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

Proposal, no results. Current status reflected in timeline. Repo @ github.com/2of

Abstract

A method for determining GeoLocation of iamges from latent features by combining image and object extraction with colour space analysis, using attention layers and deep learning on consumer grade hardware

1 Overview

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

2 Introduction

Most image formats contain geolocation information in their metadata or headers. In cases where this information is absent, we propose that geolocation can be determined with useful accuracy from latent features derived directly from the image data itself. These features, referred to as Geo Informative features, cumulatively inform a deep learning model that determines latitude and longitude coordinates. Our data is sourced almost entirely from the Google Street View API [10].

The popular online video game GeoGuessr gamifies this process for humans. Players are presented with a randomized Google Street View location, and their performance is scored based on the topological distance between the location they select on a map and the actual location of capture [8]. Naturally, narrowing down an answer involves recursively shrinking a pool of candidate locations by assessing progressively finer image features. This approach translates well to a machine learning model. Some members of our Geo Informative Pattern are the most meaningful because they significantly reduce the candidate pool, or they are uniquely identifying (e.g., a sign reading "Carter's Road Grocer, Carterton's Top Grocer!" or a recognizable landmark like the Eiffel Tower). The most telling features are often derived from text, such as phone numbers and web domains.

Our approach leverages several advanced techniques: object detection to identify relevant geoinformers, text extraction to capture informative text features, and colour extraction to analyze the colour distribution in images. We employ the YOLO model for efficient real-time object detection, docTR for robust text extraction, and custom algorithms for HSV colour histogram analysis. These extracted features form a combined embedding. The embedding is then passed through a multihead attention layer which enhance the model's ability to learn complex patterns by selectively focusing on the most relevant parts of the combined embeddings. The architecture combines sparse features to form a dense embdedding matrix.

Although the recursive nature of human cognition is not perfectly analogous to neural networks, it holds that the broader network considers features in a similar manner. Considerable progress has been made recently in the field of object recognition, which can be contextualized as feature extraction for geo-informers [16, 2]. While some exploration in automatically determining geo-informers has been made, it is sufficient for our model to manually define such features categorically. By ensuring the breadth and relative weighting of all members of our latent vector, we can sufficiently differentiate one location from another [1].

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 use inconsistent language, so we define the following:

- **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 General Geolocalizers

3.2 Convolution based Locators

One such approach to geolocalization is PlaNet [25], which implements Convolutional Neural Networks (CNNs) for learning localized image features, augmented with Long Short-Term Memory (LSTM) layers for processing sequential image sets.

The CNN component is trained on millions of geotagged images sourced from diverse public datasets, including Flickr, Google Street View, and other publicly available geotagged image collections. These images include a wide array of visual features such as vegetation, road signage, architectural styles, landmarks, and other distinctive geoinformers. This extensive training data enables the model to define responses to one of 26,263 multi-scale geocells, thereby defining an interpretable loss function and straightforward output distribution.

PlaNet automatically learns GeoInformtive Features through Convolutional Neural Network layers. The model may capture features as defined above, but also specific local features like defining features for vehicles specific to that locale (or more prevalent in that swatch of geocells).

PlaNet employs adaptive partitioning to define geocells, using Google's S2 geometry library [9]to create smaller cells in denser areas; addessing the potential imbalance of densities for capture locations.

The model performs well in diverse scenes; however, PlaNet encounters challenges in dealing with ambiguous locales. For example, the presence of an old American vehicle in Australia might erroneously suggest a location in Cuba due to strong visual similarities, especially in the absence of other significant geoinformers.

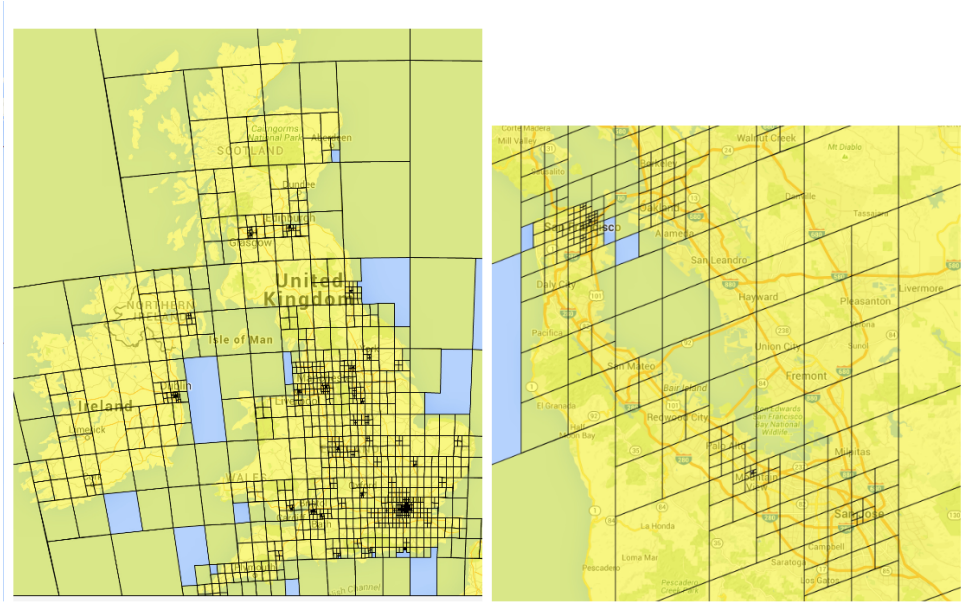


Figure 1: PLANET query cells using S2 partitioning

One key aspect of PlaNet is that all the features are learned automatically by the model. This means that the CNN component autonomously discerns and extracts relevant visual features from the images during the training process. Features such as vegetation, road signage, architectural styles, and landmarks are all identified and learned without manual intervention. This automatic feature learning is a significant advantage, as it allows the model to adapt to a wide variety of geoinformers and improves its ability to generalize across different locations.

However, this also poses challenges around interpretability and genrats, essentially a *black box* of known features. Comparably, the training of Convolutional Layers here is expensive and managing relying on unimportant features presents a challenge.

3.3 Transformer Based Locators

Lukas Haas, Michal Skreta, Silas Alberti, and Chelsea Finn propose PIGEON (Predicting Image Geolocations), PIGEON levrerages CLIP [17] (a multi-modal model trained on some 400m image / label pairs) as a backbone for extracting visual features from images, extending the model through multi-task contrastive pre-training and heirarchical guess refinement to enable extraction of local GeoInformative Features.

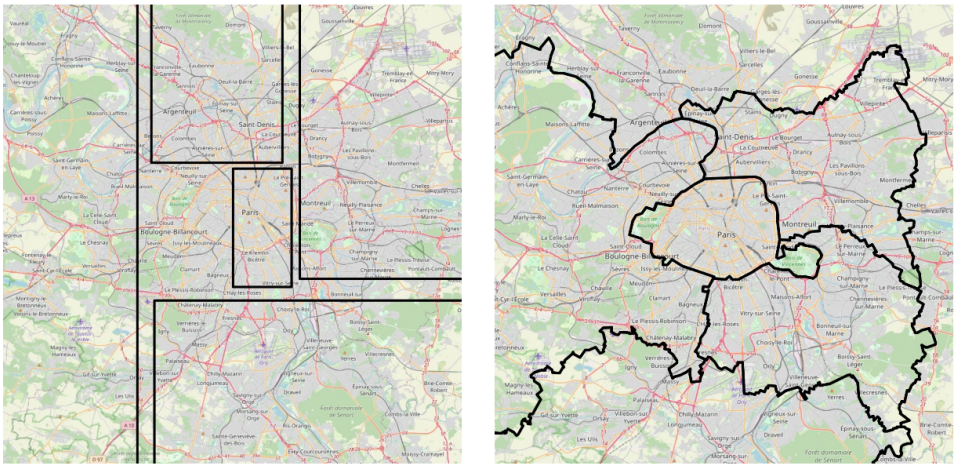
Unlike PlaNet(**?**) which defines *geocell shapes* arbitrarily, PIGEON employes semantic geocell creation, wherein cells are balanced in size (images per cell) and flattens their distribution. PIGEON also employs haversine [21] smoothing in the loss function

Unlike PlaNet [25], 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[21] 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[11].



(a) With naive, rectangular geocells.

(b) With our semantic geocells.

Figure 2: PLANET query cells using Semantic partitioning

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.

3.4 GeoInformative Features

Essentially, geolocalization is distilled to determining *which features best identify this locale?* We refer to these features as *GeoInformers*, and collectively, they form a *GeoInformative pattern.*

While Haas et al showed that automatically derviing geoinformers is now trivial with convolutional neural networks [25], other approaches can provide useful context. Convolutional layers are computationally expensive, Doersch et al [5] 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 [22] 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.

3.5 Extracting GeoInformers

3.5.1 Object Detection

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[5] 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 [20]. 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 [18]. 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 [19]. 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 [4].

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 [12]. 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 [14, 20, 18, 19, 4, 12, 3]

As shown in Table 1, the performance of different YOLO models on the COCO dataset varies significantly [20, 18, 19, 4, 12, 3]. YOLOv1 achieved a mAP of 33.4%, while YOLOv11 reached 73.8% on the common baseline COCO dataset [14]

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, making it computationally efficient without compromising accuracy [3]. 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[15]) making it a useful *drop in* GeoInformative model

3.6 Optical Character Recognition for GeoInformers

According to Pro GeoGuesser players, *script* or *text* represents some of the most informative features for pro *GeoGeussr* players [7] [24] 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 saught to combat here were artifacts of streetview imagery, such as blur, shutter distortion and other distortions.

3.7 Off-the-shelf OCR models

Optical Character Recognition (OCR) is a domain particularly well-suited to deep learning. This is demonstrated by Li et al. [**li2021transformer**], who present a transformer-based approach for document unwarping and illumination correction, as well as for text localization and character recognition. This model achieves real-time detection through the use of transformer encoders, which detect the relationship between swatches of the input image to localize and identify text. However, the model has limitations; it does not handle highly complex warped text well and cannot infer characters that it has not seen during training.

The docTR model is available as an off-the-shelf solution from the public github repository making it a viable *drop* GeoInformer . Despite its computational demands, trading off for 16-bit floating-point values can mitigate some of the load, as acknowledged in the implementation [**li2021implementation**]. This trade-off allows the model to maintain accuracy while reducing resource requirements, making it more feasible for practical use.

Computationally, docTR is demanding; however the implementation acknowledges that trading off for 16pt floating values can be considered[6]

3.8 Combined Embeddings as a GeoInformative Pattern and Attention

In 2017, Vaswani et al [23] 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.

4 Data

Google StreetView provides an extensive resource of images captured at street level. These images are captured at variable distances, typically ranging from every 10 to 20 meters in urban areas to greater distances in rural and remote locations, depending on the area's density and road layout. Updates to the dataset typically occur every 1-3 years in urban areas and every 5-10 years in rural/remote areas. Historic captures remain queriable. Streetview data will uniquely contribute to the dataset.

4.1 Dataset size and scope

This paper proposes a city scale geolocalizer. Google do not publically disclose the number of captures in a region. We intend to build our dataset with as close to 100 percent of the available images in our candidate location.

This paper takes a 10kmradius of sample images centered about the *considered middle* of Chicago.

4.2 Querying Candidate Areas within 10km

Each point within the defined 10kmspace is systematically evaluated for potential inclusion in the training dataset, with hardware constraints playing a critical role in determining viable candidates.

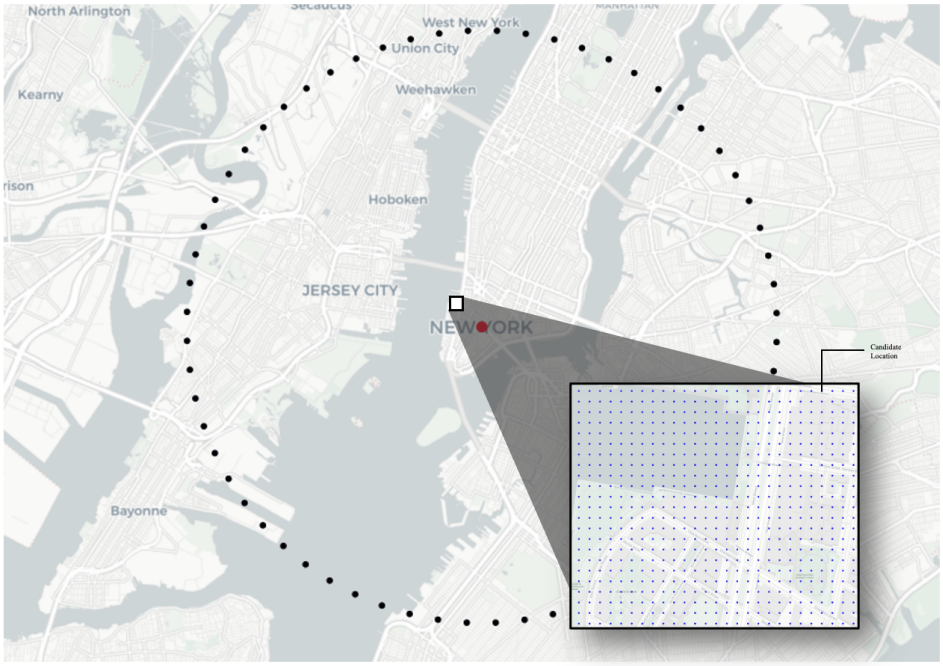


Figure 3: Example of the 10kmarea evaluated for query points. Blue dots represent plausible points to query for data collection

The identified candidate areas for queries are shown below; In order to reduce API calls, we conform the candidates to only contain those which intersect a viable StreetView location (i.e street and road) on the OpenStreetMap Datastet. The relative spacing d for each point considers the streetview capture spacing, which in itself varies from a few meters up 20 meters [10]. $d = 8$ will sufficiently capture sufficient datapoints.

Figure 4: Example of points reduced to candidate locations and the result of querying the GoogleMaps API (third image generated using Microsoft Copilot 14/11/2024)

To further refine the pool of candidate areas, we apply filtering based on OpenStreetMap data using ShapeFiles (obtained from OpenStreetMap), reducing the number of calls to the streetview API.

Note a reduction in the range of 50-90 percent of candidates occurs.

After querying the Google Street View API, the dataset comprises A yet unknown number of imagesthree-channel images, each with a resolution of *640*640*. These images are categorized by their geographic coordinates (latitude and longitude) according to the query locations returned by the streetview API, which *snaps* a query to the nearest LAT / LONG. If a candidate location queries the API successfully, the LAT / LONG becomes the new truth LAT/LONG for that data point. For the purpose of API cost minimization and the purpose of reducing noise or the potential of learning features which are local to artefacts in the query areas caused by our approach, queries within the d distance are no longer candidates unless they have already been queried.

4.3 Recency

Where there are multiple available Google StreetView Images available (at differing times) for the same location, all images become a member of the dataset.

4.4 Preprocessing and Data

This GeoLocator is scoped to Chicagoin the interest of computability; we define a 10kmradius around the center point of Chicago.

We define the basic Preprocessing pipeline as follows; noting that Google StreetView *snaps* queries to the nearest available capture

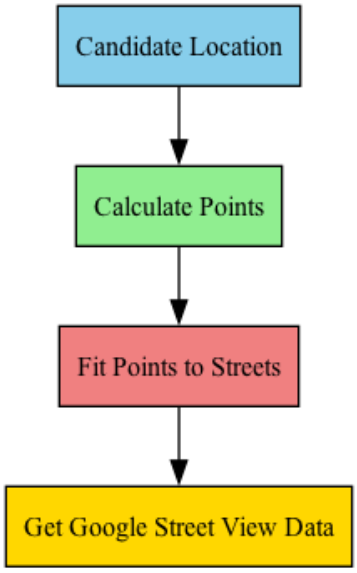


Figure 5: PreProcessing Overview

From the coordinates bounded within this radius, we define a pool of training locations to query through the Google Street View API. Noting that the distances between street view imagery range from a few meters to a dozen or so, duplication of images is undesirable in the training set, and missing images are inconsequential.

The volume of training points for a given radius is simply as follows with integer rounding:

$$NumQueries = \frac{\pi \times R^2}{d^2}$$

Where d is the distance (m) between each query point and the scaling of total candidate points about a radius is exponential. We define $d = 10$ notwithstanding that google streetview images occupy only some proportion of this location, density of streetview imagery is not consistent across locales.

A critical notion in the test data is that local features used as geolocators should be local to multiple images, and defining d thusly is critical to achieving good generalization of the model.

Radius (km)	Number of Points (N)
1	$31,416$
5	$785,398$
10	$3,141,593$
11	$3,801,327$
12	$4,523,893$

While we select d = 10, we see the scaling for increasingly large d above.

5 Method

Our goal is to *Define GeoInformers* and construct a model which, given *GeoInformers* can predict the location of an usneen image at *cityscope*.

GeoInformers are determined from a branched path and their embeddings are used as the input for further NN layers.

5.1 Street Level Imagery

Consider the following image, randomly chosen in *Chicago*. The image intuitively ocntains a number of useful geoinformers.

Figure 6: A random location in Chicago, text is a significant identifer (screencapture from MAPS public website

5.2 Pipeline

The overall structure of our implementation is defined in 8. The final model architecture is not determined.

The model has three branches:

- **Color Evaluation**:
  - Generates a histogram for the HSV color space.
  - Produces a corresponding color embedding.
- **OCR Text Extraction**:

- Utilizes an OCR model to extract text. (docTR[6]
- Creates a text embedding from the extracted textual information.

- **Object Detection**:
  - Employs an object detection model[3]trained on a manually selected pool of plausible objects.
  - Generates object embeddings for the detected items.

An attention layer then processes the combined embeddings:

- Dynamically weighs the importance of each feature.
- Enhances the overall geoinformative power of the model.

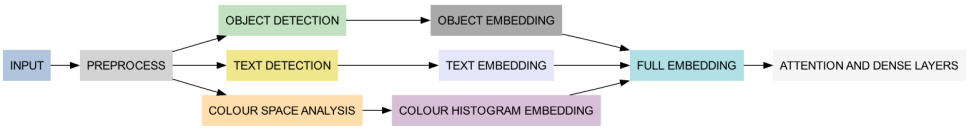


Figure 7: Overall structure of the implementation pipeline.

5.3 Architecture and Activation Functions

Our model processes 640×640 input images using YOLOv11 for object detection and docTR for text recognition.

The input tensor size is $(B, 640, 640, 3)$.

YOLOv11 outputs a tensor $(B, N, 4 + 1 + C)$, where N is the number of detections per image, 4 represents the bounding box coordinates $[x1, x2, y1, y2]$, 1 is the objectness score (Confidence), and C is the number of class probabilities (Possible OutputClasses).

docTR outputs a tensor (B, T, d), where T is the number of text instances detected, and d is the dimension of the text features.

The YOLOv11 embeddings are reshaped to (B, N, E_y) and the docTR embeddings to (B, T, E_d).

The combined tensor size before the attention model is $(B, N+T, E)$, where E is the unified embedding dimension.

Given that the model exceeds 7 layers, we use activation functions like ReLU or Leaky ReLU to mitigate gradient loss, ensuring effective learning.

5.4 Colour Histogram Embedding

The overall colourspace of an image is a telling Geolocator which permeates across input variables. The roofline in 9, for example, is a telling geolocator, yet the sky is consistent but is not a useful feature for our model.

This branch of the model 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.

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).

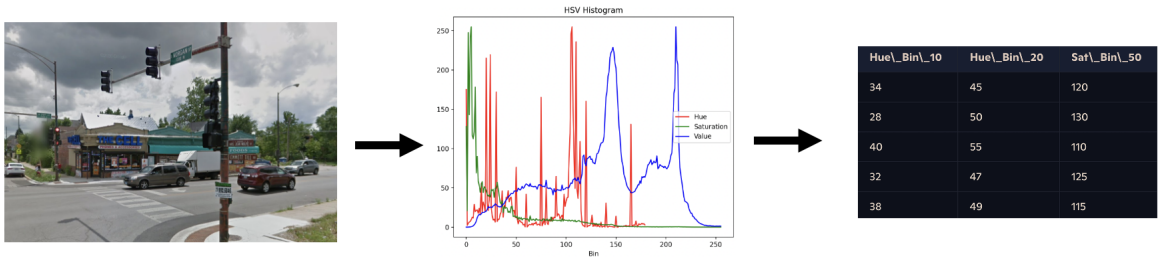


Figure 8: Example of HSV extraction from 9 and a sample embedding (truncated)

5.5 Text Extraction and Processing

Text is a significant geolocator, we employ an off-the-shelf solution: the docTR OCR model [6] [13]. No hyperparameter tuning or model refinement is applied to this branch of the model. Images are normalized before feeding into the model.

Each training set member undergoes processing via the docTR model, which yields an object containing the localized text. While docTR may not capture every piece of text flawlessly, any errors introduced are minor and do not significantly impact the performance of the greater model.

Following extraction, the text is tokenized, removing punctuation. Each word is subsequently embedded into a high-dimensional space to capture the intricate relationships between words. For this, we utilize standard, pre-built embedding and tokenization tools provided by TensorFlow.

The training for this section of the model is informed by the overall model backgpropogation. We do not define any separate loss function for the attention layer.

Additionally a multi-head attention mechanism is included in the model. To refine the significance of each word, especially given the expected frequency of common terms such as 'the', 'ph', 'call', etc.

5.6 Combining Text and Object Embeddings

Objects represent useful local features for Geolocalization and are therefore a sensible choice to include in the model.

For each image, the *YOLOV11* layer produces some feature vector. These feature vectors are transformed into embeddings through a dense layer. (i.e. resultant tensor is of shape (B*S*E), where E, *embedding length* will just be the same size per *OCR Pass*

To combine the object and word embeddings, their dimensions are aligned before they are concatenated to create a unified representation (of size B,S + N ,E)

The Attention Layer Continues as per *OCR Pass*

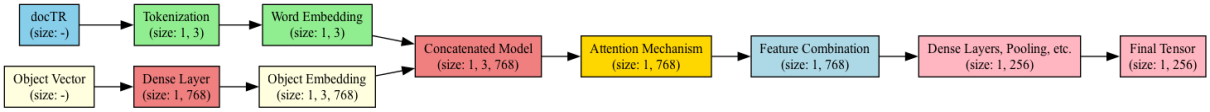


Figure 9: The model for processing the text branch of the core model and combining models

5.7 Combining Text and Object with the Colour Embedding

The same concatenation process is applied to the Text + Object Embedding with the Colour Embedding as in 4.6

5.8 Object Detection and Producing the Feature Vector

YoloV11[3] is useful as an *off-the-shelf* object detector which is capable of running in *real time* on the environment hardware. YoloV11 is, however trained on the COCO[15] dataset, which contains some *80* object catagories, including some which are not useful geoinformers such as *car*, whose inclusion in the overall model will introduce noise (our dataset is not large enough to warrant car type as a geoinformer). We tailor YoloV11 to identify only the relevant classes such as bus stops, American flags, street lights, and post boxes. (Other informers can be determined manually as we encounter more data).

We extend YOLO to detect the classes that are not already members of the COOC dataset [15] by providing annotated data which contains these classes. The first few layers are frozen, the model is trained again with a low learning rate to learn these new annotated classes.

We currently consider American flags and corporate logos like starbucks to be geoinformers and thusly will train the YOLOV11 branch to determine their appearance, however more informers are determined with time.

Embedding The embedding vectors are simply the confidence vectors produced by YoloV11, Normalized.

5.9 Attention, Dense Layers and Outputs

The model embeds colour information, text information and object information into a concantenated embedding space. An important consideration is *Attention Layers*. Our NN architecture is not fully defined at this time, yet we build up sparse geolocalizers into a denser embedding matrix. Attention captures the importance of the features in this embdedding matrix [23]. This will crucially help to mitigate nonsense words like 'the, ph' etc and colourspace consistencies introducing noise to the later dense layers. Dense layers are used to align the embeddings sizes.

5.10 Training The Model

Paths? All trainable variables are trained through a single back propagation step except those in *YOlOV11*'s fine tuning and changes to classification.

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)

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

Intuitively, geographic distance can be used to derive our loss function. The Haversine [21] 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.

Euclidean distance is a poor choice of loss metric in this case for sparse locations like rural New Zealand, but would be a sufficient choice for a small total area (dense).

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:

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

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

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

6 Results

6.1 Environment and Hardware

The entire model is intended to train on the following single GPU system.

Component	Specification
Graphics Card	1x NVIDIA RTX 3080 10GB
System Memory	32GB
Storage	Several TB of storage
Data Source	Commercially Supplied Data from Google Street View API

Table 2: System Specifications

The Model creates a regression of tuple values for Latitude and Longitude $(\hat{\lambda}, \hat{\phi})$. Accuracy is defined with the same haversine[21] calculation based loss function as is used in training.

The test portion of the dataset provides the final categorizer of the data:

6.2 Output values

On our test set we compute the loss vals in terms of the haversine function. This is done for all steps. We define a reasonable epoch bound based on the real world time to execute on hardware.

We also define areasonable accuracy in terms of units for the haversine function. The model is effective if it can reasonably accurately locate an unseen image (a member of test) if it is within the candidate points.

If so we sufficiently show that, indeed, the model has learnt features between captures that can be geiolocators.

6.3 Shortcomings

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.

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.

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.

Lastly, the reliance on Google Street View images limits the model's applicability to areas where such imagery is available and up-to-date. For regions lacking comprehensive or current Street View coverage, the model may not perform as expected, necessitating alternative data sources or reprocessing techniques to fill the gaps.

Other attempts use geocells, we used a regression for LAT and LONG, expecting the model to learn all latent features for the entire city. Learning features in a cell may make more sense given the sparsity concerns of google street view images.

7 Ethics

8 Timeline

This project is scoped to approximately 12 weeks, divided into six distinct milestones, commencing on November 18th, 2024. Our timeline does not consider breaks for holidays, new years, etc.

Our milestones are structured in a way which reflects the large amount of data required and the long training time on constrained resources.

Milestone 1: Tools, Preliminary Structure of Models and Modular Components
Dates: November 18th - December 1st, 2024
Description: Develop tools such as structured querying for street view, basic deployment of docTR, and functional YOLO models. Ensure these components are operational for further development.
Deliverables: Functional training environment and toolset
Dependencies: None
Notes: About half of the required tools already exist in the project GitHub repository at github.com/2of

Milestone 2: Data Collection
Dates: December 2nd - December 15th, 2024
Description: Annotate data for YOLO, create histograms, and develop functional combinatorial embeddings, though not yet with the complete dataset.
Deliverables: Annotated dataset, initial embeddings
Dependencies: Completion of Milestone 1

Milestone 3: Model Development and Data Capture
Dates: December 16th - December 29th, 2024
Description: Complete the data capture process and confirm the feasibility of the network on a small sample. Develop the initial model architecture, including the OCR pass, tokenization, embedding, attention mechanism, and integration with visual features.

Deliverables: Full dataset, feasibility report, and prototype model
Dependencies: Completion of Milestones 1 and 2
Notes: Focus on building a robust and flexible model structure.

Milestone 4: Initial Model Training
Dates: December 30th, 2024 - January 12th, 2025
Description: Train the initial model on the collected dataset, with writing to reflect this process. Prepare a report that awaits results and contextualization.
Deliverables: Initial trained model and preliminary report
Dependencies: Completion of Milestone 3
Notes: Monitor training progress and adjust parameters as necessary.

Milestone 5: Model Evaluation and Hyperparameter Tuning
Dates: January 13th - January 26th, 2025
Description: Evaluate the model's performance using various metrics, tune hyperparameters as needed, and contextualize results against existing models.
Deliverables: Trained model with tuned hyperparameters, performance report
Dependencies: Completion of Milestones 3 and 4
Notes: Perform thorough testing to ensure reliability and accuracy.

Milestone 6: Final Refinements and Presentation
Dates: January 27th - February 9th, 2025
Description: Refine the model based on evaluation feedback, optimize its efficiency, and prepare for deployment. Explore any shortcomings in a finalized report and back-port the presentation from the report.
Deliverables: Final optimized model, comprehensive report, and presentation
Dependencies: Completion of Milestone 5
Notes: Focus on final tweaks and ensuring readiness for real-world application.
bib

References

[1] Author(s). "GeoAI: A Review of Artificial Intelligence Approaches for the Interpretation of Complex Geomatics Data". In: *Journal Name* (Year). Overview of AI-based techniques for analyzing geomatics data. URL: URL.

[2] Author(s). "Object Detection and Image Segmentation with Deep Learning on Earth Observation Data". In: *Journal Name* (Year). Review of deep learning applications in object detection and image segmentation. URL: URL.

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

[4] 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.

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

[6] Hao Feng, Yuchen Li, and Yuchen Wang. *DocTr: Document Image Transformer for Geometric Unwarping and Illumination Correction*. https://github.com/mindee/doctr. Accessed: 2024-11-13. 2021.

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

[8] *GeoGuessr*. Gamification of geolocation for human players. URL: https://www.geoguessr.com/.

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

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

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

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

[13] Wenwen Li et al. "DocTr: Document Image Transformer for Geometric Unwarping and Illumination Correction". In: *arXiv preprint arXiv:2110.12942* (2021).

[14] 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.

[15] 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.

[16] *Pic2Map*. Online EXIF data viewer that analyzes GPS coordinates embedded in images. URL: https://www.pic2map.com/.

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

[18] 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.

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

[20] 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.

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

[22] 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.

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

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

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