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 combat-ted using ResNets. In this paper, we create a ResNet architecture where we achieve a test accuracy of 95.01%.

Overview

ResNets are deep neural networks that learn residual functions using skip 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.
Model parameters (de-sign variables) in such architectures include:

- B , the number of blocks in each block layer.
- C , the number of channels in the ith layer.
- F , the filter size in the ith layer
- K , the kernel size in the ith skip connection
- P, the pool size in the average pool layer, etc.

The widely-implemented ResNet model comprises 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 each of these block layers having the same channel size. For example, the standard ResNet-18 model has 4 block layers of size B = [2, 2, 2, 2] of channel sizes C = [64, 128, 256, 512]. 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 goal is to train a ResNet model from scratch using less than 5 million parameters. We created and trained multiple models which are variants of the above explained model implementation on the $CIFAR - 10$ dataset to classify images. Our best model achieves the test accuracy of **95.01%** for **ResNet-18** with **4,918,602** parameters.

The final configurations of our model are:

- B: [3, 5, 3]
- C: [64, 128, 256]
- F: 3x3
- K: 3x3
- P: 8x8

The hyperparameters run with our final model are:

- Batch Size: 128,
- Optimizer: sgd,
- Learning Rate: 0.01,
- Momentum: 0.9,
- Weight Decay: 0.0005,
- Annealing Cosine: 50 cycles,
- Learning Rate Decay: by 0.1,

The techniques used to achieve the best configuration of model configurations and hyperparameters is discussed in the next section.

The code can be found in our GitHub repo.

Methodology

Firstly, we partitioned our $CIFAR - 10$ dataset into training and testing subsets. Following this, we applied basic data augmentation techniques on our dataset - including Random Flipping, Random Cropping and Random Rotation - to the training dataset. Additionally, we normalized both training and test datasets. Furthermore, we integrated model checkpoints into our code, saving our model every 25 epochs.

Model Parameters

Number of Block Layers N, Blocks Layer Sizes B and Channels C: Our process began with creating a basic ResNetCustom model which can be run with both 3 and 4 block layers (the original implementation has a fixed configuration of 4 layers). We tested different permutations of

Copyright © 2024, Association for the Advancement of Artificial Intelligence (www.aaai.org). All rights reserved.

number of block layers, sizes of those block layers and their channel sizes, all while maintaining the limit of 5million total parameters. Table 1.1 summarises the different hyper parameters we used. Our experiments showed us that we achieved the best results with 3 block layers.

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 Model Configurations Experimented with

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-90 and ResNet-56 despite their deeper architectures.
- We notice that while comparing 3 an 4 block 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 **transforms**. 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, normalization using the same statistics as the training set was applied, ensuring consistency in prepossessing between training and testing phases.

Sample Images from CIFAR-10 Training Set
frog
truck
truck
deer
automobile
Transformed Images from CIFAR-10 Training Set
frog
truck
truck
deer
automobile

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.

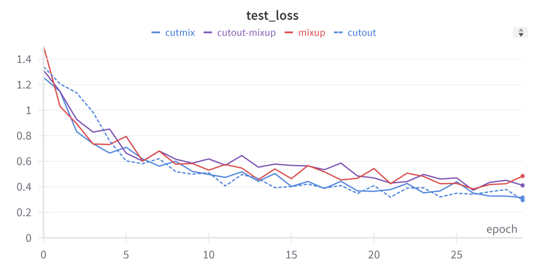

Figure 2: Tess Loss VS Epoch for 4 data augmentation techniques.

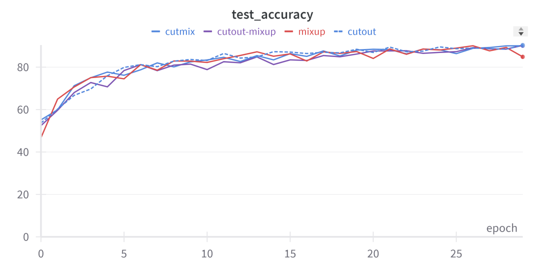

Figure 3: Tess Acc VS Epoch for 4 data augmentation techniques.

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 examined 10 models in conjunction with three distinct optimizers: $AdaDelta, SGD$, and $AdaDelta - clipping$. We decided on these optimizers after preliminary runs where we saw optimizers like Adam and AdaGrad didn't give great results with our models.

- SGD: This updates model parameters using gradients of the loss function with respect to the parameters.
- AdaDelta: It dynamically adjusts the learning rate during training, eliminating the need for manual tuning.
- AdaDelta with Gradient Clipping: Gradient Clipping is often used to better deal with issues of vanishing/exploding graidents where larger gradients are clipped off.

Additionally, we investigated two learning rates: 0.1 and 0.01.

Further, we introduced a learning rate decay strategy to optimize model performance. Specifically, this strategy involved reducing the learning rate by a factor of 10 every 80 epochs, ensuring that the model's optimization process was fine-tuned over time.

Additionally, to enhance the training process and prevent over fitting, 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.

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.

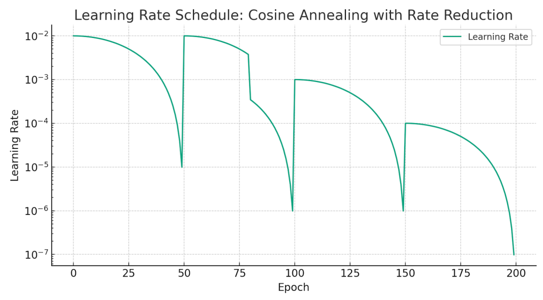

Figure 4: Cosine Annealing with Rate Reduction.

Wandb

WandB, short for Weights and Biases, is a popular platform helps track and visualize machine learning experiments. It provides tools for experiment tracking, visualization, and collaboration, making it easier to manage and understand complex machine learning projects.

In order to conduct and visualize our various parameter configurations, we utilized this platform. All our interactive visualizations can be found at All Model Experiments and Data Augmentation Strategies.

Results

After testing our model with difference configurations, we achieved the best accuracy of 95.01% and a loss of 0.1966. 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

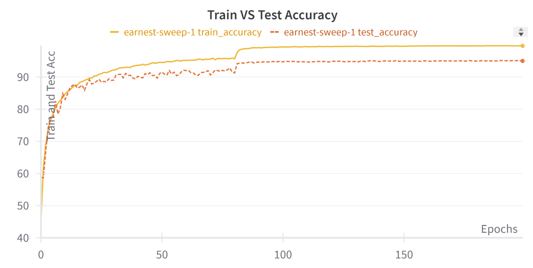

Figure 5: Train VS Test Accuracy for 200 epoch run

every 80 epochs, and implemented annealing with a cycle of 50.

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

Github. 2021. Train CIFAR10 with PyTorch. https://github.com/kuangliu/pytorch-cifar. Accessed: 2024-04-12.

PyTorch Developers. 2024. CutMix and MixUp: PyTorch Documentation.

Robinson, A. L. 2022. Non-deep Networks. In *Advances in Neural Information Processing Systems*.

Tsang, S. H. 2022a. CutMix: Regularization Strategy to Train Strong Classifiers with Localizable Features.

Tsang, S. H. 2022b. Cutout, Mixup, and CutMix: Implementing Modern Image Augmentations in PyTorch.

Yun, S.; Han, D.; Oh, S. J.; Chun, S.; Choe, J.; Yoo, Y. J.; Yoo, J.; Oh, J. H.; Lee, J.; and Kim, J. H. 2019. CutMix: Regularization Strategy to Train Strong Classifiers with Localizable Features.