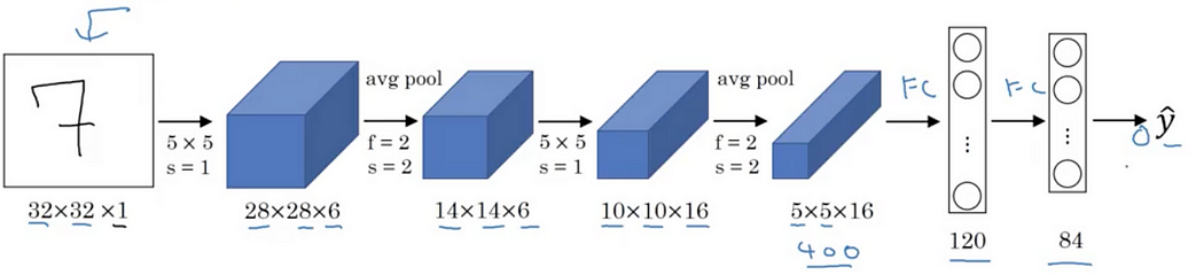
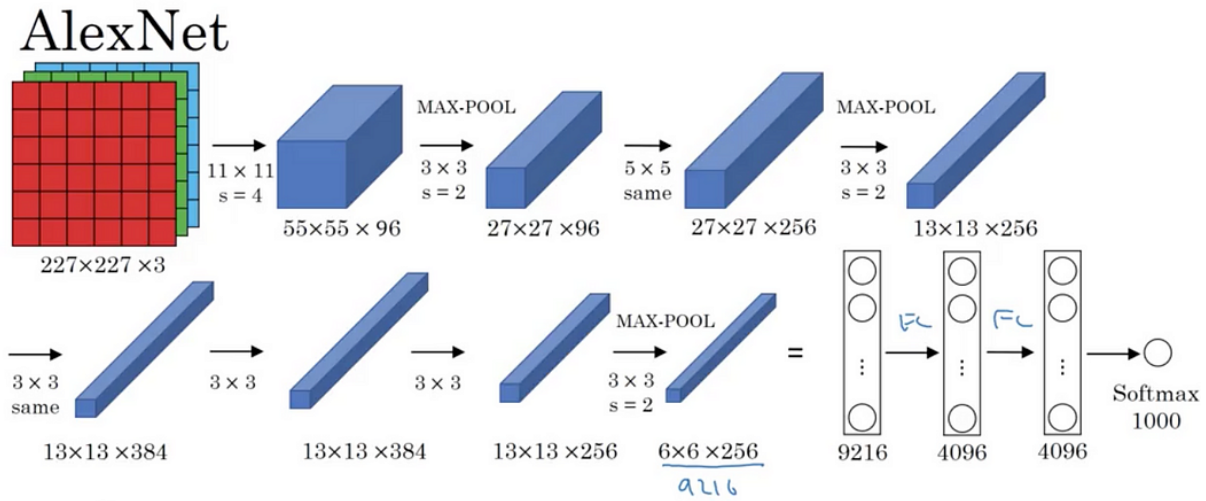
LeNet - 5



LeCun et al., 1998. Gradient-based learning applied to document recognition]

Andrew Ng

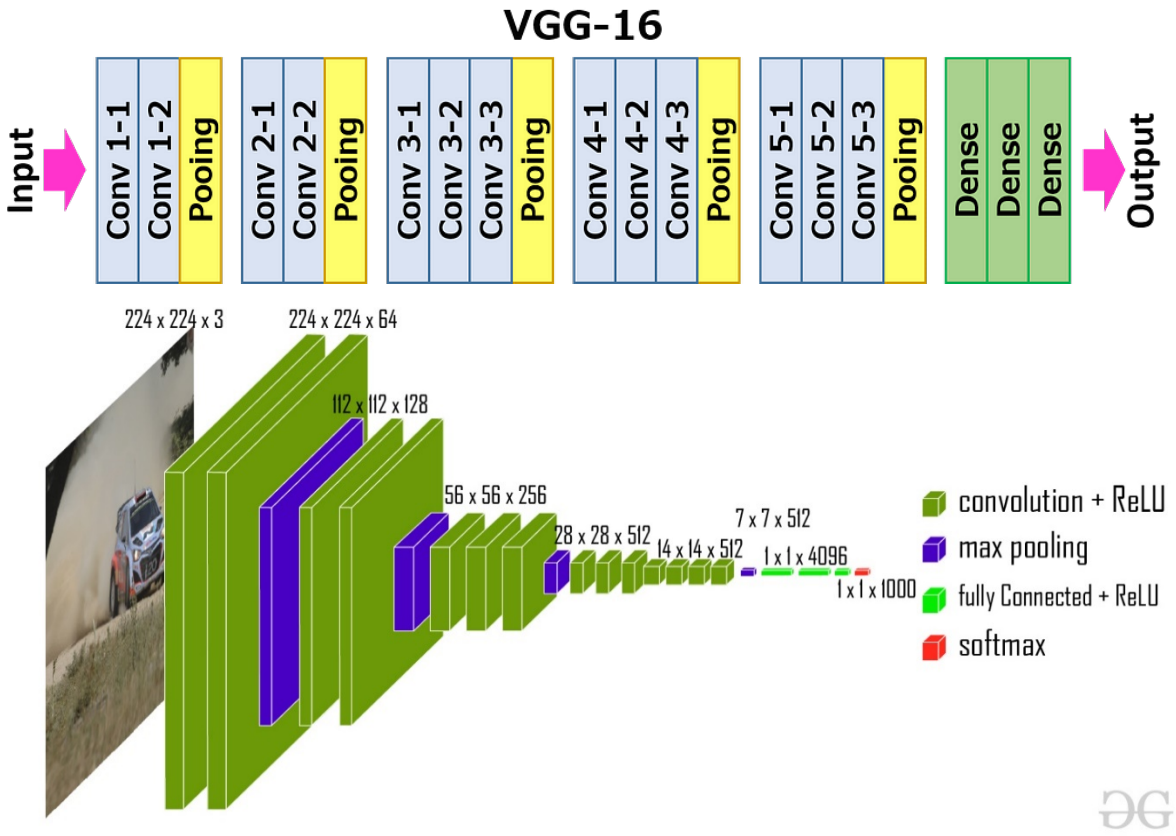
- Input image of 32 x 32 x 1 (Greyscale image)
- It uses Sigmoid or Tanh nonlinearity functions.



[Krizhevsky et al., 2012. ImageNet classification with deep convolutional neural networks]

Andrew Ng

- It uses ReLU activation function instead Sigmoid or Tanh functions. It speeds by more than 5 times faster with same accuracy
- It uses “Dropout” instead of regularisation to deal with overfitting. But the training time is doubled with dropout ratio of 0.5
- More data and bigger model with 7 hidden layers, 650K units and 60M parameters.



- Image input size: The image input size for VGG-16 is 224-by-224
- Relu
- Softmaxx

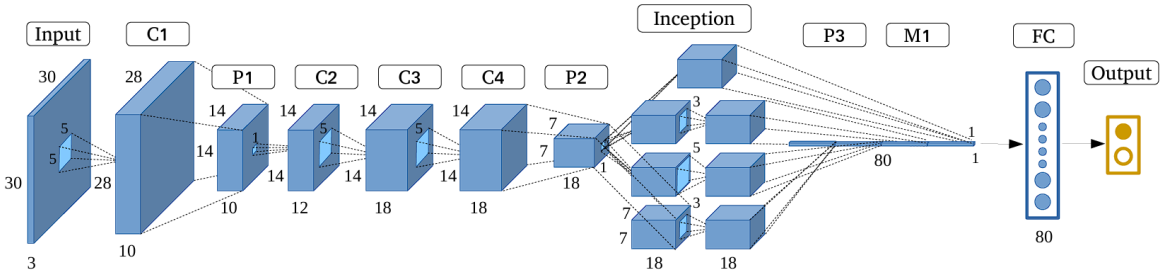
VGG-16

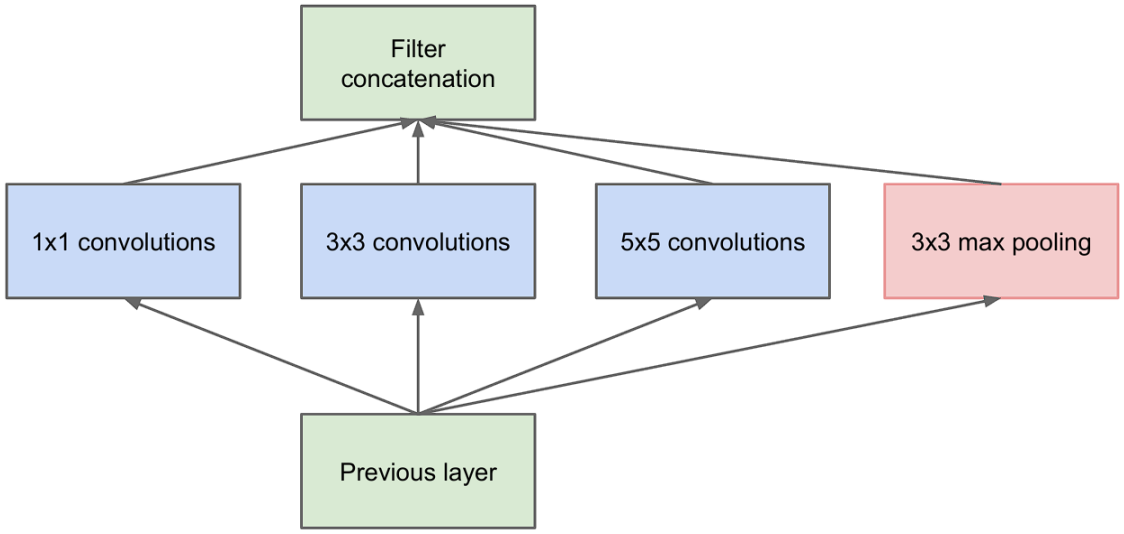
The VGG-16 model is a convolutional neural network (CNN) architecture that was proposed by the Visual Geometry Group (VGG) at the University of Oxford. It is characterized by its depth, consisting of 16 layers, including 13 convolutional layers and 3 fully connected layers. VGG-16 is renowned for its simplicity and effectiveness, as well as its ability to achieve strong performance on various computer vision tasks, including image classification and object recognition. The model's architecture features a stack of convolutional layers followed by max-pooling layers, with progressively increasing depth. This design enables the model to learn intricate hierarchical representations of visual features, leading to robust and accurate predictions. Despite its simplicity compared to more recent architectures, VGG-16 remains a popular choice for many deep learning applications due to its versatility and excellent performance.

This model achieves *92.7% top-5* test accuracy on the ImageNet dataset which contains *14* million images belonging to 1000 classes.

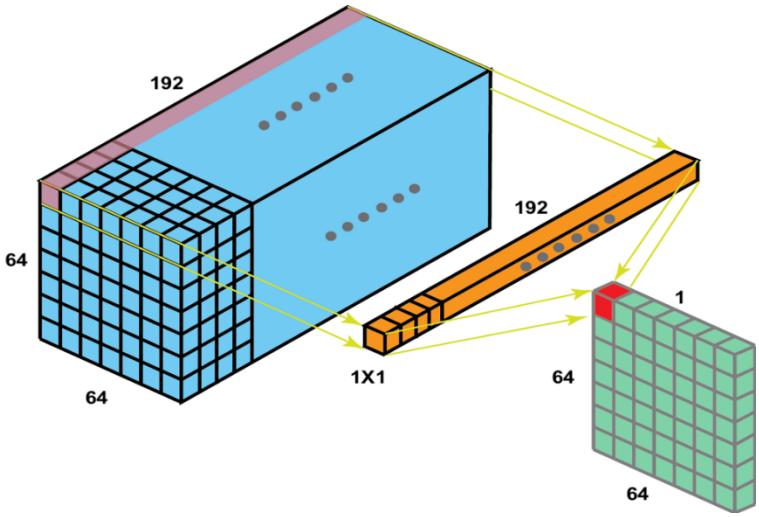
GoogLeNet

The winner of ILSVRC 2014 and the GoogLeNet architecture is also known as Inception Module. It goes deeper in parallel paths with different receptive field sizes and it achieved a top-5 error rate with of 6.67%.





Inception Module



1x1 Convolution

This architecture consists of 22 layer in deep. It reduces the number of parameters from 60 million (AlexNet) to 4 million.

The Inception Module is a building block for convolutional neural networks (CNNs) introduced by Google researchers in their seminal paper "Going Deeper with Convolutions" in 2014. This architecture, also known as GoogLeNet, represented a significant advancement in deep learning for computer vision tasks. The Inception Module is designed to allow a CNN to benefit from multi-level feature extraction by implementing filters of various sizes in the same layer of the network.

Key Features of the Inception Module

The Inception Module is characterized by several key features that differentiate it from traditional CNN layers:

- **Multi-level Feature Extraction:** The module applies several convolutional filters of different sizes (e.g., 1x1, 3x3, 5x5) to the input simultaneously. This allows the network to capture information at various scales and complexities.
- **Dimensionality Reduction:** The use of 1x1 convolutions serves as a method for dimensionality reduction, reducing computational complexity and the number of parameters without losing depth in the network.
- **Pooling:** In addition to convolutional filters, the Inception Module includes a parallel pooling branch (usually max pooling), which provides another form of spatial aggregation.
- **Concatenation:** The outputs of all filters and the pooling layer are concatenated along the channel dimension before being fed to the next layer. This concatenation ensures that the subsequent layers can access features extracted at different scales.

Challenges with the Inception Module

While the Inception Module brings many benefits, it also introduces certain challenges:

- **Increased Complexity:** The architecture of the Inception Module is more complex than traditional layers, which can make it harder to design and train.
- **Hyperparameter Tuning:** The module introduces additional hyperparameters, such as the number and sizes of filters, which require careful tuning to achieve optimal performance.
- **Resource Intensity:** Although designed for efficiency, the Inception Module can still be resource-intensive due to the large number of operations and concatenation of outputs.

ResNet (2015)

https://classic.d2l.ai/chapter_convolutional-modern/resnet.html

Residual Networks (ResNet)

As we design increasingly deeper networks it becomes imperative to understand how adding layers can increase the complexity and expressiveness of the network. Even more important is the ability to design networks where adding layers makes networks strictly more expressive rather than just different. To make some progress we need a bit of mathematics.

Function Classes

Consider F, the class of functions that a specific network architecture (together with learning rates and other hyperparameter settings) can reach. That is, for all $f \in F$ there exists some set of parameters (e.g., weights and biases) that can be obtained through training on a suitable dataset. Let us assume that f^* is the "truth" function that we really would like to find. If it is in F, we are in good shape but typically we will not be quite so lucky. Instead, we will try to find some f^*F which is our best bet within F. For instance, given a dataset with features X and labels y, we might try finding it by solving the following optimization problem:

$$f_{\mathcal{F}}^* \stackrel{\text{def}}{=} \operatorname*{argmin}_f L(\mathbf{X}, \mathbf{y}, f) \text{ subject to } f \in \mathcal{F}.$$

It is only reasonable to assume that if we design a different and more powerful architecture F' we should arrive at a better outcome.

For deep neural networks, if we can train the newly-added layer into an identity function $f(x)=x$, the new model will be as effective as the original model. As the new model may get a better solution to fit the training dataset, the added layer might make it easier to reduce training errors.

This is the question that He et al. considered when working on very deep computer vision models (). At the heart of their proposed *residual network* (*ResNet*) is the idea that every additional layer should more easily contain the identity function as one of its elements. These considerations are rather profound but they led to a surprisingly simple solution, a *residual block*. With it, ResNet won the ImageNet Large Scale Visual Recognition Challenge in 2015. The design had a profound influence on how to build deep neural networks.

7.6.2. Residual Blocks

Let us focus on a local part of a neural network, as depicted in Fig. 7.6.2. Denote the input by x. We assume that the desired underlying mapping we want to obtain by learning is $f(x)$, to be used as the input to the activation function on the top. On the left of Fig. 7.6.2, the portion within the dotted-line box must directly learn the mapping $f(x)$. On the right, the portion within the dotted-line box needs to learn the *residual mapping* $f(x)-x$, which is how the residual block derives its name. If the identity mapping $f(x)=x$ is the desired underlying mapping, the residual mapping is easier to learn: we only need to push the weights and biases of the upper weight layer (e.g., fully-connected layer and convolutional layer) within the dotted-line box to zero. The right figure in Fig. 7.6.2 illustrates the *residual block* of ResNet, where the solid line carrying the layer input x to the addition operator is called

a *residual connection* (or *shortcut connection*). With residual blocks, inputs can forward propagate faster through the residual connections across layers.

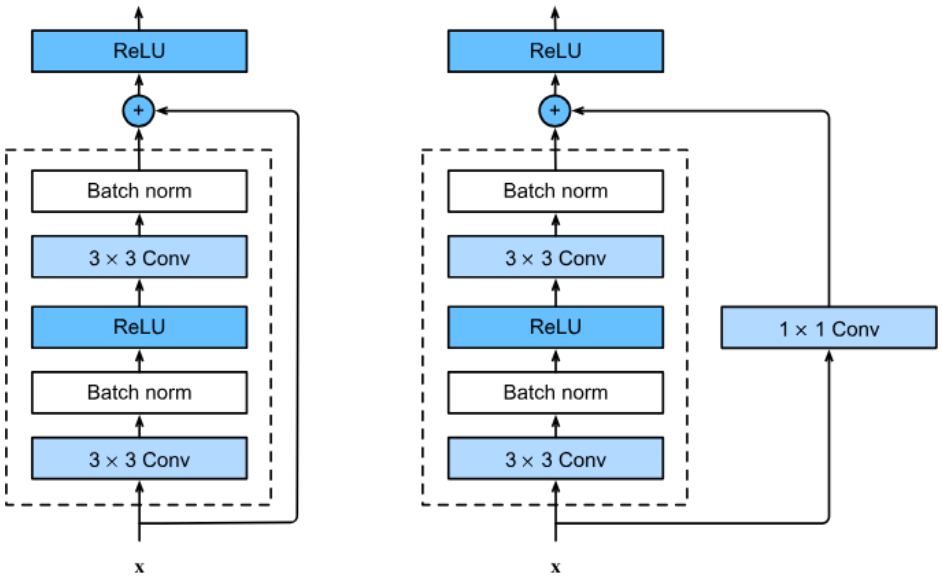


Fig. 7.6.2

ResNet follows VGG's full 3×3 convolutional layer design. The residual block has two 3×3 convolutional layers with the same number of output channels. Each convolutional layer is followed by a batch normalization layer and a ReLU activation function. Then, we skip these two convolution operations and add the input directly before the final ReLU activation function. This kind of design requires that the output of the two convolutional layers has to be of the same shape as the input, so that they can be added together. If we want to change the number of channels, we need to introduce an additional 1×1 convolutional layer to transform the input into the desired shape for the addition operation.

There are 4 convolutional layers in each module (excluding the 1×1 convolutional layer). Together with the first 7×7 convolutional layer and the final fully-connected layer, there are 18 layers in total. Therefore, this model is commonly known as ResNet-18. By configuring different numbers of channels and residual blocks in the module, we can create different ResNet models, such as the deeper 152-layer ResNet-152. Although the main architecture of ResNet is similar to that of GoogLeNet, ResNet's structure is simpler and easier to modify. All these factors have resulted in the rapid and widespread use of ResNet. Below Fig. depicts the full ResNet-18.