# Neural Networks Applied to Low Resolution Satellite Imagery for Highway Traffic Estimation

Oliver Brady
Stanford University
oqbrady@stanford.edu

Nate Keating
Stanford University
nkeating@stanford.edu

## Abstract

*We propose a novel approach for highway traffic estimation that uses low resolution satellite images to predict the number of cars that pass by a traffic station over the course of an hour. To accomplish this task we train two types of models, a Convolution Neural Network (CNN) and a Graph Convolutional Network (GCN) on satellite images of 60 miles of road around particular traffic stations, and then evaluate these models against Federal Highway Traffic data. We find that our models perform well on estimating traffic for a station that has other images in the training data, but still struggle on out-of-sample predictions. We believe our models are taking advantage of the parallax effect in satellite imagery, a displacement difference in RGB values that occurs when a satellite tries to capture a photo of an object moving quickly. The CNN generally outperforms the GCN on this task.*

## 1. Introduction

In the United States, vehicle detection and travel monitoring are well-funded and essential functions of the Federal Highway Administration [6] which operates or partners with over 5000 continuous traffic counting locations nationwide [5]. These data are employed in wide-ranging efforts to enhance public safety, reduce congestion, reduce cost and environmental impacts as well as by researchers for economic, public health, or other modeling and research. In many parts of the world, this level of infrastructure or investment is not feasible or existent. Significant value would result from an alternative and low cost method of highway usage estimation.

Our proposed solution to this problem is a model trained on publicly available Satellite imagery generated by the satellite Sentinel 2 mission which captures the entire world every five days. Sentinal 2 is a constellation mission of twin satellites on identical, antipodal orbits operated by the European Space Agency (ESA) and designed for data continuity and for change-detection applications. [2] For labels we use the United States traffic monitoring system, a collection of 5,000 stations in America with traffic totals for every hour of the day.

We have two tasks we score our model on: (1) in-sample traffic estimation and (2) out-of-sample traffic estimation. For the in-sample task we simply pool all collected data across stations and then split the data into train-val-test. For the out-of-sample task we train on data from every station except for one holdout station, and then test on the holdout station. The out-of-sample task is more challenging; however, it serves as a better test of model robustness for estimating traffic on an arbitrary stretch of highway.

A challenge of the data we are working with is that the 60 mile stretches of road bend and curve and vary from station to station. We solve this issue for the CNN by first applying a road straightening algorithm to standardize inputs across traffic stations. The GCN does not require such change the raw data as it does not have strict structural requirements for inputs. It simply treats each pixel as a node in a graph with the RGB values as features.

For both the models the input is a 2-d array that was exported from Google Earth Engine and represents 60 miles of road on one specific day for a given traffic station. Each row in the array is a pixel with 5 values. Those values are R, G, B, latitude, and longitude. Latitude and longitude are in earth engine coordinate form and are thus all integers. The CNN then processes the data into a straightened road image while the GCN processes the data into a graph object. After passing the input through the model they both output one unbounded number that is trained to represent the expected traffic score.

## 2. Related Works

Previous work like Kaack *et al*. [10] have shown that it is possible to estimate traffic by detecting cars from high-resolution satellite images. In their paper they detect vehicles from photos of resolution 31cm x 31cm per pixel. Sentinel 2 on the other hand is orders of magnitude lower

resolution, 10m x 10m per pixel. Satellite images with high enough resolution to detect vehicles are either very sparingly taken or are prohibitively expensive to acquire. Numerous free high-resolution datasets such as Bing or Google Maps do not have timestamps associated with the images that can be connected with accurate vehicle counts. Thus, high quality traffic estimates from low-resolution images are preferred for a model to be widely useful in real world applications.

Images in the Sentinel 2 database are not high enough resolution for individual cars to be identified by humans. Instead of detecting individual vehicles, we are looking to utilize the parallax effect, which is a color distortion that occurs to high-speed objects in images captured by the Sentinel 2 satellite due to differing speed of red green and blue wavelengths. Liu *et al*. [9] have shown that for large objects moving very quickly e.g. planes, both speed and direction can be calculated by differences in the RGB distortions from the parallax effect. While a plane flying at 500 mph may distort ten 10m x 10m pixels, a car moving 60 mph on the highway is not likely to distort more than one or two. The parallax effect from large vehicles such as a semi-trailer truck may have a slightly larger effect, e.g. 2-4 pixels. As shown in figure 3, the mass of moving vehicles on a highway appear as ripples of pixel distortion in a sentinel image, not as individual pixels distorted at the location off each vehicle. The 10m x 10m per pixel resolution also means that multiple cars may be contained in the same pixel. Thus, this traffic estimation problem is not a pure object detection problem, but rather a regression from an image.

Fischer *et al*. [13] has shown that it is possible, though difficult, to train CNNs on regression problems. A key insight from Fischer is that regression problems often allow for greater augmentation to the input dataset than classification tasks. In the MNIST handwritten digits task, a flipped upside down 6 becomes a 9 and the label no longer correlates. In our traffic estimation task we can splice the road up into a wide variety of different chunks and reassemble it and the same number of vehicles should be present in the image.

For our road straightening algorithm, the only similar task we could find in the literature was in a medical journal where Peng *et al*. developed an algorithm to straighten X-rays of the Caenorhabditis elegans roundworm so that standardized computer algorithms could be applied to them [12]. While this algorithm was not entirely applicable to our problem, it provided procedural inspiration: first to find the "spine" of the road, and then use each step along the spine to anchor a row in our end-state rectangular image.

We were worried during our milestone that our road straightening algorithm was injecting unnecessary noise into our model, which prompted us to build and train a GCN for our final project, which does not require road straightening. Graph Convolutional Neural Networks (GCNs) are a subfield of Graph Neural Networks (GNNs) which has been growing in popularity in recent years. The core idea behind a GNN is that it takes as an input graph $G = (V, E)$ where $V$ is a set of vertices and $E$ is a set of edges [16]. The model then has a variety of ways to propagate information throughout the graph as it trains. There are a different outputs based on the type of GNN that is being implemented. As Zhou *et al*. and others have shown, there are a variety of powerful methods for propagating information with one graph as an input, and node classification as an output [15]. Node classification is generally the most common use case for GNNs. In the past few years, however, developments in papers like Mondal *et al*. have made graph classification popular as well [11]. In graph classification, a batch of graphs is passed into the model, and classification labels are outputted.

With the current interest in batch classification, many methods have been developed to replicate the way information propagates in a CNN, and then applied to GNNs. Kipf *et al*. successfully implement a GNN that uses graph convolutions, where nodes learn information from their neighbors like standard convolutions in CNNs [8]. Networks with such convolutions have come to be known as Graph Convolution Networks (GCNs). The main issue facing researchers in building GCNs that function like CNNs was that nodes in graphs can have variable neighbors, whereas a CNN filter always applies to fixed number of pixels. This issue was solved elegantly by Deffard *et al*. with fast spectral convolutions [4]. In fast spectral convolutions, the graph adjacency matrix is used to calculate the Laplacian of the graph, which is transformed into the Fourier domain where the convolution is performed. This results in a fast and efficient convolution that propagates information locally, but is able to operate on any number of neighboring nodes. Convolutions are often then normalized by degree to not overweight high degree nodes.

With graph classification it is possible to represent images as graphs, and then run classical CNN tasks on them. One issue is that computing the adjacency matrix for a graph is incredibly computationally expensive, so often GCNs cannot perform the same tasks that CNNs can. There are some optimization tricks that can help and Hong *et al*. implement a GCN that is roughly as good and as fast as a CNN at image classification [7], and this remains an active area of research.

By removing the softmax classifier as the top layer it is possible to convert a GCN task from classification to regression. Choi *et al*. proves that this can work by implementing a GCN that predicts human poses [3].

## 3. Dataset

### 3.1. Data Collection

For our ground truth labels we have the FHWA Traffic Monitoring Analysis System (TMAS) which provides hourly traffic counts from over 4,000 traffic stations in America from 2010-2018. Out of those 4,000 stations we have selected 5 for our training as image extraction at each specific location is a labor and computation intensive process. We wanted geographic diversity so we collected images for one station each from CA, TX, MT, MS, and OH. Sentinel 2 images are collected at approximately 10:30 AM local time for timezone of the target location. To line up out inputs and output we made sure to collect traffic labels for the hourly traffic average from 10AM-11AM at each station. As evident in figure 1 and figure 2, these stations have very different distribution of average hourly traffic flow. The y-axis represents cars that are counted between the hours of 10-11 AM and the x-axis shows all dates between 2016-2018.

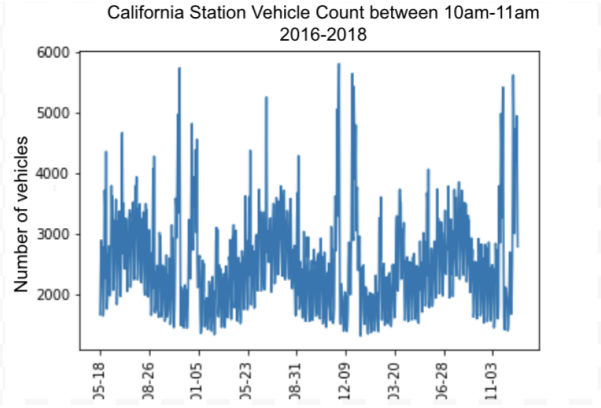


Figure 1. California traffic station time series data

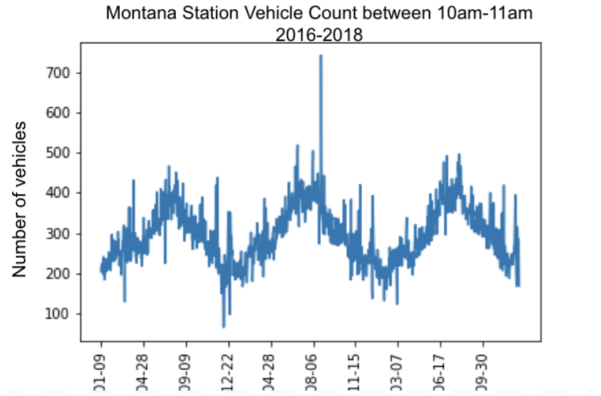


Figure 2. Montana traffic station time series data

For each of the five stations selected we created geometry objects in Google Earth Engine that correspond to 30 miles of highway extending out in either direction from the traffic station. This was accomplished by manually editing the census TIGER interstate road geometries. We chose 60 miles of road because the satellite images are at a specific point in time, but the traffic labels are counts from over the course of an hour so we want a substantial number of cars that passed that station in the hour to be in our images. After creating the road geometry we then masked every photo of the area around the traffic station from 2016-2018 with the geometry object we created. This yielded 136 images of the roads that were completely cloud free across all five stations. Figure 3 shows what a small portion of one image looks like in the earth engine display after the road mask has been applied.

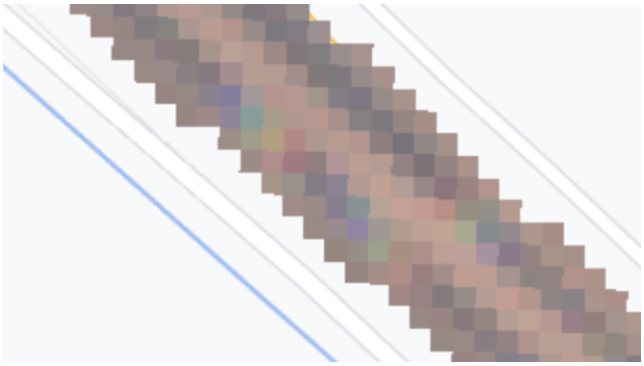

Figure 3. An example of a stretch of road with clear parallax effect.

### 3.2. Data Preparation

We had two different data preparation pipelines, one for our CNN and one for the GCN.

#### 3.2.1 CNN Data Preparation

A challenge with exporting the images from Earth Engine data types is that the road is 60 miles long and is not straight. If we were to draw a bounding box that contained the entire 60 mile road segment it would be over a million pixels even though the longest road mask in our dataset contains only 72,2662 pixels. Our solution to this issue was to add latitude and longitude coordinates as channels to the images while they're still in Earth Engine, and then export those images as two dimensional arrays of size (total pixels in image, 5). The 5 channels represent red, green, blue, lat, long. Once exported into arrays a novel road straightening algorithm we developed is applied to fit them into images of size (8,955, 7) that can be run through a CNN. We believed that this straightening algorithm was harming our results which is why we implemented a GCN where this task would not be necessary.

#### 3.2.2 GCN Data Preparation

GCNs are able to handle inputs of variable size, so no road straightening algorithm was necessary. We did however have to convert the images into graph objects. For this, we employed the Deep Graph Library that is built on top of PyTorch and can apply GraphConv layers to its graph object data types [14]. We took the same 2d array with number of pixels as one dimension and 5 (R, G, B, lat, lon) as the other dimension and converted them into DGL graph objects. To accomplish this we treated every pixel as a vertex and made edge connections between every pixel and its neighbor. We decided to draw connections between diagonal pixels because oftentimes the road runs at a diagonal angle and the car in front is in a diagonal pixel to the car behind it. We also drew edge connections between pixels and themselves (self-loops) as the literature indicates that it improves the performance of GraphConvs [1].

### 3.3. Data Augmentation

To augment our relatively small dataset of 136 images we applied a few transformations to every image. We first flip the x-axis, and then flip the y-axis. We also cut the image in half and switched the top and bottom half of the road. Either applying or not applying each of those transformations turns one single image into 8. We only augmented the training set so with an 80-10-10 train-val-test split, we had 888 images to train on. These augmentation transforms were only performed on the CNN data. The GCN has no spatial ordering of its nodes so flipping the x and y axis would not create an additional data point.

## 4. Methods

We approached the problem with two neural network architectures, a standard convolutional network (CNN) and a graph convolutional network (GCN).

### 4.1. Convolutional Neural Networks

We currently have two versions of our CNN model, one for in-sample predictions, and one for out-of-sample predictions. The first model, $CNN_1$, has two convolutional layers then the image is flattened and then 3 fully connected layers are applied. The road images are of shape (8995, 7, 3). Some of the stations have large images, but we truncate road length to standardize the input. Conv1 and conv2 have 10 3x3 filters with a stride and padding of 1. The fully connected layers have 100, 50, and 1 output nodes respectively. There are batchnorm layers after the convolution layers. Figure 4 visualizes the architecture. These architecture decisions were made through experimentation on the validation set. It is surprising that pooling layers did not help performance, we think this may be because lots of CNN architectures are designed for object detection, but we have no shapes or edges in our data. That may be why an unconventional architecture performed best.

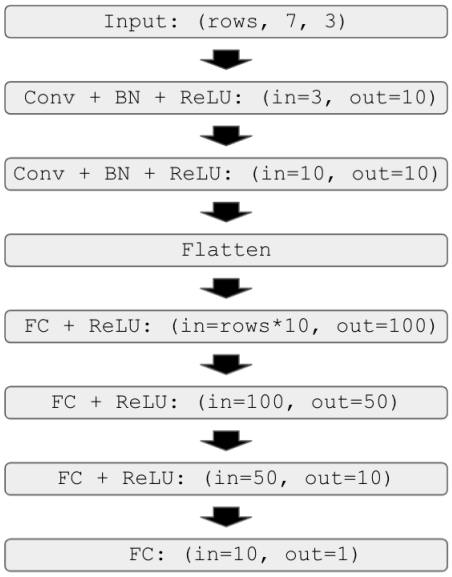


Figure 4. $CNN_1$ model architecture

For the out-of-sample task we began by training on images from four stations and then testing on one but achieved very poor results. One of our advisors pointed out that it would be relatively cheap to purchase just one high quality satellite image and manually count the number of cars in the image. Thus for our task it is reasonable to assume that we have one hour of traffic data labels, and can collect a Sentinel low-res image to accompany it. To allow our regression model to generalize better for out-of-sample we made a few changes. We removed the third convolutional layer and second fully connected layer to help decrease overfitting. We also made one pass through the network with all of our out-of-sample images, and then made a pass through the network with just our single in-sample image. After both passes we backpropagated the sum of the loss. This is similar to weighting the in-sample image as 50 percent of our dataset. This model we label $CNN_2$. For both of these models we used mean squared error as a loss function, but used the R-squared value of the correlation between actual and predicted traffic to evaluate.

### 4.2. Graph Convolutional Network

Due to concern that we had no accurate quantitative measure of the quality of our road straightening algorithm, we also implemented a GCN as an alternative model without the need for road straightening. We formulate the traffic estimation problem as a GCN to directly address the relational nature of the parallax effect, which produces subtle changes to local pixel neighborhoods. Each pixel in the satellite images is a node in the graph with edges to its neighboring pix-

els. We laid out the general use and architecture of a GCN in section 2, but will delve into the spectral graph convolutions applied in this section.

The graph convolutional layer that we implement is from [4]. The steps necessary for performing the convolution are as follows:

1. Fourier Transform

Let $G = (V, E, W)$ where $V$ is the set of vertices, $E$ is the set of edges, and $W$ is the adjacency matrix. Let there be $n$ vertices in the graph. We first compute the graph Laplacian $L = D - W$ where $D$ is the diagonal degree matrix such that $D_{ii} = \sum_j W_{ij}$. We deviate from the equation above slightly to compute a normalized Laplacian so that nodes with high degrees are not over weighted. This is accomplished by tweaking the equation to $L = I_n - D^{-\frac{1}{2}} W D^{-\frac{1}{2}}$ where $I_n$ is the identity matrix.

Because L is a real symmetric positive semidefinite matrix, we can find the complete set of eigenvectors $\{u_l\}_{l=0}^{n-1}$, and eigenvalues $\{\lambda_l\}_{l=0}^{n-1}$. Those are referred to as the Fourier nodes and frequencies. We can then diagonalize $L$ with $L = U\Lambda U$ where $U$ and $\Lambda$ are the nodes and frequencies in matrix format. To perform a Fourier transform on an $x \in R^n$ we compute $\hat{x} = U^T x$.

2. Spectral Filtering

Since we are not able to filter in the vertex domain, we now wish to filter in the Fourier domain. For signal $x$ and $y$ we can perform a filter operation in the Fourier domain by using the element-wise Hadamard operator. To transform and then apply we apply $U((U^T x) \odot (U^T y))$. To define a function $g_\theta$ that filters a signal $x$ we get:

$$y = g_\theta(L)x = g_\theta(U\Lambda U^T)x = U g_\theta(\Lambda) U^T x$$

If we let $\theta \in \mathbb{R}$ be a vector of Fourier coefficients we can reparameterize this equation to be:

$$g_\theta(\Lambda) = diag(\theta)$$

3. Localization

The last thing we need to accomplish is to make sure that the spectral convolution is propagating local information across the nodes. This is required if we want GCNs to be able to do CNN-esque tasks. To accomplish this we apply a series of polynomial filters to spectral output to make sure that we are only weighing neighbors to the specific node that we are looking at.

Now that we have a graph convolution layer established we can assemble the GCN. The GCN was far more difficult to train than the CNN which we will discuss more in our results section. The model architecture that we found to work best was two GraphConv layers with 3 filters + Leaky ReLU, then a GraphConv layer with one filter. A flatten and then a pad. Instead of truncating images like we did with the CNN, we passed variable input sizes into the GCN. That only becomes an issue at the linear layers so after flattening we padded all vectors to max size. Then we normalized by true number of nodes in the graph to make sure graphs that had less nodes and thus were were more heavily padded did not have lower scores just because their inputs had less road. we then had a linear layer with max pixels as input, 100 nodes as an output, a ReLU activation, then a linear with 100 as input 10 as output, another ReLU and lastly the final output layer with no ReLU. Figure 5 visualizes the architecture. For the GCN we used the same model for both tasks, but used the one-sample pass through loss function for the GCN as well.

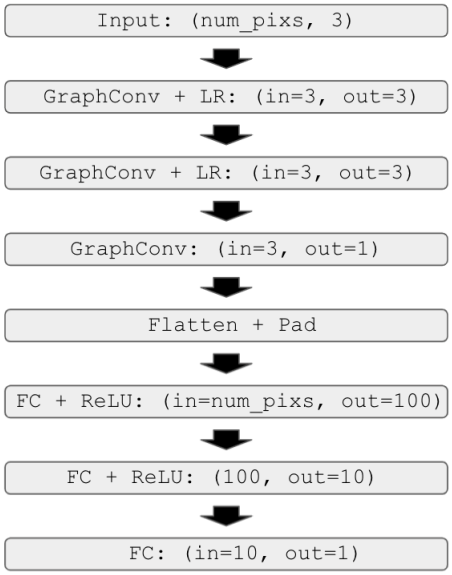


Figure 5. GCN model architecture

# 5. Results

## 5.1. Hyperparameter and Architecture Selection

Our learning rate and Adam optimizer betas we selected empirically. These models begin to over fit in under 10 minutes because of the small size of the data being inputted. Thus we were able to perform many different runs on the validation set before testing. We were using Batch Norm, so we made the batches as large as we could fit in memory and without slowing the model down too much.

We found training the GCN to be far harder than training

the CNN and made a very flexible model design where the number of layer, filter, normalization technique, and activation function were all input parameters. After many unsuccessful runs on the validation set we finally were able to get gradient flow by switching ReLU to Leaky ReLU. Then we tuned the number of layers and filters to get the best performance we could before testing.

### 5.2. Baseline

There is a large discrepancy in mean traffic flow for the five stations ranging from an average of 435 cars an hour for the Montana highway to over 2,000 for I-5 in California. A neural net might easily learn a few geographic features, classify an image as being from a certain station and then predict the mean value for the traffic station. We want to make sure our models learn actual information about the road traffic, and do not just classify the traffic station by location, so we make the mean guessing situation our baseline. If we assume the model can classify the traffic station with 100 percent accuracy and then guesses the mean traffic at that station as learned from the training data, we obtain an R-squared value of 0.87 on the test set. Figure 6 shows the best fit line for our mean guessing baseline.

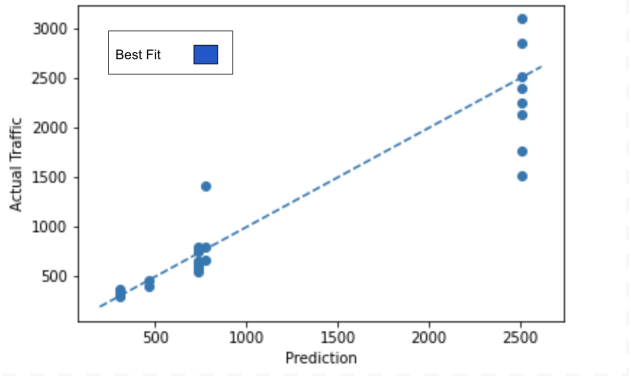


Figure 6. Baseline predictions

For our single-sample task we implemented two baselines. For the first baseline we trained on four stations and tested on one, to see if our model could perform completely out of sample. For our second baseline we trained on just one sample from the test station to see how much value just one sample adds. Both of the baselines had noisy guesses with R-squared values less that 0.001. Thus any positive R-square value with a positive slope for best fit line we will think of as improvement over baseline.

### 5.3. CNN Results

For $\text{CNN}_1$ which was tested on task 1 (in-sample) all images from all stations were pooled and shuffled, and then data was split and augmented. We achieved an R-squared of 0.94 compared to the baseline of 0.87. An encouraging sign is that all of the coordinates with predictions over 1500 are from the California station, and these results show the model is able to predict a wide range of values for images all from the same geographic traffic station, reducing concern of memorizing geography. The baseline graph in figure 6 shows the California station has very high variance for traffic flow, and this model is handling it quite well.

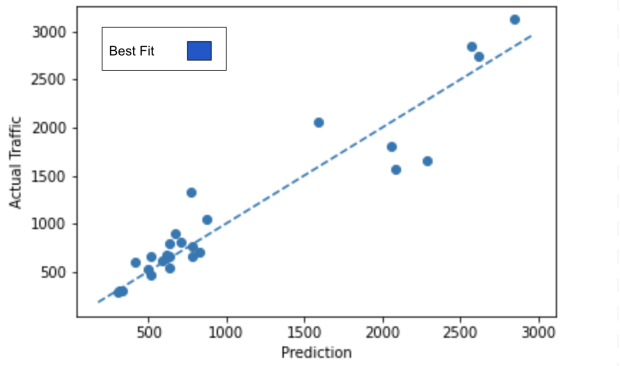


Figure 7. $\text{CNN}_1$ (In-Sample) Results

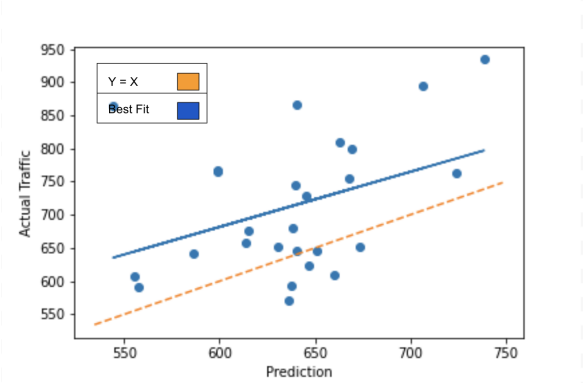


Figure 8. $\text{CNN}_2$ (One-Sample) Results

$\text{CNN}_2$, trained on four stations and one sample of a fifth, achieved an R-squared of 0.11 on the test set. In Figure 8 the orange line represents $y = x$ where the prediction matched the actual traffic, and the blue line represents the line of best fit. It is encouraging that they have similar slopes, but clear that there is still a mean shift from predictions to reality.

### 5.4. GCN Results

For our GCN in-sample results, shown in figure 9, the orange line is the line of best fit and the green line is prediction equals actual. This model slightly under performed the CNN and achieved an R squared of 0.9, but still above baseline of 0.87.

The GCN on the one pass task achieves an R squared

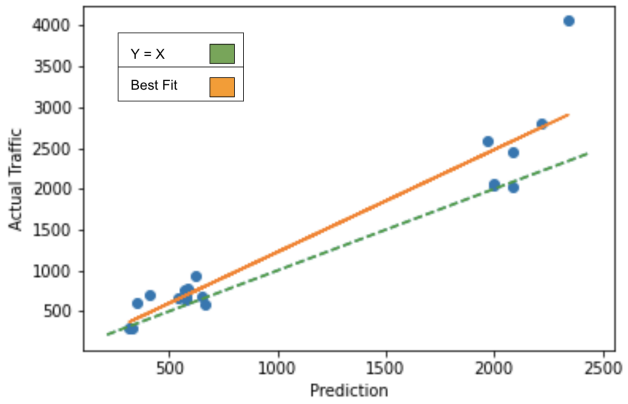


Figure 9. GCN In-Sample Results

Table 1. Model Performance by Task

| Model Type | In-Sample $r^2$ | One-sample $r^2$ |
|---|---|---|
| Baseline | 0.87 | 0 |
| CNN | 0.94 | 0.11 |
| GCN | 0.9 | 0.12 |

of 0.12. A closer inspection shows that the line of best fit is nowhere near the prediction equals actual, so even that small correlation may be just noise.

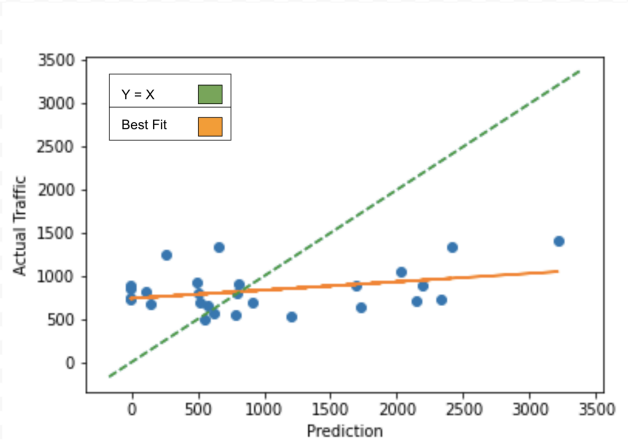


Figure 10. GCN One-Sample Results

## 5.5. Qualitative Results

We also wanted to perform some qualitative experiments on the in-sample CNN to see if our model was recognizing any of the parallax effect. We looked at an input image to find pixels that we thought had parallax effect and noted the RGB values of those pixels. We then extended buffers on either side of the RGB values of the pixel in order to get a handful of other parallax pixels in the image. This was a strategy we initially tried for feature extraction, but unfortunately it is not very effective because the RGB values for parallax pixels vary wildly based on the brightness of the day. We then masked the selected pixels and got a prediction from the CNN model. We found that those pixels had roughly 50 percent more of an the impact on the final score as an equal number of randomly sampled pixels masked out. This is an encouraging sign, though far from conclusive as there could be an underlying correlation between pixels we see as having parallax being more closely associated with the final score–for example just general brightness.

## 5.6. Discussion

We found that as shown in table 1 we achieved our highest performance on the in-sample task with our CNN, and the highest performance on our one-sample task with our GCN. We were pleased with in-sample performance for both the GCN and the CNN, but the CNN especially. The variety and accuracy of the prediction it makes for the high variance California station shows that it learning some meaningful features in these images. For the one-sample task, neither model performed especially well, and while the GCN has a slightly higher $r^2$ a qualitative inspection of the graph suggests that is misleading. The GCN line of best fit is very far off what we would want it to be which is the line $y = x$, or prediction equals reality.

Our results contradicted our initial hypothesis. We believed that by implementing the GCN and eliminating the road straightening algorithm we would see improved results. We thought this might be true for in-sample, but would definitely be true for out of sample. Our rationale is that the road straightening algorithm tended to ”buckle” at places, meaning pixels that were close together in the input array would become far apart after straightened. Buckles were unique to each specific location, so we thought they might be making it harder for the model to be geographically agnostic. We were proven wrong and have two reasons why we believe that might be the case.

The first issue may have been that our graph convolution did not care about the ordering of the neighbors. Considering the classic GCN problem of classified nodes in a graph of published papers where each connection is a citation, the GCN does not care in what order it has cited each of the papers around it. We were using this convolution for a task that was different than it was designed for, and should have either found a different convolution layer, or learned how to encode neighbor information. Another factor may have been that we were far worse at tuning and designing the GCN. It took us a very long time to even overfit a small training set. That may be an indicator that a far more optimal solution exists, but we just did not get that close.

For both models’ poor performance on the one-sample task, this may be attributed to the high variance and small size of our dataset. The California station sees almost 5

times the traffic as Montana and the models only have about 40 images from each to learn that fact. We believe that adding more stations as well as more images from each station would help the models generalize.

## 6. Conclusion

We demonstrate that in theory that traffic estimation with neural networks from satellite images is possible even for images with sufficiently low resolution that a human cannot perform the task. In addition, we add to the evidence in the literature demonstrating the parallax effect can be used to identify objects through through images, even for objects spanning 1-2 pixels.

For future work on this project as a first step we would collect more image data from more traffic stations across America and potentially the world. Then we would try and implement a graph convolutional layer that pays attention to the ordering of neighbors in a graph. If we had a lot more resources, we would devise an algorithm that could run directly on the earth engine output of sentinel images. That would likely require taking sample bounding boxes of the earth, looking for roads and traffic stations and then having a very general model make prediction. The ability to train this model without masking out the road and prepossessing the data would drastically increase the input dataset.

## 7. Contributions & Acknowledgements

We were advised by Stanford Sustain Lab on this project. We presented once every two weeks on our project to them and they would give verbal feedback. Nate worked on data collection, augmentation, and tuning of the CNN. Oliver implemented the GCN and the road straightening algorithm. Code for project: `https://github.com/oqbrady/Sentinel2_Traffic`

## References

[1] B. Acikmese. Spectrum of laplacians for graphs with self-loops, 2015.

[2] E. S. Agency. sentinel-2: The operational copernicus optical high resolution land mission, 2013.

[3] H. Choi, G. Moon, and K. M. Lee. Pose2mesh: Graph convolutional network for 3d human pose and mesh recovery from a 2d human pose. In A. Vedaldi, H. Bischof, T. Brox, and J.-M. Frahm, editors, *Computer Vision – ECCV 2020*, 2020.

[4] M. Defferrard, X. Bresson, and P. Vandergheynst. Convolutional neural networks on graphs with fast localized spectral filtering, 2017.

[5] FHWA. Travel monitoring.

[6] FHWA. Fhwa publications and statistics, 2018.

[7] D. Hong, L. Gao, J. Yao, B. Zhang, A. Plaza, and J. Chanussot. Graph convolutional networks for hyperspectral image classification, 2021.

[8] T. N. Kipf and M. Welling. Semi-supervised classification with graph convolutional networks, 2017.

[9] Y. Liu, B. Xu, W. Zhi, C. Hu, Y. Dong, S. Jin, Y. Lu, T. Chen, W. Xu, Y. Liu, B. Zhao, and W. Lu. Space eye on flying aircraft: From sentinel-2 msi parallax to hybrid computing. *Remote Sensing of Environment*, 246:111867, 2020.

[10] G. H. C. Lynn H. Kaack and M. G. Morgan. Truck traffic monitoring with satellite images. *COMPASS*, 2019.

[11] A. K. Mondal, V. Jain, and K. Siddiqi. Mini-batch graphs for robust image classification, 2021.

[12] H. Peng, F. Long, X. Liu, S. K. Kim, and E. W. Myers. Straightening caenorhabditis elegans images, 2008.

[13] T. B. Philipp Fischer, Alexey Dosovitskiy. Image orientation estimation with convolutional networks. *LNCS*, 2015.

[14] M. Wang, D. Zheng, Z. Ye, Q. Gan, M. Li, X. Song, J. Zhou, C. Ma, L. Yu, Y. Gai, T. Xiao, T. He, G. Karypis, J. Li, and Z. Zhang. Deep graph library: A graph-centric, highly-performant package for graph neural networks. *arXiv preprint arXiv:1909.01315*, 2019.

[15] J. Zhou, C. Chen, L. Zheng, H. Wu, J. Wu, X. Zheng, B. Wu, Z. Liu, and L. Wang. Vertically federated graph neural network for privacy-preserving node classification, 2021.

[16] J. Zhou, G. Cui, S. Hu, Z. Zhang, C. Yang, Z. Liu, L. Wang, C. Li, and M. Sun. Graph neural networks: A review of methods and applications, 2021.