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

Kasam Sri Harsha Vignesh, Indukuri Sai Appalraju, Venkata Sai Sarath Chandra Mendu
sk9840, si2220, vm2251

Code link:
https://github.com/harsha-kasam/Deep-Learning-mini-project

Abstract

This paper will examine why we need residual blocks and how to use them for image classification. First, we will analyze a few pre-existing architectures and use this knowledge to build our residual model. We will mainly compare the variations of ResNet18 architectures by using different optimizers and look at how we have landed on this architectural design. We tried to maximize the accuracy with a constraint of 5 million parameters.

Overview

Deep neural networks are fascinating, and they produce magic as we try to predict something, maybe with images or texts. But, when we build deeper networks, we encounter a degradation problem. The increase in training error is a problem as deeper layers are constructed. One another problem of building deeper networks includes the vanishing gradient descent.

While backpropagating, the derivatives of each layer are multiplied down the network. When we use a lot of deeper layers and hidden layers like sigmoid, we could have derivatives scaled down below 0.25 for each layer. So when n number of layers derivatives are multiplied, the gradient decreases exponentially as we propagate down to the initial layers.

To solve these problems, we come across the ResNet paper. These residual networks are stacked together to allow us to build deep networks without degradation or vanishing gradient descent. In Residual networks, we use skipped connections in a building block called a residual block. The idea is to connect the input of a layer directly to the output of a layer after skipping a few connections. These skipped connections will allow the gradient descent to take a shorter path to layers in the beginning and make the deep networks work efficiently. These blocks will also help us skip a few connections which aren't contributing much. Below we can look at a single residual block that makes a residual network.

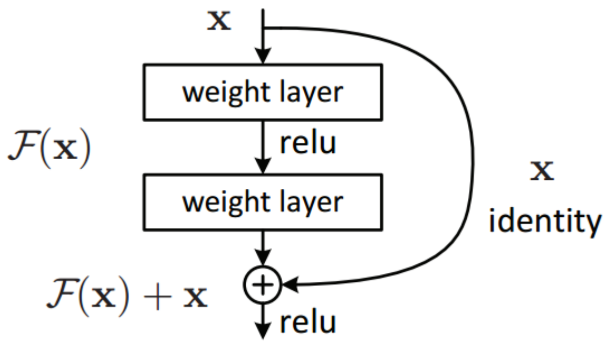

x
weight layer
relu
weight layer
$\mathcal{F}(\mathbf{x})$
x
identity
$\mathcal{F}(\mathbf{x}) + \mathbf{x}$
relu

fig. 1

We designed two choices of modified ResNet architecture with varying residual blocks and experimented by tuning various hyperparameters. All the models saturated at around 70 epochs, with test accuracy ranging from 89% to 93%.

Data Augmentation

Deep neural networks are easily prone to overfitting; hence, it is good to have errors in the data. Data augmentation helps increase training data size and regularize the network by introducing errors in the data. It does this by creating variations of the existing data in the train set. In general, data augmentation in images entails rotating images or picking a specific area of an image at random or otherwise. Here, we go over a couple of transformations. Let's first flip an image horizontally with a given probability p.

Random Crop with Padding

We add padding to the image by adding pixels with a given width, then crop the padded image to the appropriate size. For instance, if we add 4 pixels to each side of a 32*32 image, we get a 40*40 picture. We then choose a 32*32 random crop from it.

Normalize

First, convert the image to a tensor using ToTensor() and then normalize the tensor image with mean and standard deviation.

About the dataset:

The CIFAR-10 dataset consists of 60000x32x32 color images with 6000 images in each of the ten classes. Train data consists of 50000 photos, and test data consists of 10000 images. Five training batches and one test batch, each with 10,000 images, make up the dataset. Exact 1000 randomly chosen images from each class make up the test batch. The remaining images are distributed across the training batches in random order. However, specific training batches can have a disproportionate number of pictures from a particular class. The training batches consist of exactly 5000 images from each category combined.

Architecture

For selecting our architecture, we have analyzed Resnet18, Resnext, and GoogLeNet. These architectures are pretty deep for a standard CIFAR 10 dataset. In Resnext, we have bottleneck blocks where we increase the output channels in each convolution layer of each residential block. This intensified architecture is unnecessary to extract the features of a standard dataset with just ten classes and low-definition images. If we use bottleneck blocks with deep layers, we will end up with extensive feature channels where a few might be redundant. Whereas GoogLeNet takes a long time to train one epoch, it's hard to fine-tune its optimizer parameters. Hence it's best to use Resnet18, and let us discuss it in detail.

Resnet18 is one of the main architectures developed using residual blocks. A research team from Microsoft first introduced residual blocks. Resnet18 has a pre-processing convolution layer, eight residual blocks, and a linear layer. Let's first discuss why we need these layers and how to manipulate them for our constraints and requirements.

The preprocessing convolution layer extracts the features from 3 channels and gives 64 channels for the subsequent layers. In the next part, the eight residual blocks are divided into four layers, with 2 in each. The first layer is the identity layer which improves the extracted features and can be viewed in the following figure.

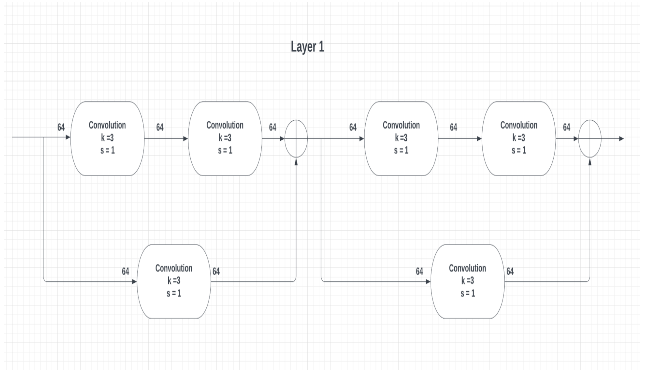


fig. 2

We extract more features from the preceding layer's outputs in the following three layers. We use stride 2 in a single convolution layer of each residual block. A stride of two allows each block to reduce the image size by 2x2. The final output of the residual layer would be 512x4x4, which will be reduced to 512 features for a single image using 2d Average pooling with a kernel size of 4. These 512 features are fed to a linear layer mapping to 10 classes.

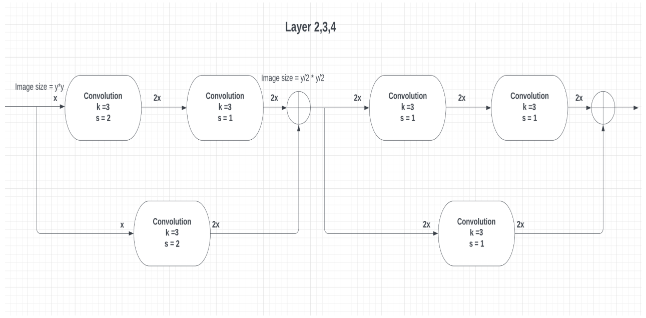


fig. 3

Methodology

Instead of extracting 64 features in the preprocessing convolution layer, we are extracting 32 features. This change allows the layer to train better since mapping for 32 would be easier.

The first layer of the Residual network has two identity layers, as shown in fig.2, which make the features more prominent. Still, it's unnecessary since we are extracting 32 features instead of 64 in the original paper. Hence we reduced the number of residual blocks in this layer to one.

We have used two variations in the architectural design. In the first variation, we added an extra residual block in layer 4 to make the features more prominent before feeding it to a fully connected network. We added additional residual blocks in the 3rd and 4th layers in the second variation. Increasing identity layers, in the end, makes the features more prominent, and if the input features to this layer are already good enough, these blocks will act as identity layers. We will talk about the results of these variations in the next section.

Now, the fully connected layer will have the input of 256 features in this variation. We changed the fully connected layer from a single linear layer to 2 layers with input sizes 256 and 512 with a ReLu activation function between them. This extra layer introduces non-linearity between features and classes and allows the model better to understand the relation between these features and categories. The first model has parameters of 4 million, and the second variation has 4.3 million.

Model 1:

As explained above, our model will consist of a preprocessing convolution layer with 32 output channels, one residual network, and one fully connected layer. The residual layer will be the first variation, as mentioned above. In our first two runs, we compared learning rates of 0.1 and 0.05 with a momentum of 0.9 and L2 regularization weight decay of 0.0001, which gave us a training accuracy of 94.72%, 97.67% and a test accuracy of 90.89% and 91.76%, respectively. As we can see, lr =0.05 gave us better results. Hence following this path and a few more runs helped us reach our optimum learning rate parameters of 0.01 and weight decay of 0.00001. These were used to compare two architectures. We have obtained a training and testing accuracy of 99.55% and 91.55%, respectively, for these parameters.

Model 2:

In this model, we have changed the residual layers in the 3rd and 4th layers from 2 to 3 to make the features more prominent. If the features are already good enough, these extra residual blocks will act as identity since we add blocks with the same in_channels and out_channels. This model's training and testing accuracies are 99.5% and 91.5%, respectively, at the 87th epoch.

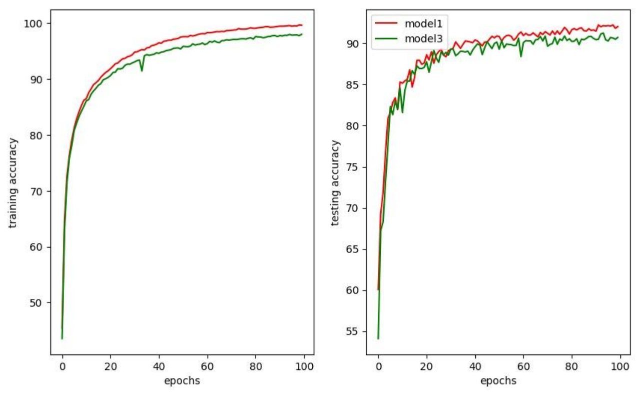
Model 3:

Here, we have used Adam optimizer in the first variation of the architecture. We first started with a higher learning rate of 0.1. The learnings were exponential, with a rapid increase in accuracy. Still, it flattened out very soon, with a test accuracy of just 68% due to the overshooting of the gradients. So we tried with a lower learning rate of 0.001.

Results

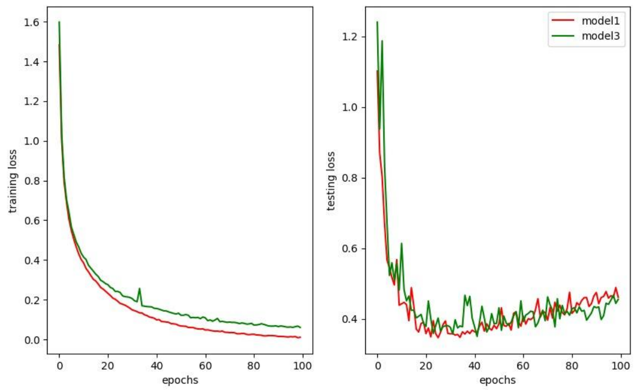
The results of model 2 and model 3 are similar, suggesting that adding an extra residual block in the 3rd layer didn't help much. We can conclude that the additional block acted as an identity layer since the features were well-defined for that layer, resulting in similar accuracies.

We changed the optimizers in models 1 and 3 and tuned them to obtain good results. Both the optimizers performed similarly; hence, let's look at their performance wrt epochs. Here is the graph of their training accuracy and testing accuracy over 100 epochs for model 1 and model 3.



As we can see here clearly, SGD outperforms Adam, which is common in the case of image classification.

Let us compare the values of training loss and testing loss over 100 epochs.



Since Adam adjusts its learning rate and as well as momentum, we can see that the testing loss of SGD started crossing the loss of Adam.

References

ArXiv Paper
He K.; Zhang X.; Ren S.; Sun J. 2015. Deep Residual Learning for Image Recognition. arXiv:1512.03385.

Website or online resource
Machine Learning Mastery. 2017. Gentle Introduction to the Adam Optimization Algorithm for Deep Learning. https://machinelearningmastery.com/adam-optimization-algorithm-for-deep-learning/.

Website or online resource
Stackoverflow. 2016. Is it a good learning rate for Adam method. https://stackoverflow.com/questions/42966393/is-it-good-learning-rate-for-adam-method

Website or online resource
Github. 2017. Pytorch-cifar. https://github.com/kuangliu/pytorch-cifar.

Website or online resource
Geeks for geeks. 2022. Residual Networks (ResNet) – Deep Learning.https://www.geeksforgeeks.org/residual-networks-resnet-deep-learning/.

Website or online resource

Analytics Vidhya. 2021. Understanding ResNet and analyzing various models on the CIFAR-10 dataset. https://www.analyticsvidhya.com/blog/2021/06/understanding-resnet-and-analyzing-various-models-on-the-cifar-10-dataset/.