Using Pytorch CNNs to Detect Pedestrians in Autonomous Vehicle Data | Chris Sternberger 2023

In this blog post, we will be exploring Lyft's "**3D Object Detection for Autonomous Vehicles**" Kaggle competition. This competition provides Lidar and Image data for train and test sets as well as a series of JSON files acting as data dictionaries that are paired with their own installable SDK named lyft-dataset-sdk. This SDK provides functionality making data exploring and processing easier out of the box. My goal with this blog post and working repository is to share an application of Pytorch's plug and play CNN models with a relevant and realistic image dataset but with a slightly different approach. Lyft's Kaggle competition was designed to work with 3D Lidar and 3D bounding boxes projected onto 2D images. In this post, I'll walk you through my attempt at an alternative yet related approach of converting their bounding boxes to 2D and attempting basic Image Classification and Object Detection on these images. This adds interesting challenges and nuances to uncover that you may not see with more procured image datasets like CIFAR that aren't guaranteed to have real world noise.

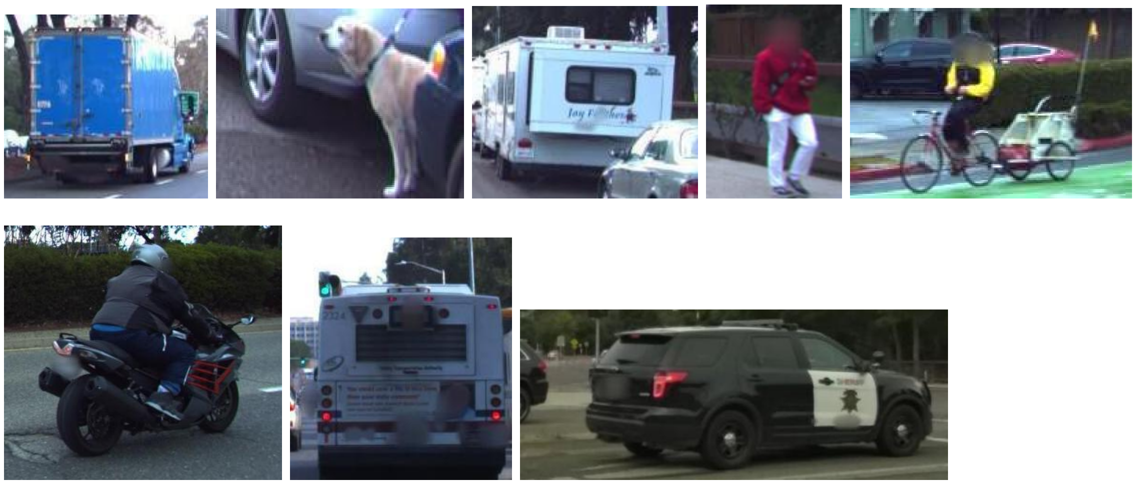
As with most Data Science endeavors, the majority of time and energy spent working was on data manipulation and processing. Being a novice in the space, this was certainly a steep learning curve. With that said, what I would like to achieve is a CNN model that can successfully differentiate between pedestrians and all other objects. Hence, this is a binary classification problem in principle, and I want to extend this further into Object Detection. This first pass will focus more on the application, data manipulation and model inference results rather than focusing on theory or experimenting with and across CNN architectures. Here is an outline of what this post will cover:

1. *Intro to all data available from Lyft, which data we will focus on and their statistics*
2. *Data preprocessing steps and important assumptions/caveats of working with image data*
3. *A brief bit on CNNs themselves*
4. *Model selection and training, hyperparameters, and finally model validation/inference*
5. *Ethical notes*
6. *Next Steps/Future Work*

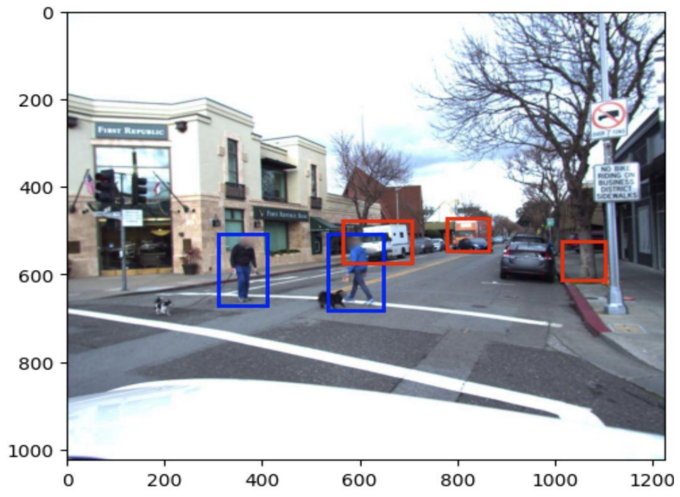
The Data: As indicated earlier, the only data we are using for our analysis is the 2D imagery provided by Lyft in their training data and are ignoring lidar in this iteration. These RGB images are one of three sizes [(3, 2048, 864), (3, 1224, 1024), (3, 1920, 1080)] taken from one of seven cameras that are mounted on their fleet of Autonomous Vehicles (henceforth AVs). All images were recorded in Palo Alto from the months of January to August. Additionally, they have organized all of their data and metadata into many interlocking JSON tables. Each table comes with identifying tokens that can be used to join with other files/tables. The image (and lidar) files all correspond to a sample in sample_data.json, and the sample_token from sample_data.json is the primary identifier used for the train and test samples. Here is a quick breakdown:

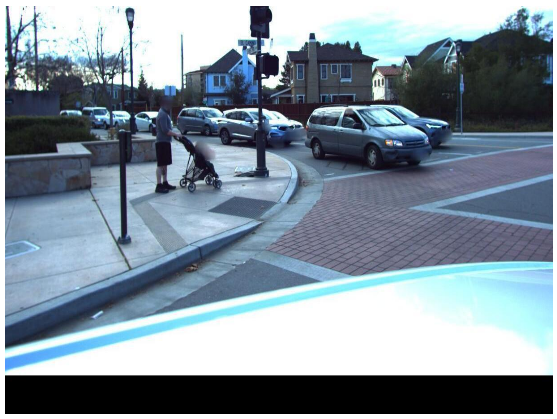
1. `Scene(180)` - 25-45 seconds snippet of a car's journey.
2. `sample(22,680)` - An annotated snapshot of a scene at a particular timestamp.
3. `sample_data(189,504)` - Data collected from a particular sensor.
4. `sample_annotation(638,179)` - An annotated instance of an object
5. `Instance (18,241)` - Enumeration of all object instances we observed.
6. `category(9)` - Taxonomy of object categories (e.g. vehicle, pedestrian).
7. `Attribute (18)` - Property of an instance that can change while category remains the same.
8. `Visibility (4)` - (currently not used)
9. `sensor(10)` - A specific sensor type.
10. `calibrated sensor(148)` - Definition of a particular sensor as calibrated on a particular vehicle.
11. `ego_pose(177,789)` - Ego vehicle poses at a particular timestamp.
12. `log(180)` - Log information from which the data was extracted.
13. `map(1)` - Map data that is stored as binary semantic masks from a top-down view.

For a tutorial on the Lyft SDK itself which streamlines working with all of these tables together, Kaggle user SHREE911 has an excellent tutorial notebook on getting familiar with this data. The main components of the data we are concerned with for our Image Classification and Object recognition tasks are the annotations (henceforth bounding boxes) that exist in sample_annotation and correspond to an image file. There are about 150,000 images included in the training data. There is an important **caveat** to note here: because these images were taken from a moving (or at rest) car at a high frequency, many different images are not very different at all. You can think of taking two photos from a car one half second apart as you accelerate from a stop sign. These two images will contain largely the same information, and this is an important point we will address later in our Methods section. Below, we have sample images of each of the eight object categories involved:



We exclude the disproportionally dominant category “cars” for this application. From upper left to lower right we have: Truck, Animal, Other Vehicle, Pedestrian, Bicycle, Motorcycle, Bus and Emergency Vehicle. These images were cropped to the bounding box originally drawn around the object. The full size images that we start from are shown below and give you the true perspective from the AV.





A quick glance at the distribution of our categories below gives us some starting assumptions for potential data wrangling and class/category balance before we feed it into a model:

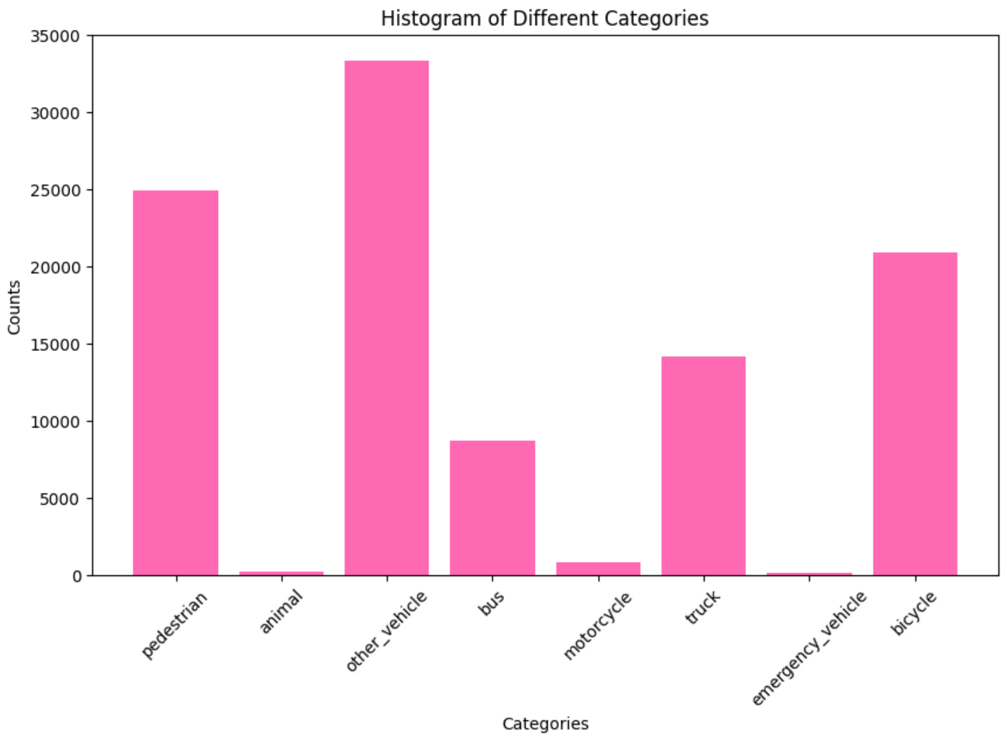


Fig 1. Distribution of object categories.

Methods | <u>Data Pitfalls and Preprocessing:</u>

I took some different approaches to both Image Classification and Object Detection, but I'll first cover the aspects that are required for both. The first pitfall I had to contend with was the fact that Lyft did not include their ground truth bounding boxes for their Test data like they did for their Training data. Lyft held this out intentionally during the competition so they could run submitted models on Test data behind closed doors. The bounding boxes are required for our Image Classification task as I crop the images wrt each bounding box (i.e. having j bounding box-sized images derived from every i-th image). For Object Detection, the bounding boxes are even more vital as I had planned to perform an Intersection Over Union (IoU) to assess the model's ability to delineate the object accurately in image space. For the time being, I had to split the Train data into Train and Validation (all of which are from the original Train set).
Below are the outlined preprocessing steps that I took to get my data ready for ingestion into a CNN model.

General Preprocessing:

1. Use the camera_intrinsic matrix associated with each image data sample to convert the bounding box coords from World space (default) to image space and then convert from 3D to 2D.

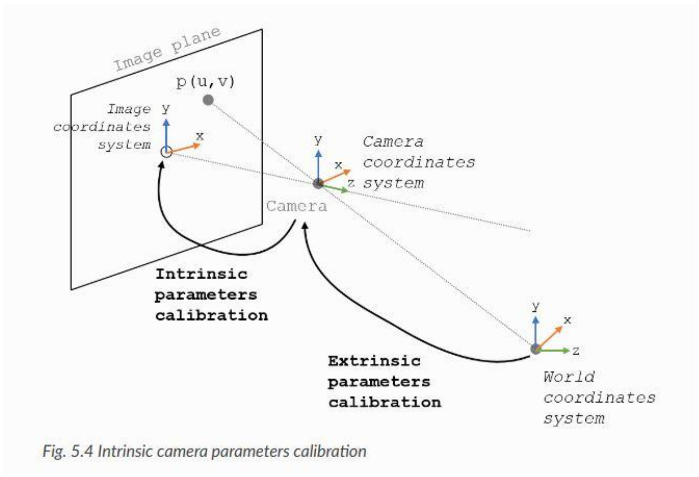
a. 
Image plane
p(u,v)
Image coordinates system
y
x
Camera
y
x
z
Camera coordinates system
Intrinsic parameters calibration
Extrinsic parameters calibration
y
x
z
World coordinates system
Fig. 5.4 Intrinsic camera parameters calibration

Figure 2 Visualization of World to Image coordinate transformation ([5.5. Intrinsic camera parameters calibration — MPHY0026 documentation]())

2. Split original Train data into Train and Validation. Perform this split on the date that the data was recorded. If we were to split the images randomly, then we may end up with an image in our Train set that was taken a half second before an image in the Validation set resulting in data leakage.
   a. **Caveat:** While splitting on date controls for seasonality and unique "drives", it does not control for the locale of data collection. In particular, it is possible that we have an image in the train set at the intersection of Pike St and Pine St at noon on a sunny day with pedestrians and we also have an image of the same location from the same perspective on a different sunny day. This is partially a weakness of the dataset. Future options are discussed in the Discussion section, but I moved forward adopting the assumption that this hypothetical is a non-issue.
3. Create the image_box_lookup table. This table is our main reference for all image preprocessing as it directly links every image in our Training and Validation set to the image's respective bounding boxes represented by their 2D coordinates, the corresponding binary labels for each of the bounding boxes and additionally empty lists ready to be populated by predicted bounding boxes and labels during Object Detection.

Image Classification:

1. For each image in Train and Validation sets, crop the image down to each bounding box and save the resulting images to respective Train and Validation directories. There will be j bounding box sized images for every i-th image.
2. Since bounding boxes vary in size, further crop down or zero pad up the resulting bounding box-sized images to conform to a uniform input size for our model downstream (I use an image size of 224x224)

3. I decided to ***not*** perform data augmentation due to the fixed nature of the orientation of the camera. While slicing images east to west or north to south could provide realistic data points (e.g. the camera only catches the left half of a person), the camera will always be oriented with the sky upward and the ground downward. Again, **caveats** abound in this space…there could be a pedestrian laying down sideways in the middle of the road. While a compelling argument, in order to keep my first attempt manageable, I am not considering such cases.

Object Detection: *I have not yet taken the Tiling approach which may be more efficient. More in Discussion.*

1. Since we need to scan the entire image searching for objects, I realized that certain spaces of every image rarely if ever contain instances of our positive class Pedestrian. The heatmap below illustrates the image locations of every bounding box in both Train and Validation sets that corresponds to a Pedestrian. The left heatmap is a standard heat map and the right heat map displays a binary mask where the purple area is the extent of all pedestrian bounding boxes in both datasets.

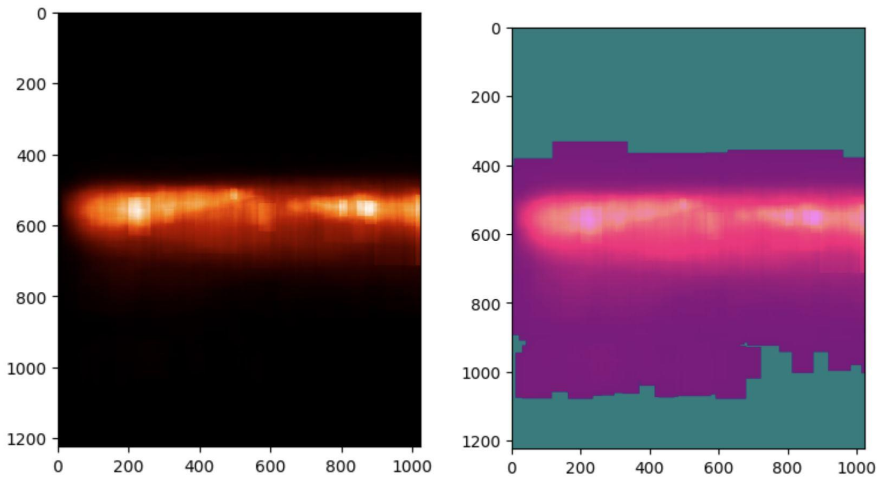


Figure 3. Heatmaps depicting the location and density of Pedestrians in image space

   a. This is actually intuitive. If you observe the two original images on Page 3, you may notice that the upper quarter of the image contains mostly sky or the roofs of buildings. And, the bottom quarter of the image is actually taken up by the hood of the car. With the exception of rare circumstances, this means that we can crop out these portions of the image and thus reduce the image input size into our model.

<u>**Methods |** Model training set up:</u>

There are three main components in a typical Pytorch model training workflow: a custom Dataset class that converts your image data and labels into Pytorch Tensors, a DataLoader class that will invoke the Dataset instance and will yield data batch by batch into your training loop and the model instance itself. The model and DataLoader batch_i can then optionally be sent from CPU to GPU (the GPU used for all training and inference was an NVIDIA GeForce RTX 2070 SUPER). I wanted to explore a greater variety of models, but due to time constraints, for Image Classification, I landed on a single Resnet50 model from Pytorch's model repository with pre-trained weights. There was less experimentation with the choice of CNN model or modification of model architecture that I had hoped for. Rather, I used the time I had to explore different training hyperparameters (see Fig. 5) all for the same Resnet50 model architecture.

For Object Detection, I had begun training with another one of Pytorch's plug and play models Fasterrcnn with a Resnet50 backbone. This again had pretrained weights assigned to it. Unlike with Image Classification, the Object Detection training was much more complex. This is mainly because the GPU available, while performing very well on the Image Classification with very small images, was insufficient to train on images on the order of 800x1224 corresponding to an input layer of size 979,200. Experimenting with batch sizes was limited and sometimes caused memory issues on the GPU. Tiling, alternatively, may be a future solution. It will tile each large image into many smaller ones, which will vastly decrease the depth of the input layer (size of the image) to the model network, but it will also largely increase the number of images required to analyze. For the Evaluation and Results section, we will focus mostly on our results from the Image Classification model.

Convolutional Neural Networks:

I will be very brief in this section. I re-emphasize a point here that I made earlier which is that a *huge* part of building a successful CNN model or any NN is to experiment with different architectures. While I did iterate over some basic hyperparameters like learning rate and batch size, I did not begin to dissect and experiment with the layers of the CNN itself. For example, the first few layers of my Resnet50 look like this:

```
ResNet(
        (conv1): Conv2d(3, 64, kernel_size=(7, 7), stride=(2, 2),
padding=(3, 3), bias=False)
        (bn1): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
        (relu): ReLU(inplace=True)
        (maxpool): MaxPool2d(kernel_size=3, stride=2, padding=1,
dilation=1, ceil_mode=False)
        (layer1): Sequential(
          (0): Bottleneck(
            (conv1): Conv2d(64, 64, kernel_size=(1, 1), stride=(1, 1),
bias=False)
```

And, just as a reminder, here are the patterns that the very first convolutional layer learned:

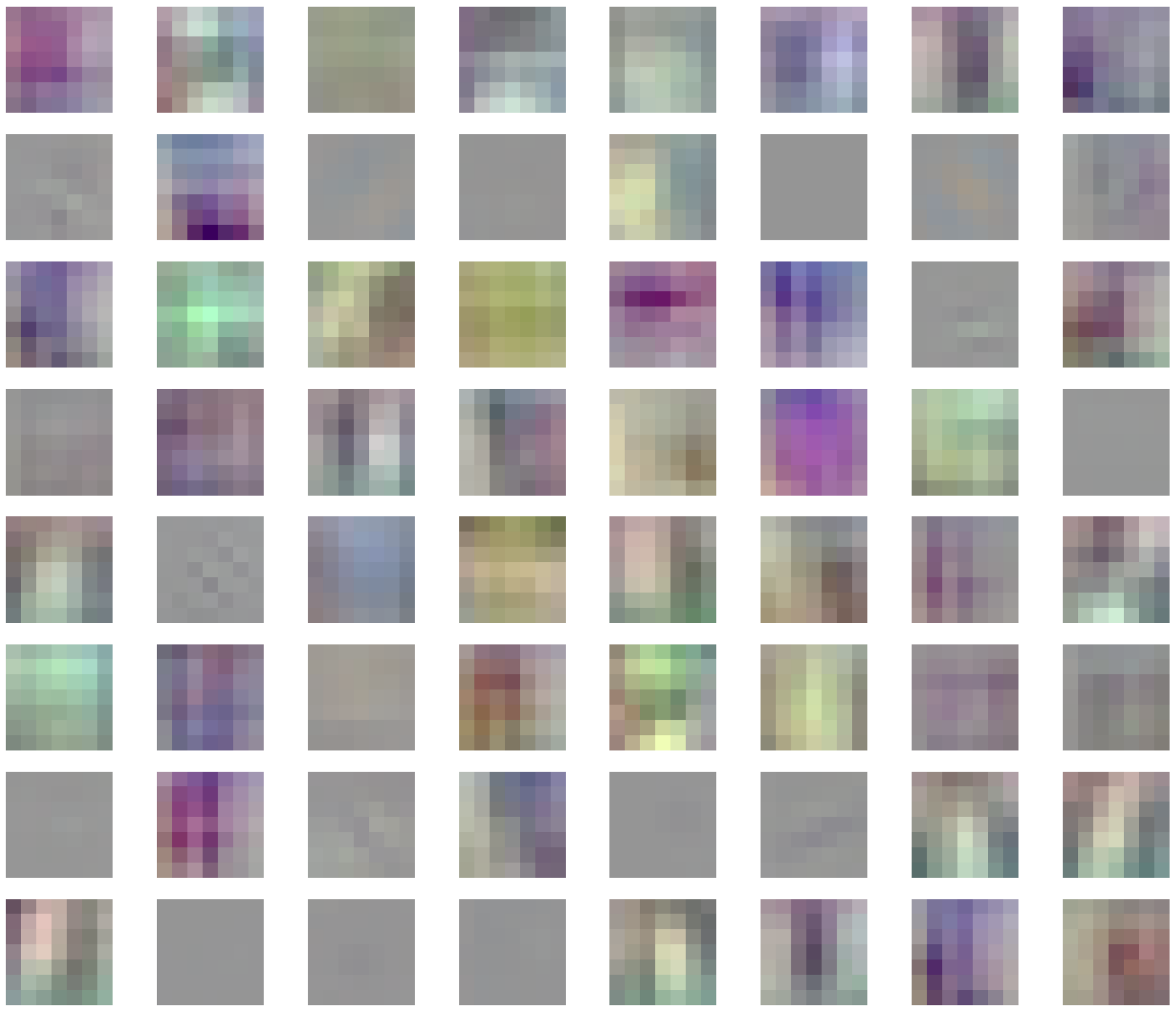


Figure 4. Output of the first, 7x7 convolutional layer in Resnet50

As some readers may already know, CNNs typically learn more basic patterns in their earlier layers and only in their later layers do larger patterns/abstract concepts emerge. As this is the very first layer, this output is in line with this notion.

I cannot understate that experimenting with and understanding these layers that compose the CNN is a vital part of a full scale project. Despite not having time to delve into this on my first pass, modifying and experimenting with these out of the box layers as well as custom layers will be a part of future work.

Evaluation Strategy, Hyperparameters and Validation:

Although I was not able to obtain a fully effective solution for Object Detection in my first pass, I want to briefly touch on the metric specific to this task which is Intersection Over Union (IoU). IoU is simply a measure of the magnitude of overlap between two objects. Lyft's evaluation metric for the competition was Mean Average Precision at different IoU thresholds ranging from 0.5 to 0.95. This means that, at a threshold of 0.5, a predicted object is considered a "hit" if its IoU with a ground truth object is > 0.5.

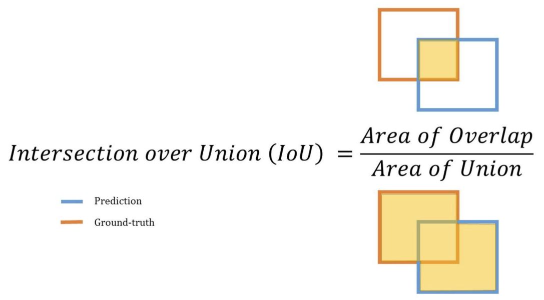

Intersection over Union (IoU) = Area of Overlap / Area of Union
Prediction
Ground-truth

Figure 5. Lyft 3D Object Detection Kaggle competition IoU illustration (https://www.kaggle.com/competitions/3d-object-detection-for-autonomous-vehicles/overview)

IoU will be employed in the next phase of analysis once I have a sufficient Object Detection workflow.

Image Classification: Training all nine models with 10 epochs each took a few hours on my GPU. The inference I then performed on the Validation set for all nine models only took about 15 minutes. I was able to achieve decent results on my Validation set for Classification. I iterated through different combinations of learning rate and batch size when training the Resnet50 models, and I put those models through a validation loop with corresponding hyperparameters to see how they do with Inference on unseen data. Below is a heatmap of the results in terms of F1 Score:

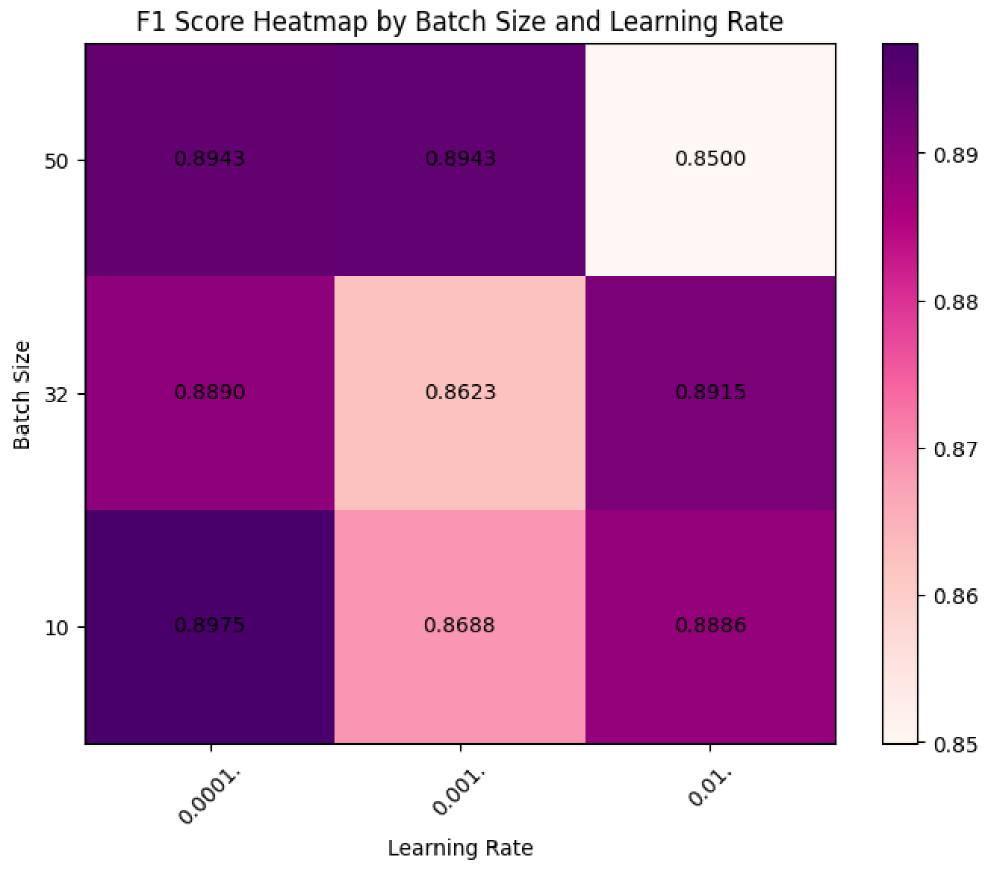


Figure 6. F1 Scores for Validation Set Inference. Each block is a Resnet50 model trained on the corresponding ith-jth hyper parameters batch size and learning rate.

One of the results I was most excited to dig into was the categories that made up most of the false positives (i.e. what did the model think is a pedestrian that actually is not). While I did not break down false positives by individual model, I did aggregate image instances that were false positives for *all nine models* involved in Inference on the Validation set. Below is a summary of the most common categories misclassified as pedestrian:

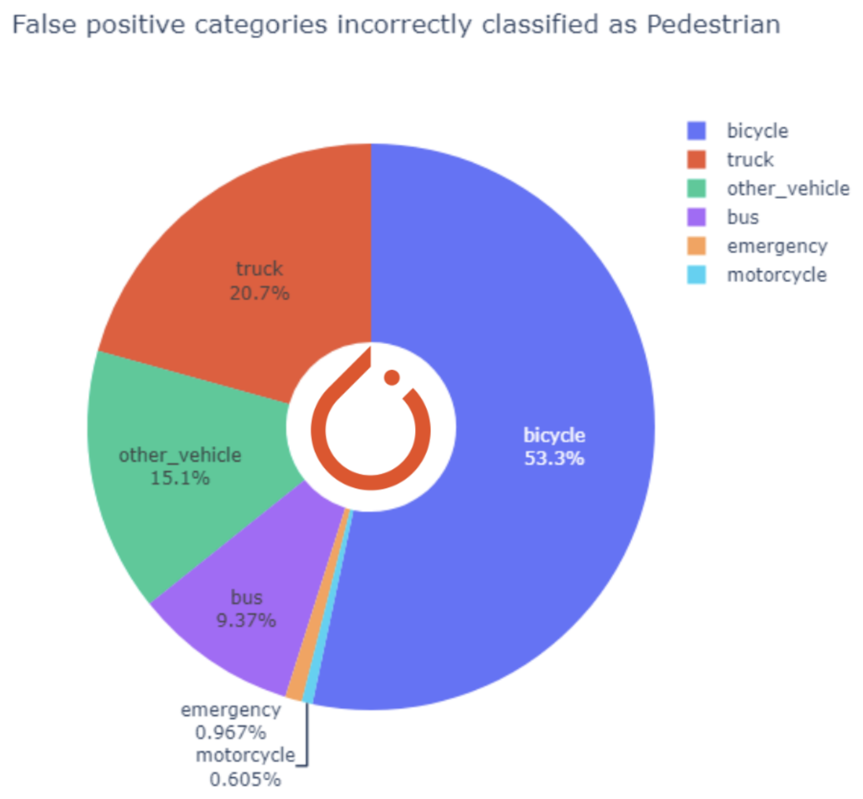
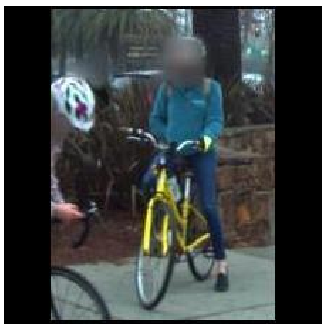
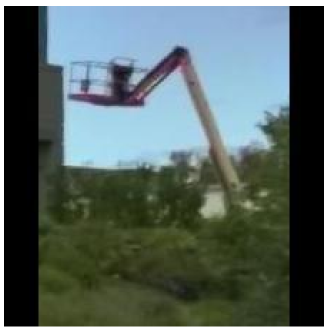
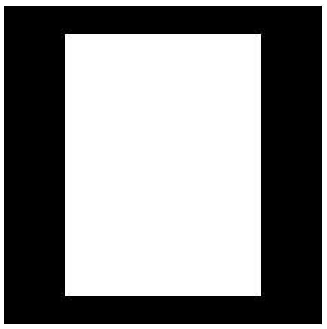


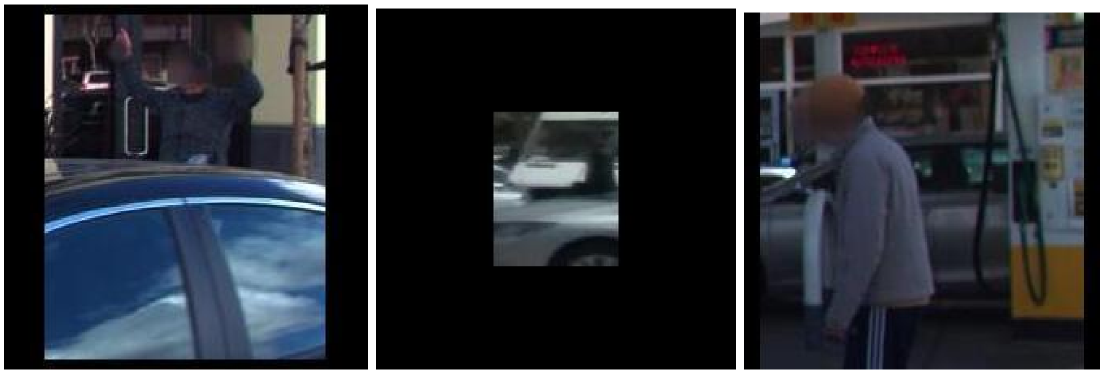
Figure 7. Of our nine Resnet50 models, these are the proportions of categories falsely labeled as positive (pedestrian) that <u>*occurred for all nine*</u> models.

Immediately, it is somewhat comforting to see that "bicycle" is the most commonly misclassified category as pedestrian. Commonsensically, we can understand why this might happen, as the majority of bicycles captured indeed have pedestrians sitting directly on them. This is also true for motorcycles, however motorcycles consist of a significantly smaller proportion of object categories captures than bicycles to begin with. Here are a few of the images that confused the model into thinking there was a pedestrian in the image:

The lefthand image of category "bicycle" is one such case of the issue discussed above. The middle image of category "other_vehicle" is less obvious, although one could make suppositions as to why this wasn't clearly the negative or positive case. The third image on the right of category "truck" is very confusing as it is essentially blank. With approximately 150,000 images collected in real world scenarios, it's not surprising that some images got into the dataset that should probably have been filtered out. Nonetheless, it is interesting to speculate why this ended up positive rather than negative.

Aside from false positives, here are some false negatives that fooled all nine models:



The left hand image of category "pedestrian" is a case where we only have half of the pedestrian visible. The right hand image of category "pedestrian" clearly has a pedestrian at the forefront and a car behind them. The middle image is probably the most difficult. I am not precisely sure myself, but I believe there is a pedestrian pictured with cars both in front and behind them…this would be tricky for most models I'd imagine. This brings us to our final **caveat**. The bad news is that, in this data, sometimes bounding boxes of different categories overlap in 2D space. This means that this cropped image could exist in both "pedestrian" and "other_vehicle" directories. If this is true, then our model has proven it's worth as it classified a pedestrian despite the ground truth label suggesting otherwise.

Broader Impacts/Ethics:

In the self-driving/autonomous vehicle space, ethical considerations are not hard to come by. In the 2008 study titled "Distribution of Road Traffic Deaths by Road User Group: A Global Comparison", H. Naci, D. Chisholm and T.D. Baker outline the global distribution of road traffic deaths by road user groups (pedestrians, bicyclists, motorcyclists, and motorized four-wheeler occupants). They found that breaking down traffic deaths by user group varies widely and, for pedestrians, it varies from near 55% in some developing countries down to 15% or less in developed countries.

More recently in 2023, two Norwegian University of Science and Technology graduate students (Aksel Lunde Aase and Mathias Wold) published an excellent thesis paper titled "Exploring Domain Shift with Autonomous Driving Agest in Simulated and Real Environments". Here is an excerpt from their Abstract: "Studies show that road traffic accidents are the leading cause of death for people aged 5-29 years old worldwide, with the number of road traffic deaths in 2016 reaching 1.35 million. Human driving errors are also the cause of 92% of road accidents.” They noted additional benefits of reducing transportation costs, increased productivity during travel, and improved mobility options for non-drivers.

I have always felt that deaths due to road incidents will be viewed by future societies as a totally preventable, wild west-esque state of existence. It is largely something that is difficult to control for as an individual unlike many other daily choices we make in an effort to avoid danger. Individuals can elect to avoid a dangerous career and to live a healthy lifestyle, but it is much harder to live a modern existence while avoiding roadways altogether. I genuinely believe that the issue of roadway injuries and deaths is one of the most significant and overlooked problems in the modern era as an absurd number of associated fatalities every year become normalized. As such, I feel Autonomous Driving should be approached with caution but should simultaneously continue being developed and advanced.

Discussion/Future Work:

Before we conclude the post, I want to review our caveat count as there were a few important ones and I want to make sure they are understood before people may use my methods.

- **Caveat 1&2:** Any two distinct images in the dataset could be nearly identical if they were taken seconds apart or if they were taken in the same location at a similar time but different day. It’s also worth noting that all images were taken in the same city. It’s feasible that the model might not perform as well in more distinctly different environments.
- **Caveat 3:** In order to manage scope, we did not perform image augmentation largely due to the orientation (sky up ground down) being the same across images. As it stands, the model might not pick up on a pedestrian lying down on the road or a bench etc. This will be considered in future iterations.
- **Caveat 4:** When we are only dealing with our 2D bounding boxes, sometimes these boxes overlap each other in 2D image space. In the image processing pipeline, parts of an object from category 1 may be inadvertently caught within the box for an object of category 2. This is an oversite. However, if your model is strong, this may cause a false negative result while also revealing the models ability to capture the “photo-bombing” category.

This personal project has been my first brush with the field of Autonomous Vehicles, but the issue has been an interest of mine for a long time. Additionally, this was my first exposure to data preparation, model training and evaluation in the deep learning space in general and for CNNs in particular. Pytorch is a fantastic library, and it allowed me to very quickly start practicing

Image Recognition and Object Detection tasks. However, I certainly feel that exploring the model layers and architecture is a vital step, and I was not able to perform this step within the time allotted.

In the next iteration, my main objective will be to get an Object Detection workflow in order. I'll likely end up using Tiling as previously mentioned, and will stick with IoU as my starting point for assessment. Once I have Object Detection in a primitive but sufficient state, I'm most excited to start tinkering with the Resnet50 and study its layers. I imagine it is also worthwhile for every novice to create their own model from scratch as a learning exercise, so this is also part of the game plan for down the line.

The Kaggle competition made use of Lidar, and most submissions I saw either used both Lidar and imagery or only Lidar. Using imagery exclusively is not likely to open the gates for any high level of AV, and I understood this initially. Once I am more well versed in the image processing and CNN space, I look forward to integrating Lidar into the mix and seeing what can actually be done with both of these data modalities combined.

Statement of Work: I, Chris Sternberger, performed all work being on the solo Capstone team Torch Traffic.

Work Cited:

1. *Lyft 3D Object Detection for Autonomous Vehicles.* Kaggle, www.kaggle.com/competitions/3d-object-detection-for-autonomous-vehicles/overview. Accessed 12/11/2023
2. Aase, Aksel Lunde, and Mathias Wold. *Exploring Domain Shift with Autonomous Driving Agents in Simulated and Real Environments*. MS thesis. NTNU, 2023. https://ntnuopen.ntnu.no/ntnu-xmlui/handle/11250/3093954 Accessed 12/11/2023
3. Naci, H., Dan Chisholm, and Timothy D. Baker. "Distribution of road traffic deaths by road user group: a global comparison." *Injury prevention* 15.1 (2009): 55-59. https://injuryprevention.bmj.com/content/15/1/55.short Accessed 12/11/2023