PadhAI: From Convolution Operation to Neural Network

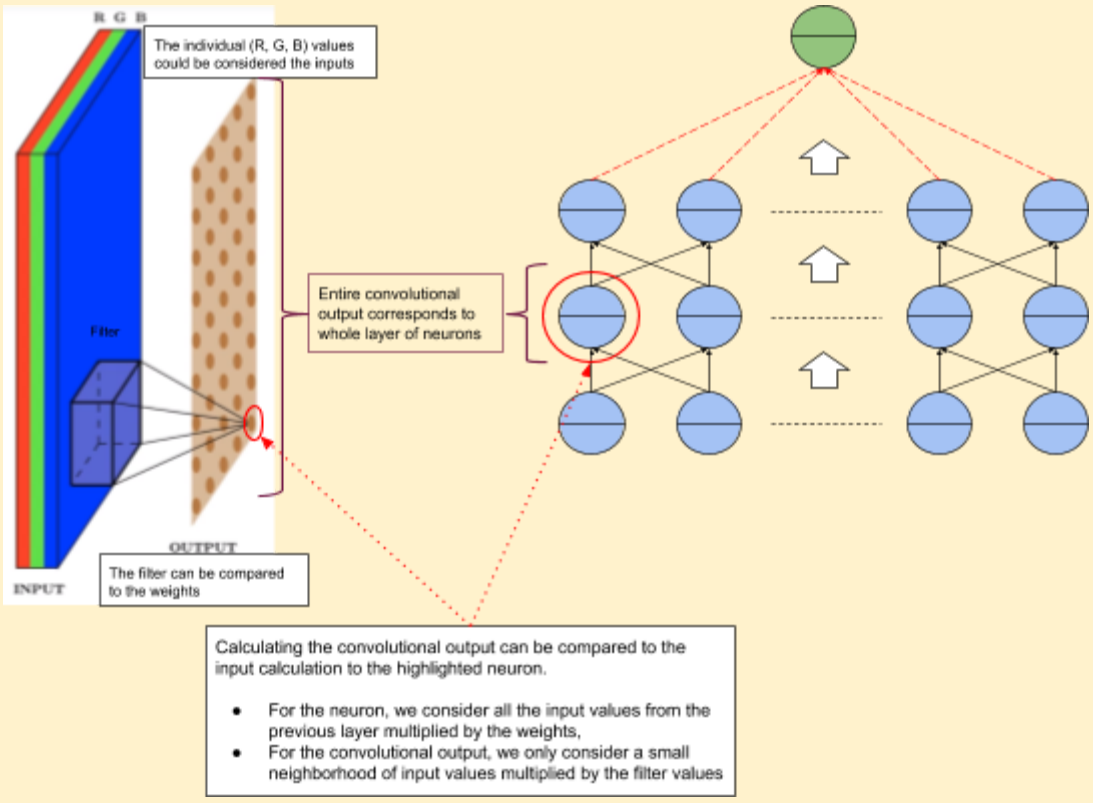
One Fourth Labs

The convolution operation and Neural Networks

Part 1

What is the relation between the convolution operation and neural networks?

1. The following diagram illustrates the similarities between the convolutional operation and DNNs



2. As we can see from the diagram, both the highlighted output neuron and the highlighted convolutional output are essentially weighted sums of the inputs provided to them
3. Let's look at a direct comparison

	Neural Network	**Convolution Operation on image**
Input	Numerical input values.	The RGB values for each pixel in the image
Output	Neuron which takes weighted sum of inputs as its input	Pixel which takes the RGB values transformed with a filter
Neighborhood	All inputs from the previous layer contribute to the output calculation	Only a localised neighborhood of inputs is considered for each output pixel.
	The entire convoluted output image corresponds to a whole layer of neurons. With **multiple filters**, **multiple convoluted outputs each correspond to separate layers of neurons**	

PadhAI: From Convolution Operation to Neural Network

One Fourth Labs

The convolution operation and neural networks

Part 2

How did we arrive here?

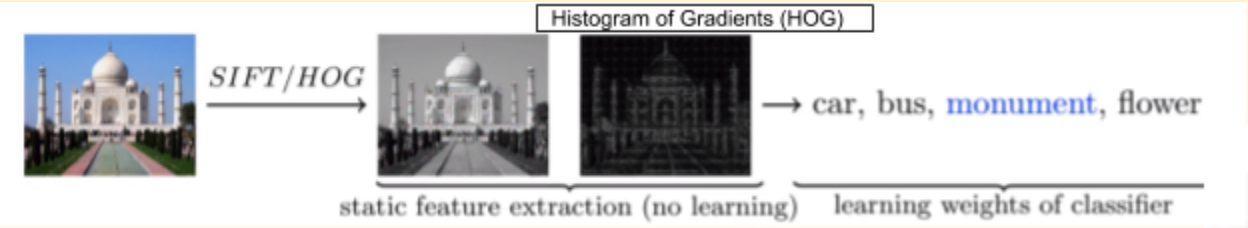
1. Let's look at the image classification task as it would've been performed with Machine Learning.

   a. Here, we flatten a 30x30x3 image into a 2700 raw pixels and feed them as input to a classifier such as a Support Vector Machine or Naive Bayes etc.
   b. There isn't much intelligence applied on the input side, we just pass the raw pixel data.

2. Now, let's look at the image classification done with some input preprocessing

   a. Here, we realise that there are certain aspects of the image (outlines/edges) that are much more critical to the classification task than other aspects
   b. So we perform feature engineering, whereby we apply some transformation to the input pixels before passing them into the classifier.

3. Let's look at the use of feature engineering with a 0 Hidden Layer NN to classify the images



   a. Here, using a deterministic algorithm like HOG or SIFT, we get a better representation of the input by cancelling out useless information.
   b. We now use these new inputs as features for our Neural Network and learn the weights for the classifier.
   c. However, in step (a), the transformation performed on the input image was static, without any learning per se, making it a hand-crafted set of features. The only learning that happens is in the classifier.

4. However, in a deep Neural Network, the input features are not directly fed to the classification/output layer, instead they are passed through hidden/representation layers, where they are distilled down to more relevant features, before being passed into the classification layer. This is why Deep Learning is also called Deep Representation Learning.
5. In the above case, we allow the DNN to learn the representation weights and apply it to the features in steps, before finally passing it onto the output layer.

PadhAI: From Convolution Operation to Neural Network

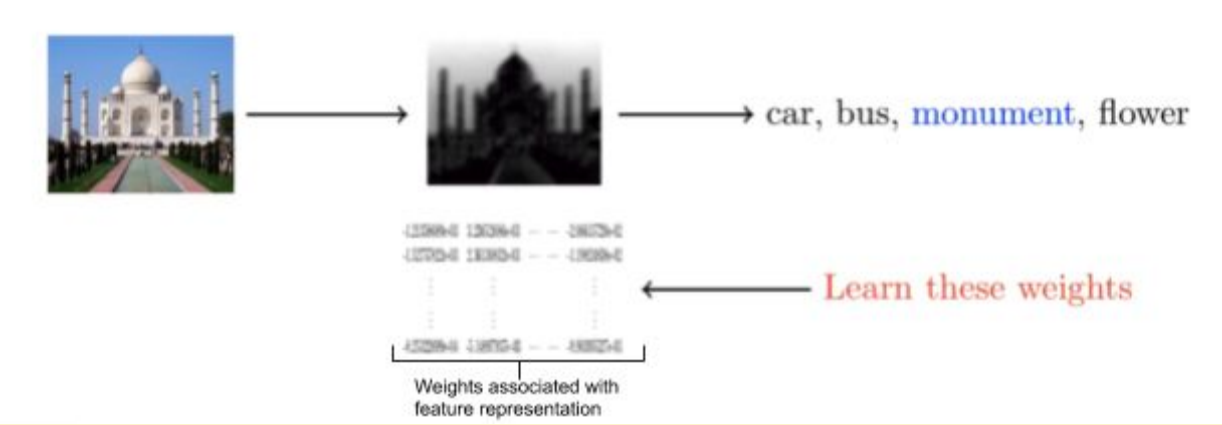
One Fourth Labs

The convolution operation and neural networks

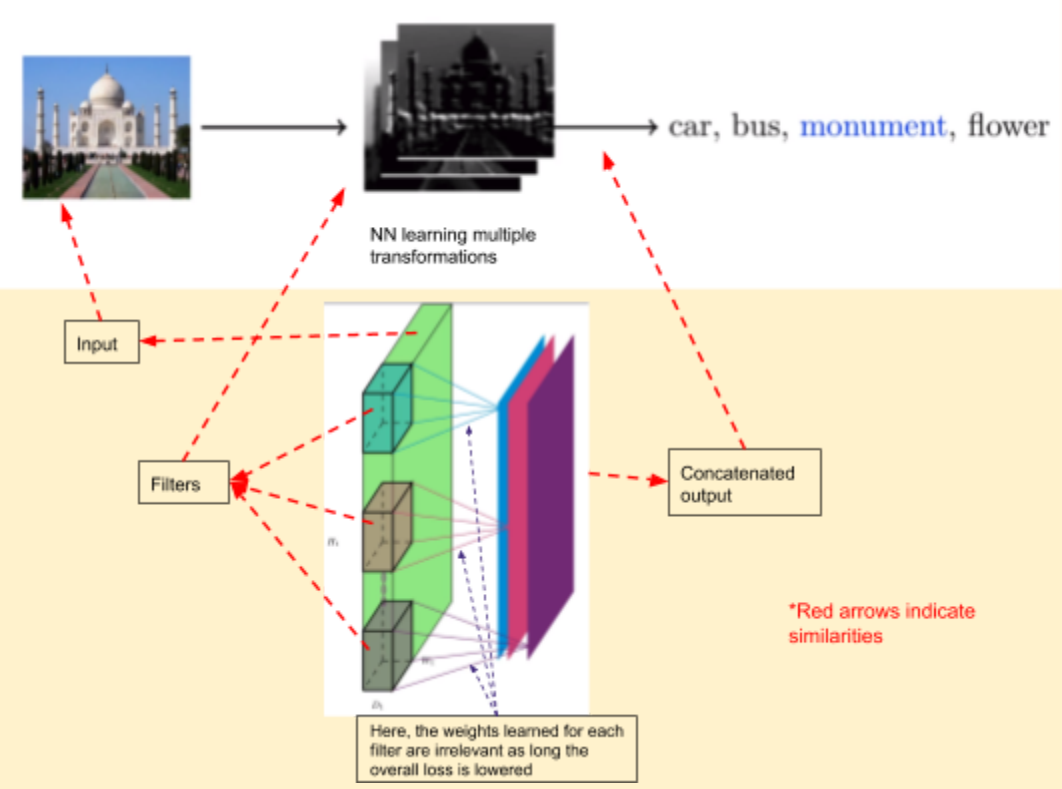
Part 3

Why not let the network learn the feature representation also?

1. So far, we have intuitively determined what kind of transformation to apply to the input image before passing it through the classifier. For eg: we saw that performing edge detection yielded higher accuracy.
2. Now, another thing to think about is, why must we leave the choice of transformation to our human intuition? Why can't we let the DNN learn for itself the best transformation to apply?



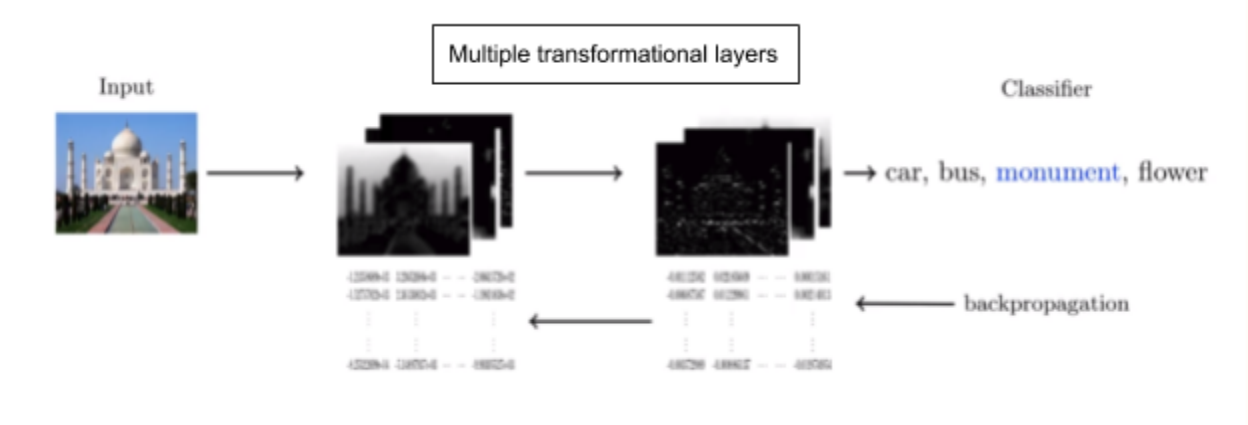
3. **Why not let the network learn the multiple feature representation**?
4. Another point to consider is that we might not necessarily benefit from only one transformation, so letting the DNN learn the number of transformations to perform is also very useful.



a. Here, we are not concerned with the type of transformation that occurs, so long as the overall loss/error is reduced. In the above image, we cannot say that the transformations are anything

definitive like edge-detection or blurring but still choose to accept them because they lower the loss.

5. **Why not let the network learn the multiple layers of feature representation**?
6. It stands to reason that we can consider several transformational layers like the one seen in the previous example.



   a. The above process is quite similar to what we have learned in the DNN. There are 3 sets of weights, one for each of the two representational layers and one for the classification layer.
   b. These weights are calculated using backpropagation
   c. The only difference between a DNN and what we're looking at now (CNN) is that for a CNN we only consider a small localised neighborhood of inputs when calculating the output, instead of the entire input layer as in the case of DNNs.
7. In a nutshell, for CNNs, instead of learning the final classifier weights directly, we should also learn to transform the input into a suitable representation through multiple layers of representations and learn the kernel weights for all of those representations instead of using hand-crafted kernels.

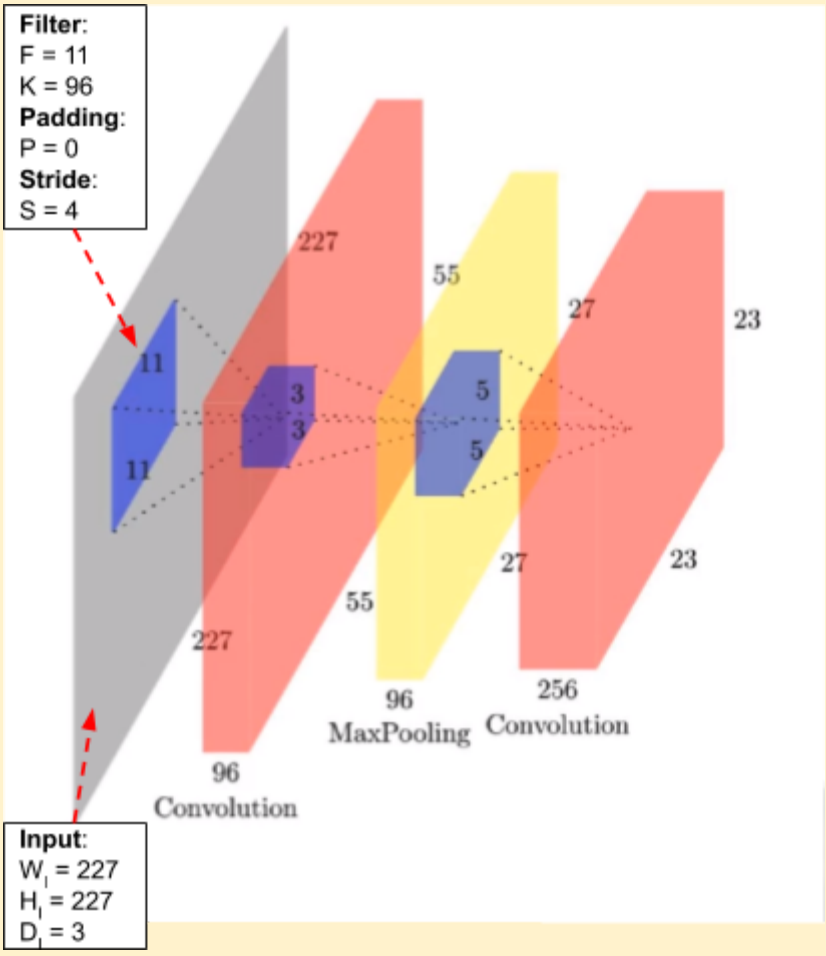
PadhAI: From Convolution Operation to Neural Network

One Fourth Labs

Understanding the input/output dimensions

Let's look at the input and output dimensions for a Convolutional Operation

1. As we have seen before, a CNN can be compared to a normal Neural Network, the difference being that CNNs take the RGB pixel values as inputs and output calculation is done with a localised neighborhood of inputs.
2. Consider the following diagram of a CNN. Let us dissect the first convolutional operation in depth.



   a. From the above diagram, we are analysing the convolutional operation on the grey input layer.
   b. The input dimensions are as follows
      i. $W_I = 227$
      ii. $H_I = 227$
      iii. $D_I = 3$
   c. The filter is of scale F = 11, i.e 11x11x3, where 3 is the same depth as D_I
   d. We apply 96 Filter operations, so therefore K = 96
   e. We do not take any padding (P=0) and we choose a stride length of S = 4
   f. Thus, going by the above information, the output volume can be calculated as follows
      i. $W_O = \frac{W_I - F + 2P}{S} + 1 = 55$
      ii. $H_O = \frac{H_I - F + 2P}{S} + 1 = 55$
      iii. $D_O = K = 96$
   g. Thus, the output of the convolutional layer has the dimensions 55x55x96

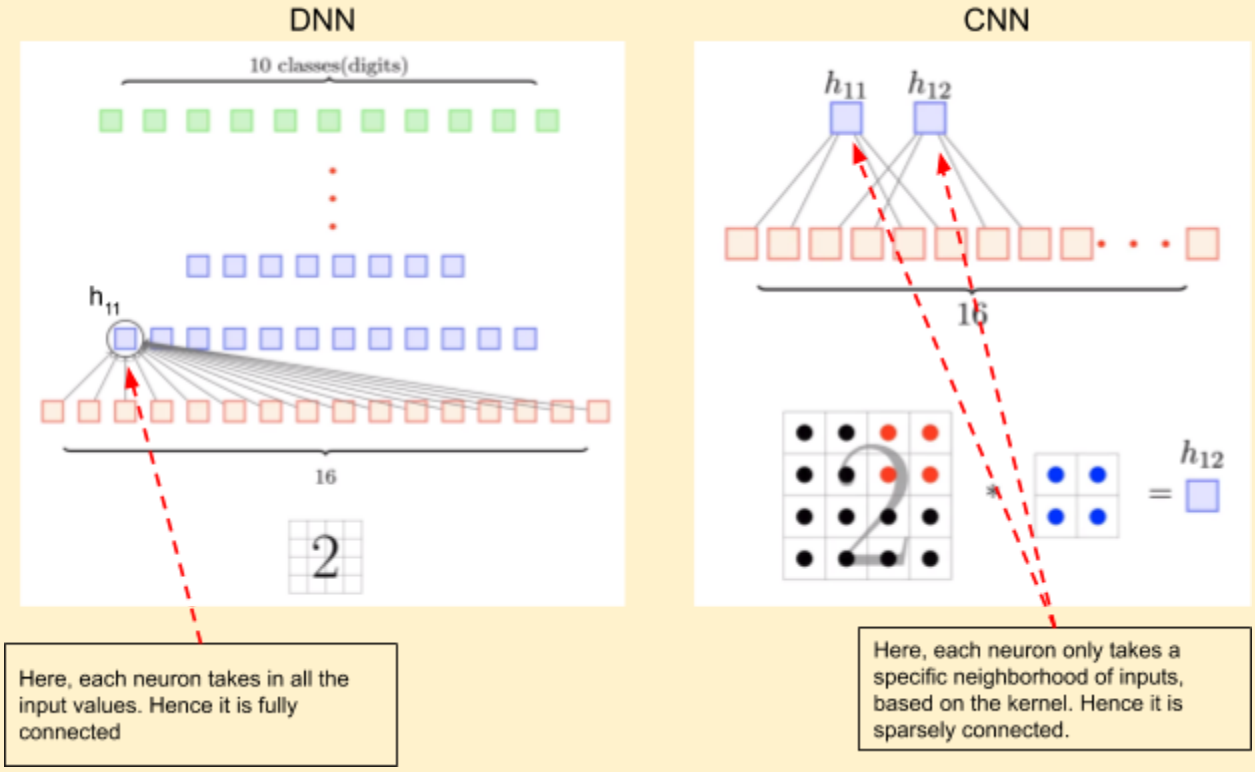
PadhAI: From Convolution Operation to Neural Network

One Fourth Labs

Sparse connectivity and Weight Sharing

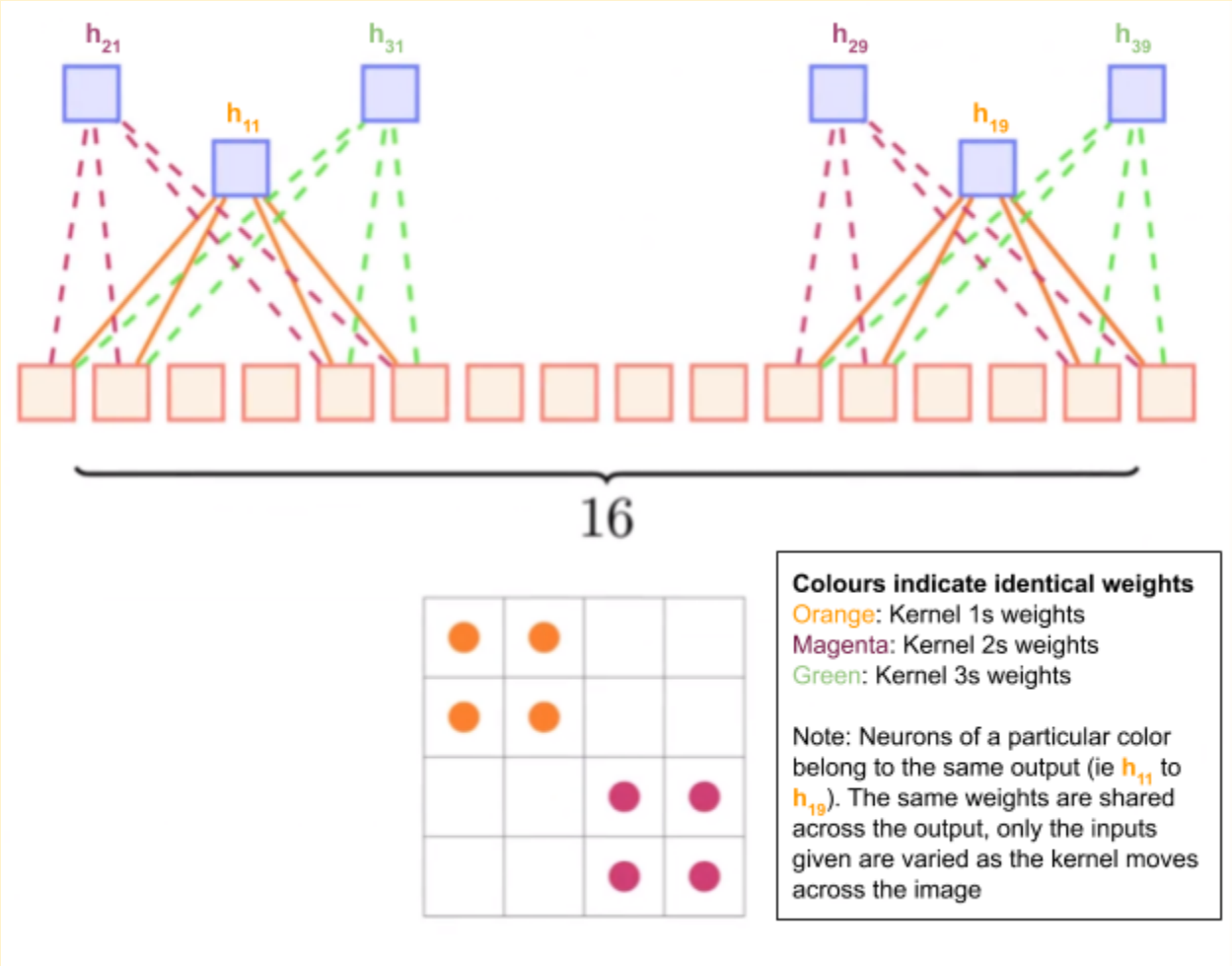
Let's look at the differences between Fully-connected DNNs and CNNs

1. We already have an intuition of how CNNs are different from DNNs, the key point being that in CNNs we take a weighted aggregation of a neighborhood of inputs instead of the all the inputs like in DNNs. Also, it is better to visualise CNNs in terms of matrices than by flat vectors as in the case of FFNs
2. More formally, two of the main aspects of CNNs are **Sparse Connectivity** & **Weight Sharing.**
3. Let us look at the **connectivity difference** between DNNs and CNNs



a. The above diagram is meant to illustrate the difference between DNNs and CNNs with regards to the input-output relationship
b. In CNNs, as the kernel moves over the input image in strides specified by the stride length S, it passes over the highlighted regions as shown in the figure. The highlighted regions vary in size with the Kernel size F, and they correspond with the input regions that are used to calculate the output for that particular neuron.
c. For the CNN, neuron h_{11} corresponds to inputs 1, 2, 5 & 6, while neuron h_{12} corresponds to inputs 3, 4, 7 & 8.
d. In contrast, in the DNN, neuron h_{11} and every other neuron corresponds to all inputs 1 to 16.
e. This is why DNNs are called fully connected and CNNs are said the be sparsely connected

4. Now, let us look at the concept of **weight sharing**, as illustrated in the figure.



a. Here, we can see how weights are shared across an an entire layer.
b. In CNNs, **weights are nothing but the kernel,** and the **kernel remains constant as we pass over the entire image**, creating different output values based on the neighborhood of inputs
c. One complete pass of the kernel over the image constitutes one Convolutional output.
d. As we have seen earlier, it is possible to have multiple convolutional operations by using multiple kernels over the input.
e. For each of these convoluted outputs, the kernel is constant, thereby **the weights get shared for all the neurons in that output area**. Only the inputs vary.
f. By combining all of these convolutional outputs, we get one convolutional layer
g. Instead of treating this Convolutional Layer as a flat vector, we treat it as a volume, whose depth is given by the number of kernels used to process the input.
h. By this practice, we are effectively reducing the number of parameters yet still retaining model complexity, thereby overcoming one of the shortcomings of DNNs.

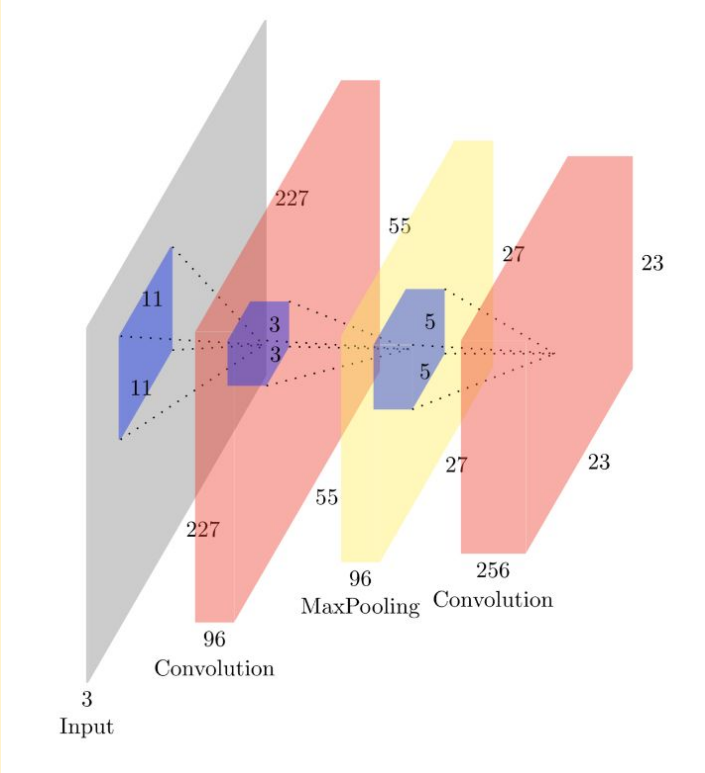
PadhAI: From Convolution Operation to Neural Network

One Fourth Labs

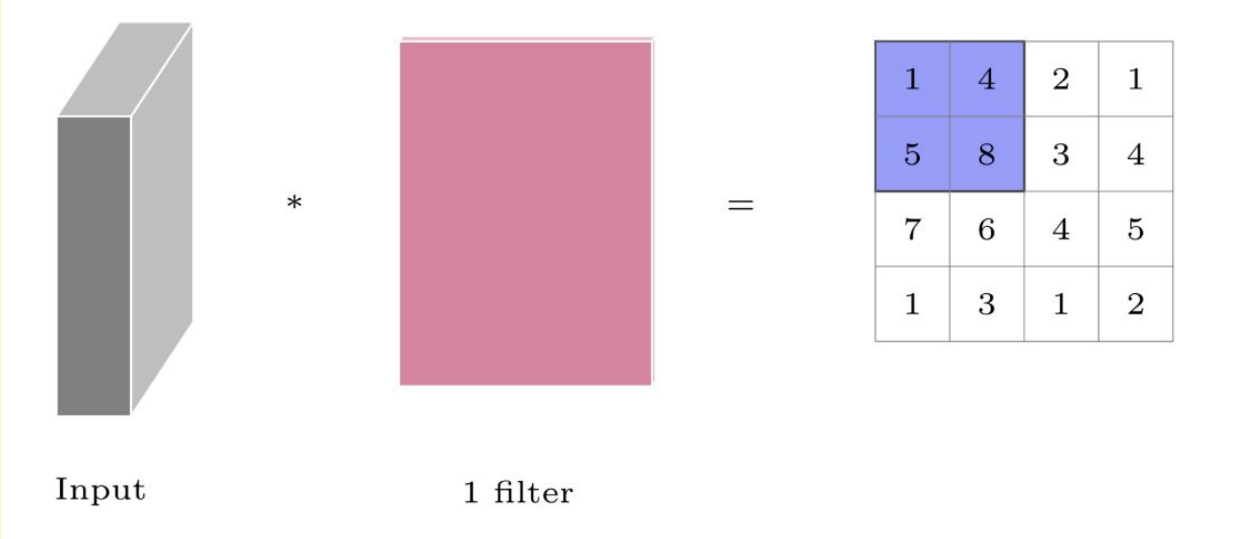
Max Pooling and Non-Linearities

What is the max pooling operation?

1. Let us look at a diagram of a CNN to better understand what max pooling does.

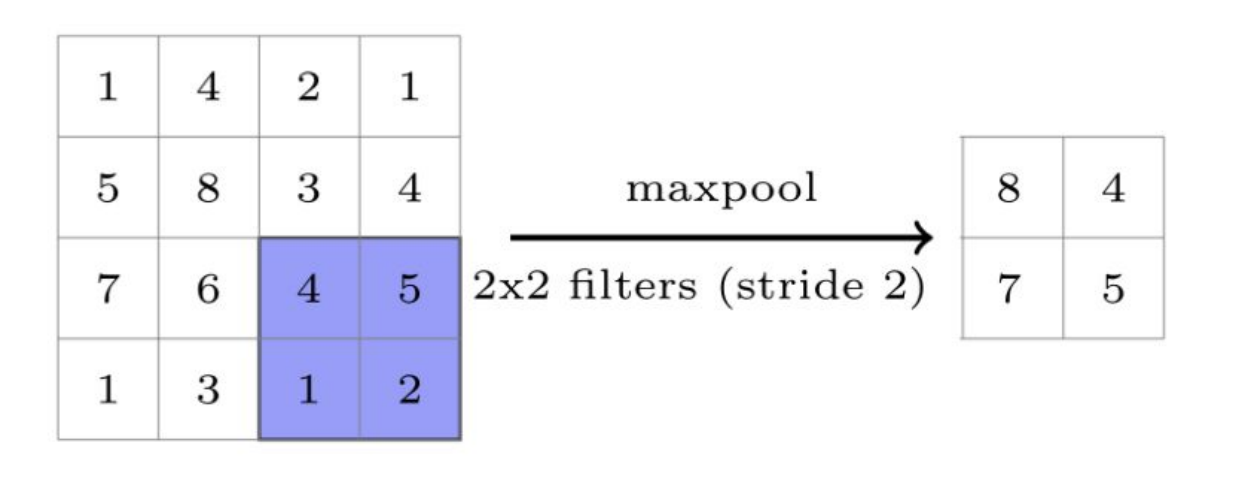


2. Here, as we have discussed earlier, the first operation performed is a convolutional transformation using a filter.



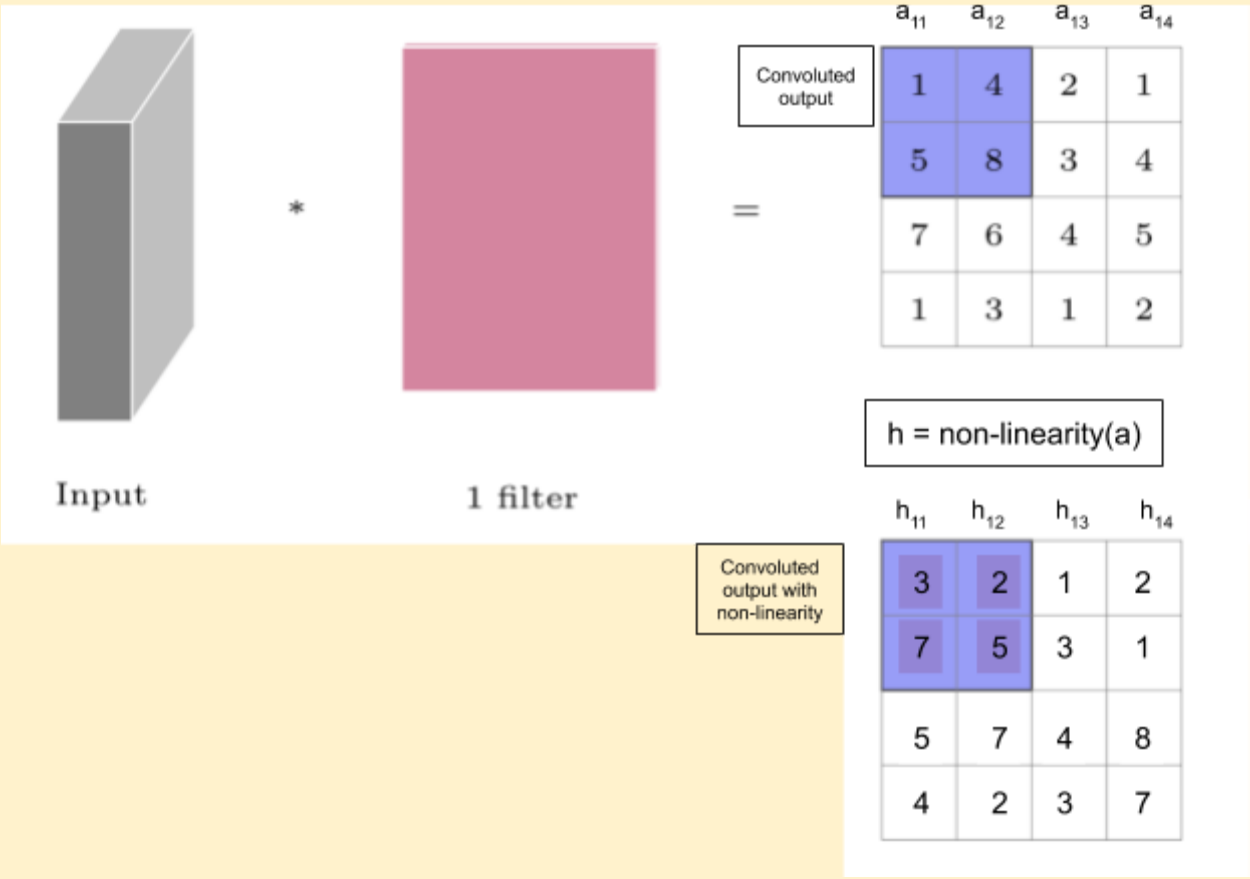
   a. Here, by passing the filter over an image (with or without padding), we get a transformed matrix of values

3. Now, we perform max-pooling over the convoluted input to select the max-value from each position of the kernel, as specified by stride length.



   a. Here, we select a stride length of 2 and a 2x2 filter, meaning the 4x4 convoluted output is split into 4 quadrants.
   b. The max value of each of these quadrants is taken and a 2x2 matrix is generated.
   c. Max pooling is done to select the most prominent or salient point within a neighborhood. It is also known as subsampling, as we are sampling just a single value from a region.
   d. Similar to Max pooling, average pooling is also done sometimes and it's carried out by taking the average value in a sampled neighborhood.
   e. The idea behind Max Pooling is to condense the convolutional input into a smaller size, thereby making it easier to manage.

4. Another point to consider is the application of nonlinearities to the convoluted output.



a. Here, we can consider the convoluted output to be similar to the pre-activation layer of a neuron.
b. By applying a non-linear transformation like sigmoid, tanh, ReLU etc, we are effectively transforming the convoluted output
c. The resultant transformed matrix is then passed through the subsequent stages in the CNN, such as Max Pooling etc.
d. Thus, we are creating a non-linear relationship between the input and the output, thereby allowing us to approximate more complex functions.
e. Yet at the same time, we avoid over-complexity by reducing the number of parameters by weight sharing and condensing the convoluted output using max pooling.

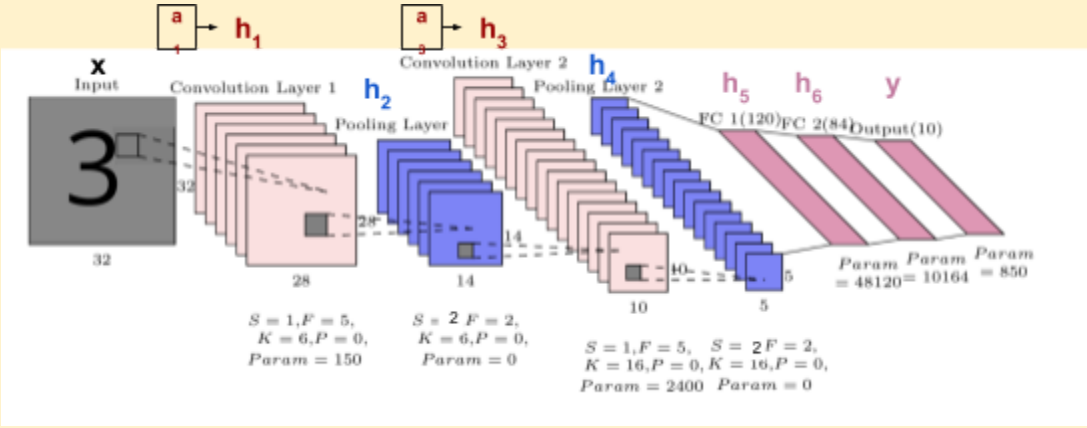
PadhAI: From Convolution Operation to Neural Network

One Fourth Labs

Our First Convolutional Neural Network (CNN)

How to use a convolutional neural network for image classification?

1. The following diagram illustrates the configuration and working of a Convolutional Neural Network. It follows the **LeNet** architecture, created by Yann LeCun



2. Let us sequentially break down the various layers in this CNN
3. **Input**:
   a. The image takes 32x32 pixel inputs.
   b. There is no depth component because the images are in black & white.
4. **Convolution Layer 1**:
   a. Here, the filter size F = 5, and the central cell is the pixel of interest
   b. Stride length S = 1
   c. We use a total of 6 filters, i.e. K = 6
   d. No padding is used, i.e. P = 0
   e. Each of the filters generate 28x28 output (calculated using W_O, H_O formula).
   f. Our hidden representation at this layer is $\mathbf{a_1}$ **= 28x28x6** (D_O = K).
   g. Non-linearity like tanh or ReLU(preferred for CNN) is applied to $\mathbf{a_1}$ making it $\mathbf{h_1}$
   h. If we were to proceed as a Fully Connected Network, we would have an extremely large number of parameters (32x32 x 28x28x6 = 4,816,896 parameters).
   i. However in this sparsely connected network, each of the 6 filters is of size 5x5x1. So the number of parameters would be much more manageable (6 x 5x5x1 = 150 parameters).
   j. This is significantly smaller than in a fully connected network, thereby reducing the chance of overfitting.
   k. Here, the values F, S, K, P etc are all counted as hyperparameters.
5. **Max Pooling Layer 1**:
   a. The hyperparameters are as follows
   b. Filter size F = 2
   c. Stride length S = 2
   d. No. of filters K = 6
   e. Padding P = 0
   f. Here, from a 2x2 filter, we select only 1 value. Therefore for a stride of 2, the output dimensions are half of the input(h_1) dimensions, i.e 14x14
   g. We apply the max pooling independently to all 6 of the h_1 layers, giving us $\mathbf{h_2}$ **= 14x14x6**
   h. No parameters for this layer as we are simply choosing the largest value in the filter and not applying any weights to it.

6. **Convolutional Layer 2**:
   a. The hyperparameters are as follows
   b. Filter size F = 5
   c. Stride length S = 1
   d. No. of filters K = 16
   e. Padding P = 0
   f. Thus, the filter dimensions are 5x5x6
   g. Here, 16 filters are applied to the input h_2, thereby giving us an output depth of $D_O = 16$
   h. Calculating W_O and H_O using the formula, we get 10x10
   i. Our hidden representation at this layer is $\mathbf{a_3}$ **= 10x10x16**
   j. Non-linearity like tanh or ReLU(preferred for CNN) is applied to $\mathbf{a_3}$ making it $\mathbf{h_3}$
   k. The number of parameters for the filters(16 x 5x5x6) is 2400 parameters
   l. This is much smaller than what we would have had in a fully connected network
7. **Max Pooling Layer 2**:
   a. The hyperparameters are as follows
   b. Filter size F = 2
   c. Stride length S = 1
   d. No. of filters K = 16
   e. Padding P = 0
   f. Here, from a 2x2 filter, we select only 1 value. Therefore for a stride of 2, the output dimensions are half of the input(h_3) dimensions, i.e 14x14
   g. We apply the max pooling independently to all 16 of the h_1 layers, giving us $\mathbf{h_4}$ **= 5x5x16**
   h. No params for this layer as we are simply choosing the largest value in the filter.
8. **Fully connected layer 1**:
   a. Number of neurons: 120
   b. Input is $\mathbf{h_4}$ flattened, i.e. 5x5x16 = 400
   c. No. of parameters in $\mathbf{h_5}$ = 120x400 + 120-bias = 48120 parameters
9. **Fully connected layer 2**:
   a. Number of neurons: 84
   b. Input is number of neurons in $\mathbf{h_5}$ = 120
   c. No. of parameters in $\mathbf{h_6}$ = 84x120 + 84-bias = 10164 parameters
10. **Output layer**:
    a. Number of neurons: 10
    b. Input is number of neurons in $\mathbf{h_6}$ = 84
    c. No. of parameters in **y** = 10x84 + 10-bias = 850 parameters
11. Overall, this combination of Convolutional and fully-connected layers is much more efficient than an entirely fully connected network. It has a significantly lower number of parameters but still is able to estimate functions of very high complexity.

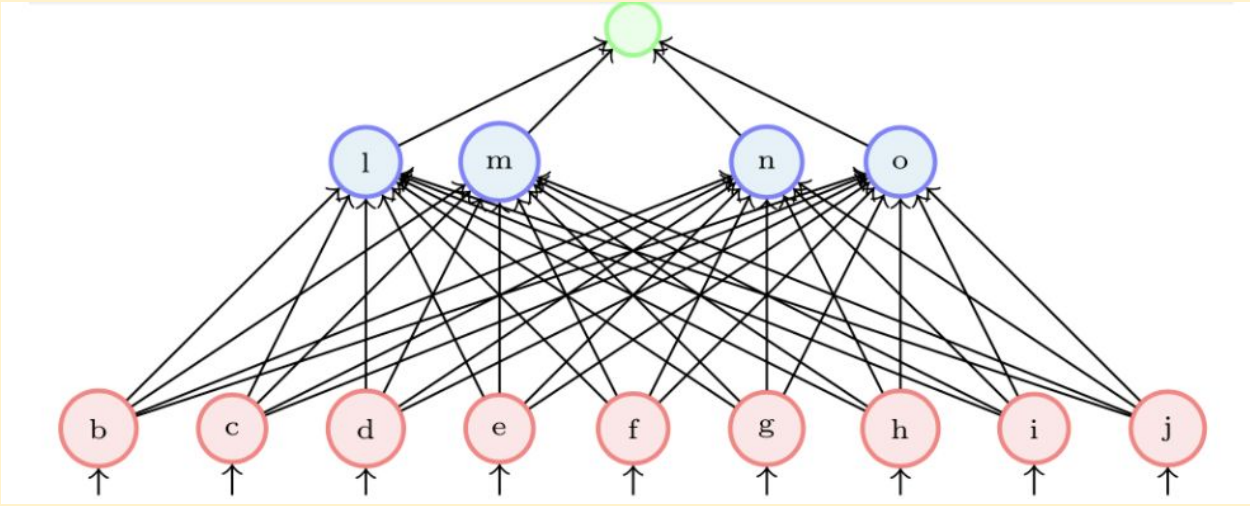
PadhAI: From Convolution Operation to Neural Network

One Fourth Labs

Training CNNs

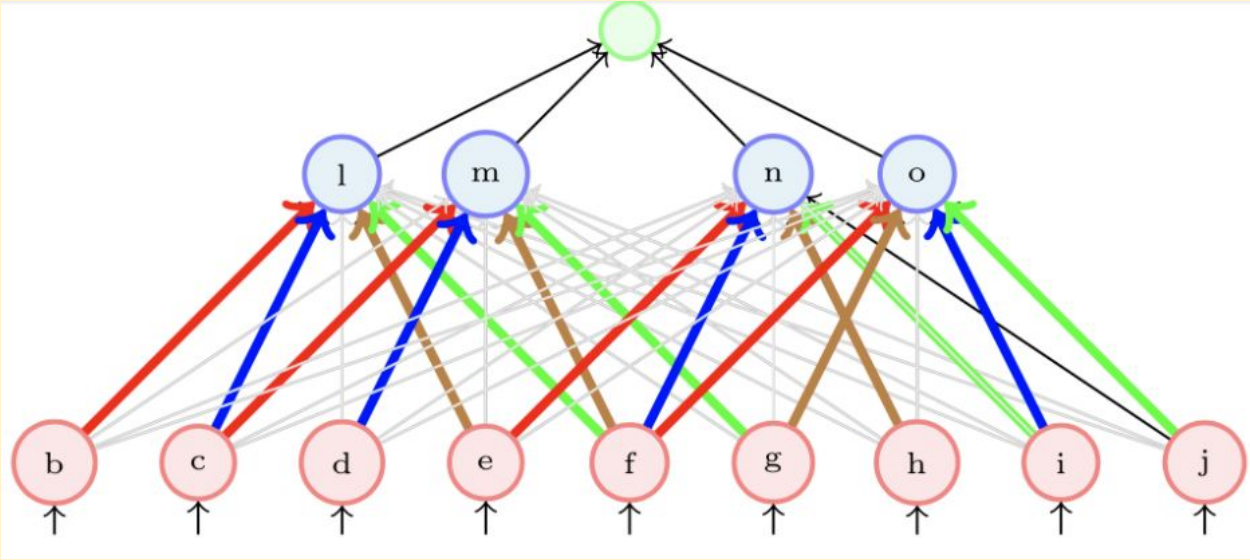
How do we train a Convolutional Neural Network?

1. Let us consider a regular Feedforward Network



   a. Here, we obtain ŷ as a prediction, and use it to calculate the loss.
   b. Using the loss value, we backpropagate to calculate the gradients w.r.t each of the parameters
   c. Using the gradient descent update rule, we update the weights such that they minimize the loss.

2. Now, we can carry over all of these processes to a CNN, with a few small modifications



   a. A CNN can be implemented as a feedforward network wherein only a few weights (coloured) are active
   b. The rest of the weights (grey) remain zero
   c. Thus, we can train a CNN using backpropagation by thinking of it as a FFN with sparse connections

3. However, in practice we don't do this, as most of the weights in the matrix end up being zero. Frameworks like PyTorch and Tensorflow don't end up creating such large matrices and only focus on dealing with the weights that are to be updated.

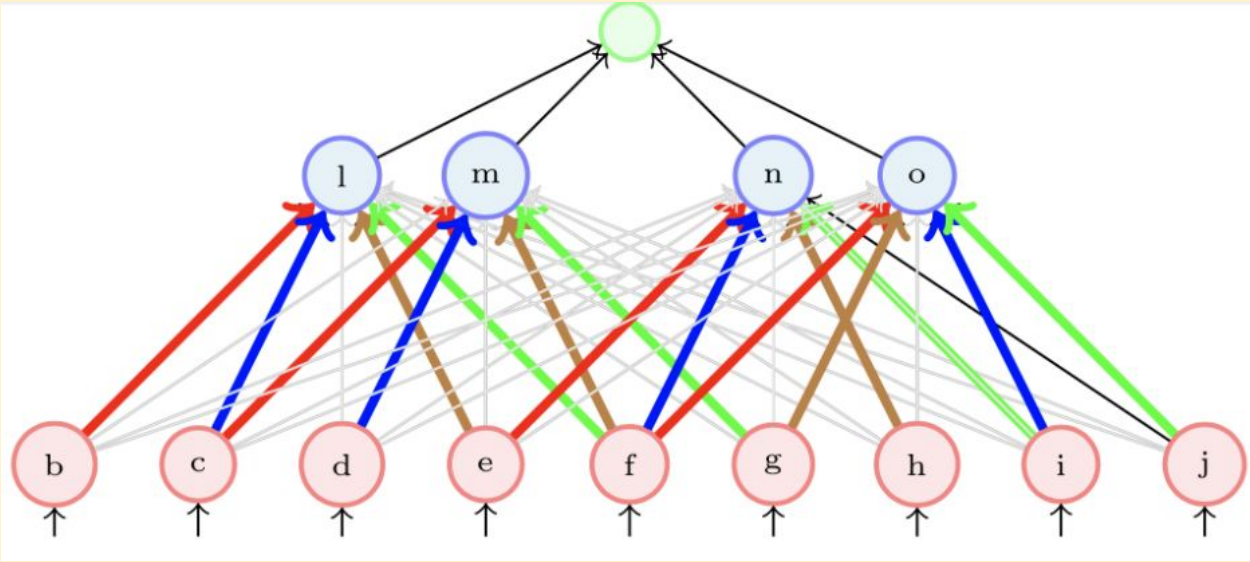
PadhAI: From Convolution Operation to Neural Network

One Fourth Labs

Summary and what next

Making sense of everything we have seen so far

1. By the Universal Approximation Theorem, we have learned that DNNs are powerful function approximators.
2. They can be trained using backpropagation
3. However, due to the large number of parameters, the function can become extremely complex, resulting in a high chance of overfitting the training data.
4. We looked into CNNs to see if we could have a Neural Network which are complex (many non-linearities) but with fewer parameters and hence be less prone to overfitting.



5. CNNs solve both of those shortcomings, with key points such as weight sharing and sparse connectivity. Non-Linearity functions such as ReLU are applied after each convolutional layer.
6. Training CNNs is also very similar to training FFNs, the only difference being we take a 0 value for all weights that we are not interested in.
7. We will then be able to apply learning to the filters(parameters), thereby reducing the overall loss of the CNN.
8. They can be easily implemented with frameworks such as PyTorch or Tensorflow.