AI6103 Homework Assignment

Sean Goh Ann Ray
Nanyang Technological University
sean0057@e.ntu.edu.sg

Abstract

This assignment investigates the effects of hyperparameters on the MobileNet deep neural network model, which includes initial learning rate, learning rate schedule, weight decay, and data augmentation, and how it performs on the CIFAR100 dataset.

Introduction

The MobileNet model used in this assignment utilizes the stochastic gradient descent (SGD) optimization algorithm, with momentum set to 0.9. Other hyperparameters of the model are explored in the respective sections, whereby only that specific hyperparameter of the model is modified.

The CIFAR100 dataset (downloaded from the python torchvision library) contains 50,000 training images and 10,000 test images, each having their respective labels, spread across 100 classes.

The first section is describes the preprocessing performed on the downloaded training data.

Each hyperparameter section that follows also includes at least 2 figures, one showing the training and validation losses against the number of epochs, and another showing the training and validation accuracy against the number of epochs.

Data Preprocessing

The training set was further split into 40,000 training and 10,000 validation images using random seed 0 for the partitioning, using the code shown in Figure 1.

```
# Randomly split downloaded training dataset into train_set and val_set with seed 0
train_set, val_set = torch.utils.data.random_split(train, [40000, 10000], generator=torch.Generator().manual_seed(0))
print(f'Size of train_set: {len(train_set)}')
print(f'Size of val_set: {len(val_set)}')
```

```
Size of train_set: 40000
Size of val_set: 10000
```

Figure 1. Code for Train/Val Split

The new training set have the respective proportion of each class as shown in Figure 2, with the horizontal axis being the class label number and the vertical axis being the number of occurrences.

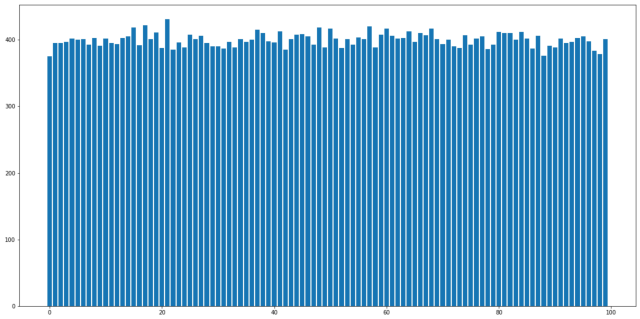

Figure 2. Proportion of Each Class in New Training Set

The computed mean and standard deviation of each color channel of the new training set is [0.5068, 0.4861, 0.4403] and [0.2671, 0.2563, 0.2759] respectively, and these numbers are used to normalize the data. Random horizontal flip and random cropping are also used as data augmentation, using the code shown in Figure 3.

```
# Normalize data
transform_set = transforms.Compose([
    transforms.ToTensor(),
    transforms.Normalize([0.5068, 0.4861, 0.4403], [0.2671, 0.2563, 0.2759]),
    transforms.RandomHorizontalFlip(p=0.5),
    transforms.RandomCrop(32, 32), padding=4)
])

train_norm = torchvision.datasets.CIFAR100(root='./temp', train=True, download=True, transform=transform_set)
train_set_norm, val_set_norm = torch.utils.data.random_split(train_norm, [40000, 10000], generator=torch.Generator().manual_seed(0))
```

Figure 3. Code for Data Augmentation

These normalized data is then used as the training set and validation set in the models for the next few sections.

Learning Rate

The first hyperparameter investigated is the learning rate of the SGD optimizer. 3 constant learning rate values of 0.5, 0.05, and 0.01 were used, without any weight decay, and the model was trained for 15 epochs.

Figure 4 shows the training and validation loss of each setting against the number of epochs, and Figure 5 shows the training and validation accuracy of each setting against the number of epochs.

Table I shows the final training and validation losses and accuracies of each learning rate setting.

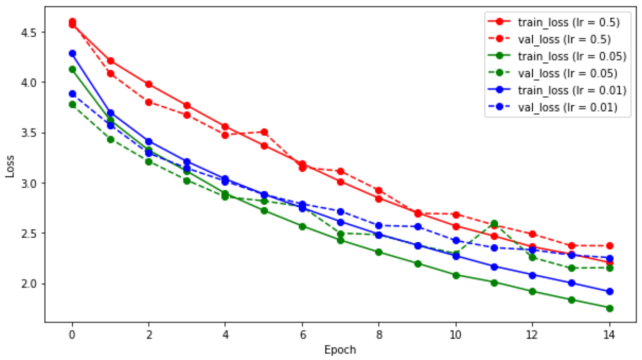

Figure 4. Loss vs Epoch

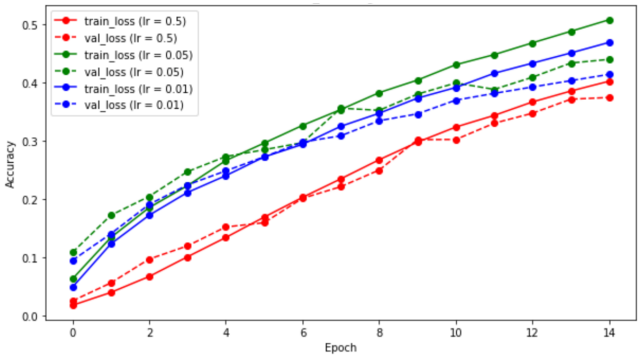

Figure 5. Accuracy vs Epoch

Table I. Final Epoch Training Statistics

Learning Rate	0.5	0.05	0.01
Train Loss	2.2070	1.7591	1.9186
Val Loss	2.3720	2.1541	2.2548
Train Accuracy (%)	40.21	50.74	46.86
Val Accuracy (%)	37.42	43.95	41.38

As observed, the setting of learning rate = 0.05 performs the best, with the least training and validation losses, as well as the highest training and validation accuracies.

Setting the learning rate to 0.5 resulted in large changes in the weights of the model, which leads to overshoot and oscillation, thus slowing the convergence of the weights. Setting the learning rate to 0.01 resulted in changes which are too small, which leads to a slow convergence. The setting of 0.05 provides a good balance of fast convergence without large oscillations of the weights.

Learning Rate Schedule

The next hyper parameter explored is the learning rate schedule. Using the initial learning rate of 0.05 from the best model in the previous section, the experiment was done using 2 learning rate schedules, first a constant learning rate and another with cosine annealing, which decreases the initial learning rate to 0, with a slow decrease at the start, quickly at the middle, and slowly at the end. The experiments were conducted using 300 epochs.

Figure 6 shows the training and validation loss of each setting against the number of epochs, while Figure 7 shows the training and validation accuracy of each setting against the number of epochs.

Table II shows the final training and validation losses and accuracies of each learning rate schedule setting.

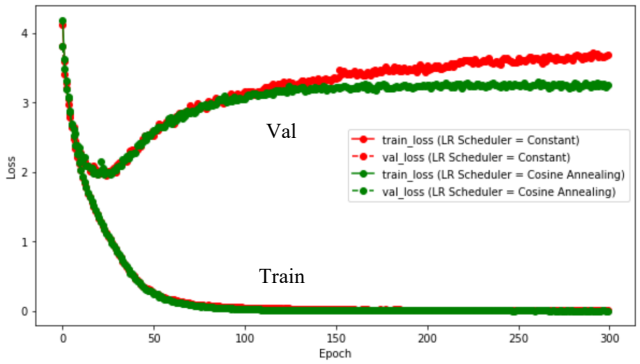

Figure 6. Loss vs Epoch

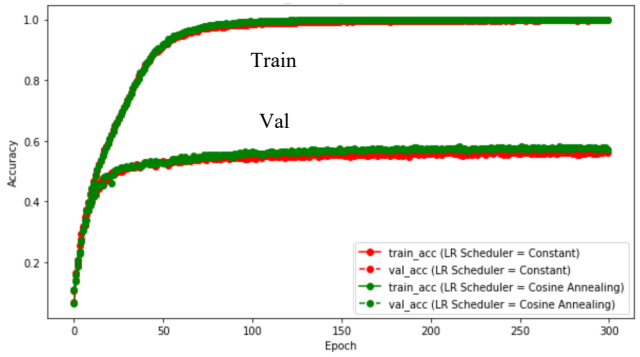

Figure 7. Accuracy vs Epoch

Table II. Final Epoch Training Statistics

Learning Rate Schedule	Constant	Cosine Annealing
Train Loss	0.0073	0.0012
Val Loss	3.6876	3.2486
Train Accuracy (%)	99.77	99.96
Val Accuracy (%)	56.24	57.19

With a close observation of Figures 6 and 7, the constant learning rate schedule had larger improvements in the training loss and accuracy in the first few epochs as compared to the cosine annealing learning rate schedule. However, both models show overfitting of the training data after epoch 25, where the validation losses start to increase. This is especially worse for the constant learning rate schedule.

By the end of the training, the model with cosine annealing performed better, with a smaller validation loss and higher validation accuracy, as compared to the model with a constant learning rate. This is likely due to change in the learning rate which allows larger gradients at the start of the training while also allowing smaller gradients at the end, which prevents overshooting.

Weight Decay

The next hyperparameter explored is the weight decay. Weight decay acts as a regularization to prevent overfitting the training data, and this helps improve generalization, especially on unseen test data.

Using an initial learning rate = 0.05 and cosine annealing learning rate scheduler, the experiment was done with 2 different weight decay values, the first being 5 x 10^{-4} and second being 1 x 10^{-4}. Previous experiments done under the learning rate and learning rate schedule sections were done with weight decay set to 0.

Figure 8 shows the training and validation losses of each setting against the number of epochs, and Figure 9 shows the training and validation accuracy of each setting against the number of epochs.

Table III shows the final training and validation losses and accuracies of each learning rate schedule setting.

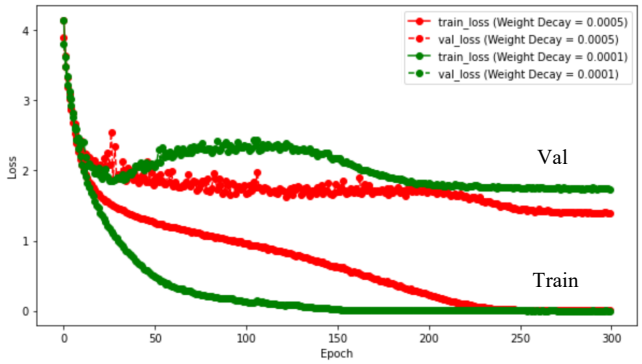

Figure 8. Loss vs Epoch

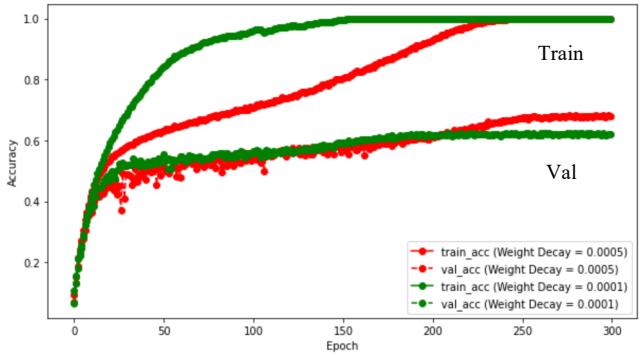

Figure 9. Accuracy vs Epoch

Table III. Final Epoch Training Statistics

Weight Decay	0.0005	0.0001
Train Loss	0.0120	0.0033
Val Loss	1.3984	1.7324
Train Accuracy (%)	99.97	99.98
Val Accuracy (%)	68.07	62.08

As observed in Figures 9 and 10, the improvement in loss and accuracy for the model with weight decay set to 0.0005 appears to be slower than the model with weight decay set to 0.0001.

By comparing the cosine annealing learning rate scheduler loss in Figure 6, it can be observed that the model with the weight decay set to 0.0001 fixes the overfitting when reducing the loss after epoch 100. However, the model with weight decay set to 0.0005 does not seem to have the overfitting issue with the validation loss having a consistent decline.

From Table III, although the model with weight decay set to 0.0005 have worse training loss and accuracy then the other model, it generalizes the image data better as seen from the better validation loss and accuracy. As such, a weight decay of 0.0005 is the better choice.

Data Augmentation

The last hyperparameter explored is data augmentation. Although basic data augmentation is already applied as per the section of Data Preprocessing, an additional data augmentation technique is used here, and that is mixup.

Mixup augmentation is to mix 2 random images with a certain lambda value of one image and (1 – lambda) of the second, and the label of the mixed image is also the same proportional mix of both original labels. This lambda value is taken from a beta distribution with parameter set as alpha = beta = 0.2. Probability Density Function (PDF) is shown in Figure 10.

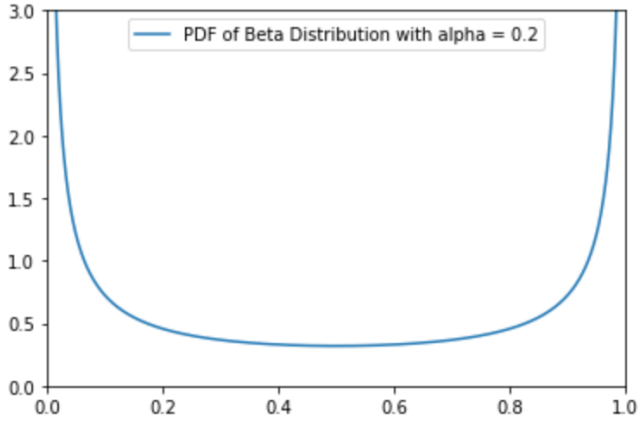

Figure 10. PDF of Beta Distribution with alpha = 0.2

The lambda value taken from the beta distribution is most likely to be near 0 or 1, which means that the mixed image will be mostly from one of the original images. An extreme example (when lambda is 0.5) of the mixup augmentation is given in Figure 11, which shows the original 2 images and the new, mixed image. The original images were already normalized before the mixup augmentation, hence some parts of the image may appear too dark or too bright due to clipping when displaying the image.

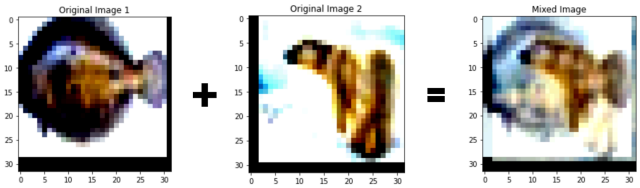

Figure 11. Mixup Augmentation Example

With the mixup augmentation (code based on Zhang et al., 2018) performed on the training dataset, the experiment was performed with the model using an initial learning rate of 0.05, a cosine annealing learning rate schedule, a weight decay of 0.0005 and alpha value of 0.2, the model was trained for 300 epochs.

Figure 12 shows the training and validation loss against the number of epochs, and Figure 13 shows the training and validation accuracy against the number of epochs.

Table IV shows the final statistics of the trained model.

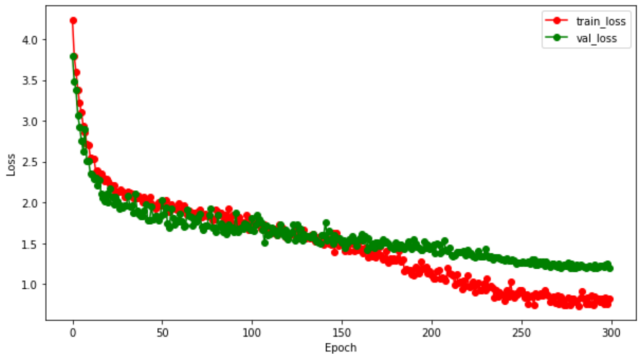

Figure 12. Loss vs Epoch

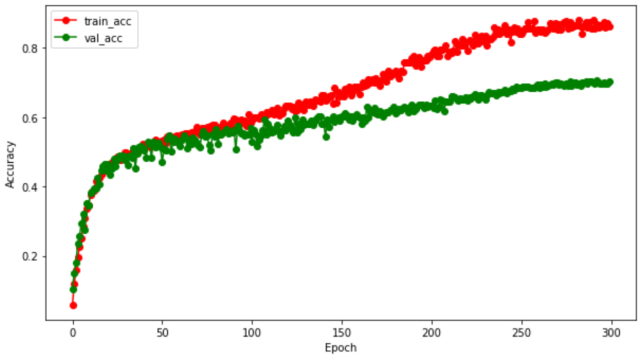

Figure 13. Accuracy vs Epoch

Table IV. Final Training and Testing Statistics

Train Loss	0.8297
Val Loss	1.1977
Train Accuracy (%)	86.22
Val Accuracy (%)	70.30
Test Loss	1.1846
Test Accuracy (%)	70.90

When comparing to the equivalent model in the Weight Decay section, the mixup augmentation increases the training loss (from 0.0120 to 0.8297) and decreases the training accuracy (from 99.97% to 86.22%). This is to be expected as the mixup augmentation of the training images acts as a form of regularization by mixing of features of each image class. On the other hand, the validation loss decreased (from 1.3984 to 1.1977) and validation accuracy increased (from 68.07% to 70.30%). This is also to be expected due to the regularization from the mixup augmentation, which prevents overfitting to the training data and better generalization on unseen data.

The final testing was done using the trained model on the test set, which was only normalized according to the mean and standard deviation of the training set, without any random flipping or random cropping. The test loss and accuracy is similar to the performance on the validation set, and hence can be assumed to be the performance when using the model on other unseen images.

References

Zhang H.; Moustapha C.; Dauphin Y. N.; and Lopez-Paz D. 2018. mixup: Beyond empirical risk minimization. ICLR 2018. doi.org/10.48550/arXiv.1710.09412.