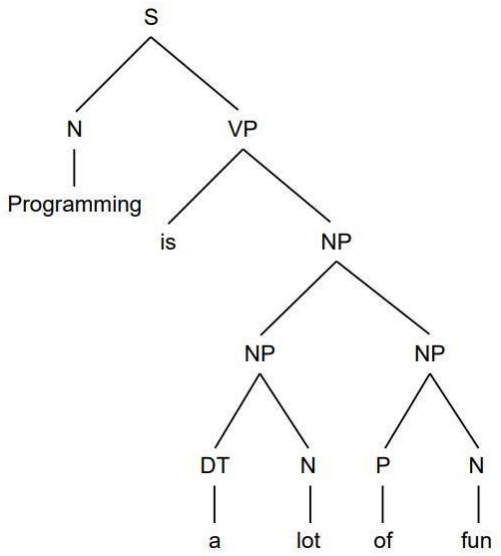
Syntactic Parsing

One of the main applications of RvNNs in NLP is the syntactic parsing of natural language sentences. When parsing a sentence, we want to identify its smaller components such as noun or verb phrases, and organize them in a syntactic hierarchy.

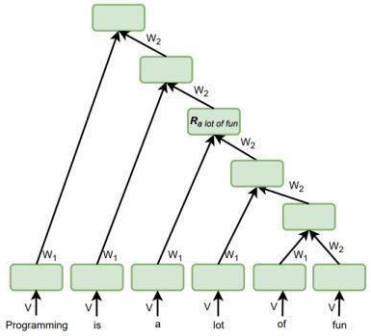


Since RNNs consider only sequential relations, they're less suitable for handling hierarchical data in contrast compared to RvNNs. Let's say that want to capture the representation of the phrase *a lot of fun* in this sentence:

Programming is a lot of fun.

What representation would an RNN give to this phrase? Since each state depends on the preceding words' representation, we can't represent a subsequence that doesn't start at the beginning of the sentence. So, when our RNN processes the word *fun*, the hidden state at that time step will represent the entire sentence.

In contrast, the hierarchical architecture of RvNNs can store the representation of the exact phrase:



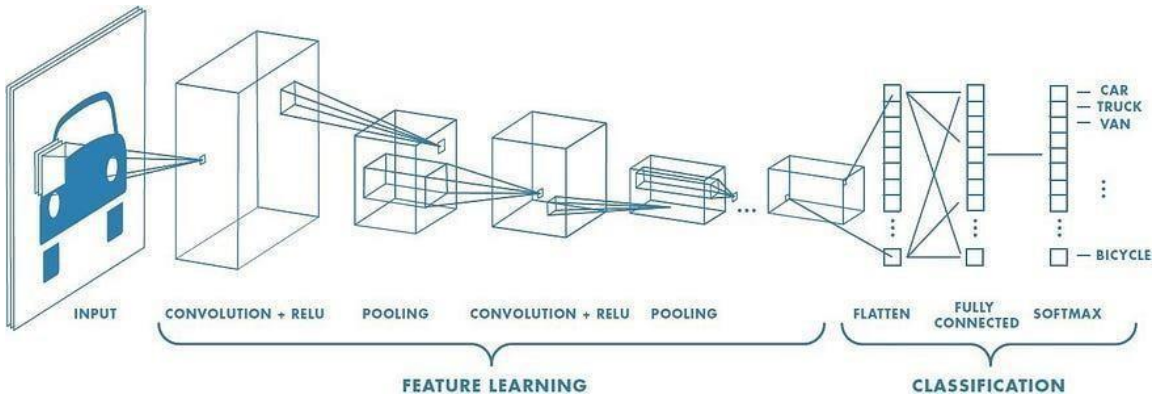
one hidden layer between the input at any time step and the corresponding output, there is a sense in which these networks are deep. Inputs from the first time step can influence the outputs at the final time step (often 100s or 1000s of steps later). These inputs pass through applications of the recurrent layer before reaching the final output. However, we often also wish to retain the ability to express complex relationships between the inputs at a given time step and the outputs at that same time step. Thus we often construct RNNs that are deep not only in the time direction but also in the input-to-output direction. This is precisely the notion of depth that we have already encountered in our development of MLPs and deep CNNs.

The standard method for building this sort of deep RNN is strikingly simple: we stack the RNNs on top of each other. Given a sequence of length the first RNN produces a sequence of outputs, also of length. These, in turn, constitute the inputs to the next RNN layer. In this short section, we illustrate this design pattern and present a simple example for how to code up such stacked RNNs. Below, in we illustrate a deep RNN with hidden layers. Each hidden state operates on a sequential input and produces a sequential output. Moreover, any RNN cell

(white box) in at each time step depends on both the same layer's value at the previous time step and the previous layer's value at the same time step.

Let's consider that we have access to multiple images of different vehicles, each labeled into a truck, car, van, bicycle, etc. Now the idea is to take these pre-label/classified images and develop a machine learning algorithm that is capable of accepting a new vehicle image and classify it into its correct category or label. Now before we start building a neural network we

need to understand that most of the images are converted into a grayscale form before they are process.



The challenge with images having multiple color channels is that we have huge volumes of data to work with which makes the process computationally intensive. In other worlds think of it like a complicated process where the Neural Network or any machine learning algorithm has to work with three different data (R-G-B values in this case) to extract features of the images and classify them into their appropriate categories.

The role of CNN is to reduce the images into a form that is easier to process, without losing features critical towards a good prediction. This is important when we need to make the algorithm scalable to massive datasets.

We understand that the training data consists of grayscale images which will be an input to the convolution layer to extract features. The convolution layer consists of one or more Kernels with different weights that are used to extract features from the input image. Say in the example above we are working with a Kernel (K) of size 3 x 3 x 1 (x 1 because we have one color channel in the input image), having weights outlined below.

Kernel/Filter, K =

1 0 1

0 1 0

1 0 1

When we slide the Kernel over the input image (say the values in the input image are grayscale intensities) based on the weights of the Kernel we end up calculating features for different pixels based on their surrounding/neighboring pixel values. E.g. when the Kernel is applied on the image for the first time as illustrated in Figure 5 below we get a feature value equal to 4 in the convolved feature matrix as shown below.

If we observe Figure 4 carefully we will see that the kernel shifts 9 times across image. This process is called **St**ride**.** When we use a stride value of 1 (Non-Strided) operation we need 9 iterations to cover the entire image. The CNN learns the weights of these Kernels on its own**.** The result of this operation is a feature map that basically detects features from the images rather than looking into every single pixel value.

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.

- A similar process is followed for scene labeling as well.

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.

1. 3. Collecting Historic and Environmental Elements

CNNs are also used for more complex purposes such as natural history collections. These collections act as key players in documenting major parts of history such as biodiversity, evolution, habitat loss, biological invasion, and climate change.

2. 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.

3. Understanding Gray Areas

Introduction of the gray 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 gray. Allowing the machine to understand and process fuzzier logic will help it understand the gray area we humans live in and strive to work against. This will help CNNs get a more holistic view of what human sees.

4. Advertising

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

5. Other Interesting Fields

CNNs are poised to be the future with their introduction into driverless cars, robots that can mimic human behavior, aides to human genome mapping projects, predicting earthquakes and natural disasters, and maybe even selfdiagnoses 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.