

Group 39 - Car Classification using Neural Networks

Bing Li, Zhaobin Huang, Jinwei Zhu
{bil006, zhh007, j8zhu}@eng.ucsd.edu

Abstract—In this paper we study the problem of fine-grained vehicle make model recognition with deep learning. We use data augmentation, transfer learning and test different models including CNN, inceptionV3, ResNet50 and ResNet152 on this question.

Index Terms—Transfer learning, car classification, deep learning, make and model recognition

I. INTRODUCTION

Detecting the cars' make & model based on the images is an important problem, it helps a lot for researchers to make statistics, for policeman to track a car through multiple cameras and even provide scientific evidences for traffic legislation. Nowadays, deep learning is widely used in different fields and it can help us address difficult fine-grained classification tasks. Here, we are planning to use different deep learning algorithms to solve the car classification problem and compare their performance. The input of our algorithm is car images. We then use a neural network to predict its make, model and released year.

II. RELATED WORK

People have tried many methods on solving the car make and model recognition problem. One possible method is use traditional computer vision techniques[1], use car image from its front or its back as input, detect Region of interest, extract feature and use SVM(support vector machine) for classification. Another possible approach is reconstructing a 3D model from the 2D image and use SVM for classification[2]. Many people try deep learning in extracting features[3] and directly car classification[4]. Some try improvements like combining 3D model and neural network[5] and get good results. We want to try some deep learning models on this problem.

III. DATASET AND FEATURES

The dataset we use in this project is the Stanford car dataset[2]. The training set contains 8,144 images of 196 classes and the test set contains 8,041 images of 196 classes, 16,185 images in total. Each class has roughly about 50 images in training set and 50 images in test set. As the histogram shows below in Figure 1. Most of the set contains about 40 to 50 images and the dataset is roughly balanced. The data set also contains names of cars, and labels and bounding boxes for all images. With this dataset, we can do something like determine whether the image has a car or not, find the bounding box of cars and recognize the car model. The last one

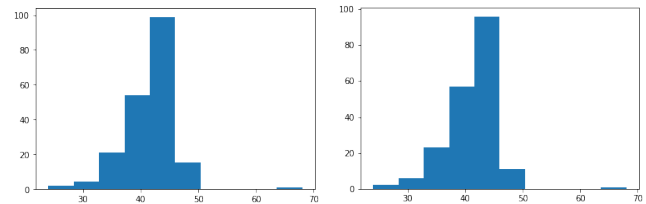


Fig. 1. Left: histogram of training set Right: histogram of test set. x-axis represents number of images in a class, y-axis represents number of classes

is the most interesting problem and we want to try powerful deep learning methods on this problem.

Within the same class, images are different in size, color, viewpoint, background and illumination. Some examples are shown below in Figure 2. Those 4 images belong to the class Acura Integra Type R 2001: Since we have only limited data,



Fig. 2. Different pictures in training set of class Acura Integra Type R 2001

about 100 per class, we use the whole training set as training set and the whole test set as validation set. Also, we use data augmentation such as zooming, rotating and horizontal flipping to deal with limited data and improve the overall performance. For the data preprocessing part, we rescale the images through dividing the value of RGB channels by 255 for normalization. We also resize those images to 299X299 and 224X224, so that the data can fit models like InceptionV3 and ResNet50. Some samples of preprocessed image are show below in Figure 3.



Fig. 3. Samples of preprocessed images

IV. METHODS

Since Each class has limited number of images, when using deep neural network, it is almost impossible to train the whole network parameters from scratch, we can just use simple CNN

to train from scratch and try to use transfer learning for complicated model. Transfer learning is a research problem in machine learning that focuses on storing knowledge gained while solving one problem and applying it to a different but related problem. It is always used in solving problems that is similar to the trained problem when we don't have enough data in our problem. Here we use the parameters that pre-trained using Imagenet and made some modification on the network structure. In this project, we implemented the following networks and made some comparison between them: Simple CNN, ResNet50, ResNet152, InceptionV3.

A. Simple CNN

First, we tried simple CNN and trained it from scratch using random initialized parameters, the structure is shown as below: We just want to see if it is possible to use simple CNN network

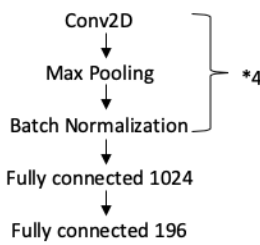


Fig. 4. Simple CNN structure

to finish this difficult classification problem.

B. ResNet50 & ResNet152

ResNet50 and ResNet152[6] is introduced by Kaiming He et al, they used the residual learning framework to ease the training of networks that are substantially deeper than those used previously. Deep convolutional neural networks have led to a series of breakthroughs for image classification, it seems that with the increasing of number of layers, the accuracy starts increasing, but when deeper networks are able to start converging, a degradation problem has been exposed: with the network depth increasing, accuracy gets saturated (which might be unsurprising) and then degrades rapidly. The degradation (of training accuracy) indicates that not all systems are similarly easy to optimize. In order to solve this problem, the author introduced a deep residual learning framework. Instead of hoping each few stacked layers directly fit a desired underlying mapping, they explicitly let these layers fit a residual mapping. They let the stacked nonlinear layers fit a mapping of $F(x) := H(x) - x$ and the original mapping is recast into $F(x) + x$. It is easier to optimize the residual mapping than to optimize the original, unreferenced mapping. Using this way, we can solve the gradient vanishing problem and the ResNet50 means the whole network mostly have 3x3 filters with 34-layer residual network. As shown in the figure below, we can see the structure of ResNet50 compared with VGG16 and simple plain network. According to the paper, the plain 34 layer network has higher validation error than plain 18 layers network, but the 34 layer residual

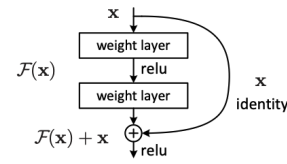


Fig. 5. Residual learning: a building block

block solve this problem and can get the lowest validation error. What's more, the residual network framework can also achieve an incredible 152-layer ResNets by using more 3-layer blocks and the 152-layer ResNet (11.3 billion FLOPs) still has lower complexity than VGG-16/19 nets (15.3/19.6 billion FLOPs). The figure4 shows the detailed structure of ResNet50

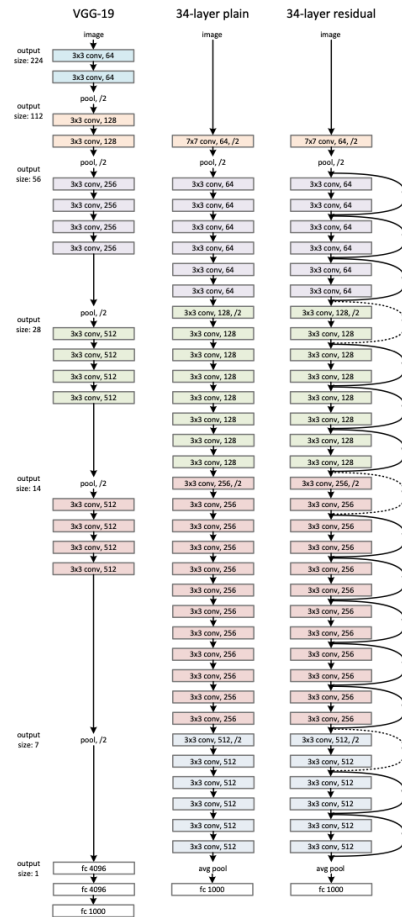


Fig. 6. Structure of ResNet50 compared with other networks

and ResNet152.

C. InceptionV3 Net

With the aim of making the neural network deeper and making the number of parameters in the network smaller, in 2015, Google introduced Google Net[7]. They found that even a modest number of 5x5 convolutions can be prohibitively expensive on top of a convolutional layer with a large number of filters. So they use 1x1 convolutions to compute reductions

layer name	output size	18-layer	34-layer	50-layer	101-layer	152-layer
conv1	112×112	7×7, 64, stride 2				
		3×3 max pool, stride 2				
conv2.x	56×56	$\begin{bmatrix} 3 \times 3, 64 \\ 3 \times 3, 64 \end{bmatrix} \times 2$	$\begin{bmatrix} 3 \times 3, 64 \\ 3 \times 3, 64 \end{bmatrix} \times 3$	$\begin{bmatrix} 1 \times 1, 64 \\ 3 \times 3, 64 \\ 1 \times 1, 256 \end{bmatrix} \times 3$	$\begin{bmatrix} 1 \times 1, 64 \\ 3 \times 3, 64 \\ 1 \times 1, 256 \end{bmatrix} \times 3$	$\begin{bmatrix} 1 \times 1, 64 \\ 3 \times 3, 64 \\ 1 \times 1, 256 \end{bmatrix} \times 3$
conv3.x	28×28	$\begin{bmatrix} 3 \times 3, 128 \\ 3 \times 3, 128 \end{bmatrix} \times 2$	$\begin{bmatrix} 3 \times 3, 128 \\ 3 \times 3, 128 \end{bmatrix} \times 4$	$\begin{bmatrix} 1 \times 1, 128 \\ 3 \times 3, 128 \\ 1 \times 1, 512 \end{bmatrix} \times 4$	$\begin{bmatrix} 1 \times 1, 128 \\ 3 \times 3, 128 \\ 1 \times 1, 512 \end{bmatrix} \times 4$	$\begin{bmatrix} 1 \times 1, 128 \\ 3 \times 3, 128 \\ 1 \times 1, 512 \end{bmatrix} \times 8$
conv4.x	14×14	$\begin{bmatrix} 3 \times 3, 256 \\ 3 \times 3, 256 \end{bmatrix} \times 2$	$\begin{bmatrix} 3 \times 3, 256 \\ 3 \times 3, 256 \end{bmatrix} \times 6$	$\begin{bmatrix} 1 \times 1, 256 \\ 3 \times 3, 256 \\ 1 \times 1, 1024 \end{bmatrix} \times 6$	$\begin{bmatrix} 1 \times 1, 256 \\ 3 \times 3, 256 \\ 1 \times 1, 1024 \end{bmatrix} \times 23$	$\begin{bmatrix} 1 \times 1, 256 \\ 3 \times 3, 256 \\ 1 \times 1, 1024 \end{bmatrix} \times 36$
conv5.x	7×7	$\begin{bmatrix} 3 \times 3, 512 \\ 3 \times 3, 512 \end{bmatrix} \times 2$	$\begin{bmatrix} 3 \times 3, 512 \\ 3 \times 3, 512 \end{bmatrix} \times 3$	$\begin{bmatrix} 1 \times 1, 512 \\ 3 \times 3, 512 \\ 1 \times 1, 2048 \end{bmatrix} \times 3$	$\begin{bmatrix} 1 \times 1, 512 \\ 3 \times 3, 512 \\ 1 \times 1, 2048 \end{bmatrix} \times 3$	$\begin{bmatrix} 1 \times 1, 512 \\ 3 \times 3, 512 \\ 1 \times 1, 2048 \end{bmatrix} \times 3$
	1×1	average pool, 1000-d fc, softmax				
FLOPs		1.8×10 ⁹	3.6×10 ⁹	3.8×10 ⁹	7.6×10 ⁹	11.3×10 ⁹

Fig. 7. Detailed structure of ResNet with different number of layers

before the expensive 3×3 and 5×5 convolutions using Inception module and in 2016, Szegedy et al[8] used two 3×3 to replace the traditional 5×5 convolutions to reduce the number of parameters. As shown in the figure below. Also, in InceptionV3 net, they have factorized the traditional 7×7 convolution into three 3×3 convolutions based on the same ideas as described before. This setup clearly reduces the parameter count by sharing the weights between adjacent tiles. InceptionV3 net is a great combination of

- Factorizing Convolutions with Large Filter Size
- Utility of Auxiliary Classifiers
- Efficient Grid Size Reduction

By using those technique and some training method, we can get a deep neural network with small number of parameters and high accuracy. It is light weighted compared with it's size and is very suitable for classification problems. The combination of lower parameter count and additional regularization with batch-normalized auxiliary classifiers and label-smoothing allows for training high quality networks on relatively modest sized training sets. We'll discuss the training result using those model in detail below and please keep in mind, except for simple CNN, all other networks are using transfer learning with pretrained parameters in Imagenet.

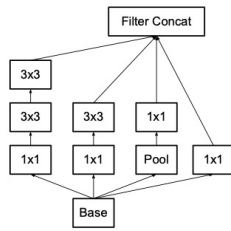


Fig. 8. Inception modules using two 3×3 kernel instead of 5×5

V. EXPERIMENTS, RESULTS AND DISCUSSION

In this project, we tried four types of deep neural network models, including Simple CNN, Inception V3, ResNet 50 and ResNet 152 on Google Colab.

For all these model, we used adam as optimizer, categorical cross entropy as loss function. Adam is the optimizer have quick convergence. Since our datasets are complex and large, we choose to use adam to bring a quick convergence and reduce significant computational time. Since there are 196

classes totally, and each image belongs to only one class, we choose to use categorical cross entropy as our loss function. We used 64 batch size for Simple CNN, 128 batch size for ResNet 50, and 32 batch size for ResNet 152. Larger batch size bring faster computation. These batch size are in balance between computational speed and GPU tolerance.

We evaluate the performance of our models by computing accuracy and loss for training and validation data. Since this is a classification problem, we also demonstrate confusion matrix to help visualize the result intuitively.

The equation of computing categorical cross entropy is:

$$L(y, y') = \sum_{j=0}^M \sum_{i=0}^N (y_{ij} * \log(y'_{ij})) \quad (1)$$

Where y' is the predicted value, y is the actual value.

The principle behind computing accuracy is:

$$Accuracy = \frac{TP + TN}{TP + TN + FP + FN} \quad (2)$$

Where the numerator represents the number of correct predictions, the denominator represent the total number of predictions. TP means True Positives, TN means True Negatives, FP means False Positives, and FN means False Negatives. True Positives represents the cases that both predict and actual are YES. True Negatives represents the cases that both predict and actual are YES. True Negative and False Negative represent in false cases.

Confusion matrix also use TP, TN, FP, FN. Let's take one class as example, the confusion matrix work like this:

Total examples	Predicted: No	Predicted: Yes
Actual: No	TN	FP
Actual: Yes	FN	TP

TABLE I
CONFUSION MATRIX EXAMPLE FOR ONE CLASS

Obviously, large numbers concentrate the diagonal of the matrix represents good result.

A. Plots of accuracy and loss



Fig. 9. Accuracy (Left) and Loss (Right) of Simple CNN.

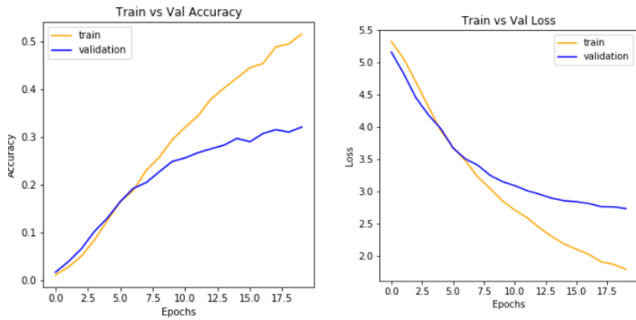


Fig. 10. Accuracy (Left) and Loss (Right) of Inception V3.

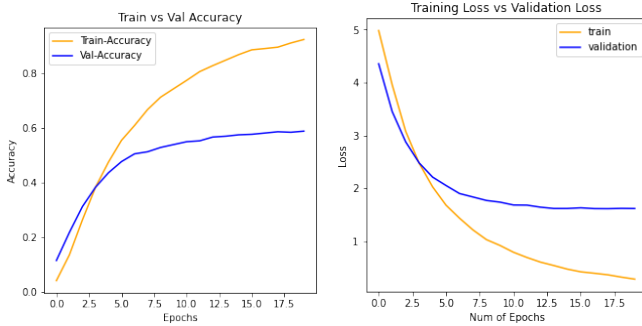


Fig. 11. Accuracy (Left) and Loss (Right) of ResNet 50.

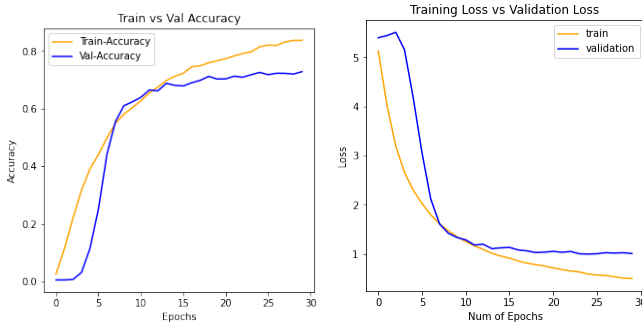


Fig. 12. Accuracy (Left) and Loss (Right) of ResNet 152.

From these plots we can see that ResNet 152 has the best performance. Simple CNN is not suitable in this case since there are too many classes and in each classes there are limited numbers of samples. We need model with much more layers. We know Inception V3 has 48 layers, ResNet 50 has 50 layers, and ResNet 152 have 152 layers, which is deepest model we used.

B. Confusion Matrix

Figure 13 shows the confusion matrix with and without normalization for ResNet 152 model. We can see that a clear diagonal shows in the matrix, which means that most of the predicted class labels correspond to the actual classes. The confusion matrix demonstrates the impressive prediction results we have.

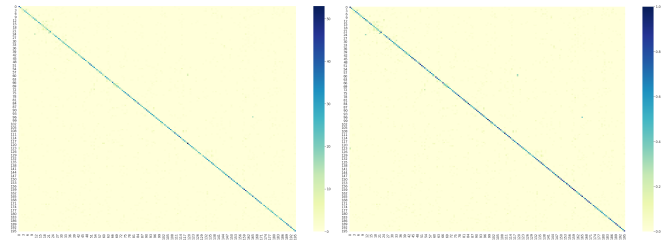


Fig. 13. Confusion Matrix without (Left) and with (Right) Normalization.

C. Final Results Table

Models	Accuracy (Train/Val)	Loss (Train/Val)
Simple CNN	0.95/0.08	0.18/7.90
Inception V3	0.51/0.32	1.79/2.73
ResNet50 (no dropout)	0.99/0.31	0.04/3.70
ResNet50	0.80/0.57	0.70/1.60
ResNet152	0.84/0.73	0.50/1.01

TABLE II
FINAL RESULTS OF ALL MODELS

From the above table we can visualize a clear comparison regarding accuracy and loss for train and validation data among all models. We can see that ResNet 152 has the best performance. We can also see how dropout could improve the performance of a model. Last, we test with SGD optimizer with learning rate 0.01 and momentum 0.8 as hyper parameters. We also try to adjust data augmentation such as image translation and brightness adjustment based on image features to further improve our results. However, SGD optimizer is much slower to converge than adam optimizer, and it need more epochs and GPU memory ability to compute, but we fails to compute it because of limitation of hardware. It should bring a better results since SGD could converge better than adam with longer training time, with the supplement of more adjustment on data augmentation.

VI. CONCLUSION AND FUTURE WORK

In this report, we introduce and evaluate four deep neural network models, including Simple CNN, Inception V3, ResNet 50 and ResNet 152 regarding car classification problem. From the result, we can see that ResNet 152 performs best among them with most layers. The reason is that its ability to extract features most deeply is necessary in limited amount of samples in each class. This project is finished on Google Colab. It is has worse GPU and memory than UCSD Datahub. We fail to use Datahub since we meet the environment problems and get bad results, so we have to choose Colab.

In the Future, if we have less limitation on GPU and memory stuff, we want to test out with our SGD optimizer and adjusted data augmentation in hundreds of epochs. We also want to try other deep neural network models such as VGG 16 to try to maximize the results.

REFERENCES

- [1] Remigiusz Baran, Andrzej Glowacz, and Andrzej Matiolanski. The efficient real-and non-real-time make and model recognition of cars. *Multimedia Tools and Applications*, 74(12):4269–4288, 2015.
- [2] Jonathan Krause, Michael Stark, Jia Deng, and Li Fei-Fei. 3d object representations for fine-grained categorization. In *Proceedings of the IEEE international conference on computer vision workshops*, pages 554–561, 2013.
- [3] Linjie Yang, Ping Luo, Chen Change Loy, and Xiaoou Tang. A large-scale car dataset for fine-grained categorization and verification. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 3973–3981, 2015.
- [4] Heikki Huttunen, Fatemeh Shokrollahi Yancheshmeh, and Ke Chen. Car type recognition with deep neural networks. In *2016 IEEE Intelligent Vehicles Symposium (IV)*, pages 1115–1120. IEEE, 2016.
- [5] Jakub Sochor, Adam Herout, and Jiri Havel. Boxcars: 3d boxes as cnn input for improved fine-grained vehicle recognition. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 3006–3015, 2016.
- [6] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 770–778, 2016.
- [7] Christian Szegedy, Wei Liu, Yangqing Jia, Pierre Sermanet, Scott Reed, Dragomir Anguelov, Dumitru Erhan, Vincent Vanhoucke, and Andrew Rabinovich. Going deeper with convolutions. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 1–9, 2015.
- [8] Christian Szegedy, Vincent Vanhoucke, Sergey Ioffe, Jon Shlens, and Zbigniew Wojna. Rethinking the inception architecture for computer vision. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 2818–2826, 2016.

VII. INDIVIDUAL CONTRIBUTIONS

Bing Li

He worked on Simple CNN model, test SGD optimizer and adjust data augmentation on ResNet152 model, and write report.

Zhaobin Huang

He worked on literature survey, experiment ResNet50 and ResNet152, visualize result and write report.

Jinwei Zhu

He worked on the framework of the code and improve the model in many ways like input's feed, data augmentation, early stop and weight decay. He also wrote part of the report.

VIII. REPLIES TO CRITICAL REVIEWS

Critical review from group 7:

Typos in the slides

- Thank you for pointing out.

What are some ways that you could have reduced the CNN overfit?

- Add dropout layers or use data augmentation to create more data.

Are there any improvements that you can make to the model architecture itself that would improve the car classification task?

- We add a dropout layer and increase accuracy, see Fig 11. and Fig 12.

Confusion matrix was hard to read

- We changed color so that the matrix is more readable, see Fig 14.

No conclusion derived

- See section 5. Experiments, results and discussion.

Critical review from group 11:

The number of EPOCHS is really low in the loss and accuracy plots in slides 15 and 16. We believe the models should have been trained for a higher number of EPOCHS to get a sense of the model performance (such underfitting, ... etc

- Since we used Adam optimizer, so it converges rapidly, 30 epochs could see enough performance. We used Colab and the computation ability is limited so it takes so long to compute 100 above epochs. But we agree more epochs is better.

The results seem to showcase overfitting. This should be addressed to improve results.

- See the third reply to group 28.

Using pictures for the models may be useful instead of writing it all out in words.

- See section 4. Methods.

The plots and words on the slides can be hard to follow, maybe label the plots with titles to make them more clear. Also, use colors (or markers of any type) for the numbers in slides 15 and 16 to make it clear what is validation/training (we are referring to the final numbers and text, not the plots and plot lines).

- Thank you for your advice.

You may want to consider using a histogram to illustrate the distribution of images across car classes.

- See Fig.1

How did augmentation change your dataset? Did you get more images in certain classes for example?

- After add some items of data augmentation such as image rotation, horizontal flip, the final accuracy has been increased. See Fig.3, we didn't add more image because it will be too slow to get results.

What is the distribution of images/class in the testing dataset? This is only mentioned for the training and validation sets - this could be interesting to see as part of the results.

- The distribution of image/class can be seen in Fig 1.

Critical review from group 28:

However, it would be more clear for the audience to understand the whole research if several points are elaborated. At the beginning of the literature, the authors talked about using 2D pictures to build a 3D object and use the reconstructed object to make classification. It's a very interesting topic, but I didn't see anything more about that in the code demonstration. I would appreciate it if they elaborate more about this topic.

- This is not about deep learning so we skip that part.

What's more, the authors didn't talk much about transfer learning. It was until the code demonstration that I realized they used transfer learning instead of training from scratch. It would be better to emphasize it in the presentation and report.

- See section 4. Methods.

Finally, I noticed that there exists some overfitting in the training progress. Could you please explain the measures adopted to prevent the overfitting? I'm also curious about the structure design for the top layers(1024 dense layer and 196 dense layer).

- It is true that there has some overfitting in the progress. We used adam optimizer and 30 epochs to get quick converges, so we could use SGD optimizer with more epochs should, and use more dropout layers could fix.

Minor Point:

There is a trick to save the output of predicted features from existing models like ResNet and Inception, instead of fixing the weights of existing models, and feed these features to the top layers. By doing so, the computational cost of forward propagation in the existing models will be saved in each epoch.

- This is a good idea, we may use this advice in the future work.