CIFAR-10 Image Classification with ResNet Architecture

Team Members:
Neha Patil [redacted]
Ruchi Jha [redacted]
Satvik Upadhyay [redacted]

Abstract

We are using Residual Networks to train our image classification model using CIFAR-10 dataset. The reason why this architecture makes sense over other architectures like CNN (Convoluted Neural Network) is that we can fix the vanishing gradients problem. We used various techniques like Data Augmentation, Learning Scheduling, and Optimizers to get an accuracy of 81.6%.
Link to the code: https://github.com/patilneha08/CIFAR-10-Image-Classification-with-ResNet-Architecture-/tree/main

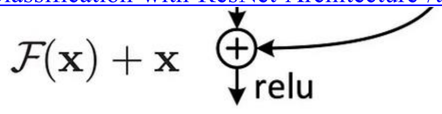


Figure 1: Residual Block Structure of ResNet Architecture [1]

Simply put, Residual block is:

H(x) = 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. H(x) is learning the residuals.

The residual layers are Convolutional layers followed by normalization, ReLu, and finally average pooling. We started by augmenting the data first to better clean it. Using various transforms like cropping, normalizing, rotating, etc., we were able to augment the data for our model. We then trained this model for 75 epochs as we have kept a low learning rate for accuracy and Gradient descent for coming to the final values. We are also using momentum to alleviate this process.

We are using 4,697,162 model parameters which is under the 5M model parameters mark. This provides us with the right model complexity as we were previously overfitting with higher parameters and underfitting with 300K parameters.

Finally, we are also using a learning scheduler to add a weight decay as a form of regularization. This is done at various milestones (different epoch values). We are also using the CrossEntropyLoss function as it performs better for classification related tasks.

The hyperparameters that we tweaked (and their final values) to get the best result:

Ci, the number of channels in the ith layer = 64, 128, 256

Fi, the filter size in the ith layer = 3, 3, 3

Ki, the kernel size in the ith skip connection = 3, 3, 1

P, the pool size in the average pool layer = 1

Lr, learning rate = 0.1

M, milestones = [30, 40, 60, 80]

Epochs = 75

Weight Decay = 1e-4

Methodology

We started with loading the CIFAR-10 data and then creating training and testing data. We then augmented this data for better inputs. Finally, using various other techniques and tweaking hyperparameters, we were able to improve the accuracy of this model. We will discuss more about these techniques and hyperparameters.

Optimizers: We are using standard gradient descent with a learning rate of 0.1 and a weight decay of 1e-4 at various milestones. This is great for adjusting the weights and adding regularization at various points. We are using Learning

Scheduler (MultiStepLr) for this purpose. This helps in converging faster. We tried using Cosine Annealing but that was not as effective as MultiStepLr.

Learning rate, epochs, weight decay: We tweaked these hyperparameters to get better accuracy and convergence. Learning rate specifies the % by which a weight will be subtracted (0.1 = 10%). Epochs specify the number of times this action will be performed, and Weight Decay specifies the weight reduction after reaching various epoch values. Our momentum was 0.9 and our batch sizes were 256 for training and testing.

Data Augmentation: This process involved manipulating the images to "get more images". This is because the model can be overly familiar with certain parts of the image. Augmenting it provides with us more inputs for the model to learn from. We did so by cropping the image to size 32 and rotating it. We also normalized the color palette, adding jitters, etc. After this manipulation, this new data was fed into the model.

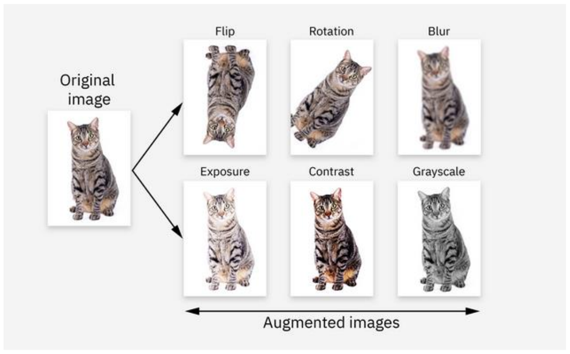


Figure 2: Examples of Data Augmentation Techniques Applied to Image Data [2]

Number of layers: We used 3 layers of Residual Block for this model. These residual blocks are basically convolutional layers. But they have skipped connections which helps with the vanishing gradients problem in traditional CNN. We had to opt for 3 layers as anything more would balloon our total parameters past 12M.

Given that our model did a decent job in image classification, we noticed while training that our model would get to about 90% accuracy in about 30 epochs and then our testing accuracy would often rise and fall but our training accuracy kept getting better. Our final training accuracy would be ~99% but testing would be ~93%.

Epoch	Train Loss	Train Acc	Val Loss	Val Acc
1/75	1.8399	32.06%	1.5187	43.08%
5/75	0.9068	67.88%	0.9020	69.79%
10/75	0.5490	80.92%	0.5742	80.95%
15/75	0.3930	86.27%	0.4636	84.28%
20/75	0.3082	89.26%	0.6083	81.78%
21/75	0.2981	89.63%	0.3447	88.25%
25/75	0.2571	90.97%	0.3258	89.20%
29/75	0.2181	92.46%	0.5916	83.79%
30/75	0.2129	92.37%	0.3648	88.18%
31/75	0.1334	95.38%	0.2262	92.89%

Interesting Observation

10 interesting trends before the 32nd epoch. We were quickly about to achieve a high training and testing accuracy but the final accuracy at epoch 75 was ~93% for testing and ~99% for training.

Our validation accuracy over epochs. It starts plateaues after 30.

Model Architecture

We implemented a custom ResNet architecture with the following components:

- *Initial Layer*: 3×3 convolution with 64 output channels
- *Layer 1*: 4 residual blocks with 64 channels (stride=1)
- *Layer 2*: 4 residual blocks with 128 channels (stride=2)
- *Layer 3*: 3 residual blocks with 256 channels (stride=2)
- *Output Layer*: Adaptive average pooling followed by a fully connected layer

Each residual block consists of two 3×3 convolutional layers with batch normalization and ReLU activation, along with a skip connection to enable better gradient flow during backpropagation.

Design Considerations and Trade-offs

1. *Depth vs. Width*: We opted for a moderately deep network (11 residual blocks) with a gradual increase in channel width. This balanced computational efficiency with representational capacity.

2. *Skip Connections*: The inclusion of skip connections in each residual block was crucial for training stability and performance. Skip connections

allow gradients to flow directly through the network, mitigating the vanishing gradient problem common in deeper networks.

3. *Batch Normalization*: We included batch normalization after each convolutional layer to stabilize training, reduce internal covariate shift, and enable higher learning rates.

Training Progression

The training process showed three distinct phases:

1. *Initial Learning Phase (Epochs 1-30)*: Rapid improvement from 36.04% to 89.16% validation accuracy with the initial learning rate.

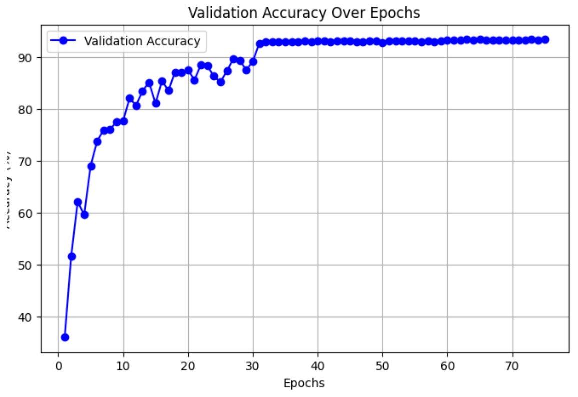
2. *Refinement Phase (Epochs 31-60)*: After the first learning rate reduction, accuracy jumped to 92.65% and gradually improved to 93.24%, showing the effectiveness of the learning rate schedule.

3. *Fine-tuning Phase (Epochs 61-75)*: Slight improvements in accuracy from 93.31% to the final 93.41%.

Conclusions and Future Work

This project successfully implemented a custom ResNet architecture for CIFAR-10 classification with competitive performance. The model's strong accuracy demonstrates the effectiveness of residual connections and careful training strategies.

Future improvements could include:
- Exploring additional architectural variations like SE-ResNet (Squeeze-and-Excitation) or ResNeXt
- Implementing more advanced regularization techniques like Cutout or MixUp
- Exploring ensemble methods to further improve accuracy
- Knowledge distillation to create smaller, more efficient models



References

For this model to work, we took some help from LLMs like ChatGPT for debugging. But the underlying model was all our work. We also used LLAMA for making graphs.

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

Citation

1. Residual block
2. Data Augmentation