Simonyan et al. [1] developed VGG in a paper that examines the depth of a Convolutional Neural Network with an in-depth study on ImageNet. By stacking a greater number of 3x3 convolutional layers on top of each other, the paper focuses on addressing all other design parameters and increasing the model's scope. This paper corresponds to current research at the time, which was convergent on the assumption that deeper networks contribute to better results, such as Goodfellow et al. [2], who used deep ConvNets for street number recognition, and GoogLeNet by Szegedy et al. [3], which won ILSVRC-2014 with a rather complex deep CNN.

The classification error on ImageNet decreased as the ConvNet depth increased, from 11 layers to 19 layers in a network, according to the VGG paper. Another noteworthy finding is that the number of parameters was not significantly higher than in previously proposed shallow networks.

The image above depicts VGG16, a well-known network derived from the paper. The current convolutional layers progressively decrease the size of the 224x224 image to a 7x7 image. The main benefit of using a stack of layers with smaller kernel sizes rather than a single layer with a large kernel size is that the former allows several ReLU to be incorporated, making the decision function more discriminative.

ResNet was introduced by He et al. [4] as a method for effectively training deeper neural networks. The paper tackles the issue of accuracy loss in very deep networks. They do this by constructing a residual network consisting of the blocks seen on the right. We can now let these layers match a residual mapping instead of hoping that each few stacked layers directly fit a desired underlying mapping. A residual network makes sense because we expect the deeper network to have no more training error than the shallower counterpart if we simply add layers that map identity only to the shallower network. As a result, if the identity mappings are ideal, mapping the residual to zero is simpler than fitting the underlying mapping identity.

The baseline plain network they create has less filters and lower complexity than VGG, according to the paper. In contrast to VGG19, the plain baseline has 19.6 billion FLOPs, while the residual network with 34 layers has 3.6 billion FLOPs.

The paper also adds Batch Normalization to network architecture, which means that forward propagated signals have non-zero variance, meaning that the gradient vanishing problem is addressed in both directions.

VGG accepts a 224x224 pixel RGB image as input. To keep the input image size constant for the ImageNet competition, the authors cropped out the middle 224x224 patch in each image.

VGG's convolutional layers have a very small receptive field (3x3, the smallest scale that captures both left and right and up and down). There are also 1x1 convolution filters that perform a linear transformation of the input before passing it through a ReLU unit. The convolution stride is set to 1 pixel in order to maintain spatial resolution after convolution.

VGG has three completely connected layers, the first two of which each have 4096 channels and the third of which has 1000 channels, one for each class.

VGG's hidden layers all use ReLU (a huge innovation from AlexNet that cut training time). Local Response Normalization (LRN) is not used by VGG because it increases memory consumption and training time without improving accuracy.

VGG uses very small receptive fields instead of large receptive fields like AlexNet (11x11 with a stride of 4). (3x3 with a stride of 1). The decision function is more discriminative now that there are three ReLU units instead of only one. There are also fewer parameters (27 times the number of channels vs. 49 times the number of channels in AlexNet).

Without modifying the receptive fields, VGG uses 1x1 convolutional layers to make the decision feature more non-linear.

VGG may have a large number of weight layers due to the limited size of the convolution filters; of course, more layers means better efficiency. However, this isn't an unusual element. In the 2014 ImageNet competition, GoogLeNet, another model that uses deep CNNs and small convolution filters, was also presented.

ResNet, like VGG, has a variety of configurations that define the number of layers and their sizes. Each layer is made up of convolutional layers, batch normalisation layers, and residual connections, which are all made up of blocks (also called skip connections or shortcut connections). ResNets use the word "layer" to refer to a collection of blocks, such as "layer 1 has two blocks," as well as the total number of layers in the entire ResNet, such as "ResNet18 has 18 layers."

The gradient signal either explodes (becomes very large) or vanishes (becomes very small) as it backpropagates over several layers, making training extremely deep neural networks difficult. Residual connections teach the model how to "skip" layers by lowering all of their weights and relying solely on the residual relation. As a result, if the ResNet152 model can learn the desired function between input and output using only the first 52 layers, the remaining 100 layers can set their weights to zero, and the output of the 52nd layer will simply move through the residual connections unaffected.

As can be seen, all ResNet models start with a typical 7x7 convolutional layer and a max pooling layer; these layers also have padding, which is not shown in the table. After that, there are four "layers," each with a different number of blocks.

There are two different blocks used, one called the Basic Block, and one called the Bottleneck block.

The initial 7x7 convolutional layer, batch normalisation, a ReLU activation feature, and a downsampling max pooling layer are all specified in our ResNet class. The four layers are then built using the config file, which defines the block to use, the number of blocks in the layer, and the number of channels in that layer. The number of channels in a layer for the BasicBlock is simply the number of filters for both convolutional layers within the block.

Also, the first layer has a one-stride stride, while the last three layers have a two-stride stride. This stride is only used in the "downsampling" residual route and to adjust the stride of the first convolutional layer within a block.

The BasicBlock comes first.

Two 3x3 convolutional layers make up the BasicBlock. Conv1 has a stride that differs depending on the layer (one in the first layer and two in the other layers), while conv2 always has a stride of one. Each of the layers has a padding of one, which means that before applying the filters to the input image, we add a single pixel around the entire image that is zero in every channel. A ReLU activation feature and batch normalisation follow each convolutional sheet.

When downsampling, we apply a convolutional layer to the residual path with a 1x1 filter and no padding. This is accompanied by batch normalisation and has a variable stride. A 1x1 filter with a stride of one does not affect the height or width of an image; instead, it has out channels number of filters, each with a depth of in channels, i.e. it is increasing the number of channels in an image through a linear projection rather than downsampling. With a stride of two, the image's height and width are reduced by two, since the 1x1 filter just passes over any other pixel - this time, the image is simply downsampled as well as the channels' linear projection.

Because each of the convolutional layers within a block uses the same number of filters, the BasicBlock has an expansion of one.

The ResNet paper also covers variants specifically for the CIFAR10/100 datasets.

There is no pooling layer after the first convolutional layer, which has a smaller filter scale, lower stride, and less padding. It also only has three layers instead of four, and its own block form.

One stipulation is that each layer's number of channels must be exactly double that of the previous layer.

The only difference between the CIFARBasicBlock and the regular BasicBlock is the downsampling residual relation.

According to the paper, the ResNet models for CIFAR use a downsampling relation with "zero padding" and "parameter-free shortcuts." This is accomplished by the use of an Identity module.

There are 70,000 photos in total. Each MNIST picture is a digit between 0 and 9 written by government employees and census bureau employees. Each picture also has ten labels attached to it: zero, one, two, three, four, five, six, seven, eight, and nine. Each image is 28 pixels wide and 28 pixels tall, with a width of 28 pixels and a height of 28 pixels. The MNIST data is in reasonable quality. The digits have been preprocessed to ensure that they are perfectly aligned.

CIFAR is pronounced C-FAR (see-far). The number suffix in CIFAR10 and CIFAR100 refers to the number of classes/categories. Each CIFAR has a 32x32 pixel resolution. A total of 50,000 training images and 10,000 label images are available. Each CIFAR10 image corresponds to one of ten categories that have been labelled. Each of the CIFAR100 images corresponds to one of the 100 groups. ImageNet compares 1000 different categories. The 10 categories are: aeroplane, car, bird, cat, deer, dog, frog, horse, ship, and truck, in no specific order.

Instead of reserving a portion of the data from the training set for validation, we will simply use the test set as our validation set. This simply adds to the amount of data available for training. We will also normalise the image tensors through each channel by subtracting the mean and dividing by the standard deviation. In addition, when loading images from the training dataset, we can apply transformations at random. Take a random 32 x 32-pixel crop, and then flip the image horizontally with a 50% chance.

I'd like to point out that VGG was not optimised for a dataset with such a limited input size as MNIST, which has images that are 28x28 pixels in size, so I'd like to talk about how I dealt with that first. I considered shrinking MNIST to the smallest size permitted by VGG and adjusting the network accordingly. MNIST was resized to 32x32, which means that the convolutional layers would minimise the image size to 1x1 with 512 filters over time. As a result, instead of the original 512x7x7, my input size for fully connected layers will be 512x1x1. We also normalised the image tensors through each channel by subtracting the mean and dividing by the standard deviation.