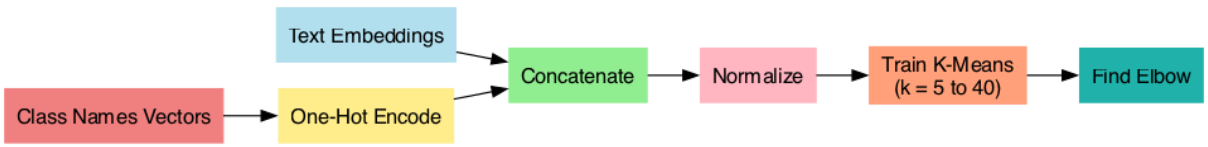


Figure 31: Enter Caption

7.23.1 Contributions of K-means

We determine the optimal number of clusters, **K**, using the *Elbow Method*. This involves plotting the explained variance against the number of clusters and identifying the "elbow" point—where the rate of decrease in variance sharply slows. At this point, adding more clusters no longer provides significant improvement in capturing the underlying structure of the data.

Greater **K** from this point can lead to overfitting.

In the context of our 20km$\hat{2}$ area in chicago, clusters would have equal geographic *areas* covered by their *decision boundaries*; however we expect this to not be the case given the non uniform geographic distribution of the dataset.

K-means also validates the effectiveness of our text embeddings and one-hot encoded geographic objects as a *geoinformative pattern*. By evaluating cluster separation with metrics like the **silhouette score**, we can confirm that the clusters are distinct and meaningful, indicating that our embeddings capture real patterns.

The clarity of clusters and their alignment with geographic patterns are crucial. High silhouette scores that correspond with real-world features demonstrate the reliability of our embeddings and encodings, enhancing the trustworthiness of our results.

7.24 Putting it Altogether and Quantifying Results

The *ensemble* in this proposal is a *psuedo ensemble*. Each model is trained on a unique test / train / validation split and the outputs are averaged.

Efficacy of this implementation is ¡word for things building on each other¿

7.25 Training The Model

With these in mind, the model is trained in the following way:

Discretion All trainable variables are trained through a single (overall) backprop step. The *off-the-shelf* and *preprocessing* models are not retrained as the Two core models are trained from the generated Embeddings.

Loss and MSE metrics Are recorded per epoch, per batch, per data type (train / test /val). and inform the ...

... Stopping Criteria and Evaluation Metric While the environment limits the scope of training and the model is therefore unlikely to converge before any reasonable number of epochs, we define a stopping criterion as an assessment of the mean of the average loss.

Where possible **50 Epochs** are simply used; the **MSE** values at the end (MSE being simply euclidean distance relative to the normalization). The MSE at this point is *as far as we can go*.

$$\text{Mean Loss} = \frac{1}{N}\sum_{i=1}^{N}\mathcal{L}_i$$

If this metric is below a defined threshold, the training is stopped early.

Data Splits Our Data is separated into Training, Test and Validation sets, This separation is random and across time as well as LAT / LONG. Due to the sparsity of data, Cross validation and K-fold validation may be employed (depending on the performance of the model)

Optimizer For training the model, we utilize the Adam optimizer, which is well-suited for handling sparse gradients on noisy problems.

K-means / Clustering is trained in one pass as a supporting Model to the core 2 Models.

8 Results

Our results are downstream dependant; the performance of the object detection model imapcts the performance of the regression network for geolocation.

50 epochs are used for all training instances across the proposal to generate all results. There is no early stopping.

A single fit of Hyperparameters are chosen except where specified.

8.1 Dataset Size

Total Images:	150,510
Images with Detections:	111,125
Total Number of Detections of Signs:	308,618
Total Number of Detections of Objects:	86324
Detection Rate:	73.83% of images contain detectable signage
Total Non-Sign Detections:	86,324
Images with Non-Sign Detections:	49,956
Proportion of Dataset with Non-Sign Detections:	33.15%

Table 11: Detection Summary

8.2 Environment and Hardware

The model(s) are trained on one of the following two systems, rather interestingly, we see an insignificant difference in performance between the two. Models are descriebd where

Component	Specification
Graphics Card	1x NVIDIA RTX 3080 10GB
System Memory	32GB
Storage	Several TB of storage
ML Library and Acceleration FW	TensorFlow - CuDA Nvidia

Table 12: System Specifications

Component	Specification
Processor	Apple M1 Pro chip
CPU Cores	10-core CPU (8 performance cores, 2 efficiency cores)
GPU	16-core GPU
NPU	16-core *Apple Neural Engine*
System Memory	16GB unified memory
Storage	512GB SSD
Memory Bandwidth	200GB/s
ML Library and Acceleration FW	TensorFlow - Metal

Table 13: Apple M1 Pro 16GB Laptop Specifications

8.3 Results Context

The Model creates a regression of tuple values for Latitude and Longitude $(\hat{\lambda}, \hat{\phi})$. Accuracy is the *Euclidean Distance* metric or *Mean Squared Error* Metric.

Train / Validation and Test sets are proportionally 80%, 10% and 10% of the rows od the dataset and are randomized before training.

Raw results are normalized in and out of the candidate area.

The three branches (the sign branch, clustering and overall image pass) are discrete and their aggregate (average) values are the output of the total architecture.

Initially, consider each *Branch* of the network results independently

8.4 Converting MSE values and prediction values to Interpretable Distances

To convert the MSE values (in normalized degrees^2) to real-world distances (meters), we denormalize by the inverse to the normalization process relative to the bounding box and midpoints.

8.4.1 Training

Training Process The model was trained on the Full Image Embeddings dataset for 50 epochs in the CuDA environment.

Hyperparameter	Value
Main Activation	ReLU
Optimizer	Adam
Learning Rate	0.000001
Layers	[Insert Number]

Figure 32: Key Hyperparameters for Training

$$\text{lr} = \text{lr} \cdot \begin{cases} 1 & \text{if epoch} < 25 \\ e^{-0.1} & \text{otherwise} \end{cases}$$

Figure 33: LR Schedule

We break Latitude and Longitude Performance out into separate logs in addition to the combined values.

8.5 Results of Training the *Full Image Pass* with Attention

The *Full Image Pass* here is the sub architecture responsible for precessing agcross the *1 image 1 row* dataaset embeddings. Defined in 10

Initial results are given as:

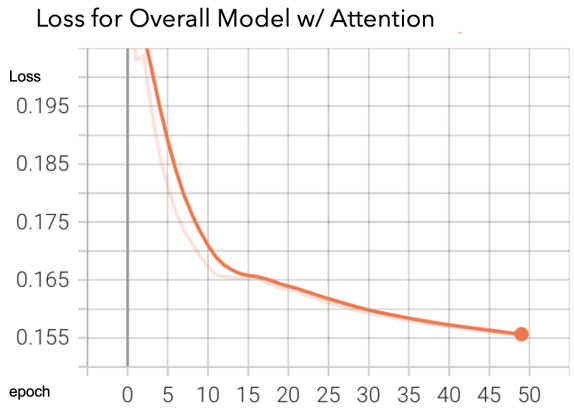


Figure 34: Loss total for Attention-Inclusive Model

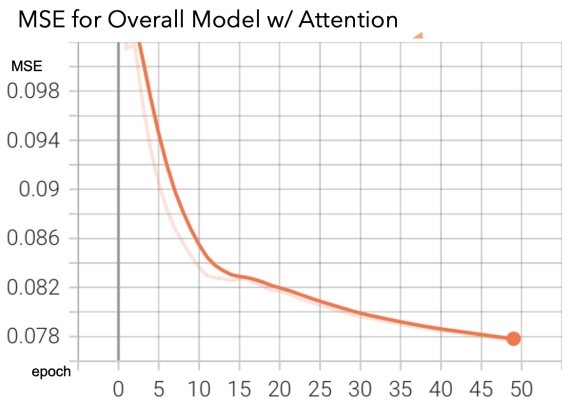


Figure 35: MSE (Lat/Lon Euclidean Distance) – Epochs 0-3 Omitted

Figure 36: Comparison of Loss and MSE for the Attention-Inclusive Model

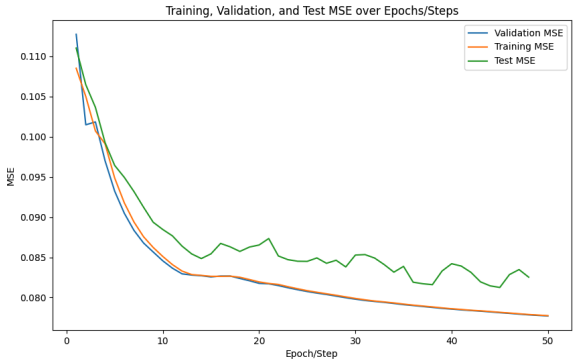


Figure 37: Attended Dataset MSE Comparison

Note that these two figures appear identical but are shifted about loss / MSE.

Note; MSE's are given in terms of the normalized result per 7.12 within the 20km$\hat{2}$ area.

Data Type	Final MSE	Mean MSE	Best Epoch	Best MSE	Variance
Test OVERALL	0.082532	0.087067	44	0.081253	0.000041
Validation OVERALL	0.077703	0.082938	49	0.077703	0.000049
Train OVERALL	0.077740	0.083131	49	0.077740	0.000049

Table 14: Model Performance Metrics for OVERALL Data Types

Data Type	Final MSE	Mean MSE	Best Epoch	Best MSE	Variance
Validation LAT	0.071460	0.075565	42	0.071093	0.000091
Train LAT	0.072827	0.074534	41	0.072429	0.000007
Train LONG	0.074751	0.076801	47	0.074190	0.000014
Validation LONG	0.076576	0.076316	48	0.072802	0.000024
Test LONG	0.080674	0.078415	29	0.062598	0.000542
Test LAT	0.073600	0.072296	30	0.065130	0.000020

Table 15: Model Performance Metrics for LAT and LONG Data Types

Data Fit The trained Attention model at 50 epochs does not show telling signs of overfitting. The final MSE values for the *Training (Overall)*, *Test (Overall)*, and *Validation (Overall)* sets are close. The *Training (Overall)* set has a final MSE of **0.078898**, similar to *Test (Overall)* at **0.078820**. The best MSE values are also similar, with *Training (Overall)* at **0.078898**, slightly higher than *Test (Overall)* at **0.078820**, suggesting similar performance on both datasets.

The mean MSE values show a slight increase for the training datasets. For example, *Training (Overall)* has a mean MSE of **0.086003**, while *Test (Overall)* is lower at **0.078823**. This indicates minor fluctuation during training, but the insignificant differences in final and best MSE suggest the model isn't overfitting. Thse results suggest that the model reacches a performance plateau reasonably early.

The model's predictions are, **on average,** off by approximately **1.2–1.3 km** from the true values over the test set. But *concerningly* the training, validation and test sets follow similar curves.

In terms of a 20km$\hat{2}$ area; this is a useful result; however the distribution of these results is an important consideration given the ill uniformed distribution of the source dataset.

differences.

The rates of change for MSEs on each set is considered for the purpose of analyzing convergence rate.

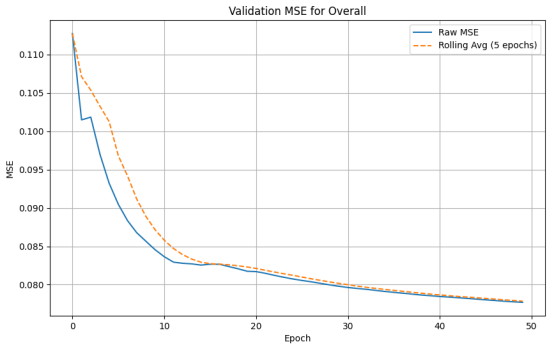


Figure 38: Validation MSE Plot

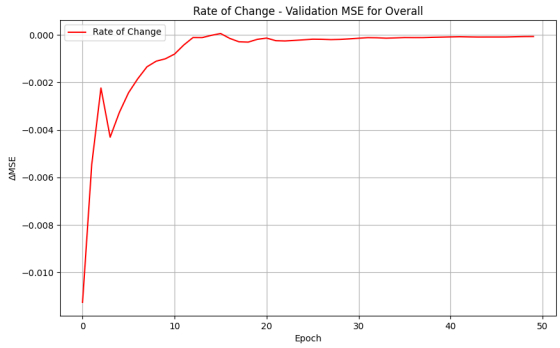


Figure 39: Validation MSE Rate of Change

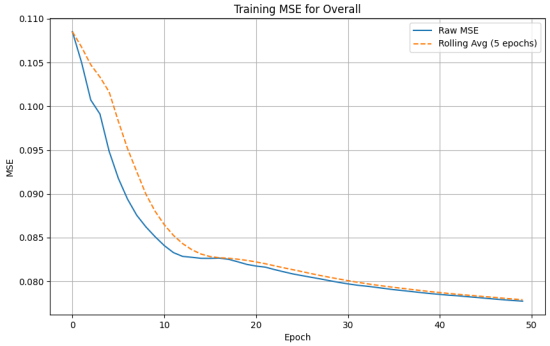


Figure 40: Training MSE Plot

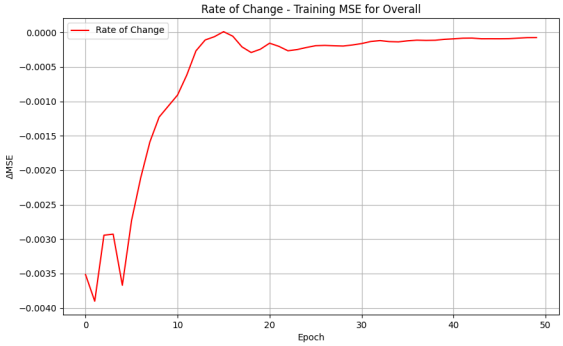


Figure 41: Training MSE Rate of Change

8.6 Full Image Pass *without Attention Layers*

The model is trained to the same Epochs with the same hyperparameters and the same dataset, the same test / train and validation split on the same hardware in *as close to as feasible* the same environment as the attention containing model.

We expect that attentnion will improve the performance of the model by reinforcing the importance of particular geoinformers or geoinformative pattern; training an additional layer naturally incrases the processing time and the memory load.

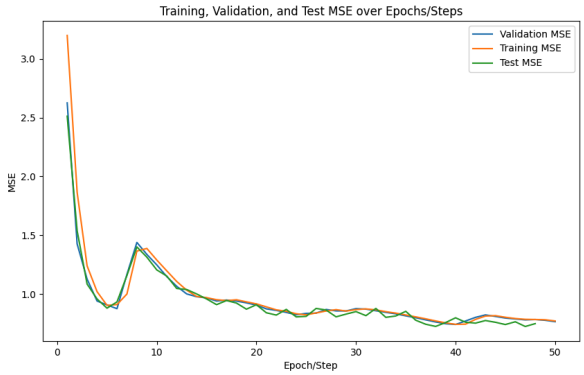


Figure 42: unattended Dataset MSE Comparison

A breif Ablation Study is used to determine the efficacy of introducing Attention to the Model talk about memory load constraining the kind of attention back in the attention bit

Metric	Original Time	New Time	Delta
Total Training Time	07:49:33	07:06:33	00:43:20
Time per Batch	00:00:01.316	00:00:01.195	00:00:00.121
Time per Epoch	00:09:23.66	00:08:32.04	00:00:51.62

Table 16: Training Time Summary with Improvements

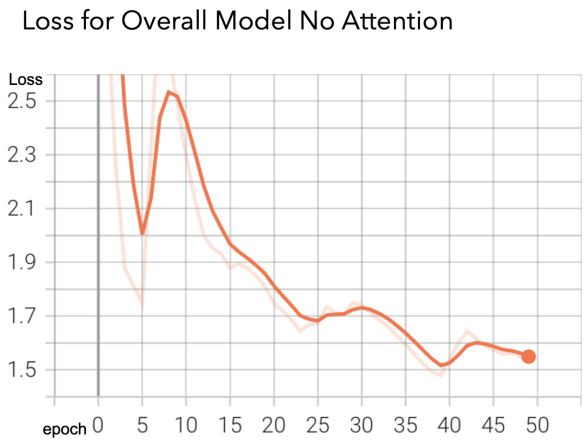


Figure 43: Enter Caption

Dataset	No Attention	Attention	Difference
Validation MSE (LAT)	0.085201	0.071460	-0.013741
Test MSE (LAT)	0.081793	0.073600	-0.008193
Train MSE (LAT)	0.174190	0.072827	-0.101363

Table 17: Comparison of Attention vs No Attention for LAT Metrics

Dataset	No Attention	Attention	Difference
Validation MSE (LONG)	0.112935	0.076576	-0.036359
Test MSE (LONG)	0.086168	0.080674	-0.005494
Train MSE (LONG)	0.143742	0.074751	-0.068991

Table 18: Comparison of Attention vs No Attention for LONG Metrics

Dataset	No Attention	Attention	Difference
Validation MSE (OVERALL)	0.763991	0.077703	-0.686288
Test MSE (OVERALL)	0.747200	0.082532	-0.664668
Train MSE (OVERALL)	0.769808	0.077740	-0.692068

Table 19: Comparison of Attention vs No Attention for OVERALL Metrics

Performance Metrics

8.6.1 Sample Results of the Attention Exclusive Model

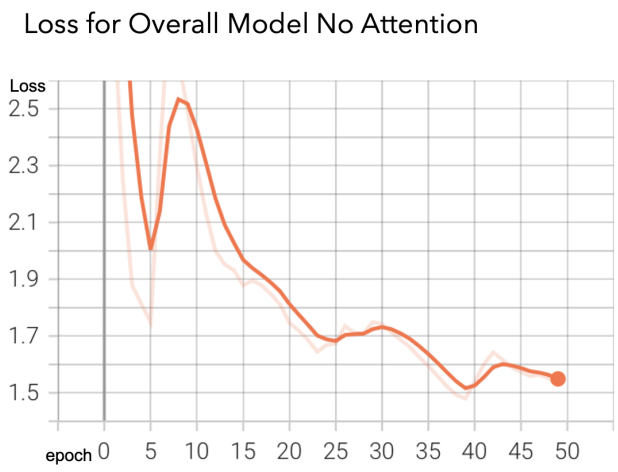


Figure 44: Test MSE Plot

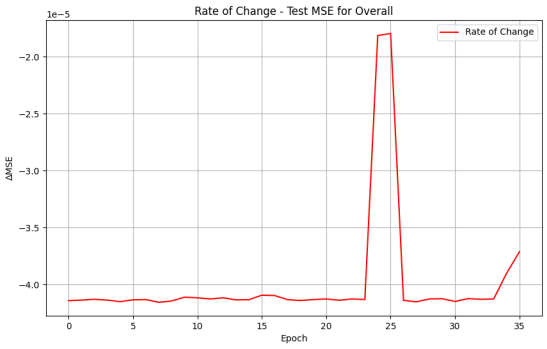


Figure 45: Test MSE Rate of Change

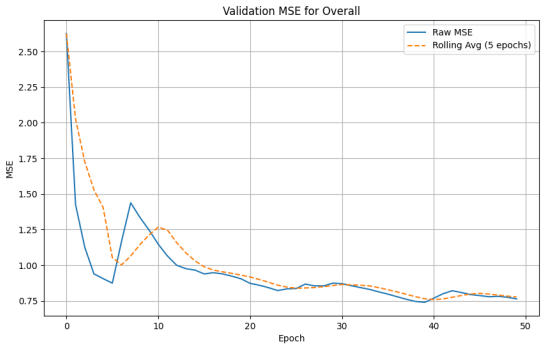


Figure 46: Validation MSE Plot

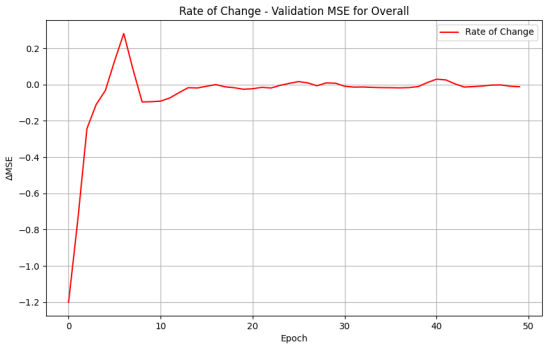


Figure 47: Validation MSE Rate of Change

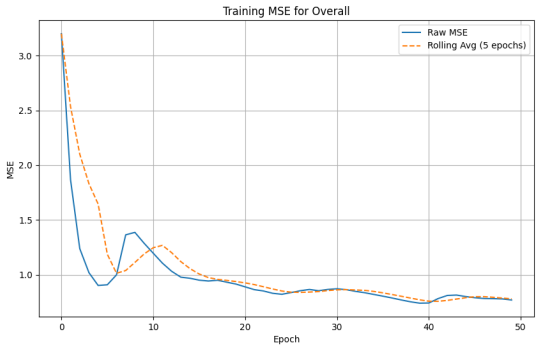


Figure 48: Training MSE Plot

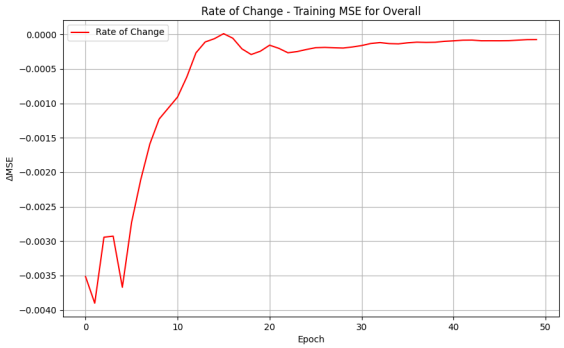


Figure 49: Training MSE Rate of Change

8.7 Comparison Of Attention Exclusive and Inclusive Networks

Predictions Density without the Attention Layer

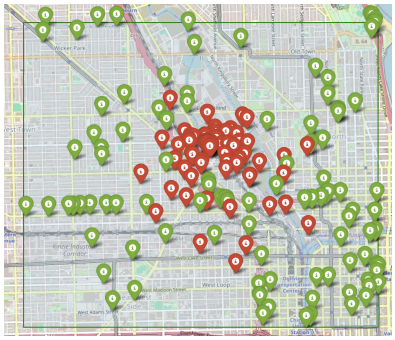


Figure 50: Sample Predictions and true locations (red, green)

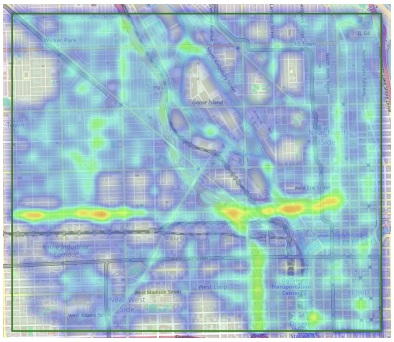


Figure 51: True Locations in a Test set Density

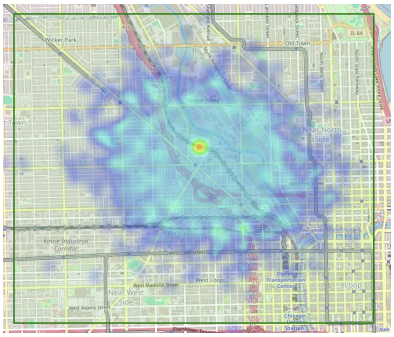


Figure 52: Predicted Locations in a Test set Density

With Attention

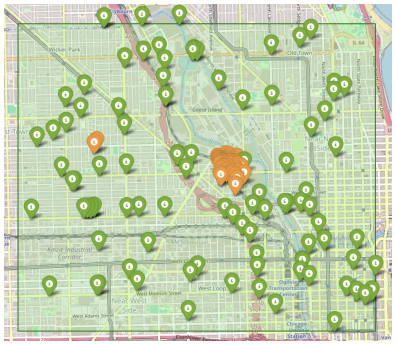


Figure 53: Sample Predictions and true locations (red, green)

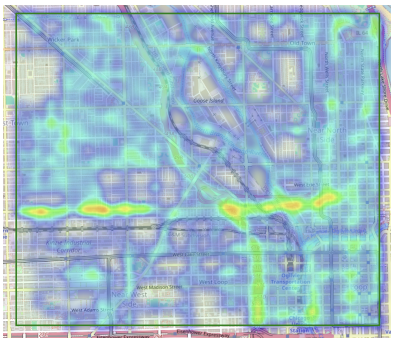


Figure 54: True Locations in a Test set Density

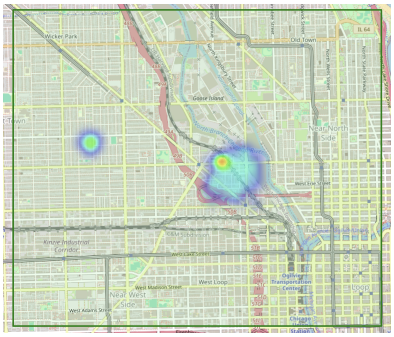


Figure 55: Predicted Locations in a Test set Density

Attention appears non beneficial to the network; the spready-out ness of the model is hugely reduced by the introduciton on the attention ;ayers; predictions are far more concentrated.

Final MSE			
Model Variant	MSE	Train	Val
With Attention	0.077696	0.077740	0.077703
Without Attention	0.761858	0.769808	0.763991
Difference	0.684162	0.692068	0.686288

Table 20: Final MSE Comparison: With vs. Without Attention

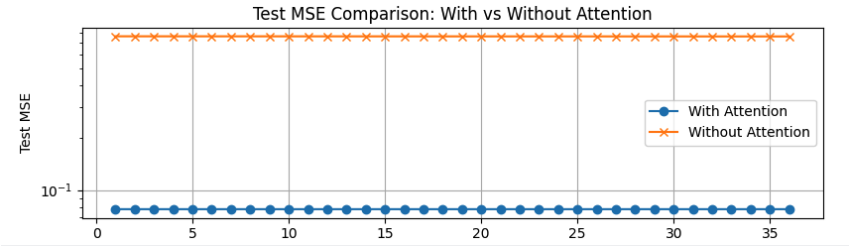


Figure 56: Test Set MSE on final Epoch For Full Dataset

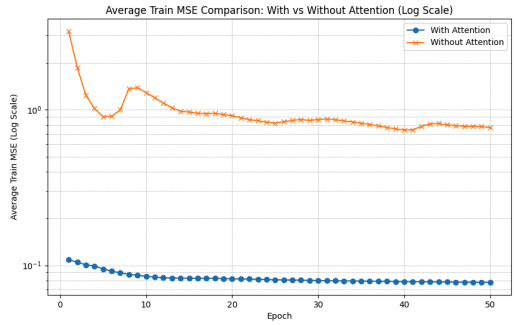


Figure 57: Train Set MSE on final Epoch For Full Dataset

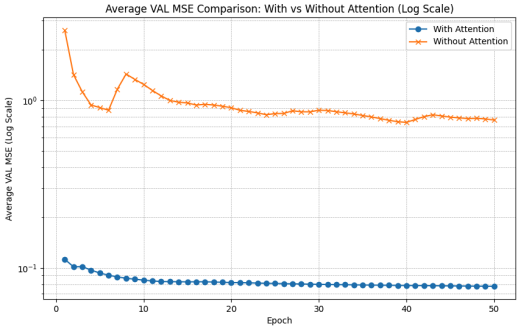
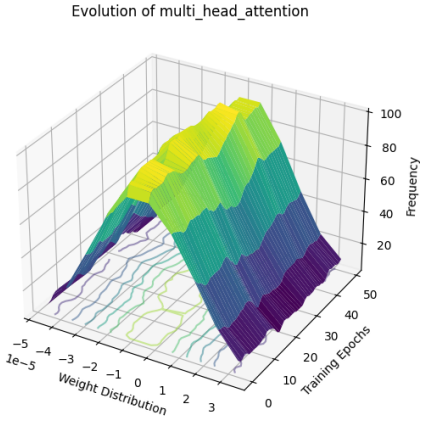


Figure 58: Val Set MSE on final Epoch For Full Dataset

Figure 59: Comparison of Attention-Based MSE Results

Such a response would indicate that the attention model is *narrowing in* on some centralized feature *see overfitting* pr it is not generalizing *away* from the initial state.



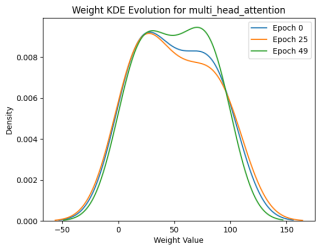


Figure 61: Weight Evolution for MultiHead Attention Layer

Figure 60: Weights Distribution Over Epochs (Multihead Attention)

And indeed the shape of the model is not sigicniantly *flattened* or *skewed* between epochs 0 and 50; indicating

8.8 The Signage Only Network Pass

The signage only pass is trained to the same learning rate curve, optimizer and the ssame normalization / denormalization as a 1-per row sign embedding from the **signs dataset**.

table: Number rows: Number Batches Batch size: Number Epochs Training Time:

A short ablation study is used to assess the efficacy of the attebtuib layer in this mdoel. Predictions are given as the aggregate of all isgns extracted from an image when it is applied ot an unseen image; however for the purpose of training signs are autonomously linked to the locale of capture.

Model responses once more are given as a normalized float value.

8.9 Signage Results

8.9.1 Results in Context

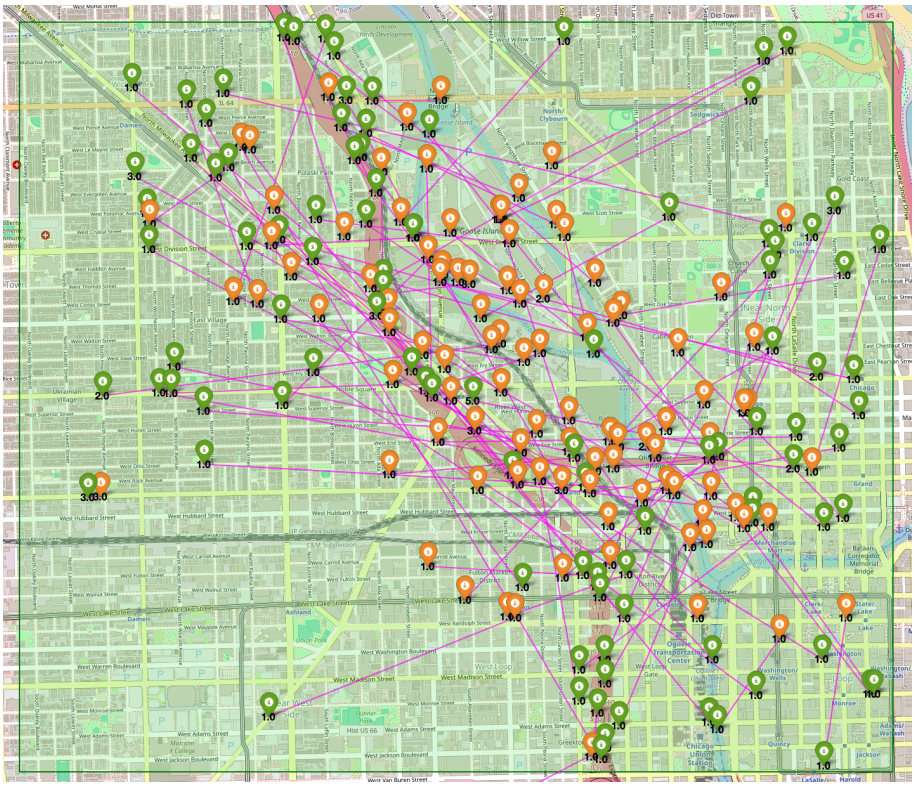


Figure 62: Predictions Of Locations of Images Based on Signage with Attention sampled from unseen / test set, Values indicate the number of signs detected at that location (for that image in the test set)

As the final predictions are based on an *aggregate* of outputs from the network for each sign, we can filter the detections by a particular signage frequency cutoff, here **Num Signs** $\geq$ **8**

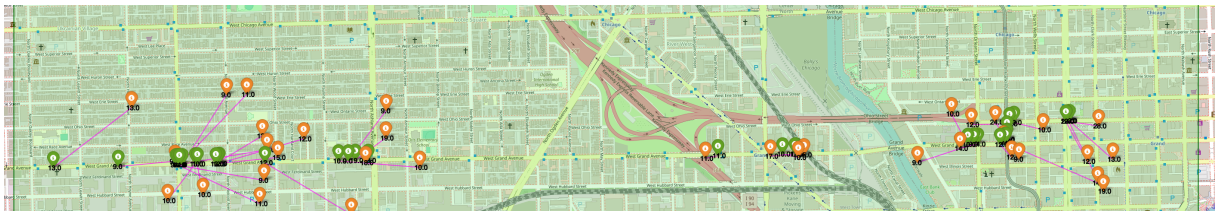


Figure 63: Images with Detections less than 10 for the Attended Signage Network

Herein, the results are given with the context that the network architecture has no threshold for number of detections and that multiple images may have shared the same location; we can preface the results with the inclination that increasing detections increases the predictive accuracy of the model; there is a direct correlation between the discriptivenes of the text informer and the usefulness of that iagme as a geoinformer:

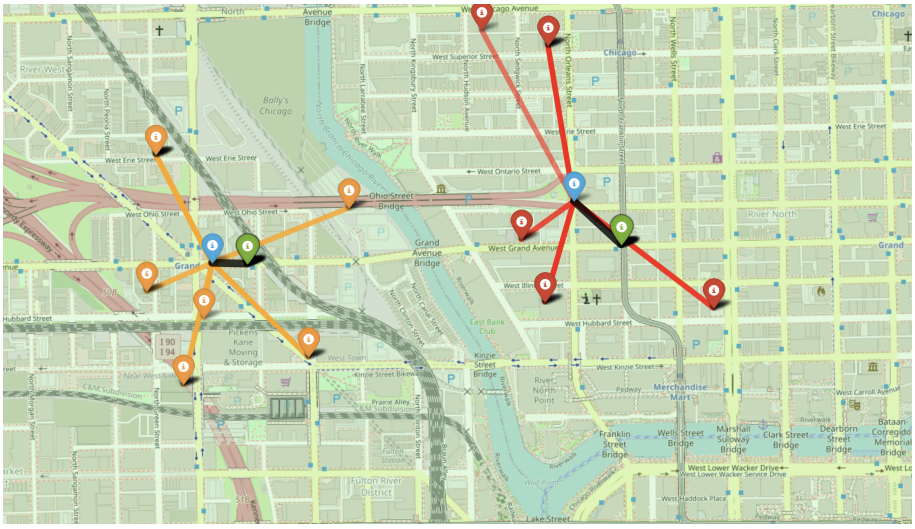


Figure 65: Centroids Predicting the location of capture through averaging; blue representing prediction ,green the true values and the colours the various predictions for signs

.

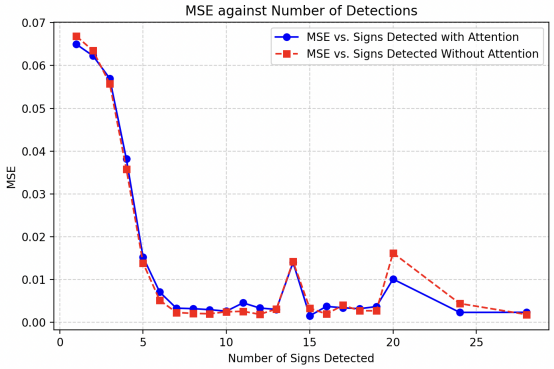


Figure 64: MSE falloff with increased i

In fact,this result is directly observable in a given sample from the test dataset.

Locations with large numbers of detections tend to be the product of overlap in the dataset and are once more concentrated about Kingsman Highway in the Chicago Dataset.

However, unlike the Overall Image only network, signage through our model is a clear and useful Geoinformer in it's own right.

It is, however plausible that this is a combinatorial product of the density of the training set.

8.10 Ablation Comparison for Attention within the Signage Location Model

Attention in this model does not appear to have a significant impact, as observed in the *overall model results.*

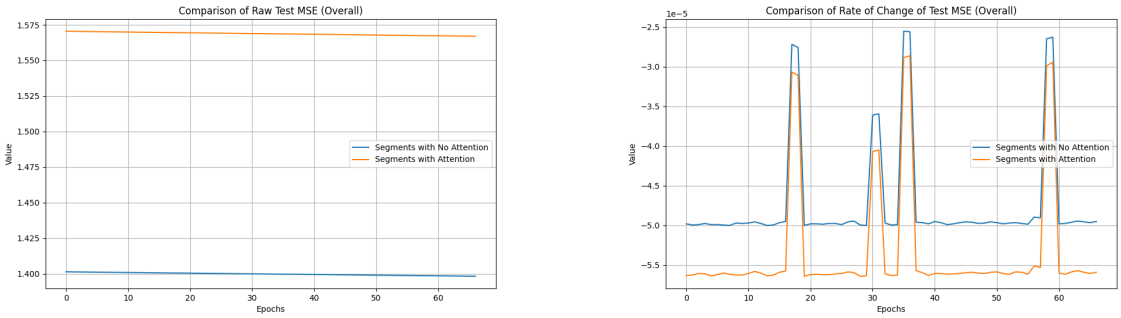


Figure 66: Comparison of Attended and Non-Attended MSE for the Test Set

The loss results, specifically the MSE component of the loss function, indicate that attention provides refinement from the *start*, similar to the overall pass.

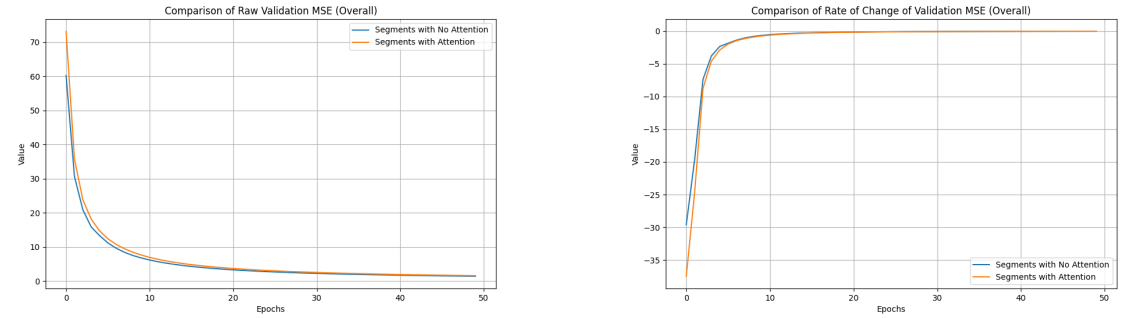


Figure 67: Comparison of Attended and Non-Attended MSE for the Validation Set

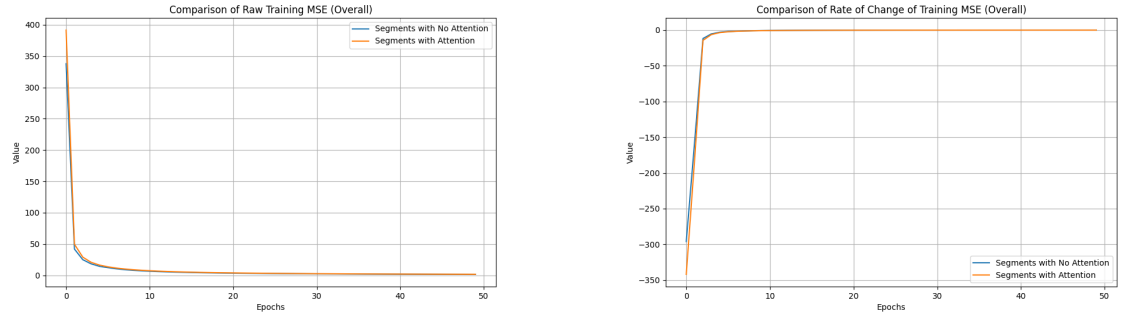


Figure 68: Comparison of Attended and Non-Attended MSE for the Training Set

Dataset	**No Attention**	**Attention**	**Difference**
Validation MSE (LAT)	0.061959	0.061064	-0.000895
Test MSE (LAT)	0.071141	0.061974	-0.009167
Train MSE (LAT)	0.021392	0.022545	+0.001153

Table 21: Comparison of Attention vs No Attention for LAT Metrics

Dataset	No Attention	Attention	Difference
Validation MSE (LONG)	0.057704	0.056260	-0.001444
Test MSE (LONG)	0.059359	0.052322	-0.007037
Train MSE (LONG)	0.020633	0.021838	+0.001205

Table 22: Comparison of Attention vs No Attention for LONG Metrics

Dataset	No Attention	Attention	Difference
Validation MSE (OVERALL)	1.402898	1.572188	+0.169290
Test MSE (OVERALL)	1.398279	1.566982	+0.168703
Train MSE (OVERALL)	1.417001	1.587753	+0.170752

Table 23: Comparison of Attention vs No Attention for OVERALL Metrics

8.10.1 Summary of Effectiveness of Attention on the Signage Dataset and Model

The results suggest that the impact of attention is relatively modest and may introduce noise rather than significant improvements.

From the MSE comparisons across validation, test, and training sets, the attention layer provides a slight refinement in performance for latitude (LAT) and longitude (LONG) predictions, particularly in the test set. For example, the test MSE for LAT decreases by approximately **0.009** when attention is included, and the test MSE for LONG decreases by **0.007**. These improvements, while present, are relatively small, indicating that the attention layer contributes only marginally to the model's performance

The overall MSE (combining LAT and LONG) shows a slight increase when attention is applied, suggesting that the attention layer might be introducing noise or overfitting to certain patterns in the training data. This is further supported by the observation that the training MSE for both LAT and LONG increases *slightly* with attention, indicating that the model may be learning spurious correlations from the text embeddings rather than meaningful geoinformative information

Once more the combination of the pre-learned embedding model and attention mechanism may lead to redundancy and overemphasis on features which should be *subtle* in their representation.

In summary, while the attention layer demonstrates a small refinement in certain metrics, its overall contribution to the model's performance is limited.

8.11 Clustering

Clustering is assessed for K = 5 through K = 40;

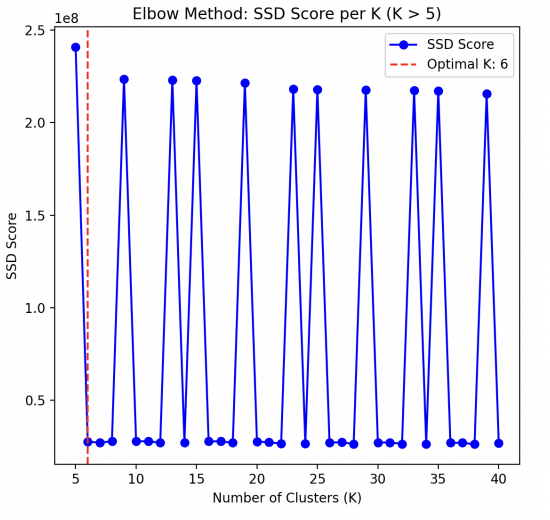


Figure 69: SSD Elbow Method (Optimal K)

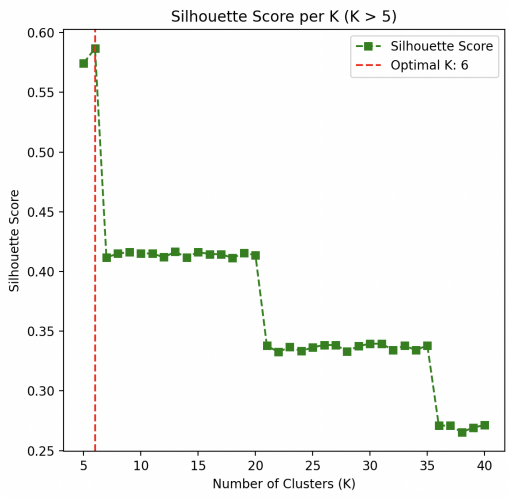


Figure 70: Silhouette Scores for All Test/-Train/Val

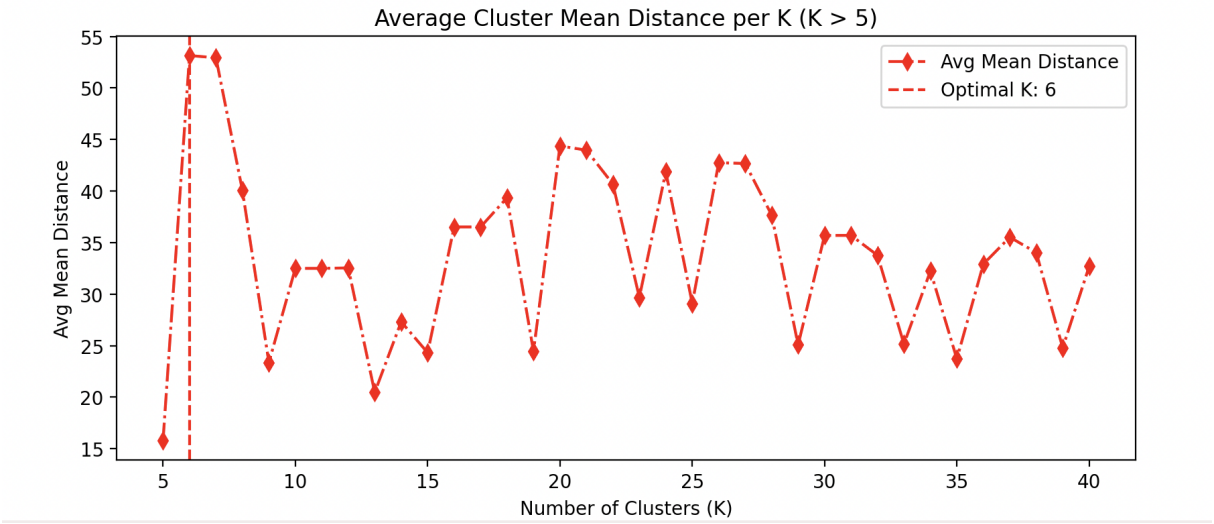


Figure 71: Average Cluster Mean Distance per K

Optimal K = 6

Cluster	Avg Latitude	Avg Longitude	Num Points	Mean Distance	Avg Geo Distance (km)
0	41.8926	-87.6488	2	50.5869	0.0016
1	41.8956	-87.6513	131815	18.6340	1.6317
2	41.9065	-87.6616	1	0.0	0.0
3	41.9089	-87.6482	1	0.0	0.0
4	41.8949	-87.6368	1	0.0	0.0
5	41.8943	-87.6503	18230	88.2470	1.5859

Table 24: Cluster information for Clustering at **K = 6**; poor separability and delineation

The 6 clusters are not particularly well separated and the two *main* clusters are geographically close to each other *in the average LAT / LON* and thusly are not useful. The Low average distance is a product of the source dataset density, there are a signfiicant number of outliers; however the concentration of

source images *pulls down* this distance metric. *By the nature of the dataset distribution, a relatively low Geo Distance (KM) is unavoidable*

8.11.1 Clustering Contexts

The data at hand is not sufficiently separable for effective clustering. Our analysis reveals that the optimal separation occurs at just $K = 6$, which, given the large search area and the complexity of the dataset, is insufficient for the intended use case. This suggests that the underlying structure of the data does not lend itself well to clustering, as the natural groupings are either too subtle or too overlapping to be clearly distinguished. Noting that each record consists of the concatenated *Text Embeddings* and *Object Embeddings*, it is a surprising result that there is insufficient separability for any meaningful cluster analysis

The optimal number of clusters, K, was determined using the **Elbow Method**, as illustrated in Figure 69. The Elbow Method evaluates the sum of squared distances (SSD) between data points and their assigned cluster centroids for different values of K. The "elbow" point, where the rate of decrease in SSD sharply changes, is typically chosen as the optimal number of clusters. However, in our case, the elbow is not sharply defined, further indicating that the data lacks clear separability.

Zig-Zagging Trend in SSD The **zig-zagging** pattern, yet overall *slight* downward trend in SSD scores across different values of K, indicates a poor fit of the clustering model to the data. Typically, a sudden *spike* in SSD suggests overfitting, where the model captures noise rather than the underlying structure. However, the results show alternating spikes and drops in SSD, which *complicates* the interpretation. This irregular pattern may be attributed to the inherent noise and lack of distinct clusters in the data, making it difficult for the clustering algorithm to converge to a stable solution.

Silhouette Scores The **Silhouette scores** for each cluster (averaged across points) reinforce that there is insufficient separability to define distinct clusters within the dataset; We see a sharp drop in separability and some stable Plateaus; however the further drops indicate over fitting.

Average Cluster Distance The **average cluster distance** further supports the idea that the data lacks clear separability. The zig-zagging pattern in the average cluster distances reinforces the instability observed in the SSD and Silhouette scores. Peaks in the average cluster distance correspond to overfitting, where the algorithm creates clusters that are once more, not well-separated or meaningful.

This instability is likely exacerbated by the high dimensionality of the embeddings, as the distances between points in high-dimensional spaces tend to converge, making it difficult to distinguish between clusters, as the object embedding is particularly sparse.

K-mean clustering does not sufficiently generate distinct clusters for this dataset without feature selection or dimensionality reduction

8.11.2 Future Work in Refining Clustering

Given the *generally poor* performance of k-means clustering here; several approaches could be implemented to improve separability and robustness:

- **Dimensionality Reduction:** Techniques such as Principal Component Analysis *(PCA)*, *t-SNE,* or *UMAP* could be applied to reduce the feature space while preserving the structure of the data. This *may* enhance clustering effectiveness by focusing on the most informative dimensions in the entire embedding (section of the) dataset; however this approach should be considered with caution as the text embeddings and the object one hot embeds are potentially already at their minimum useful dimensionality without losing assocation information.
- **The curse of dimensionality** could also be mitgated by encoding the text vector embeddings to a space less than the **128**d in **??**; however this may have *negative* performance outcomes for the other ML methods.

- **Alternative Clustering Methods:** Consider alternate methods for clustering like DBSCAN or Gaussian Mixture Models (GMM) may be beneficial, as the number of clusters is learnt and these better handle noise / irregular cluster shape (in the similarity space, not the geo space).
- **Embedding Optimization:** Using **more compact and dense** representations of object embeddings than the one hot embedding, such as fine-tuned embeddings, could mitigate sparsity issues.
- **Cluster Stability Analysis:** Running clustering multiple times with different initializations and evaluating the consistency of results could provide insights into the performance of the model and the *robustness* of the model

9 Results Interpretation

It is particularly clear from 57 that attention causes some significant centralization of predictions in the context of our implementation;

9.1 The Full Image Pass - Overfitting to the Data Distribution

9.1.1 Mismatch between Prediction Visualization and Results; Overfitting to Dense Areas

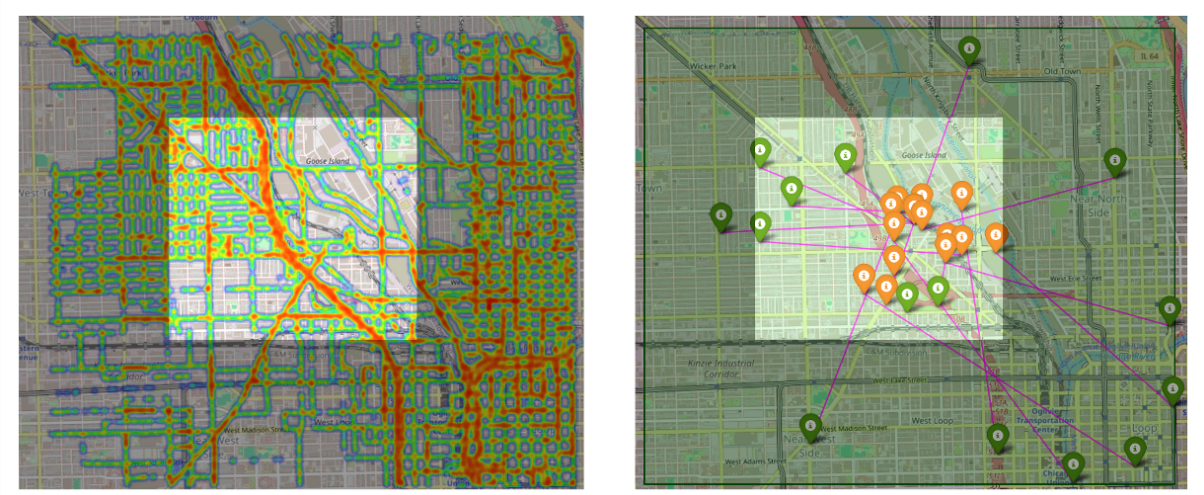


Figure 72: Density (left) of images in the dataset; Predictions (right) of the neural network model for the full image (green = actual, orange = prediction).

The observed concentration of predictions around the most densely populated training areas suggests that the model is overfitting to these particular sub-areas of the dataset.

We expect the MSE should equate to some *10000 - 12000 m* average delta, yet the distribution of predictive labels indicate that this is not the case. We recall that the *Average MSE* for the *Attentionless* model is some 0.722971; denormalized:

9.1.2 Batch Size *Strongly* impacts MSE

The inconsistency between the finalized model's average Mean Squared Error (MSE) and the visualizations in Figure 53 highlights a particulary clear issue: the model's predictions are not uniformly accurate across the entire *Chicago* Dataset. Although MSE suggests an average prediction error of approximately 1.2–1.3 kilometres, the sample points in Figure 53 reveal that this "good" MSE value may be deceptive. Rather than learning the intended local features for geolocalization, the model appears to have **overfitted** to the underlying density distribution of the dataset. By leveraging the domain size and spatial density, the model minimizes error without truly generalizing to less frequented areas. In other words, it adopts a strategy of *'the middle is enough for most data'* to optimize the loss function over batches.

This observation suggests that **batch size** is a critical hyperparameter, particularly given the centralised distribution of the data set. Since the training method relies on *batched gradient descent*, the distribution of the data likely causes all batches to converge to similar loss values beyond a certain point. (As more records are added to the batch, the distribution more closely resembles the distribution of the overall dataset / train set) As a result, the model fails to capture the granular information necessary for meaningful updates, especially in less dense regions.

Distilled; the concentration of points about the center of the *Kennedy Expressway* in our dataset is mirrored in any average 512 sampled batch and the entire gradient update step is always *approximately* the same. The loss here is always the same and the model fails to learn meaningful patterns, if any patterns at all.

A higher intitial learning rate, lower batch size and better gradient management to address the similarities in the final layer distributions (their shape not their correlation) may be needed in *future work*

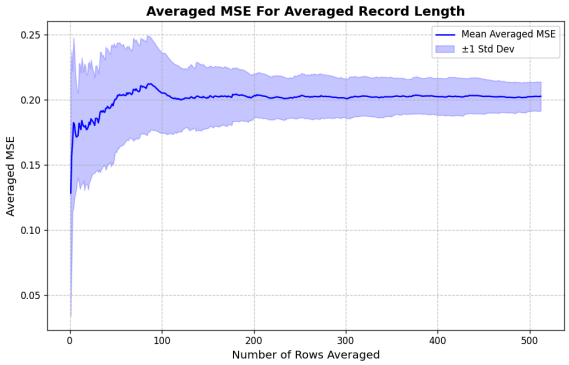


Figure 73: Average TEST MSE on EPOCH = 50 by Batch Size

This behavior is empirically validated in Figure 73, where the MSE stabilizes and becomes uniform as the batch size approaches *100*. At this point, **the batch size effectively erases the granular information required for the model to make meaningful updates**, leading to suboptimal performance in less represented areas.

Furthermore, the distribution of predicted labels indicates that the model performs significantly better in regions with higher image density, as evidenced by the heatmap in **??**. This suggests that the model has overfitted to areas with abundant training data, while underperforming in regions with sparse representation. This discrepancy underscores the importance of addressing data imbalance and refining the training process to ensure robust generalization across the entire dataset.

9.2 Future Work and Approaches to Mitigating Density Issues

9.3 Subsampling

Subsampling the data to a consistent density relative to road coverage—i.e., using shapefiles to cross-sample candidate locations and the source dataset points through the Mapillary API—may prove imprudent depending on the expected density. A core tenet and requirement for GeoInformers is that there must be some overlap between images. While unilaterally discrete GeoInformative patterns remain viable for informing a neural network (NN) or machine learning (ML) model's convergence, studies such as PlaNet[49], GeoReasoner[**Georeasoner**], and PiGeon[17] demonstrate that sequences serve as valuable combinatorial informers for GeoInformative pattern recognition. Consequently, future development is likely best directed toward encoding information from sequences rather than sampling it away.

9.3.1 Upsampling through Swatches or Muli Datasets

With this in mind, up-sampling presents another option for addressing density concerns. A particular approach could involve defining a fixed-size swatch of, say, *640x640px*, and subsampling higher-resolution

dataset members by passing a sliding window across the images. A simpler and potentially more effective method would be to combine data from multiple datasets and services (i.e. google streetview, mappilary and other open source repositories). This has the natural advantage of diminishing *time invariance* issues, which are not explicitly considered in this proposal or method.

9.3.2 Replacing Regression with Cells could Improve Accuracy

A fundamental shift in approach could involve forgoing regression in favor of a *GeoCell* method similar to PlaNet[49]. However, at a 20km^2 scale, the tradeoff between accuracy within the candidate area (which is arguably a reasonably sized geocell in its own right) and the imprecision of geocells remains domain- and application-dependent. Additionally, the inseparability of text and (albeit one-hot) object embedding vectors suggests that this approach may not be viable within the current implementation.

Another consideration is integrating such a model with a the sign model and employing transfer learning to develop a combined model. This could be effective, provided there is sufficient GeoInformative material within the sign location data, as per **??**, 5 signs has proven to be a minimum for a viable (usefully precise) prediction in our application. Processing sequences in this manner could preserve relative geo-distribution and geo-relative information within the signage dataset as well. However, this would require a significant architectural rewrite and reconsideration of the model design.

The **relative density** of the source images may well be a signficant contributing factor; wherein the model has not learnt from the less frequent sections of the dataset.

9.4 Convolutional Neural Networks in Context for Signage

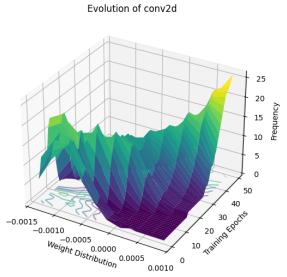


Figure 74: Weight Distribution over Epochs of CNN layer 1

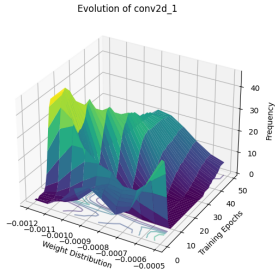


Figure 75: Weight Distribution over Epochs of CNN layer 2

Examining the average weights of the two CNN[25] layers over epochs, we observe a clear trend toward overfitting as training progresses toward 50 epochs. This suggests that the model is increasingly zeroing in on some specific feature of the CSV embedding, likely at the cost of generalization. Identifying which feature is driving this behavior is particularly challenging without further analysis of feature importance or weight attribution methods.

Understanding the extent of this overfitting requires examining activation patterns, weight distributions, and layer-wise relevance propagation. With the scope of reporting being the averaged weights at each epoch, we simply lack the information to make a precise determiniation.

The secondary dense layer shows a more normal distribution towards epochs, early *overfitting* is mitigated over time as we reach a flattened distribution towards epoch 50.

9.5 Over-fitting and efficacy of Learned Features in the Signage Model

Examining the average weights of the two CNN[25] layers over epochs, we observe a clear trend toward overfitting as training progresses toward 50 epochs on the first CNN layer. This suggests that the model is increasingly zeroing in on specific features of the CSV embedding, likely at the cost of generalization.

Identifying which feature is driving this behavior is particularly challenging without further analysis of feature importance or weight attribution methods, as we are limited by the reporting to average weights alone.

To better understand the nature and extent of this overfitting, *future work* should involve examining activation patterns, weight distributions, and layer-wise relevance propagation. Since our currnt analysis is based solely on averaged weights per epoch, we lack the granularity needed to determine whether the observed trends are due to true feature learning or simply overfitting to the distribution of the training data.

In contrast, the secondary CNN layer shows a more normal weight distribution as training progresses. While early epochs suggest some degree of overfitting, this effect diminishes over time, leading to a more balanced, flattened distribution by epoch 50; indicating a more generalized fit.

9.5.1 Future Work, specifically in Reporting and Analysis

To fully comprehend the scope of the model's error and better assess overfitting, future reporting should include the recording of gradient propagation, relevance propagation, and other relevant metrics over epochs. Incorporating methods like those presented in WeightScale: Interpreting Weight Change in Neural Networks from Argwal et al[1], which emphasize measuring relative weight change on a per-layer basis and dynamically aggregating trends through dimensionality reduction and clustering, notably trivializing the *implementation* of generating these reports, can provide a clearer picture of how the model learns and generalizes beyond current weight distribution reporting; lending more insight to precisely which features are being learnt.

9.6 Verification of Fit through Weights Analysis for the Overall Model

While the MSE values for the training, validation, and test datasets show no significant differences, which is usually a strong indicator of good generalization; it is possible that the concentration of data-points is learnt moreso.

Weight histograms provide a useful tool for diagnosing the behavior of our neural network layers through training; undersatnding convergence. For instance, histograms that show weights clustering around extreme values (e.g., very large or very small magnitudes) may indicate overfitting, as the model is likely memorizing specific patterns in the training data rather than generalizing well.C onversely, a *healthy* distribution of weights, with values spread across a reasonable range, would suggest that the model is learning diverse features and relationships across the text embeddings, object embeddings and the colourspace and maintaining a balanced representation of the entire candidate locale.

9.6.1 Redundancy

Our model is rather likely *not* learning features of the dataset effectively, but rather learning to replicate the distribution of the source dataset. To understand whether the layers are playing *uniquie* roles, we consdier their correalation *(per Weight Histogram Distribution Per Layer Average Per Epoch)*

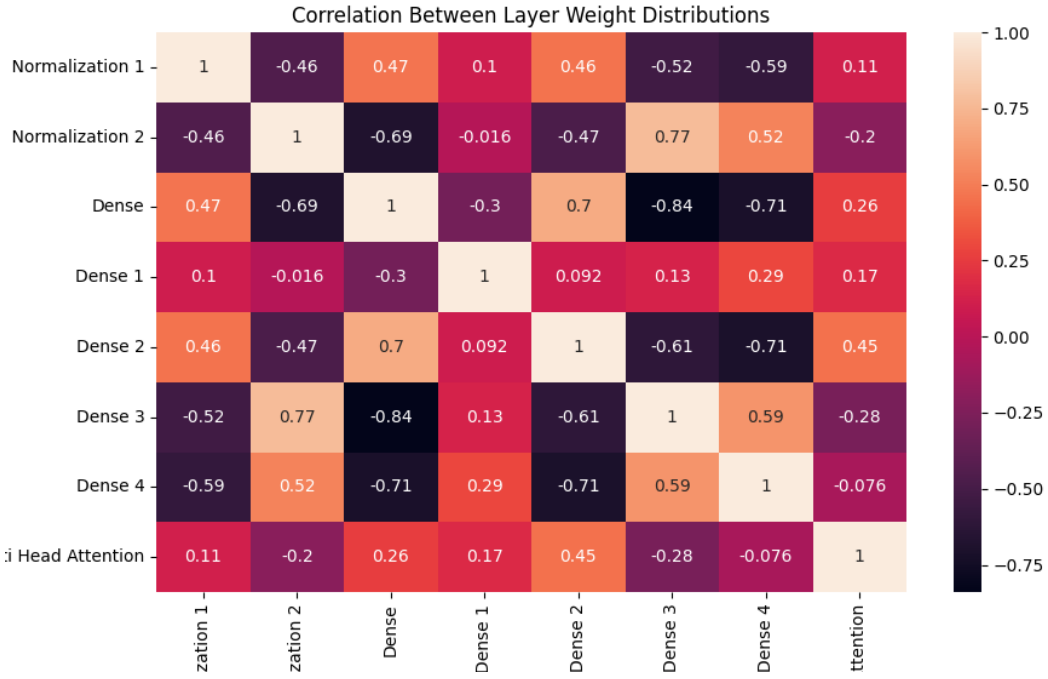


Figure 76: Correlation of Layers for Attended Model

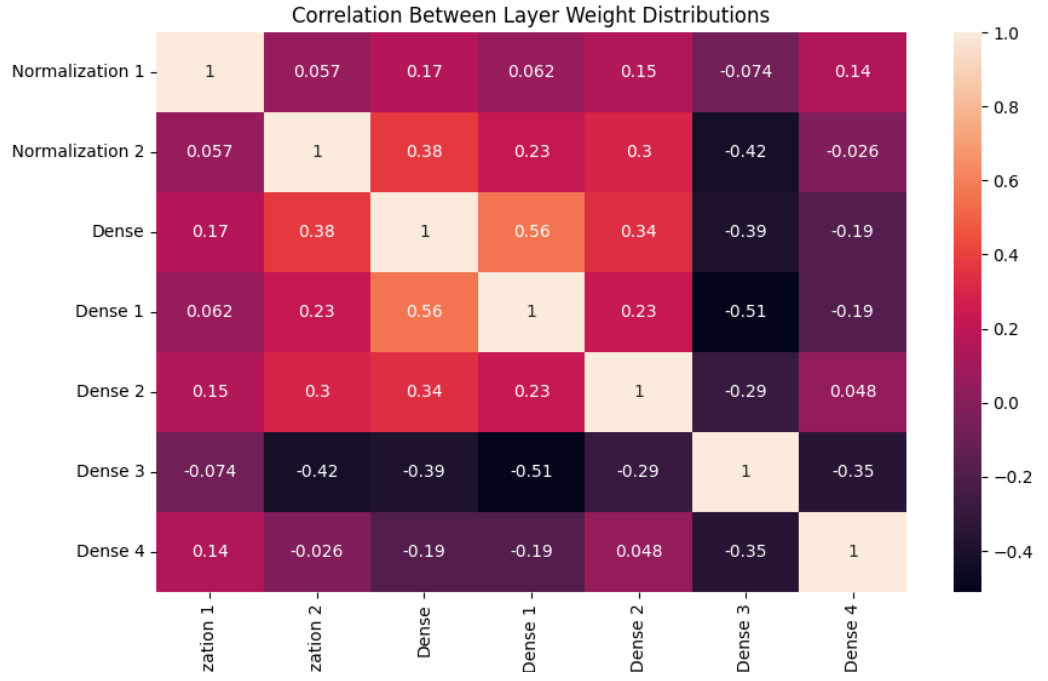


Figure 77: Correlation of Layers for unattended Model

The absence of high correlation between layers is a positive sign, as it indicates that the model does not show redundancy; each layer learns distinct aspects of the dataset. However, the fact that the model still **overfits** to the high-density regions suggests that while the layers are not redundant, they may not be learning local features well, if at all.

The distribution of weights observed in Figures 76 and 77 shows that the addition of attention removes any *structure* (or observable gradient) in the correlation maps. This suggests that attention may be impeding the model's ability to learn effectively. By removing the natural progression of feature extraction between layers, attention might be introducing noise or disrupting the hierarchical flow of information, making it harder for the model to capture meaningful relationships. As a result, the model could struggle to develop coherent internal representations, ultimately limiting its performance.

By removing the natural progression of feature extraction between layers, attention might be introducing noise or disrupting the hierarchical flow of information, making it harder for the model to capture meaningful relationships. As a result, the model could struggle to develop coherent internal representations, ultimately limiting its performance.

We note that attention is applied post embedding and that the inclusion could potentially even prove redundant in the face of learned embeddings and the pretrained sBert[35] embedding model.

9.6.2 The Overall Model learns to *center* around the most concentrated point in the dataset

We expect the weight distributions to initially follow a normal distribution, centered around their initialization values, and to gradually flatten over the course of training as the model learns. This flattening reflects the diversification of weights as the model adapts to capture a broader range of features and patterns in the data.

For the purpose of Analysis, we now consider only the Non-Attention Model

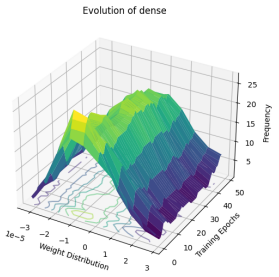


Figure 78: Weights change over Epochs : Dense 1

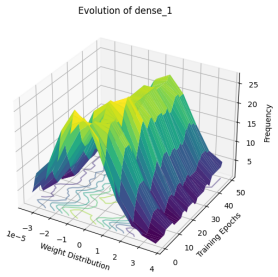


Figure 79: Weights change over Epochs : Dense 2

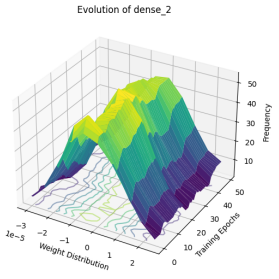


Figure 80: Weights change over Epochs : Dense 3

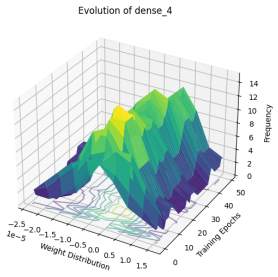


Figure 81: 4D Plot of Dense Model

We observe the expected behavior across all of the dense layers, where the weight distributions initially follow a normal distribution centered around their initialization values. Over the course of training, the distributions gradually flatten, particularly after approximately 25 epochs. However, this flattening is subtle, and we observe almost no significant change in the weights beyond this point.

This suggests that the model has reached a point of stabilization, where further weight adjustments are minimal. After around 25 epochs, the model appears to have converged to a stable set of weights that capture the underlying patterns in the data *as best they will*. However, the subtle flattening and the lack of further significant changes indicate that the model has *likely* failed to fully learn and converge to the true distribution of the data. It suggests that the model has not fully captured the complex patterns and relationships present, and may be struggling to learn the complete distribution. This could imply that the training process has plateaued prematurely, and additional training or adjustments may be needed for the model to effectively capture the entire distribution and improve performance.

Alternatively, the implication is that this is 'as much as the model can feasibly learn' from the dataset. In other words, the model has reached the limit of what it can extract from the available data, and further training is unlikely to yield significant improvements without additional data or adjustments to the model's architecture.

The most likely of these candidates is that the model has learned as much as it can feasibly learn from the dataset as per fig42 which shows a convergence of Loss across all three sets.

Once more, inseparability of the data and batch size are the core contributors in this result

9.6.3 Batch Normalization Layers over Epochs

Note that the final dense layers in the Network was punctuated by Batch Normalziation layers

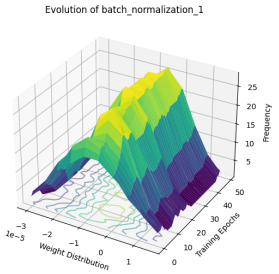


Figure 82: Batch Normalization 1 - 3D Plot

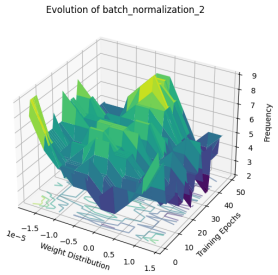


Figure 83: Batch Normalization 2 - 3D Plot

Notably, the first Batch Normalization layer flattens after around 25 epochs, exhibiting a strong concentration around a center (which is not necessarily zero in the generated graphics) that is shifted slightly. This concentration appears around -1×10^{-5} in the figure below, indicating the *stable state* for this layer. It suggests that the first layer has effectively learned to normalize its inputs and stabilize the training process (see Figure 84).

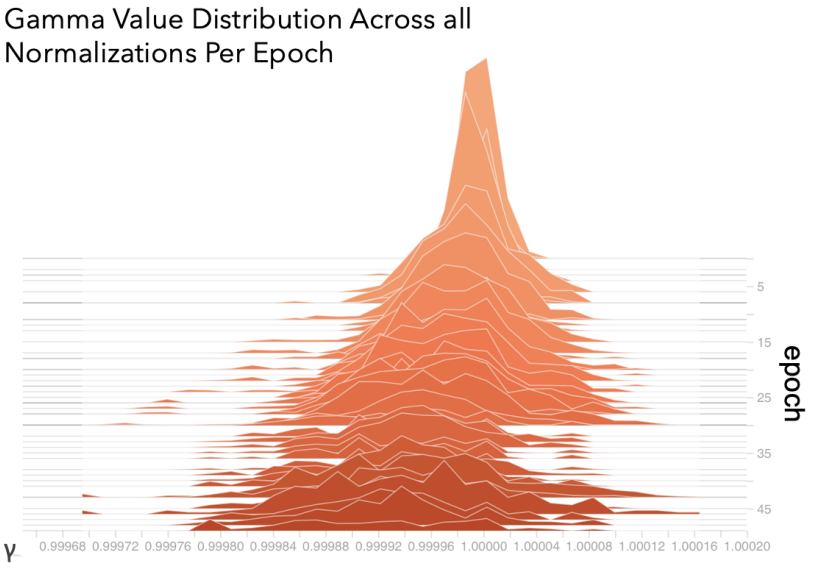


Figure 84: All Batch Norm Gamma Values

However, the secondary Batch Normalization layer does not conform to the same flattening behavior. Instead, it shows values concentrated around a center forming after 40 epochs, without the same degree of stabilization. This suggests that while the first layer has reached a stable equilibrium, the secondary layer may still be adapting or struggling to normalize its inputs as effectively.

The difference in behavior between the two layers could indicate a few possibilities. First, the first layer may be processing simpler, lower-level features that stabilize more quickly, while the secondary layer deals with more complex, higher-level features that require longer to converge. This could reflect the intended behavior of the architecture design.

10 Future work and Shortcomings of the Implementation

10.0.1 Overfitting of the Overall Pass

Through weights analysis we confirm that the overall model has simply learned the distribution and no significant underlying features in the core dataset.

10.1 Clustering

The results of clustering reveal that the data from the text embeddings are insufficiently separable in their high dimensional embedding when concatenated to the OneHot vector. A better future approach is dimensionaltiy reduction; either through dim. reduction techniques or by encoding to a smaller space in the first instance. The relationships between text from the embeddings are still a valid feature to include for clustering as underlying semantic relationships are, essentially, the intended leraning of the model. (i.e. derived clusters of locations for *important* words such as place names).

10.2 Text Embeddings with *sBERT*[35] and Attention Could Be *Too Much*

Attention is applied to all models that inherit embeddings from the pretrained *sBERT* model. *sBERT* is designed to represent sentences by encoding the relationships between the text and sentences, capturing semantic meaning and contextual nuances. However, applying attention to these embeddings after they have already been processed by *sBERT* might introduce redundancy.

The reason for this is that *sBERT* has already learned a sophisticated representation of the relationships between words and sentences. Adding another layer of attention on top could cause the model to

focus again on these relationships, rather than enhancing its understanding of the underlying text itself. Essentially, the model might be **overemphasizing** relationships that are already well-represented, while not fully utilizing the raw text itself in a meaningful way.

10.3 Attention Trampling on Learned Embeddings

Similarly, embeddings for the *Overall Image Pass Model* are translated into learned embeddings, which capture meaningful representations of the data.

However, applying attention after the learned embeddings have already been generated can lead to redundancy. The embeddings, are designed to encapsulate relationships and patterns within the dataset amd may already capture the essential structure and context. Adding an attention mechanism could cause the model to revisit these learned relationships in a way that doesn't necessarily add value; the attention mechanism might just **re-emphasize what has already been learned** by the embedding layer, essentially "trampling" over the existing structure. This could reduce the effectiveness of the attention mechanism, preventing it from providing additional insights into the data and once more hindering the model's ability to learn further.

10.4 Sequences to Address Single Image Subsets

As we've seen in earlier sections, such as in **??**, the model is expected to learn a vector representing specific objects or features present in an image. For example, the model might generate a vector like:

```
[The, Cell, Road, ......]
+
[Traffic Light * 4, Hydrant * 1, ....]
```

However, a detection vector for the same location could differ significantly if the camera is rotated. In this case, the diversity of vectors may become too small to effectively capture the variety of visual features across different viewpoints.

To address this issue, the approach used in PlaNet [49] mitigates the problem of single image subsetting by leveraging sequences of images. By incorporating Long Short-Term Memory (LSTM) networks, PlaNet captures the temporal dynamics of features across sequences of frames, rather than relying solely on a single still image. This allows the model to better respond to features whose patterns may only be partially captured in a single image, improving its ability to generalize across different perspectives.

10.4.1 Does the Object Embeddings Vector Provide Useful Context?

We produced an embedding for the object vectors, of note, from the entire 151,510image dataset, **produced just 42** unique embeddings. Discarding the all zero vector (as this includes vectors wherein the tfrecord is padded with zeroes for processing) this is not particularly unexpected given the domain and, relative, scope of the YOLO models / classes.

Index	Vector	Count
0	(0, 0, 0, 0, 0, 0, 0)	15618
1	(2, 0, 0, 0, 0, 0, 0, 0, 0)	13955
2	(2, 2, 0, 0, 0, 0, 0, 0, 0)	7951
..	.....	..
40	(9, 0, 0, 0, 0, 0, 0, 0)	38
41	(4, 3, 3, 0, 0, 0, 0, 0, 0)	38
42	(2, 2, 2, 2, 2, 2, 2, 0, 0)	38

Table 25: Object Embedding Vectors and their Counts

10.5 Data Handling Compromises

Hardware constraints necessitate certain shortcuts in implementation. While our embedding data is stored in 512-length *batches* (as TensorFlow Record Files), handling these files poses memory constraint

issues. Standard, inbuilt methods for the *TensorFlow* library were bypassed in favor of custom implementations, which negatively impact the performance of the Neural Network.

Our data was procured *as and when* from the Mapillary Service API, meaning *Sequences* impact the order in which images are ingested. Specifically, if a user uploaded stills from driving, they are ingested in order. To prevent our model from learning unintended features or *focusing* on specific features (columns in our embedding and relationships), we should mitigate these relationships. The prudent approach is to shuffle each row; however, the memory footprint is untenable. Instead, we shuffle the order in which we query files and randomly shuffle the tensor record files when they are loaded.

10.6 Time Invariance and Geolocalization as a 'Point in Time'

None of the models reviewed in the *Background and Prior Work* section explicitly address strategies for mitigating time variance. Similarly, our model does not attempt to account for temporal changes beyond training on the full dataset, which includes data points containing geo-informers that may have changed or been removed over time.

However, time invariance may not necessarily be *critical* if the model learns underlying patterns rather than relying too heavily on any single geo-informer or subset of geo-informers. Some geo-informers remain useful even after their physical removal. For instance, the name of a suburb on a sign might still provide valuable contextual information, reinforcing broader patterns that the model has learned. In such cases, the persistence of learned relationships, rather than the presence of specific geo-informers, may be sufficient for accurate geolocalization.

10.7 Scalability and features across images

The resultant model can scale to increasing search areas; however, it relies strongly on local features appearing across multiple images or appearing with particular frequency *(i.e., advertising naming suburbs)*. Consequently, our parameter d (the relative distance between each training point) must be sufficiently small to capture this information. If the distance d is too large, the model may fail to learn and generalize these local features effectively, leading to poor prediction.

10.8 Local Feature Requirements

Similarly, the model architeture may generalize poorly to sparse areas, where there are few unique local features, or they are absent entirely. The lack of extractable text or objects may hinder the model's ability to make useful predictions. Similarly, in areas with few training points, the model may overfit to specific details and not well generalize.

10.9 Stacking Discrete Models

The complexity of integrating multiple models (YOLOv11, docTR, attention layers) increases the computational load and resource requirements, which may not be feasible for real-time applications or large-scale deployments. Moreover, the combination of these models can introduce additional noise and variability, which might affect the overall accuracy and reliability of the predictions.

10.10 Transfer Learning Instead of Averaging

A core benefit of trained multi-layered neural networks is that they can be modified and adapted with relative ease. Our model initially proposes simply averaging the latitude/longitude outputs for the cluster centroids, aggregate sign centroids, and the predicted latitude and longitude of the overall image pass. However, a more prudent approach is almost certainly freezing the models—excluding the output layers—removing them, and replacing them with some mix of dense or attention layers. These layers could learn aggregate features to better capture patterns across the dataset.

This approach would likely benefit from incorporating spatial data for signs, as the only relative information at ingest was the true label, despite all originating from the same locale.

10.11 Spatial Information and Sequences for Signage Spatial Information Preservation

Sequences and LSTM methods are employed by PlaNet[49] to address the distribution of geoinformers among images by leveraging sequence learning. Our model *forgets* the semantic spatial relationship between signs at ingest and instead relies on learned patterns to *reconstruct* these relationships in the model weights.

An alternative approach that embeds signs as an aggregate or groups them from a collection of source images within a *cell* is likely to produce more descriptive features and richer patterns for training the neural network models, ultimately leading to more accurate predictions. Precision is hampered by the current approach, as our model was trained with the implicit expectation that the highest precision would match the granularity of the source dataset.

11 Ethics and Data Soverignty In Context

All images used in this proposal are publicly available under a license. The use of geolocation data and street-level imagery to infer new information raises significant ethical concerns, particularly regarding data sovereignty, ownership, and privacy. Geospatial data can be highly sensitive, especially when it pertains to specific locations or individual movements, potentially infringing on privacy rights if not handled responsibly.

Broader soveriegnty concerns arise when the communities from which the data originates have limited input into its collection, use, storage, or the conclusions drawn from it. While the dataset in this proposal is based on urban imagery, street-level imagery is widely accessible, making it essential to address concerns related to the model's temporal invariance, dataset composition, and potential misuse as a sufficiently advanced tool.

A key factor in the model's efficacy is the geospatial density of GeoInformers, which in this context includes infrastructure, signage, and color profiles. Initially, Detroit was considered as a candidate location due to its extensive coverage in Mapillary's dataset. However, a significant portion of the available imagery consisted of geoinformation-sparse areas, limiting its suitability for training and evaluation.

A final note, it is important to emphasize that while this model aims to improve geolocation accuracy, its predictions should not be solely relied upon for critical applications. Factors such as dataset biases, temporal variations, and inherent limitations in street-level imagery can impact performance, necessitating careful interpretation of results.

The model is not time invariant.

12 Comparison to Other Work

None of the discussed models—PlaNet[49], streetCLIP[16], PiGeon[17], or others—are designed to geolocate at the scale intended by this project.

All of these models demonstrate *better* efficacy when compared to the source dataset distributions. As such, direct quantitative comparisons are largely meaningless.

This implementation does not resolve the issue of weight distribution for learned GeoInformers[49]. While the attention layers did reduce MSE/Loss across all models, the distribution issue overshadows these improvements, ultimately invalidating the comparison.

We used explicit GeoInformers, similar to the implicit-explicit geoinformers in streetCLIP[16] and GeoReasoner[51]. However, we found that defining a distinct, evaluable pattern of informers is particularly challenging. In practice, despite an initially large set of local feature embeddings, the total pool of unique object-type and frequency combinations amounted to just 42.

The use of CLIP[31]-based models for determining geo-informative features is *likely* a better approach. While a direct comparison is beyond the scope of this study, CLIP's ability to encode features dynamically—rather than relying solely on counts and class labels—suggests a more robust methodology. Additionally, the increasing ubiquity and decreasing computational footprint of these models, particularly in contrast to large language models, raises questions about whether further development of distinct geo-informers is a *fool's errand*.

A major contributing factor to the success of this project was the density of the source data. We touched on alternative approaches in the results analysis 9.2.

13 Conclusion

This paper proposed *Multi Neighbourhood scale* geolocalization of imagery from local image features by extractign local iamge features and training three distinct neural networks and machine learning models and averaging their rresults.

Attention improved the average precision of the two applicable models by an order of magnitude, yet ultimately did not discern the results from simply mirroring the distributioon of any given batch.

The applicaiton of Clustering revealed tha the core data embeddings are insufficiently separable based purely on their distance in a higher dimensional embedding. This supports the idea that ther ear einsufficient GeoInformative patterns to learn.

We did successfully show that sign geolocation can usefull triangulate signs beyond the scope of the distribution; however such an approach is inherently limited ot be *close to* the most dense areas in it's predicitons; however within this space, the predictions for the signage network are usefully descrtiptibve.

The model, with refinements, with approaches to data density management, ingestion and potentially from applying learning from sequences ala PlaNet[49] still shows *promise* in terms of generating useful GeoLocalization.

References

[1] Ayush Manish Agrawal et al. "WeightScale: Interpreting Weight Change in Neural Networks". In: *arXiv preprint arXiv:2107.07005* (2021).

[2] Authors. "Your new reference title". In: *ArXiv* (2024). URL: https://arxiv.org/html/2407.02988v1.

[3] Alexey Bochkovskiy, Chien-Yao Wang, and Hong-Yuan Mark Liao. "YOLOv4: Optimal speed and accuracy of object detection". In: *arXiv preprint arXiv:2004.10934* (2020). URL: https://arxiv.org/abs/2004.10934.

[4] Andrew Burnes. *DLSS 4 With Multi Frame Generation & Enhancements For All DLSS Technologies Available Now In Over 75 Games and Apps*. Accessed: 2025-02-07. Jan. 2025. URL: https://www.nvidia.com/en-us/geforce/news/dlss-4-multi-frame-generation-out-now/.

[5] Nicolas Carion et al. "End-to-End Object Detection with Transformers". In: *arXiv preprint arXiv:2005.12872* (2020).

[6] Chicago Metropolitan Agency for Planning. *CMAP Community Data Snapshot — Chicago*. 2024. URL: https://www.cmap.illinois.gov/wp-content/uploads/dlm_uploads/Chicago.pdf.

[7] Google Cloud. *Google Cloud Vision*. https://cloud.google.com/vision. 2020.

[8] DeepSeek. *DeepSeek-V3 Redefines LLM Performance and Cost Efficiency*. https://www.deeplearning.ai/the-batch/deepseek-v3-redefines-llm-performance-and-cost-efficiency/. 2025.

[9] Jacob Devlin et al. "BERT: Pre-training of deep bidirectional transformers for language understanding". In: *Proceedings of NAACL-HLT 2019*. 2018. URL: https://doi.org/10.48550/arXiv.1810.04805.

[10] Carl Doersch et al. "What makes Paris look like Paris?" In: *ACM Transactions on Graphics (TOG)*. Vol. 31. 4. ACM. 2012, pp. 101–108.

[11] dwz. *zebracrossing Dataset*. https://universe.roboflow.com/dwz/zebracrossing-g9vao-y5kkw. Open Source Dataset. visited on 2025-01-15. Dec. 2024. URL: https://universe.roboflow.com/dwz/zebracrossing-g9vao-y5kkw.

[12] Geekflare. *Best GeoGuessr Tips, Tricks to go from Beginner to Champion*. 2023. URL: https://geekflare.com/consumer-tech/geoguessr-tips/.

[13] Google. *S2 Geometry Library*. Accessed: 2024-11-17. 2023. URL: https://github.com/google/s2geometry.

[14] *Google Street View API*. Comprehensive data source for geolocation. URL: https://developers.google.com/maps/documentation/streetview.

[15] Priya Goyal et al. "Accurate, Large Minibatch SGD: Training ImageNet in 1 Hour". In: *arXiv preprint arXiv:1706.02677* (2017).

[16] Lukas Haas, Silas Alberti, and Michal Skreta. *Learning Generalized Zero-Shot Learners for Open-Domain Image Geolocalization*. 2023. arXiv: 2302.00275 [cs.CV].

[17] Lukas Haas et al. "PIGEON: Predicting Image Geolocations". In: *arXiv preprint arXiv:2307.05845* (2024).

[18] Kaiming He et al. "Deep Residual Learning for Image Recognition". In: *arXiv preprint arXiv:1512.03385* (2015).

[19] Kaiming He et al. "Deep Residual Learning for Image Recognition". In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)* (2016).

[20] Sepp Hochreiter and Jürgen Schmidhuber. "Long Short-Term Memory". In: *Neural Computation* 9.8 (1997), pp. 1735–1780. DOI: 10.1162/neco.1997.9.8.1735.

[21] Marcie Hogan. *Detroit: the Most Covered City on Mapillary*. https://blog.mapillary.com/update/2024/11/18/detroit-the-most-covered-city-on-mapillary. Accessed: 2024-12-29. Nov. 2024.

[22] JaidedAI. *EasyOCR*. https://github.com/JaidedAI/EasyOCR. 2020.

[23] Glenn Jocher. *YOLOv5*. 2020. URL: https://ultralytics.com/yolov5.

[24] street lamps. *street lamps Dataset*. https://universe.roboflow.com/street-lamps/street-lamps-dpeqc. Open Source Dataset. visited on 2025-01-15. Dec. 2023. URL: https://universe.roboflow.com/street-lamps/street-lamps-dpeqc.

[25] Yann LeCun et al. "Gradient-based learning applied to document recognition". In: *Proceedings of the IEEE* 86.11 (1998), pp. 2278–2324.

[26] Tsung-Yi Lin et al. "Microsoft COCO: Common Objects in Context". In: *arXiv preprint arXiv:1405.0312* (2014). URL: https://arxiv.org/abs/1405.0312.

[27] Tsung-Yi Lin et al. "Microsoft COCO: Common Objects in Context". In: *arXiv preprint arXiv:1405.0312* (2014). URL: https://arxiv.org/abs/1405.0312.

[28] Mapillary. *Mapillary Coverage Analysis*. 2024. URL: https://colab.research.google.com/github/GIScience/sotm-2024-ohsome-data-insights-workshop/blob/main/book/03_mapillary_data_analysis.ipynb.

[29] Jay Nwaobueze. *Mastering Optimizers with TensorFlow: A Deep Dive Into Efficient Model Training*. https://dev.to/jaynwabueze/mastering-optimizers-with-tensorflow-a-deep-dive-into-efficient-model-training-1k8p. Accessed: 2024-12-07. 2020.

[30] PaddlePaddle. *PaddleOCR*. https://github.com/PaddlePaddle/PaddleOCR. 2020.

[31] Alec Radford et al. *CLIP: Contrastive Language-Image Pretraining*. https://openai.com/index/clip/. Accessed: 2024-11-17.

[32] Joseph Redmon and Ali Farhadi. "YOLO9000: Better, faster, stronger". In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. 2017, pp. 7263–7271. URL: https://arxiv.org/abs/1612.08242.

[33] Joseph Redmon and Ali Farhadi. "YOLOv3: An incremental improvement". In: *arXiv preprint arXiv:1804.02767* (2018). URL: https://arxiv.org/abs/1804.02767.

[34] Joseph Redmon et al. "You only look once: Unified, real-time object detection". In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. 2016, pp. 779–788. URL: https://arxiv.org/abs/1506.02640.

[35] Nils Reimers and Iryna Gurevych. "Sentence-BERT: Sentence embeddings using siamese BERT-networks". In: *arXiv preprint arXiv:1908.10084* (2019). URL: https://doi.org/10.48550/arXiv.1908.10084.

[36] Roboflow. *Billboard Detection*. https://universe.roboflow.com/brodic/billboard-detection-o1tk5. Provided by a Roboflow user, License: CC BY 4.0. 2024.

[37] Roboflow. *Board Detection*. https://universe.roboflow.com/board-0m6pt/board-detection-uap8n. Provided by a Roboflow user, License: CC BY 4.0. 2024.

[38] Roboflow. *Signages*. https://universe.roboflow.com/signage-measurement/signages. Provided by a Roboflow user, License: Public Domain. 2024.

[39] C.C. Robusto. "The Cosine-Haversine formula". In: *The American Mathematical Monthly* 64.1 (1957), pp. 38–40.

[40] Amazon Web Services. *Amazon Textract*. https://aws.amazon.com/textract/. 2020.

[41] Nitish Srivastava et al. "Dropout: A Simple Way to Prevent Neural Networks from Overfitting". In: *Journal of Machine Learning Research* 15.56 (2014), pp. 1929–1958.

[42] Digital Synopsis. *Samsung's creative bus stop ad for the Galaxy Z Flip4 is brilliant*. Accessed: 2025-01-31. Aug. 2022. URL: https://digitalsynopsis.com/advertising/samsung-galaxy-z-flip4-bus-stop-ad/.

[43] Antonio Torralba and Alexei A Efros. "Unbiased look at dataset bias". In: *CVPR 2011*. IEEE. 2011, pp. 1521–1528.

[44] u/[USERNAME]. *When the chef marketing company bests on*. Accessed: 2025-01-31. May 2023. URL: https://old.reddit.com/r/ireland/comments/13zfr9l/when_the_chef_marketing_company_bests_on/.

[45] Ultralytics. *Ultralytics Documentation and Models*. https://ultralytics.com. Accessed: 2024-12-3. 2025.

[46] Ultralytics. *YOLOv11: State-of-the-Art Object Detection*. https://ultralytics.com/yolov11. Accessed: 2024-12-3. 2023.

[47] Ashish Vaswani et al. "Attention is all you need". In: *Advances in neural information processing systems*. 2017, pp. 5998–6008.

[48] VICE. *Meet the Geoguessr player who only needs .1 seconds to be right — Georainbolt Interview*. 2022. URL: https://www.youtube.com/watch?v=PeXvpxPj1mo.

[49] Tobias Weyand, Ilya Kostrikov, and James Philbin. "PlaNet - Photo Geolocation with Convolutional Neural Networks". In: *arXiv preprint arXiv:1602.05314*. arXiv. 2016.

[50] World Population Review. *Christchurch and Sydney Population Density*. 2025. URL: https://worldpopulationreview.com/world-cities/christchurch-population.

[51] Yibo Yan and Joey Lee. *GeoReasoner: Reasoning On Geospatially Grounded Context For Natural Language Understanding*. 2024. arXiv: 2408.11366 [cs.CL].