CSC8635 Report

January 6, 2022

1 Introduction

The status of drivers when driving is critical to making driving safe. The State Farm Distracted Driver Detection Dataset is a dedicated dataset that includes nine driver statuses: safe driving, texting, chatting on the phone, manipulating the radio, reaching behind the wheel, applying makeup, and conversing with a passenger to record the drivers' status while driving. As deep learning and deep neural networks see rapidly development in these years, it is a natural step to think of using deep neural networks to classify these images to nine status.

This project firstly trains a deep convolutional neural network model, MobileNet, to learn features in the provided dataset. Subsequently, this work analyses how the setting of hyper-parameters, such as epoch number, batch size, learning rate, the choose of optimizer, as well as the size of training dataset will affect the training process and training accuracy. In the end, suggestions and discussions while performing neural network training is given.

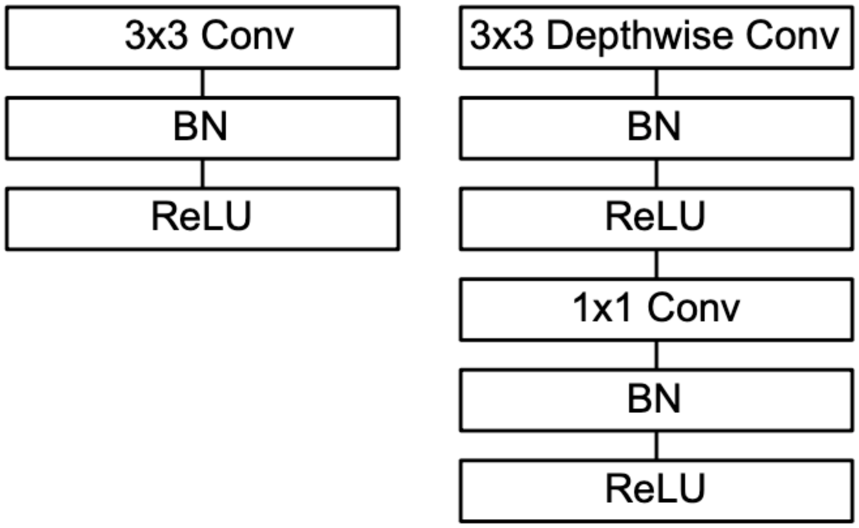
2 Methods

2.1 Deep Leaning method: MobileNet

To identify photos, a variety of deep learning approaches can be utilized. Lenet[1], VGG[2], and other convolutional neural network-based image categorization approaches are examples. In general, deep learning-based image classification technology uses a multi-layer convolution module to extract features from images, followed by a fully connected layer to turn the acquired features into a one-dimensional vector.After that, several fully connected layers are followed and eventually output the possibility of each classification label. The number of final classifications is equal to the number of neurons in the output layer.

MobileNet[3] is a light-weight deep neural network model used in this research that uses depth-wise separable convolutions instead of typical convolutions. MobileNet's parameter number is substantially lower than that of and the network is also faster to train thanks to deep-wise separable convolutions. Accordingly, MobileNet is suitable to be deployed on mobile devices.

A 3x3 convolutional layer, a batch normalization layer, and a ReLU activation layer make up a standard convolutional layer. The 3x3 convolutional layer is replaced by a 3x3 Depthwise convolutional layer in the depth-wise separable convolutional layer. Following that, there is a 1x1 convolutional layer with a batch normalization layer and a ReLU layer. The comparison between the two architectures is depicted in the diagram below[3].



MobileNet performs picture feature extraction using a combination of deep-wise convolutional layers and normal convolutional layers. MobileNet with depth-wise separable convoutional layers performs somewhat worse than Conv MobileNet (MobileNet built with full convolutions) (71.7 percent and 70.6 percent on ImageNet). However, the latter has a far fewer number of parameters (29.3 millions versus 4.2 millions). A smaller number of model parameters is a significant advantage in reducing model size and inference time, which is very suitable to be deployed on mobile devices.

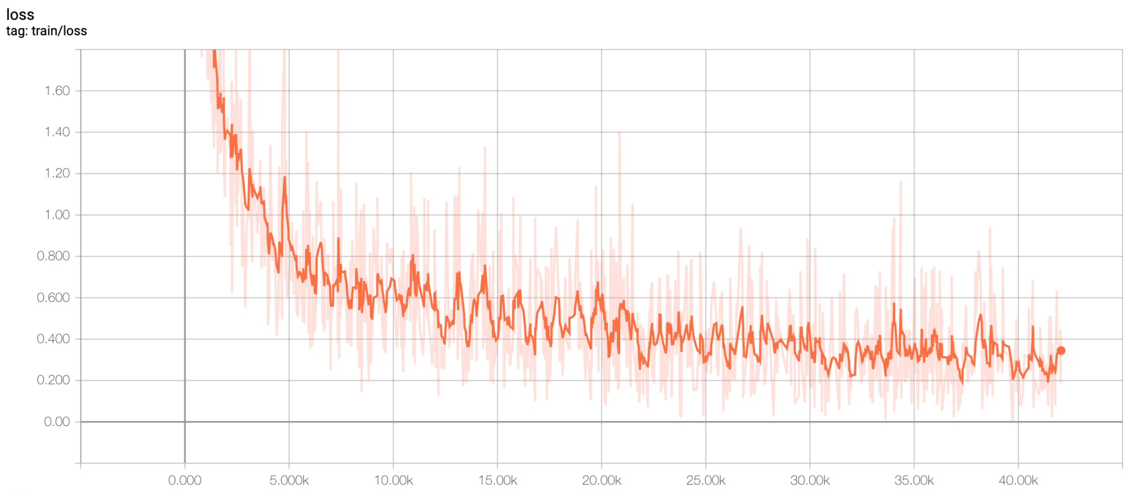
2.2 Experiment Process

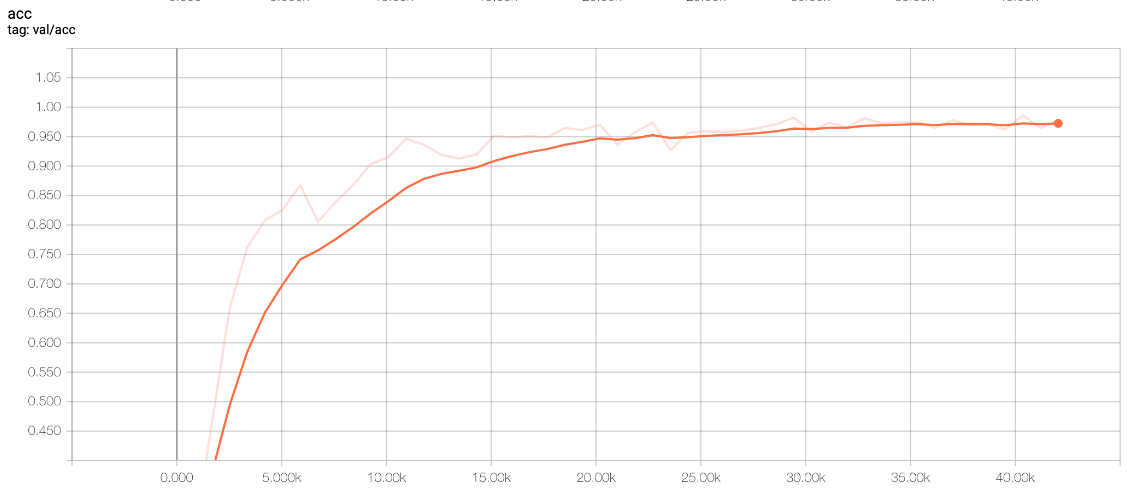
This project consists of three files: load data.py, main.py, and VGG.py. The first method is for reading raw picture data into tensor. When it comes to training data, the RandomHorizontalFlip technique is utilized to horizontally flip training images. With the use of parameter traindata_size and valdata_size, load data.py separates training and validation data and makes the size controllable . Special methods are used in the design of load data.py to make the class method more efficient. The parameter setting function, as well as the training and validation programs, are all included in main.py. To acquire the performance of the present model parameters, a validation set is implemented immediately after training. The model is saved as model.pt once each epoch is completed. VGG.py contains the neural network models VGG16 and VGG19, which I created by myself.

2.2.1 Parameters comparison

This section reports experiment result. There are six experiments in total: epoch number, learning rate, batch size, optimizer and training data size. In each of them, one parameter is modified. The first experiment group is the standard group, which can be regarded as the baseline.

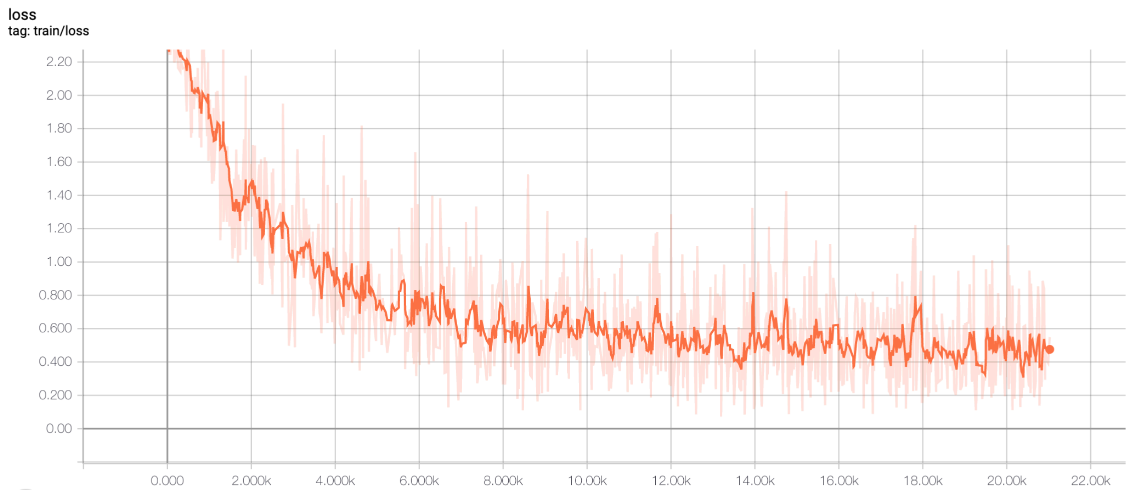
Standard Group This experiment group set epoch number as **50**, learning rate as **0.0001**, batch size as **16** and dataset parameters as **0.6 and 0.95**. Following is the training process:

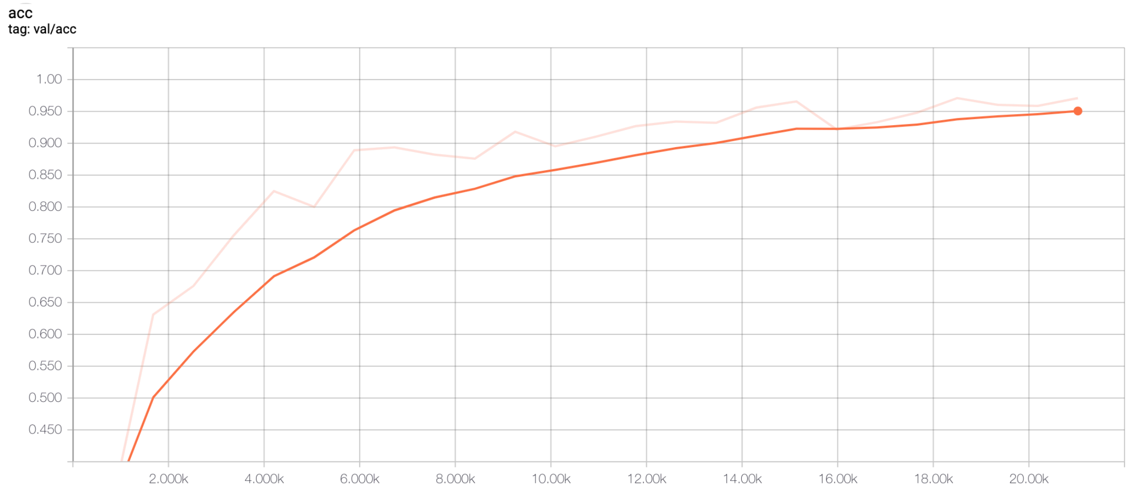




The loss value starts to drop from the beginning of training and stabilizes at the global step of 25k, as shown in the graphs above. After a while, accuracy hits around 0.97, and loss drops to approximately 0.2 to 0.4.

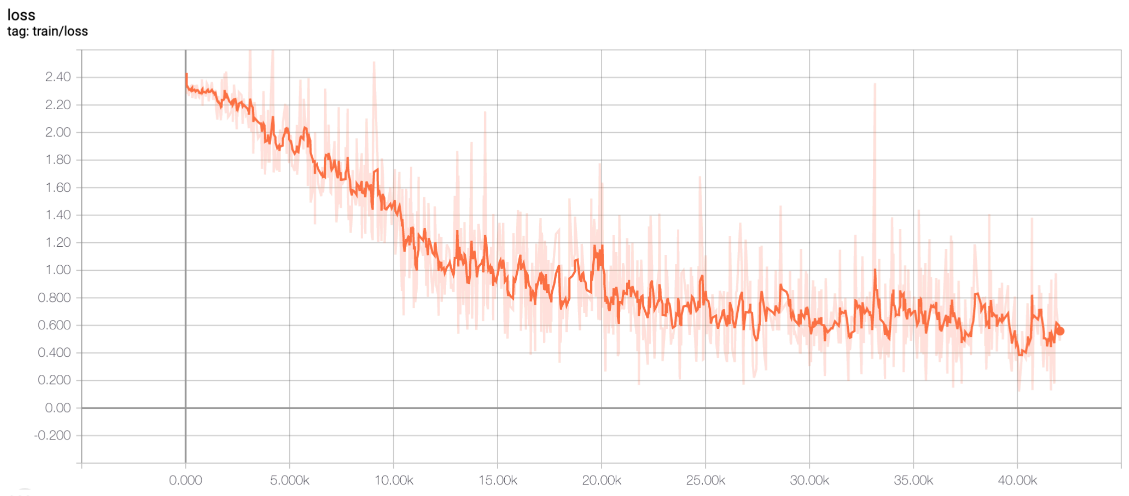
Different Epoch number This experiment group set epoch number as **25**, learning rate as **0.0001**, batch size as **16** and dataset parameters as **0.6 and 0.95**. Following is the training process:

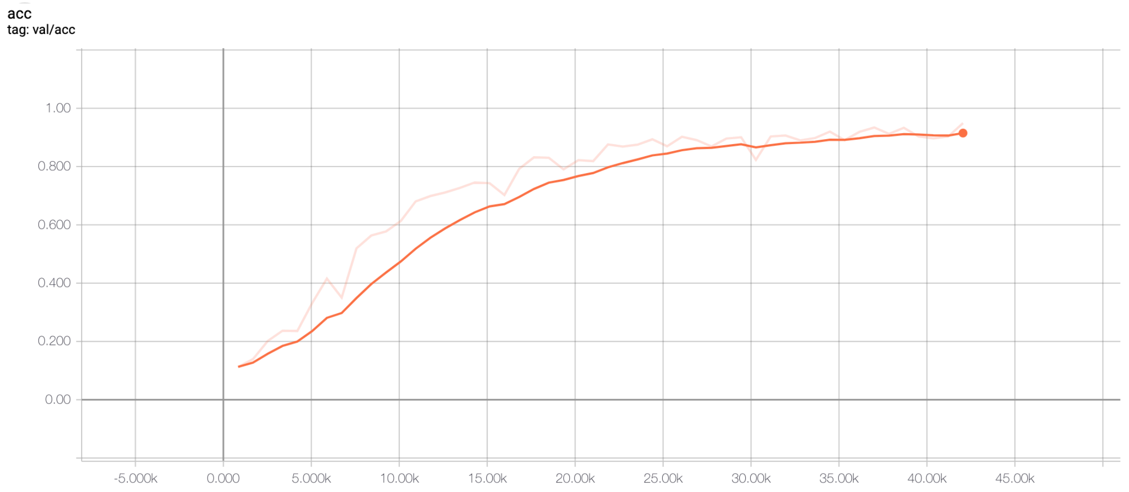




In comparison to the standard group, the preceding figures show that, while the loss value is nearly stable, accuracy is not. In the end, it hovers around 0.95, which is somewhat lower than the typical group's accuracy rate. Finally, sufficient training epoches are required to ensure that the neural network is properly trained and that its best performance is demonstrated.

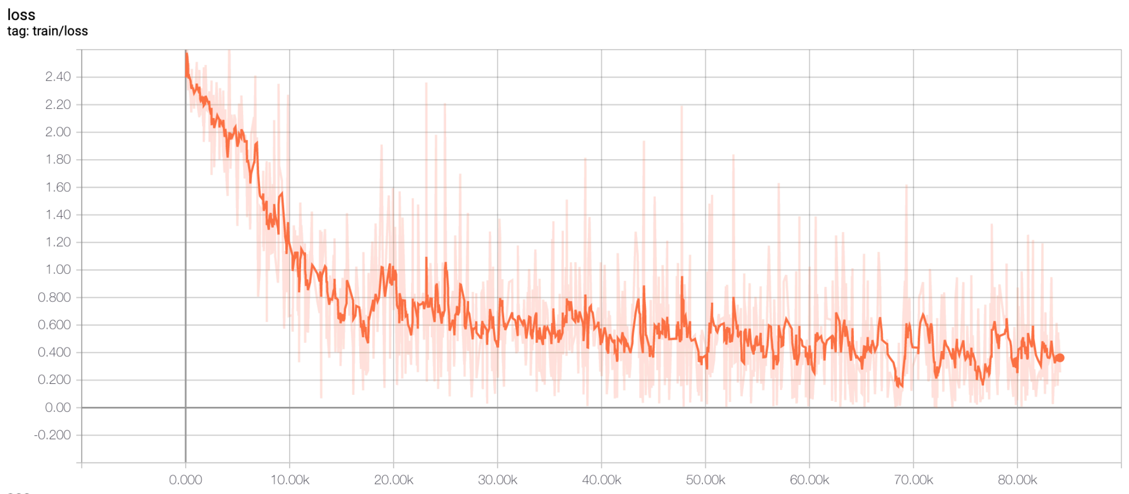
Different Learning Rate This experiment group set epoch number as **50**, learning rate as **0.001**, batch size as **16** and dataset parameters as **0.6 and 0.95**. Following is the training process:

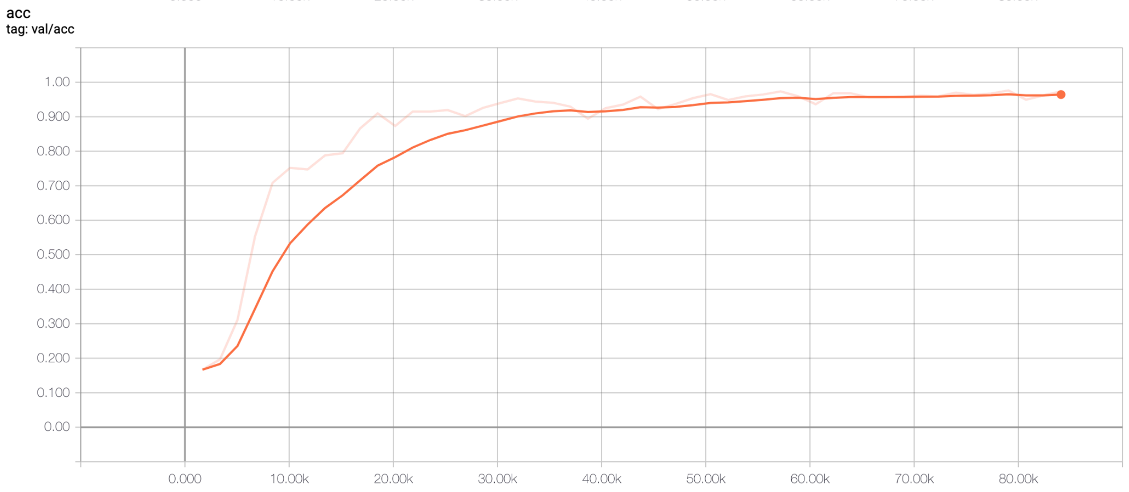




The learning rate is changed from 0.0001 to 0.001 in this experiment. When it comes to the loss graph, it's worth noting that the loss curve sways a lot during training. It fluctuates between 0.4 and 0.8 after 50 epoches of training, compared to 0.4 to 0.6 in the control group. This experiment group's accuracy (about 95%) is also lower than the standard group's (around 96 percent).

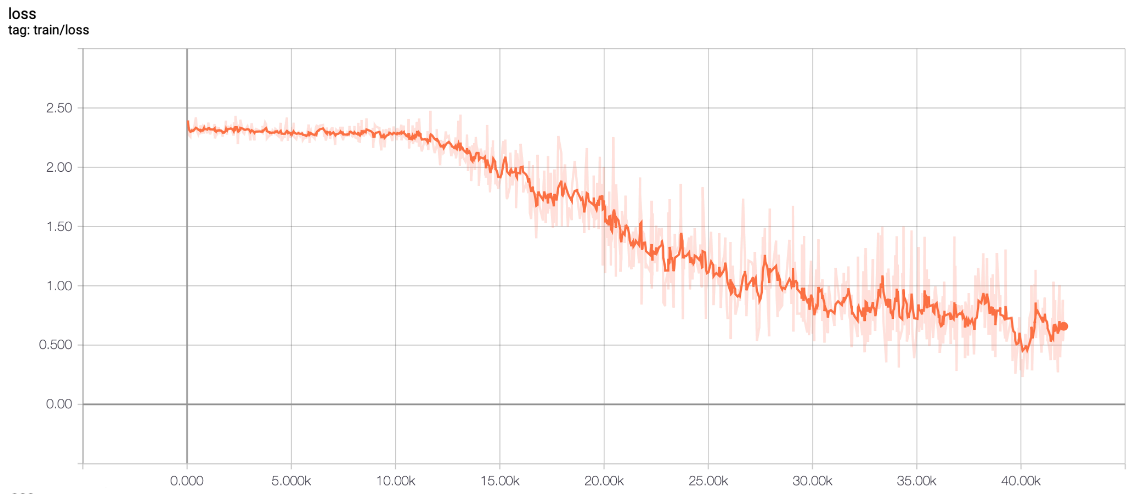
Different Batch Size This experiment group set epoch number as **50**, learning rate as **0.0001**, batch size as **8** and dataset parameters as **0.6 and 0.95**. Following is the training process:

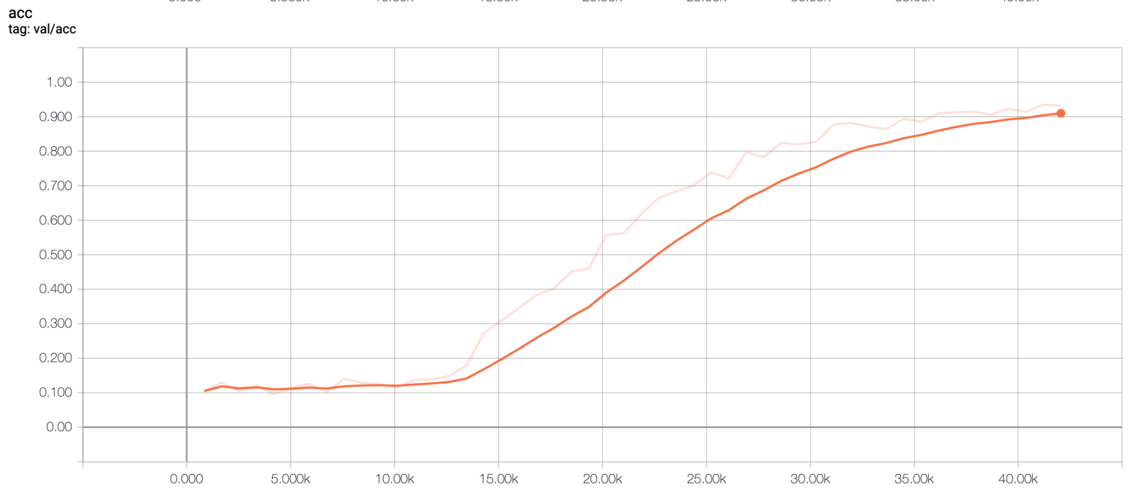




Batch size is set to 8 instead of 16 in this experiment group. As a result, the worldwide step count has risen to 100,000. Meanwhile, it's worth noting that the training period will be longer and the loss curve appears to oscillate more than the standard group's.

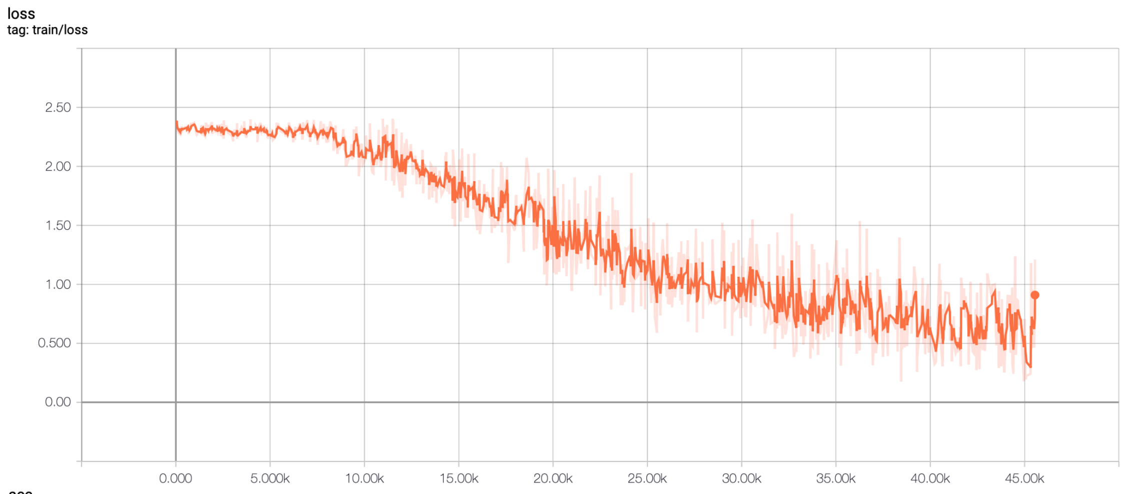
SGD over Adam This experiment group set epoch number as **50**, learning rate as **0.0001**, batch size as **16** and dataset parameters as **0.6 and 0.95**. However, the optimizer uses SGD rather than Adam[4]. Following is the training process:

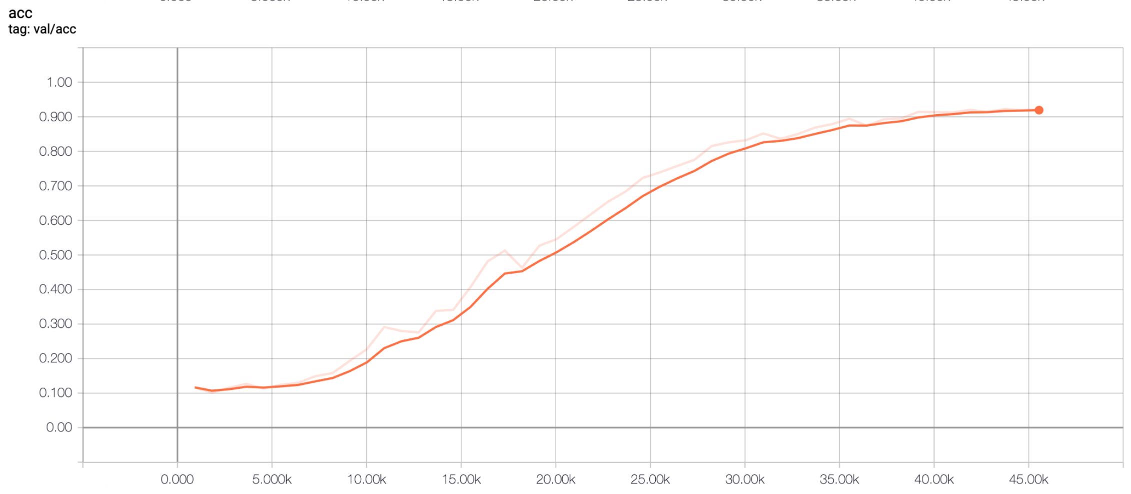




This study group compares the effects of several optimizers on the training process. When compared to the Adam optimizer, the loss in SGD decays far more slowly. It only starts to decay at a global step of roughly 10k, owing to the fact that the SGD optimizer is more likely to fall into a local minimum. The accuracy curve follows a similar pattern. After the 13th global step, it only starts to increase. The accuracy eventually arrives at 90%, which is lower than the standard group. Adam is a superior Optimizer than SGD, according to the trial.

Different Training Batch Size This experiment group set epoch number as **50**, learning rate as **0.0001**, batch size as **16** and dataset parameters as **0.6 and 0.95**. Following is the training process:





The training data amount is increased in this trial group, and the index value is raised from 0.6 to 0.65. Meanwhile, the size of the validation data has been reduced from 0.95 to 0.9. The loss value after training (between 0.5 to 1) is much bigger than the standard group, as shown in the graphs above. Finally, the accuracy rate reaches around 91 percent. The conclusion is intriguing: as the size of the training dataset grows, the training loss and validation accuracy decrease for the same number of training epochs. Meanwhile, training the neural network with a larger dataset necessitates a higher number of training epoch sizes.

2.3 Summary

The chart above shows how different hyperparameters affect the training process. Overall, epoch refers to the overall number of training iterations, and having a sufficient number of training iterations is crucial for excellent training accuracy. Because the learning rate determines how quickly the gradient lowers, a low learning rate usually results in a slower training process, whereas a high learning rate produces a lot of oscillation as the loss decreases. When it comes to batch size, a lower batch size lengthens the training process overall, but the end accuracy remains the same. The choice of optimizer, according to the preceding trials, is also crucial in achieving good training accuracy. Adam outperforms SGD significantly. The Adam technique can help the training process get to the fitting state faster and cross the local minimum value more effectively.

3 Conclusion and Discussion

This project uses MobileNet to classify driver images. Meanwhile, it investigates how hyperparameter settings affect the training process. According to the previous analysis, all five parameters have a direct influence on the training process, with the exception of batch size, which only influences the overall global step size and then the total training duration.

Despite the foregoing findings, a deeper investigation into parameter tweaking is still required. Despite the fact that a lower learning rate means a lower oscillation, a low learning rate slows down the training process. The employment of a dynamic learning rate is a frequent solution to this problem. This approach starts with a high learning rate, but it gradually decreases after each epoch. Both the training time and the oscillation issue are balanced as a result. The batch size is in a similar scenario. If the training is done on GPU, a large batch size necessitates a large graphic memory, and if the dataset is too huge, it may become a problem. Another element that must be carefully studied is the epoch number. The neural network model may not be fully trained if the epoch number is too small. However, if it becomes too large, the additional training time will become unnecessary. Finally, selecting an optimizer is critical. As can be seen in the prior analysis, Adam is a newer and superior optimizer than SGD.

In the future, a more detailed comparison should be considered. For example, a test of whether the aforementioned dynamic learning rate method improves the overall training process in the sense of accuracy and training duration. In addition, a smarter way of training the neural network can be used. For example, as proposed in Vision Transformer[5], comparing the results of the current training round with the results of the verification set regularly until the accuracy does not improve beyond a given range is a superior training strategy. This strategy is smarter and better than giving a fixed number of training epcoh.

4 Reference

[1] LeCun, Y., Boser, B., Denker, J.S., Henderson, D., Howard, R.E., Hubbard, W. and Jackel, L.D., 1989. Backpropagation applied to handwritten zip code recognition. Neural computation, 1(4), pp.541-551.

[2] Simonyan, K. and Zisserman, A., 2014. Very deep convolutional networks for large-scale image recognition. arXiv preprint arXiv:1409.1556.

[3] Howard, A.G., Zhu, M., Chen, B., Kalenichenko, D., Wang, W., Weyand, T., Andreetto, M. and Adam, H., 2017. Mobilenets: Efficient convolutional neural networks for mobile vision applications. arXiv preprint arXiv:1704.04861.

[4] Kingma, D.P. and Ba, J., 2014. Adam: A method for stochastic optimization. arXiv preprint arXiv:1412.6980.

[5] Dosovitskiy, A., Beyer, L., Kolesnikov, A., Weissenborn, D., Zhai, X., Unterthiner, T., Dehghani, M., Minderer, M., Heigold, G., Gelly, S. and Uszkoreit, J., 2020. An image is worth 16x16 words: Transformers for image recognition at scale. arXiv preprint arXiv:2010.11929.

[5] Géron, A., 2019. Hands-on machine learning with Scikit-Learn, Keras, and TensorFlow: Concepts, tools, and techniques to build intelligent systems. O'Reilly Media.

[6] Pointer, I., 2019. Programming PyTorch for Deep Learning: Creating and Deploying Deep

Learning Applications. ” O’Reilly Media, Inc.”.