AI6103 Homework Assignment

Kee Ming Yuan
mkee004@e.ntu.edu.sg
G2304842E

Abstract

This paper aims to explore how variations in hyper-parameters, including the initial learning rate, learning rate schedule, weight decay, and data augmentation, impact the performance of deep neural networks.

Data Pre-Processing

The CIFAR-100 training dataset, which consists of 50,000 images from torchvision, is randomly partitioned into training and validation sets using a seed value of 0. These sets contain 40,000 and 10,000 images respectively. The code snippet used are as follows:

```
cifar100Data=datasets.CIFAR100(root=\
'./data',train=True,download=True,\
transform=transforms.ToTensor())
trainSize=int((4/5)*len(cifar100Data))
validationSize=len(cifar100Data)\
-trainSize
trainData,testData=random_split\
(cifar100Data,\
[trainSize,validationSize],generator\
=torch.Generator().manual_seed(0))
```

Proportion of classes in the new training set:

```
apple: 0.9375%; aquarium_fish: 0.9875%;
baby: 0.9875%; bear: 0.9925%;
beaver: 1.0050%; bed: 1.0000%;
bee: 1.0025%; beetle: 0.9825%;
bicycle: 1.0075%; bottle: 0.9775%;
bowl: 1.0050%; boy: 0.9875%;
bridge: 0.9850%; bus: 1.0075%;
butterfly: 1.0125%; camel: 1.0475%;
can: 0.9800%; castle: 1.0550%;
caterpillar: 1.0025%; cattle: 1.0275%;
chair: 0.9700%; chimpanzee: 1.0775%;
clock: 0.9625%; cloud: 0.9900%;
cockroach: 0.9725%; couch: 1.0200%;
crab: 1.0025%; crocodile: 1.0150%;
cup: 0.9875%; dinosaur: 0.9750%;
dolphin: 0.9750%; elephant: 0.9675%;
flatfish: 0.9925%; forest: 0.9725%;
fox: 1.0025%; girl: 0.9925%;
hamster: 1.0000%; house: 1.0375%;
kangaroo: 1.0250%; keyboard: 0.9950%;
lamp: 0.9900%; lawn_mower: 1.0325%;
leopard: 0.9625%; lion: 1.0025%;
lizard: 1.0200%; lobster: 1.0225%;
man: 1.0125%; maple_tree: 0.9825%;
motorcycle: 1.0475%; mountain: 0.9725%;
mouse: 1.0425%; mushroom: 1.0050%;
oak_tree: 0.9700%; orange: 1.0025%;
orchid: 0.9825%; otter: 1.0100%;
palm_tree: 1.0025%; pear: 1.0500%;
pickup_truck: 0.9725%; plain: 1.0425%;
pine_tree: 1.0200%; plate: 1.0150%;
poppy: 1.0050%; porcupine: 1.0075%;
possum: 1.0325%; rabbit: 0.9925%;
raccoon: 1.0250%; ray: 1.0175%;
road: 1.0425%; rocket: 1.0025%;
rose: 0.9850%; sea: 1.0000%;
seal: 0.9750%; shark: 0.9700%;
shrew: 1.0175%; skunk: 0.9825%;
skyscraper: 1.0050%; snail: 1.0125%;
snake: 0.9650%; spider: 0.9825%;
squirrel: 1.0300%; streetcar: 1.0250%;
sunflower: 1.0250%; tank: 1.0050%;
table: 1.0300%; sweet_pepper: 1.0000%;
telephone: 0.9675%; tractor: 0.9775%;
tiger: 0.9400%; television: 1.0150%;
train: 0.9725%; trout: 1.0050%;
tulip: 0.9875%; turtle: 0.9925%;
wardrobe: 1.0075%; whale: 1.0125%;
willow_tree: 0.9950%; wolf: 0.9600%;
woman: 0.9475%; worm: 1.0025%;
```

The mean and standard deviation of each colour channel on the new training set are as follows:

```
Mean: tensor([0.5068, 0.4861, 0.4403])
Standard deviation: tensor([0.2671,
0.2563, 0.2759])
```

The training set then undergoes standard random horizontal flip with a probability 0.5 and random cropping where a padding of 4 pixels are added and images are cropped to a 32-by-32 patch. After that, the each channel of the training

set is normalized by the mean and standard deviation of the respective colour channel. The codes are as follows:

```
transformation = transforms.Compose([
transforms.Pad(4),
transforms.RandomCrop(32),
transforms.RandomHorizontalFlip\
(p=0.5), transforms.ToTensor(),
transforms.Normalize(mean, std)])
```

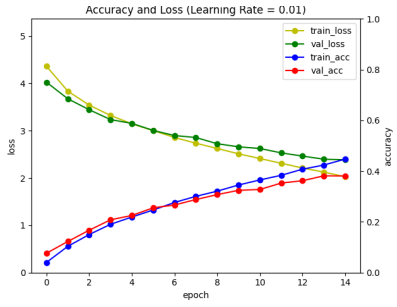
The transformed training set is used to train the model when the hyper-parameters are tuned.

Tuning of Learning Rate

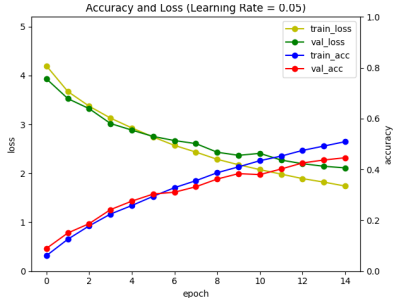
The learning rate (η) affects the extent of update to the new model weights (w_t) in the gradient descent formula where L=Loss and w_{t-1} = previous model weights.

$$w_t = w_{t-1} - \eta \frac{dL}{dw_{t-1}}$$

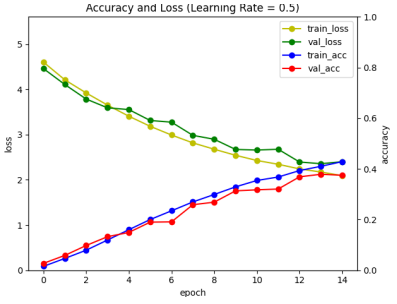
Without the use of weight decay and learning rate schedule, 3 experiments were ran with the learning rate (LR) of 0.01, 0.05 and 0.5 with a batch size of 128 over 15 epochs.



(a) Fig 1: LR = 0.01



(b) Fig 2: LR = 0.05



(c) Fig 3: LR = 0.5

Looking at fig 1 to 3 as well as Table 1, it is evident that when the learning rate (LR) is set to 0.05, the model achieves the highest training accuracy and validation accuracy. Additionally, this LR has the lowest training loss and validation loss. Table 2 below shows that Learning Rate=0.05 has the highest test accuracy and least test loss.

Table 1: Validation/Train Loss and Validation/Train Accuracy of 3 LRs

LR	Val Loss	Val Acc	Train Loss	Train Acc
0.01	2.386	0.381	2.0283	0.4477
0.05	2.1113	0.4465	1.7398	0.5097
0.5	2.4005	0.3751	2.0919	0.4289

Table 2: Test Loss and Test Accuracy of 3 LRs

LR	Test Loss	Test Accuracy
0.01	0.0035	0.4053
0.05	0.0032	0.4648
0.5	0.0040	0.3992

If the learning rate is set too low, for instance at 0.01, the convergence of the model will be slow because the updates to the model weights are relatively small after each epoch. In such cases, it may require more epochs for the model to be optimized to the optimal model weights. This can be why the test accuracy is lower and test loss of is higher for LR=0.01 than that of LR=0.05.

Conversely, if the learning rate is too high, such as 0.05, it can lead to overshooting away from the local minimum. In the subsequent iteration, the gradient descent algorithm will move in the opposite direction as the previous iteration to get closer to the minimum. Nevertheless, because the learning rate is too large, each adjustment to the gradient descend algorithm is big, resulting in oscillation where the model fails to converge to a local minimum. Consequently, this can be why the test accuracy of LR=0.5 is the lowest and its test loss is the highest.

We have determined that the initial learning rate of 0.05 delivers the best performance after 15 epochs, and this value is chosen as the initial learning rate for subsequent hyper-parameter tuning.

Different types of Learning Rate Schedule

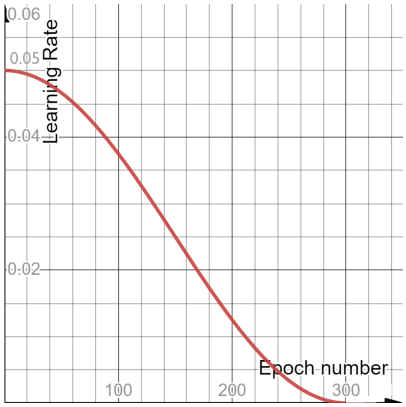
The Learning Rate Schedule (LRS) is a technique used to adjust the learning rate during the optimization process. The Cosine Annealing LRS changes the learning rate by the below equation:

$$\eta_t = \eta_{min} + \frac{1}{2}(\eta_{max} - \eta_{min})(1 + Cos\frac{t\pi}{T})$$

where η_t is the current learning rate, η_{min} is the minimum learning rate, η_{max} is the maximum learning rate, t is the current epoch number and T is the total number of epochs.

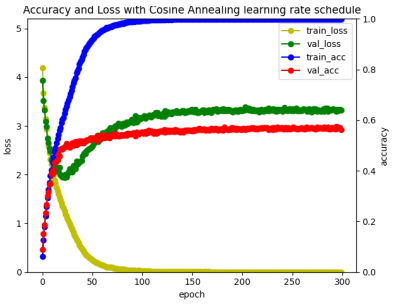
We will conduct two experiments spanning 300 epochs, using two different Learning Rate Schedules (LRSs): the

Cosine Annealing (CA) LRS and the Constant (C) LRS with a fixed initial learning rate of 0.05. Referring to the equation above, we have η_{max}=0.05, η_{min}=0 and T=300 for the Cosine Annealing LRS. By looking at the gradient of the graph in Figure 4 below, we see that the learning rate decreases slowly at the beginning epochs, at the middle of the total epoch number there is a rapid decrease of learning rate, and finally at the end of the total epoch number there is a slower decrease of learning rate. A larger learning rate at the middle of the optimization process will give bigger updates to the model parameter weight. This helps the gradient descend algorithm to reach the minimum point faster without the concern of overshooting since the learning rate will decreases near the end of the optimization process. The decrease in learning rate can be beneficial to the gradient descend algorithm as smaller updates are given to the model parameters when approaching a local minimum. This helps to prevent overshooting away from the local minimum and can converge easily to the local minimum.

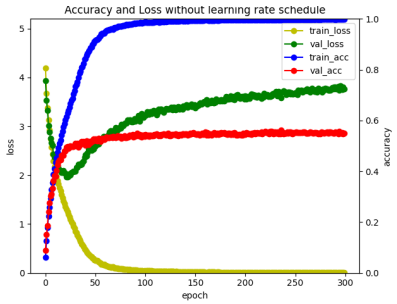


(a) Fig 5: Change of Learning Rate for Cosine Annealing

From Figures 6 and 7, we observe a noteworthy trend. In the case with a Constant LRS, the validation loss increase more than that of with Cosine Annealing LRS, starting from around 100 epochs onward. Interestingly, the validation accuracy remains relatively constant for Constant LRS. This behavior can be attributed to the large learning rate employed in Constant LRS, causing the model to deviate away from the local minimum.



(a) Fig 6: Learning Rate Schedule=Cosine Annealing



(a) Fig 7: Constant Learning Rate

The deviation from the local minimum could have led to the rise in validation loss, though the deviation is not substantial enough to reduce the validation accuracy. This could be because the new model where updates were made to the model weights is still accurate in predicting classes. Moreover, from fig 6 and fig 7, both models are still over-fitting the training data as the train accuracy is close to 1 while the train loss is close to 0.

Table 3: Validation/Train Loss and Validation/Train Accuracy of 2 LRSs

LRS	Val Loss	Val Acc	Train Loss	Train Acc
CA	3.3232	0.5638	0.0011	0.9998
C	3.7616	0.5495	0.006	0.9981

Table 3 shows that Cosine Annealing LRS stands out with a higher level of train accuracy, validation accuracy, as well as a lower train and validation loss.

Table 4: Test Loss and Test Accuracy of 2 LRSs

LRS	Test Loss	Test Accuracy
CA	0.0047	0.5686
C	0.0042	0.5633

From the results in Table 4, Cosine Annealing LRS achieves the highest test accuracy and the lowest test loss, outperforming the Constant LRS. This can be because the small learning rate near the end of the optimization for Cosine Annealing LRS leads to less chances of overshooting and oscillation when the gradient descent algorithm is near the local minimum as the model weights are updated to a smaller extent. This allows the model to converge to a more accurate local minimum in a more stable way. Hence, the model with Cosine Annealing is able to generalize better with bigger test accuracy than that of Constant LRS. Hence, we use Cosine Annealing LRS for the next hyper-parameter tuning.

It is interesting to note that Cosine Annealing LRS did not have a significantly higher test accuracy than Constant LRS. This could be because we have already tune the initial learning rate and used the best initial learning rate from the previous section in the experiment in this section. This initial learning rate might be good enough to allow us to

converge to local minimum and bring good test accuracy results. Moreover, the loss function might be uncomplicated where a constant LRS is enough for the model to converge to a good local minimum. In more complex models, a Constant LRS might not be able to allow the model get high test accuracy and low test loss. Using Cosine Annealing LRS in those more complex scenarios might give a lot better test accuracy and lower test loss.

Tuning of Weight Decay

The aim of Weight Decay (WD) is to prevent over-fitting of models (Zhang, A., Lipton, Z. C., Li, M., and Smola, A. J., 2023). It does so by using L2 Regularization term to the original loss function, giving the new Loss function:

$$L'(w) = L(w) + \frac{1}{2}(\lambda||w||^2)$$

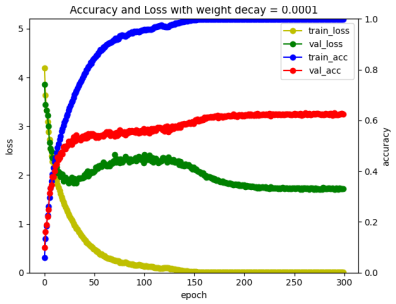
where L(w) is the original loss function defined as the Cross Entropy Loss, w is the model weight value and lambda is the weight decay coefficient.

The second term puts a soft constraint on the complexity of the model. While the addition of the second term to the loss function promotes the minimization of the model weight value to 0 during training, it allows some degree of deviation from the ideal case where the loss (L'(w)) is low because the Cross Entropy term is low, not because the second term is low. That said, the trained model will likely still be less complex as the second term decreases the ability for the model to fit noise from the data. The model will then focus on identifying more important features and improve feature selection, which should increase the model's generalization to unseen data. With a new loss function (L'(w)), the update rule of the gradient descent process becomes:

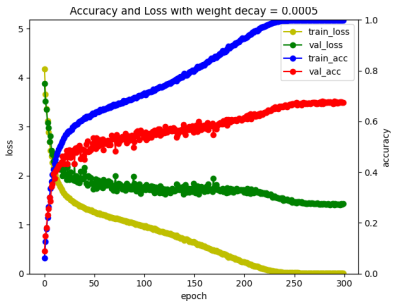
$$w_t = w_{t-1} - \eta(\frac{dL(w_{t-1})}{dw_{t-1}} - \lambda w_{t-1})$$

Since λ controls the regularization strength, a λ too large will lead to under-fitting where the model will try to minimize model weights to the extent where the weights of important features get diminished in the trained model. On the other hand, a too small λ will lead to over-fitting since reducing model weights will not reduce the loss greatly, leading to L'(w)≈L(w). Both cases will lead to reduced test accuracy of the trained model.

Hence, it is important to tune the weight decay coefficient well such that the model captures only important features and become more robust. We will run 2 experiments over 300 epochs with λ=$5 * 10^{-4}$ and λ=$1 * 10^{-4}$.



(a) Fig 8: weight decay = 0.0001



(a) Fig 9: weight decay = 0.0005

Comparing fig 8 and 9, we see that the validation loss when weight decay=$1 * 10^{-4}$ has been larger than when weight decay=$5 * 10^{-4}$ from epoch 0 to 150. Notably at epoch 50 of fig 8, there is a rise in validation loss and at epoch 125, the validation loss starts to decrease. This could be because weight decay=$1 * 10^{-4}$ might be too small for the model, leading to over-fitting at epoch 0 to 125 where the validation loss increases while the train loss decreases. Nevertheless, the over-fit is not to a large extent as the validation accuracy still increased from epoch 50 to 125, only at a very slow rate. The model only start to stabilize from 200 epochs onward where there is not much change to the loss and accuracy of the train and validation sets.

Comparing fig 8 and 9, we also see that it takes more epochs to train model with weight decay=$5*10^{-4}$. The train accuracy and train loss only stabilize at around 225 epochs while that of weight decay=$1*10^{-4}$ stabilized at 150 epochs. The validation accuracy and validation loss only stabilize at around 250 epochs for weight decay=$5 * 10^{-4}$ while that of weight decay=$1 * 10^{-4}$ stabilized at 200 epochs. This is because weight decay=$5 * 10^{-4}$ brings greater change in model weights as per the new gradient descend formula stated above when λ is larger. This causes more instability to the optimization process. Hence, the loss and accuracy of the train and validation sets only stabilize at later epochs.

From fig 8 and fig 9, we see that the model for both weight decays are still over-fitting the training data as the train accuracy is close to 1 while the train loss is close to 0.

Table 5: Validation Loss and Validation Accuracy of 2 WDs

WD	Val Loss	Val Acc
$5 * 10^{-4}$	1.4243	0.6757
$1 * 10^{-4}$	1.7263	0.6247

Table 6: Train Loss and Train Accuracy of 2 WDs

WD	Train Loss	Train Acc
$5 * 10^{-4}$	0.0121	0.9995
$1 * 10^{-4}$	0.0033	0.9999

Table 7: Test Loss and Test Accuracy of 2 WDs

WD	Test Loss	Test Accuracy
0.0005	0.0016	0.6818
0.0001	0.0023	0.6325

From table 5 and 6, we see that weight decay=$5*10^{-4}$ has the lower train accuracy and higher train loss but has higher validation accuracy and lower validation loss than weight decay=$1 * 10^{-4}$. The validation results aligns with the test performance from table 7, where weight decay=$5*10^{-4}$ has higher test accuracy and lower test loss compared to weight decay=$1 * 10^{-4}$. This meant that weight decay=$5 * 10^{-4}$ is able to generalize the unseen data better than weight decay=$1 * 10^{-4}$. The value of weight decay=$5 * 10^{-4}$ overfit the model to the training data to a smaller extent than weight decay=$1 * 10^{-4}$, which is why the model performed worse at train accuracy and has higher train loss for weight decay=$5 * 10^{-4}$. Hence, we choose weight decay=$5 * 10^{-4}$ for the next hyper-parameter tuning.

Data Augmentation

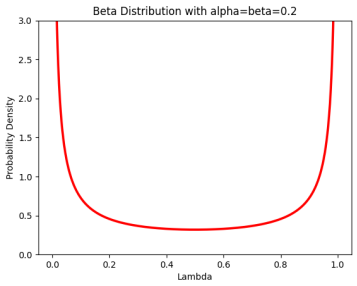
Data augmentation increases the input image size by applying different transformations to the original images to artificially create new images. It is interesting to note that the data augmentation steps we conducted during data preprocessing might not be give the same performance for other data-sets as the effect of data augmentation is dependent on the data-set used. Each data-set has their unique transformation steps to bring good test accuracy and test loss. Mix-up is a type of data augmentation which creates new inputs by linear-interpolation between 2 input and 2 output labels. The new data point $(\tilde{x}, \tilde{y})$ are created as follows:

$$\tilde{x} = (1 - \lambda)x^{(1)} + \lambda x^{(2)}$$

$$\tilde{y} = (1 - \lambda)y^{(1)} + \lambda y^{(2)}$$

where λ is a random variable sampled in the range of 0 to 1, $x^{(1)}$ and $x^{(2)}$ are 2 images from the data-set and $y^{(1)}$ and $y^{(2)}$ are the images' corresponding labels.

The strength of mix-up is controlled by α which ranges from 0 to 1. The higher the α value, the greater the mix up. From the fig 10 below, we see that the probability density function when α=0.2 has high probability of λ being close to 0 or close to 1 and very small probability of λ value being fat from 0 or 1. Looking at the equations above, this means that the $\tilde{x}$ and $\tilde{y}$ generated will be close to one of the 2 input images and one of the 2 labels from the data-set. This is desired as we do not want to generate artificial images which deviates too far from the original images from the data-set.



(a) Fig 10: Probability Density Function

Equating a=1-λ and b=λ, when the model receives an input of f(a$x^{(1)}$+b$x^{(2)}$), it will predict a label of a$y^{(1)}$+b$y^{(2)}$ where the label resembles a linear equation. Mix up encourages the network to behave like a linear function at least locally. Hence, mix up favours smooth network functions and places regularization on the network.

In our model, we used Cross Entropy to calculate the loss. Cross Entropy loss is widely used for data-sets with multi-class classification, which applies to our case as we have 100 classes in the training set as shown in the earlier section of the report. The output of the model can be interpreted as the multinomial distribution of the 100 classes. Each output to the classes is the probability that the image belongs to one of the classes. The below is the Cross Entropy formula:

$$L = \frac{1}{N} - \sum_{i=1}^{n} l(x^{(i)}, y^{(i)}, w)$$

$$l(x^{(i)}, y^{(i)}, w) = -{y^i}^T log P(\hat{y}|x^{(i)}, w)$$

$$y^i = [0, ..., 1, ...0]^T$$

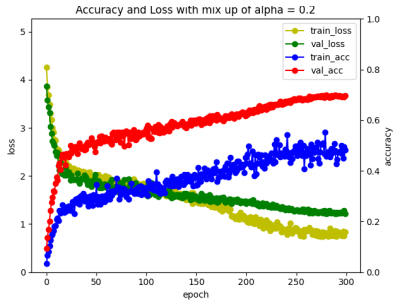
where L is the Cross Entropy Loss, w is the weight, N is the number of images in the batch which is 128 in our case, $P(\hat{y}|x^{(i)}, w)$ is the probability of predicted label and y^i is a one-hot vector which corresponds to the ground truth class, c(i). From the second equation, we see that the one-hot vector is transposed with the probability of predicted label as the loss of that particular image.

The Cross Entropy tries to minimizing the negative log by maximising the probability that the predicted class is the same as the ground truth class, regardless of how confident about the prediction the model is. Especially for deep networks where there are more parameters than the training data points, tha optimization mechanism is strong and can memorize a lot of information. With a capability to fit the training data, the probability of the ground truth label being predicted will be pushed to high values.

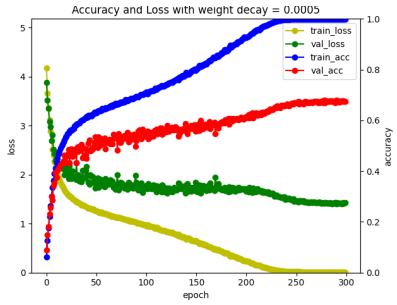
The other benefit of mix up is that it calibrates the confidence of the predictions. A well calibrated model is where

the confidence of the prediction of the label is the same as the accuracy of the prediction. By linear interpolation of 2 class labels, mix up blurs the boundaries of those 2 classes, making the model less confident when making predictions. The network will not rely on sharp decision boundaries which will can cause the model to make predictions where the confidence exceeds the accuracy.

We will conduct an experiment with mix up where α=0.2. Fig 12 shown below is the same image as fig 9 but it is placed together with fig 11 for easier comparison of the effects of mix up and no mix up. Comparing fig and 11 and 12, it is obvious that the validation accuracy is greater than train accuracy when a mix up of α=0.2 is applied compared to when there is no mix up. There is a great reduction of the maximum train accuracy from close to 1.0 to around 0.48. This is due to the randomness and variation that mix up introduces from the artificially created images, leading to the model not fitting the noise in the training data and ultimately increasing generalization of the model trained. This explains why there is an increase in train loss when mix up is introduced as per fig 11. The trained model fits less well to the training data, leading to increased train loss.



(a) Fig 11: alpha = 0.2 for mix up



(a) Fig 12: No mix up

Results from table 8 tallies with the observation from fig 11 and 12. The validation accuracy is higher and validation loss is lower when there is mix up. When there is no mix up, the train loss is lower and train accuracy is higher than when there is mix up.

Table 8: Validation/Train Loss and Validation/Train Accuracy of mix up (MU) of alpha=0.2 and no MU

MU	Val Loss	Val Acc	Train Loss	Train Acc
Yes	1.2223	0.6983	0.8321	0.4841
No	1.4243	0.6757	0.0121	0.9995

Table 9: Test Loss and Test Accuracy of MU of alpha=0.2 and no MU

MU	Test Loss	Test Accuracy
Yes	0.0017	0.7066
No	0.0016	0.6818

Table 9 shows that optimization with mix up gives higher test accuracy than when there is no mix up due to less overfitting. The test loss however, is slightly greater when there is no mix up. This could be because the calibration effect that mix up has, which reduces the confidence of the class prediction, leading to greater test loss. Nevertheless, the slight increase in test loss is not significant enough to make the model make large amounts of false predictions since the test accuracy with mix up is still higher than without.

Conclusion

In this report, we explored the effects to the model performance by changing learning rates, learning rate schedules, weight decay and with and without mix up. I conclude that the optimal model conditions are as follows:

- Learning Rate = 0.05
- Learning Rate Schedule = Cosine Annealing
- Weight Decay = $5 * 10^{-4}$
- Mix up with α = 0.2

The above model conditions might not be the best as more experiments can be run to better tune the values of learning rate, weight decay and the α value of mix up. More types of learning rate schedule can be studied too. Additionally, different types of techniques can be examined as well such as drop out, label smoothing and stochastic depth.

Reference

Zhang, A., Lipton, Z. C., Li, M., Smola, A. J. (2023). Dive into deep learning. Cambridge University Press.