Introduction to Resnet or Residual Network

By Hussain Mujtaba - Sep 28, 2020

Over the last few years, there have been a series of breakthroughs in the field of Computer Vision. Especially with the introduction of deep Convolutional neural networks, we are getting state of the art results on problems such as image classification and image recognition. So, over the years, researchers tend to make deeper neural networks (adding more layers) to solve such complex tasks and to also improve the classification/recognition accuracy. But, it has been seen that as we go adding on more layers to the neural network, it becomes difficult to train them and the accuracy starts saturating and then degrades also. Here ResNet comes into rescue and helps solve this problem. In this article, we shall know more about ResNet and its architecture.

1. What is ResNet
   1. Need for ResNet
   2. Residual Block
   3. How ResNet helps
2. ResNet architecture
3. Using ResNet with Keras
   1. ResNet 50
What is ResNet?

ResNet, short for Residual Network is a specific type of neural network that was introduced in 2015 by Kaiming He, Xiangyu Zhang, Shaoqing Ren and Jian Sun in their paper “Deep Residual Learning for Image Recognition”. The ResNet models were extremely successful which you can guess from the following:

- Won 1st place in the ILSVRC 2015 classification competition with a top-5 error rate of 3.57% (An ensemble model)
- Replacing VGG-16 layers in Faster R-CNN with ResNet-101. They observed relative improvements of 28%
- Efficiently trained networks with 100 layers and 1000 layers also.

Need for ResNet

Mostly in order to solve a complex problem, we stack some additional layers in the Deep Neural Networks which results in improved accuracy and performance. The intuition behind adding more layers is that these layers progressively learn more complex features. For example, in case of recognising images, the first layer may learn to detect edges, the second layer may learn to identify textures and similarly the third layer can learn to detect objects and so on. But it has been found that there is a maximum threshold for depth with the traditional Convolutional neural network model. Here is a plot that describes error% on training and testing data for a 20 layer Network and 56 layers Network.

![Graph showing error% on training and testing data for different layer networks]

Source: Original paper

We can see that error% for 56-layer is more than a 20-layer network in both cases of training data as well as testing data. This suggests that with adding more layers on top of a network, its performance degrades. This could be blamed on the optimization function, initialization of the network and more importantly vanishing gradient problem. You might be thinking that it could be a result of overfitting too, but here the error% of the 56-layer network is worst on both training as well as testing data which does not happen when the model is overfitting.

Residual Block

https://www.mygreatlearning.com/blog/resnet/
This problem of training very deep networks has been alleviated with the introduction of ResNet or residual networks and these Resnets are made up from Residual Blocks.

![Diagram of residual learning: a building block.](image)

The very first thing we notice to be different is that there is a direct connection which skips some layers (may vary in different models) in between. This connection is called ‘skip connection’ and is the core of residual blocks. Due to this skip connection, the output of the layer is not the same now. Without using this skip connection, the input ‘x’ gets multiplied by the weights of the layer followed by adding a bias term.

Next, this term goes through the activation function, f() and we get our output as H(x).

\[ H(x) = f(wx + b) \]

or \[ H(x) = f(x) \]

Now with the introduction of skip connection, the output is changed to

\[ H(x) = f(x) + x \]

There appears to be a slight problem with this approach when the dimensions of the input vary from that of the output which can happen with convolutional and pooling layers. In this case, when dimensions of f(x) are different from x, we can take two approaches:

- The skip connection is padded with extra zero entries to increase its dimensions.
- The projection method is used to match the dimension which is done by adding 1×1 convolutional layers to input. In such a case, the output is:

\[ H(x) = f(x) + w_1x \]

Here we add an additional parameter w1 whereas no additional parameter is added when using the first approach.
**How ResNet helps**

The skip connections in ResNet solve the problem of vanishing gradient in deep neural networks by allowing this alternate shortcut path for the gradient to flow through. The other way that these connections help is by allowing the model to learn the identity functions which ensures that the higher layer will perform at least as good as the lower layer, and not worse. Let me explain this further.

Say we have a shallow network and a deep network that maps an input ‘x’ to output ‘y’ by using the function $H(x)$. We want the deep network to perform at least as good as the shallow network and not degrade the performance as we saw in case of plain neural networks (without residual blocks). One way of achieving so is if the additional layers in a deep network learn the identity function and thus their output equals inputs which do not allow them to degrade the performance even with extra layers.

![Diagram showing shallow and deep network](https://www.mygreatlearning.com/blog/resnet/)

It has been seen that residual blocks make it exceptionally easy for layers to learn identity functions. It is evident from the formulas above. In plain networks the output is

$$H(x) = f(x),$$

So to learn an identity function, $f(x)$ must be equal to $x$ which is grader to attain whereas incase of ResNet, which has output:

$$y = F(G(M(x))))$$

where $G$ and $M$ act as identity functions. Both the Network Give same output.
$$H(x) = f(x) + x,$$
$$f(x) = 0$$
$$H(x) = x$$

All we need is to make $f(x) = 0$ which is easier and we will get $x$ as output which is also our input.

In the best-case scenario, additional layers of the deep neural network can better approximate the mapping of ‘x’ to output ‘y’ than it’s the shallower counterpart and reduces the error by a significant margin. And thus we expect ResNet to perform equally or better than the plain deep neural networks.

Using ResNet has significantly enhanced the performance of neural networks with more layers and here is the plot of error% when comparing it with neural networks with plain layers.

![Error Plot](https://www.mygreatlearning.com/blog/resnet/)

Clearly, the difference is huge in the networks with 34 layers where ResNet-34 has much lower error% as compared to plain-34. Also, we can see the error% for plain-18 and ResNet-18 is almost the same.

**ResNet architecture**

ResNet network uses a 34-layer plain network architecture inspired by VGG-19 in which then the shortcut connection is added. These shortcut connections then convert the architecture into the residual network as shown in the figure below:
Using ResNet with Keras

https://www.mygreatlearning.com/blog/resnet/
Keras is an open-source neural network library written in Python which is capable of running on top of TensorFlow, Microsoft Cognitive Toolkit, R, Theano, or PlaidML. It is designed to enable fast experimentation with deep neural networks. Keras Applications include the following ResNet implementations and provide ResNet V1 and ResNet V2 with 50, 101, or 152 layers:

- ResNet50
- ResNet101
- ResNet152
- ResNet50V2
- ResNet101V2
- ResNet152V2

The primary difference between ResNetV2 and the original (V1) is that V2 uses batch normalization before each weight layer.

**ResNet 50**

To implement ResNet version 1 with 50 layers (**ResNet 50**), we simply use the function from Keras as shown below:

```python
1 | tf.keras.applications.ResNet50(
2 |     include_top=True,
3 |     weights="imagenet",
4 |     input_tensor=None,
5 |     input_shape=None,
6 |     pooling=None,
7 |     classes=1000,
8 | **kwargs)
```

**Arguments**

- **include_top**: whether to include the fully-connected layer at the top of the network.
- **weights**: one of None (random initialization), ’Imagenet’ (pre-training on ImageNet), or the path to the weights file to be loaded.
- **input_tensor**: optional Keras tensor (i.e. output of layers.Input()) to use as image input for the model.
- **input_shape**: optional shape tuple, only to be specified if include_top is False (otherwise the input shape has to be (224, 224, 3) (with ‘channels_last’ data format) or (3, 224, 224) (with ‘channels_first’ data format). It should have exactly 3 inputs channels, and width and height should be no smaller than 32. E.g. (200, 200, 3) would be one valid value.
- **pooling**: Optional pooling mode for feature extraction when include_top is False.
  - None means that the output of the model will be the 4D tensor output of the last convolutional block.
  - avg means that global average pooling will be applied to the output of the last convolutional block, and thus the output of the model will be a 2D tensor.
  - max means that global max pooling will be applied.
- **classes**: optional number of classes to classify images into, only to be specified if include_top is True, and if no weights argument is specified.
Similarly, we can use the rest of the variants of ResNet with Keras which you can find in their official documentation.

This brings us to the end of this article where we learned about ResNet and how it allows us to make deeper neural networks. Click the banner below for a free course on deep learning.

Hussain Mujtaba