Convolutional Neural Networks: theory and application

Noemi Benci
noemi.benci@stud.unifi.it

Abstract Even though Artificial Neural Networks are becoming always more employed and studied in different fields, it is difficult to get an idea of how they really work. In this paper we present some theory about Convolutional Neural Networks and a practical point of view, using pictures and application to understand better the operations behind them. We focus on how Convolutional and Pooling Layers are necessary to transform images and how to build a CNN architecture. We do not deal with more complicated topics such as Optimisation Algorithms and other possible layers that can be added to the model. At the end we introduce a practical example, concerning image classification of Simpsons characters. We show that a simple CNN are able to solve difficult problems with great results.

Introduction Artificial Neural Networks take name from the brain structure, made by many neurons swapping each other information. In particular a neuron takes an input, elaborates it in some way and at the end sends an output to other neurons. This structure is reproduced by the Artificial Neural Networks (ANNs). These powerful tools learn how to classify object on images, how to predict outputs and how to capture information. To understand better we can think about the Neural Network as a vertical chain of connected layers composed by neurons. Each layer runs a specific task (i.e. they apply a specific function) and each neuron applies a specific transformation to the input to get an output. The output of any layer is then used as input in the following layer and so on.

The Figure 1 shows the general structure of an ANN, in which we can recognise three main element: the Input Layer, the one that takes the data; one (or more) Hidden Layer, in which the Network applies all the transformations; and the Output Layer, that gives us the result (vector of categories, of numbers, etc..).

This structure is run many times and at each steps the network improves the estimates of the weights that each connection has thank to an optimisation algorithm. Once the model has run a lot of times, it has (hopefully) found the best weight for each connection and these estimates can be applied to data never seen before.

Keeping in mind this pattern, we will focus on Convolutional Neural Network (CNN) structure, a specific Network used when data are signals, i.e the features are individual and highly correlated quantities. In particular we can employ this type of

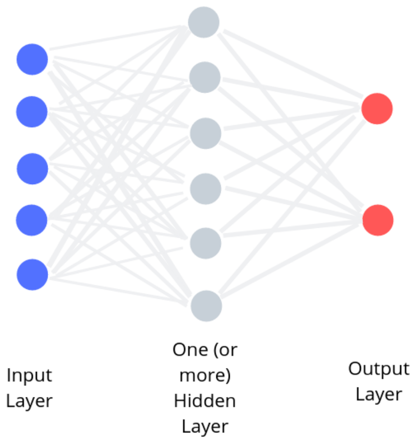

Figure 1. *General structure of an Artificial Neural Network.*

Network when we working with images or videos. Both object detection and image classification can be well computed with CNNs.

In this paper we start introducing some theory about Convolutional Neural Networks, pointing out the meaning of convolution and describing the main types of layers used in those types of Networks (i.e. Convolutional Layers, Pooling Layers and Dense Layers). Successively we present our application of this Network on the Simpsons Data Set provided by Alexattia 2017 on Kaggle. The Data Set contains many photos of 42 Simpsons characters and our goal is to correctly label each photo. With this application we want to clarify how to construct a CNN architecture and show how this Network actually works .

Convolutional Neural Network

Convolutional Neural Network is a specific network that allows to work with images in particular it can find object or distinguish two or more subjects that appear in an image or in a video.

These networks have the same general structure of ANNs but each layer is multidimensional and this difference allows this type of network to be the best tool to use with images. In fact every images can be seen as a multidimensional matrix in which height and width are the number of pixels of the image and the depth is the number of channels of the image. Each element of the matrix is a number that can assume value within the range $[0; 255]$. This number represent the value of the colour of the pixel.

Due to the fact that images are multiple matrices, Convolutional Networks use multidimensional connected layers that are able to reduce images, detect small patterns and at the end flatten these matrices to get a single vector. More precisely instead of connecting every input to the neurons in a given layer as happen in a generic Artificial Neural Network, CNNs shorten connections so that only one neuron accepts inputs from a small subsection of the layer before so that each neuron transforms a specific part of the image at time (Justin Johnson, n.d.).

The typical layers that compose a CNN are: Convolutional Layers, Pooling Layers and Fully Connected Layers. Each layer can appear more than once in the Network architecture.

To clarify how CNNs work, we explain every component and the function used in every layer.

Convolutional Layer

The most characteristic layer is the Convolutional one. This layer, as we can understand from the name, apply to a small part of the image, the convolution. This mathematical operation that is a specialised kind of linear operation (Goodfellow, Bengio, and Courville 2016). The convolution takes two argument, an input and a kernel, and gives as output the feature map. The generic form of the convolution is

$$s(t) = (x * w)(t) = \sum_{a=-\infty}^{\infty} x(a)w(t-a) \tag{1}$$

As we have already said, CNNs use multidimensional layers, so the arguments must have more than one dimension. In particular if we are working with two-dimensional images I and two-dimensional kernel K, then the formula is:

$$S(i,j) = (I * K)(i,j) = \sum_m \sum_n I(m,n)K(i-m, j-n) \tag{2}$$

Many libraries change this formula getting one of the equations 3 or 4 to make computational calculations easier, but without changing the substantial convolution operation.

$$S(i,j) = (I * K)(i,j) = \sum_m \sum_n I(i-m, j-n)K(m,n) \tag{3}$$

$$S(i,j) = (I * K)(i,j) = \sum_m \sum_n I(i+m, j+n)K(m,n) \tag{4}$$

Convolution can be seen as multiplication of matrices as shown in the Fig. 2. In particular each pixel in the input is multiplied to the kernel and the sum of each value compose one new pixel of the feature map. Then the final output is an image with reduced dimensions.

To underline the task of this layer, it is often called as filter. In fact it filters the image, detecting important features of the image. We add an example of the appearance of an image in which is applied the convolution in Fig. 3. We can see

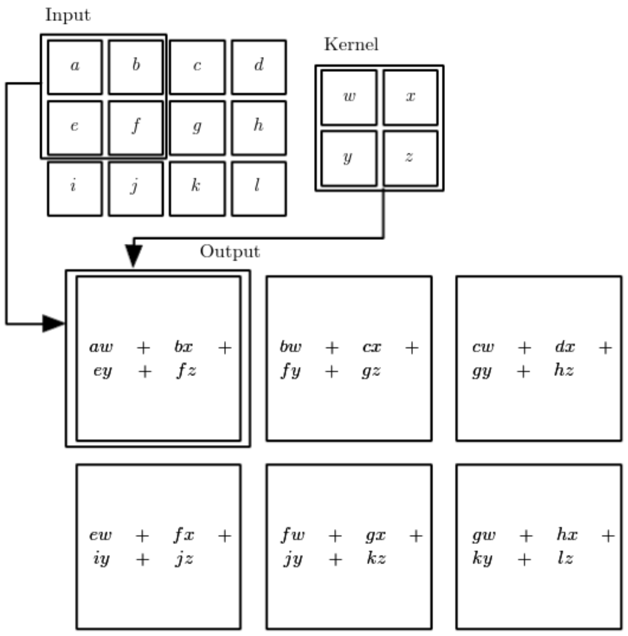

Figure 2. *Representation of convolution operation taken from the book of Goodfellow, Bengio, and Courville 2016. Applying convolution to a 3x4 image and 2x2 kernel, we get as output a 2x3 transformed image.*

that each filter changes the image, like a filter does. In a Convolutional Layer the Network change the weights in the kernel, looking for the best ones to catch as much information as possible.

At the end of the Convolutional Layer, it is applied a Non-Linear Activation function. This function transforms the output adding non-linearity to the output image and this brings to a better generalisation of the output. The most used is the ReLU (Rectified Linear Unit) Activation function (Nair and Hinton 2010 for more details).

Pooling Layer

Immediately after a Convolutional Layer, we can find a Pooling Layer. The pooling function replaces a specific area of the output coming from the previous layer, with a summary statistics, e.g. maximum, average, minimum and so on.

This step is very useful to reduce the image dimensions and, more important, to make the representation approximately invariant to small translations of the input. This means that if we have many little translations we would get the same outputs. This is very important to generalize better so that we can reduce overfitting.

The most used statistic is maximum, from which derives the name Max Pooling

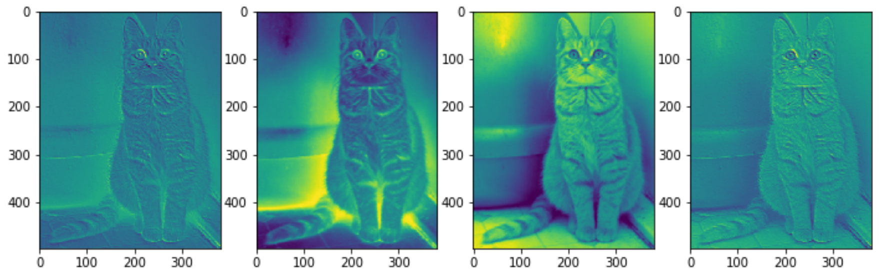

Figure 3. *Example of convolution. In figure we applied four different random kernels of size 3x3 to the image. The convolution applied to a coloured image transforms it in a green and yellow image, reducing the dimension of the image.*

Layer. It simply takes a square in an image (i.e. areas of dimension 2x2, 3x3, 4x4, etc) and returns the greatest value in that area. In this way a 16x16 matrix can be reduced, with a 4x4 size of pooling, to a 4x4 matrix with a stride of 2. The stride represents how many pixels the square jumps from an area to the next one. We present an example in Fig. 4.

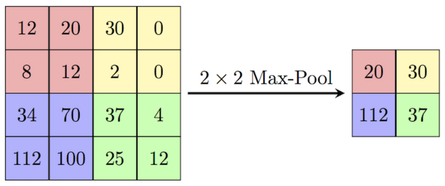

Figure 4. *Example of Max-Pooling.*

Fully Connected Layer

CNN combines many times Convolutional and Pooling Layers together and at the end the Network flattens the output and applies one or more Fully Connected Layers.

This type of layer has connections to every output in the previous layer. It applies a function such as the Softmax with the aim of classifying the input (in our case the image) into a label (in a simple classification example). The Softmax gives as output a vector of values, representing the probability that an image belongs to a category. At the end the highest probability gives the label to the image.

CNN architecture

Any type of Neural Network has a very complex structure, that gives the name to the well-known "Deep Learning". This means that the architecture of a CNN is made by many different overlapped layers. The more layers the network has, the more it is able to learn more detailed and more abstract relationships within the data.

The most common way to build a CNN architecture is to repeat many times Convolutional and Pooling Layers to make the image smaller and more abstract. A common architecture is :

`INPUT→CONV (5x5)→MAX POOL (2X2)→CONV (5x5) → MAX POOL (2X2)→`
`→FLATTEN → FULLY CONN→ OUTPUT`

Then we can change this structure as we need. For example we can add more layers, we can alternate two consecutive Convolutional Layers and two Pooling Layers and so on. Usually is advisable to begin the Network with small and few filters in convolutional layers (often with odd sizes i.e. (3x3), (5x5), (7,7)...) and gradually increase the dimensions and the number of filters. This helps to reduce overfitting. Another important tool to reduce overfitting is Data Augmentation Krizhevsky, Sutskever, and Hinton 2012. This is about flipping, cropping and rotating an image to represent the same subject in different ways. We can see an application of this tool in Fig. 5.

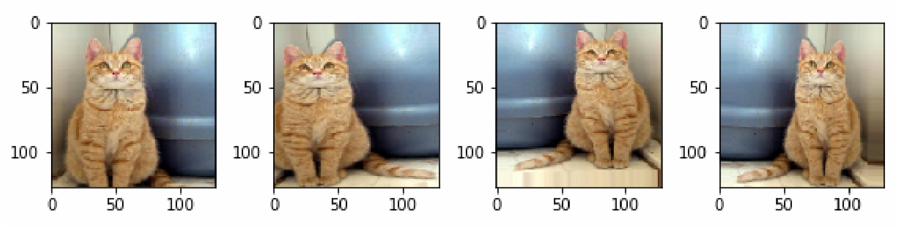

Figure 5. *Example of Data Augmentation.*

Practical example

In this section we present our application of a Convolutional Neural Network. We apply most of theory and tools previously explained to get the best model. We use the Simpsons Data Set provided by Alexattia 2017 on Kaggle. The data set is composed by 16746 images in the training set and 4187 in the test set. The images show 42 different characters of the American animated sitcom created by Matt Groening. The number of images for each character is not fixed. In Fig. 6 we display the number of images provided for each character. The goal of the example is to correctly label every image in the Test Set using the model weights trained on different images belonging to the Training Set. To prepare data and run the model we used `Tensorflow`, `Keras` and

`Sklearn`, three Python libraries that allow to build the ANN architecture combining the component as needed.

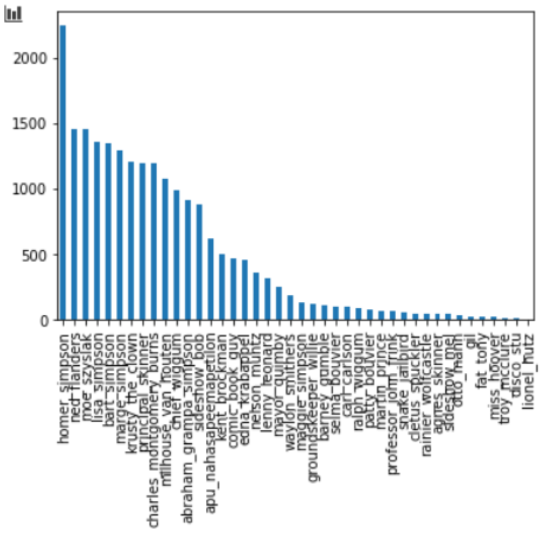

Figure 6. *Bar plot showing the number of images for each characters.*

Model Description

Before feeding the model on the data is necessary to build the Training and the Validation Set. These two sets are parts of the Training Data Set provided by Kaggle. It is important to validate the model during the execution to control if the model is doing well, if it is overfitting or it is not learning. To build these sets we used the `Sklearn` function called `train_test_split` that, given a specific percentage of validation images, randomly choose some images to be part of the validation set and the other of the train. In our example we use 3349 images for the Validation and 13397 as Training set.

We augment data to reduce overfitting, so we created 15 different images flipping and rotating each image. Once processed the data we decide the model to use.

After few attempts we decide to use the model shown in Fig. 7.

The model is a sequence of Convolutional and Max Pooling Layers which are applied three times in the network.

Convolutional Layers increase every time the number of filters applied (16, 32 and 64) keeping constant the size of the filters, that is always 3x3. The Max Pooling Layer consist in a 2x2 size reduction every time that is applied.

After these layers, we include two Dense (or Fully Connected) Layers. The first one presents 512 units and the second has many units as the number of characters that

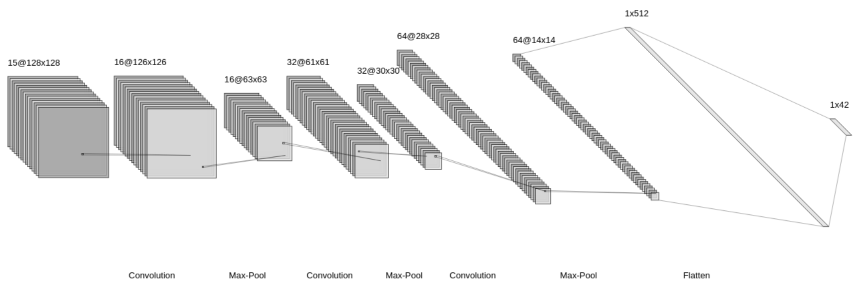

Figure 7. *Architecture of the model used to classify Simpsons characters made with a tool available on the website* `http://alexlenail.me/NN-SVG/LeNet.html`

we want to recognize, i.e. 42 units.

In this last layer is applied the softmax activation function because the aim of this layer is to classify the subjects represented in the images. Then we run the model for 20 epochs, using the Adam as optimisation algorithm and the Categorical Cross-Entropy as loss function.

Result

In Fig. 8 we can see the accuracy at the end of every epoch. From this figure we can see that the model is doing well in terms of accuracy. It overfits a bit in the last epochs, but the final result on the test set is incredibly good. Then we use the model

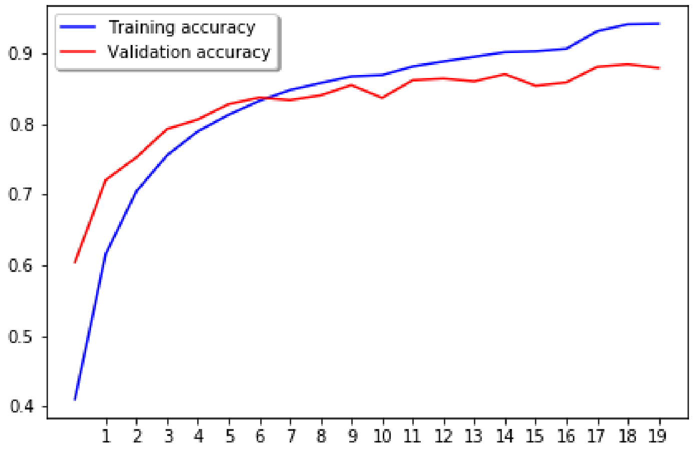

Figure 8. *Accuracy of every epoch of the model.*

weights to predict the characters shown in the Test Set. This latter Set is composed by 990 images of 20 Simpsons characters. We show in Table 1 the accuracy on train, validation and in the test set. The results are pretty good, taking into account that the model is quite simple.

Train accuracy	0.9417
Validation accuracy	0.8792
Test accuracy	0.8657

Table 1. *Train, validation and test accuracy at the last epoch.*

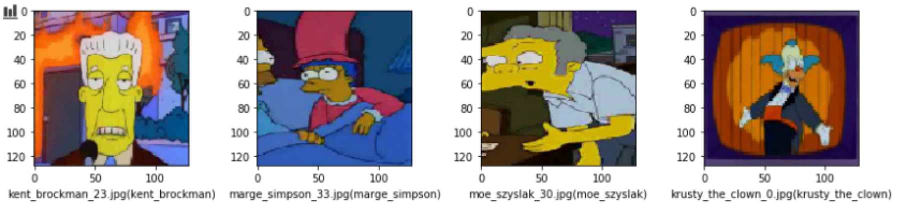
0
20
40
60
80
100
120
0
50
100
kent_brockman_23.jpg(kent_brockman)
marge_simpson_33.jpg(marge_simpson)
moe_szyslak_30.jpg(moe_szyslak)
krusty_the_clown_0.jpg(krusty_the_clown)

Figure 9. *Some test images labelled by the model.*

Concluding We got into details of Convolutional Neural Networks both in theoretical and practical points of view. We explained briefly the main layer of a CNN and its aim, such as the Convolutional and the Pooling Layers. We applied this tool on the Simpsons data set to classify many images of 42 Simpsons characters and the results were almost good. The model needs further insights in choosing hyperparameters in order to improve performance and reduce overfitting.

References

Alexattia. 2017. "The simpson Characters Dataset." Accessed June 15. `https://www.kaggle.com/alexattia/the-simpsons-characters-dataset`.

Goodfellow, Ian, Yoshua Bengio, and Aaron Courville. 2016. *Deep learning.* Chap. 9, 326–366. MIT press.

Justin Johnson, Andrej Karpathy. n.d. "Convolutional Neural Networks (CNNs / ConvNets)." `http://cs231n.github.io/convolutional-networks/`.

Krizhevsky, Alex, Ilya Sutskever, and Geoffrey E Hinton. 2012. "Imagenet classification with deep convolutional neural networks." In *Advances in neural information processing systems,* 1097–1105.

Nair, Vinod, and Geoffrey E Hinton. 2010. "Rectified linear units improve restricted boltzmann machines." In *Proceedings of the 27th international conference on machine learning (ICML-10),* 807–814.