Disclaimer: This paper is a work of fiction written from the perspective of a 2020 researcher traveling back in time to late 2012 to share some 2020 network design, training and implementation ideas; references to credit the actual inventors of the various ideas is provided at the end

Skip Those Layers, You Don't Need Them

Jaisurya Nanduri
UT Dallas Research
jxn180029@utdallas.edu
Dec 31, 2012

Abstract

State Of The Art Neural Networks have failed to improve results with increase in model complexity. We present a framework to build networks which scale efficiently and are substantially easier to train than those used so far. We allow the model to learn the identity function and thereby let the model filter out the layers where learning weights adds to the vanishing/exploding gradients and the degradation problem. We provide comprehensive evidence that unlike previous architectures, these modified residual networks gain accuracy with increased depth. On the CIFAR10 dataset, we evaluate Residual Networks with a depth of upto 50 layers, 8x deeper than AlexNet. These Residual Networks see a minimum improvement of 10% from the previous State Of The Art models , improving as much as 20% with increased depth. This is of utmost importance to several Image Recognition tasks as classification reaches 95% on previously unsolved problems, solely due to the innovations in the mechanism of building layers.

1. Introduction

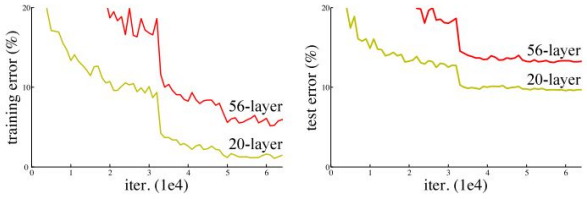
Image recognition is the task of using visual information to draw inferences. Several tasks such as Face Recognition , Object Classification, Augmented Reality make use of the state of the art technologies discovered in this field. A key driver contributing towards advancements in this field is the advancement in Deep Learning.

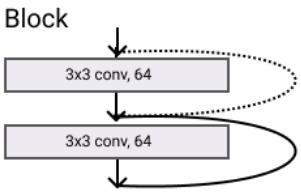
Deep Learning is the science of mimicking the human brain by making use of technology called Neural Networks. Deep Learning is the study of these Neural Networks which have a high layer count. Image Recognition has benefited greatly from the advancement of Deep Learning and as hardware powers increase and as we explore Neural Networks with ever increasing complexity and depth, we have been able to solve problems with an increasing amount of accuracy. One such problem which benefits greatly is the task of Image Classification. Image Classification has been attempted to be solved by Convolutional Neural Networks(convnets) ever since the 1990s. [1], but the space of convnets to solve the CIFAR-10 dataset has exploded since last year when most notably (Krizhevsky et al., 2012) delivered State-of-the-Art performance by achieving an error rate of 16.4 % beating the 2nd place which scored an error rate of 26.1%. Ever since, ConvNets have exploded in popularity and have become the industry standard in Image Classification , with further promise of improvement.There are several reasons behind the success and renewed interest and acceptance of convnets. Some of them being :: (i) abundant availability of data ; (ii) powerful GPU and hardware (iii) better model regularization strategies, such as Dropout (Hinton et al., 2012).

However, there is huge room for improvement for the ConvNet models existing in research. With added stacking of layers, the models increase in complexity but their accuracy seems to plateau after a certain point which hampers potential to solve bigger problems and to improve existing accuracy and error rates. This is deeply unsatisfying as despite the existing infrastructure and hardware to support upscaling of existing models, building deeper models does not improve any metric of success.

In this paper, we explore ways of solving the aforementioned problems existing in ConvNets so we can find ways of effectively adding layers while improving accuracy. We do this by introducing something called a "Skip-Connection" between layers of the convnets which allows the layers to independently learn the Identity function making the Neural Net flexible enough to approximate more complex functions. In the Design Section, we propose

(Fig1.) Training error (left) and test error (right) on CIFAR-10 with 20-layer and 56-layer "plain" networks. The deeper network has higher training error, and thus test error





(Fig2.) Residual Block Figure showing the skip connection.

the Skip-Connection Mechanism which is the central innovation which yields us the necessary results. In the Training Section, we discuss existing training methods and algorithms and their shortcomings as well as how we overcame their problems with a few modifications relevant to the task at hand. In Implementation, we finally discuss the hardware and the resources used and their efficiency in handling the task along with their limitations as well.

2 Related Work

2.1 Design

The problems in Image Classification have been marked by the introduction of two breakthroughs, namely, LeNet and AlexNet.

LeNet or rather lenet-5 specifically attempted to classify the MNIST dataset using an extremely simple architecture of 5 layers having 60k parameters. Despite being 20 years old, it incorporates several mechanisms used in this present day such as ConvLayers with stride and padding, MaxPooling and average pooling. It however lacked the usual final layer of Softmax to classify the input images. It could achieve an error rate below 1% on the MNIST which was state of the art at the time. However, LeNet is not scalable or is it capable of approximating tougher problems. Furthermore, research on LeNet was carried forward due to lack of availability in processing power required to build more complex models and was therefore largely forgotten.

In 2012 again, CNNs saw the spotlight with the introduction of AlexNet, a variation of the CNN developed by Alex Krizhevsky. Originally developed for the ImageNet challenge, it has since become a seminal technology , influencing a new era of research in image recognition. The classic AlexNet comprises 5 ConvLayers and 2 FC layers. It also introduced dropout and ReLu activation functions and was trained on the Nvidia GTX 580 GPUs.However, AlexNet too quickly encounters problems of scalability, often seeing an increase in training error with increase in the number of layers. This is because of a phenomenon called the Vanishing/Exploding Gradient problem which sees extreme values of the gradient weights dominate the more normal values.

We attempt to solve this problem by introducing a mechanism called a residual block which allows a CNN to learn lower level abstractions even at high or deep layers if needed , by letting certain layers mimic the identity function whenever necessary.

2.2 Training

The model was trained using mini batch SGD which is a compromise between fully stochastic GD and Batch Gradient Descent and often provides the best possible results. The training dataset is split into small batches of size . (We used a size of 128, although 256 works too often). These batches are used to calculate model error and the parameters are updated for the 128 input data samples at once. Pure SGD is computationally too expensive and models take significantly longer to train as they have to update parameters according to every input data. This frequent change can often lead to noisy gradients and can stifle convergence. Batch GD however may result in premature convergence and may suffer from local minima. It also makes it difficult for the model to train in parallel as the full input dataset needs to be available at once for the final weights to be updated. There are some downsides to mini-batch SGD too though. The configuration of batch-size is an additional hyper parameter which needs to be experimented with to come to a good point. Accumulating error information across several parallel mini batches is also computationally expensive.

2.3 Implementation

For the purposes of training this model, a 16GB NVIDIA GPU to perform the tensor based operations

such as convolutional mapping and element-wise multiplication. GPUs allow us to parallelize several processes thereby speeding up the training process. The hardware while adequate is not specialized for Deep Learning Tensor Process is built for graphics processing purposes. CPUs are completely inadequate to be used in any serious Deep Learning application due to their lack of tensor specific computational capabilities. Matrix multiplications are at the heart of Deep Learning computations in our model too which uses GPU to calculate the convolutional calculations and CPU for the rest of the processing and ETF pipelines. Hardware which could be built with the specific intent on processing Tensors in the near future could further accelerate research with faster training times than what we have currently.

3 Design

3.1 Intuition

Previously successful attempts at solving the CIFAR10 dataset have been the now industry standard architectures of LeNet and AlexNet. However, as introduced earlier, they quickly run into failure on attempting to solve larger problems for various explored reasons. This is not a problem unique to these architectures but rather something that impacts deep neural networks regardless of task and architecture.

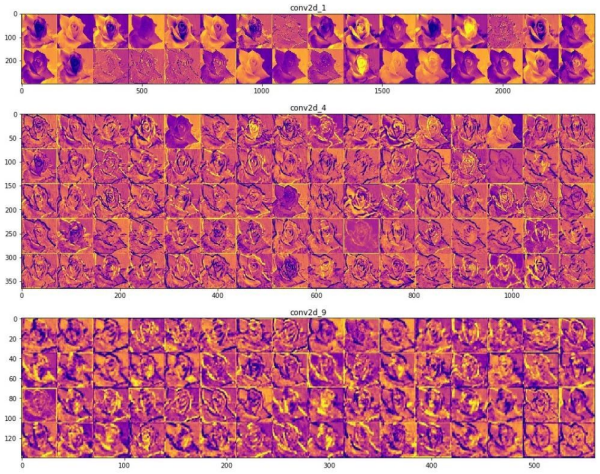
One key reason behind their failure is the problem of exploding/vanishing gradients which has a detrimental impact on weight convergence from the beginning. As we add and stack layers upon existing layers, while the expected performance is for the accuracy to increase, we find out that it decreases without interference. This phenomenon is seen independent of dataset and architecture. This is counter-intuitive as adding more complexity to a model should allow it to solve and fit more complex functions. But we notice that the opposite effect takes place[1]. This is because, using BackPropagation and Gradient Descent, weights replicated and carried forward multiple times combined with the activation functions makes the extremely smaller and larger activations stand out[1]. As we add more layers to the model, we increase the probability of extreme weights to stand out , leaving most weights either high or too low to approximate any function. This has been attempted to solve by using normalized initialization and intermediate normalization layers to reasonable success[1].

However, another key problem remains unsolved, which is the degradation problem. As depth increases, not only is it that, despite normalized weights, the accuracy saturates, but also that beyond a certain depth, it even begins to degrade rapidly. [cite]. To begin solving this problem of unintuitive degradation, we must first take a higher level look at what it is that layers do and what they represent.

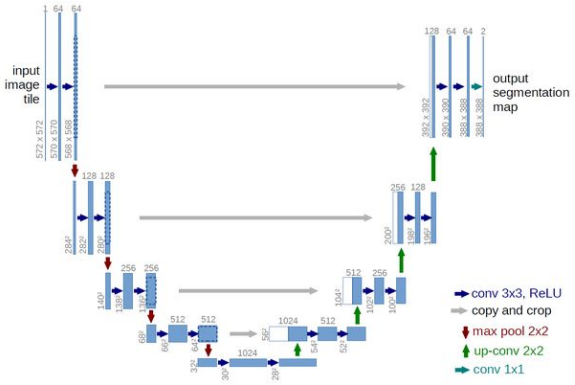
Each Layer in a Convolutional Neural Net can be thought of as a level of abstraction or features to be extracted from the input to that particular layer. In our dataset of CIFAR10, we are presented with 10 input classes of images , which are - 'airplane', 'automobile', 'bird', 'cat', 'deer', 'dog', 'frog', 'horse', 'ship', 'truck'.

Each of these categories has a precise hierarchy of visual features which make it distinct and unique from the other categories. What makes an image a ‘cat’ is different from what makes it a ‘dog’. If there is a function which can accurately separate one class of images from the others, it is an accurate model’s job to approximate this function. Furthermore, neural nets being universal function approximators, all that distinguishes a working neural network model from a poor one is the relevance of layers that comprises it, relative to the task.

The question then arises what makes a layer “relevant”. To answer, it is important to understand an exact layer’s functions. Lower level layers , the ones close to the input layer extract lower level abstractions and higher level layers extract higher level abstractions.
A lower level abstraction is a low complexity abstraction which is to say formally, simple curves and shapes. A model attempting to classify the CIFAR-10 dataset will have the simplest of curves after the first layer of convolution, such as straight lines, semi-arcs and other arcs. Having extracted these in the first layer, we move on to combining two or more of these simple curves to get slightly more complex abstractions such as a semi-circle which can be understood as a combination of a straight line and an arc. As we go deeper by layer, our features become increasingly more complex or abstract , approximating complex edges and shapes until finally we have kernel maps which represent the mathematical models of shapes such as “paws”, “eyes”, “beak”, “bonnets” etc. Given these high level features, it becomes trivial to categorize images into their most probable final classes.



(Fig 3.) Abstractions decrease in clarity and relevance with increasing depth.



(Fig 4.) Skip Connections allowing the network to bypass layers altogether.

In the case of existing models, while useful abstractions are extracted , upon adding multiple layers, it is possible that a lot of noisy features are extracted too which are not relevant to the task. A shape or edge which does not necessarily provide any meaningful information towards classifying images into the specific categories would only add computational cost and noise thereby doubly reducing accuracy while making the model more inefficient. Therefore, relevant features are those edges which contribute meaningfully to the classification task. For example, an edge resembling "fins" or some other fluid abstract combination of simpler edges, would only add noise to our model and therefore be irrelevant. Making an efficient model therefore is reduced to extracting only the most relevant of features and to making a refined hierarchy which accurately is able to extract the correct features.

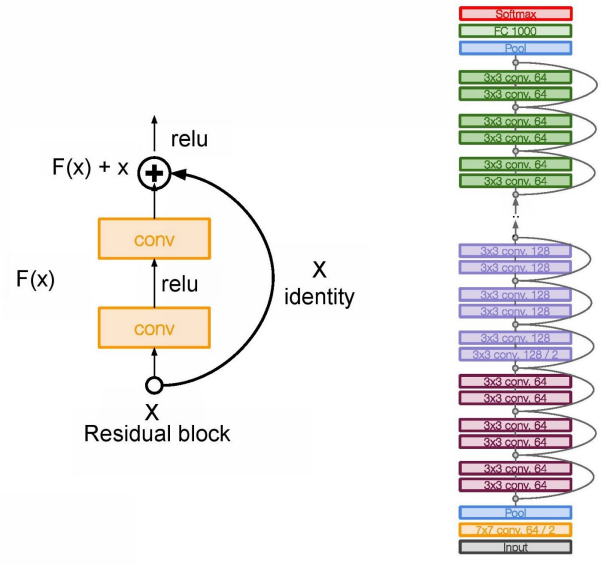
In the existing architecture however, there is no mechanism to prevent noisy features from being mined. This is a significant factor causing the degradation problem.Our experimentation solves the problem of filtering the most relevant features by using the "skip-connection".

The intuition behind the skip-connection mechanism behind the design is the need to allow previously used up "leftover" and unused lower level abstractions to be used in higher levels again if there is a need for them.The key logic is the "need" to unlearn certain layers and filters. This is the way to filter the noise. If we can let the model learn where it seems it necessary and allow it to not learn where it doesn't see the need to, we in effect allow the model to learn only what is necessary. For example, if there was some information that was captured in the initial layers and was required for reconstruction during the up-sampling done using the FCN layer, If we would not have used the skip architecture that information would have been lost . So the information that we had in the primary layers can be fed explicitly to the later layers using the skip architecture.

3.2. Residual Learning

Let us consider a function H(x) that has an underlying mapping awaiting to be fit by a neural network model or by a few layers at a time. Let x be the input to this function that we are trying to map. Now, we have two choices. We can try to approximate H(x) or try approximating some new function F(x) := H(x) - x or H(x) = F(x) + x, and since we already know x, this task should be as difficult if not simpler than calculating H(x). Now, this F(x), known as the residual function is attempted to be approximated by the model. There is no difference between attempting to approximate H(x) and F(x) + x (which is to approximate F(x)). However, what could be different is the ease of training.

Let us assume that to our known approximation of H(x), we add a few layers which approximate just the Identity Function (I(x) = x). This should not ideally not affect the training error at all as these layers do not change the underlying function, they just depth to the model. By effect of the degradation effect discussed above, we know that the training error could be increased despite no change to the underlying model. However, we find that in our experimentation, it becomes safer to train deeper layers without



(Fig 5) Depicting the residual block attempting to mimic the identity function. F(x) here would be the residual function. On the right is a full architecture with the skip connections shown

encountering the degradation problem by much because in the worst case, the added Identity blocks do no harm to the performance. This is due to the training driving weights of the ReLu layers close to zero and so we have in effect, skipped those mappings altogether. Thus, our connection bypasses the faulty nonlinear filters and just mimics the Identity function. In cases, where the filter contributes meaningfully, the ReLu activations allow the weight layers to approximate the F(x).

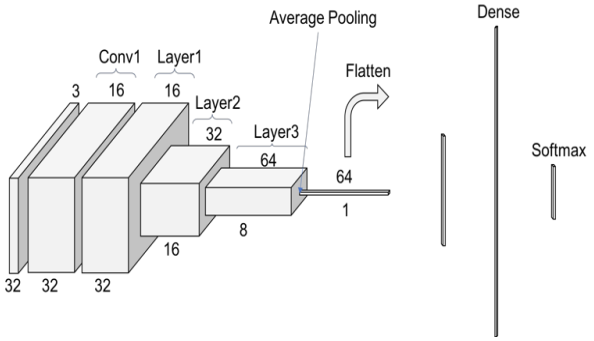
In effect, we have allowed the network to slow down and not "learn" anything wherever it is not completely necessary. We show by experiments (Fig. 7) that the learned residual functions in general have small responses, suggesting that identity mappings provide reasonable "outs" to the model allowing it to refine the hierarchy of abstractions.

3.3. Identity Mapping by Shortcuts

After every 'N' number of layers (A Hyperparameter), we introduce the residual block as shown in Fig. 2. Formally, in this paper we consider a building block as being defined by

$$\mathbf{y = F(x, \{Wi\}) + x}$$

where x, y are the input and output vectors, F represents the residual function which is a Function of the form



(Fig 6.) We show the architecture of ResNet-20 specifically, the model used to describe the benchmarks later in the implementation section.

$$\mathbf{F = W2 \times A(W1x)}$$

where , A stands for our activation, which in this case would be the ReLu function. We have not included the bias for the sake of simplicity. This shortcut connection neither increases the parameter count nor computational complexity or cost.
This is great for practice and implementation but also important in comparing the plain neural networks with our residual block neural networks; especially with the same number of hyperparameters, dimensions of layers and parameters. The comparison thereby becomes fair.

The Hyperparameter N (no of layers through which we allow skip connections) is typically a good value between 2 or 3. Making it 1 effectively makes it a plain neural network. Skipping a large number of layers is also possible. This skipping can apply to both FCNs and Conv layers as well.

In the case of Conv layers, the element-wise addition would be performed on two maps, channel by channel.

3.4 Benefits of Design

Our design has consistently across different experiments shown scalability by layers. As we add layers increasingly , our training error doesn't increase but behaves as expected, which is to decrease with increase in model complexity, thereby solving the degradation problem and the exploding/vanishing gradient problem by the _usage of residual blocks mimicking Identity
Functions where necessary. The design itself requires minimal modification to existing architecture but solves

Input: Values of x over a mini-batch: $\mathcal{B} = \{x_{1...m}\}$;
Parameters to be learned: γ, β
Output: $\{y_i = \mathrm{BN}_{\gamma,\beta}(x_i)\}$

$$\mu_{\mathcal{B}} \leftarrow \frac{1}{m}\sum_{i=1}^{m} x_i \qquad \text{// mini-batch mean}$$

$$\sigma^2_{\mathcal{B}} \leftarrow \frac{1}{m}\sum_{i=1}^{m}(x_i - \mu_{\mathcal{B}})^2 \qquad \text{// mini-batch variance}$$

$$\widehat{x}_i \leftarrow \frac{x_i - \mu_{\mathcal{B}}}{\sqrt{\sigma^2_{\mathcal{B}} + \epsilon}} \qquad \text{// normalize}$$

$$y_i \leftarrow \gamma \widehat{x}_i + \beta \equiv \mathrm{BN}_{\gamma,\beta}(x_i) \qquad \text{// scale and shift}$$

Algorithm 1: Batch Normalizing Transform, applied to activation x over a mini-batch.

a big problem of scalability. With the new found implications being that we can solve bigger tasks and datasets by stacking on a larger number of layers. Neural Network Models can approximate a larger range and a more complex variety of functions.

4 Training

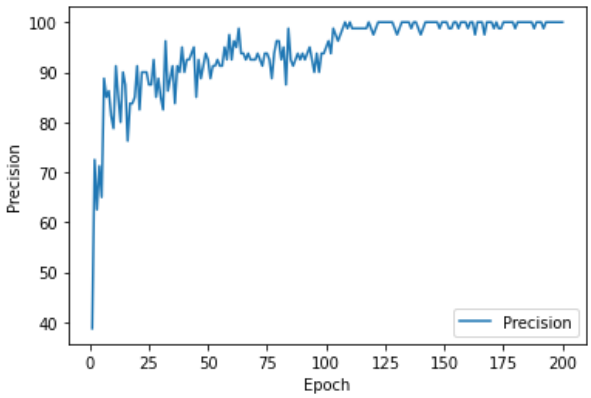
4.1 Batch Normalization

Training deep networks with mini-batch gradient descent is challenging as it makes the model sensitive to random weights and configurations. One possible reason for this difficulty is the distribution of the inputs

to layers deep in the network may change after each mini-batch when the weights are updated. This can cause the learning algorithm to forever chase a moving target. This change in the distribution of inputs to layers in the network is referred to by the technical name "internal covariate shift."

We propose a technique called Batch Normalization to train layers for a mini-batch.In Batch Normalization, we normalize the input layer by scaling the activations. For example, if we have features ranging from 0 to 1 and some from 1 to 1000, we normalize each of these batches so that they learn independently of each other. To increase the stability of a neural network, batch normalization normalizes the output of a previous activation layer by subtracting the batch mean and dividing by the batch standard deviation.

This reduces overfitting as it has a slight regularization effect similar to dropout. As a result, we can use higher



(Fig 8.) Test Curve of ResNet-32 showing high performance at 50 layers depth with precision at a high 90%.

learning rates as Batch Normalization reduces the extreme cases and reduces variance among the activations.

4.2 Learning Rate Scheduling

When training deep neural networks, it is often useful to reduce learning rate as the training progresses. This can be done by using pre-defined learning rate schedules. Learning rate schedules seek to adjust the learning rate during training by reducing the learning rate according to a predefined schedule. In our experimentation, we design a mechanism called step decay. Step decay schedule drops the learning rate by a factor every few epochs. It is mathematically denoted by

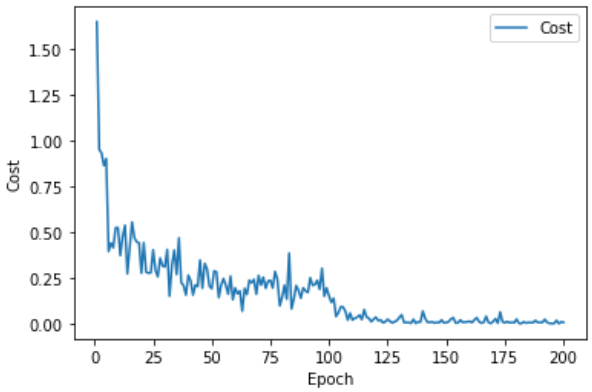
lr = lr0 * drop^floor(epoch / epochs_drop)

A typical way is to drop the learning rate by half every 10 epochs.

4.3 Training Architecture

For the sake of comparing results between existing State-of-the-Art models, the plain Conv Network[cite] and our Residual Network (hereby called ResNet for short), we have trained both networks uniformly across multiple experimentation with changing architecture, density and model hyperparameters.

In the case of training the Plain Network , we have trained the model on CIFAR-10 data which loads images at dimensions of 3 x 64 x 64. The CIFAR-10 dataset itself consists of 3 x 32 x 32 images but these



(Fig 8.) Training Curve of ResNet-32 showing high performance at 50 layers dept with decreasing cost.

have been upscaled to 3 x 64 x 64 for ease of Convolutional processing and to retain the core architecture of the model. The first layer is a conv layer with 11 x 11 filters applied with a stride of 4 and a padding of 2. The full network architecture can be found in [image]. The network ends with a series of fully connected layers and then a final softmax layer to classify the image. The total number of parameters is 57m with 0-non trainable parameters.

The plain network can be converted into a Residual Network with the simple addition of skip connections. We construct blocks of layers which can be skipped at a time, our hyperparameter for skipping to be 2 layers of convnets. We have 3 such layers each consisting of 5 blocks each , giving us a total of 3 * (5 * 2) + 2 conv layers. This value of depth itself can be easily modified to accommodate the skip connections as per requirement. The performance was measured on 18 and 32 layered models. On the data itself, batch normalization is used right before the input layer transformations. For the sake of data augmentation, the input images are flipped randomly horizontally and

randomly cropped to the dimensions of 32 x 4 and then normalized. We use SGD with a mini batch size of 128. Learning rate is initialized to 0.1 and then is decayed according to schedule of epochs. A momentum of 0.9 and a weight decay of 1e-4 is used. No dropout is used throughout the experiment. The hyperparameters have been reached as a result of previous suggestions from exiting research and by a certain amount of trial and error.

5 Implementation

The computational cost of deep networks is a relatively rare topic of academic paper. Of course, it doesn't make sense to discuss compression when the early network accuracy is not enough. Engineers need to implement models and let the network work in as many environments as possible. The resource consumption and speed of the model are critical.
You can use FLOPS (floating point operations per second) to measure the speed of the model. The other method is MACCs (multiply-accumulate operations, also called MAdds). Our model which is the 50 layer ResNet on predicting performance by using an architecture of 1MB internal memory and 1GB/s DDR bus for data movement between internal and external memories. We also have about 10 GFLOPS general hosts and 1 TFLOPS of matrix computation available to us.

The model for which there is a need to calculate inference performance is the deep neural network of ResNet-18 or our variant of the model with 18 convolutional network layers and a total of 51 layers.

All but the last 5 layers have input feature maps which can be stored inside our internal memory. All of our output feature maps can be stored inside our internal memory as they are each of size less than 1mb.

The formula to calculate input feature maps is -
Ni x Lr x Lc
The formula to calculate output feature maps is -
No x Mr x Mc
The formula to calculate filter coefficients is -
No x Ni x Fi x Fc

The total layer data movement time is approximately 5 * (10 ^ -3) or about 0.0055 seconds according to our calculations. Upon calculating, we get a computer time across layer as 3.2 * (10 ^ -4) seconds. per layer of CNN computation and 1.6 * (10 ^ -3) seconds for other operations.Summing up the times across all layers, we get 2.2 * (10 ^ -2) seconds per input across all layers as the predicted inference performance.

From our analysis it is clear that the first layers of convolution in a residual block takes longer than the others and it would improve future results to find a way to parallelize the convolutional computations that take place in a residual block. The biggest bottleneck is the data movement and the non-convolutional operations as we have been unable to parallelize those layers and mathematical functions.

Future architectures can improve by finding ways of parallelizing CNN 2D convolutions and matrix multiplications. Increasing the speeds from 1TFLOPS and 10 GFLOPS will see better performance speeds as models get increasingly more efficient.

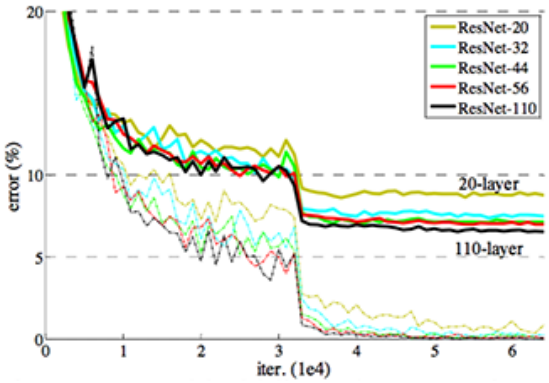
5.1 Findings

We found the following results upon training the ResNet model with increasing numbers of layers -

Model	Precision
ResNet-20	91.87%
ResNet-34	92.88%
ResNet-50	93.31%

As we can observe, our model improves with an increasing number of layers. Upon comparing with previous architectures, we observe that while increasing density decreases performance in AlexNet and LeNet, our ResNet successfully improves upon increase in complexity. We claim that ResNet successfully overcomes the previous models and provides results not seen thus far.

Graphs and Calculations have been provided in Appendix. The code used to build the models is fully reproducible and has been replicated several times to the same results.



(Fig 9.) Depicting the end-result of ResNet successfully decreasing error with increasing depth.

6 Conclusion

Neural Networks previously have seen a drop in training accuracy with increase in depth of layers. This is because of Vanishing/Exploding gradients and the Degradation phenomenon. Our model successfully solves the degradation problem by making use of a newly devised mechanism called as a skip-connection. The skip connection and the residual block allow the network to learn the Identity function through some layers thereby letting the network to refine the hierarchy of features. This modified network known as a Residual Network or a ResNet successfully scales with addition of layers as shown above , thereby allowing future research to build deeper and more complex models. In addition, the model does not use any additional parameters or computational cost, making the implementation efficient and simple.

7 References

Note: As stated above, this paper is a work of fiction. The following are the actual inventors of the ideas described in this paper.

...

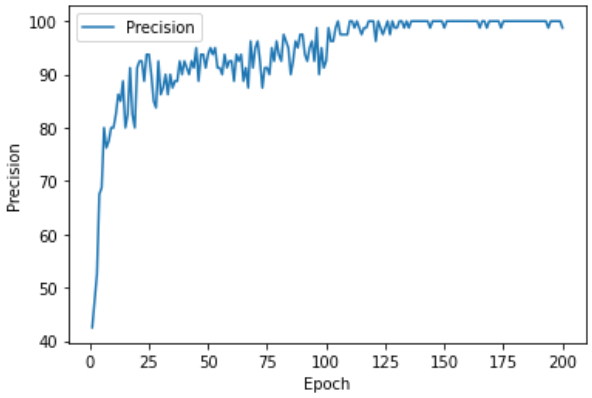
1. Krizhevsky, A., Sutskever, I., and Hinton, G. E. ImageNet classification with deep convolutional neural networks. In NIPS, pp. 1106–1114, 2012.
2. He, Kaiming, et al. "Deep residual learning for image recognition." arXiv preprint arXiv:1512.03385 (2015).
3. Szegedy, Christian, Sergey Ioffe, and Vincent Vanhoucke. "Inception-v4, inception-resnet and the impact of residual connections on learning." arXiv preprint arXiv:1602.07261 (2016).
4. Huang, Gao, Zhuang Liu, and Kilian Q. Weinberger. "Densely Connected Convolutional Networks." arXiv preprint arXiv:1608.06993 (2016).
5. [1605.06431] Residual Networks are Exponential Ensembles of Relatively Shallow Networks
6. Kurt Hornik,Maxwell Stinchcombe,Halbert White. "Multilayer feedforward networks are universal approximators".
7. S. Ioffe and C. Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. In ICML, 2015.
8. Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift{Sergey Ioffe and Christian Szegedy, 2015}
9. R. K. Srivastava, K. Greff, and J. Schmidhuber. Training very deep networks. 1507.06228, 2015.
10. V. Nair and G. E. Hinton. Rectified linear units improve restricted boltzmann machines. In ICML, 2010.
11. C. M. Bishop. Neural networks for pattern recognition. Oxford university press, 1995
12. Y. LeCun, B. Boser, J. S. Denker, D. Henderson, R. E. Howard, W. Hubbard, and L. D. Jackel. Backpropagation applied to handwritten zip code recognition. Neural computation, 1989. [23] Y. LeCun, L. Bottou, G. B. Orr, and K.-R. Muller. Efficient backprop. ¨ In Neural Networks: Tricks of the Trade, pages 9–50. Springer, 1998
13. https://github.com/Jaisu-1/ResNets-Cifar10
14. Excel link to calculations

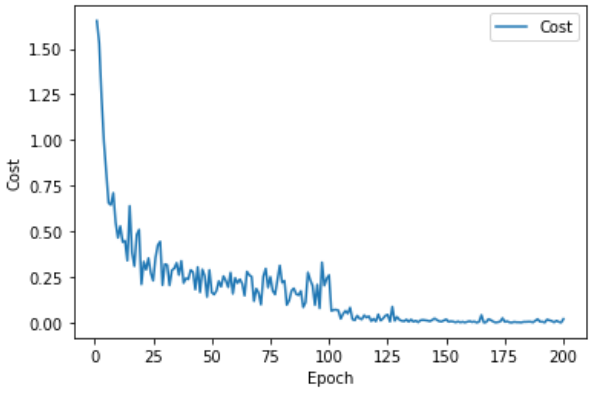
8 Appendix

Here lies various graphs and charts developed during the development of our model. We also provide rough calculations which were reflected in the implementations section.

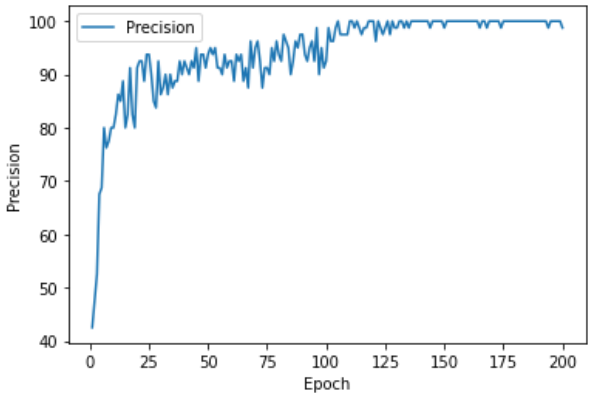
8.1 Graphs for Alternate ResNet Models

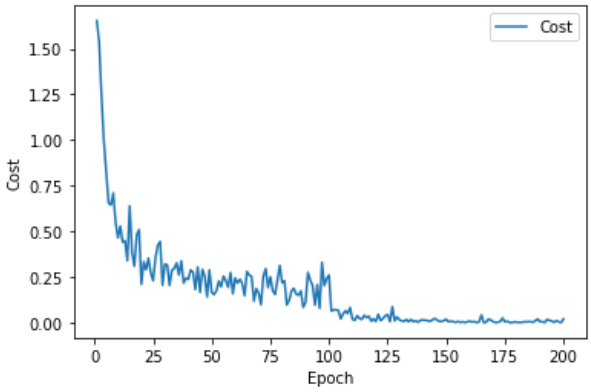
1) Resnet 20





2) Resnet 50





8.2 Calculation in a rough Excel Sheet

bytes	channels	o-dim	Total		no of kernels	kernel sizes	filter coefficients		bytes	channels	i-dim	Total
4	16	32	65536		48	9	1728		4	3	16	3072
4	16	32	65536		256	9	9216		4	16	16	16384
4	16	32	65536		256	9	9216		4	16	16	16384
4	16	32	65536		256	9	9216		4	16	16	16384
4	16	32	65536		256	9	9216		4	16	16	16384
4	16	32	65536		256	9	9216		4	16	16	16384
4	16	32	65536		256	9	9216		4	16	16	16384
4	32	16	32768		512	9	4608		4	16	32	65536
4	32	16	32768		1024	9	9216		4	32	32	131072
4	32	16	32768		1024	9	9216		4	32	32	131072
4	32	16	32768		1024	9	9216		4	32	32	131072
4	32	16	32768		1024	9	9216		4	32	32	131072
4	32	16	32768		1024	9	9216		4	32	32	131072
4	64	8	16384		2048	9	18432		4	32	64	524288
4	64	8	16384		4196	9	37764		4	64	64	1048576
4	64	8	16384		4196	9	37764		4	64	64	1048576
4	64	8	16384		4196	9	37764		4	64	64	1048576
4	64	8	16384		4196	9	37764		4	64	64	1048576
4	64	8	16384		4196	9	37764		4	64	64	1048576
1000000000	0.000001728	0	TFLOPS	MACS	computer time		10 GFLOPS	ops	compute time			
1000000000	0.000009216		1000000000000	1769472	0.000003538944		10000000000	1769472	0.0001769472			
1000000000	0.000009216		1000000000000	9437184	0.000018874368		10000000000	9437184	0.0009437184			
1000000000	0.000009216		1000000000000	9437184	0.000018874368		10000000000	9437184	0.0009437184			
1000000000	0.000009216		1000000000000	9437184	0.000018874368		10000000000	9437184	0.0009437184			
1000000000	0.000009216		1000000000000	9437184	0.000018874368		10000000000	9437184	0.0009437184			
1000000000	0.000009216		1000000000000	9437184	0.000018874368		10000000000	9437184	0.0009437184			
1000000000	0.000004608		1000000000000	9437184	0.000018874368		10000000000	9437184	0.0009437184			
1000000000	0.000009216		1000000000000	4718592	0.000009437184		10000000000	4718592	0.0004718592			
1000000000	0.000009216		1000000000000	9437184	0.000018874368		10000000000	9437184	0.0009437184			
1000000000	0.000009216		1000000000000	9437184	0.000018874368		10000000000	9437184	0.0009437184			
1000000000	0.000009216		1000000000000	9437184	0.000018874368		10000000000	9437184	0.0009437184			
1000000000	0.000009216		1000000000000	9437184	0.000018874368		10000000000	9437184	0.0009437184			
1000000000	0.000018432		1000000000000	9437184	0.000018874368		10000000000	9437184	0.0009437184			
1000000000	0.00108634		1000000000000	4718592	0.000009437184		10000000000	4718592	0.0004718592			
1000000000	0.00108634		1000000000000	9437184	0.000018874368		10000000000	9437184	0.0009437184			
1000000000	0.00108634		1000000000000	9437184	0.000018874368		10000000000	9437184	0.0009437184			
1000000000	0.00108634		1000000000000	9437184	0.000018874368		10000000000	9437184	0.0009437184			
1000000000	0.00108634		1000000000000	9437184	0.000018874368		10000000000	9437184	0.0009437184			
1000000000			1000000000000	9437184	0.000018874368		10000000000	9437184	0.0009437184			
1000000000												
1000000000					0.0003244032				0.01622016			
	0.005557844											
	0.000292518105											

8.3 Links to files

[Reproducible notebooks for the ResNets]
[Excel link to calculations]