ResNet on CIFAR - 10

Deep Residual Networks

<https://arxiv.org/pdf/1512.03385.pdf>

It seems completely obvious and intuitive at first glance that increasing the depth of a simple neural network should also increase it's training accuracy if not the test accuracy. One can reason that the test accuracy may decrease after some point due to overfitting, and it's completely correct to say so.

Is learning better networks as easy as stacking more layers?

The answer to this question is no , since another kind of problem (*degradation*) is exposed on training deeper networks due to which the training accuracy **decreases** .

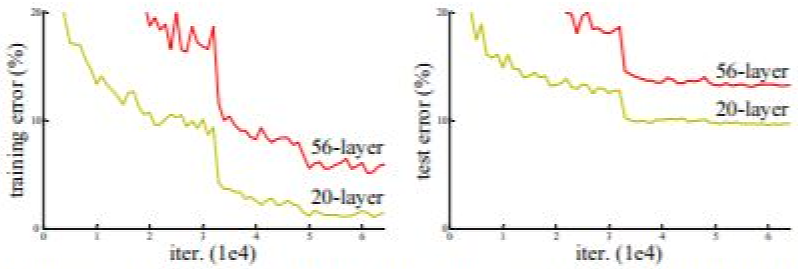


Figure 1. Training error (left) and test error (right) on CIFAR-10 with 20-layer and 56-layer "plain" networks. The deeper network has higher training error, and thus test error. Similar phenomena on ImageNet is presented in Fig. 4.

To overcome the above problem the authors of the paper introduced a residual learning framework in which the training accuracy actually does decrease on increasing the network depth.
Consider increasing the depth of a network as adding layers with identity mapping (worst case) . In a residual network these identity mappings can be learnt very easily since it only requires the weights to be close to zero . On the other hand, it becomes quite hard to learn this mapping leading to slower training and hence lesser accuracy .

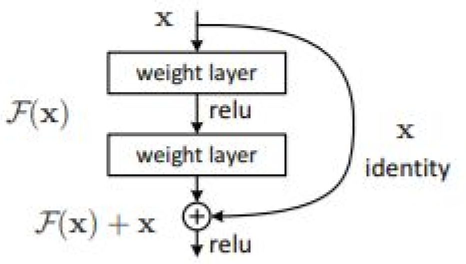

x
weight layer
$\mathcal{F}(\mathbf{x})$
relu
weight layer
x
identity
$\mathcal{F}(\mathbf{x}) + \mathbf{x}$
relu

Figure 2. Residual learning: a building block.

The architecture of a residual network as compared to plain networks having the same number of layers is shown in the following image . Note that even though the number of parameters in a ResNet is same it performs better than the simple network.

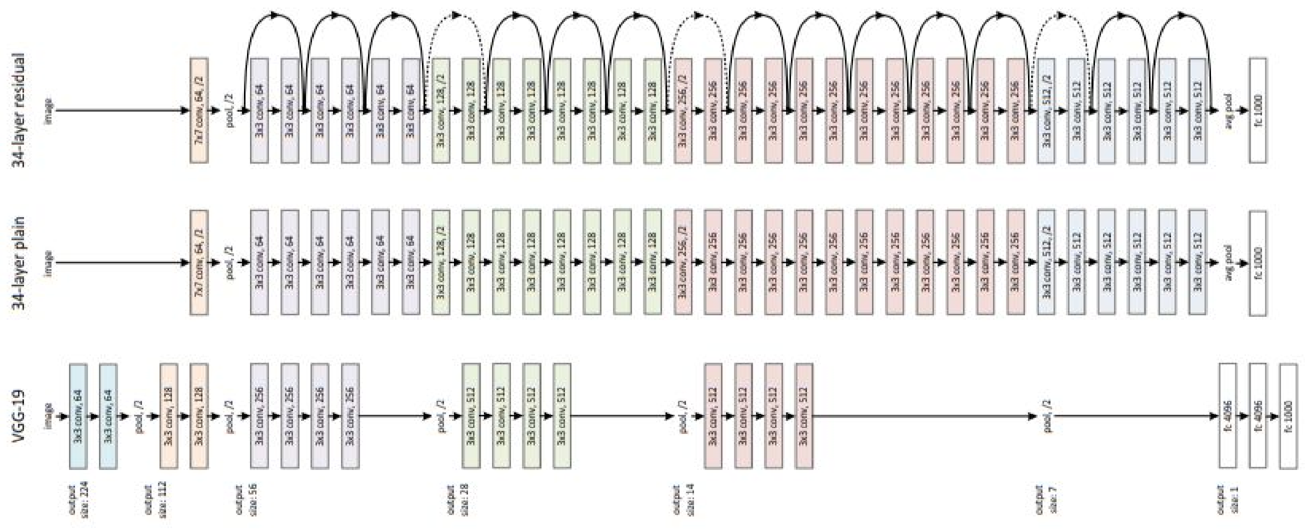
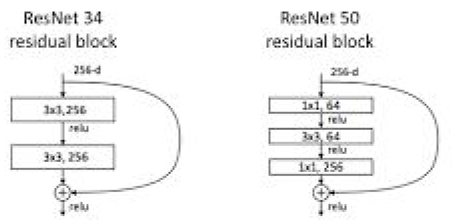
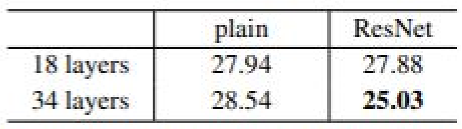


Figure 3. Example network architectures for ImageNet. **Left**: the VGG-19 model [41] (19.6 billion FLOPs) as a reference. **Middle**: a plain network with 34 parameter layers (3.6 billion FLOPs). **Right**: a residual network with 34 parameter layers (3.6 billion FLOPs). The dotted shortcuts increase dimensions. **Table 1** shows more details and other variants.





	plain	ResNet
18 layers	27.94	27.88
34 layers	28.54	**25.03**

Table 2. Top-1 error (%, 10-crop testing) on ImageNet validation. Here the ResNets have no extra parameter compared to their plain counterparts. Fig. 4 shows the training procedures.

Deeper ResNets can be further optimized by introducing bottleneck blocks as shown in the above figure (ResNet 50) . The advantage of using bottleneck architecture is that while it preserves the dimensions the computational complexity is significantly low as compared to normal residual blocks .

Training a ResNet on CIFAR-10 using Tensorflow 2

CIFAR-10 Dataset :

The network inputs are 32*32*3 images . The training set contains 50,000 images and the test set contains 10,000 images.

Model Architecture :

The first layer is 3×3 convolutions. Then we use a stack of 6n layers with 3×3 convolutions on the feature maps of sizes {32, 16, 8} respectively, with 2n layers for each feature map size. The numbers of filters are {16, 32, 64} respectively. The subsampling is performed by convolutions with a stride of 2. The network ends with a global average pooling, a 10-way fully-connected layer, and softmax. There are totally 6n+2 stacked weighted layers.

When shortcut connections are used, they are connected to the pairs of 3×3 layers (totally 3n shortcuts). On this dataset we use identity shortcuts in all cases .

Basic Block (n=5)	Bottleneck Block (n=10)
Total params: 491,018	Total params: 91,738
Trainable params: 487,626	Trainable params: 86,106
Non-trainable params: 3,392	Non-trainable params: 5,632

Our code uses class based models using tf.keras module .
(https://www.tensorflow.org/guide/keras/custom_layers_and_models)

We use Adam optimizer and Sparse Categorical Cross Entropy loss (since we want to apply softmax activation on the last layer , using one-hot encoding).

Conclusion :

I tried training the model in various settings. The results are shown below :

MODEL	**Epochs**	**Training Accuracy**	**Test Accuracy (Final)**	**Test Accuracy (Best)**	**Training Time**
BasicBlock ● **L2 Regularization 0.0001** ● **Data Augmentation** ● **n = 5**	100	93%	**87%**	**89%**	**50 min**
BasicBlock ● L2 Regularization 0.0001 ● Data Augmentation ● n = 5 ● elu activation	110	92%	86%	88%	92 min
BasicBlock ● L2 Regularization 0.0001 ● Data Augmentation ● n = 10	**75**	**95%**	87%	88%	115 min
BottleneckBlock ● **L2 Regularization 5* 1e-5** ● **Data Augmentation** ● **n = 10**	**100**	**86%**	**80%**	**82%**	**58 min**

Comments :

- As it is apparent the model with 'elu' activation takes approximately twice the time as compared to 'relu' model without improving on the accuracy. I expected it to perform better .

- ResNet 34 vs ResNet 64 : The latter takes approximately thrice the time to train an epoch, and clearly does not show any accuracy gains on the train set. The reason is that our dataset is fairly small for deeper nets due to which the model starts overfitting.
- I also tried increasing the learning rate of Adam optimizer to 0.01 , to my surprise , it slightly slowed learning as compared to the first model .
- The Bottleneck Block Model , despite having 5 times less number of parameters performed fairly well, I feel increasing it's depth further and training for more epochs may further increase it's performance.

Also, I would like to provide some tips from my experience - Use multiple Google accounts and run different models simultaneously . Do not train for all the epochs at one go, for example it is better to train 25 epochs 4 times than to train 100 epochs at one go. It helps to manually tune hyperparameters if required.