ECEP 596, Autumn 2019 Final Project
Predicting 6DoF Pose From 2D Images - Kaggle Peking Competition

Drew Clark

December 8, 2019

1 Introduction

Autonomous driving is a popular computer vision topic in both research and industry. One of the more difficult challenges for autonomous vehicles is being able to navigate real-world traffic. In order to navigate traffic, the car must be aware of all the position and pose of all vehicles around it. The objective of this project is to predict the absolute pose of vehicles in real-world traffic environments using only RGB images and a library of 3D car models. The project is provided by a competition from Peking University along with Baidu's Robotics and Autonomous Driving Library and is hosted on Kaggle. I chose this project because I have a strong interest in computer vision and machine learning and find autonomous driving to be a very interesting challenge. I've yet to attempt a public Kaggle competition and found this to be a great opportunity as the project falls under computer vision, while also being a competition of strong interest to me.

2 Related Work

6DoF pose estimation is a topic of heavy research recently and many new techniques are being introduced each year with great success such as YOLO6D [7], PVNet [5], PoseCNN [9], CenterNet [11] and DeepIM [3].

I chose to use CenterNet as a basis for this project due to it's performance on the KITTI dataset combined with the speed at which it runs. This paper focuses on the techniques used within CenterNet, such as the backbone architecture, normalization layers, and non-linear activation layers. Furthermore, data augmentation is experimented with, as well as variable loss in order to try to decouple the mask loss and the regression loss such that one won't overfit while the other is still learning. These techniques are discussed further in the methods section, as well as some pose optimization methods and results of these experiments are discussed in the proceeding "Experiments and Results" section.

3 Data Overview

The dataset used in this competition was provided by Baidu's Robotics and Autonomous Driving Library along with Peking University. The dataset consists of 4262 images consisting of 49684 labeled instances of vehicles. Each vehicle is labeled by a center point (x, y, z), pose (yaw, pitch, roll), and model id (i.e. Audi a6). Additionally, the competition provides the camera intrinsic parameters and 79 3D car models. Figure 1 shows an overview of the data.

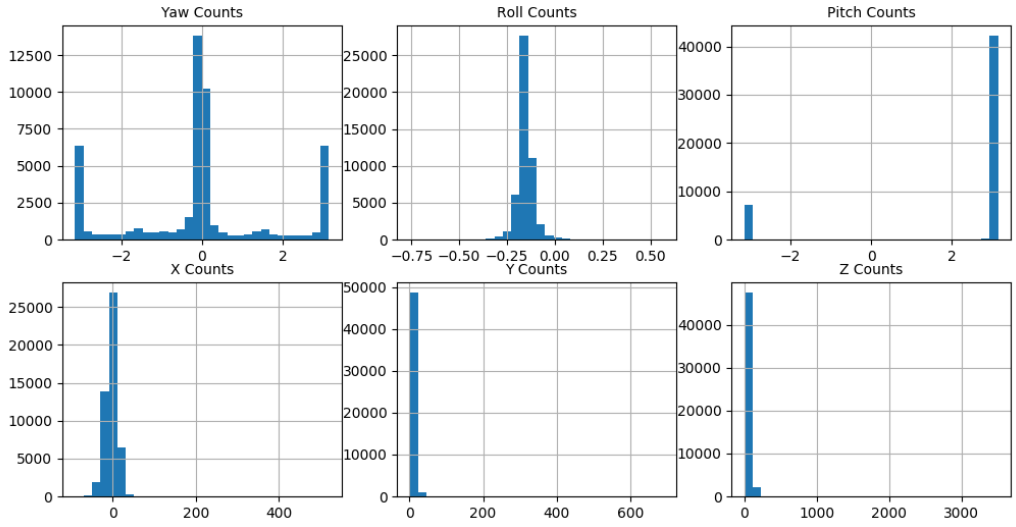

Figure 1: Histograms of data labels

Figure 2 shows an example of one of the training images provided for the competition, as well as some examples of specific vehicles we are trying to predict the pose for. Each image has multiple vehicle instances, some of which are very clear, while others are obstructed by other vehicles or object, have very poor resolution due to being far away, or are in spots with very bad lighting.

Figure 2: Sample of image showing various challenges including occlusion, resolution, and lighting

4 Methods and Experiments

The main method used in this project was CenterNet, which I implemented from some start code on Kaggle. Testing was performed to find the best backbone, normalization layer, non-linear activation layer, data augmentation, and loss weighting. The backbones tested were ResNet18 [1], ResNet101 [1] and ResNeXt101-32x8d [10]. The CenterNet paper uses both ResNet18 and ResNet101, I chose to use ResNet18 because it was the fastest backbone architecture to train of the models architectures used in the ResNet paper. ResNet101 was chosen due to the simplicity of implementing it once ResNet18 was implemented. Finally, ResNeXt101-32x8d was chosen since it was a common architecture used by comparable models, and it allowed a direct comparison of ResNet vs ResNeXt. Due to not having time to train all possible combinations, ResNet18 was used to test the normalization layer, non-linear activation layer, data augmentation, and loss weighting. Ideally and exhaustive test would be performed to find the best combination, but due to each model taking significant time to train and the length of the project, the different techniques were instead tested individually with the best technique moving forward to the next round for testing the next technique. The final round tested the larger backbones with the best configuration found from the previous rounds and compared to ResNet18. Figure 3 shows the the testing structure used to find the best results.

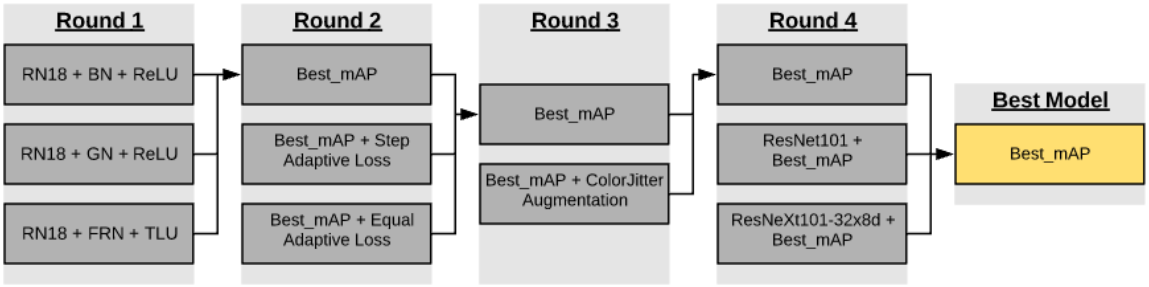

Figure 3: Overview of experiment flow. Each column is tested against each other and the best version moves to the next round of testing.

4.1 Normalization and Non-Linear Activation Layers Comparison

Below I will discuss the differences in the tested normalization layers and non-linear activation layers tested in round 1.

Normalization techniques between convolutional layers is a common technique used by almost all models today. The most common form of normalization is batch normalization. Batch normalization (BN) was introduced in 2015 to address the issue of internal co-variate shift due to the distribution of each layer's inputs changing throughout training [2]. Prior to network-internal normalization techniques, the only way to address this issue was to reduce the learning rate and carefully initialize the parameters of the network. However, reduction of the learning rate directly corresponds with longer training time, making this technique less than ideal. BN addresses this issue by performing internal normalization of convolution layer outputs for each mini-batch during training such that the next convolutional layer input is normalized. Building on this technique, group normalization (GN) was introduced in 2018 to help address the issue of applying batch normalization to small batches [8]. Unlike BN, GN doesn't normalize over each mini-batch, but normalizes over each "group". A group is a subset of a layer, which can also be thought of as a subset of the features output by a layer. The number of groups is a hyperparameter, which I arbitrarily set as 16 for this project. By normalizing over the features, GN isn't influenced by batch size and performs significantly better when using a batch size of 8 or less. The third activation layer I experimented with was Filter

Response Normalization (FRN) [6] which also recommends using a different non-linear activation layer called a Threshold Linear Unit (TLU) as opposed to the most common Rectified Linear Unit (ReLU) which I used for the BN and GN normalization layers. Similar to GN, FRN aims to reduce the batch size dependency without sacrificing accuracy for larger batch sizes. The main differences in FRN are that it doesn't remove the mean prior to normalization, like BN and GN do, normalization is done on a per-channel basis such that all features have the same relative importance, and FRN does global normalization over the spatial extent. In order to to address the lack of mean centering in FRN, TLU is used as the activation layer which simply augments ReLU with a learned threshold τ which can be expressed as $ReLU(y - \tau)$. The FRN paper reports an increased AP of at least 0.3-0.5% for batch sizes 8, 4, and 2 compared to GN, and significantly more improvement compared to BN.

4.2 Variable Weight Loss Comparison

In Round 2, I tested various ways to weight the loss between the model finding the center point mask and the regression to the pose values. My baseline model is such that the loss is the summation of the mask loss and regression loss, however I have each loss weighted such that the first 6 epochs only uses the mask loss, and after that the total loss is $(mask\_loss\_weight) * mask\_loss + (1 - mask\_loss\_weight) * regression\_loss$. After training a few models, I noticed that the mask would begin to overfit before the regression, so I introduced a step adaptive loss version and a equal adaptive loss version, both of which take the validation loss and adjust the weights of the regression and mask. The pseudo code for the step adaptive loss is shown in Algorithm 1, the purpose is if one loss is overfitting while the other is still learning, shift the emphasis towards the learning portion.

Algorithm 1 Pseudo Code of Step Adaptive Loss

```
if mask_loss > prev_mask_loss AND regr_loss <= prev_regr_loss then
  if loss_weight > 0.0: then
    loss_weight− = 0.1
  else
    loss_weight = 0.0
  end if
else if mask_loss <= prev_mask_loss AND regr_loss > prev_regr_loss then
  if loss_weight < 1.0: then
    loss_weight+ = 0.1
  else
    loss_weight = 1.0
  end if
end if
```

The equal adaptive loss version is shown in Algorithm 2, the purpose is similar to step loss, but instead equalizes the loss such that when one of the losses increases, the weight will decrease compared to the weight of the other loss. While experimenting with the step adaptive loss, I noticed that the regression loss during my training is an order of magnitude less than mask loss and I found significant increases in performance once the loss weight was at 0.1 such that they were relatively equal.

Algorithm 2 Pseudo Code of Equal Adaptive Loss

```
if mask_loss > prev_mask_loss OR regr_loss > prev_regr_loss then
  loss_weight = regr_loss/mask_loss
end if
```

4.3 Data Augmentation Comparison

In round 3, I tested a data augmentation technique called ColorJitter. ColorJitter is a function that is part of the torchvision.transforms package that randomly changes the brightness, contrast, saturation and hue of images between a predetermined range that is set as a hyperparameter. For my experiment I used 20% for all categories during training. Additionally during all the experiments I have random horizontal flipping implemented with a probability of 10%.

4.4 Backbone Architecture Comparison

Finally, in round 4, I tested the different backbone architectures with the best combination of techniques found in the previous 3 rounds. The additional backbones used were ResNet101 and ResNeXt101-32x8d, as well as ResNet18 which was used for the previous experiments.

4.5 Training CenterNet

Aside from the techniques being tested, various other techniques were implemented to improve results. AdamW optimization was used instead of the more common Adam packaged in pytorch. AdamW decouples weight decay and l2 regularization in Adam and is reported to improve Adam's generalization performance [4]. ReduceOnPlateau is part of the torch.optim.lr_scheduler package and was used with the validation mAP metric to reduce the learning rate by 25% if mAP hasn't improved over 3 epochs. Early stopping was used with the validation mAP such that the model would stop training if the mAP hasn't improved over 5 epochs. AdamW, ReduceOnPlateau and Early Stopping decrease the importance of choosing the optimal initial learning rate and total epochs. All models had an initial learning rate of 0.001 and maximum epochs of 100 (though many stopped around 20-40). Additionally, a batch size of 4 was used for all ResNet18 models, 2 for ResNet101, and 1 for ResNeXt101_32x8d due to memory constraints. Algorithm 3 describes the pseudo-code for training.

Algorithm 3 Pseudo Code for training CenterNet

```
Split data into train/validation sets, load labels and create data loaders for batch learning
Load CenterNet with backbone, normalization, activation, and AdamW Optimizer
for Epoch in Total_Epochs do
  for Batch in Total_Batches Training do
    Predict x, y, z, roll, pitch, yaw
    Calculate loss, perform gradient descent, and update optimizer
  end for
  for Batch in Total_Batches Validation do
    Predict x,y,z,roll,pitch,yaw
    Calculate loss and mAP
  end for
  if Validation mAP < Best Validation mAP then
    if Validation mAP hasn't improved in 3 epochs then
      Reduce learning rate
    end if
    if Validation mAP hasn't improved in 5 epochs then
      End training
    end if
  end if
  Perform adaptive loss algorithm if using one
end for
Save model, plot results and make submission file on test data
```

4.6 Pose Optimization

Once I have the output from my trained model, I would then like to optimize the pose using feature matching with the 3D models provided by the competition. In order to do this I attempted to train a classifier using ResNet18 pretrained on ImageNet. Using the 3D models, I found the max dimensions for all the vehicles and created regions of interest (ROIs) for each labeled car instance in an image. I then resized the image to 224x224 as used as the input for ResNet18 and trained the model to classify each car.

Once the classifier is trained, I then use CenterNet to detect all the vehicles in the image and provide an initial pose estimation. For each vehicle found in the image, I extract the ROI, apply the classifier to classify the make and model of the car, perform SIFT to extract the descriptors and finally perform RANSAC to match the features with the 3D model of the classified car. This process is shown as pseudo code in Algorithm 4

Algorithm 4 Pseudo Code for end to end prediction of Vehicle Pose

```
Load image to be evaluated
Load CenterNet with backbone, normalization, activation, and AdamW Optimizer
Predict x,y,z,roll,pitch,yaw with CenterNet
for Each car found in image do
    Use bounding box to create ROI
    Reshape ROI to 224x224 for classifier input
    Predict model id of vehicle
    Load 3D model for classified vehicle
    Perform SIFT on ROI
    Perform cv2.solvePnPRANSAC between keypoints from ROI and 3D model
    Use extracted rotation matrix to optimize pose
end for
Make submission file on test data
```

5 Results and Discussion

This section will present the results from the experiments described above and discuss the findings and things that didn't work. The main metric used for evaluation is mean average precision (mAP), however I will also present the number of epochs needed to train, and the training time per epoch in techniques that showed significant difference of these values.

5.1 Normalization and Non-Linear Activation Layer Comparison

For the first round of comparisons, I compared results from the normalization and non-linear activation layers on CenterNet-ResNet18. Table 1 shows the results of this comparison.

Table 1: Comparison of Normalization and Non-Linear Activation Layers on ResNet18

Technique	*mAP*	*Epochs*	*Final LR*
BN + ReLU	0.0920	27	0.00025
GN + ReLU	0.0891	37	0.00025
FRN + TLU	**0.0946**	38	0.000016

Due to large image sizes and deep networks, the maximum batch size I was able to use was 4. As such, I expected the best performance to come by using GN or FRN. FRN did perform the best

by mAP, surprisingly though GN performed the worst. BN was able to train in 11 fewer epochs, but since FRN resulted in the higher mAP, it was decided as the best normalization/activation layer technique for this problem and used on all the proceeding comparisons.

5.2 Variable Weight Loss Comparison

For the second round of comparisons, I compared mask loss and regression loss weighting strategies on the the CenterNet-ResNet18-FRN+TLU model. Table 2 shows the results of this comparison.

Table 2: Comparison of Mask loss vs Regression loss weighting strategies

Technique	*mAP*	*Epochs*	*Mask loss*	*Regr loss*	*Final LR*
Static Loss Weight	0.0946	38	31.07	0.1497	0.000016
Step Adaptive Loss Weight	0.1157	21	32.42	0.1329	0.001
Equal Adaptive Loss Weight	**0.1162**	38	26.50	0.1222	0.000063

The Static loss weight was simply the results carried over from round 1. Both the adaptive weight strategies resulted in significant improvement of mAP. However, step adaptive loss weight resulted in unstable results at the end of training, once mask weight reached 0.0 it's loss began to grow significantly. Equal adaptive loss weight performed consistently, but took 17 more epochs to achieve the high mAP score. Looking at the loss values in the table, we see that equal adaptive loss weight was able to outperform the other methods despite having the most epochs, suggesting that it helped prevent overfitting of the mask problem while allowing the regression problem to continue to learn. Despite the promising results of the adaptive methods, this is an area where I think the model can improve significantly more by freezing weights entirely once one mask or regression begins to overfit and allow the other to continue to learn.

5.3 Data Augmentation Comparison

For the third round of comparisons, I compared using further data augmentation by using ColorJitter. Table 3 shows the results of this comparison.

Table 3: Comparison of Data Augmentation

Technique	*mAP*	*Epochs*	*Mask loss*	*Regr loss*	*Final LR*	*Training Time*
Horizontal Flip Only	0.1162	38	26.50	0.1222	0.000063	**∼12 min/ep**
Horizontal Flip + ColorJitter	**0.1192**	28	15.61	0.1122	0.00025	∼24 min/ep

The horizontal flip only results were carried over from round 2 using the equal adaptive loss weight technique. By adding ColorJitter, I was able to achieve slightly better mAP, and significantly better mask loss, while also improving my regression loss in 10 less epochs. However, the augmentation did end up taking twice as long to train per epoch, so the overall training time took longer. It appears that the additional data augmentation was successful in preventing the center prediction mask to overfit as quickly, allowing the regression of the pose values to continue to learn. Despite the additional time to train, inference isn't increased at all by adding data augmentation since it is only added during training, therefore additional data augmentation should be explored in order to continue to improve the mAP without increasing inference time of the model.

5.4 Backbone Comparison

For the final round of comparisons, I compared using different ResNet variations as the CenterNet backbone architecture. Table 4 shows the results of this comparison.

Table 4: Comparison of Backbone Architecture

Technique	*mAP*	*Epochs*	*Mask loss*	*Regr loss*	*Final LR*	*Batch Size*	*Training Time*
ResNeXt101_32x8d	0.1049	25	41.96	0.5014	0.00025	1	∼39 min/ep
ResNet101	0.1158	28	41.76	0.2296	0.000063	2	∼24 min/ep
ResNet18	**0.1192**	28	15.61	0.1122	0.00025	4	∼24 min/ep

The ResNet18 results were carried over from round 3 using ColorJitter. To summarize the previous comparisons, this final round of comparisons used ColorJitter data augmentation, equal adaptive loss weight, FRN layers after each convolution and TLU activation layers for each model. The lighter ResNet18 significantly outperformed the larger networks in this test, although the mAP was only slightly better than ResNet101 the loss was less than half the amount from the larger networks. This result isn't too surprising based on the trends of the previous tests. In the previous comparisons, the model was often overfitting the center predictions, while still learning the regression, and as such using a larger network resulted in more overfitting, as the table shows the higher mask loss. In order to make the larger networks more feasible, either more data should be used, or more data augmentation, especially augmentation that would make the centers harder to learn.

Figures 4 to 6 show the output results on 3 validation figures compared to the ground truth after training on CenterNet with ResNet18 backbone. Overall it appears that the model is doing quite well, it misses a few cars and seems like the center points and pose is a bit off on others but given the difficulty of the data it seems to perform pretty well. Using this model alone, I am currently ranked 82/475, which puts my model in the top 20%.

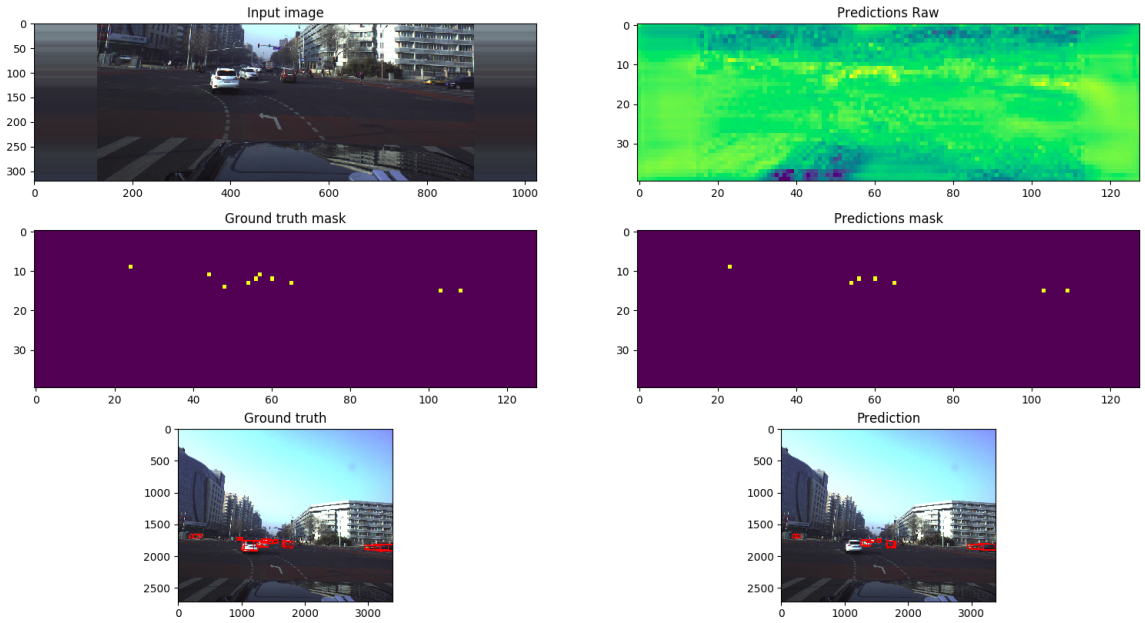

Figure 4: Example of best performing model on validation image. Left column is all ground truth information, while right column is predicted outputs. Going clockwise, *Top Left:* Input image to model. *Top Right:* Mask Prediction from CenterNet. *Middle Right:* Predictions mask thresholded to get center predictions. *Bottom Right:* Original image with 3D predicted bounding boxes overlayed. *Bottom Left:* Original image with 3D ground truth bounding boxes overlayed. *Middle Left:* Ground truth centers.

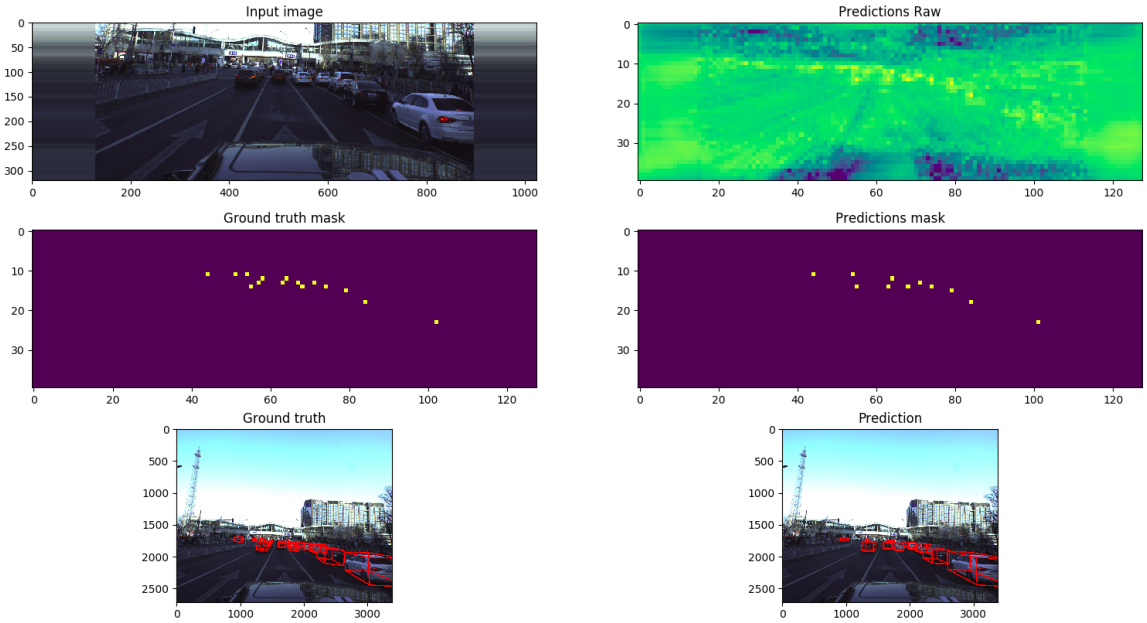

Figure 5: Example of best performing model on validation image. See Figure 4 for details.

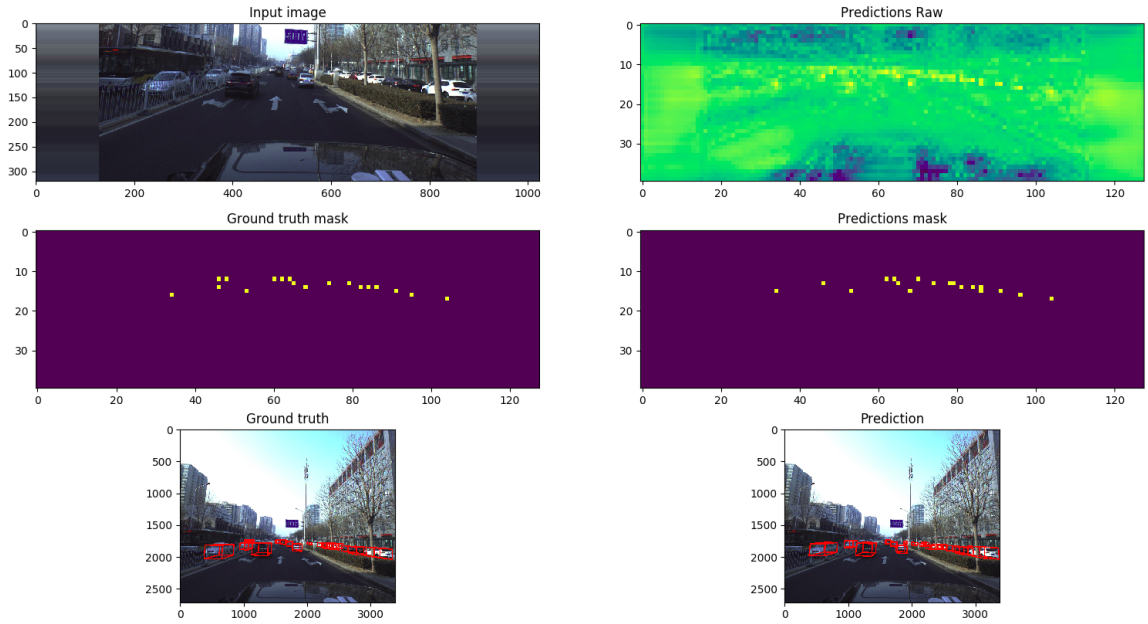

Figure 6: Example of best performing model on validation image. See Figure 4 for details.

5.5 Pose Optimization

With the promising results from CenterNet, the idea for the pose optimization was take the predictions, classify the vehicle type, and then use the 3D models to perform feature matching to get a more precise pose than we were able to get from performing regression in CenterNet.

The first step of this process was to classify the the vehicle type. However, after training the model I was only able to achieve 38% Top1 accuracy. Considering that there are 34 labeled vehicle types in the data, this is much better than random, however the class distribution is imbalanced with one class accounting for ∼22% of the data. I expected much better results since many cars models have pretty distinguishable features, but upon further inspection of the data, I found that the labels provided for vehicle type was very inconsistent. Figure 7 shows an example of three cars that are labeled as Audi Q7's, none of which are Audi's or consistent between each other.

Figure 7: The three cars above are all labeled as an Audi Q7, however the one on the left is an SUV, middle is a sedan, and right is a mini-van.

Upon realizing that the data labels were inconsistent, I decided to scrap the idea of a classifier since I knew I wouldn't be able to achieve good results, and if I had, I wouldn't trust the results anyway. Therefore, my next attempt was to iterate through all 79 provided car models and perform SIFT and RANSAC and use the model that resulted in the most feature matches. I first attempted to do this by creating 2D images for each model using the pose and location provided by CenterNet, and comparing this image with the masked input image. This method was first tested on training data by using ground truth pose and location to try to match a model to a car in the image. Figure8 shows an example of this process on an unobstructed clear vehicle.

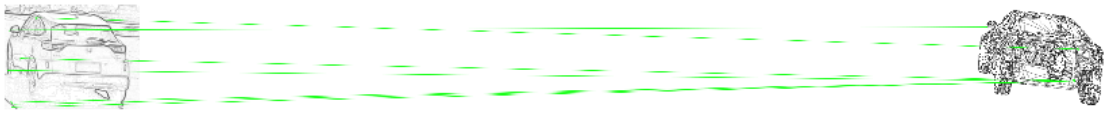

Figure 8: A car with pencil filter applied compared to a 2D wire frame model of the same pose and location as the ground truth data.

Unfortunately, results were very poor. It appears that the the side mirrors were able to be matched, but no other point on the car was correctly matched. Since 2D matching wasn't working well, I then attempted to try to implement 3D to 2D feature matching by using the model directly, but am still working on implementing cv2.solvePnPRANSAC. As another possible alternative, I could try the 2D matching by adding texture first to the model and using the original RGB image and a better mask to just show the car without any background and I may be able to achieve better results by doing this.

6 Future Work

In this section I'll discuss future planned experiments and optimizations that I hope will increase the performance of my model. The official deadline for the Kaggle competition is January 21st and as such there is time for significant future work.

The most obvious place for future work is in finishing pose optimization for 3D to 2D PnP. Aside from the issues of I ran into with finding matches, my 2D to 2D feature matching was very slow as I had to iterate through 79 models. My first priority is to try to see if 3D to 2D feature matching works, and is able to achieve a boost in mAP on the test set. However, once I'm able to get the feature matching to work, the next challenge will likely to make it so that it runs efficiently. Since I am first using CenterNet to predict location and pose of vehicles, any second stage I add to optimize the results needs to perform very quickly in order to be realistic. There are voting techniques for feature matching that could replace RANSAC, since RANSAC is an iterative approach these voting techniques can be much faster. Additionally, I may not need to use all 79 models for matching, it's possible I only need to use models that represent the majority of the vehicles such as one or two SUV models that a capable of matching with all SUVs.

Aside from pose optimization, there is still a lot of room to improve CenterNet as well. My testing showed the largest differences between loss weighting and data augmentation, and both of these areas should be explored further. For loss weighting, I would like to implement my model such that once it notices the mask, or regression, starting to overfit on the validation data it no longer updates those weights and only updates the weights in the areas still improving. As far as data augmentation is concerned, I only explored a couple methods that were easy to implement without having to perform changes to the labels. I would like to implement further augmentation such as

rotation, affine, crop and scale. All of these methods are easy to apply to classification tasks, but for this specific problem, caution needs to be performed to ensure that the labels update appropriately with the image. Additionally, I am currently performing normalization within the model, but I am not normalizing the input images which should help accuracy.

Finally, there were many other areas which I was hoping to explore but didn't have time due to the amount of time it takes to train these models. I would like to try using dilation convolution layers, explore different color spaces, and turn our ground truth masks into Gaussian masks to potentially aid training. In order to make larger networks more feasible, I would also like to experiment with synthetic data using the 3D models to increase the data size and help the models better generalize.

7 Summary and Conclusion

This project attempted to predict the absolute 6DoF pose of vehicles in 2D RGB images in real-world traffic. The dataset was provided by Kaggle and included over 4000 images containing nearly 50,000 vehicle instances. CenterNet was used to obtain initial predictions of location and pose of vehicles. Within the CenterNet architecture, this paper reviewed and compared various aspects of deep learning models such as normalization layers, activation layers, data augmentation, training loss, and various architectures. Ultimately the experiments found that the best performing combination on this dataset so far was a ResNet18 backbone with filter response normalization after and threshold linear unit after each convolutional layer. Additionally, loss was weighted such that mask loss and regression loss were normalized to have equal contribution each time the validation loss increased compared to the previous epoch and data augmentation was implemented to help prevent overfitting. The experiments led to a mAP of 0.1192, which corresponds with a 0.044 (unknown metric) in the Kaggle competition which is good for 82nd place out of 475 participants in the competition. There is still a lot of room for improvement and future work was discussed.

References

[1] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep Residual Learning for Image Recognition. *arXiv:1512.03385 [cs]*, December 2015. arXiv: 1512.03385.

[2] Sergey Ioffe and Christian Szegedy. Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift. *arXiv:1502.03167 [cs]*, March 2015. arXiv: 1502.03167.

[3] Yi Li, Gu Wang, Xiangyang Ji, Yu Xiang, and Dieter Fox. DeepIM: Deep Iterative Matching for 6d Pose Estimation. *International Journal of Computer Vision*, November 2019. arXiv: 1804.00175.

[4] Ilya Loshchilov and Frank Hutter. Decoupled Weight Decay Regularization. *arXiv:1711.05101 [cs, math]*, January 2019. arXiv: 1711.05101.

[5] Sida Peng, Yuan Liu, Qixing Huang, Hujun Bao, and Xiaowei Zhou. PVNet: Pixel-wise Voting Network for 6dof Pose Estimation. *arXiv:1812.11788 [cs]*, December 2018. arXiv: 1812.11788.

[6] Saurabh Singh and Shankar Krishnan. Filter Response Normalization Layer: Eliminating Batch Dependence in the Training of Deep Neural Networks. *arXiv:1911.09737 [cs, stat]*, November 2019. arXiv: 1911.09737.

[7] Bugra Tekin, Sudipta N. Sinha, and Pascal Fua. Real-Time Seamless Single Shot 6d Object Pose Prediction. *arXiv:1711.08848 [cs]*, December 2018. arXiv: 1711.08848.

[8] Yuxin Wu and Kaiming He. Group Normalization. *arXiv:1803.08494 [cs]*, June 2018. arXiv: 1803.08494.

[9] Yu Xiang, Tanner Schmidt, Venkatraman Narayanan, and Dieter Fox. PoseCNN: A Convolutional Neural Network for 6d Object Pose Estimation in Cluttered Scenes. *arXiv:1711.00199 [cs]*, May 2018. arXiv: 1711.00199.

[10] Saining Xie, Ross Girshick, Piotr Dollár, Zhuowen Tu, and Kaiming He. Aggregated Residual Transformations for Deep Neural Networks. *arXiv:1611.05431 [cs]*, April 2017. arXiv: 1611.05431.

[11] Xingyi Zhou, Dequan Wang, and Philipp Krähenbühl. Objects as Points. *arXiv:1904.07850 [cs]*, April 2019. arXiv: 1904.07850.