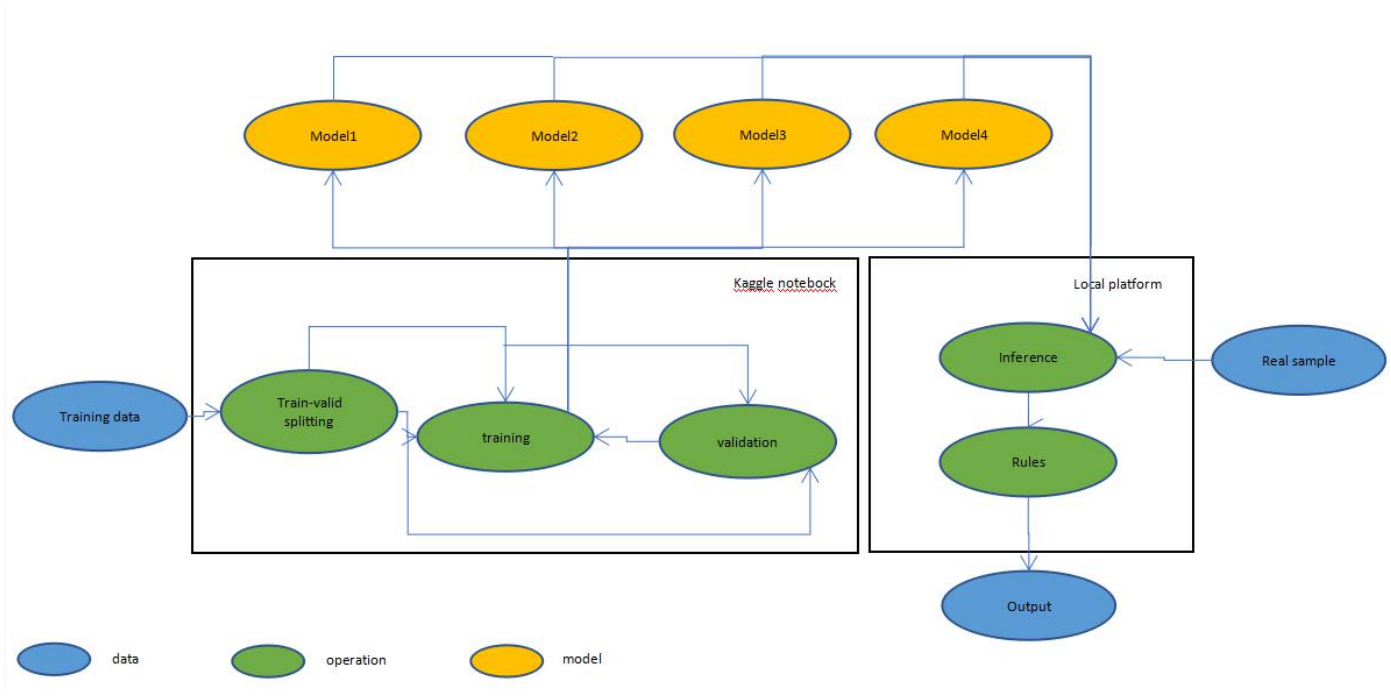
Deep Learning-Based Car Collision Warning

1 Architectural Components Overview



1.1 Data Source and Use Case

1.1.1 Technology Choice

Data source: https://www.kaggle.com/c/pku-autonomous-driving/data (from a Kaggle's competition),
Use case: Deep Learning-Based Car Collision Warning,

1.1.2 Justification

The data source provides a dataset that is provided by Peking University and hosted on Kaggle and is also matched with a competition about autonomous driving. The dataset contains files of training images, masks and labels; test images and masks; car models; camera parameters; and a submission sample.
Note that masks are actually used for masking cars which don't need to be predicted.
You can see https://www.kaggle.com/c/pku-autonomous-driving/data for more information about this dataset.

The use case relates to driving assistance.
In the most cases, one car is only provided with one driver, and the driver is unable to always highly focus on all surrounding things.
Therefore, the target of this article is to build a model, which can identify positions and poses of surrounding cars on the basis of the dataset from the data source, and then add some warning rules (mentioned below) to judge if warning information needs to be displayed to the driver.
For example, warning information of rear-end collision may be delivered due to a lane change in a case of the following picture.

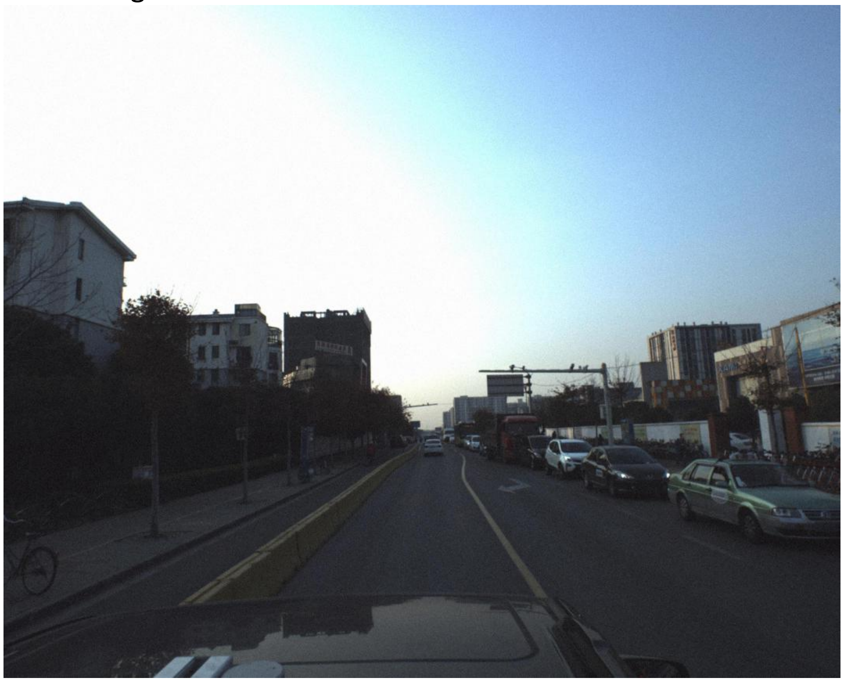
1.2 Data Exploration

1.2.1 Technology Choice

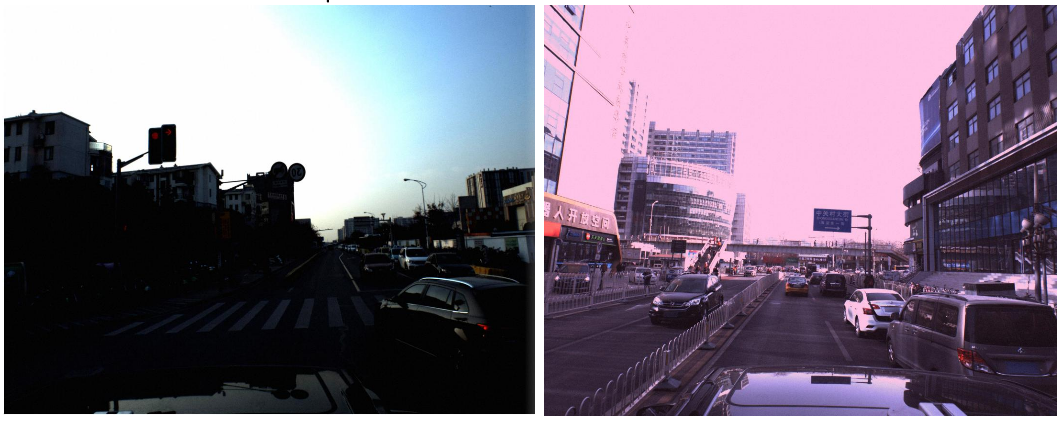
Data Visualization, Correlation Matrices, Python, Pytorch

1.2.2 Justification

Since data contains both images and structured labels, I browsed some images and did data visualization and calculated correlation matrices for the structured labels.
Conclusion of is that:
some images have noises like:



and some are with color problems like:



the distribution of car types is imbalanced;
the distribution of each other dimension of labels is highly imbalanced;
the upper halves of images are less useful or useless;
the dimensions y and z are extremely correlated;
the dimensions x and z are less correlated;
and masks are distributed mostly at far ends.
More details of implementation are in https://www.kaggle.com/a1850961785/data-exploration

1.3 Extract Transform Load (ETL)

1.3.1 Technology Choice
Image Reshaping, String Splitting

1.3.2 Justification
Extract and Load: because the model was trained in Kaggle's notebooks, there was no complicated extraction operation except direct loading.

Transform: the upper halves of images and masks are not used because they don't contain useful information, but the lower halves are reshaped to [512, 2048] ([height, width]) for input into the model. Labels are also transformed from a string type into an array type by string splitting.
Note that the Transform part is done dynamically when the model runs.

1.4 Feature Creation

1.4.1 Technology Choice
Convolutional Neural Networks (CNN), EfficientNet, ResNet

1.4.2 Justification
Some features are generated by one part of the model, namely feature extractors due to the use of deep learning. After some experiment, I made two best plans for this part: EfficientNet-b0 and ResNet-18.
Note that EfficientNet-b0 and ResNet-18 are the smallest models of EfficientNet and ResNet. Bed performance of other deeper extractors was probably because training data is less, the proportion of test data to the training data is also large, thus overfitting will be a big problem when larger models were used. Also, multiple small models facilitate real-time running because you can customize the number of used models if hardware resources are limited.

The other features are depth images, which are obtained by another model-monodepth2 https://github.com/nianticlabs/monodepth2 on the basis of the lower halves of training images.

1.5 Model Definition

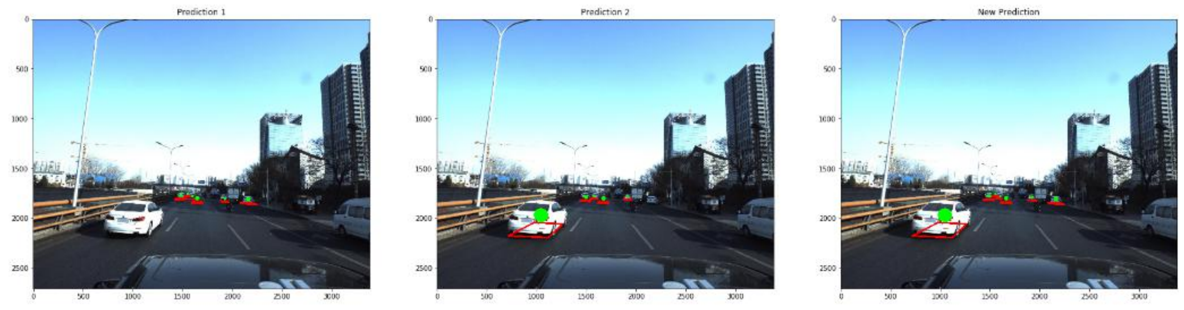
1.5.1 Technology Choice
Convolutional Neural Networks (CNN), U-net, Ensemble Learning

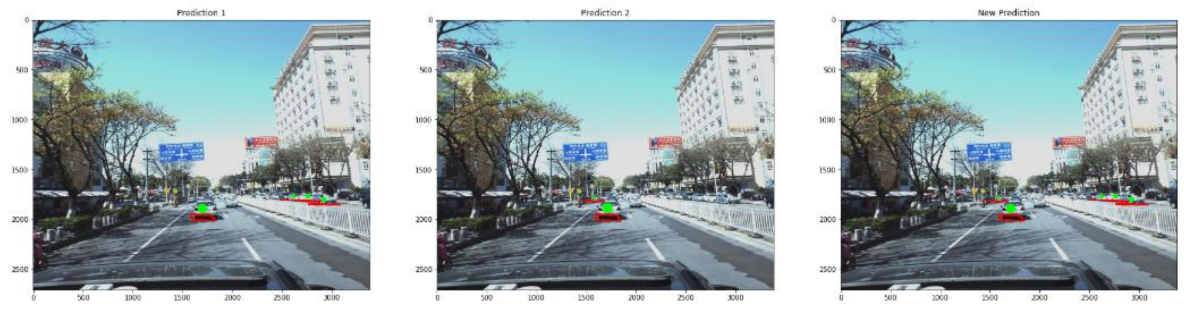
1.5.2 Justification
The main idea of model formation is to train multiple small models with diverse settings, then assemble the output of these models, and finally set some corresponding warning rules of car collision.

Compared with training a large model, this approach enabled the whole model to be highly scalable and easy to update when I find or come up with a new model configuration, and also capable of being customized for different hardware conditions.

Currently 4 models are used:
Model1: ResNet-18 + U-net,
Model2: ResNet-18 + U-net + masks + depth images,
Model3: EfficientNet-b0 + U-net,
and Model4: EfficientNet-b0 + U-net + masks + depth images.
The U-net originated from a kaggle notebook https://www.kaggle.com/hocop1/centernet-baseline. It's more time-saving to use and modify existing code. Note that an unsupervised optimization method of Powell is used for more accurate coordinates.
The masks are used to mask the corresponding images, these were an option because images from the real world will not have corresponding masks to mask far-end cars.
The depth images are above-mentioned. The use of these depth images means that the corresponding model structures were changed.
You can see the model architecture details in Model1.py, Model2.py, Model3.py and Model4.py in https://github.com/FishLikeApple/IBM_ADD_code.

The assembling of these small models is not in a machine/deep learning manner. Two predictions A and B are merged if at least the image distance (not 3D distance) between them are not greater than T(A, B), where T(A, B) = (C/D(A, B)) * f, C and f are constants (5400 and 0.3 in this project), and D(A, B) is the average depth value of A and B. In short, the greater the depth of A and B is, the lesser T(A, B) will be.
More implementation details can be found in
https://github.com/FishLikeApple/IBM_ADD_code/blob/master/output_merging.py.
The following is two examples of assembling (C=5400, f=0.3):





Note that for multiple outputs from the models, two outputs with lowest scores will be assembled in each turn until only one output is left.

The warning rules are highly flexible. In this project I just designed 4 simple rules:
1, if the distance to a surrounding car is less than D1 and the assisted vehicle is facing the car, warning of the car is carried out;

2, if the distance to a surrounding car is less than D2 and the car is facing the assisted vehicle, warning of the car is carried out;
3, if the distance to a surrounding car is less than D3 and the car is facing the front of the assisted vehicle (like the image in 1.1.2), warning of the car is carried out;
and 4, if the distance to a surrounding car is less than D4 and the assisted vehicle is facing the front of the car, warning of the car is carried out.
Ideally D1, D2, D3 and D4 are related to the speed of the assisted vehicle and/or that of the surrounding car, but in the current case I just set them to fixed values: 100, 100, 75 and 75.
More information can be found in
https://github.com/FishLikeApple/IBM_ADD_code/blob/master/rules.py.
And warning now is simply a red dot.

1.6 Model Training

1.6.1 Technology Choice
Gradient Descent, Train-valid splitting, Adam

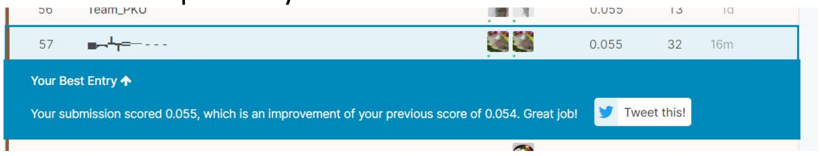
1.6.2 Justification
By default, the optimizer was Adam, validation data was 1% of training data, and test data was provided independently by the data provider. Learning rates of all the small models were initially 0.001 with a learning rate scheduler. Batch sizes are all 2. Losses are the sums of mask losses (losses of whether there is a car) and regression losses (label losses).
You can see the training details of Model3 and Model1 in
https://www.kaggle.com/hocop1/centernet-baseline and
https://www.kaggle.com/phoenix9032/center-resnet-starter respectively, those of Model2 and the same as Model1, and those of Model4 are the same as Model3 except for epochs (Model4 passed 7 epochs).
Note that I left the validation data proportion such small because the details of official evaluation were not published, so that local validation which just relied on losses was much different from the official evaluation, and just for some suggestions.

1.7 Model Evaluation

1.7.1 Technology Choice
mean Average Precision (mAP)

1.7.2 Justification
A special type of mAP realized by the host of this competition was used because by the official evaluation I can compare the performance of the model to know if my model is relatively accurate. As mentioned above, official implementation details of mAP were unclear, and the official evaluation is the only test data evaluation which can be used before the end of this competition, so that this part was regarded as a black box.
Some information of the official mAP evaluation can be found in
https://www.kaggle.com/c/pku-autonomous-driving/overview/evaluation.

In addition, the public LB (leaderboard) scores of Model1, Model3, Model1+Model3, Model2+Model4 and (Model1+Model2)+(Model3+Model4) are 0.039, 0.038, 0.050, 0.054 and 0.055 respectively.



56	Team_PKU	0.055	13	1d
57		0.055	32	16m

Your Best Entry

Your submission scored 0.055, which is an improvement of your previous score of 0.054. Great job!

Evaluation of the warning rule part was fully subjective.

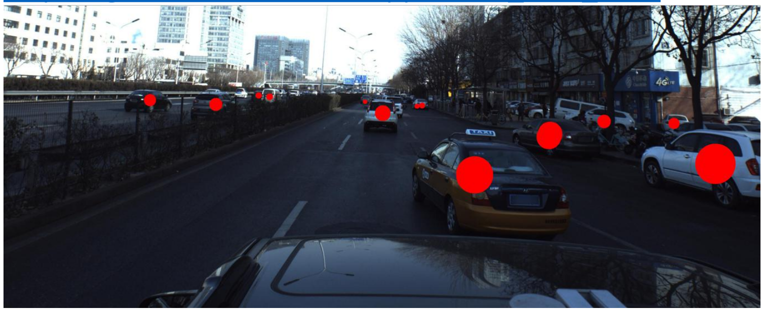
1.8 Model Deployment and Data Product

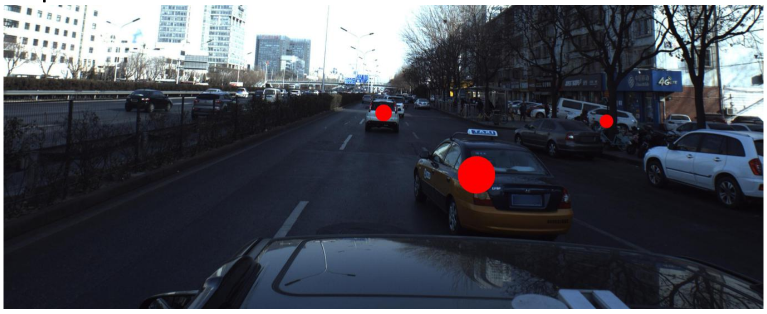
1.8.1 Technology Choice

Github

1.8.2 Justification

One can type the commands in How To Use in README.md in https://github.com/FishLikeApple/IBM_ADD_code to see the following output:


all predicted cars


cars of warning

1.9 Additional Information

Shortcomings:
1, Labeled data in the dataset used are only of cars and SUVs, so the obtained model is unable to identify buses, trucks and the like;
and 2, the dataset is very monotonic in the angle of view (only contains data of the front direction), that is, poses of cars in the dataset are monotonic, thus the model may be not good at identifying cars at the sides of the subject vehicle even if the camera angle is changed to those.

Other methods I will try later if I have time to join this competition:
1, adding image augmentation on training and test data;
2, balancing training samples to deal with the highly unbalanced distribution of labels;
3, changing the loss function since the current loss is less related to the official evaluation;
4, diversifying the feature extractors because feature extractors are really a lot;
5, using Gridsearch to find several sets of better hyperparameters, and thus obtaining some new models from apparently different hyperparameter sets to assemble;
6, implementing a novel network-CenterNet https://github.com/xingyizhou/CenterNet, and then getting more model configurations;
7, trying some one-step object detection architectures like YOLO, but modifying 4D coordinate regression parts into 6D (or 7D like the above models) regression parts;
8, getting bounding boxes and/or segmentation masks of cars by car models, and use a two-step method to first predict positions and then evaluate the other dimensions;
9, utilizing intrinsic relations among cars in the same image to correct regression results since most of them are on the same plane or are related to some extent;
and 10, adopting a machine learning approach for model assembling, in which predictions of training samples from every two different models are used as data, ground truth is used as labels, and then a machine learning model such as XGBoost, Light GBM, CatBoost or a combination of these for assembling is trained on the basis of the data and the labels.