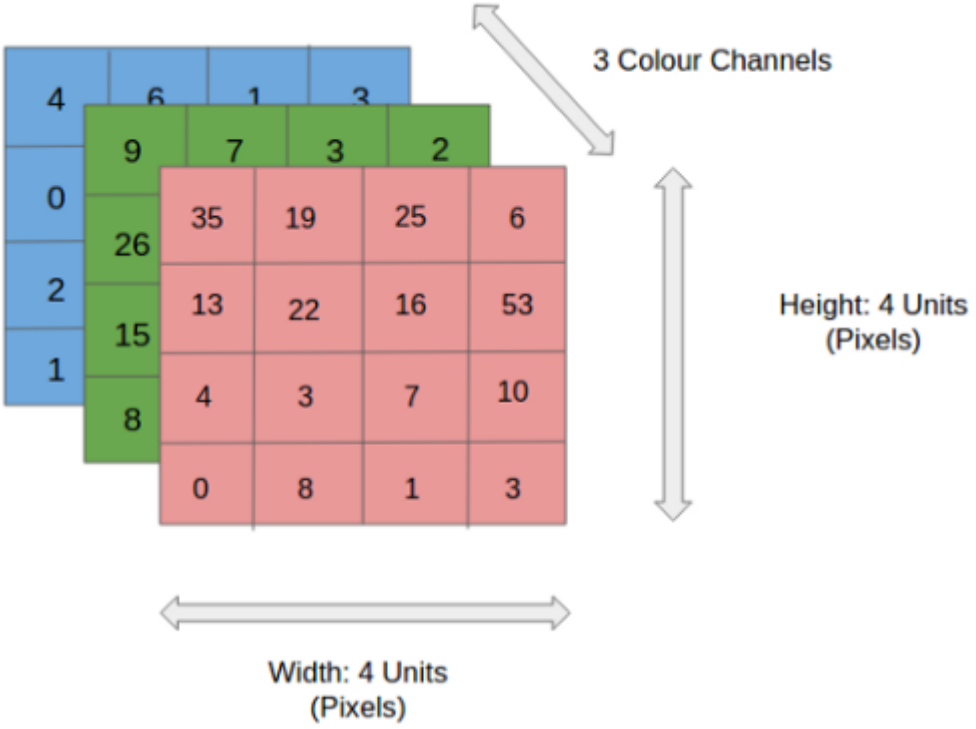
Week 09: CNN

Limitations of feedforward neural networks (FNN) for image processing

In a fully connected FNN (Figure 1), all the nodes in a layer are connected to all the nodes in the next layer. Each connection has a weight wi,j

wi,j that needs to be learned by the learning algorithm. Let's say our input is a 64 pixel by 64 pixels grayscale image. Each grayscale pixel is represented by 1 value, usually, between 0 to 255, where 0 represents black, 255 represents white, and the values in between represent various shades of grey. Since each grayscale pixel can be represented by a 1 value, we say the *channel* size is 1. Such an image can be represented by 64 X 64 X 1 = 4,096 values (rows X columns X channels). Hence, the input layer of an FNN processing such an image has 4096 nodes.

Let's assume the next layer has 500 nodes. Since all the nodes in subsequent layers are fully connected, we will have 4,096 X 500 = 2,048,000 weights between the input and the first hidden layer. For complex problems, we usually need multiple hidden layers in our FNN, as a simpler FNN may not be able to learn the model mapping the inputs to outputs in the training data. Having multiple hidden layers compounds the problem of having many weights in our FNN. Having many weights makes the learning process more difficult as the dimension of the search space is increased. It also makes the training more time and resource consuming and increases the likelihood of overfitting. This problem is further compounded for colour images. Unlike grayscale images, each pixel in a colour image is represented by 3 values, representing red, green, and blue colours (Called RGB colour mode), where every colour can be represented by various combinations of these primary colours. Since each colour pixel can be represented by 3 values, we say the channel size is 3. Such an image can be represented by 64 X 64 X 3 = 12,288 values (rows X columns X channels). The number of weights between the input layer and the first hidden layer with 500 nodes is now 12,288 X 500 = 6,144,000. It is clear that an FNN cannot scale to handle larger images and that we need a more scalable architecture.



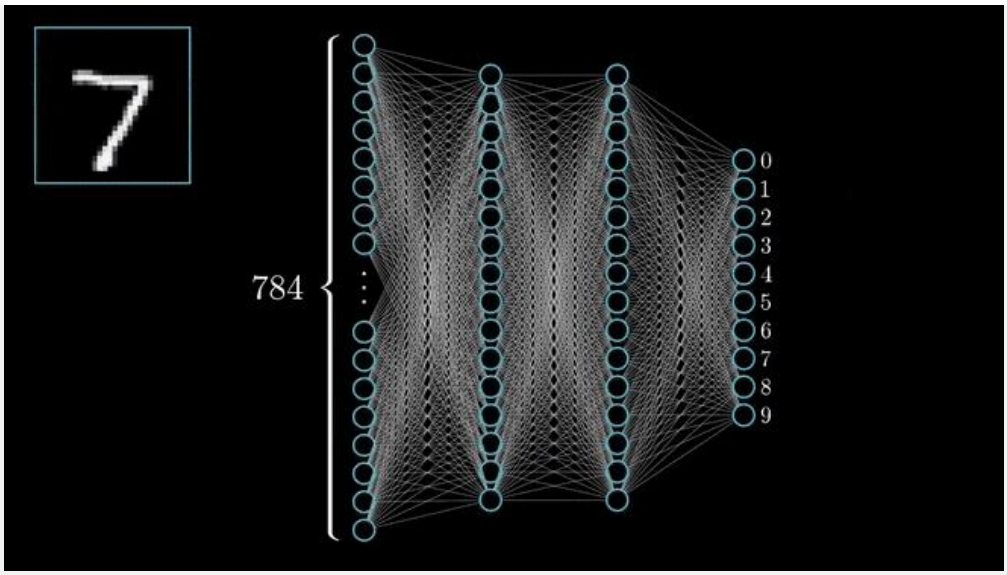


Figure 1: Feedforward neural network with a hidden layer. Biases to hidden/output layer neurons are omitted for clarity

Another problem with using FNN for image processing is that a 2-dimensional image is represented as a 1-dimensional vector in the input layer, hence, any spatial relationship in the data is ignored. CNN, on the other hand, maintains the spatial structure of the data and is better suited for finding spatial relationships in the image data.

Architecture of CNN

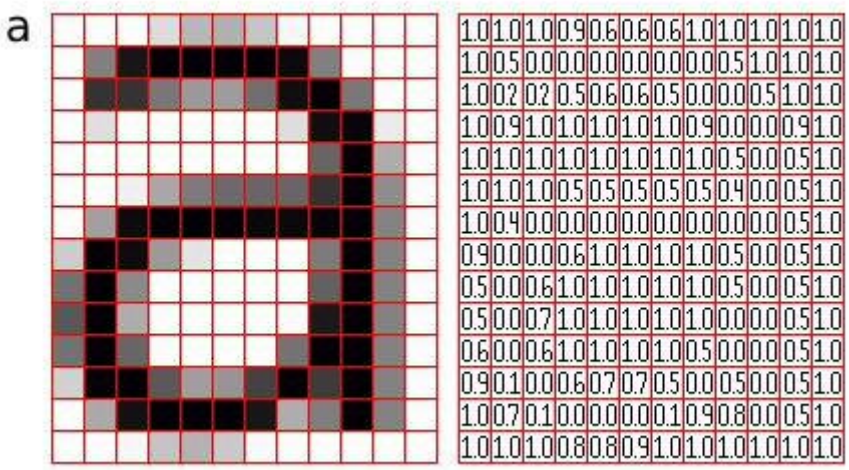
A typical CNN has the following 4 layers (O'Shea and Nash 2015)

1. Input layer
2. Convolution layer (Filtering + ReLU)
3. Pooling layer
4. Fully connected layer (FNN)

Please note that we will explain a 2 dimensional (2D) CNN here. But the same concepts apply to a 1 (or 3) dimensional CNN as well.

Input layer

The input layer represents the input to the CNN. An example input could be a 28 pixel by 28 pixels grayscale image. Unlike FNN, we do not "flatten" the input to a 1D vector, and the input is presented to the network in 2D as a 28 x 28 matrix. This makes capturing spatial relationships easier.



Convolution layer

The convolution layer is composed of multiple filters (also called kernels). Filters for a 2D image are also 2D. Suppose we have a 28 pixel by 28 pixels grayscale image. Each pixel

is represented by a number between 0 and 255, where 0 represents the colour black, 255 represents the colour white, and the values in between represent different shades of grey. Suppose we have a 3 by 3 filter (9 values in total), and the values are randomly set to 0 or 1. Convolution is the process of placing the 3 by 3 filter on the top left corner of the image, multiplying filter values by the pixel values and adding the results, moving the filter to the right one pixel at a time and repeating this process. When we get to the top right corner of the image, we simply move the filter down one pixel and restart from the right. This process ends when we get to the bottom right corner of the image.

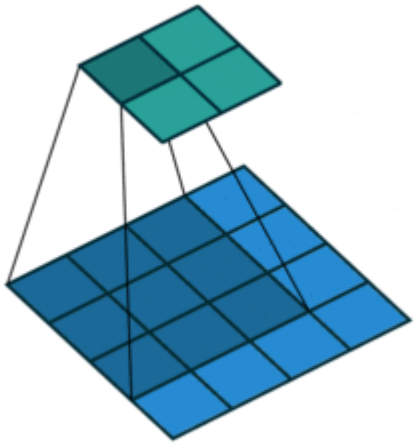


Figure 2: A 3 by 3 filter applied to a 4 by 4 image, resulting in a 2 by 2 image (Dumoulin and Visin 2016)

The convolution operator has the following parameters:

1. Filter size
2. Padding
3. Stride
4. Dilation
5. Activation function (ReLU)

Filter size can be 5 by 5, 3 by 3, and so on. Larger filter sizes should be avoided as the learning algorithm needs to learn filter values (weights), and larger filters increase the number of weights to be learned (more compute capacity, more training time, more chance of overfitting). Also, odd-sized filters are preferred to even sized filters, due to the nice geometric property of all the input pixels being around the output pixel.

If you look at Figure 2 you see that after applying a 3 by 3 filter to a 4 by 4 image, we end up with a 2 by 2 image – the size of the image has gone down. If we want to keep the resultant image size the same, we can use *padding*. We pad the input in every direction

with 0's before applying the filter. If the padding is 1 by 1, then we add 1 zero in every direction. If it's 2 by 2, then we add 2 zeros in every direction, and so on.

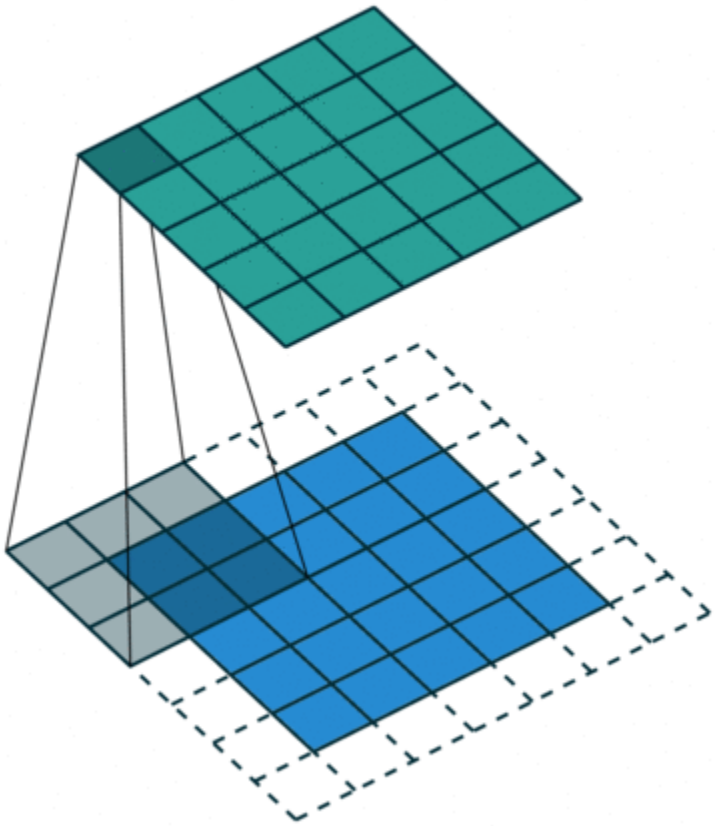


Figure 3: A 3 by 3 filter applied to a 5 by 5 image, with padding of 1, resulting in a 5 by 5 image (Dumoulin and Visin 2016)

As mentioned before, we start the convolution by placing the filter on the top left corner of the image, and after multiplying filter and image values (and adding them), we move the filter to the right and repeat the process. How many pixels we move to the right (or down) is the *stride*. In Figures 2 and 3, the stride of the filter is 1. We move the filter one pixel to the right (or down). But we could use a different stride. Figure 4 shows an example of using a stride of 2.

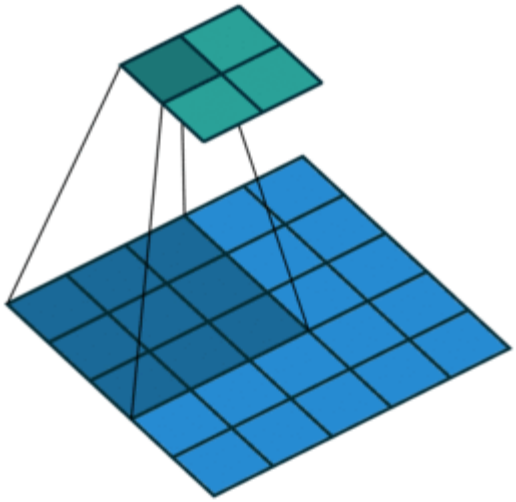


Figure 4: A 3 by 3 filter applied to a 5 by 5 image, with a stride of 2, resulting in a 2 by 2 image (Dumoulin and Visin 2016)

When we apply a, say 3 by 3, filter to an image, our filter's output is affected by pixels in a 3 by 3 subset of the image. If we like to have a larger *receptive field* (a portion of the image that affects our filter's output), we could use *dilation*. If we set the dilation to 2 (Figure 5), instead of a contiguous 3 by 3 subset of the image, every other pixel of a 5 by 5 subset of the image affects the filter's output.

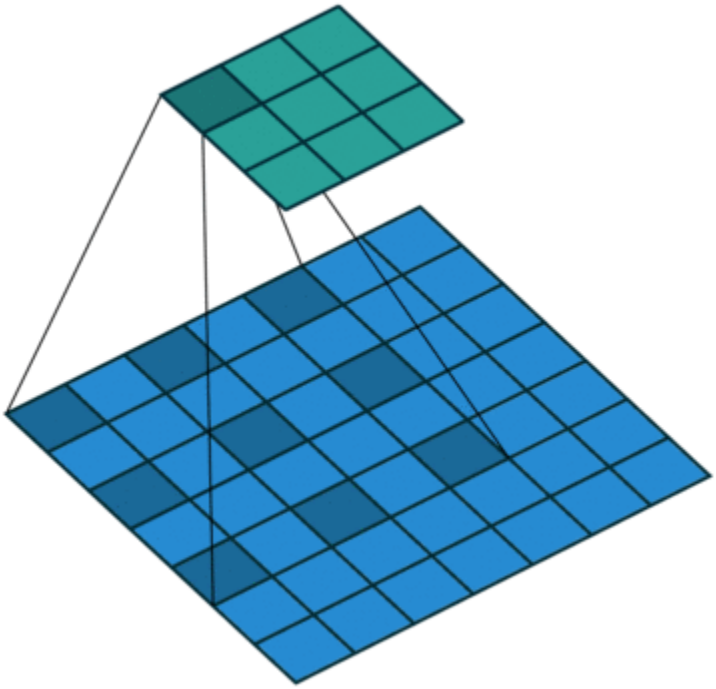


Figure 5: A 3 by 3 filter applied to a 7 by 7 image, with dilation of 2, resulting in a 3 by 3 image (Dumoulin and Visin 2016)

After the filter scans the whole image, we apply an activation function to filter output to introduce non-linearity. The preferred activation function used in CNN is ReLU or one of its variants like Leaky ReLU (Nwankpa *et al.* 2018). ReLU leaves pixels with positive values in filter output as-is and replaces negative values with 0 (or a small number in the case of Leaky ReLU). Figure 6 shows the results of applying the ReLU activation function to filter output.

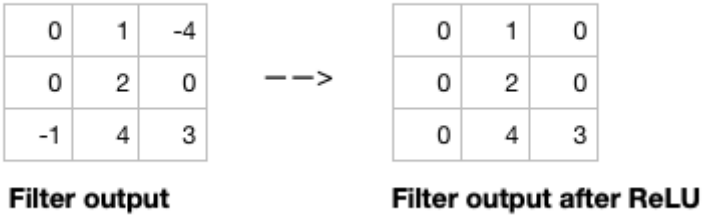


Figure 6: Applying ReLU activation function to filter output

Given the input size, filter size, padding, stride and dilation you can calculate the output size of the convolution operation as below.

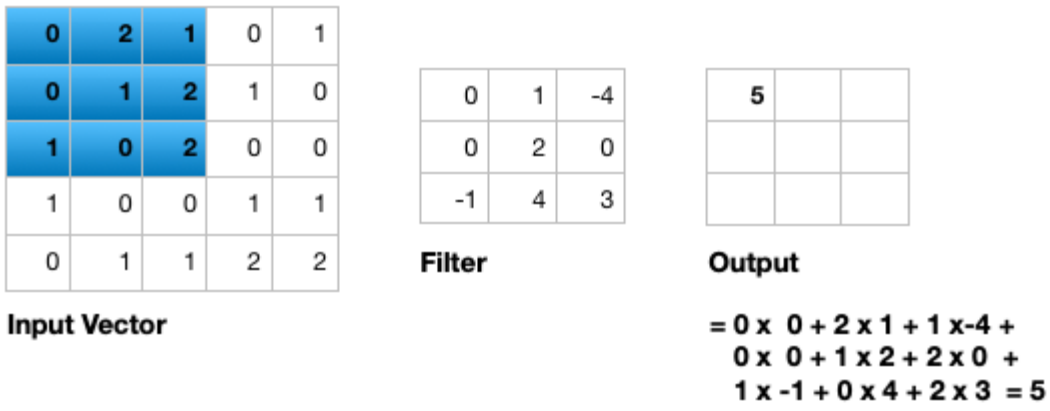
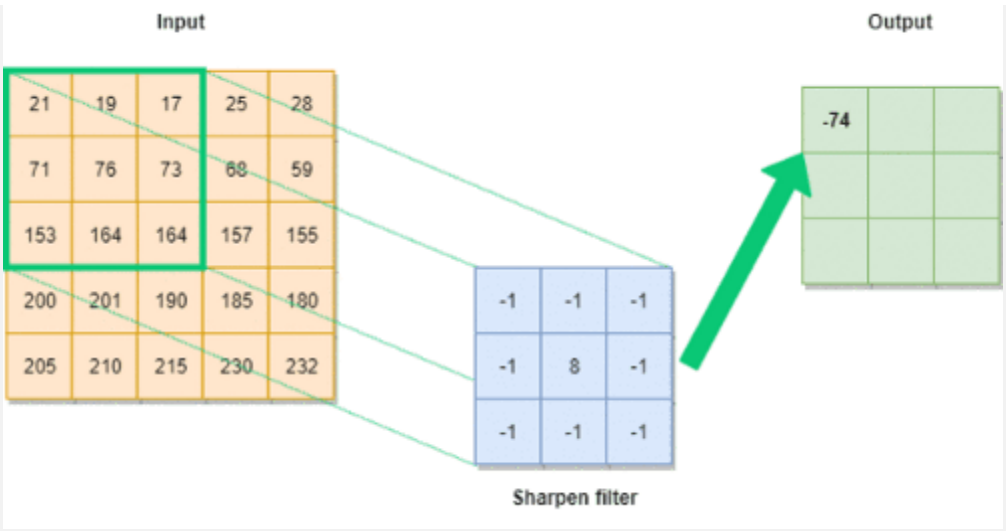


Figure 7: Illustration of single input channel two dimensional convolution


Input
Output
21 19 17 25 28
71 76 73 68 59
153 164 164 157 155
200 201 190 185 180
205 210 215 230 232
-1 -1 -1
-1 8 -1
-1 -1 -1
-74
Sharpen filter

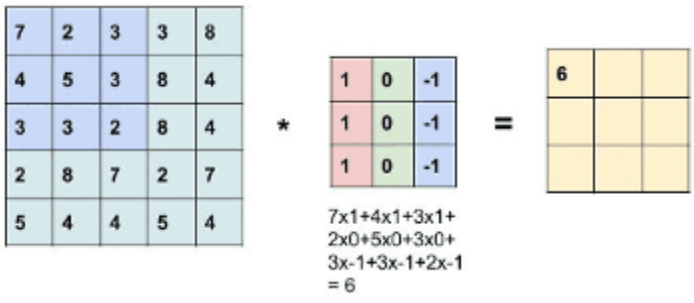

7 2 3 3 8
4 5 3 8 4
3 3 2 8 4
2 8 7 2 7
5 4 4 5 4
*
1 0 -1
1 0 -1
1 0 -1
=
6
7x1+4x1+3x1+
2x0+5x0+3x0+
3x-1+3x-1+2x-1
= 6

Figure 7 illustrates the calculations for a convolution operation, via a 3 by 3 filter on a single channel 5 by 5 input vector (5 x 5 x 1). Figure 8 illustrates the calculations when the input vector has 3 channels (5 x 5 x 3). To show this in 2 dimensions, we are displaying each channel in the input vector and filter separately. Figure 9 shows a sample multi-channel 2D convolution in 3 dimensions.

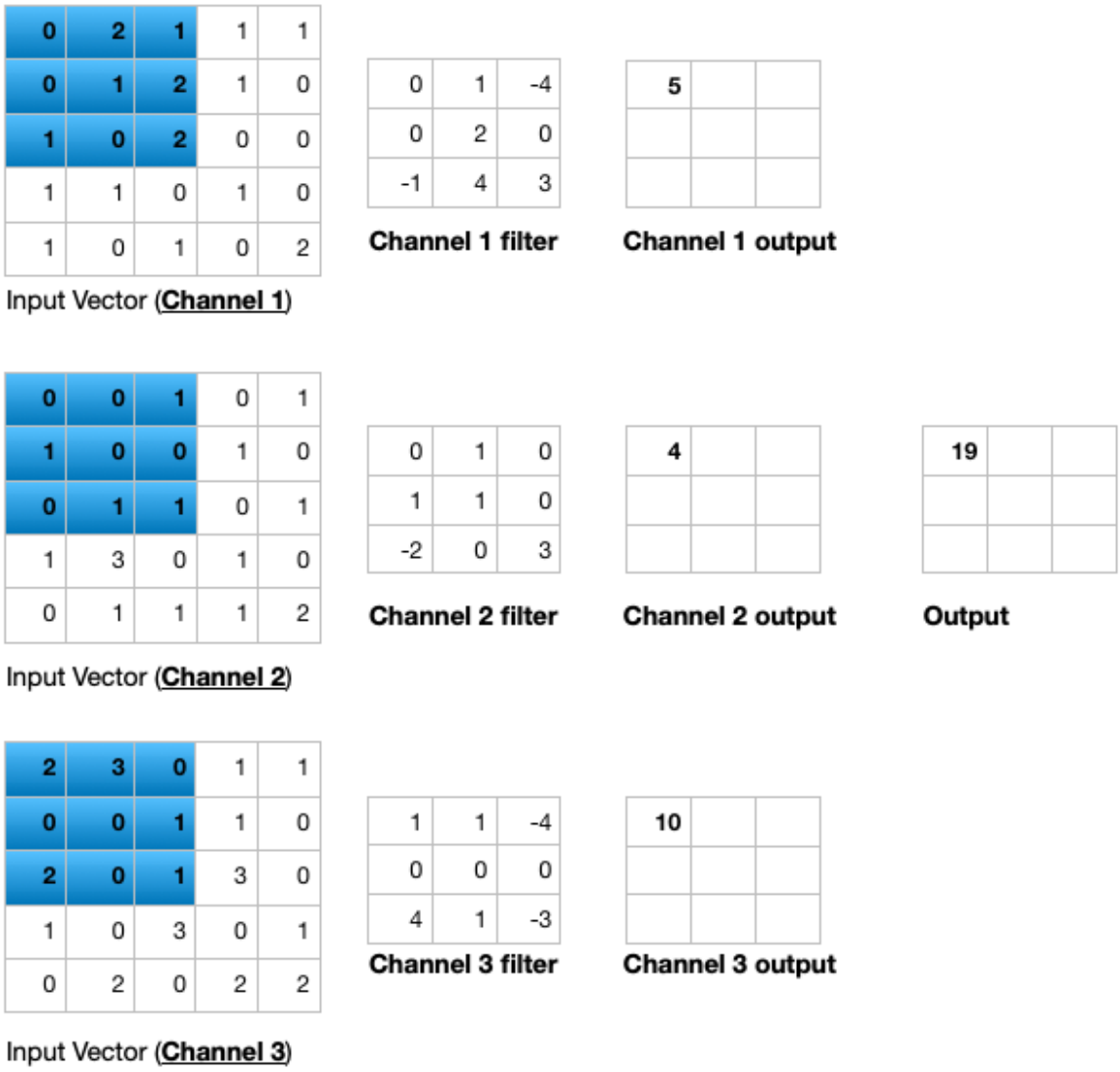
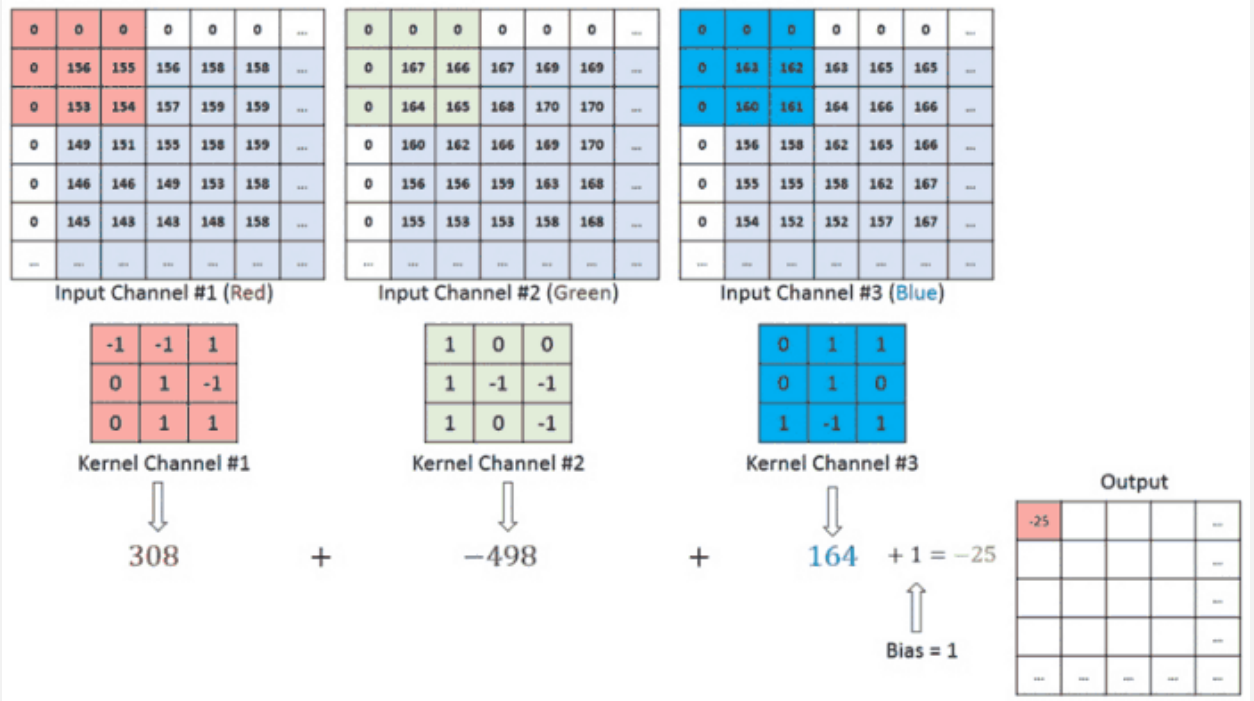


Figure 8: Illustration of multiple input channel two dimensional convolution



Figures 8 and 9 show the output of a multi-channel 2-dimensional filter is a single channel 2 dimensional image. Applying *multiple* filters to the input image results in a multi-channel 2-dimensional image for the output. For example, if the input image is 28 by 28 by 3 (rows and columns x channels), and we apply a 3 by 3 filter with 1 by 1 padding, we would get a 28 by 28 by 1 image. If we apply 15 filters to the input image, our output would be 28 by 28 by 15. Hence, the number of filters in a convolution layer allows us to increase or decrease the channel size.

Padding

Sometimes the filter does not perfectly fit the input image. We have two options:

- Pad the picture with zeros (zero-padding) so that it fits
- Drop the part of the image where the filter did not fit. This is called valid padding which keeps only a valid part of the image.

Non Linearity (ReLU)

ReLU stands for Rectified Linear Unit for a non-linear operation. The output is $f(x) = max(0,x)$. Why ReLU is important: ReLU's purpose is to introduce non-linearity in our ConvNet. Since the real-world data would want our ConvNet to learn would be non-negative linear values.

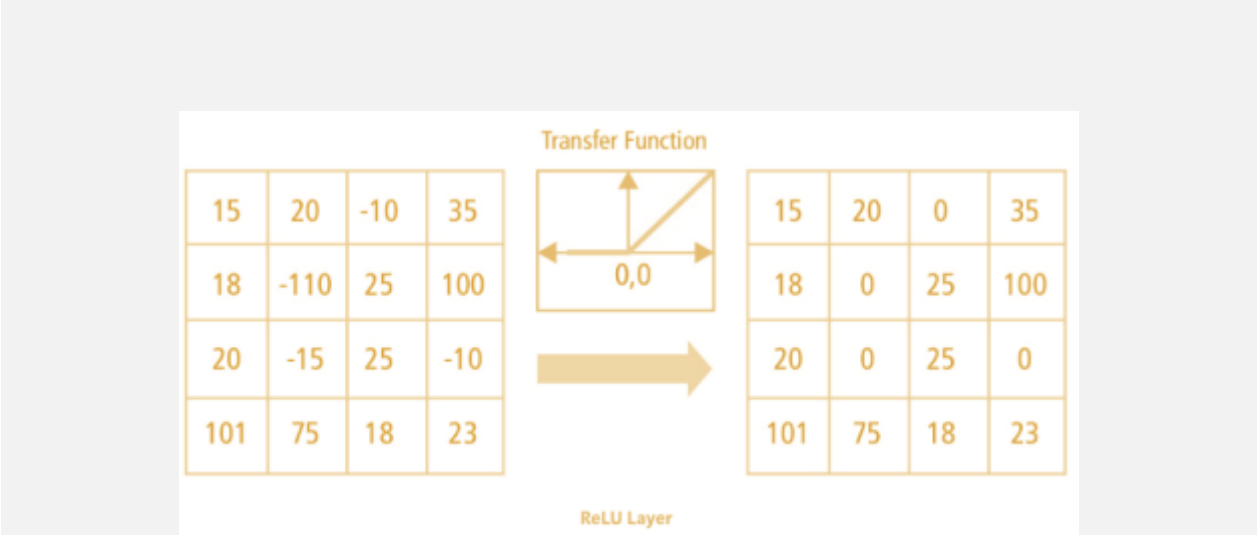


Figure 7: ReLU operation

There are other non-linear functions such as tanh or sigmoid that can also be used instead of ReLU. Most of the data scientists use ReLU since performance-wise ReLU is better than the other two.

Pooling layer

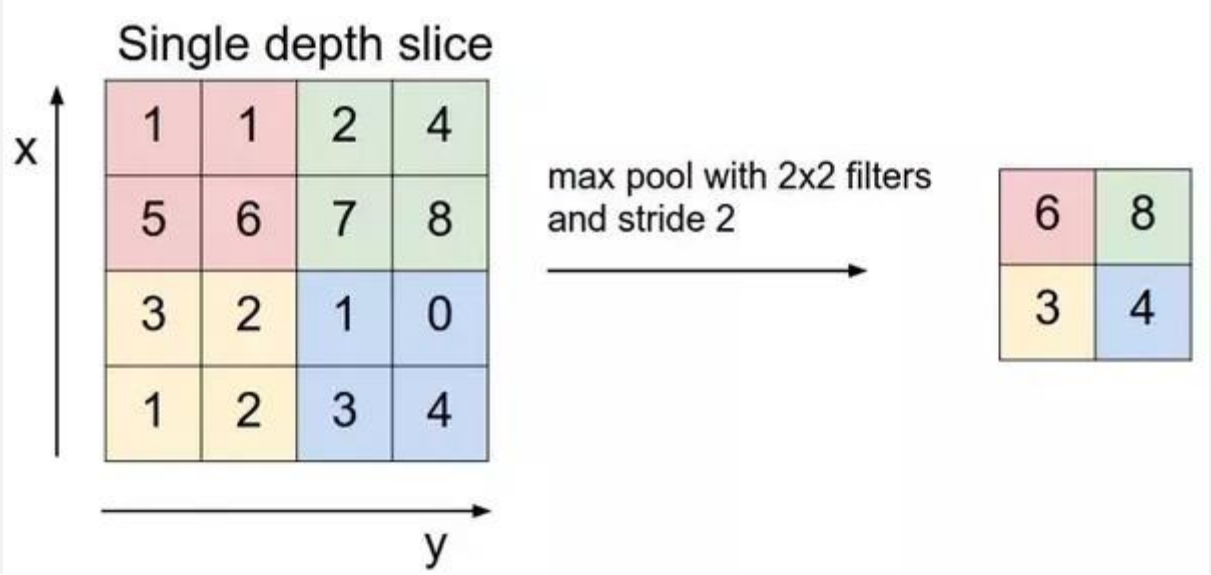
The pooling layer performs downsampling to reduce the spatial dimensionality of the input. This decreases the number of parameters, which in turn reduces the learning time and computation, and the likelihood of overfitting. The most popular type of pooling is *max pooling*. It's usually a 2 by 2 filter with a stride of 2 that returns the maximum value as it slides over the input data (similar to convolution filters).

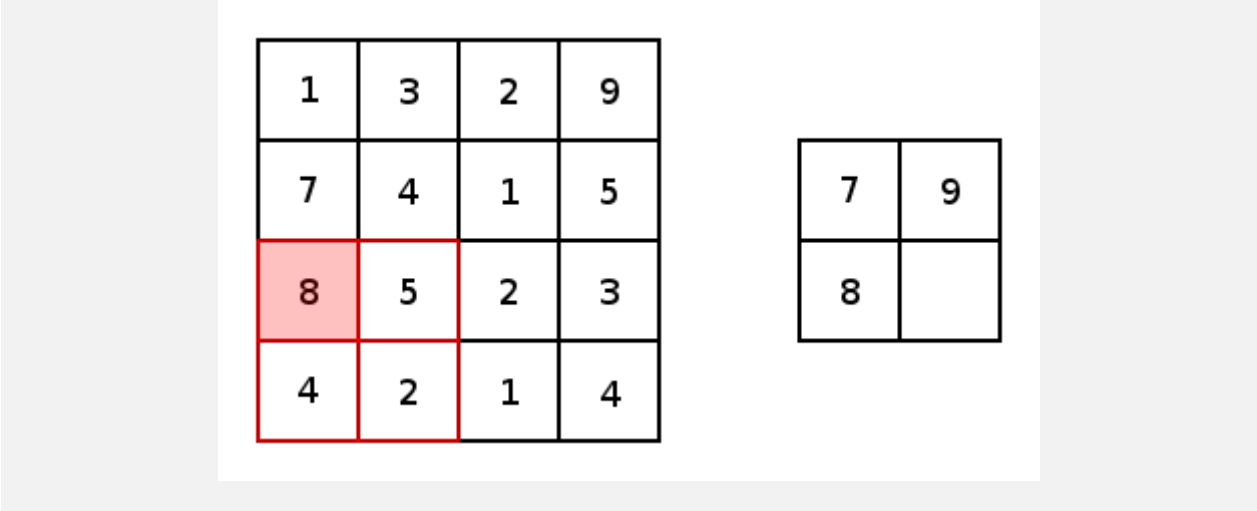
The pooling layers section would reduce the number of parameters when the images are too large. Spatial pooling also called subsampling or downsampling which reduces the

dimensionality of each map but retains important information. Spatial Pooling can be of different types:

- Max Pooling
- Average Pooling
- Sum Pooling

Max pooling takes the largest element from the rectified feature map. Taking the largest element could also take the average pooling. Sum of all elements in the feature map call as sum pooling.





Fully connected layer

The last layer in a CNN is a fully connected layer. We connect all the nodes from the previous layer to this fully connected layer, which is responsible for the classification of the image.

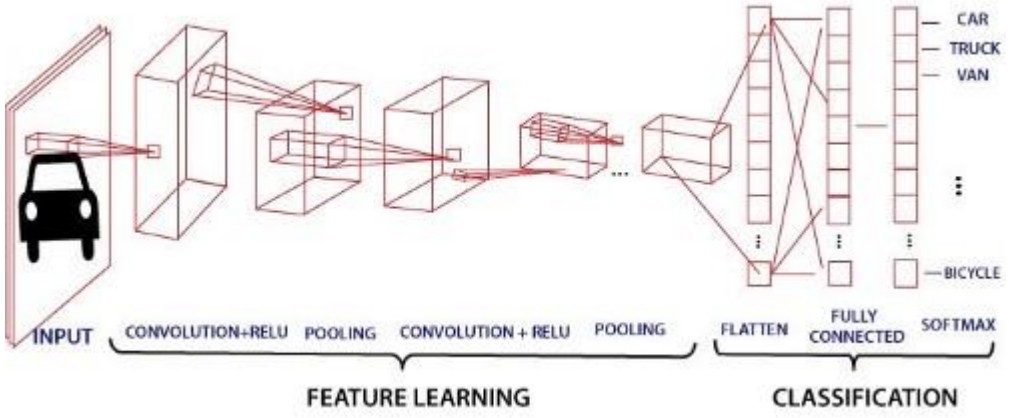
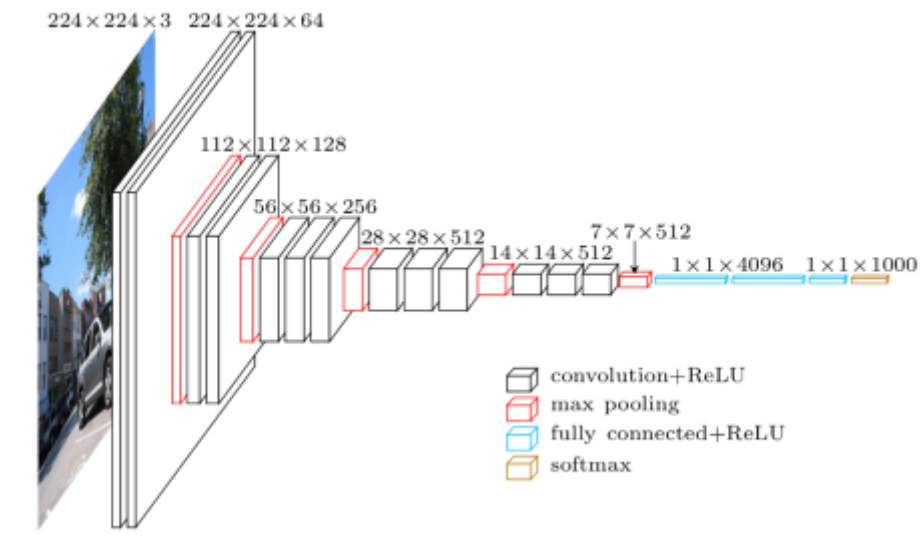


Figure 10: A convolutional neural network with 3 convolution layers followed by 3 pooling layers (O'Shea and Nash 2015)

As shown in Figure 10, a typical CNN usually has more than one convolution layer plus a pooling layer. Each convolution plus pooling layer is responsible for feature extraction at a different level of abstraction. For example, the filters in the first layer could detect horizontal, vertical, and diagonal edges. The filters in the next layer could detect shapes, and the filters in the last layer could detect a collection of shapes. Filter values are

randomly initialized and are learned by the learning algorithm. This makes CNN very powerful as they not only do classification but can also automatically do feature extraction. This distinguishes CNN from other classification techniques (like Support Vector Machines), which cannot do feature extraction.



There are various architectures of CNNs available which have been key in building algorithms that power and shall power AI as a whole in the foreseeable future. Some of them have been listed below:

1. LeNet
2. AlexNet
3. VGGNet
4. GoogLeNet
5. ResNet
6. ZDNet

Application of CNN

1. Decoding Facial Recognition: Facial recognition is broken down by a convolutional neural network into the following major components -
   - Identifying every face in the picture
   - Focusing on each face despite external factors, such as light, angle, pose, etc.
   - Identifying unique features

- Comparing all the collected data with already existing data in the database to match a face with a name.

2. Analyzing Documents: **Convolutional neural networks** can also be used for document analysis. This is not just useful for handwriting analysis, but also has a major stake in recognizers. For a machine to be able to scan an individual's writing, and then compare that to the wide database it has, it must execute almost a million commands a minute. It is said with the use of CNNs and newer models and algorithms, the error rate has been brought down to a minimum of 0.4% at a character level, though it's complete testing is yet to be widely seen.

3. Understanding Climate: CNNs can be used to play a major role in the fight against climate change, especially in understanding the reasons why we see such drastic changes and how we could experiment in curbing the effect. It is said that the data in such natural history collections can also provide greater social and scientific insights, but this would require skilled human resources such as researchers who can physically visit these types of repositories. There is a need for more manpower to carry out deeper experiments in this field.

4. Grey Areas: Introduction of the grey area into CNNs is posed to provide a much more realistic picture of the real world. Currently, CNNs largely function exactly like a machine, seeing a true and false value for every question. However, as humans, we understand that the real world plays out in a thousand shades of grey. Allowing the machine to understand and process fuzzier logic will help it understand the grey area us humans live in and strive to work against. This will help CNNs get a more holistic view of what humans see.

5. Advertising: CNNs have already brought in a world of difference to advertising with the introduction of programmatic buying and data-driven personalized advertising.

7. Other Interesting Field: CNNs are poised to be the future with their introduction into driverless cars, robots that can mimic human behaviour, aides to human genome mapping projects, predicting earthquakes and natural disasters, and maybe even self-diagnoses of medical problems. So, you wouldn't even have to drive down to a clinic or schedule an appointment with a doctor to ensure your sneezing attack or high fever is just the simple flu and not the symptoms of some rare disease. One problem that researchers are working on with CNNs is brain cancer detection. The earlier detection of brain cancer can prove to be a big step in saving more lives affected by this illness.

References

https://training.galaxyproject.org/training-material/topics/statistics/tutorials/CNN/tutorial.html#classification-of-mnist-dataset-images-with-cnn

https://www.kaggle.com/kanncaa1/convolutional-neural-network-cnn-tutorial

https://medium.com/analytics-vidhya/plant-disease-detection-using-convolutional-neural-networks-and-pytorch-87c00c54c88f

How do Convolutional Neural Networks work? (e2eml.school)

https://medium.com/analytics-vidhya/convolutional-neural-network-cnn-and-its-application-all-u-need-to-know-f29c1d51b3e5

Compiled By : Dr Md Ashraf Uddin