CIFAR-10 image classification with a modified residual network (ResNet) architecture

Aman Ayan Shah(as16600), Jay Rana(jpr8961), Meet Nirav Diwan(md5517)

Github Repository:
https://github.com/jyrana/Mini-Project-CS-GY-6953-ECE-GY-7123-

Abstract

This mini-project involved designing a modified ResNet architecture for image classification on the CIFAR-10 dataset, with the constraint that the model must have no more than 5 million trainable parameters. The ResNet architecture was implemented with skip connections and residual blocks, and hyperparameters such as the number of channels, filter size, kernel size, and pool size were adjusted to optimize model accuracy. Various optimization techniques, data augmentation strategies, regularizers, and choice of the learning rate, batch size, and epochs were experimented with to further improve model performance. The final model was trained from scratch and achieved high accuracy on the CIFAR-10 test set within the constraints of the project.

Introduction

Deep neural networks have revolutionized the field of image classification by producing remarkable results. However, as the number of layers in a network increases, a degradation problem and vanishing gradient descent can occur, which hinder the model's ability to learn and improve. In this report, we investigate the Modified ResNet-18 model on the CIFAR-10 dataset, while adhering to a constraint of 5 million parameters. Residual networks enable the stacking of deep networks without degradation or vanishing gradient descent, by utilizing skipped connections in a building block known as a residual block (figure 1). We compare different variations of ResNet-18 architectures using various optimizers and hyperparameters to maximize accuracy. After training, all three models reached their peak performance at approximately the same number of epochs. The accuracy of the models on the test dataset varied between 87.76% and 90.58%. Our study presents the final test accuracy, model architecture, and the number of parameters, while highlighting the significance of residual blocks in building efficient deep neural networks.

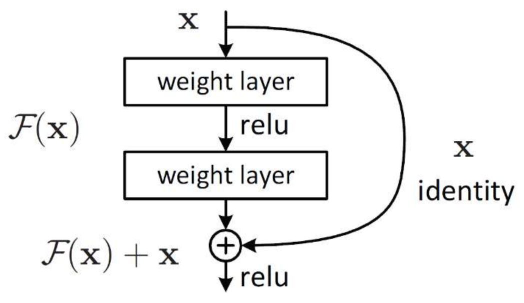

Fig 1 Residual Block

Dataset

The CIFAR-10 dataset is a widely used benchmark dataset for image classification tasks. It consists of 60,000 color images in 10 classes, each containing 6,000 images of size 32x32 pixels. The classes include commonly found objects such as airplanes, automobiles, birds, cats, and dogs, among others. This dataset is challenging due to the small size of the images and the diversity of the classes, making it an ideal choice for evaluating the performance of deep learning models.

Augmentation

Data Augmentation is a widely used technique in deep learning, which is particularly useful when working with small datasets like CIFAR-10. In this project, we have utilized data augmentation to increase the diversity and quantity of the training data. Specifically, we applied a range of transformations to the training images, including random rotations, horizontal flips, and shifts in the directions. These transformations help to create additional training samples with variations that the model is likely to encounter during the testing phase.

We used the PyTorch library's built-in data augmentation functions to implement these transformations. By augmenting the training data, we were able to reduce the risk of overfitting and improve the generalization ability of our model. Through data augmentation, we were able to increase the number of samples for each class, leading to better performance on the test data. Overall, data augmentation is a powerful technique that can significantly improve the performance of deep learning models, particularly on small datasets like CIFAR-10. The following data augmentation techniques were used to build the models.

1. Random crop with padding:
This transformation randomly crops a 32x32 pixel patch from the input image, after first adding a padding of 4 pixels to the boundaries by using transforms.RandomCrop(32, padding=4) function. The padding helps to ensure that important features in the input image are not lost during cropping. Random cropping helps to introduce variations in the training data, by selecting different regions of the image during each epoch of training. This transformation can help the model to learn more robust features, by forcing it to focus on different regions of the input image during training.
Additionally, the padding helps to reduce the impact of border effects, which can occur when the model is applied to the input image's edges. By padding the image, we ensure that the cropped patches always contain important features of the input image, regardless of their position in the image.

2. Random horizontal flip:
Another data augmentation technique that we used in our project is transforms.RandomHorizontalFlip(). This transformation randomly flips the input image horizontally with a probability of 0.5 during each epoch of training. Horizontal flipping can introduce variations in the training data, which helps to make the model more robust to changes in the orientation of the input image. For example, an image of a dog facing left might appear differently to the model than the same image flipped horizontally to show the dog facing right. By randomly flipping the input images during training, we can help the model learn to recognize features that are invariant to such changes.
By flipping the images, we can effectively double the size of the training set for each class, helping to balance the number of samples for each class.

3. Random rotation:
Another data augmentation technique that we employed in our project is transforms.RandomRotation(degrees=15). This transformation randomly rotates the input image by a certain number of degrees (up to 15 degrees in our case) during each epoch of training.
Random rotation is a powerful technique for introducing variations in the training data, which can help the model learn to recognize features that are invariant to different orientations. For example, an image of a car might appear differently to the model if it is rotated slightly, but the model should still be able to recognize it as a car. By randomly rotating the input images during training, we can help the model learn to recognize these invariant features.

Moreover, the transforms.RandomRotation() transformation can also help to reduce overfitting, which occurs when the model memorizes the training data rather than learning the underlying patterns in the data. By introducing variations in the training data through random rotations, we can help the model learn to generalize better to new, unseen data.

4. Normalize:
Another important data transformation that we used in our project is transforms.Normalize(). This transformation standardizes the input data by subtracting the mean and dividing it by the standard deviation of the dataset. Specifically, we used the following normalization values for the CIFAR-10 dataset: mean = [0.4914, 0.4822, 0.4465], and standard deviation = [0.2023, 0.1994, 0.2010].

The above-mentioned data augmentation techniques are common across all models.

Methodology

ModifiedResNet builds upon the ResNet architecture by making several modifications to improve its performance. The architecture consists of several layers of convolutional neural networks, with each layer having a number of blocks. The blocks themselves are made up of convolutional layers followed by batch normalization layers and a residual connection. The residual connection allows the network to learn the difference between the output of a layer and its input, which helps to reduce the vanishing gradient problem and allows the network to be trained much deeper than traditional networks.

The first layer of the model is a convolutional layer with 64 filters and a kernel size of 3x3. This is followed by a batch normalization layer and a ReLU activation function. The output of this layer is then passed through a series of three layers, each containing several blocks of convolutional layers with batch normalization and ReLU activation functions, and a residual connection. The number of blocks in each layer is specified by the num_blocks parameter passed to the constructor.

The first layer contains 2 blocks, the second layer contains 2 blocks with a stride of 2, and the third layer contains 3 blocks with a stride of 2 [model 3]. The stride of 2 reduces the spatial resolution of the feature maps, which allows the network to learn higher-level features that are more invariant to small changes in the input image. The dropout layer is added after the second layer to prevent overfitting.

After the final layer of blocks, the output is passed through an average pooling layer with a kernel size of 8x8, which reduces the size of the feature maps to 1x1. Finally, the output is flattened and passed through a fully connected layer with 10 output nodes, corresponding to the 10 classes in the CIFAR-10 dataset.

We have experimented with multiple models, altering various configurations such as the number of stages, the optimizer used, and the number of blocks employed. Each model was trained on 60 epochs with a batch size of 32 for both train and test data. Convolutional channel of 64, 128, 256 is used in all the model. We have provided a visual representation of the training accuracy and loss values for each model in a graph that is included in this report.

Model 1:
The modified ResNet model was configured with three layers, consisting of 2, 2, and 3 blocks respectively in each stage, trained using stochastic gradient descent (SGD) as an optimizer, with a learning rate (Lr) of 0.005, weight decay of 5e-4, and dropout of 0.7, achieving a training accuracy of 94.5% and a test accuracy of 90.10%. The test accuracy is relatively lower than training accuracy, which suggests overfitting to some extent. Trainable parameters were 3,958,346. Figure 2 displays both the loss and accuracy graph.

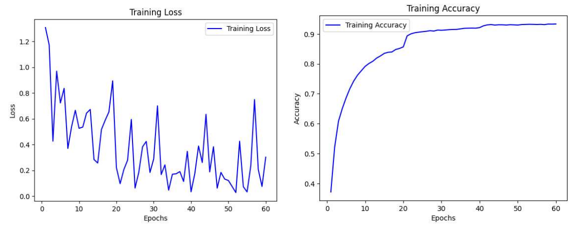

Fig 2.1) Training Loss Vs Epoch for Model 1
2.2) Training Accuracy Vs Epoch for Model 1

Model 2:
The modified ResNet model was configured with three layers, composed of 2, 2, and 3 blocks respectively in each stage, trained using the ADAM optimizer, with a learning rate (Lr) of 0.01 and no weight decay. The model achieved a training accuracy of 89.85% and a test accuracy of 87.76%, with a dropout rate of 0.5. Used ADAM optimizer, which can result in faster convergence compared to SGD. But, the dropout rate of 0.5 might be too low for this model, leading to overfitting. Trainable parameters were 3,958,346.

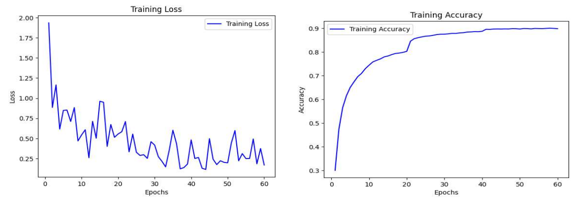

Fig 3.1) Training Loss Vs Epoch for Model 2
3.2) Training Accuracy Vs Epoch for Model 2

Model 3:
The modified ResNet model consisted of three layers, each stage containing 2, 2, and 2 blocks respectively. The model was trained using stochastic gradient descent (SGD) as an optimizer, with a learning rate (Lr) of 0.005 and a weight decay of 5e-4. The model achieved a training accuracy of 93.7% and a test accuracy of 90.21%, with a dropout rate of 0.5. Trainable parameters were 2,777,674.

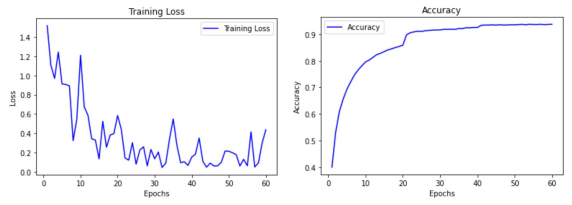

Fig 4.1) Training Loss Vs Epoch for Model 3
4.2) Training Accuracy Vs Epoch for Model 3

Model 4:
This modified ResNet model had four layers, with 2, 2, 2, and 1 blocks in each stage, respectively. The model was trained using stochastic gradient descent (SGD) as an optimizer, with a learning rate (Lr) of 0.005 and a weight decay of 5e-4. The model achieved a decent training accuracy of 94.45% and a test accuracy of 90.58% the model was trained with a dropout rate of 0.5. Trainable parameters were 4,024,394.

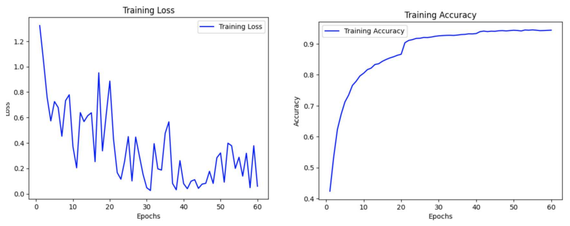

Fig 5.1) Training Loss Vs Epoch for Model 4
5.2) Training Accuracy Vs Epoch for Model 4

Result

Model 4 appears to be a viable option for image classification tasks based on its test accuracy of 90.58%, and its number of trainable parameters is 4,024,394. Additionally, the use of SGD optimizer has shown better performance over ADAM optimizer in this particular model, as suggested by the paper (Smith et al.).

The architecture of Model 4:

The convolutional layer 1 has 3 input channels and 64 output channels, with a kernel size of 3 and a stride of 1. The normalization layer performs batch normalization on the output of the convolutional layer 1.

The layer 1 consists of 2 BasicBlock modules, each with 64 input channels and 64 output channels. The output of layer 1 is then passed through layer 2, which consists of 2 BasicBlock modules. The same pattern continues for layer 3 and layer4 each with two and one BasicBlock respectively. The out_channels of the final convolutional layer in layer4 are set to 256, and the output of the avg_pool layer will also have 256 channels with kernel size of 4.

The next layer performs global average pooling on the output of layer4. The Dropout layer [probability = 0.5] randomly drops out some of the values in this tensor to prevent overfitting. The linear layer takes the output of the Dropout layer and maps it to the number of classes (10).

Furthermore, the results suggest that Model 3, which has three layers with each stage containing 2, 2, and 2 blocks, respectively, performs better than Model 1 and Model 2. This implies that the architecture of the ResNet model plays an important role in achieving high accuracy on image classification tasks.

Overall, the results are promising and provide insights into the importance of selecting an appropriate optimizer and architecture for image classification tasks.

Through experimentation with different optimization techniques, regularizers, and choice of the learning rate, batch size, and epochs, we were able to achieve high accuracy on the CIFAR-10 test set within the constraints of the project. Lessons learned from this design process include the importance of balancing the trade-off between model complexity and generalization ability, and the effectiveness of data augmentation in improving model performance among others.

References

ArXiv Paper
Smith, S.L., Kindermans, P.J., Ying, C. and Le, Q.V., 2017. Don't decay the learning rate, increase the batch size. arXiv preprint arXiv:1711.00489.

Technical paper
He, K., Zhang, X., Ren, S. and Sun, J., 2016. Deep residual learning for image recognition. In Proceedings of the IEEE conference on computer vision and pattern recognition (pp. 770-778).

Thesis
Liang, R.X., 2019. Experiments on the generalization and learning dynamics of deep neural networks (Doctoral dissertation, Massachusetts Institute of Technology).

Github repository
https://github.com/kuangliu/pytorch-cifar

Technical paper
Jafar, A. and Myungho, L., 2020, August. Hyperparameter optimization for deep residual learning in image classification. In 2020 IEEE International Conference on Autonomic Computing and Self-Organizing Systems Companion (ACSOS-C) (pp. 24-29). IEEE.

Website or online resource
https://paperswithcode.com/method/residual-block