Designing CNN Architectures for CIFAR-10 Image Data Classification

Raktim Roychoudhury(rr4081)

Abstract

In our project, we are tasked with designing a modified residual network architecture to classify the CIFAR-10 image dataset. For our model, we decided to adopt the residual block module from MobileNetV2 architecture (Sandler et al. 2019) based on an inverted residual structure where the shortcut connections are between the thin bottleneck layers. We propose a model which includes 17 of these residual blocks, resulting in a final network architecture with 2.6 million parameters. Our empirical results show that this architecture is very robust and achieves a 94.8% accuracy on the CIFAR-10 test set, without any regularization. Our implementation is available in the following GitHub repository.

Introduction

Deep convolutional neural networks (Deep CNNs) have led to a series of breakthroughs, particularly for image classification tasks in the last decade. AlexNet (Krizhevsky, Sutskever, and Hinton 2012) shot to fame in 2012 when it won the ImageNet Large Scale Visual Recognition Challenge (ILSVRC). This early iteration of a deep CNN architecture achieved a top-5 error of 15.3%, more than 10.8 percentage points lower than that of the runner up.

(Krizhevsky, Sutskever, and Hinton 2012) conclude that the depth of the AlexNet model architecture is essential for its high performance. This observation and the success of AlexNet led some to suggest that learning better networks could be as simple as stacking more layers, using modern GPUs which could handle the high computational costs of training very deep networks. However, (Bengio, Simard, and Frasconi 1994) and (Glorot and Bengio 2010) show that a significant obstacle to training deep networks is the problem of vanishing/exploding gradients which hamper convergence from the beginning. This problem has been largely addressed in recent years, by normalized initialization of layer weights and intermediate normalization layers which enable networks with tens of layers to start converging for stochastic gradient descent (SGD) with back-propagation (Lecun et al. 1989).(He et al. 2015) highlight that when deeper networks are able to start converging, accuracy gets saturated for increasing network depth and then starts degrading rapidly. This observation is not attributed to overfitting but to optimization issues with the underlying mapping that the network tries to fit.

(He et al. 2015) propose a new framework called Residual Networks or ResNets to tackle the issues highlighted previously. Instead of network layers trying to fit a desired underlying mapping, ResNets explicitly let the layers fit a residual mapping. Formally, denoting the desired underlying mapping as $\mathcal{H}(x)$ with respect to the layer inputs x, the stacked nonlinear layers fit another mapping of $\mathcal{F}(x) := \mathcal{H}(x) - x$. The original mapping is thus recast into $\mathcal{F}(x) + x$. This relatively simple reformulation is largely responsible for resolving the degradation issues frequent in very deep networks and is considered a major breakthrough in the history of deep learning. Architectures featuring residual blocks are now a part of almost all large deep learning models, featuring prominently across all GPT (Radford et al. 2018) architectures powering the now-famous ChatGPT .

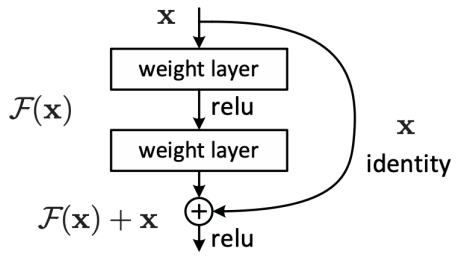

Figure 1: Residual learning : a building block

Methodology

Given the task of designing a CNN based neural network for image recognition with a cap on the maximum number of parameters, we decided to focus on models specifically designed for resource constrained environments. Both MobileNetV1 (Howard et al. 2017) and MobileNetV2 (Sandler et al. 2019) were specifically designed to be efficient for mobile and embedded applications which make them ideal for our use case. We adopt the residual block module from the MobileNetV2 model and incorporate it into our proposed architecture. These residual blocks include Depthwise Separable Convolutions, linear bottlenecks layers and inverted residuals which we discuss in the following section.

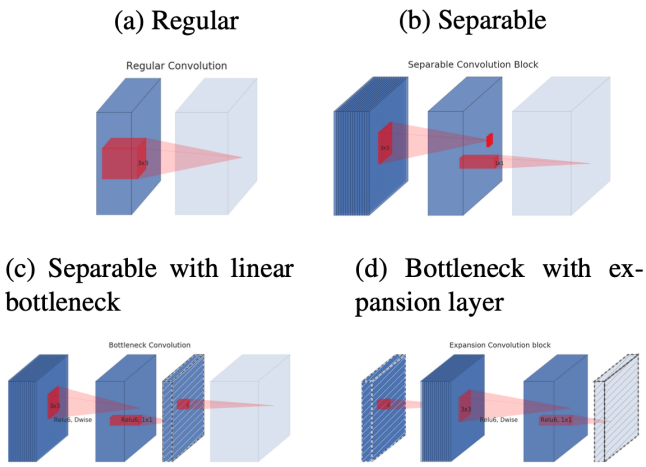

Figure 2: Evolution of separable convolution blocks

Depthwise Separable Convolutions

Depthwise Separable Convolutions are a key building block for many efficient neural network architectures (Howard et al. 2017; Chollet 2017). The basic idea is to replace a full convolutional operation with a factorized version that splits convolution into two separate layers. The first layer is called a depthwise convolution, which performs lightweight filtering by applying a single convolutional filter per input channel. The second layer is a 1×1 convolution, called a pointwise convolution which is responsible for building new features, by computing linear combinations of the input channels. Empirically, depthwise separable convolution work almost as well as regular convolutions but is much more computationally efficient.

Linear Bottlenecks

(Sandler et al. 2019) hypothesize that for an input set of real images, the layer activation forms a *manifold of interest* and that this manifold is low-dimensional. If it is indeed low-dimensional, we can then use linear bottleneck layers to capture the information from the manifold. (Sandler et al. 2019) show support for their assumption empirically and also conclude that using linear layers is crucial as it prevents non-linearities from destroying too much information.

Inverted Residuals

The bottleneck blocks appear similar to residual blocks where each block contains an input followed by several bottlenecks then followed by expansion. However, based on the assumption that the bottleneck layers actually contain all the necessary information, the shortcut connections are applied directly between the linear bottleneck layers. The shortcut connections improve the ability of a gradient to propagate across multiple layers. However, (Sandler et al. 2019) highlight that the inverted residual design is more memory efficient and works slightly better in their experiments.

The bottleneck residual block forms the fundamental basis of our model architecture which we describe in detail in the next section.

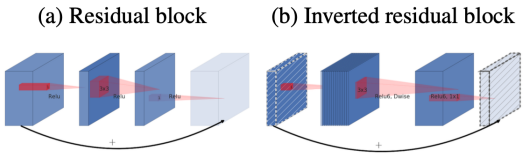

Figure 3: Residual v/s inverted residual block. The thickness of each block indicates the relative number of channels. Classical residuals connect the layers with high number of channels whereas the inverted residuals connect the bottlenecks

Model Architecture

As discussed in the previous section, the basic building block of our proposed network is a bottleneck depth separable convolution with residual connections (Fig.4). Except for the point wise convolutional layers, we use the standard 3×3 kernel for the CNNs and utilize batch normalization layers to avoid exploding/vanishing gradients during training.

We describe our proposed model architecture in Table 1. The model consists of an initial fully convolution layer with 32 channels followed by 17 residual bottleneck layers. The output of the final residual block is fed into another fully convolution layer with 1280 channels. We then apply a 7×7 average pooling layer before the final dense linear layer. The output of the final layer is a $k = 10$ dimensional vector corresponding to the class probabilities of the CIFAR-10 dataset.

With the exception of the first layer, we use a constant expansion factor throughout the network. The expansion factor works as follows, for a bottleneck layer that takes a 64-channel input tensor and produces a tensor with 128-channels, the intermediate expansion layer is then $64 \times expansionfactor$ channels. The *num block* parameter in Table 1 represents the number of times the same block is repeated in the network sequentially. All layers in each block contain the same number of channels. Moreover, the first convolutional layer of each block has a stride as shown in Table 1, with all subsequent layers using stride $= 1$.

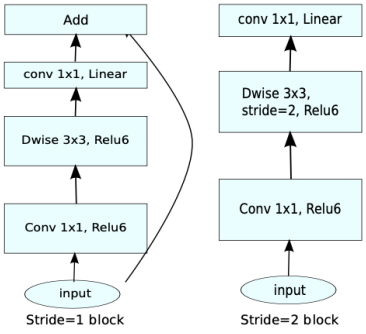

Figure 4: Bottleneck depth separable convolution with residuals

Operator	expansion	channels	num blocks	stride
conv2d	-	32	1	1
bottleneck	1	16	1	1
bottleneck	7	24	2	1
bottleneck	7	32	3	2
bottleneck	7	64	4	2
bottleneck	7	96	3	1
bottleneck	7	160	3	2
bottleneck	7	320	1	1
conv2d	-	1280	1	1
avgpool	-	-	1	-
linear	-	k	-	-

Table 1: Proposed model architecture and configuration using bottleneck blocks (2,598,074 total model parameters)

Experiments and Results

We test this architecture on the CIFAR-10 image classification dataset. We first perform data augmentation transformations on the training data which include random cropping, random horizontal flips and normalization of the pixel values. For our experiments, we use a training batch size of 128 and test batch size of 100. We choose a different batch size for the test set to ensure all batches are of equal size.

We then experiment with the expansion factor parameter of our architecture. The model is trained for 200 epochs in each of our experiments. Fig. 5 shows our results for expansion factors 5, 6 and 7. We also use a SGD optimizer with values ranging from 0.001 to 0.01 and find that the learning rate of 0.01 works well, with maximum test accuracy of 89%. However, increasing the learning rate further increases accuracy, and we find a learning rate = 0.045 with an exponentially decaying scheduler to work best for our network architecture. While we observe similar results for the expansion factors tested, the test accuracy for expansion factor = 7 converges faster and we obtain a highest test set accuracy of 94.8%.

We also tested different strides for the CNN layers and found that a stride = 1 outperforms stride = 2 for the second bottleneck layer. An explanation for this result could be due to the lower resolution of the CIFAR-10 dataset as compared to a higher resolution dataset like ImageNet. The higher stride therefore results in a loss of important information on the CIFAR-10 dataset.

Apart from the architecture, we also test different optimizers for our network, namely SGD and ADAM(Kingma and Ba 2017) with learning rate schedulers. Although ADAM converges much faster as seen in Fig 7, SGD achieves a higher test accuracy.

Based on our experiments, we highlight the final proposed architecture in Table 1 with an SGD optimizer with learning rate = 0.045 and an exponential scheduler. The total number of parameters in our proposed model are **2,598,074**. The highest test set performance of this architecture is 94.8%.

We would also like to mention that our implementation of the bottleneck residual block in PyTorch is based on the following GitHub repository(kuangliu 2021).

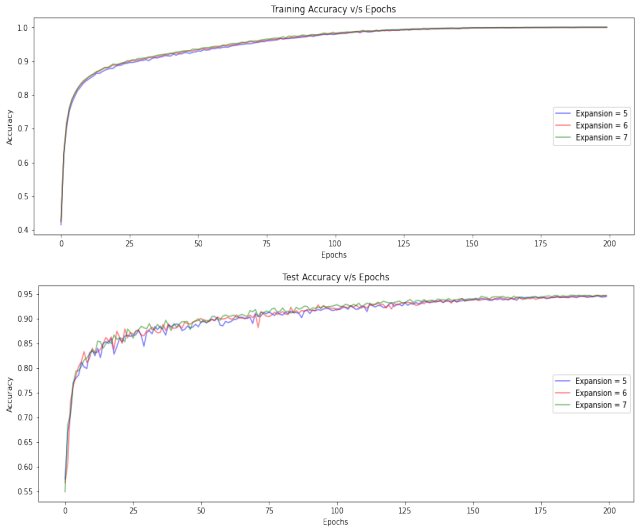

Figure 5: Accuracy v/s Epochs

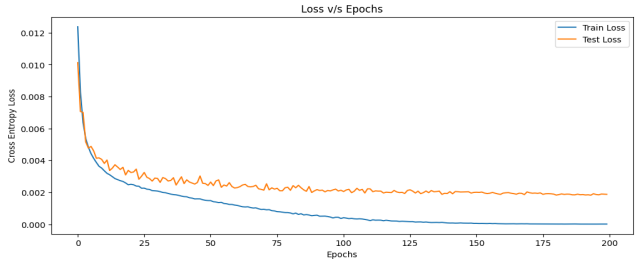

Figure 6: Cross-entropy loss v/s Epochs for expansion = 7

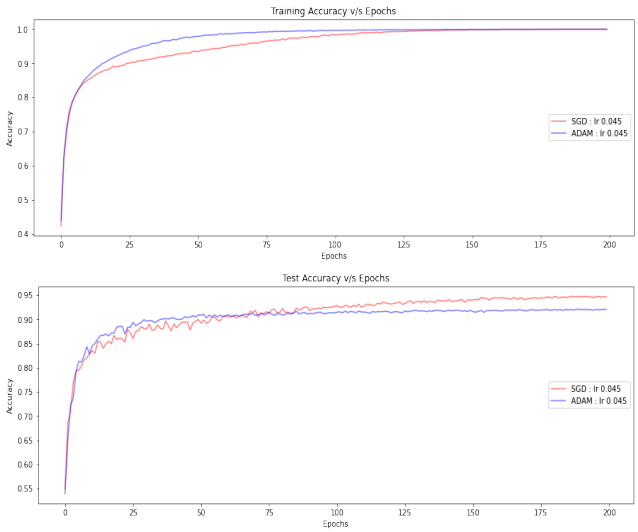

Figure 7: Accuracy v/s Epochs for SGD and ADAM optimizers

References

Bengio, Y.; Simard, P. Y.; and Frasconi, P. 1994. Learning long-term dependencies with gradient descent is difficult. *IEEE transactions on neural networks*, 5 2: 157–66.

Chollet, F. 2017. Xception: Deep Learning with Depthwise Separable Convolutions. In *2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 1800–1807.

Glorot, X.; and Bengio, Y. 2010. Understanding the difficulty of training deep feedforward neural networks. In *International Conference on Artificial Intelligence and Statistics*.

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

Howard, A. G.; Zhu, M.; Chen, B.; Kalenichenko, D.; Wang, W.; Weyand, T.; Andreetto, M.; and Adam, H. 2017. MobileNets: Efficient Convolutional Neural Networks for Mobile Vision Applications. arXiv:1704.04861.

Kingma, D. P.; and Ba, J. 2017. Adam: A Method for Stochastic Optimization. arXiv:1412.6980.

Krizhevsky, A.; Sutskever, I.; and Hinton, G. E. 2012. ImageNet Classification with Deep Convolutional Neural Networks. In Pereira, F.; Burges, C.; Bottou, L.; and Weinberger, K., eds., *Advances in Neural Information Processing Systems*, volume 25. Curran Associates, Inc.

kuangliu. 2021. Train CIFAR-10 with PyTorch. https://github.com/kuangliu/pytorch-cifar.git.

Lecun, Y.; Boser, B.; Denker, J.; Henderson, D.; Howard, R.; Hubbard, W.; and Jackel, L. 1989. Backpropagation applied to handwritten zip code recognition. *Neural Computation*, 1(4): 541–551.

Radford, A.; Narasimhan, K.; Salimans, T.; and Sutskever, I. 2018. Improving language understanding by generative pre-training.

Sandler, M.; Howard, A.; Zhu, M.; Zhmoginov, A.; and Chen, L.-C. 2019. MobileNetV2: Inverted Residuals and Linear Bottlenecks. arXiv:1801.04381.