Parker Erickson
CSCI 8980
Final Report

Abstract

In this work, we introduce Contrastive Point Cloud-Image Pretraining, a method that aims to train a model to embed images and their corresponding point clouds to the same point in the embedding space. Using a self-driving car dataset, we train image and point cloud encoders and evaluate on a sensor validation task. This task could be applied to human in the loop self driving systems. Additional tasks could leverage the sensor embeddings produced by this model in the future.

Introduction

The need of sensor validation and verification in field robots, especially autonomous cars, is essential to them operating safely. This project aims to develop a method for validation of input point cloud and image data using a deep learning algorithm. The algorithm is composed of two sub-models, one for images and the other for point clouds, that produce highly similar embeddings if the inputs correspond to one another through a process known as contrastive learning. This idea is similar to the "Arguing Machines" concept introduced in [1], as it is using two neural networks that cross-check the output of each other and put humans in-the-loop if there is a disagreement. For this project, we will use the Oxford RobotCar Dataset [2]. While this algorithm could be applied to a much wider range of applications than self-driving cars, the RobotCar Dataset was chosen due to the amount of data available and well-documented SDK.

The approach is largely based on OpenAI's CLIP [3] model, which learns joint embeddings for images and captions. CLIP showed that learning pairs of images and captions produces state of the art quality in representation learning of the image and text. This property is extremely useful, as high-quality embeddings can be then used directly in tasks such as cosine similarity between embeddings. The CLIP model consisted of an image model, such as ResNet-50 [4], and a transformer-based text model. Both of these models have a linear layer output that was then used to create a cosine similarity matrix between all of the embeddings within the batch. The loss function then compares this matrix to a target matrix, in order to maximize the cosine similarity between the correct pairs of embeddings, while minimizing the cosine similarity between incorrect pairs. In this project, we will replace the text model with a Graph Neural Network, a model that operates on point clouds.

Convolutional Neural Networks have been used extensively in computer vision tasks, and that trend holds in robotics, especially autonomous cars. [5] uses a CNN to learn an end-to-end self-driving model. We will be using a CNN to encode the image data in the model.

Graph Neural Networks have been used on point clouds due to their ability to take a flexible number of inputs that are not regularly arranged in a grid. The architecture of PointNet [6] has been expanded upon to perform object detection using an attention mechanism [7]. Other variations of GNNs such as Graph Convolutional Neural

Networks (GCNs) and Graph Attention Networks (GATs) have also been evaluated for point cloud processing applications [8].

Data

The project uses the Oxford RobotCar Dataset, which contains both point cloud data collected from a LIDAR, as well as camera data. Specifically, the training set was the data collected on July 14, 2014, consisting of 18,580 image/point cloud pairs. The testing set was collected on the same date, consisting of 4,645 image/point cloud pairs. Neither of these datasets were recorded at night, so future work can include training the model on more data, including with different environmental conditions.

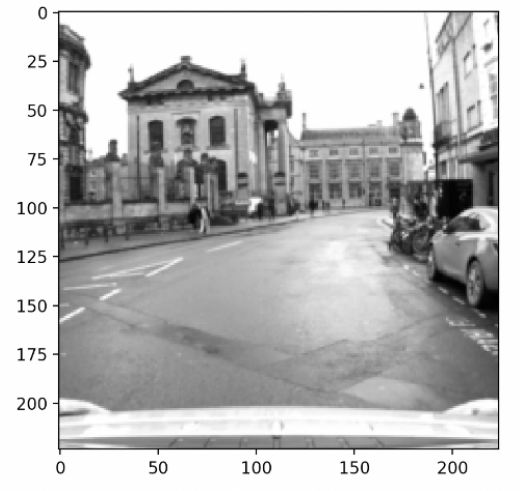

Fig. 1: Image from dataset.

The data was processed so that each frame of video collected by the front-facing camera had a LIDAR point cloud associated with it. There are many LIDAR sensor readings between images, so the dataset is preprocessed such that each image has a set of sensor readings associated with it, forming a more complete point cloud compared to one LIDAR reading. The point clouds are processed into graphs by using the k-Nearest Neighbor method, with the value k = 6. Further processing techniques on the point cloud could be an avenue of future work.

Point Cloud Models

We will evaluate three different point cloud models: PointNet, Graph Attention Networks, and Graph Convolutional Networks.

PointNet [6] was introduced to perform point cloud segmentation and classification, and works off of a set of (x,y,z) coordinates. The architecture is notable because it operates on a set of unordered points, instead of a regular, structured input that architectures such as CNNs rely upon. The architecture is permutation-invariant, meaning any order of the same input data will result in the same classification. The global representation of the point cloud is produced by a pooling function. A hidden layer size of 128 is used in both the original paper and this implementation. The output layer is also of dimension 128.

Graph Convolutional Networks were introduced in [9], originally for vertex classification. However, we can perform a pooling operation on the vertex representations within the graph to generate a graph representation of the data. To be consistent with the PointNet architecture, the hidden layer size is 128, with an output size of 128.

Graph Attention Networks were introduced in [10], and added the concept of an attention mechanism to graph neural networks. 4 attention heads are used in this configuration. Like with the GCN and PointNet, a hidden layer size of 128 is used with an output dimension of 128. For all point cloud models, we evaluate both mean and max pooling functions.

Vision Models

For this project, we will be using ResNet18, as it is available as a pre-trained network and easily accessible from the PyTorch model hub. Additionally, it is computationally lighter than some of the alternatives such as vision transformers like ViT or the modified ResNet50 used in the original CLIP paper [3]. Two model variations are tested: one with pre-trained weights from ImageNet, and one with randomly initialized weights.

Putting it All Together

Each model has an n-dimensional output that serves as that sensor's embedding. The image model embedding might not be the same dimension as the point cloud model, so in order to compute the cosine similarity between embeddings, they need to be mapped into the same dimensional space. To do this, a linear projection layer is added to the output of both models, standardizing the joint embedding dimension of 256. This linear projection layer has a dropout parameter that can be tuned and a ReLU activation function.

Once the embeddings are in the same space, the cosine similarity between the outputs of the two models can be computed. To do this, the fact that data is passed to the model in a mini-batch fashion is taken advantage of. The cross product between the output of the image model with the output of the point cloud model. This results in a matrix that contains the pairwise-similarity between every datapoint in the batch.

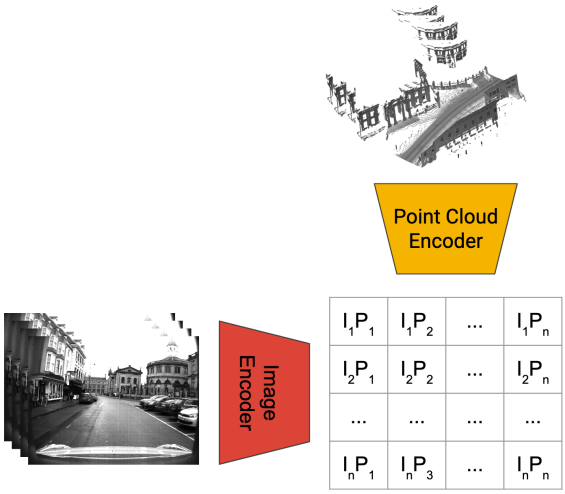

Fig. 2: Model Architecture. A batch of point cloud-image pairs are encoded, and then a similarity matrix is created by performing a dot product between each item in the batch.

Two more similarity matrices are calculated for the batch as well. Both the vision and point cloud models output's are compared against themselves in order to determine how similar data points within the batch are to each other. The average similarity between data points within the batch is then determined by averaging the

point cloud and vision similarity matrices together. This is used as the target in the loss function.

The loss function compares the cross-model similarity to the average similarity for each batch. Formally, the loss function is:

$$L_{PC} = \text{CrossEntropy}(IG^T, (II^T + GG^T)/2)$$

$$L_{Image} = \text{CrossEntropy}(IG^T, (II^T + GG^T)/2)$$

$$L_{Complete} = 1/n\ \Sigma(L_{Image} + L_{PC})/2$$

Where I is the image model output, and G is the point cloud model output. L_{PC} represents the loss function of the point cloud model.

We use the Adam optimizer, with Cosine Annealing learning rate scheduler, with a learning rate of 0.0005 over 400 epochs to train the model. Different batch sizes were evaluated, with the general trend that larger batch sizes result in higher accuracy, in agreement with the original CLIP paper. Unfortunately, larger batch sizes lead to higher memory utilization, making the model difficult to train on a single GPU machine.

Evaluation

Model performance is evaluated by the test set F1 Score of determining if an image and point cloud pair is correct. F1 Score was chosen as the main evaluation metric due to the imbalanced nature of the similarity matrix - the only correct pairs are on the diagonal, while the rest of the similarity matrix consist of incorrect pairs. To make the prediction, the cosine similarity between the output embeddings of the two models will be determined, and if the score is greater than a certain threshold, the pair will be deemed a match. Evaluated are 6 different model architectures, between the 3 different point cloud models and pretrained/randomly initialized image models.

Final Validation Loss - Mean Pooling

	PointNet	**GAT***	**GCN**
ResNet18 - Pretrained	3.365	1.781	3.407
ResNet18 - Not Pretrained	3.355	1.176	3.32

Validation F1 Score - Similarity Threshold 0.5 - Mean Pooling

	PointNet	**GAT***	**GCN**
ResNet18 - Pretrained	0.1771	0.3333	0.1587
ResNet18 - Not Pretrained	0.1728	0.5455	0.1787

Validation F1 Score - Similarity Threshold 0.90 - Mean Pooling

	PointNet	**GAT***	**GCN**
ResNet18 - Pretrained	0.1714	0.3333	0.1548
ResNet18 - Not Pretrained	0.1733	0.400	0.1609

Final Validation Loss - Max Pooling

	PointNet	GAT*	GCN
ResNet18 - Pretrained	3.320	1.789	3.293
ResNet18 - Not Pretrained	3.411	OOM	3.370

Validation F1 Score - Similarity Threshold 0.5 - Max Pooling

	PointNet	GAT*	GCN
ResNet18 - Pretrained	0.1760	0.4000	0.1832
ResNet18 - Not Pretrained	0.1500	OOM	0.1651

Validation F1 Score - Similarity Threshold 0.90 - Max Pooling

	PointNet	GAT*	GCN
ResNet18 - Pretrained	0.1737	0.4000	0.1841
ResNet18 - Not Pretrained	0.1500	OOM	0.1638

* Note: The GAT-based models were trained with a batch size of 32, opposed to 64 for PointNet and GCN models due to memory constraints. OOM indicates the training process halted due to an out-of-memory error.

The GAT-based models outperform the other point-cloud models in both final validation loss and F1 score, which is probably due to the extra parameters the attention mechanism uses compared to the other point cloud encoders. A downside of the extra parameters is that the models were difficult to train - sometimes we would get an out of memory error halfway through the training process.

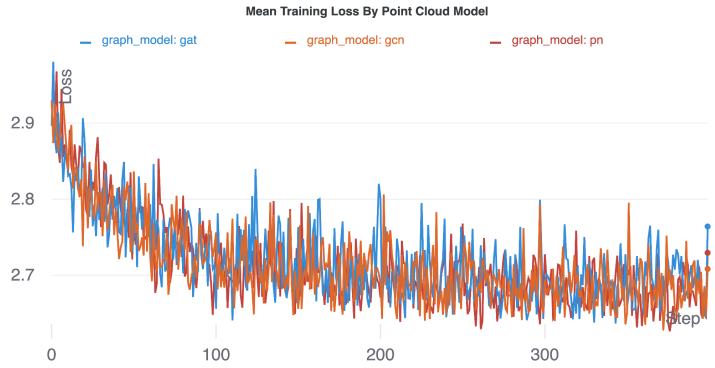

Fig. 3: Training Loss Curve by Point Cloud Model Type. Note that the curve is not smooth, an artifact of small batch sizes.

These initial quantitative results should be taken with a grain of salt due to the extreme variation between loss values between epochs. As seen in Fig. 3, all models experienced a very unsmooth training process, due to the limited batch size. All results were the values taken at the end of the training process, and therefore could have wide ranges of variation.

The results of the GAT models do show promise of what could be achieved with more model parameters, and it has been shown that it is not beneficial to start with a pre-trained ResNet model. This would be a very good starting point for future experiments.

Qualitative Results

Shown below is a side-by-side comparison of an image-point cloud pair and the similarity between the two embeddings produced by the trained model.

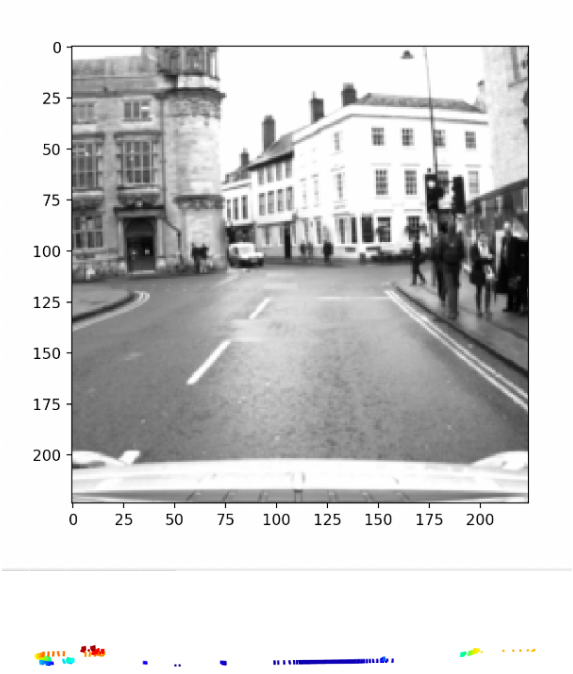

Fig. 4: Image/Point Cloud Pair. A GCN-based model inferred that this pair had a similarity of 0.83.

As shown by the figure above, the point cloud data is very sparsely populated, and does not cover the full view of the image frame the cloud is associated with. This could be very detrimental to the model training, as each image will have a point cloud that is pretty ambiguous in terms of where the point cloud was sampled from in the space. If increasing model parameters does not solve the performance issues presented, this would be the next thing to investigate. The fix to this would be to sample the LIDAR for a longer period of time, instead of utilizing each frame of video. Unfortunately, this comes at a cost. First off, a larger point cloud means more data inside of a batch, increasing memory consumption. Secondly, and possibly more importantly, is that if this is utilized in a sensor validation task, collecting enough LIDAR data would be a bottleneck when running in real-time. If a prediction about sensor correctness can only be made one or two times a second, the prediction might be too late for a human to intervene.

Improvements

Two big improvements and further investigation can be made to the work presented above, and were not addressed due to lack of time and/or computational power. Firstly, it appears as though the model is generally underfit. This means that increasing the number of parameters could be beneficial. To do this, the vision model could be converted to something like ResNet-50 or a vision transformer such as ViT. Additionally, the hidden size in the point cloud models could be increased to add extra capacity. This was not tested due to compute limitations, namely the GPU memory required to store larger models.

Furthermore, increasing the batch size would be expected to produce better training results. This is due to gaining a better representation of the entire dataset in a batch. This better representation would decrease the variance between batches, improving training performance. As seen in Fig. 3, the loss curve while training is very spiky, which can be attributed to high

variance between batches during the training process. Larger batch sizes have not been tested due to GPU memory constraints. All experiments were performed on a single GTX 2080 TI due to package/driver issues on a multi-GPU environment.

If these improvements don't have an appreciable impact on the accuracy of the model, another thing to look at would be expanding the point cloud that is input into the model, but that comes at the expense of more memory consumption and the amount of inference time.

Future Work

An avenue of further research into this project would be to evaluate a wider range of vision models, such as ViT and other CNN architectures, which should help performance due to the underfitting issue stated above. Additionally, further hyperparameter tuning and architecture search could be performed, as these experiments will not be carried out due to lack of compute resources. Additional work could include using the image embedding to generate the pointcloud for depth estimation using a single camera or investigating the possibility of pretraining the model on a dataset and evaluating the transferability of this model from one dataset to another.

Conclusion

In this work, we introduced Contrastive Point Cloud-Image Pretraining, a method that aims to train a model to embed images and their corresponding point clouds to the same point in the embedding space. Using a self-driving car dataset, we trained image and point cloud encoders and evaluated on a sensor validation task, with promising, but not good results. A majority of these issues stemmed from GPU memory consumption, and the lack of multi-GPU training due to driver/package conflicts. The sensor validation task could be applied to human-in-the-loop self driving systems.

All code for the project can be found here: https://github.com/parkererickson/clip-graph

References

[1] L. Fridman, L. Ding, B. Jenik and B. Reimer, "Arguing Machines: Human Supervision of Black Box AI Systems That Make Life-Critical Decisions," 2019 IEEE/CVF Conference on Computer Vision and Pattern Recognition Workshops (CVPRW), 2019, pp. 1335-1343, doi: 10.1109/CVPRW.2019.00173.

[2] W. Maddern, G. Pascoe, C. Linegar and P. Newman, "1 Year, 1000km: The Oxford RobotCar Dataset", *The International Journal of Robotics Research (IJRR)*, 2016.

[3] Alec Radford, Jong Wook Kim, Chris Hallacy, Aditya Ramesh, Gabriel Goh, Sandhini Agarwal, Girish Sastry, Amanda Askell, Pamela Mishkin, Jack Clark, Gretchen Krueger, Ilya Sutskever, "Learning Transferable Visual Models From Natural Language Supervision," *CoRR*, March 2021.

[4] K. He, X. Zhang, S. Ren and J. Sun, "Deep Residual Learning for Image Recognition," *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2016, pp. 770-778, doi: 10.1109/CVPR.2016.90.

[5] Mariusz Bojarski, Davide Del Testa, Daniel Dworakowski, Bernhard Firner, Beat Flepp, Prasoon Goyal, Lawrence D. Jackel, Mathew Monfort, Urs Muller, Jiakai Zhang, Xin Zhang, Jake Zhao, Karol Zieba, "End to End Learning for Self-Driving Cars," *CoRR*, April 2016.

[6] R. Q. Charles, H. Su, M. Kaichun and L. J. Guibas, "PointNet: Deep Learning on Point Sets for 3D Classification and Segmentation," *2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2017, pp. 77-85, doi: 10.1109/CVPR.2017.16.

[7] A. Paigwar, O. Erkent, C. Wolf and C. Laugier, "Attentional PointNet for 3D-Object Detection in Point Clouds," 2019 IEEE/CVF Conference on Computer Vision and Pattern Recognition Workshops (CVPRW), 2019, pp. 1297-1306, doi: 10.1109/CVPRW.2019.00169.

[8] Y. Li et al., "Deep Learning for LiDAR Point Clouds in Autonomous Driving: A Review," in IEEE Transactions on Neural Networks and Learning Systems, vol. 32, no. 8, pp. 3412-3432, Aug. 2021, doi: 10.1109/TNNLS.2020.3015992.

[9] T. Kipf, M. Welling, "Semi-Supervised Classification with Graph Convolutional Networks," *ICLR,* February 2017

[10] P. Velickovic, G. Curcurull, A. Casanova, A. Romero, P. Lio, Y. Bengio, "Graph Attention Networks", *ICLR*, January 2018