AI6103 - Homework Assignment

Rohin Kumar Maheswaran - G2303513K

School of Computer Science and Engineering
rohinkum001@e.ntu.edu.sg

Abstract

This study investigates how **deep neural network** performance is affected by hyperparameters like **data augmentation**, **weight decay**, **learning rate schedule**, and **initial learning rate**. The **MobileNet** network and the **CIFAR-100** dataset are used for experimentation in this study, which focuses on the trade-off between **optimization** and **regularization**. The findings demonstrate that the network's **training accuracy**, **validation accuracy**, **training loss**, and **validation accuracy** are all significantly affected by the selection of hyperparameters. We gain a better understanding of how to **optimize deep neural networks** for various tasks by analyzing empirical results and discussing possible explanations for observed phenomena.

Introduction

MobileNet

MobileNet is a convolutional neural network (CNN) designed specifically for mobile and embedded vision applications. It is based on a streamlined architecture that creates lightweight, low-latency deep neural networks for mobile and embedded devices using depthwise separable convolutions.[1]

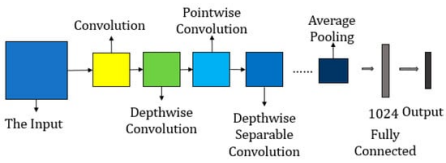

Figure 1: MobileNet Architecture Diagram

Depthwise separable convolutions are a type of factorized convolutions that form the foundation of MobileNet's architecture. A standard convolution is factorized into two parts: a depthwise convolution and a 1×1 pointwise convolution. Each input channel receives a single filter from the depthwise convolution, whereas the outputs of the depthwise convolution are combined by the pointwise convolution.[2]

MobileNets are small, low-latency CNNs with faster performance and smaller complexity compared to traditional CNNs. They are specifically designed for mobile and embedded devices, making them highly efficient and useful in real-world applications

CIFAR100

CIFAR-100 is a labeled subset of an 80 million tiny image dataset with 100 classes and 600 images each.Though there are more classes and images per class, it is comparable to CIFAR-10. For every class, there are 500 training images and 100 testing images. CIFAR-100 is frequently utilized as a benchmark dataset for image classification tasks.The combination of MobileNet and CIFAR-100 can lead to models that are both accurate and efficient, making them suitable for real-world applications that require image classification on mobile and embedded devices

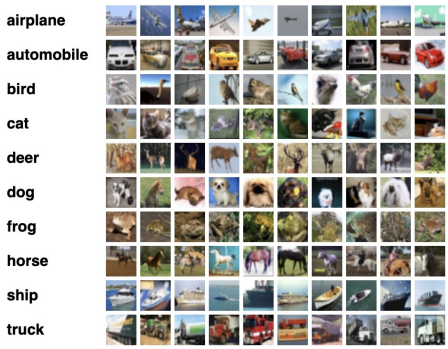

Figure 2: CIFAR-100 Dataset

CIFAR-100 images are 32x32 pixels in size and contain colored images of objects from 100 different classes organized into 20 superclasses. Every image has two labels: a coarse label (superclass) and a fine label (class). Ten thousand test images and fifty thousand training images make up the dataset. These images contain objects that are not uniformly positioned throughout the dataset and can be found at different points within the image. As a result, the models must be trained to identify and categorize objects in any image, even when using CIFAR-100.

Procedure

Data Preprocessing

A thorough data preprocessing step was carried out before the CIFAR-100 dataset was trained. This is an important step because it helps to understand the dataset in a more nuanced way and lays the groundwork for building a strong deep learning model. The following sections provide an explanation of the careful steps taken during the data preprocessing stage.

The CIFAR-100 training set was randomly divided, using a random seed of 0, into a new training set and a validation set. Sets with 40,000 and 10,000 data points, respectively, were produced by this division.

Listing 1: Random Split of CIFAR-100 Dataset

```
# Set random seed for reproducibility
torch.manual_seed(seed)

# Randomly split the dataset into
    training and validation sets
train_size = 40000
valid_size = 10000
train_dataset, valid_dataset =
    random_split(full_dataset, [
    train_size, valid_size])
```

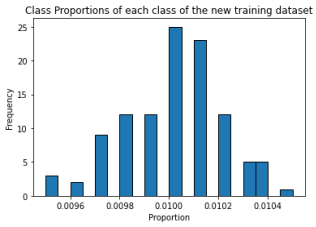

Figure 3: Class Propotions of New Training Dataset

Listing 2: Transformations

```
def get_train_valid_loader(dataset_dir,
    batch_size, shuffle, seed,
    save_images=False):
    transform = transforms.Compose([
        transforms.RandomHorizontalFlip(
            p=0.5),
        transforms.RandomCrop(32,
            padding=4),
        transforms.ToTensor(),
        transforms.Normalize(([0.4385,
            0.4181, 0.3775]), ([0.3004,
            0.2872, 0.2937]))
    ])
```

The mean and standard deviation values for every color channel in the training set were carefully calculated to guarantee the best possible data normalization. The model training phase is stabilized and accelerated by this normalization process.The data are then normalized using the calculated mean and standard deviation values.Random horizontal flips and random cropping with 4-pixel padding were two crucial data augmentation techniques that were adopted during training in an attempt to improve the generalization of the model.Together, these meticulously carried out preprocessing stages result in a carefully selected dataset that is ready for reliable training and validation. The Mean and Standard Deviation computed is as follows : Mean :[0.4385, 0.4181, 0.3775]), STD :[0.3004, 0.2872, 0.2937]

Listing 3: Calculation of Mean and Standard Deviation

```
# Compute mean and standard deviation
    for each color channel
loader = DataLoader(train_dataset,
    batch_size=len(train_dataset),
    shuffle=False)
data = next(iter(loader))
images, _ = data

mean = torch.mean(images, dim=(0, 2, 3))
std = torch.std(images, dim=(0, 2, 3))

# Print mean and standard deviation for
    each channel
print("Mean for each channel:", mean)
print("Standard Deviation for each
    channel:", std)
```

Intial Learning Rate

We looked into how various learning rates affected a neural network's functionality. There were three learning rates selected: 0.5, 0.05, and 0.01; each had a fixed batch size of 128. Random horizontal flipping and random cropping were two of the data augmentation techniques used; no learning rate schedule or weight decay was used.

Experimental Results:

Learning Rate: 0.5:

- Training Loss: 1.7054, Training Accuracy: 51.74%
- Validation Loss: 2.0027, Validation Accuracy: 45.94%

Training Curve: The model exhibits a steady decrease in training loss and an increase in training accuracy over epochs. However, the validation loss and accuracy seem to plateau after a certain point, indicating potential overfitting.

Learning Rate: 0.05:

- Training Loss: 1.7054, Training Accuracy: 51.74%
- Validation Loss: 2.0027, Validation Accuracy: 45.94%

Training Curve: Similar to the 0.5 learning rate, the model demonstrates improvement in training metrics, but the validation performance plateaus, suggesting a potential issue of overfitting.

Learning Rate: 0.01:

- Training Loss: 1.9655, Training Accuracy: 45.90%

- Validation Loss: 2.2476, Validation Accuracy: 40.66%

Training Curve: This learning rate leads to slower convergence, with both training and validation metrics improving gradually. However, the model achieves lower final training accuracy and validation accuracy compared to higher learning rates.

The following are the graphs of above observed results :

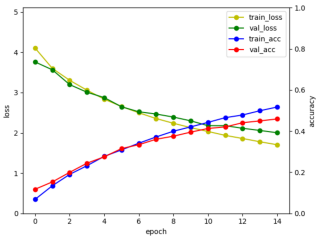

Figure 4: Training Curve for Learning rate = 0.5

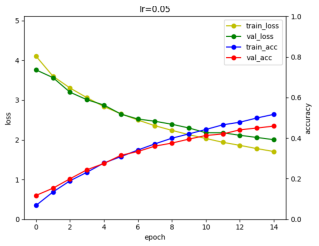

Figure 5: Training Curve for Learning rate = 0.05

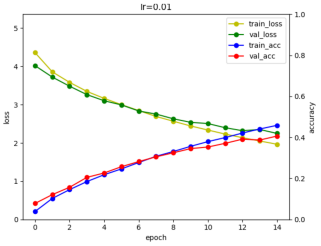

Figure 6: Training Curve for Learning rate = 0.01

Best Performing Learning Rate: The learning rate of 0.05 works best in terms of minimizing training loss. After 15 epochs, it achieves a lower training loss than the other learning rates. On the other hand, it seems that a learning rate of 0.5 produces better results in terms of validation metrics, with slightly lower validation loss and higher accuracy.

Possible reasons for the results

Overfitting: There may have been overfitting as evidenced by the plateau in validation metrics that was seen at both the higher learning rates (0.5 and 0.05). Reducing model complexity or implementing regularization strategies like dropout could help with this.

Impact of Learning Rate: A learning rate of 0.5 accelerates convergence at the risk of exceeding ideal parameters. A learning rate of 0.01 is more conservative but converges more slowly, whereas 0.05 finds a balance but may still be too high.

Hyperparameter tuning: To gain a more thorough understanding of the model's behavior, more experiments with a larger range of learning rates and other hyperparameters, such as batch size or model architecture, may be necessary.

Model Complexity: Influence on Learning Rates: The choice of learning rate might also be influenced by the complexity of the model. More complex models may require careful tuning of hyperparameters, including learning rate, to balance convergence and generalization.

The choice of the learning rate depends on the trade-off between training speed and generalization performance. The optimal learning rate lies between 0.05 and 0.5, and fine-tuning hyperparameters is essential for achieving better model performance.

Learning Rate Schedule

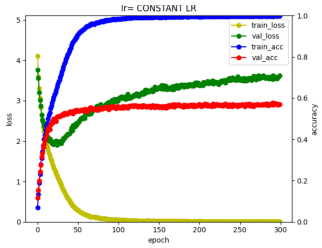

Figure 7: Constant Learning Rate of 300 Epochs

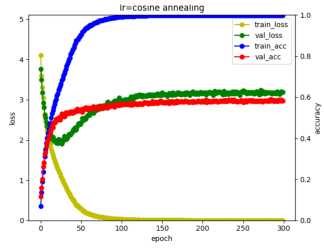

Figure 8: Cosine Annealing of 300 Epochs

Through a comparison with a constant learning rate strategy, we investigated the effects of cosine annealing on the training process in this experiment. A dynamic method for adjusting learning rates, the cosine annealing schedule gradually lowers learning rates over training epochs in a cosine-shaped fashion, starting at a higher initial learning rate.

The cosine annealing learning rate schedule can be represented mathematically as follows:

$$\eta_t = \eta_{min} + \frac{1}{2}(\eta_{max} - \eta_{min}) \times \left(1 + \cos\left(\frac{t}{T} \times \pi\right)\right)$$

η_t is the learning rate at epoch t, η_{min} is the minimum learning rate, η_{max} is the maximum learning rate (chosen based on the best learning rate identified earlier), T is the total number of epochs, and t is the current epoch.

In the first experimental setting, the model was trained for 300 epochs with a constant learning rate. The final losses and accuracy values for both the training and validation sets were as follows:

Constant Learning Rate (300 epochs):

- Final Training Loss: 0.0068
- Final Training Accuracy: 99.80%
- Final Validation Loss: 3.6180
- Final Validation Accuracy: 56.97%

In the second setting, the model was trained for 300 epochs with cosine annealing, gradually decreasing the initial learning rate to zero. The results were:

Cosine Annealing (300 epochs):

- Final Training Loss: 0.0013
- Final Training Accuracy: 99.97%
- Final Validation Loss: 3.1959
- Final Validation Accuracy: 58.25%

Constant Learning Rate:

Accuracy and Training Loss: The model attains a high accuracy of 99.80% and a very low training loss of 0.0068. This suggests that the model has effectively memorized the training data.

Validation Accuracy and Loss: On the validation set, however, the accuracy is lower (56.97%) and the loss is higher (3.6180). This indicates overfitting; the model doesn't generalize well to unseen data.

Cosine Annealing:

Accuracy and Loss Training: The training accuracy is marginally higher (99.97%) and the training loss is even lower (0.0013) when cosine annealing is used. When using the training set of data, the model performs exceptionally well.

Validation Accuracy and Loss: In comparison to the constant learning rate, the validation accuracy is higher (58.25%) and the validation loss is likewise lower (3.1959). Even though overfitting persists, it seems to generalize better.

Potential Causes of Disparities:

Overfitting: The model memorized the training data without adapting well to new data, which was probably caused by the constant learning rate. Cosine annealing modifies the learning rate dynamically to help reduce overfitting.

Learning Rate Schedule: A dynamic learning rate schedule is produced by cosine annealing, which enables the model to converge more quickly and possibly break out of local minima.

Generalization: It appears that the cosine annealing method promotes greater generalization, which raises validation accuracy.

Weight Decay

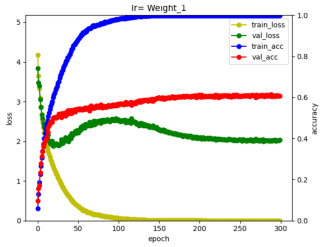

Figure 9: $\lambda = 5 \times 10^{-4}$

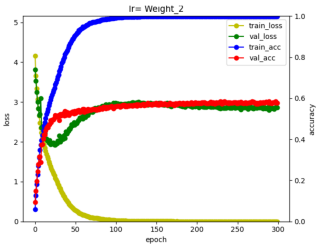

Figure 10: $\lambda = 1 \times 10^{-4}$

For $\lambda = 5 \times 10^{-4}$:

- Final Training Loss: 0.0024
- Final Training Accuracy: 0.9998
- Final Validation Loss: 2.0284
- Final Validation Accuracy: 0.6079

For $\lambda = 1 \times 10^{-4}$:

- Final Training Loss: 0.0013
- Final Training Accuracy: 0.9997
- Final Validation Loss: 2.8629
- Final Validation Accuracy: 0.5775

These values represent the state of the model after 300 epochs of training. Training and validation losses provide insights into how well the model is fitting the training data and generalizing to new data. Accuracy values indicate the model's overall performance.

Data Augmentation :

Probability Density Function (PDF) for Beta Distribution:

When using the mixup augmentation technique, a beta distribution is sampled and the hyperparameter is set to 0.2. A probability density function (PDF) plot for the beta distribution with =0.2 can be made to see this. The probability of sampling different values during a mixup would be displayed in this plot.training process involved running the training for 300 epochs. This is a substantial number of epochs and suggests that the model had ample opportunity to learn from the data.

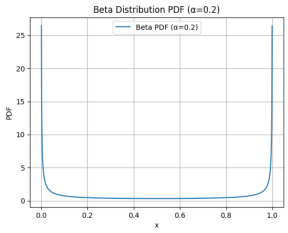

Figure 11: BETA DISTRIBUTION

Final Losses and Accuracy Values:

Training Set:

- Final Loss: 0.0024
- Final Accuracy: 99.98%

Validation Set:

- Final Loss: 2.0554
- Final Accuracy: 60.03%

These metrics indicate that the model has achieved near-perfect accuracy on the training set but shows a lower accuracy on the validation set. This discrepancy between training and validation accuracies suggests a possibility of overfitting.

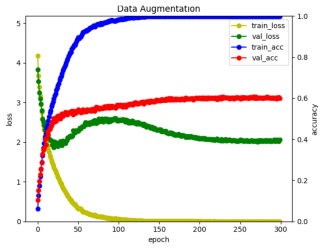

Figure 12: Final Training Curve with Best Parameters

Analysis: Effects of Mixup Augmentation

Improved Training Accuracy:

- The model achieved a remarkable training accuracy of 99.98%, indicating that the mixup augmentation helped the model generalize well to the training data.

Generalization Challenge on Validation Set:

- The validation accuracy (60.03%) is lower than the training accuracy, suggesting that there might be challenges in generalizing the mixup augmentation to unseen data.
- This discrepancy could be due to the model overfitting or the need for fine-tuning hyperparameters.

Test Set Accuracy

Test Accuracy: 60.69%

The mixup augmentation technique, with α set to 0.2, demonstrated promising results in enhancing the model's training accuracy. However, challenges were observed in generalizing the benefits to the validation set. Further experimentation and fine-tuning of hyperparameters are recommended for better performance on unseen data. The hold-out test accuracy of 60.69% provides an estimate of the model's expected performance on similar images in the future.

Conclusion

In conclusion, the experimentation with MobileNet and CIFAR-100 dataset has provided valuable insights into the optimization of deep learning models for image classification tasks

The utilization of MobileNet, with its streamlined architecture and depthwise separable convolutions, proved effective in creating lightweight and low-latency CNNs suitable for real-world applications. The focus on CIFAR-100, a labeled subset with 100 classes and 32x32 pixel images, allowed for a practical benchmark for image classification.

With its simplified architecture and depthwise separable convolutions, MobileNet was successfully used to create lightweight, low-latency CNNs that are appropriate for real-world applications. The emphasis on CIFAR-100, a labeled subset of 32x32 pixel images and 100 classes, made a useful benchmark for image classification possible.

The foundation for creating a strong deep learning model was laid by the meticulous data preprocessing procedures, which included splitting the dataset at random and applying deliberate transformations. Data augmentation methods like random horizontal flips and random cropping improved model generalization, and the computed mean and standard deviation values were essential for data normalization. trade-offs between convergence speed and generalization performance. The findings suggested that a learning rate of 0.05 struck a balance, demonstrating better results in terms of both training and validation metrics.

Further investigation into learning rate schedules revealed that cosine annealing exhibited advantages over a constant learning rate strategy. The dynamic adjustment of learning rates contributed to improved generalization, as reflected in higher validation accuracy and lower validation loss.

The introduction of weight decay was explored, and the results indicated its impact on the final training and validation metrics. Fine-tuning the weight decay parameter allowed for a balance between fitting the training data and generalizing to new data.

The incorporation of mixup augmentation with a beta distribution parameter of 0.2 showcased impressive training accuracy but raised challenges in generalization to the validation set. The observed discrepancy between training and validation accuracies suggests the need for further investigation, possibly involving fine-tuning hyperparameters or addressing potential overfitting.

References

[1] MobileNets: Efficient Convolutional Neural Networks for Mobile Vision Applications by Andrew G. Howard, Menglong Zhu, Bo Chen, Dmitry Kalenichenko, Weijun Wang, Tobias Weyand, Marco Andreetto, and Hartwig Adam, arXiv preprint arXiv:1704.04861, 2017.

[2] Searching for MobileNetV3 by Andrew Howard, Mark Sandler, Grace Chu, Liang-Chieh Chen, Bo Chen, Mingxing Tan, Quoc V. Le, and Vivienne Sze, arXiv preprint arXiv:1905.02244, 2019.

[3] OpenAI's GPT-3.5 model