Report 1: Basics of Convolutional Neural Nets and Tensorflow

Adithya Narayan

Manipal, Karanataka

Abstract

This is the first of four reports covering the topics necessary for the mandatory industrial training. This particular report will cover the basics of Neural Networks and Convolutional Neural Networks.

As a summary, by the date of the submission of this report, the basics of CNNs were gone over having completed four of the five course specialization on DeepLearning offered by DeepLearning.ai. In particular the fundamentals of different layers in a typical CNN and their mathematics was gone over.

In terms of CNN architectures, the architectures behind some classic CNNs were gone over. This includes the VGGNet, ResNet and InceptionNet.

Furthermore, the basics of building a model using Keras was also gone over.

Keywords: Tensorflow, CNN, Semantic Segmentation

1. Introduction to CNNs

Convolutional Neural Net(CNN) is a class of deep learning networks. For a typical neural network, inputs between layers are often one dimensional vectors for each training example[1]. When it comes to image datasets with thousands of pixels per image dimension, a typical Deep Neural Net becomes inefficient quite quickly[2]. CNNs however, allows for layers in the network to be multi-dimensional. Additionally, CNNs offer a fixed number of weights that is independent of the number of input pixels and only depends on the filter dimensions and the number of filters used[1].

1.1. Architecture

Typical CNNs consist of a series of Convolutional Layers followed by Pooling Layers[3]. Once this has been done to abstract a sufficient number of features from the image, the output is 'flattened' and passed into a fully connected layers followed by a logistic unit(for binary classification) or a softmax layer for the final output[3].

The image below shows and example of a typical CNN.

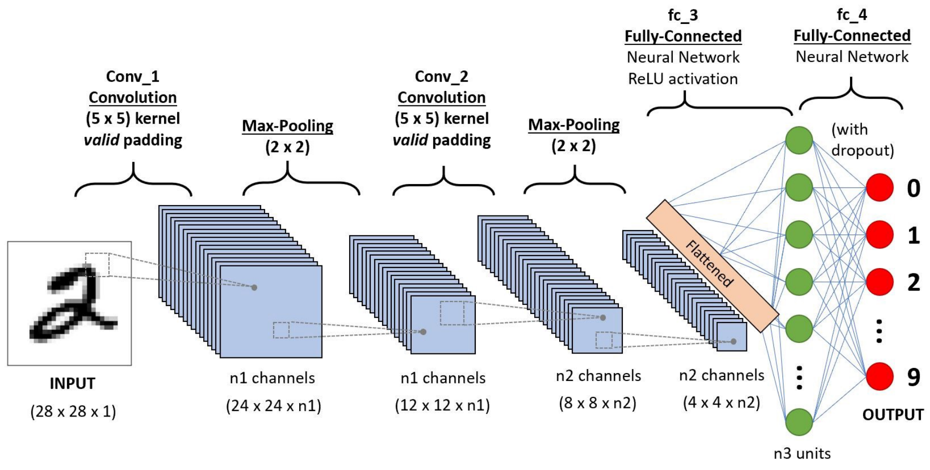

Figure 1: A generic CNN

1.2. Layers Of a CNN

While there exist a number of different architectures in CNNs, there are a few fundamental building blocks that are present in most CNNs. These are expanded upon in the following sections. Additionally, while the term "layer" is loosely used in this report, general literature is fairly inconsistent when it comes to the definition of a layer. Certain papers consider a CONV+POOL as a single layer whereas others consider these to be separate layers. In this report, they have been taken as separate layers.

1.2.1. Convolutional Layer

While generally called a convolutional layer, most architectures use cross-correlation as opposed to convolution(mostly because it does not make a difference). Hence, in this layer, a series of 2D filters are cross-correlated/convolved with the input from the previous layer[4]. Each filter in this layer, generates a 2D matrix as it output, and outputs of all the filters are stacked together to generate the initial output[4]. The outputs of the filters have a bias vector added and are then passed into an activation function(such as leaky ReLu). This gives the final output of the Convolutional Layer.

A typical convolution process leads to a drop in the size of the ouput of the layer when compared to the original input. Often, the data between layers are padded to ensure same size[3].

1.2.2. Pooling

In the pooling layer, the outputs of the convolutional layer and is passed through another filter. This filter varies across architecture but generally, it extracts the most important features within a region defined by filter[5]. For instance, in the case of max-pooling, it takes an $f * f$ window with stride s and outputs the max value within the window. This typically, downsamples outputs between convolutional layers(making it more efficient) while still retaining important features.

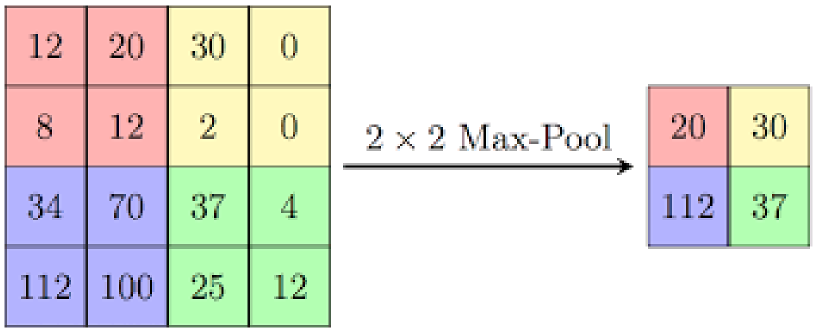

Figure 2: Max Pooling with a $2 * 2$ window

1.2.3. Fully Connected Layer

Typically at the final stages of a CNN the multi-dimensional output form a Pooling layer is flattened into a vector and passed into a Fully Connected Layer[6]. Every neuron in this layer is connected to every neuron in the previous layer with a weight matrix to define the connections. These layers function as a normal Deep Neural Net would.

1.2.4. Sigmoid/Softmax Layer

Typically found at the very end of a standard CNN, this layer performs the final classification of the input image.This layer can use a sigmoid function(when looking for probabilities between $0 - 1$) or a softmax layer for classification[6]. Softmax is generally used when there is more than one output class[7].

2. Semantic Segmentation

Semantic segmentation is a from of dense image segmentation. In effect, networks which perform semantic segmentation classify each pixel in the image as one of the output classes[8]. This allows you to generate image masks with which various operation can be performed on particular segmented objects classes within the image. Interestingly, since this is separate from instance segmentation, it cannot differentiate between different instances of the same class[8]. An example of semantic segmentation can be seen below.

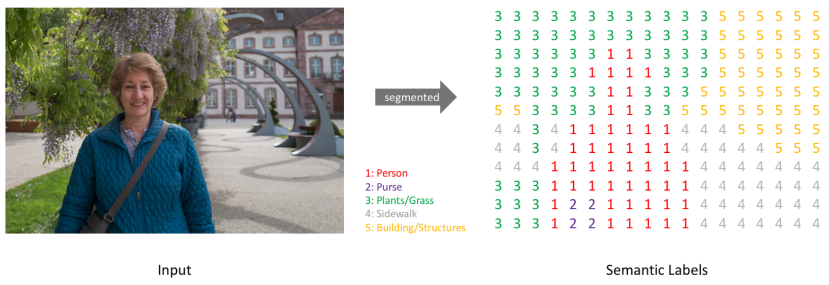

Figure 3: Example of Semantic Segmentation

Typically, CNNs which perform semantic segmentation use an encoder-decoder architecture[9]. CNNs lose resolution deeper in the network due to the convolution operation. Hence, while a classic CNN can be used to classify pixels in the image, the borders between objects will be fuzzy[4]. This is solved in the decorder component of the Encoder-Decoder architecture. As can be seen below, the layers are upsampled using transposed convolution before classification.

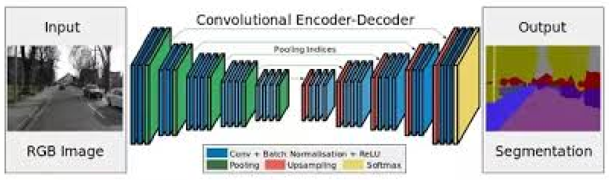

Figure 4: Example of an Encoder-Decoder Architecture

3. Tensorflow with Keras

While Tensorflow allows for more complex models to be made ground up, Keras is a high level API built on a Tensorflow back-end that allows for the quick prototyping of Convolutional Neural Networks. Important function for building a CNN is listed below.

- $Conv2D()$ - Implements a convolutional block. You can input the filter size, number of filters, stride and and applies it to the input activation. It returns a multi-dimensional tensor.
- $Activation()$ - Selects the activation function to pass the outputs of the CONV layer through.
- $MaxPooling2D$ - Implements a pooling layer. It accepts the pooling window size as well as stride as inputs.
- $Flatten()$ - Flattens a multi-dimensional tensor into vector. This vector is then fed into a fully-connected layer.
- $Dense()$ - Creates a fully-connected layer with a specified activation
- $model.compile()$ - Uses a specified loss function with a specified optimizer to compute optimal parameter values for the CNN.
- $model.fit()$ - Fits the compiled model to the training/dev set.

References

[1] Albawi, Saad Abed Mohammed, Tareq & ALZAWI, Saad. (2017). Understanding of a Convolutional Neural Network. 10.1109/ICEngTechnol.2017.8308186.

[2] http://www.cs.umd.edu/ djacobs/CMSC733/CNN.pdf

[3] https://cs.nju.edu.cn/wujx/teaching/15_CNN.pdf

[4] https://web.stanford.edu/class/cs231a/lectures/intro_cnn.pdf

[5] https://machinelearningmastery.com/pooling-layers-for-convolutional-neural-networks/

[6] https://cs231n.github.io/convolutional-networks/

[7] https://deepai.org/machine-learning-glossary-and-terms/softmax-layer

[8] https://courses.cs.washington.edu/courses/cse576/17sp/notes/Sachin_Talk.pdf

[9] https://d2l.ai/chapter_recurrent-modern/encoder-decoder.html