2 Convolutional Neural Network

Unit - II

Syllabus

At the end of this unit, you should be able to understand and comprehend the following syllabus topics:

- Introduction to CNN
  - Convolution Operation
  - Parameter Sharing
- Equivariant Representation
  - Pooling
  - The basic Architecture of CNN
- Variants of the Basic Convolution Function
- Popular CNN Architecture – AlexNet

2.1 Convolution Neural Network (CNN or ConvNets)

- To classify more complex images, you typically use a more sophisticated type of artificial neural network called a convolution neural network. The neural networks that were discussed so far were fully connected networks, meaning that each neuron in the next layer is connected to each neuron of the preceding layer. In the convolution network, each neuron in the next layer is connected only to a group of closely located neurons of the preceding layer, typically called a local receptive field (or a patch).
- For example, the Fig. 2.1.1 shows connections of a hidden neuron to a patch of neurons.

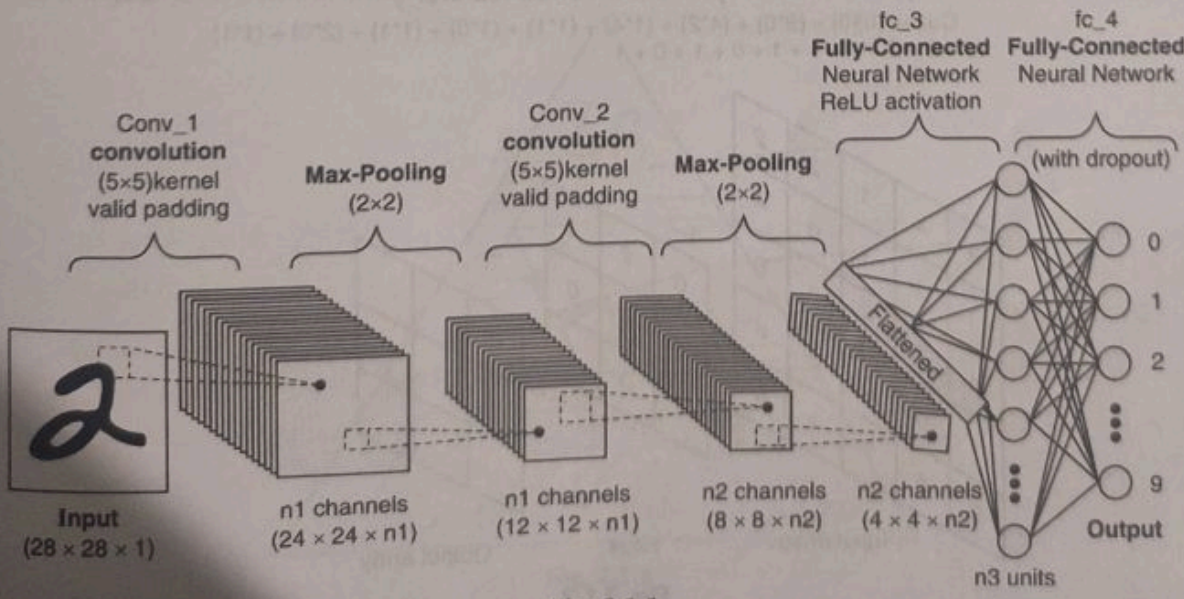


Fig. 2.1.1

- The next neuron in the hidden layer is connected to a patch obtained by sliding the patch over by one pixel, and so on. So, each hidden neuron learns to analyse its particular local receptive field, rather than all input neurons. The convolution network structure is much more complex than the fully connected networks.

2.1.1 How do Convolutional Neural Networks Work ? (Components of CNN)

- Convolutional neural networks are distinguished from other neural networks by their superior performance with image, speech, or audio signal inputs. They have three main types of layers as following.

1. Convolutional layer 2. Pooling layer 3. Fully-connected (FC) layer

- The convolutional layer is the first layer of a convolutional network. While convolutional layers can be followed by additional convolutional layers or pooling layers, the fully-connected layer is the final layer. With each layer, the CNN increases in its complexity, identifying greater portions of the image. Earlier layers focus on simple features, such as colours and edges. As the image data progresses through the layers of the CNN, it starts to recognise larger elements or shapes of the object until it finally identifies the intended object.

Let's learn about these layers in little more detail.

2.1.2 Convolutional Layer

- The convolutional layer is the core building block of a CNN, and it is where the majority of computation occurs. It requires a few components, which are input data, a filter, and a feature map. Let's assume that the input will be a colour image, which is made up of a matrix of pixels in 3D.
- This means that the input will have three dimensions a height, width, and depth which correspond to RGB in an image. You also have a feature detector, also known as a kernel or a filter, which will move across the receptive fields of the image, checking if the feature is present. This process is known as a convolution.
- The feature detector is a two-dimensional (2-D) array of weights, which represents part of the image. While they can vary in size, the filter size is typically a 3x3 matrix; this also determines the size of the receptive field. The filter is then applied to an area of the image, and a dot product is calculated between the input pixels and the filter. This dot product is then fed into an output array. Afterwards, the filter shifts by a stride, repeating the process until the kernel has swept across the entire image. The final output from the series of dot products from the input and the filter is known as a feature map, activation map, or a convolved feature.

Output [0][0] = (9*0) + (4*2) + (1*4) + (1*1) + (1*0) + (1*1) + (2*0) + (1*1)
= 0 + 8 + 1 + 4 + 1 + 0 + 1 + 0 + 1
= 16

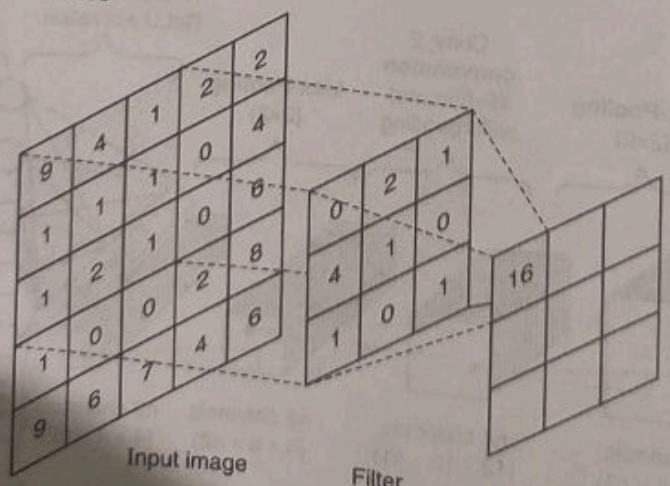


Fig. 2.1.2

- As you can see in the image, each output value in the feature map does not have to connect to each pixel value in the input image. It only needs to connect to the receptive field, where the filter is being applied. Since the output array does not need to map directly to each input value, convolutional (and pooling) layers are commonly referred to as "partially connected" layers. However, this characteristic can also be described as local connectivity.
- Note that the weights in the feature detector remain fixed as it moves across the image, which is also known as parameter sharing. Some parameters, like the weight values, adjust during training through the process of backpropagation and gradient descent. However, there are three hyperparameters which affect the volume size of the output that need to be set before the training of the neural network begins.

They are as following.

1. The number of filters affects the depth of the output. For example, three distinct filters would yield three different feature maps, creating a depth of three.
2. Stride is the distance, or number of pixels, that the kernel moves over the input matrix. While stride values of two or greater is rare, a larger stride yields a smaller output.
3. Zero-padding is usually used when the filters do not fit the input image. This sets all elements that fall outside of the input matrix to zero, producing a larger or equally sized output. There are three types of padding.
   (a) Valid padding: This is also known as no padding. In this case, the last convolution is dropped if dimensions do not align.
   (b) Same padding: This padding ensures that the output layer has the same size as the input layer.
   (c) Full padding: This type of padding increases the size of the output by adding zeros to the border of the input.

- After each convolution operation, a CNN applies a Rectified Linear Unit (ReLU) transformation to the feature map, introducing nonlinearity to the model.
- As mentioned earlier, another convolution layer can follow the initial convolution layer. When this happens, the structure of the CNN can become hierarchical as the later layers can see the pixels within the receptive fields of prior layers. As an example, let's assume that you are trying to determine if an image contains a bicycle. You can think of the bicycle as a sum of parts. It is comprised of a frame, handlebars, wheels, pedals, et cetera. Each individual part of the bicycle makes up a lower-level pattern in the neural net, and the combination of its parts represents a higher-level pattern, creating a feature hierarchy within the CNN.

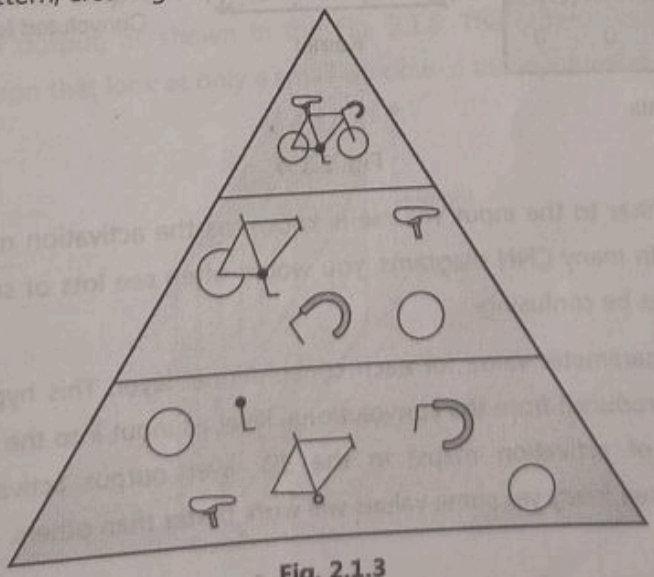


Fig. 2.1.3

Ultimately, the convolutional layer converts the image into numerical values, allowing the neural network to interpret and extract relevant patterns.

2.1.3 Components of Convolution Layers

- Convolutional layers have parameters for the layer and additional hyperparameters. Gradient descent is used to train the parameters in this layer such that the class scores are consistent with the labels in the training set. Following are the major components of convolutional layers.
  1. Filters
  2. Activation maps
  3. Parameter sharing
  4. Layer-specific hyperparameters
- Let's briefly learn about them.

Filters

- The parameters for a convolutional layer configure the layer's set of filters. Filters are a function that has a width and height smaller than the width and height of the input volume. Note here that you can have a filter size equal to the input volume, but typically only in one dimension, not both.
- Filters (e.g., convolutions) are applied across the width and height of the input volume in a sliding window manner. Filters are also applied for every depth of the input volume. You compute the output of the filter by producing the dot product of the filter and the input region. The Fig. 2.1.4 illustrates the concept of a filter.

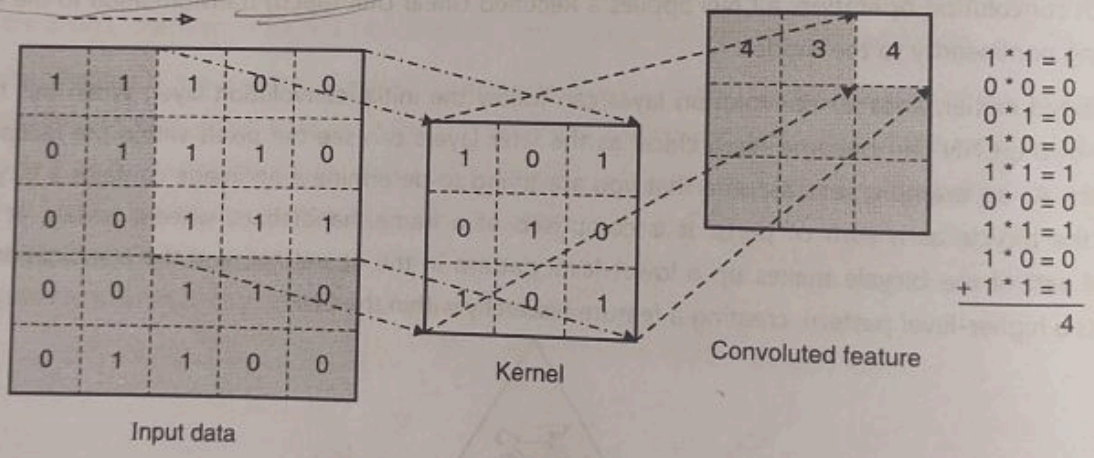


Fig. 2.1.4

- The output of applying a filter to the input volume is known as the activation map (sometimes referred to as a feature map) of that filter. In many CNN diagrams, you would often see lots of small activation maps; how these are produced can sometimes be confusing.
- The filter count is a hyperparameter value for each convolutional layer. This hyperparameter also controls how many activation maps are produced from the convolutional layer as input into the next layer and is considered the third dimension (number of activation maps) in the 3D layer output activation volume. The filter count hyperparameter can be chosen freely yet some values will work better than others.

- The architecture of CNNs is set up such that the learned filters produce the strongest activation to spatially local input patterns. This means that filters are learned that will activate on patterns (or features) only when the patterns occur in the training data in their respective field. As you move farther along in layers in a CNN, you encounter filters that can recognise nonlinear combinations of features and are increasingly global in how they can detect patterns. High-performing convolutional architectures have shown network depth to be an important factor in CNNs.

Activation Maps

- As you understand, an activation is a numerical result if a neuron decided to let information pass through (fire). This is a function of the inputs to the activation function, the weights on the connections (for the inputs, and the type of activation function itself). When you say the filter "activates", you mean that the filter lets information pass through it from the input volume into the output volume.
- You slide each filter across the spatial dimensions (width, height) of the input volume during the forward pass of information through the CNN. This produces a two-dimensional output called an activation map for that specific filter. The Fig. 2.1.5 depicts how this activation map relates to the previously introduced concept of a convoluted feature.

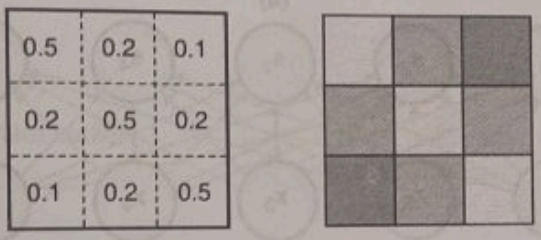


Fig. 2.1.5

- To compute the activation map, you slide the filter across the input volume depth slice. You calculate the dot product between the entries in the filter and the input volume. The filter represents the weights that are being multiplied by the moving window (subset) of input activations. Networks learn filters that activate when they see certain types of features in the input data in a specific spatial position.
- You create the three-dimensional output volume for the convolution layer by stacking these activation maps along the depth dimension in the output, as shown in the Fig. 2.1.6. The output volume will have entries that you consider the output of a neuron that look at only a small window of the input volume.

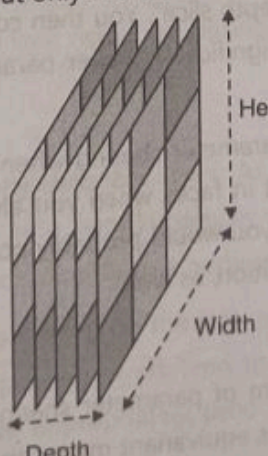


Fig. 2.1.6

Parameter Sharing

- Parameter sharing refers to using the same parameter for more than one function in a model. In a traditional neural network, each element of the weight matrix is used exactly once when computing the output of a layer. It is multiplied by one element of the input and then never revisited. As a synonym for parameter sharing, you can say that a network has tied weights, because the value of the weight applied to one input is tied to the value of a weight applied elsewhere. In a convolutional neural network, each member of the kernel is used at every position of the input (except perhaps some of the boundary pixels, depending on the design decisions regarding the boundary). The parameter sharing used by the convolution operation means that rather than learning a separate set of parameters for every location, you learn only one set.

The Fig. 2.1.7 illustrates parameter sharing.

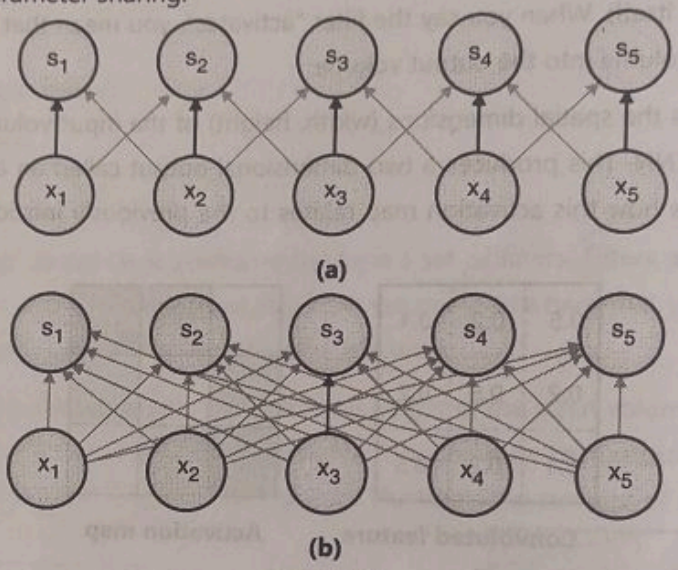


Fig. 2.1.7

- In the Fig. 2.1.7, the black arrows indicate the connections that use a particular parameter in two different models. In the Fig. 2.1.7(a), the black arrows indicate uses of the central element of a 3-element kernel in a convolutional model. Because of parameter sharing, this single parameter is used at all input locations. In the Fig. 2.1.7(b), the single black arrow indicates the use of the central element of the weight matrix in a fully connected model. This model has no parameter sharing, so the parameter is used only once.
- CNNs use a parameter-sharing scheme to control the total parameter count. This helps training time because you will use fewer resources to learn the training dataset. To implement parameter sharing in CNNs, you first denote a single two-dimensional slice of depth as a "depth slice". You then constrain the neurons in each depth slice to use the same weights and bias. This gives you significantly fewer parameters (or weights) for a given convolutional layer.
- You will not be able to take advantage of parameter sharing when the input images you are training on have a specific centred structure. You see this effect in faces when you always expect a specific feature to appear in a specific place (for centred faces). In this case you would probably not use parameter sharing. Parameter sharing is what gives CNNs invariance to translation/position, as well.

Equivariant Representation

- In the case of convolution, the particular form of parameter sharing causes the layer to have a property called equivariance to translation. To say a function is equivariant means that if the input changes, the output changes in the same way.

- Specifically, a function f(x) is equivariant to a function g if f(g(x) = g(f(x)). In the case of convolution, if we let g be any function that translates the input, that is, shifts it, then the convolution function is equivariant to g. For example, let *l* be a function giving image brightness at integer coordinates. Let g be a function mapping one image function to another image function, such that I′ = g(I) is the image function with I′ (x, y) = I (x – 1, y).
- This shifts every pixel of *l* one unit to the right. If you apply this transformation to *l* and then apply convolution, the result will be the same as if you applied convolution to *l*′ and then applied the transformation g to the output. When processing time-series data, this means that convolution produces a sort of timeline that shows when different features appear in the input. If you move an event later in time in the input, the exact same representation of it will appear in the output, just later.

Layer-specific Hyperparameters

- Following are the hyperparameters that dictate the spatial arrangement and size of the output volume from a convolutional layer.
  1. Filter (or kernel) size (field size)
  2. Output depth
  3. Stride
  4. Zero-padding
- You have already learnt about them earlier.

2.1.4 Pooling Layer

- Pooling layers, also known as down sampling, conducts dimensionality reduction, reducing the number of parameters in the input. Similar to the convolutional layer, the pooling operation sweeps a filter across the entire input, but the difference is that this filter does not have any weights. Instead, the kernel applies an aggregation function to the values within the receptive field, populating the output array. There are two main types of pooling.
  1. Max pooling: As the filter moves across the input, it selects the pixel with the maximum value to send to the output array. As an aside, this approach tends to be used more often compared to average pooling.
  2. Average pooling: As the filter moves across the input, it calculates the average value within the receptive field to send to the output array.
- While a lot of information is lost in the pooling layer, it also has a number of benefits to the CNN. They help to reduce complexity, improve efficiency, and limit risk of overfitting.

2.1.5 Fully-Connected Layer

- The name of the full-connected layer aptly describes itself. As mentioned earlier, the pixel values of the input image are not directly connected to the output layer in partially connected layers. However, in the fully-connected layer, each node in the output layer connects directly to a node in the previous layer.
- This layer performs the task of classification based on the features extracted through the previous layers and their different filters. While convolutional and pooling layers tend to use ReLu functions, FC layers usually leverage a softmax activation function to classify inputs appropriately, producing a probability from 0 to 1.

2.2 Variants of the Basic Convolution Function

- There are a few variants of basic convolution function as shown in Fig. 2.2.1.

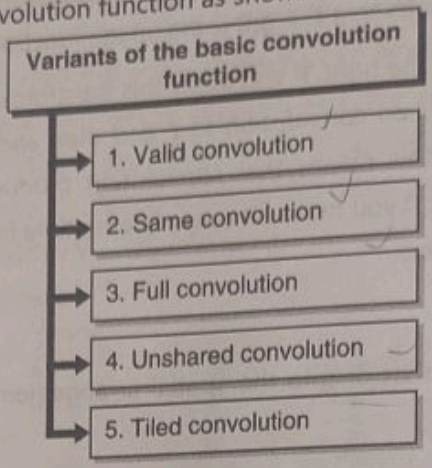


Fig. 2.2.1

Let's learn about them.

2.2.1 Valid Convolution

- First one is the extreme case in which no zero-padding is used whatsoever, and the convolution kernel is allowed to visit only positions where the entire kernel is contained entirely within the image. In this case, all pixels in the output are a function of the same number of pixels in the input, so the behaviour of an output pixel is somewhat more regular.
- However, the size of the output shrinks at each layer. If the input image has width m and the kernel has width k, then the output will be of width $m - k + 1$. The rate of this shrinkage can be dramatic if the kernels used are large. Since the shrinkage is greater than 0, it limits the number of convolutional layers that can be included in the network. As layers are added, the spatial dimension of the network will eventually drop to 1×1, at which point additional layers cannot meaningfully be considered convolutional.
- The Fig. 2.2.2 illustrates valid convolution.

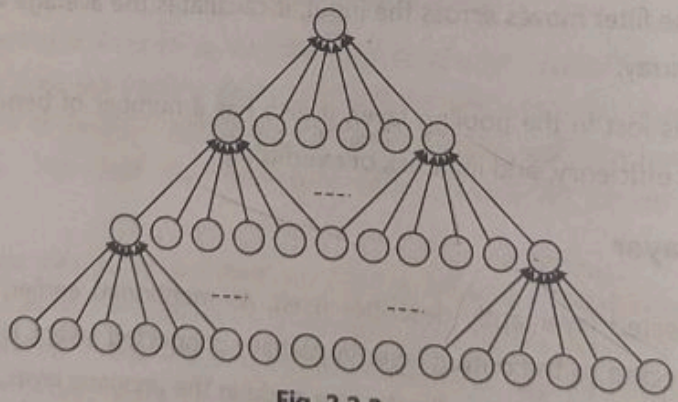


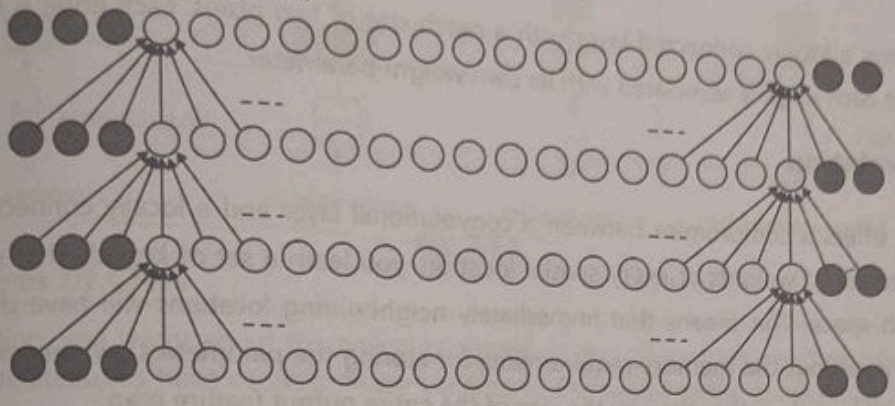
Fig. 2.2.2

- Note here that in this convolutional network, you do not use any implicit zero padding. This causes the representation to shrink by five pixels at each layer. Starting from an input of sixteen pixels, you are only able to have three convolutional layers, and the last layer does not ever move the kernel, so arguably only two of the layers are truly convolutional. The rate of shrinking can be mitigated by using smaller kernels, but smaller kernels are less expressive, and some shrinking is inevitable in this kind of architecture.

2.2.2 Same Convolution

Another special case of the zero-padding setting is when just enough zero-padding is added to keep the size of the output equal to the size of the input. In this case, the network can contain as many convolutional layers as the available hardware can support, since the operation of convolution does not modify the architectural possibilities available to the next layer. The input pixels near the border, however, influence fewer output pixels than the input pixels near the center. This can make the border pixels somewhat underrepresented in the model.

The Fig. 2.2.3 illustrates same convolution.



Valid
Same
Full
unshared
tiled

Fig. 2.2.3

Note here that by adding five implicit zeros to each layer, you prevent the representation from shrinking with depth. This allows you to make an arbitrarily deep convolutional network.

2.2.3 Full Convolution

In this case enough zeros are added for every pixel to be visited k times in each direction, resulting in an output image of width $m + k - 1$. In this case, the output pixels near the border are a function of fewer pixels than the output pixels near the center. This can make it difficult to learn a single kernel that performs well at all positions in the convolutional feature map. Usually the optimal amount of zero-padding (in terms of test set classification accuracy) lies somewhere between "valid" and "same" convolution.

2.2.4 Unshared Convolution

In some cases, you do not actually want to use convolution, but want to use locally connected layers instead. In this case, the adjacency matrix in the graph of your multilayer perceptron is the same, but every connection has its own weight, specified by a 6-D tensor W. The indices into W are as following.

- i is the output channel
- j is the output row
- k is the output column
- l is the input channel
- m is the row offset within the input
- n is the column offset within the input

The linear part of a locally connected layer is then given as following.

$$Z_{i,j,k} = \sum_{i,m,n} [V_{i,j+m-1,k+n-1} W(i,j,k,l,m,n)]$$

- The Fig. 2.2.4 illustrates unshared convolution.

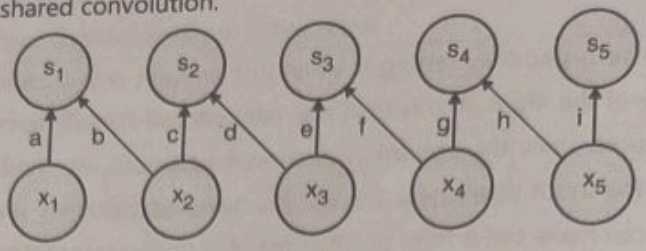


Fig. 2.2.4

- The Fig. 2.2.4 shows a locally connected layer with a patch size of two pixels. Each edge is labelled with a unique letter to show that each edge is associated with its own weight parameter.

2.2.5 Tiled Convolution

- Tiled convolution offers a compromise between a convolutional layer and a locally connected layer. Rather than learning a separate set of weights at every spatial location, you learn a set of kernels that you rotate through as you move through space. This means that immediately neighbouring locations will have different filters, as in a locally connected layer, but the memory requirements for storing the parameters will increase only by a factor of the size of this set of kernels, rather than by the size of the entire output feature map.
- The Fig. 2.2.5 illustrates tiled convolution.

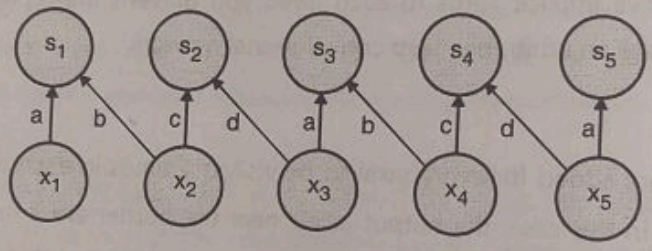


Fig. 2.2.5

- Note here that the tiled convolution has a set of different kernels. The diagram illustrates the case of t = 2. One of these kernels has edges labelled a and b, while the other has edges labelled c and d. Each time you move one pixel to the right in the output, you move on to using a different kernel. This means that, like the locally connected layer, neighbouring units in the output have different parameters. Unlike the locally connected layer, after you have gone through all t available kernels, you cycle back to the first kernel. If two output units are separated by a multiple of t steps, then they share parameters.

2.3 Properties of CNN

Let's understand some of the common terminologies and properties used in developing and training various CNNs.

1. **Wider network :** A wider network means that there are more feature maps (filters) in the convolutional layers.
2. **Deeper network :** A deeper network means that there are more convolutional layers.
3. **Higher resolution :** A CNN with higher resolution means that it processes input images with larger width and depth (spatial resolutions). That way the produced feature maps will have higher spatial dimensions.

The Fig. 2.3.1 illustrates the above three concepts.

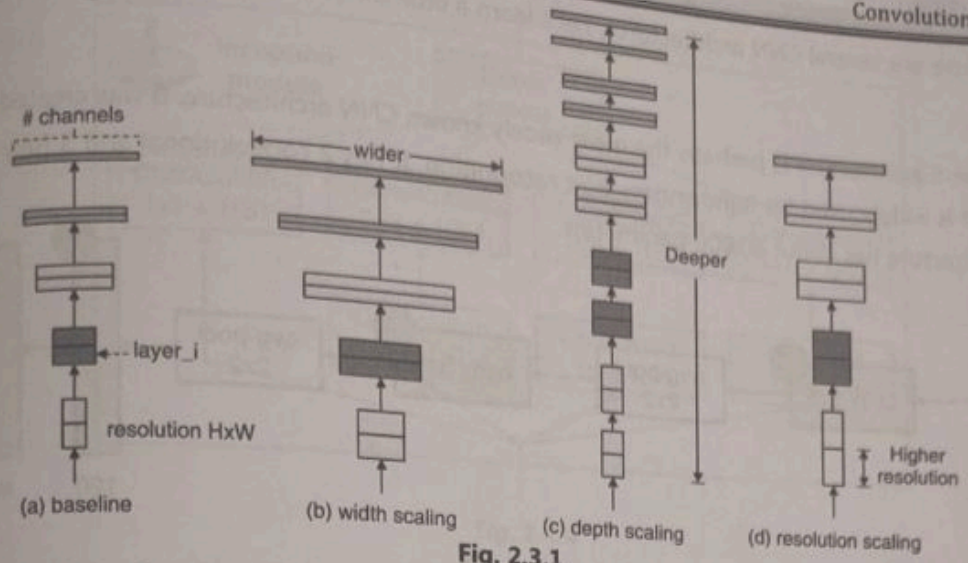


Fig. 2.3.1

2.4 Architectures of CNN

- Earlier in this section, you learnt about the typical components and general architecture of a CNN. However, over the years, CNN architectures have evolved. Many variants of the fundamental architecture have been developed, leading to amazing advances in the field.
- For example, the Fig. 2.4.1 provides a quick snapshot of various CNN architectures developed over time with respect to accuracy and computing requirements (measured in Giga Flops).

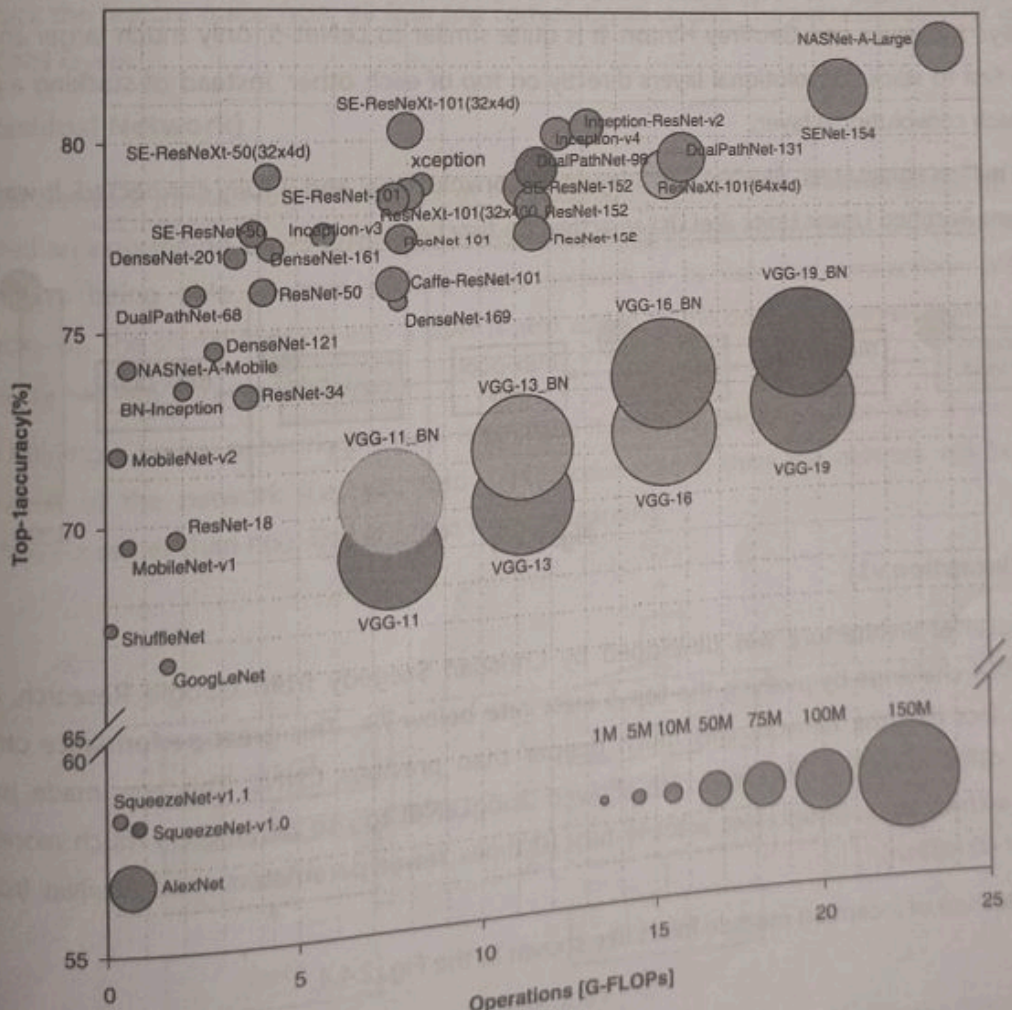


Fig. 2.4.1

As you see, there are several CNN architectures. Let's learn a little bit more about a few popular ones.

1. LeNet-5

- The LeNet-5 architecture is perhaps the most widely known CNN architecture. It was created by Yann LeCun in 1998 and is widely used for handwritten digit recognition. It has 2 convolutional and 3 fully-connected layers. This architecture has about 60,000 parameters.

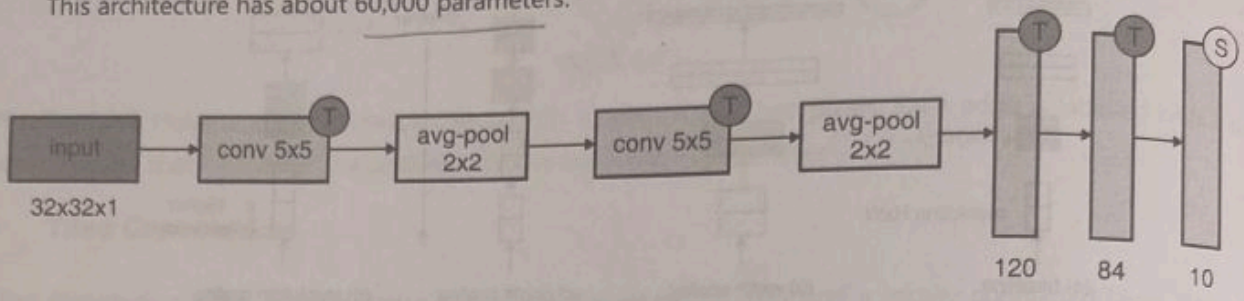


Fig. 2.4.2

- The ability to process higher resolution images requires larger and more convolutional layers. So, this technique is constrained by the availability of computing resources.

2. AlexNet

- The AlexNet CNN architecture won the 2012 ImageNet ILSVRC challenge by a large margin by achieving 17% top-5 error rate while the second best achieved only 26%! It was developed by Alex Krizhevsky (hence the name), Ilya Sutskever, and Geoffrey Hinton. It is quite similar to LeNet-5, only much larger and deeper, and it was the first to stack convolutional layers directly on top of each other, instead of stacking a pooling layer on top of each convolutional layer.
- With 60 million parameters, AlexNet has 8 layers, 5 convolutional and 3 fully connected. It was also the first to implement Rectified Linear Units (ReLUs) as activation functions.

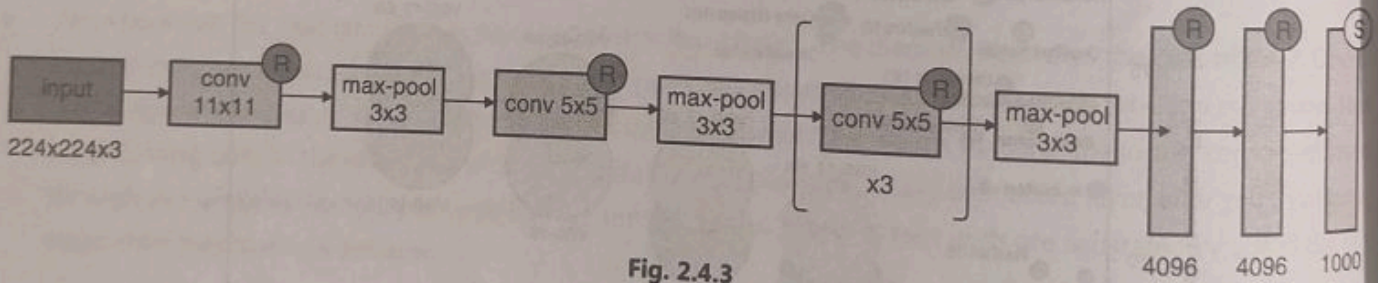


Fig. 2.4.3

3. GoogleNet (Inception v1)

- The GoogLeNet architecture was developed by Christian Szegedy from Google Research, and it won the ILSVRC 2014 challenge by pushing the top-5 error rate below 7%. This great performance came in large part from the fact that the network was much deeper than previous CNNs. This was made possible by sub-networks called inception modules which allowed GoogLeNet to use parameters much more efficiently than previous architectures. GoogLeNet actually has 10 times fewer parameters than AlexNet (roughly 6 million instead of 60 million).
- The architecture of inception module looks like shown in the Fig. 2.4.4.

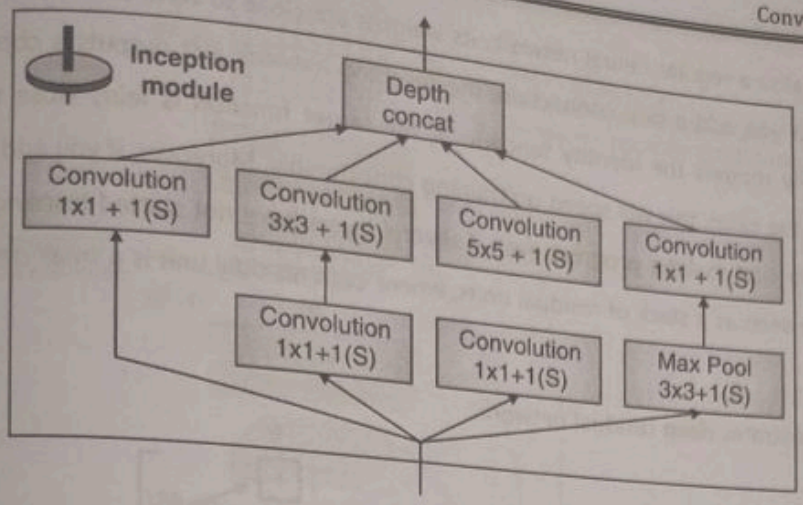


Fig. 2.4.4

- The notation "3 × 3 + 2(S)" means that the layer uses a 3 × 3 kernel, stride 2, and SAME padding. The input signal is first copied and fed to four different layers. All convolutional layers use the ReLU activation function. Note that the second set of convolutional layers uses different kernel sizes (1 × 1, 3 × 3, and 5 × 5), allowing them to capture patterns at different scales. Also note that every single layer uses a stride of 1 and SAME padding (even the max pooling layer), so their outputs all have the same height and width as their inputs. This makes it possible to concatenate all the outputs along the depth dimension in the final depth concat layer (i.e., stack the feature maps from all four top convolutional layers). The overall GoogleNet architecture has 22 deep CNN layers.

4. ResNet (Residual Network)

- Residual Network (or ResNet), the winner of the ILSVRC 2015 challenge, was developed by Kaiming He which delivered an astounding top-5 error rate under 3.6%, using an extremely deep CNN composed of 152 layers. The key to being able to train such a deep network is to use skip connections (also called shortcut connections). The signal feeding into a layer is also added to the output of a layer located a bit higher up the stack. Let's see why this is useful.
- When training a neural network, the goal is to make it model a target function h(x). If you add the input x to the output of the network (i.e., you add a skip connection), then the network will be forced to model f(x) = h(x) – x rather than h(x). This is called residual learning.

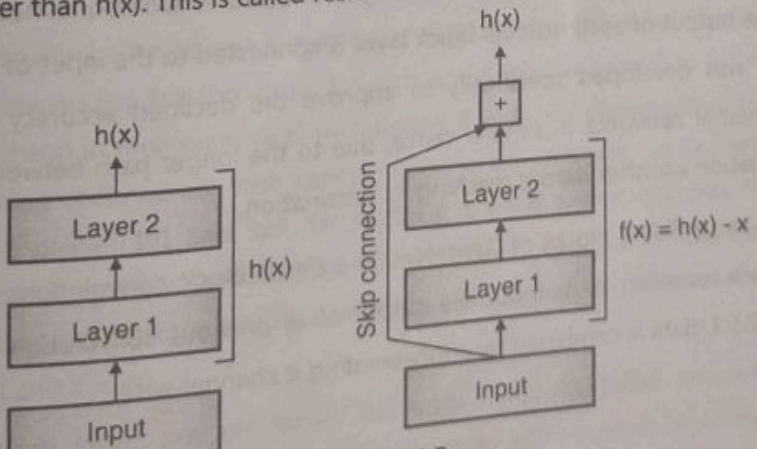


Fig. 2.4.5

- When you initialise a regular neural network, its weights are close to zero, so the network just outputs values close to zero. If you add a skip connection, the resulting network just outputs a copy of its inputs, in other words, it initially models the identity function. If the target function is fairly close to the identity function (which is often the case), this will speed up training considerably. Moreover, if you add many skip connections, the network can start making progress even if several layers have not started learning yet. The deep residual network can be seen as a stack of residual units, where each residual unit is a small neural network with a skip connection.
- The Fig. 2.4.6 illustrates deep residual network.

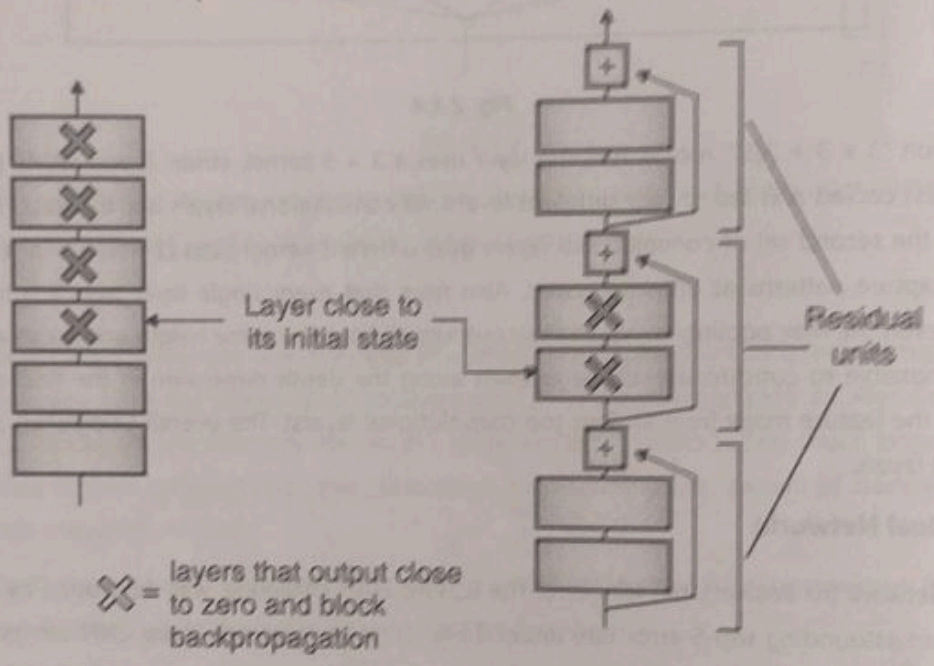


Fig. 2.4.6

2.5 DenseNet

- The DenseNet model introduced the concept of a densely connected convolutional network. The design is based on the principle that the output of each residual block layer is connected to the input of every subsequent residual block layer. DenseNet was developed specifically to improve the declined accuracy caused by the vanishing gradient in high-level neural networks. In simpler terms, due to the longer path between the input layer and the output layer, the information vanishes before reaching its destination.
- Dense blocks are the basic building blocks of DenseNet. In a dense block, convolutions are grouped in pairs, with each pair of convolutions receiving as its input the output of all previous convolution pairs. For example, in the dense block in the Fig. 2.5.1, data is combined by concatenating it channel-wise.

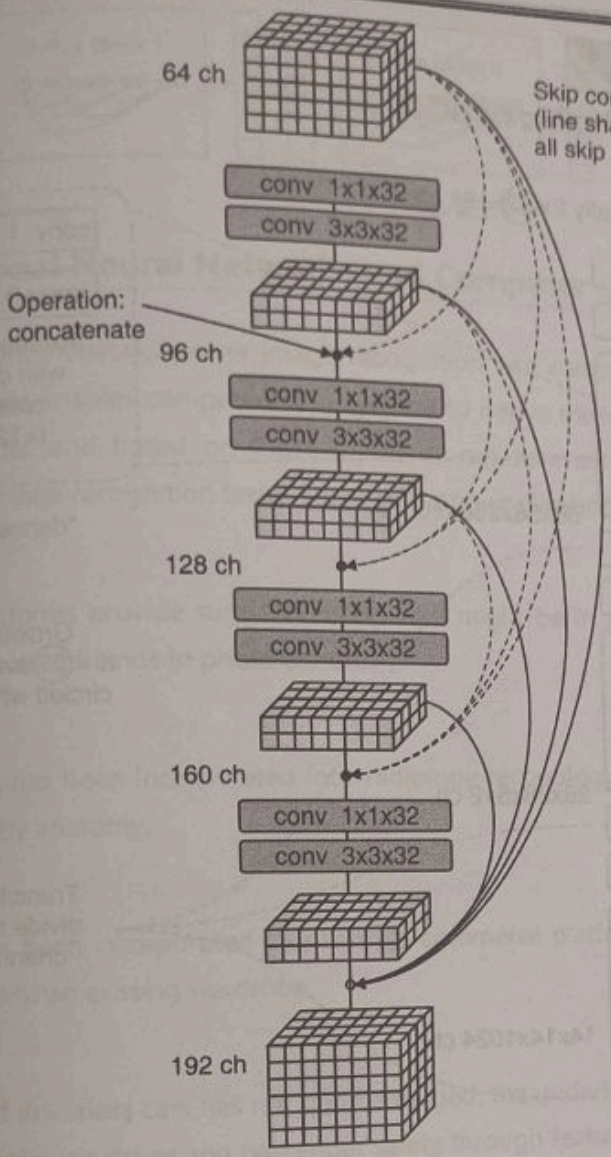


Fig. 2.5.1

- All convolutions are ReLU-activated and use batch normalisation. Channel-wise concatenation only works if the height and width dimensions of the data are the same, so convolutions in a dense block are all of stride 1 and do not change these dimensions. Pooling layers are inserted between dense blocks.
- Intuitively, you would think that concatenating all previously seen outputs would lead to an explosive growth of the number of channels and parameters, but that is not in fact the case. DenseNet is surprisingly economical in terms of learnable parameters. The reason is that every concatenated block, which might have a relatively large number of channels, is always fed first through a 1x1 convolution that reduces it to a small number of channels, K. 1 x 1 convolutions are cheap in their number of parameters. A 3x3 convolution with the same number of channels (K) follows. The K resulting channels are then concatenated to the collection of all previously generated outputs. Each step, which uses a pair of 1x1 and 3x3 convolutions, adds exactly K channels to the data. Therefore, the number of channels grows only linearly with the number of convolutional steps in the dense block. The growth rate K is a constant throughout the network, and DenseNet has been shown to perform well with fairly low values of K, such as between K = 12 and K = 40.
- Dense blocks and pooling layers are interleaved to create a full DenseNet network. The Fig. 2.5.2 shows a DenseNet121 with 121 layers, but the architecture is configurable and can easily scale beyond 200 layers.

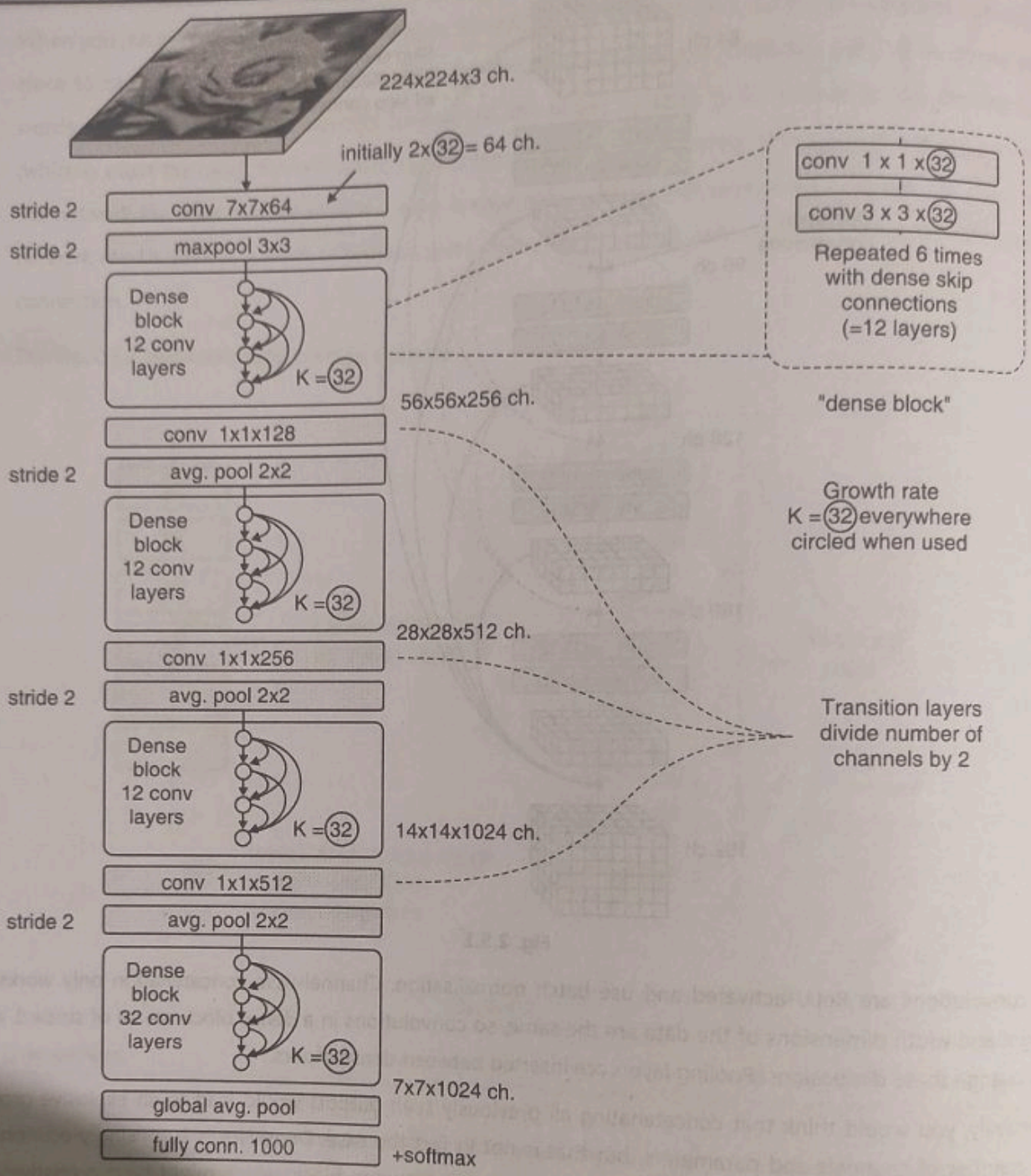


Fig. 2.5.2

- The use of shallow convolutional layers (K = 32 for example) is a characteristic feature of DenseNet. In previous architectures, convolutions with over one thousand filters were not rare. DenseNet can afford to use shallow convolutions because each convolutional layer sees all previously computed features. In other architectures, the data is transformed at each layer and the network must do active work to preserve a channel of data as-is, if that is the right thing to do. It must use some of its filter parameters to create an identity function, which is wasteful. DenseNet is built to allow feature reuse and therefore requires far fewer filters per convolutional layer.
- Fig. 2.5.3 is a simple figure illustrating a deep DenseNet network with three dense blocks. The layers between two adjacent blocks are referred to as transition layers and change feature-map sizes via convolution and pooling.

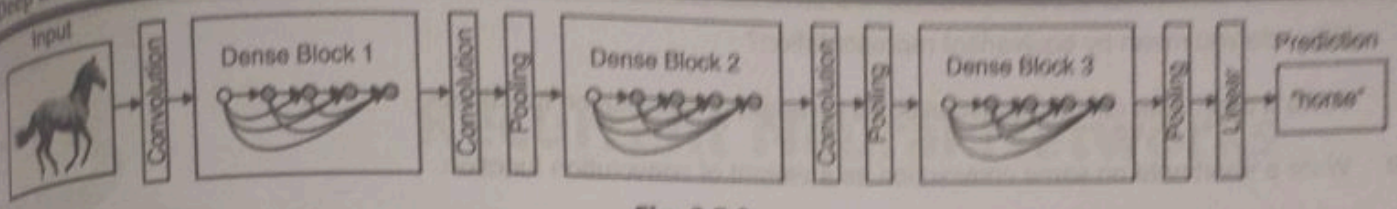


Fig. 2.5.3

2.6 Convolutional Neural Networks and Computer Vision (Applications of CNN)

Convolutional neural networks power image recognition and computer vision tasks. Computer vision is a field of artificial intelligence that enables computers and systems to derive meaningful information from digital images, videos and other visual inputs, and based on those inputs, it can take action. This ability to provide recommendations distinguishes it from image recognition tasks. Some common applications of this computer vision is as following.

- **Marketing**

  Social media platforms provide suggestions on who might be in photograph that has been posted on a profile, making it easier to tag friends in photo albums.

- **Healthcare**

  Computer vision has been incorporated into radiology technology, enabling doctors to better identify cancerous tumours in healthy anatomy.

- **Retail**

  Visual search has been incorporated into some e-commerce platforms, allowing brands to recommend items that would complement an existing wardrobe.

- **Automotive**

  While the age of driverless cars has not quite emerged, the underlying technology has started to make its way into automobiles, improving driver and passenger safety through features like lane line detection.

Review Questions

Here are a few review questions to help you gauge your understanding of this chapter. Try to attempt these questions and ensure that you can recall the points mentioned in the chapter.

Convolution Neural Network (CNN)

Q.1 Explain Convolution Neural Networks. **(6 Marks)**

Q.2 How do convolution neural networks work? **(6 Marks)**

Q.3 Explain the components of a convolution neural networks. **(6 Marks)**

Q.4 In a CNN, how are number of filters, stride, and padding used? **(4 Marks)**

Q.5 Write a short note on filters with respect to a CNN. **(4 Marks)**

Q.6 Write a short note on activation maps with respect to a CNN. **(4 Marks)**

Q.7 Describe parameter sharing with respect to a CNN. **(4 Marks)**