Car Pose Estimation Using ANNs

John Berroa
Felix Meyer zu Driehausen

for the class:
ANNs for Depth Estimation, 3D Reconstruction, and 3D Printing

September 30, 2017

1 Project Overview

The goal of the project is to estimate the pose of cars in relation to the camera's point of view. In other words, to find the compass direction of a car if direct north ($360°$) is facing directly away from the camera. This was achieved through the usage of artificial neural networks trained in a two step process: first, a classification problem answering the question "what are the probabilities the car is pointed in some general directions?" and second, "With these probabilities, what is the most likely angle of the car?". The source code for this project can be found on GitHub: https://github.com/johnberroa/CORY.

1.1 Evolution of the Project

Originally, the idea was the replicate a network similar to Deep MANTA (Chabot et al. 2017). This was difficult due to the lack of data and their usage of specially designed 3D models. We searched the literature for other car pose estimators, but it seemed the vast majority utilized 3D models. We decided that instead of just replicating existing research, we would try and attempt car pose estimation purely through a convolutional network. This would thus require only pictures of cars as data, which was easily acquired, while still being challenging.

2 Dataset

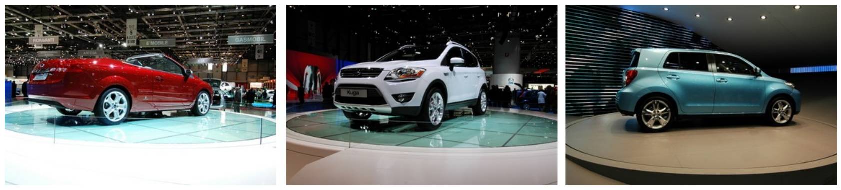
In order to train our network on car poses, we needed a dataset detailing almost every pose there is that a car can be in, i.e. $0° - 360°$. We found the EPFL Multi-View Car Dataset by Ozuysal, Lepetit, and P.Fua 2009 which contains images of cars at a car show rotating on display platforms. This was a perfect dataset to train on. Below are some statistics about the dataset:

- Number of Images: 2299

- Number of Car Sequences (one car per sequence): 20
- Number of Images per Sequence: around 115
- Dimensions of Images: (376,250,3)

For the train/test split, we split the images 80/20 with the first 16 car sequences going into training, and the last 4 sequences being used for testing.

Figure 1: Example images from the EPFL dataset



2.1 Generating Targets[1] (John and Felix)

Although the images were exactly what we were looking for, the authors felt it was permissible to provide their dataset without labels. Therefore, we had to calculate them ourselves.

The authors of the dataset also provided the frontal facing indices and the times each image was taken, as well as the number of pictures per 360°. With this information, we were able to calculate the targets as follows:

1. Calculate how many degrees per second the car was rotating at (assuming constant rotation speed): $\texttt{deg\_per\_sec} = 360/\texttt{time\_per\_circle}$
2. Take the time difference between the frontal facing image and the current image in order to get seconds elapsed: $\texttt{seconds\_elapsed} = abs(\texttt{time[frontal]} - \texttt{time[now]})$
3. Depending on whether the time was before or after the frontal index, add or subtract from 180 and multiply the time change by the degrees per second and the sequence direction (clockwise or counterclockwise), e.g.: $180 - \texttt{seconds\_elapsed} * \texttt{deg\_per\_sec} * \texttt{seq\_direction}$

With this formulation, we were able to generate targets to within 1° of actual angles (judged visually). We could not get the angles closer to reality since the times provided to us were rounded.

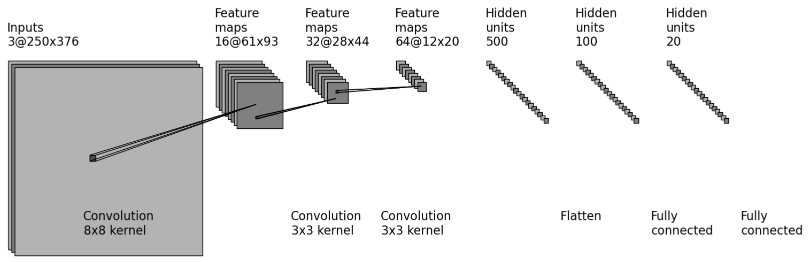
We also implemented binned rotations as per Hara, Vemulapalli, and Chellappa 2017. This was done by creating M rotation lists with N bins each, and seeing if the target angle was within the target bin. If so, this bin was set to 1 to create a set of one-hot vectors per rotation. This methodology is explained in more detail in Section 5.

[1]epfl_data.py and epfl_data2.py (the latter for use with notebooks and model_simplified.py)

3 Regressing Target Angles

As orientation angle prediction is a regression task, we searched for neural network structures which were proven useful in other regression tasks. We found an architecture by Bojarski et al. 2016 that showed success in regressing steering angles for driving a car on a highway. Our first architecture is based on this network.

Figure 2: Schematic network structure of the direct regression model



We attempted to regress the angles directly from the images. We normalized them before sending them into the network in batches of 128. At first, we were using mean squared error as our loss function. When the network did not learn, we took a step back and thought about the nature of circles. We came to the realization that mean squared error actually makes no sense when considering differences between angles. For instance, $359° - 1° = 358°$, but $1° - 359^{\cdot} \neq -358°$, but $2°$. This lead us to devise our own loss function, named *circular loss*. Circular loss is defined as:

$$loss = 180 - abs(abs(y_{pred} - y) - 180)$$

When plugged into the neural network, this loss exploded into huge negative numbers. At this point we decided to split the task into two parts: Felix took on the task of implementing a transfer learning network that would have more feature extraction power than the current network, by using AlexNet. I took on the task of trying to figure out how to get the basic convolutional network to work, because it did not make sense to me that it could not learn such a task.

3.1 Loss Functions[2] (John)

In order to get the convolutional network working, I stripped away all the fancy generators and normalization and went for a very simple straightforward Keras model. With this model, I plugged in various loss functions and tried to get results. In total, I tried seven loss functions[3]. I will detail their results below.

[2] model_simplified.py

[3] As well as loss functions, I tried each loss function with degrees and radians. There was no noticeable difference between these two input "scales."

Circular Loss

This loss was the same we tried in the original network. Using Keras backend code (K.*), the function was rewritten. However, this still leads to an exploding loss towards negative infinity.

Cosine Proximity

After some searching, I found that cosine proximity is exactly the function we were looking for. Keras has a built in implementation so I plugged it in and ran the code. The error approached -1, while accuracy stayed at 0.0000. At first, I thought it wasn't working, but after some examination of what cosine proximity calculates, I realized the loss function was probably working. In this loss function, angles that are the same get a value of 1, angles orthogonal get a value of 0, and angles opposite get a value of -1. Since neural networks are coded to perform gradient descent, it would make sense to minimize the *negative* cosine proximity. However, generating predictions from this trained model yielded values in the thousands, which is clearly not the angles we are looking for.

Cosine Proximity 2

When the original cosine proximity failed, I turned to Google. A Github post[4] provided a rewritten form of cosine similarity that was supposed to avoid some `NaN` errors the original built in function was producing in some cases. I named this function `cos_distance`. This function also approached -1, but returned huge negative values as predictions.

Median Angular Error

In another paper that also used the EPFL Multi-View Car Dataset (Ghodrati, Pedersoli, and Tuytelaars 2014), a different loss function was used: median angular error. This is calculated as:

$$min\{|\theta - \theta^*|, 360 - (|\theta - \theta^*|)\}$$

This loss function resulted in an exploding loss like the circular loss function.

Negative Loss Functions

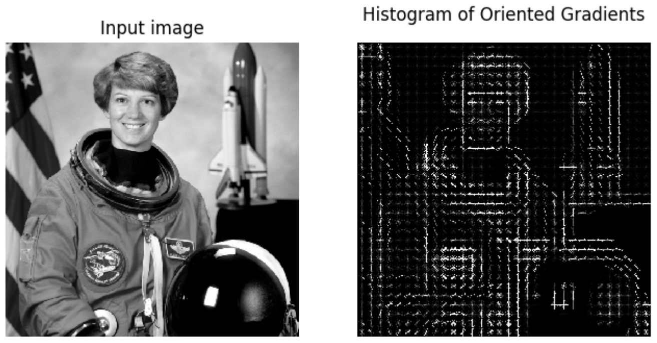
Because a few losses lead to exploding huge negative numbers, I figured take the negative of the losses I could (everything except the builtin cosine proximity) so that it descends towards zero instead of starting negative and descending to negative infinity. However, doing so just made the loss functions start positive and then shoot through 0 and continue their way towards negative infinity.

[4]https://github.com/fchollet/keras/issues/3031

3.2 Regression with Histogram of Oriented Gradients[5] (John)

At this point, it was clear that regression with the convolutional network was not going to work. Either the simple six layer architecture was not powerful enough, there was not enough data, or there was something wrong with the various loss functions. In any case, I then turned my efforts to trying a histogram of oriented gradients approach. A histogram of oriented gradients (HOG), made popular in Dalal and Triggs 2005, is a computer vision technique that returns the number of angled gradients in an image. I figured this would be very successful because the angles of the edges of a car change drastically over a full revolution. In addition, I flipped the car images to double the size of the training set. I created a feed forward network with layers of 1000, 100, 10, 10, and 1 neurons, activated via ReLU. Dropout was utilized in between each layer at a rate of 0.5. Lastly, the error was calculated as the mean squared error.

Figure 3: Histogram of Oriented Gradients Example (retrieved from http://scikit-image.org



With dropout active, the network trains with a decreasing loss to what seems like an asymptote, but an accuracy that remains at zero. Printing the predictions for the training set reveal that all the numbers are very close to 2.8, no matter what the target was supposed to be. After seeing this, I deactivated dropout in order to get as much overfitting as possible, in an effort to see if the network has the ability to find any sort of pattern at all. Without dropout, the outputs ranged from 0.482-2.929 (radians). This was better, but still not within the true range (0-2π), as well as jumping up and down when it should be decreasing or increasing steadily (as is the case in a single sequence).

With these results, regression was off the table. I then transformed the HOG approach into the discretized classification approach that Felix was using the transfer learning on, detailed in Section 5.2.

[5]HOG Regression.ipynb

4 Discretization Based Approach

4.1 Transfer Learning on AlexNet (Felix)

One reason we believe why learning from scratch and directly regressing the car pose did not work is that we were not able to learn meaningful features from such little training data and with this relatively small amount of layers. To circumvent this issue we used fixed and precomputed weights of AlexNet (Krizhevsky, Sutskever, and Hinton 2012) for all its five convolutional layers and the subsequent max-pooling layer. As AlexNet is trained on the Imagenet database (Russakovsky et al. 2015) it has seen many instances of cars and we can safely assume that it can do meaningful feature extraction for our purposes. We are aware that today there are even better performing models on ImageNet than AlexNet, but in this case we wanted to stick to this classic and clean architecture. On top of this architecture we build an orientation estimation unit that is described in the next section.

5 Discretization Based Approach with AlexNet features (Felix)

In order to overcome some of the difficulties we encountered while trying to regress the angle directly, we adopted an approach by Hara, Vemulapalli, and Chellappa 2017. This approach is characterized by the conversion of the regression task into a set of classification tasks and then subsequent restoration of the continuous orientation via a probability distribution and a mean-shift algorithm.

We discretized the $0-360^\circ$ into N bins and assigned every training example the label of the bin that its orientation falls into. This yields an N-class classification problem. Now, we obviously lost information and in the best case can only predict with very limited accuracy, bounded by the number of classes N. In order o limit the loss of information but still keep the classification task simple for the network, we use a multitude of simple classification tasks. For each of these separate classification tasks the discretization starts with a different offset such that the class boundaries are offset and spread uniformly across the $0-360^\circ$ range. Now we have M separate classification tasks with N classes each. We implemented these M independent classification branches in Tensorflow on top of the precomputed weights of the AlexNet architecture and trained all of them jointly. We adopted most of the training parameters from Hara, Vemulapalli, and Chellappa 2017. The weights of the classification branches are initialized with random numbers drawn from the zero mean Gaussian distribution with standard deviation of 0.001. We used AdamOptimizer and minimize the sum of the cross entropies across the classification branches. After some trials, we strayed away from the M and N values provided in the paper (9 and 8) and used 2 and 8. We trained with a constant learning rate of 0.000001 for 5 epochs with a batch size of 32.

5.1 Von-Mises Distribution (Felix)

In testing mode, we calculate softmax probabilities for each of the unique orientations. We then use a weighted kernel density estimation with a von-Mises kernel to define a probability density function from the unique orientations and the softmax probabilities. The von-Mises kernel is a special adaptation of a Gaussian kernel in order to handle inherently circular data. All calculations around the von-Mises Kernel Density Estimation (KDE) are performed by an implementation which we found on Github (https://github.com/engelen/vonmiseskde). We use the resulting probability density function and evaluate it in order to find the maximum. The degree representation of this maximum is the prediction.

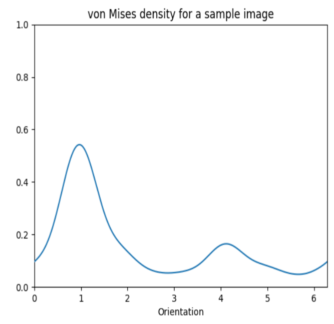


Figure 4: A von-Mises PDF for one data sample. The orientation is given in radians. The orientation at the maximum is converted to degrees and compared with the ground truth.

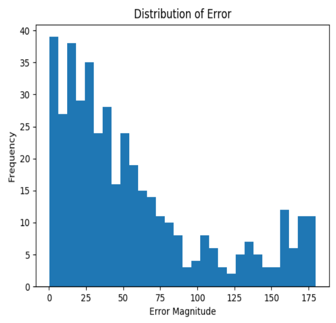


Figure 5: The distribution of errors for the test set.

Figure 5 displays the distribution of errors for the test set. Small peaks can be seen around 180°, which is likely due to similar appearance of the car when horizontally mirrored. We expound upon this more in the conclusion.

5.2 Discretization with Histogram of Oriented Gradients[6] (John)

In order to utilize the HOG with the discretized approach, a new network structure had to be created. M (number of rotations) outputs of N numbers needed to be generated and compared with the target values for each rotation. Softmax probabilities would be necessary in order to do the second step (as denoted in Hara, Vemulapalli, and Chellappa 2017) of generating the actual angles. Therefore, the network had the following structure: $1000 \rightarrow 100 \rightarrow 10 \rightarrow [N_1, N_2, ...N_M]$ neurons, with dropout of 0.5 between each layer and cross entropy loss. Training of this classification task worked quite well. After 20 epochs, average error across the outputs was 1.2809 and average accuracy was 79%. This proves that, with enough data, the HOG approach can learn the discretized category labels of pose fairly well. Now, the second part of reconstructing the actual angle from the class probabilities must be performed. As a side test, I created a network in an attempt to utilize the "universal function approximator" property of neural networks to approximate the von-Mises distribution based mean-shift algorithm. This was done by flattening the rotations of the incoming predictions from the HOG network and sending them through 5 layers of small size (between 5 and 18 neurons), with dropout. The output was compared with the ground truth angle via mean squared error. Sadly, this network failed to learn anything and just output the same number no matter what.

Attaching the von-Mises code from the transfer learning implementation yielded interesting results. Errors weren't as frequently low with the HOG (not many errors at 0), but close to (around 25 or so), as can be seen in Figure 6. The peaks were more pronounced. We theorize in the conclusion as to the nature of the second peak.

6 Conclusion

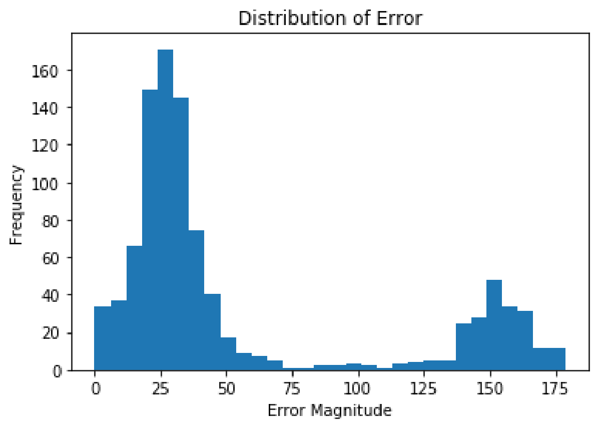
We believe that our results show that 3D car pose from 2D images can be learned with a convolution network for 2 reasons:

1. The accuracy increases while loss decreases — something is being learned.
2. The skewed distribution of error after recomputing the angles clearly shows our results were not chance level.

A in-depth discussion of the last point is important. The majority of errors between the predicted angles and the target angles were close to 0. The errors

[6]HOG Class.ipynb

Figure 6: HOG Angle Error Histogram



died off in an exponential manner until reaching near 180°, at which point it increased again. The skew is verification enough that something is being learned, after all, if nothing was learned the histogram would be relatively flat with an equal probability of all errors. But even more so, the increase in error near 180° is proof of learning in this task. The exponential decrease from 0 should continue on forever, but it doesn't. Something was learned towards the "other end," and since these angles were completely off, it increased the error there. "Completely off" is the keyword here, because 180° is the opposite angle from the true angle. This appears when the network, in essence, gets the pose correct, but can't differentiate between the car direction in relation to front/back.

7 Future Work

As with almost all neural network explorations, more data would most definitely help us here. A larger dataset with different settings to help generalize outside of indoor lighting and into the real world would be valued. Data augmentation would also be helpful. Once such a classifier would be trained on sufficient data, we believe the next logical step would be to plug it into a network that detects cars, and then sends those detections into our network to get the pose. This pose can then be used for traffic avoidance, determining lane directions, among other things. Due to the simplicity of only needing 2D images to predict pose, such a network would be useful in lightweight applications, where space for large models and equipment are limited.

References

Bojarski, Mariusz et al. (2016). "End to End Learning for Self-Driving Cars". In: *CoRR* abs/1604.07316. URL: `http://arxiv.org/abs/1604.07316`.

Chabot, Florian et al. (2017). "Deep MANTA: A Coarse-to-fine Many-Task Network for joint 2D and 3D vehicle analysis from monocular image". In: *CoRR* abs/1703.07570. URL: `http://arxiv.org/abs/1703.07570`.

Dalal, Navneet and Bill Triggs (2005). "Histograms of Oriented Gradients for Human Detection". In: *CVPR*. URL: `https://courses.engr.illinois.edu/ece420/fa2017/hog_for_human_detection.pdf`.

Ghodrati, Amir, Marco Pedersoli, and Tinne Tuytelaars (2014). "Is 2D Information Enough For Viewpoint Estimation?" In: *BMVA*. URL: `http://www.bmva.org/bmvc/2014/files/paper048.pdf`.

Hara, Kota, Raviteja Vemulapalli, and Rama Chellappa (2017). "Designing Deep Convolutional Neural Networks for Continuous Object Orientation Estimation". In: *CoRR* abs/1702.01499. URL: `http://arxiv.org/abs/1702.01499`.

Krizhevsky, Alex, Ilya Sutskever, and Geoffrey E Hinton (2012). "ImageNet Classification with Deep Convolutional Neural Networks". In: *Advances in Neural Information Processing Systems 25*. Ed. by F. Pereira et al. Curran Associates, Inc., pp. 1097–1105. URL: `http://papers.nips.cc/paper/4824-imagenet-classification-with-deep-convolutional-neural-networks.pdf`.

Ozuysal, M., V. Lepetit, and P.Fua (2009). "Pose Estimation for Category Specific Multiview Object Localization". In: *Conference on Computer Vision and Pattern Recognition*. Miami, FL.

Russakovsky, Olga et al. (2015). "ImageNet Large Scale Visual Recognition Challenge". In: *International Journal of Computer Vision (IJCV)* 115.3, pp. 211–252. DOI: `10.1007/s11263-015-0816-y`.