Deep Learning Mini-Project

Team Cerberus
Anubha Singh[1], Kavya Gupta[2], Khushi Sharma[3]
New York University
[1]as18806@nyu.edu [2]kg3373@nyu.edu [3]ks7406@nyu.edu

Abstract

Convolutional Neural Networks (CNNs) have a tendency to use a big number of layers in their network, but as there is a increase in the number of layers the problem of vanishing/exploding gradient arises. Therefore, increasing the test and training error rates. This issue can be combatted using ResNets. In this paper, we create a ResNet architecture where we achieve an accuracy of 95.01%. The code can be found on https://github.com/itskavyagupta/NYU-DL-Cerberus

Overview

We solve the problem of vanishing/exploding gradients using ResNets, a network with skipped connections. The key component in ResNet models is a residual block that implements:

$$ReLU(S(x) + F(x))$$

where $S(x)$ refers to the skipped connection and $F(x)$ is a block that implements $conv- > BN- > relu- > conv- > BN$; here, "BN" stands for batch normalization. Chaining such blocks serially gives a deep ResNet. Hyper-parameters (de-sign variables) in such architectures include:

- Ci , the number of channels in the ith layer.
- Fi , the filter size in the ith layer
- Ki , the kernel size in the ith skip connection
- P, the pool size in the average pool layer, etc.

Here our goal is to create a ResNet model from scratch using less than 5 million parameters.

Our model is a variant of the ResNet model that comprises of basic building blocks called BasicBlocks, each consisting of two convolutional layers with batch normalization and ReLU activation. It consists of multiple layers of these blocks, with the number of blocks per layer specified by the num_blocks parameter.

The final parameters and hyper parameters in our model are:

- Ci: [64, 128, 256]
- Fi: 3x3
- Ki: 3x3
- P: 8x8
- Blocks: [3, 5, 3]

The model processes input images through convolutional layers, applies residual blocks to extract features, and uses average pooling to reduce spatial dimensions. Finally, it employs a linear layer to produce class predictions. Our model is designed to address vanishing/exploding gradients by utilizing skip connections, enabling the training of deeper neural networks more effectively.

We are creating and training multiple models on the $CIFAR-10$ dataset to classify images. Our best model achieves the accuracy of **95.01%** for **ResNet-18** with **4,918,602** parameters. To implement this we used a learning rate of **0.01**, along with the loss function as **Cross Entropy Loss** and the optimizer as **sgd**.

Our final model has the following hyper-parameters: Batch Size: 128, Optimizer: sgd, Learning Rate: 0.01, Momentum: 0.9, Weight Decay: 0.0005, N: 3, B: [3, 5, 3], C: [64, 128, 256]

The techniques used to achieve the best configuration of hyper parameters is discussed in the next section.

Methodology

Firstly, we partitioned our $CIFAR-10$ dataset into training and testing subsets. Following this, we applied transformation methods to both sets. Furthermore, we integrated model checkpoints into our code, saving our model every 25 epochs.

Hyper Parameters

Layers N, Residual Blocks B and Channels C: Our process began with creating a basic ResNet model with 3 and 4 layers. We tested this different permutations of layers with another set of permutations of residual block sizes. Based on our experiments, we realized that we achieved the best results with 3 layers.

Overall, on the basis of our finding we realized that the models worked better with 3 layers.

Table 1.1 summarises the different hyper parameters we used.

Model	No. of Params	N	B	C
ResNet-26	4771146	3	5,4,3	64, 128, 256
ResNet-28-1	4845130	3	6,4,3	64, 128, 256
ResNet-24	4918602	3	3,5,3	64,128,256
ResNet-30	4919114	3	7,4,3	64, 128, 256
ResNet-90	4947402	3	16, 16,12	32, 64, 128
ResNet-26-1	4992586	3	4,5,3	64, 128, 256
ResNet-32	4754218	4	4,4,4,3	32,64,128,256
ResNet-10	4903242	4	1, 1, 1, 1	64,128,256,512
ResNet-12	4977226	4	2, 1, 1, 1	64, 128, 256, 512
ResNet-56	4989194	4	16, 4, 4, 3	32, 64, 128, 256

Table 1.1: List of Hyper parameters used

Our observations based on the experiments are as follows:

- Comparing models with different depths, we observe that deeper models do not necessarily lead to higher accuracy. For instance, ResNet-28, with a depth of 28 layers, achieves the highest accuracy of 87.836%, outperforming both ResNet-10 and ResNet-26 despite their shallower architectures.
- We notice that while comparing 3 an 4 layer architectures, the average accuracy is better for the 3 layer models.
- There's a trade-off between model complexity (measured by the number of parameters) and accuracy. While some models, like ResNet-32, have relatively fewer parameters (4,754,218) and achieve high accuracy (86.11%), others, such as ResNet-26-1, have more parameters (4,992,586) but slightly lower accuracy (85.49%).

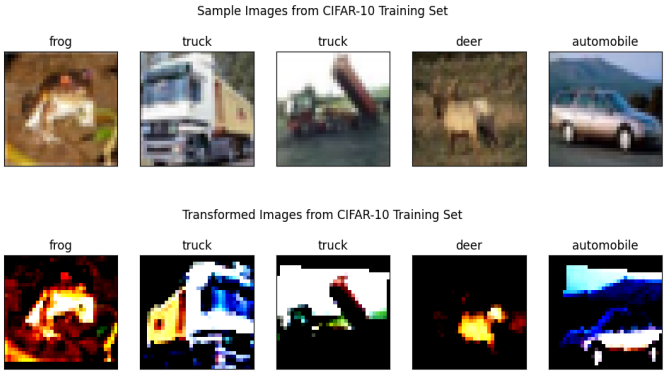
Data Augmentation

To begin with, we performed basic data augmentation to increase the number of data points by altering the existing data. This data augmentation was carried out using **transformers**. Specifically, we applied transformations such as random cropping, horizontal flipping, and random rotation to the training dataset. These transformations augmented the dataset and standardized input images, enhancing the model's ability to generalize and learn effectively. Additionally, for the testing dataset, simpler transformations were applied, including conversion to tensors and normalization using the same statistics as the training set, ensuring consistency in prepossessing between training and testing phases.

Apart from using transformations, we also used additional data augmentation techniques. The dynamic visualisations of each are also present in our Github.

- **Cutout** is a data augmentation technique widely used for image classification tasks. It involves randomly masking out square regions of input images during training by setting the pixel values in these regions to zero.

  Here, we introduced two parameters: number of holes (size of the above mentioned square) and size of holes. Where we used one hole of size 8 (i.e. we used an 8x8 square.)



- **Mixup** is another data augmentation technique that blends pairs of input samples and their corresponding labels to create new synthetic samples. By interpolating between the features and labels of two randomly chosen samples, mixup encourages the model to learn more robust and generalized representations, ultimately reducing overfitting and improving generalization performance.

  In our implementation, we used the parameter alpha as 0.5, which makes the distribution becomes more skewed towards 0 and 1. This results in λ values that are often very close to 0 or 1, meaning that one of the images will dominate the mix more frequently.
- **Cutout and Mixup** Combining Cutout and Mixup in deep learning can yield even better regularization and generalization performance. By integrating both techniques into the data augmentation pipeline, models can benefit from increased diversity in the training data and more robust feature learning.

  Here we realised that this performed better than Mixup, but worse than Cutout. Refer Figure 2.
- **CutMix** During training, CutMix randomly selects a patch from one image and replaces it with a patch from another image in the dataset. This process blends the two images together, creating a new mixed image. The corresponding labels are also mixed proportionally to the area of the patches. In our implementation, we used alpha again, where alpha was equal to 1. This special case makes the Beta distribution uniform across [0, 1], giving all mixing ratios an equal chance. This provides a very random and varied mixing behavior.

At the end, based on the test accuracies and test losses observed, we realised that using cutout was the most efficient technique. Refer Figure 2 and Figure 3.

Optimizers and Learning Rates

We utilized Wandb to explore the impact of various optimizers and learning rates on our top models. In this endeavor, we

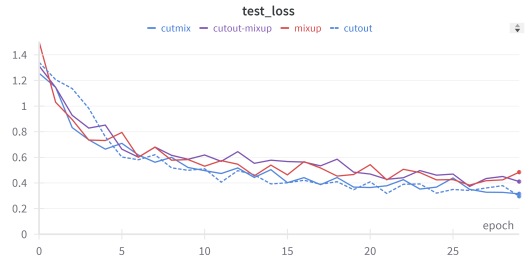


Figure 2: Tess Loss VS Epoch for 4 data augmentation techniques.

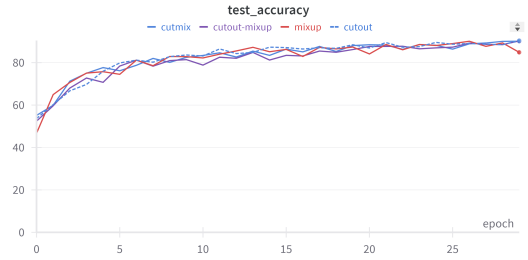


Figure 3: Tess Acc VS Epoch for 4 data augmentation techniques.

examined 10 models in conjunction with three distinct optimizers: $AdaDelta$, SGD, and $AdaDelta - clipping$. Additionally, we investigated two learning rates: 0.1 and 0.01.

Further, we introduced a learning rate decay strategy to further optimize model performance. Specifically, this strategy involved reducing the learning rate by a factor of 0.1 every 80 epochs, ensuring that the model's optimization process was fine-tuned over time.

Additionally, to enhance the training process and prevent overfitting, we implemented annealing with a cycle of 50 epochs. [Ref Figure 4.] This annealing technique allowed for periodic adjustments to the learning rate, promoting stable convergence and improving the model's ability to generalize to unseen data. Through these strategic adjustments, we aimed to maximize the effectiveness of the training process and ultimately enhance the model's predictive capabilities.

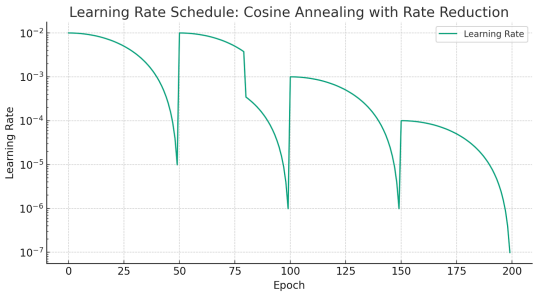


Figure 4: Cosine Annealing with Rate Reduction.

We also used the concept of weight decay, where the learning rate decreases as the number of epochs increases. For each of our configurations we use the weight decay as 0.0005 and momentum as 0.9.

Results

After testing our model with difference configurations, we achieved the best accuracy of 95.01%. The entire model is as follows:

Total Parameters: 4,918,602
Batch Size: 128
Optimizer: sgd
Learning Rate: 0.01
Momentum: 0.9
Weight Decay: 0.0005
N: 3
B: [3, 5, 3]
C: [64, 128, 256]

We experimented with various ResNet architectures on our dataset, and this particular model yielded the most promising results. During the initial training phase for 30 epochs with cutout augmentation, we achieved a notable test accuracy of 90.43%, surpassing the performance of all other models at that stage. Subsequently, we extended the training duration to 200 epochs, incorporating a learning rate decay strategy, where the learning rate decreased by a factor of 0.1 every 80 epochs, and implemented annealing with a cycle of 50.

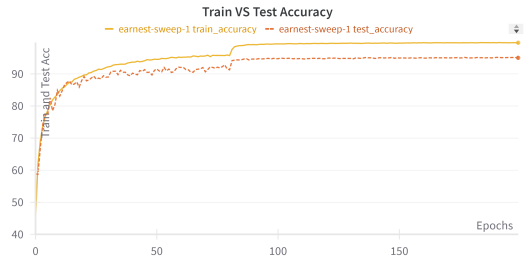


Figure 5: Train VS Test Accuracy for 200 epoch run

Conclusion

In this paper, we experimented with different hyper parameters, optimizers and learning rates to classify images in the $CIFAR-10$ dataset. We achieved an accuracy of 95.01% after training the model for over 200 epochs.

References

Engelmore, R.; and Morgan, A., eds. 1986. *Blackboard Systems*. Reading, Mass.: Addison-Wesley.