AI6103 Deep Learning: Homework Assignment

Adnan Azmat (G2303265K)
MSc Artificial Intelligence, Nanyang Technological University

Abstract

This report explores the optimization of MobileNet model on the CIFAR-100 dataset, focusing on hyperparameter tuning and data augmentation techniques. The study investigates the impact of learning rates, the use of a learning rate scheduler, weight decay and MixUp data augmentation. The findings highlight the effectiveness of specific configurations in improving model performance on the CIFAR-100 dataset.

Introduction

A. CIFAR-100

The CIFAR-100 dataset, short for the "Canadian Institute For Advanced Research 100," is a widely-used benchmark dataset in the field of computer vision. It was created to facilitate research on object recognition and classification. CIFAR-100 is an extension of the CIFAR-10 dataset.

CIFAR-100 consists of 60000 32x32 colour images in 100 classes, with 600 images per class. There are 500 training images and 100 testing images per class. There are 50000 training images and 10000 test images. The 100 classes are grouped into 20 superclasses. There are two labels per image - fine label (actual class) and coarse label (superclass).

Figure 1: Examples of CIFAR-100

Recent advancements have led to significant improvements in model performance on this dataset. For instance, the EffNet-L2 model achieved a top accuracy of 96.08%. Other successful models include the Swin-L + ML-Decoder with an accuracy of 95.1%, and the µ2Net (ViT-L/16) with an accuracy of 94.95%. These models leverage techniques such as Sharpness-Aware Minimization, Scalable and Versatile Classification Head, and an Evolutionary Approach to Dynamic Introduction of Tasks. Despite these advancements, achieving high accuracy on the CIFAR-100 remains a challenging task due to the dataset's low resolution, complexity and diversity.

B. MobileNet

MobileNet is a family of compact convolutional neural network architectures, designed and open-sourced by Google for mobile and embedded devices. They employ depthwise separable convolutions to reduce computation, offer width and resolution multipliers for flexibility, and come in various versions (e.g., MobileNetV1, V2, V3) with trade-offs between model size and accuracy. These models are optimized for real-time inference, making them ideal for resource-constrained applications like image classification and object detection on mobile devices and IoT platforms.

MobileNet employs a two-step convolution process for efficiency: depth-wise convolution and point-wise convolution. Depth-wise convolution performs separate convolutions on each input channel, significantly reducing computation. Then, point-wise convolution combines the results from depth-wise convolution, mixing and transforming features across channels. This design reduces model size and computational requirements while maintaining performance.

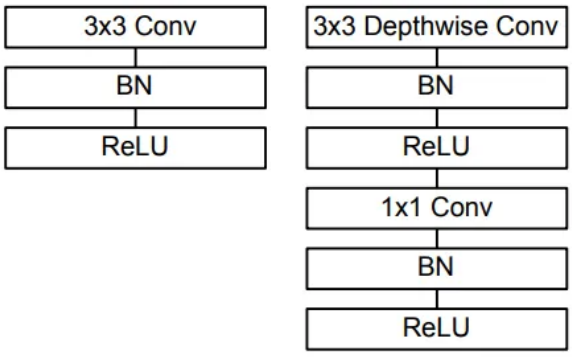

Figure 2: Left: Standard Convolutional layer, Right: Depthwise Separable Convolutional layers in MobileNet

Type / Stride	Filter Shape	Input Size
Conv / s2	$3 \times 3 \times 3 \times 32$	$224 \times 224 \times 3$
Conv dw / s1	$3 \times 3 \times 32$ dw	$112 \times 112 \times 32$
Conv / s1	$1 \times 1 \times 32 \times 64$	$112 \times 112 \times 32$
Conv dw / s2	$3 \times 3 \times 64$ dw	$112 \times 112 \times 64$
Conv / s1	$1 \times 1 \times 64 \times 128$	$56 \times 56 \times 64$
Conv dw / s1	$3 \times 3 \times 128$ dw	$56 \times 56 \times 128$
Conv / s1	$1 \times 1 \times 128 \times 128$	$56 \times 56 \times 128$
Conv dw / s2	$3 \times 3 \times 128$ dw	$56 \times 56 \times 128$
Conv / s1	$1 \times 1 \times 128 \times 256$	$28 \times 28 \times 128$
Conv dw / s1	$3 \times 3 \times 256$ dw	$28 \times 28 \times 256$
Conv / s1	$1 \times 1 \times 256 \times 256$	$28 \times 28 \times 256$
Conv dw / s2	$3 \times 3 \times 256$ dw	$28 \times 28 \times 256$
Conv / s1	$1 \times 1 \times 256 \times 512$	$14 \times 14 \times 256$
$5\times$ Conv dw / s1	$3 \times 3 \times 512$ dw	$14 \times 14 \times 512$
Conv / s1	$1 \times 1 \times 512 \times 512$	$14 \times 14 \times 512$
Conv dw / s2	$3 \times 3 \times 512$ dw	$14 \times 14 \times 512$
Conv / s1	$1 \times 1 \times 512 \times 1024$	$7 \times 7 \times 512$
Conv dw / s2	$3 \times 3 \times 1024$ dw	$7 \times 7 \times 1024$
Conv / s1	$1 \times 1 \times 1024 \times 1024$	$7 \times 7 \times 1024$
Avg Pool / s1	Pool 7×7	$7 \times 7 \times 1024$
FC / s1	1024×1000	$1 \times 1 \times 1024$
Softmax / s1	Classifier	$1 \times 1 \times 1000$

Figure 3: MobileNet architecture

Data Preprocessing

A. Train-Validation Split

First we partition the CIFAR-100 training dataset into into a new training set and a validation set, containing 40,000 and 10,000 data points respectively. The below code was used to partition training set into the train and validation:

```
def train_valid_split(train_dataset,
    valid_size, shuffle):
    num_train = len(train_dataset)
    indices = list(range(num_train))
    split = int(np.floor(valid_size *
    num_train))

    if shuffle:
        np.random.shuffle(indices)

# Create training and validation samplers
train_sampler = torch.utils.data.sampler.
    SubsetRandomSampler(train_idx)

valid_sampler = torch.utils.data.sampler.
    SubsetRandomSampler(valid_idx)

# Create training and validation dataloaders
train_loader = torch.utils.data.DataLoader(
    train_dataset, batch_size=batch_size,
    sampler=train_sampler, num_workers=
    num_workers, pin_memory=pin_memory)

valid_loader = torch.utils.data.DataLoader(
    valid_dataset, batch_size=batch_size,
    sampler=valid_sampler, num_workers=
    num_workers, pin_memory=pin_memory)
```

B. Class-Proportion

As we can see in the figure below, the class proportion of the new training set (after train-valid split) is fairly balanced across all 100 classes.

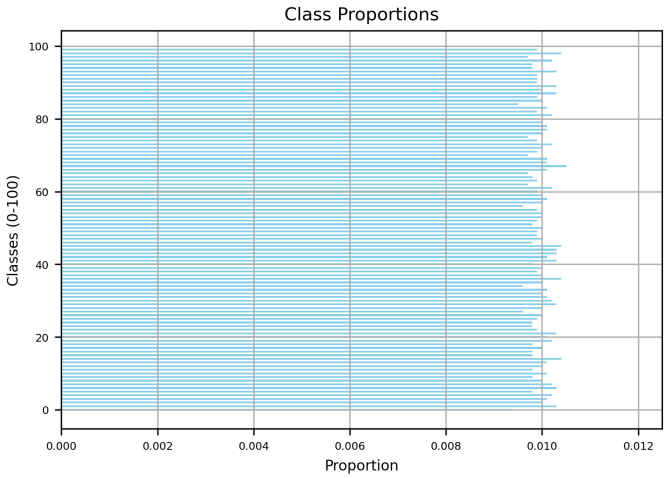

Figure 4: Proportion of each class in new training set

C. Mean and Standard Deviation

The mean and standard-deviation for each color channel in the entire training set was computed as below:

	Red	Blue	Green
Mean	0.5071	0.4865	0.4409
Standard Deviation	0.2009	0.1984	0.2023

Table 1: Mean and STD for each color channel

D. Augmentation

We use standard random horizontal flip and random cropping as data augmentation. The horizontal flip probability is 0.5. For random cropping, we apply 4-pixel paddings on all sides and crop a 32-by-32 patch from the image as instructed in the handout. We apply this augmentation on the new training dataset. The code below was used to perform the augmentation.

```
mean = (0.5071, 0.4865, 0.4409)
std = (0.2009, 0.1984, 0.2023)

if augment:
    transform_train = transforms.Compose([
        transforms.RandomCrop(32, padding=4),
        transforms.RandomHorizontalFlip(p=0.5),
        transforms.ToTensor(),
        transforms.Normalize(mean, std),
    ])
```

Learning Rate

A. Optimization using Learning Rates

The learning rate hyper-parameter determines the step size at each iteration while moving toward a minimum of a loss function. It's used in the update rule of the weights in gradient descent as follows:

$$\nabla J(w) = \left(\frac{\partial J}{\partial w_1}, \frac{\partial J}{\partial w_2}, \ldots, \frac{\partial J}{\partial w_n} \right)$$

Gradient of J(w) with respect to the weights

$$w_{\text{new}} = w_{\text{old}} - \eta \nabla J(w)$$

Update the weight values along the gradient

Here, w represents the weights, η is the learning rate, and $\nabla J(w)$ is the gradient of the loss function J *wrt* weights w.

The learning rate plays a crucial role in optimization. If it's too large, the algorithm might overshoot the optimal point and diverge, leading to unstable training. On the other hand, if it's too small, the algorithm will converge slowly, possibly getting stuck in a sub-optimal solution. Therefore, choosing an appropriate learning rate is essential for efficient training and achieving good performance.

B. Experiment parameters

I used the MobileNet model to conduct three trials. In each trial the model was trained for 15 epochs and random cropping and random horizontal flip methods of augmentation were used. Different learning rates was used for the trials.

C. Outcome

Learning rate	Epochs
0.05	15
0.5	15
0.01	15

Table 2: Parameter Settings for LR

	LR = 0.05	LR = 0.5	LR = 0.01
Training Loss	1.7617	1.9483	1.9655
Validation Loss	2.1723	2.1710	2.2422

Table 3: Training and validation loss after 15 epochs

	LR = 0.05	LR = 0.5	LR = 0.01
Training Accuracy	0.5041	0.4635	0.4606
Validation Accuracy	0.4371	0.4266	0.4144

Table 4: Training and validation accuracy after 15 epochs

D. Plots

Loss and accuracy on the training and validation set was monitored and is displayed in Figure 5.

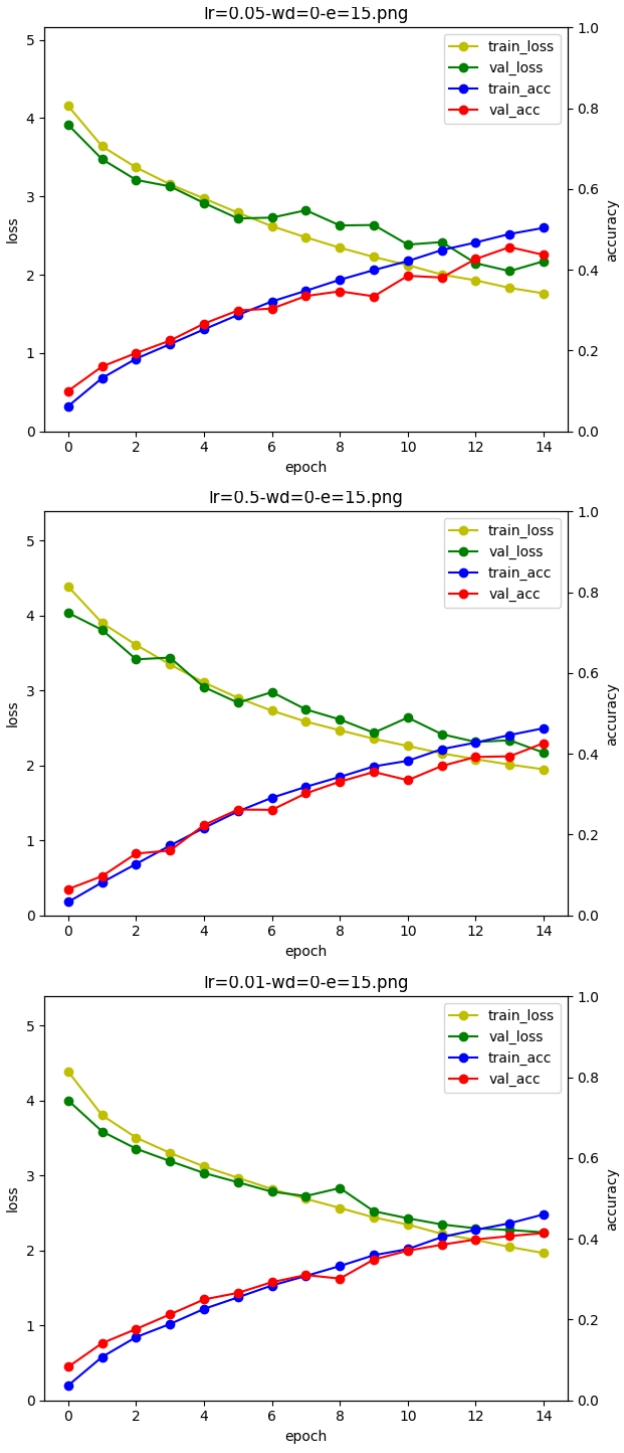

Figure 5: Plot of training and validation set loss and accuracy after 15 epochs.

E. Inference

LR = 0.05: This learning rate resulted in the **highest training and validation accuracy** among the three. It seems to be a good balance, allowing the model to learn effectively without overshooting the minima in the loss landscape.

LR = 0.5: This learning rate might be too high. A high learning rate can cause the model to converge quickly but it may also overshoot the minima or cause oscillations in the loss landscape.

LR = 0.01: This learning rate might be too low for the model. A low learning rate makes the model learn slowly, which might not be sufficient within 15 epochs. This could be why it has the lowest validation accuracy.
We identified **0.05 as the optimal learning rate** for our model.

Learning Rate Schedule

A. Cosine Annealing

Learning rate is one of the most important hyper-parameters while training neural networks. Instead of choosing a fixed learning rate throughout the training, it is often more effective to adjust the learning rate dynamically. This can be done using various scheduling methods such as step decay, exponential decay, or methods like cosine annealing.

The cosine annealing schedule is inspired by the shape of the half cosine curve, which starts at its maximum value and gradually decreases to its minimum. The learning rate at each epoch (or iteration) t is calculated using the formula:

$$\eta(t) = \eta_{\text{min}} + \frac{1}{2}(\eta_{\text{max}} - \eta_{\text{min}})(1 + \cos(\frac{T_{\text{cur}}}{T_{\text{max}}}\pi))$$

The cosine annealing learning rate schedule is a strategy used in training neural networks. The variable $\eta(t)$ is the learning rate at time t. The learning rate starts at η_{max} and gradually decreases to η_{min} in a half cosine cycle as training progresses from epoch 0 to T_{max}. After reaching η_{min} at T_{max}, it restarts the cycle by going back to η_{max}, creating a saw-tooth pattern.

B. Experiment parameters

I conducted two trials with a learning rate (lr) of 0.05 for 300 epochs. In the first trial we did not use any Learning Rate Scheduler (LRS). In the second trial I used the Cosine Annealing LRS.

Learning rate	Epochs	Scheduler
0.05	300	False
0.05	300	True

Table 5: Parameter Settings for LRS

C. Outcome

	LRS = False	LRS = True
Training Loss	0.0071	0.0012
Validation Loss	3.7522	3.2680

Table 6: Training and validation loss after 300 epochs

	LRS = False	LRS = True
Training Accuracy	0.9976	0.9997
Validation Accuracy	0.5682	0.5770

Table 7: Training and validation accuracy after 300 epochs

D. Plots

Loss and accuracy on the training and validation set was monitored and is displayed in Figure 6.

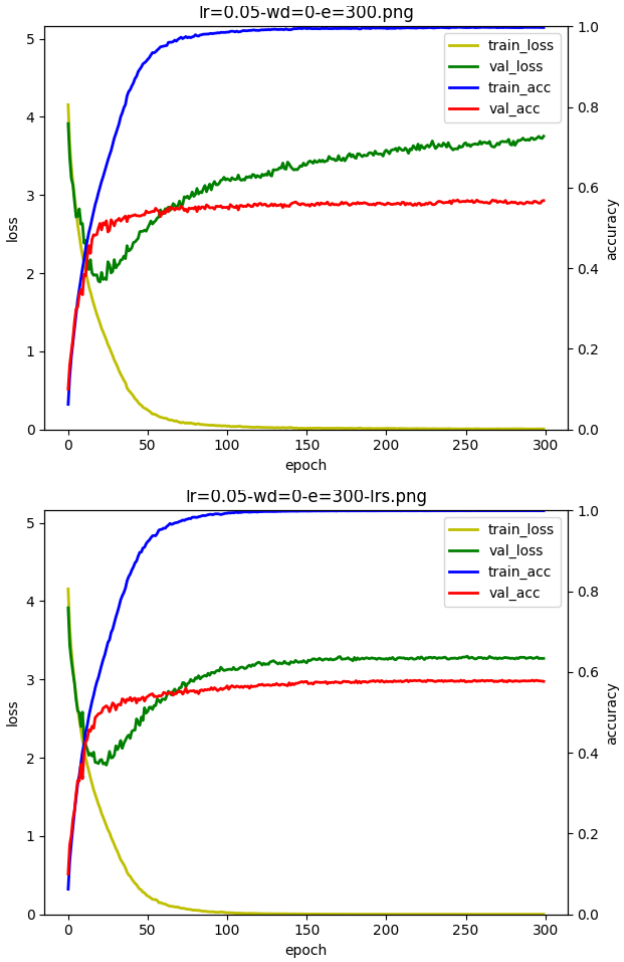

Figure 6: Plot of training and validation set loss and accuracy after 300 epochs without and with Cosine Annealing LRS

E. Inference

- The validation accuracy after 300 epochs in the two trials reached 56.82% and 57.70% respectively.
- The validation loss gradually increased throughout the 300 epochs in the first trial. The validation loss became almost constant after 100 epochs for the second trial.
- The plots for training accuracy, validation accuracy, and training loss exhibit similar trends.

The observations indicated that using the **Cosine Annealing** learning rate scheduler for our experiment is a better choice. It **achieved a better validation accuracy** as well as **managed to control the validation loss from overshooting**. The validation loss did not overshoot in the case when using the scheduler is possibly because the learning rate decreased with each passing epoch. This ensured that when a valley or a region of local minima was reached, the learning rate was not large enough to allow ascending the valley.

Weight Decay

A. Weight Decay in Gradient Descent

Weight decay (WD) is a regularization technique to prevent over-fitting by adding a penalty term to the loss function. The penalty term is proportional to the sum of the squares of the model parameters, which encourages the model to learn smaller weights, leading to a simpler model. Weight decay modifies the loss function as:

$$L_{\text{new}} = L_{\text{old}} + \frac{\lambda}{2} \sum_i w_i^2$$

$$w_{\text{new}} = w_{\text{old}} - \eta \left(\frac{\partial L_{\text{old}}}{\partial w} + \lambda w_{\text{old}} \right)$$

Weight decay adds a complexity penalty to the loss function, encouraging the learning of smaller weights and thus simpler models. This helps in reducing over-fitting by avoiding to learn a more complex or flexible model.

B. Experiment parameters

I conducted two trials to test our model with different weight decay (WD) values.

Learning rate	Epochs	Scheduler	WD co-eff
0.05	300	True	0.0001
0.05	300	True	0.0005

Table 8: Parameter Settings for weight-decay

C. Outcome

	WD = 0.0001	WD = 0.0005
Training Loss	0.0034	0.0122
Validation Loss	1.6740	1.3723

Table 9: Training and validation loss after 300 epochs

	WD = 0.0001	WD = 0.0005
Training Accuracy	0.9998	0.9997
Validation Accuracy	0.6311	0.6832

Table 10: Training and validation accuracy after 300 epochs

D. Plots

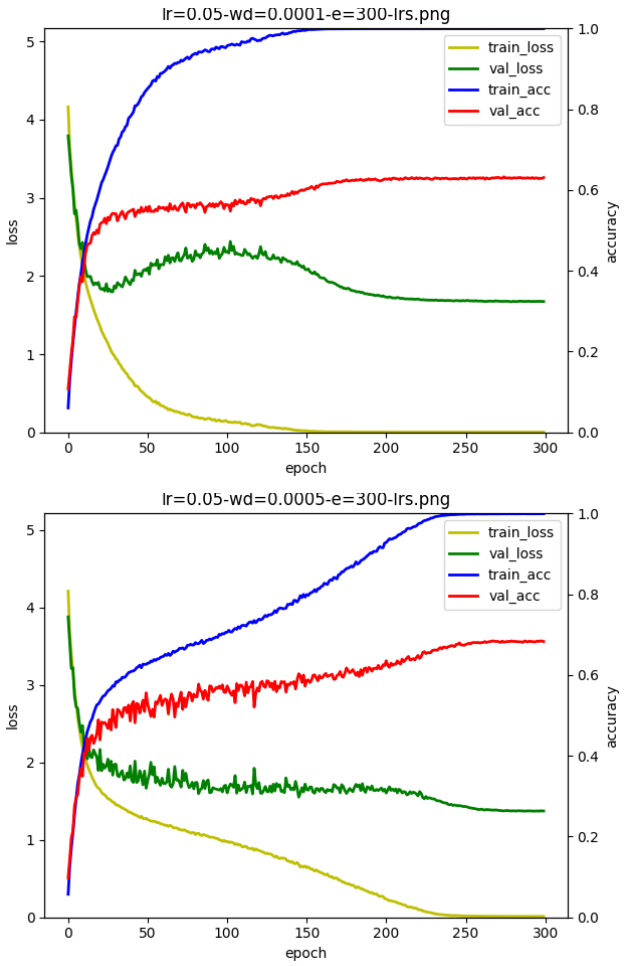

Figure 7: Plot of training and validation set loss and accuracy after 300 epochs with WD=0.0001 and WD=0.0005

E. Inference

- The validation accuracy after 300 epochs in the two trials reached 63.11% and 68.32% respectively.
- Validation loss decreased and accuracy increased significantly compared to last sets of trial when we were not using weight decay.
- The gap between the training accuracy and validation accuracy curve reduced (for $WD = 0.0005$).

The most significant observation was the **affect of regularization** introduced by the weight decay. We noticed (especially in the case of $WD = 0.0005$) that the training

and validation **accuracy had similar upward trends** during the first half of the training with **little gap**. This shows that weight decay prevented over-fitting by learning more general features and avoiding training data bias improving the accuracy on validation set. We observed that **weight decay co-efficient of** 0.0005 **is clearly the better choice** as it achieved **higher accuracy and reduced over-fitting**.

MixUp

A. MixUp and Data Augmentation

MixUp combines pairs of data points by linearly blending their features and labels. Given two input samples, X_i and X_j, and their labels, Y_i and Y_j, MixUp creates a new augmented sample X_{mix} and label Y_{mix} as:

$$X_{\text{mix}} = \lambda \cdot X_i + (1 - \lambda) \cdot X_j,$$
$$Y_{\text{mix}} = \lambda \cdot Y_i + (1 - \lambda) \cdot Y_j,$$

Here λ is a random value sampled from a beta distribution. MixUp works effectively by providing a form of regularization and promoting the model's ability to make predictions in regions between classes.

B. MixUp on CIFAR-100

We can visualize MixUp on the CIFAR 100 dataset using following plots:

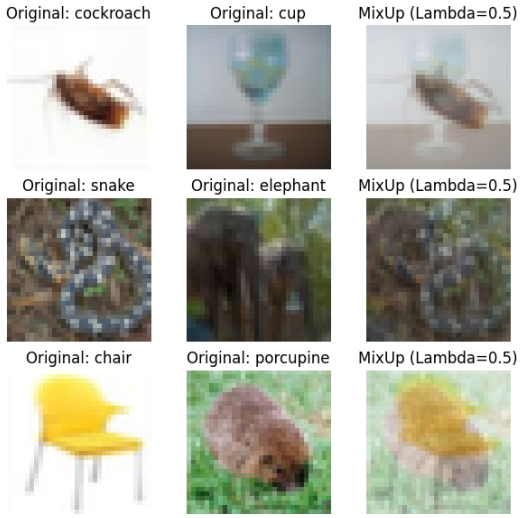

Figure 8: MixUp results on CIFAR 100 samples ($\alpha = 0.5$)

C. Experiment parameters

We used the below parameters to train the model and tested the model on the untouched CIFAR 100 test dataset.

LR	Epochs	Scheduler	WD co-eff	MixUp α
0.05	300	True (CA)	0.0005	0.2

Table 11: Parameter Settings for weight-decay

D. Plots

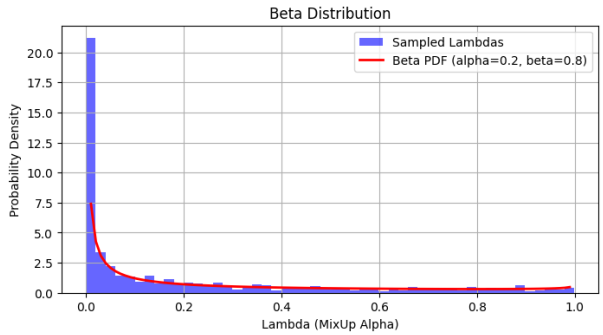

Figure 9: Beta Distribution | PDF and sampled λ

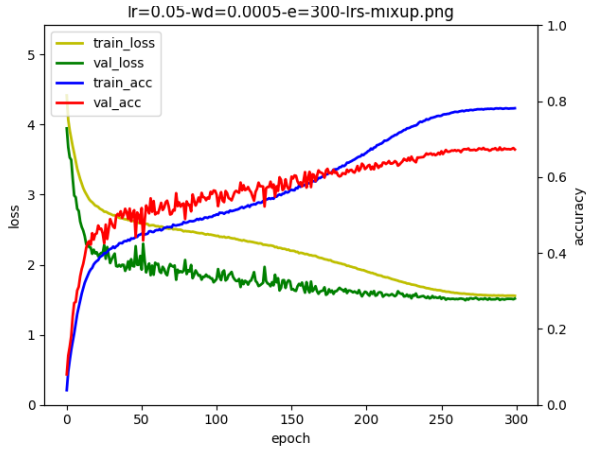

Figure 10: Loss and accuracy with mixup data augmentation

E. Inference

MixUp **reduced over-fitting even further** such that validation accuracy was leading the training accuracy till almost half the training period. It is interesting to note that the **validation accuracy decreased slightly**. This maybe due to the side-effects of MixUp. The model was now well trained and ready to be tested on unseen test dataset.

Results

I tested the trained MobileNet model on the CIFAR-100 test dataset with parameters from Table 11.

Train loss	Val loss	Train acc	Val acc	Test acc
1.5565	1.5169	78.18%	67.30%	67.63%

Table 12: Summary of Model Performance

The model achieved **test accuracy of 67.63%.** These findings demonstrate the importance of hyper-parameter tuning and data augmentation techniques in accomplishing better performance in deep learning tasks.