



IJRASET

International Journal For Research in
Applied Science and Engineering Technology



INTERNATIONAL JOURNAL FOR RESEARCH

IN APPLIED SCIENCE & ENGINEERING TECHNOLOGY

Volume: 10 **Issue:** XII **Month of publication:** December 2022

DOI: <https://doi.org/10.22214/ijraset.2022.47789>

www.ijraset.com

Call:  08813907089

E-mail ID: ijraset@gmail.com

An Introduction to Convolutional Neural Networks

Aarush Saxena

Manipal Institute of Technology

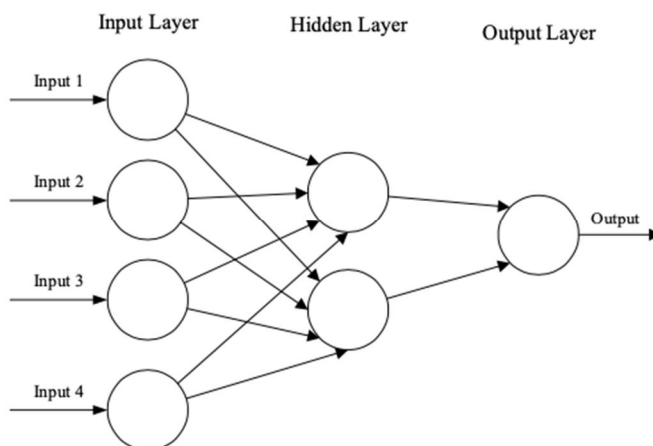
Abstract: *The field of machine learning has taken a dramatic twist in recent times, with the rise of the Artificial Neural Network (ANN). These biologically inspired computational models can far exceed the performance of previous forms of artificial intelligence in common machine-learning tasks. One of the most impressive forms of ANN architecture is that of the Convolutional Neural Network (CNN). CNNs are primarily used to solve difficult image-driven pattern recognition tasks and with their precise yet simple architecture, offer a simplified method of getting started with ANNs.*

This document briefly introduces CNNs, discussing recently published papers and newly formed techniques in developing this bill- leniently fantastic image recognition models. This introduction assumes you are familiar with ANNs and machine learning fundamentals.

Keywords: *Pattern recognition, artificial neural networks, machine learning, image analysis*

I. INTRODUCTION

Artificial Neural Networks (ANNs) are computational processing systems which are heavily inspired by the way biological nervous systems (such as the human brain) operate. ANNs are mainly comprised of a high number of interconnected computational nodes (referred to as neurons), which work together in a distributed fashion to collectively learn from the input to optimise its final output. The basic structure of an ANN can be modelled as shown in Figure 1. We would load the input, usually in the form of a multidimensional vector to the input layer which will distribute it to the hidden layers. The hidden layers will then make decisions from the previous layer and weigh up how a stochastic change within itself detracts or improves the final output, and this is referred to as the process of learning. Having multiple hidden layers stacked upon each other is commonly called deep learning.



The two key learning paradigms in image processing tasks are supervised and unsupervised learning. Supervised learning is learning through pre-labelled inputs, which act as targets. For each training example, there will be a set of input values (vectors) and one or more associated designated output values. The goal of this form of training is to reduce the model's overall classification error, through correct calculation of the output value of training example training. Unsupervised learning differs in that the training set does not include any labels. Success is usually determined by whether the network can reduce or increase an associated cost function. However, it is important to note that most image-focused pattern-recognition tasks usually depend on classification using supervised learning. Convolutional Neural Networks (CNNs) are analogous to traditional ANNs in that they are comprised of neurons that self-optimize through learning. Each neuron will still receive input and operate (such as a scalar product followed by a non-linear function) - the basis of countless ANNs. From the input raw image vectors to the final output of the class score, the entire network will still express a single perceptive score function (the weight). The last layer will contain loss functions associated with the classes, and all of the regular tips and tricks developed for traditional ANNs still apply.

The only notable difference between CNNs and traditional ANNs is that CNNs are primarily used in the field of pattern recognition within images. This allows us to encode image-specific features into the architecture, making the network more suited for image-focused tasks - whilst further reducing the parameters required to set up the model.

One of the largest limitations of traditional forms of ANN is that they tend to struggle with the computational complexity required to compute image data. Common machine learning benchmarking datasets such as the MNIST database of handwritten digits are suitable for most forms of ANN, due to its relatively small image dimensionality of just 28×28 . With this dataset, a single neuron in the first hidden layer will contain 784 weights ($28 \times 28 \times 1$ where 1 bears in mind that MNIST is normalised to just black and white values), which is manageable for most forms of ANN.

If you consider a more substantial coloured image input of 64×64 , the number of weights on just a single neuron of the first layer increases substantially to 12, 288. Also, take into account that to deal with this scale of input, the network will also need to be a lot larger than the one used to classify colour-normalised MNIST digits, and then you will understand the drawbacks of using such models.

A. Overfitting

But why does it matter? Surely we could just increase the number of hidden layers in our network, and perhaps increase the number of neurons within them? The simple answer to this question is no. This is down to two reasons, one being the simple problem of not having unlimited computational power and time to train these huge ANNs.

The second reason is stopping or reducing the effects of overfitting. **Overfitting** is basically when a network is unable to learn effectively due to several reasons. It is an important concept of most, if not all machine learning algorithms and every precaution must be taken to reduce its effects. If our models were to exhibit signs of overfitting then we may see a reduced ability to pinpoint generalised features for not only our training dataset but also our test and prediction sets.

This is the main reason behind reducing the complexity of our ANNs. The fewer parameters required to train, the less likely the network will overfit - and of course, improve the predictive performance of the model.

II. CNN ARCHITECTURE

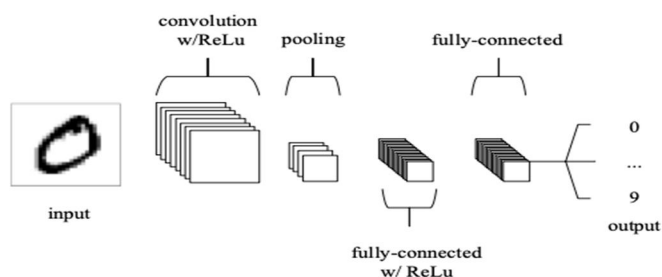
As noted earlier, CNN primarily focuses on the basis that the input will be comprised of images. This focuses on the architecture to be set up in a way to best suit the need for dealing with the specific type of data.

One of the key differences is the neurons that the layers within the CNN are comprised of neurons organised into three dimensions, the spatial dimensionality of the input (**height** and **width**) and the **depth**. The depth does not refer to the total number of layers within the ANN, but the third dimension of an activation volume. Unlike standard ANNS, the neurons within any given layer will only connect to a small region of the layer preceding it.

In practice this would mean that for the example given earlier, the input 'volume' will have a dimensionality of $64 \times 64 \times 3$ (height, width and depth), leading to a final output layer comprised of a dimensionality of $1 \times 1 \times n$ (where n represents the possible number of classes) as we would have condensed the full input dimensionality into a smaller volume of class scores filed across the depth dimension.

A. Overall Architecture

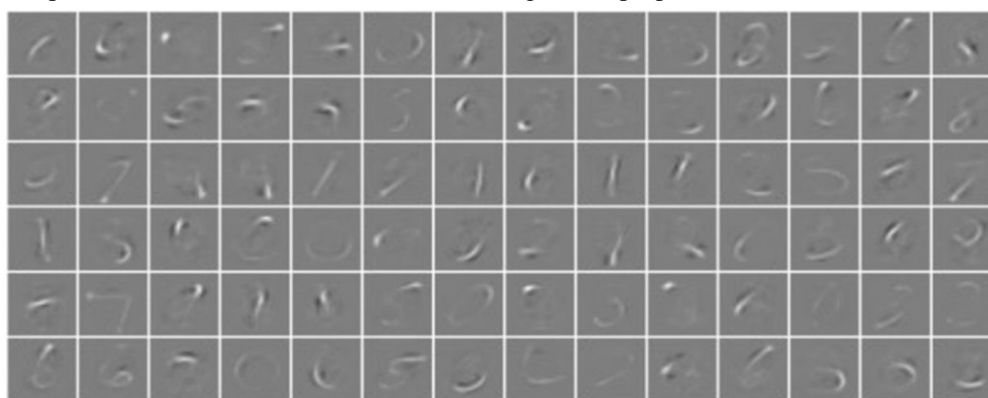
CNN's are comprised of three types of layers. These are convolutional layers, pooling layers and fully-connected layers. When these layers are stacked, a CNN architecture has been formed. A simplified CNN architecture for MNIST classification is illustrated in Figure 2.



The basic functionality of the example CNN above can be broken down into four key areas.

- 1) As found in other forms of ANN, the input layer will hold the pixel values of the image.
- 2) The convolutional layer will determine the output of neurons which are connected to local regions of the input through the calculation of the scalar product between their weights and the region connected to the input volume. The rectified linear unit (commonly shortened to ReLu) aims to apply an 'elementwise' activation function such as sigmoid to the output of the activation produced by the previous layer.
- 3) The pooling layer will then simply perform down-sampling along the spatial dimensionality of the given input, further reducing the number of parameters within that activation.
- 4) The fully-connected layers will then perform the same duties found in standard ANNs and attempt to produce class scores from the activations, to be used for classification. It is also suggested that ReLu may be used between these layers, as to improve performance.

Through this simple method of transformation, CNNs can transform the original input layer by layer using convolutional and down-sampling techniques to produce class scores for classification and regression purposes.



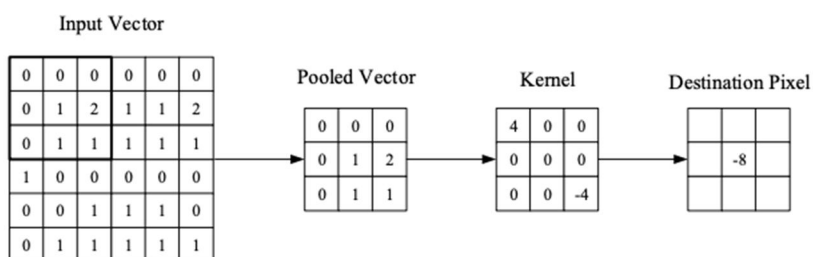
However, it is important to note that simply understanding the overall architecture of a CNN architecture will not suffice. The creation and optimisation of these models can take quite some time and can be quite confusing. We will now explore in detail the individual layers, detailing their hyperparameters and connectivities.

B. Convolutional Layer

As the name implies, the convolutional layer plays a vital role in how CNNs operate. The layer's parameters focus on the use of learnable kernels.

These kernels are usually small in spatial dimensionality but spread along the entirety of the depth of the input. When the data hits a convolutional layer, the layer convolves each filter across the spatial dimensionality of the input to produce a 2D activation map. These activation maps can be visualised, as seen in Figure 3.

As we glide through the input, the scalar product is calculated for each value in that kernel. (Figure 4) From this, the network will learn kernels that 'fire' when they see a specific feature at a given spatial position of the input. These are commonly known as activations.



Every kernel will have a corresponding activation map, of which will be stacked along the depth dimension to form the full output volume from the convolutional layer.

As we alluded to earlier, training ANNs on inputs such as images results in models which are too big to train effectively. This comes down to the fully- connected manner of standard ANN neurons, so to mitigate against this every neuron in a convolutional layer is only connected to a small region of the input volume. The dimensionality of this region is commonly referred to as the receptive field size of the neuron. The magnitude of the connectivity through the depth is nearly always equal to the depth of the input.

For example, if the input to the network is an image of size $64 \times 64 \times 3$ (an RGB- coloured image with a dimensionality of 64×64) and we set the receptive field size as 6×6 , we would have a total of 108 weights on each neuron within the convolutional layer. ($6 \times 6 \times 3$ where 3 is the magnitude of connectivity across the depth of the volume) To put this into perspective, a standard neuron seen in other forms of ANN would contain 12, 288 weights each.

Convolutional layers are also able to significantly reduce the complexity of the model through the optimisation of its output. These are optimised through three hyperparameters, the depth, the stride and setting zero-padding.

The depth of the output volume produced by the convolutional layers can be manually set through the number of neurons within the layer to the same region of the input. This can be seen with other forms of ANNs, where all of the neurons in the hidden layer are directly connected to every single neuron beforehand. Reducing this hyperparameter can significantly minimise the total number of neurons in the network, but it can also significantly reduce the pattern recognition capabilities of the model.

We are also able to define the stride in which we set the depth around the spatial dimensionality of the input to place the receptive field. For example, if we were to set a stride as 1, then we would have a heavily overlapped receptive field producing extremely large activations. Alternatively, setting the stride to a greater number will reduce the amount of overlapping and produce an output of lower spatial dimensions.

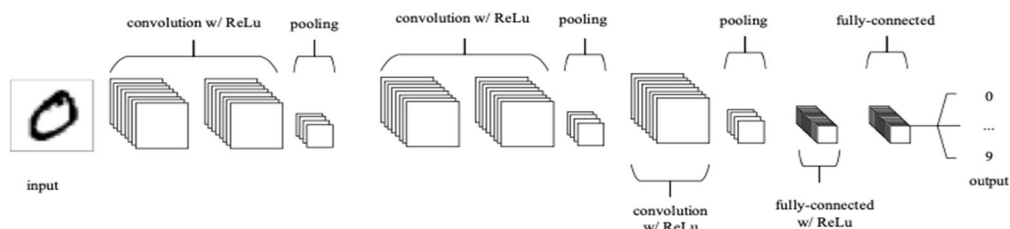
C. Pooling Layer

Pooling layers aim to gradually reduce the dimensionality of the representation, thus further reducing the number of parameters and the computational complexity of the model. The pooling layer operates over each activation map in the input and scales its dimensionality using the “MAX” function. In most CNNs, these come in the form of max-pooling layers with kernels of a dimensionality of 2×2 applied with a stride of 2 along the spatial dimensions of the input. This scales the activation map down to 25% of the original size - whilst maintaining the depth volume to its standard size. Due to the destructive nature of the pooling layer, there are only two generally observed methods of max-pooling. Usually, the stride and filters of the pooling layers are both set to 2×2 , which will allow the layer to extend through the entirety of the spatial dimensionality of the input. Furthermore, overlapping pooling may be utilised, where the stride is set to 2 with a kernel size set to 3. Due to the destructive nature of pooling, having a kernel size above 3 will usually greatly decrease the performance of the model. It is also important to understand that beyond max-pooling, CNN architectures may contain general pooling. General pooling layers are comprised of pooling neurons that can perform a multitude of common operations including L1/L2-normalisation, and average pooling. However, this tutorial will primarily focus on the use of max-pooling.

III. RECIPES

Despite the relatively small number of layers required to form a CNN, there is no set way of formulating a CNN architecture. That being said, it would be idiotic to simply throw a few of layers together and expect it to work. Through reading related literature it is obvious that much like other forms of ANNs, CNNs tend to follow a common architecture. This common architecture is illustrated in Figure 2, where convolutional layers are stacked, followed by pooling layers in a repeated manner before feeding forward to fully-connected layers.

Another common CNN architecture is to stack two convolutional layers before each pooling layer, as illustrated in Figure 5. This is strongly recommended as stacking multiple convolutional layers allows for more complex features of the input vector to be selected.



It is also advised to split large convolutional layers up into many smaller-sized convolutional layers. This is to reduce the amount of computational complexity within a given convolutional layer. For example, if you were to stack three convolutional layers on top of each other with a receptive field of 3×3 . Each neuron of the first convolutional layer will have a 3×3 view of the input vector. A neuron on the second convolutional layer will then have a 5×5 view of the input vector. A neuron on the third convolutional layer will then have a 7×7 view of the input vector. As these stacks feature non-linearities which in turn allows us to express stronger features of the input with fewer parameters. However, it is important to understand that this does come with a distinct memory allocation problem - especially when making use of the backpropagation algorithm.

The input layer should be recursively divisible by two. Common numbers in- include 32×32 , 64×64 , 96×96 , 128×128 and 224×224 .

Whilst using small filters, set stride to one and make use of zero-padding to ensure that the convolutional layers do not reconfigure any of the dimensions- quality of the input. The amount of zero-padding to be used should be calculated by taking one away from the receptive field size and dividing by two. activation

CNN's are extremely powerful machine learning algorithms, however, they can be horrendously resource-heavy. An example of this problem could be in filtering a large image (anything over 128×128 could be considered large), so if the input is 227×227 (as seen with ImageNet) and we're filtering with 64 kernels each with a zero padding of then the result will be three activation vectors of size $227 \times 227 \times 64$ - which calculates to roughly 10 million activations - or an enormous 70 megabytes of memory per image. In this case, you have two options. Firstly, you can reduce the spatial dimensionality of the input images by resizing the raw images to something a little less heavy. Alternatively, you can go against everything we stated earlier in this document and opt for larger filter sizes with a larger stride (2, as opposed to 1).

IV. CONCLUSION

Convolutional Neural Networks differ from other forms of Artificial Neural Networks in that instead of focusing on the entirety of the problem domain, knowledge about the specific type of input is exploited. This in turn allows for a much simpler network architecture to be set up. This paper has outlined the basic concepts of Convolutional Neural Networks, explaining the layers required to build one and detailing how best to structure the network in most image analysis tasks.

Research in the field of image analysis using neural networks has somewhat slowed in recent times. This is partly due to the incorrect belief surrounding the level of complexity and knowledge required to begin modelling these superbly powerful machine learning algorithms. The authors hope that this paper has in some way reduced this confusion, and made the field more accessible to beginners.



10.22214/IJRASET



45.98



IMPACT FACTOR:
7.129



IMPACT FACTOR:
7.429



INTERNATIONAL JOURNAL FOR RESEARCH

IN APPLIED SCIENCE & ENGINEERING TECHNOLOGY

Call : 08813907089  (24*7 Support on Whatsapp)