DALHOUSIE UNIVERSITY

FACULTY OF COMPUTER SCIENCE

Submitted by	Balaji Dhakshinamoorthy B00777437

Abstract

Fine Grained Image Classification is a Image classification project, part of the Assignment 2 and also the sub-part of the MAFAT Online Challenge to classify the object in the aerial image. The task of the assignment is to classify the type of vehicle in the aerial image. The image files are classified through the combination of image processing technique and Resnet models. Thus the designed model produce good test accuracy of around **71.77%** with some decent generalization score.

Introduction:

This Project is about building a Aerial Image Classifier which contains image of various categories of vehicle like bus cement mixer so that it will help to enable the fine grained classification of images in high resolution aerial imagery as stated in the competition page [1].

The model designed for this Classification uses the knowledge gathered from the pretrained model called Resnet 18 and it produces good test accuracy of around 71.77%. If modified with better augmentation of Generated inputs to compensate the unbalanced data and with more deeper models this model has the capability to produce even better results with the improvement in the generalization aspect.In terms of accuracy, The results are almost close to the top few results achieved by the competitors in the competition even though the task was just reduced to sub-classification of vehicle rather than the full classification as listed in the website.

Dataset:

The Dataset is collected from the codalab competition[1] and it consists of aerial imagery taken from diverse geographical locations. The Dataset has image of the various vehicle and for this task there are 15 sub-classes of the vehicle like Truck,cement mixer etc.

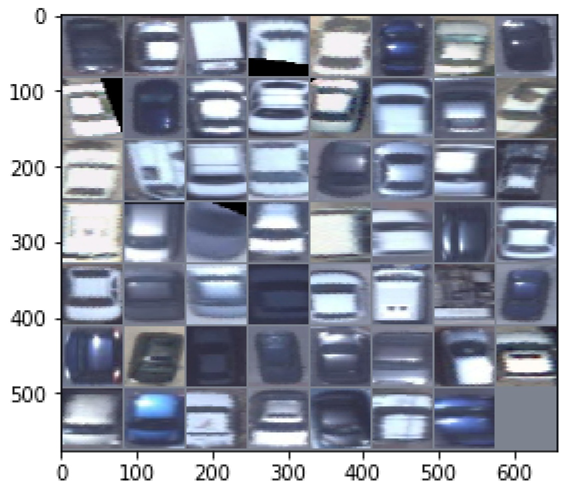
The Dataset has around 11617 images with largest distribution of images for the classes hatchback and sedan respectively. The Dataset will be split using the functionality by the scikit learn so that **94% of the data will go to the train a**nd **6% of the data will go to the test data.** All data will be stratified so their distribution is balanced. **From the 94% training data, 4% of the data will be fixed to the validation data.**

Data Augmentation:

Data Augmentation part of the experiment is handled by the **transformer model** mentioned in the pytorch website.Before Data Augmentation, Data needs to be normalized by using the Image Folder functionality. The entire image mean and standard deviation is computed using this method.The **Mean** of the images are **[0.4956,0.5150,0.5682]** for R,G,B channels.

The **Standard deviation** of the images are **[0.238,0.0246,0.0224]** for R,G,B channels. Then the custom Data Loader should be designed with transformer function defined for the problem. The **image** will be resized to **80x80.** After repeated iterations, this dimension turned out to be good one for better performance. So all the dimension of the image will be rescaled to 80X80. The images will then be converted to the tensor. Then the Two Data Augmentation technique will be applied to the train transformers. **Random Horizontal Flip** and **Random Vertical Flip** will be applied for Augmenting the data. The images will then be normalized by using the values calculated from the previous step.

For validation and test image, Only image resize and normalize will be applied to the transformer function. Following are the image of the training set after applying the resize operation.



Models:

The Model design is impressed by the pytorch tutorial [2]. The inputs are processed with the batch size of around **64.** The model which produced the high accuracy is achieved through the aid of **Resnet 18 pretrained model** given by the pytorch library. Some Other models like Alexnet and Resnet 50 are all trained for the same input and the model with the knowledge of Resnet 18 outperformed all of them.

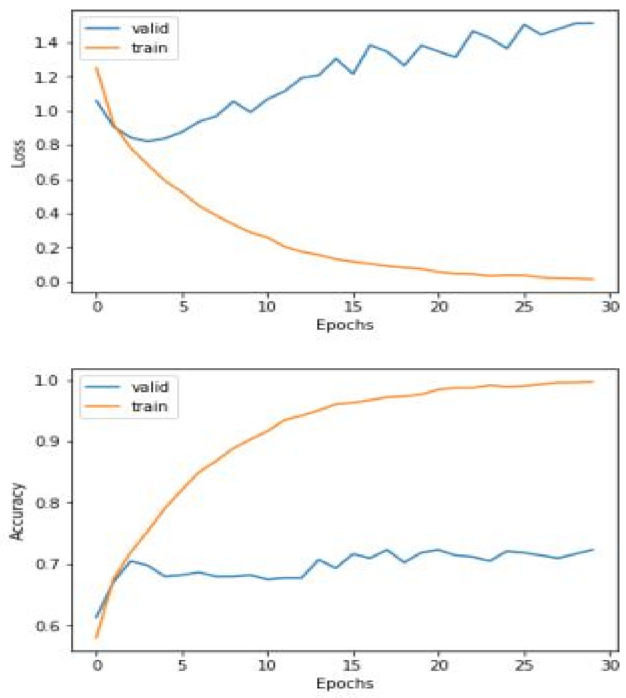
The model is designed along with the fine-tuning architecture as mentioned in the pytorch blog. So that the parameters are optimized during the run. The models are designed with feature_extract option to learn more about the feature representation of the image. **SGD optimizer is chosen for training the model considering their better results compared to the Adam Optimizer.**

Experimentation:

The experiments are conducted as per the code structure as mentioned in the pytorch website. The model is trained for **30 epochs** with SGD optimizer learning rate of around **0.001**. The parameters will be tuned when the training happens and the feature extract feature of the input is set to false.

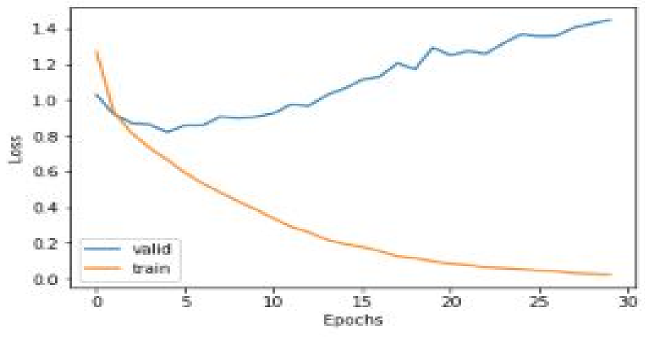
The experiment is conducted for Alexnet, Resnet 18 and Resnet 50. Of them Resnet models produced considerably better results and their learning curve and accuracy is plotted below for the demonstration.

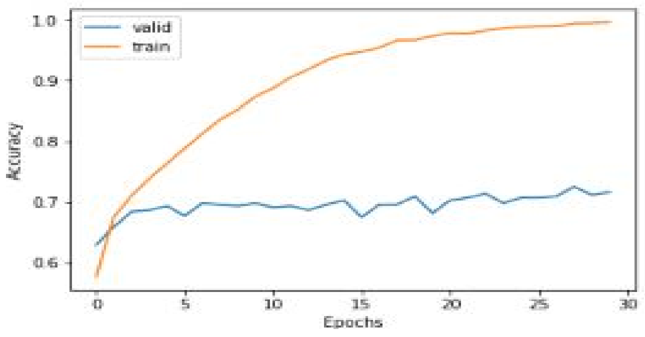
The following is the result of **Resnet 50 Model.**



Resnet 18 Model

Resnet 18 Model is implemented with almost the same configuration for optimizers and transformers. Their Average Pool is modified to Adaptive Average Pooling 2D to deal with the input dimension of any size.The experiment was conducted with 30 epochs and 0.001 learning rate and feature_extract is set to false. The following are the results obtained for the configuration.





Model	Valid Accuracy	Test Accuracy
Resnet 18	72.54	71.77
Resnet 50	72.31	71.2
Alexnet	65	66

Conclusion

Thus Resnet 18 with SGD optimizer helped to classify the vehicle labels reasonably well with pre-trained model inputs. It also helped me to learn about the deep learning model behaviour for generalization of new data. So with more Data Inputs, and with more data samples for other classes,further good results can be obtained. So in future work, the more emphasis should be placed on the addition of generated inputs for the Vehicle classes with fewer data inputs, so that the better Test Accuracy can be achieved.

References

1) https://competitions.codalab.org/competitions/19854
2) https://pytorch.org/tutorials/beginner/finetuning_torchvision_models_tutorial.html