VIETNAM GENERAL CONFEDERATION OF LABOR

TON DUC THANG UNIVERSITY

FACULTY OF INFORMATION TECHNOLOGY

PROJECT

INTRODUCTION TO MACHINE LEARNING

FINAL PROJECT

Authors: **TRINH BAO TOAN – 520K0332**

NGUYEN CONG MINH THANH – 520K0215

Class**: 20K50301**

Admission Course**: 24**

Supervisor: **Prof. LE ANH CUONG**

HO CHI MINH CITY, 2022

VIETNAM GENERAL CONFEDERATION OF LABOR

TON DUC THANG UNIVERSITY

FACULTY OF INFORMATION TECHNOLOGY

PROJECT

INTRODUCTION TO MACHINE LEARNING

FINAL PROJECT

Authors: **TRINH BAO TOAN – 520K0332**

NGUYEN CONG MINH THANH – 520K0215

Class**: 20K50301**

Admission Course**: 24**

Supervisor: **Prof. LE ANH CUONG**

HO CHI MINH CITY, 2022

ACKNOWLEDGEMENTS

We are really grateful because we managed to complete our report assignment about Introduction To Machine Learning within the time given. We sincerely thank our lecturer, Mr. Le Anh Cuong for the guidance and encouragement in finishing this assignment and also teaching us in this course.

PROJECT COMPLETED AT TON DUC THANG UNIVERSITY

We hereby declare that this is our own project and is under the guidance of Mr. Le Anh Cuong. The research contents and results in this topic are honest and have not been published in any publication before. The data in the tables for analysis, comments and evaluation are collected by the author himself from different sources, clearly stated in the reference section.

In addition, the project also uses a number of comments, assessments as well as data of other authors, other agencies and organizations, with citations and source annotations.

If we find any fraud, we will take full responsibility for the content of our project. Ton Duc Thang University is not related to copyright and copyright violations caused by us (if any).

Ho Chi Minh City, 16th October , 2022

Authors

(signature and full name)

Nguyen Cong Minh Thanh

Trinh Bao Toan

LECTURER'S ASSESSMENT

Ho Chi Minh, date

(sign)

ABSTRACT

CONTENT

LIST OF ABBREVIATION

LIST OF FIGURE

LIST OF TABLE

PROBLEM 1 AND 2

I. Describe dataset:

II. Preprocessing

1. Loading Initial Libraries

```
import pandas as pd
import numpy as np
```

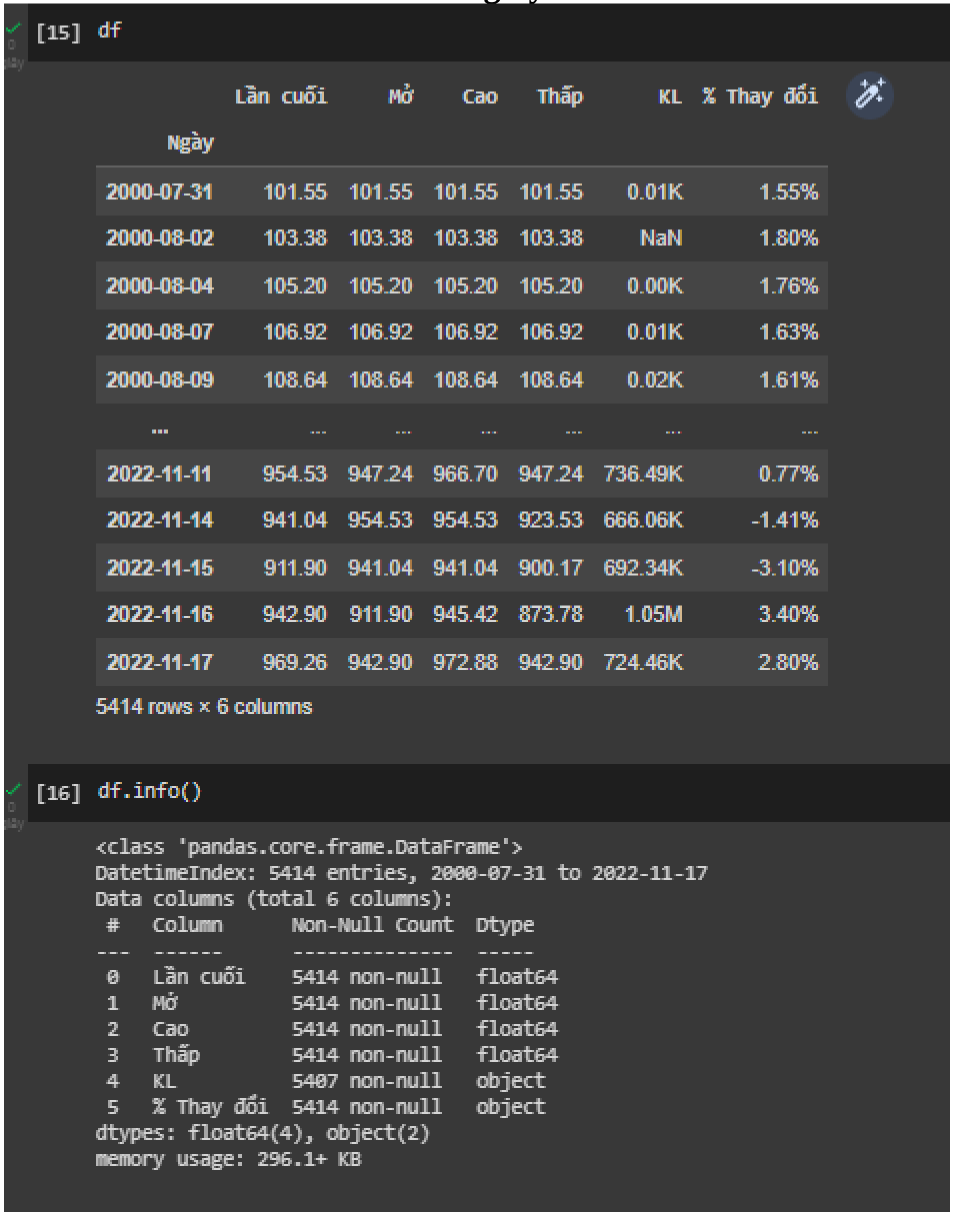
2. Importing dataset

```
df = pd.read_csv('Dữ liệu Lịch sử VN Index.csv')
```

3. Change date column to right format

```
df.index = pd.to_datetime(df.index, format='%d/%m/%Y')
```

4. Sort the dataset increasing by date => the latest value will locate at the end



```
[15] df
```

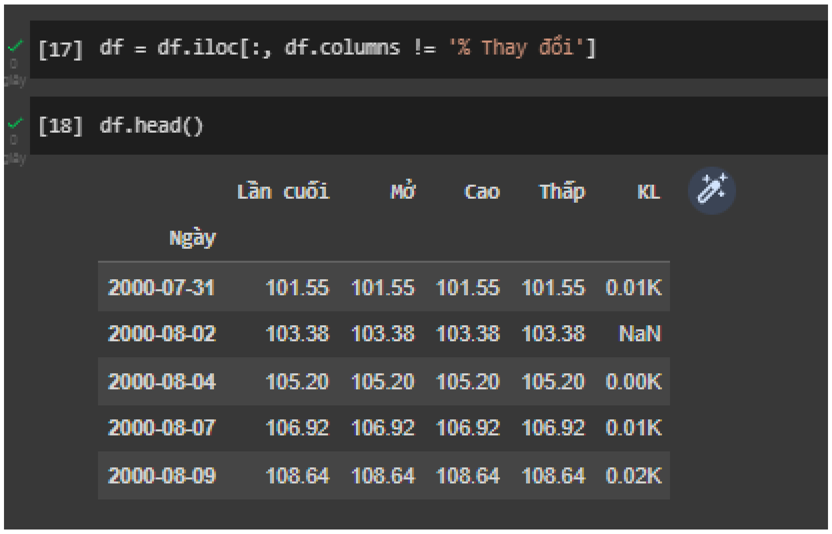
Ngày	Lần cuối	Mở	Cao	Thấp	KL	% Thay đổi
2000-07-31	101.55	101.55	101.55	101.55	0.01K	1.55%
2000-08-02	103.38	103.38	103.38	103.38	NaN	1.80%
2000-08-04	105.20	105.20	105.20	105.20	0.00K	1.76%
2000-08-07	106.92	106.92	106.92	106.92	0.01K	1.63%
2000-08-09	108.64	108.64	108.64	108.64	0.02K	1.61%
...	...	...	...	...	...	...
2022-11-11	954.53	947.24	966.70	947.24	736.49K	0.77%
2022-11-14	941.04	954.53	954.53	923.53	666.06K	-1.41%
2022-11-15	911.90	941.04	941.04	900.17	692.34K	-3.10%
2022-11-16	942.90	911.90	945.42	873.78	1.05M	3.40%
2022-11-17	969.26	942.90	972.88	942.90	724.46K	2.80%

5414 rows × 6 columns

```
[16] df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
DatetimeIndex: 5414 entries, 2000-07-31 to 2022-11-17
Data columns (total 6 columns):
 #   Column      Non-Null Count  Dtype
---  ------      --------------  -----
 0   Lần cuối    5414 non-null   float64
 1   Mở          5414 non-null   float64
 2   Cao         5414 non-null   float64
 3   Thấp        5414 non-null   float64
 4   KL          5407 non-null   object
 5   % Thay đổi  5414 non-null   object
dtypes: float64(4), object(2)
memory usage: 296.1+ KB
```

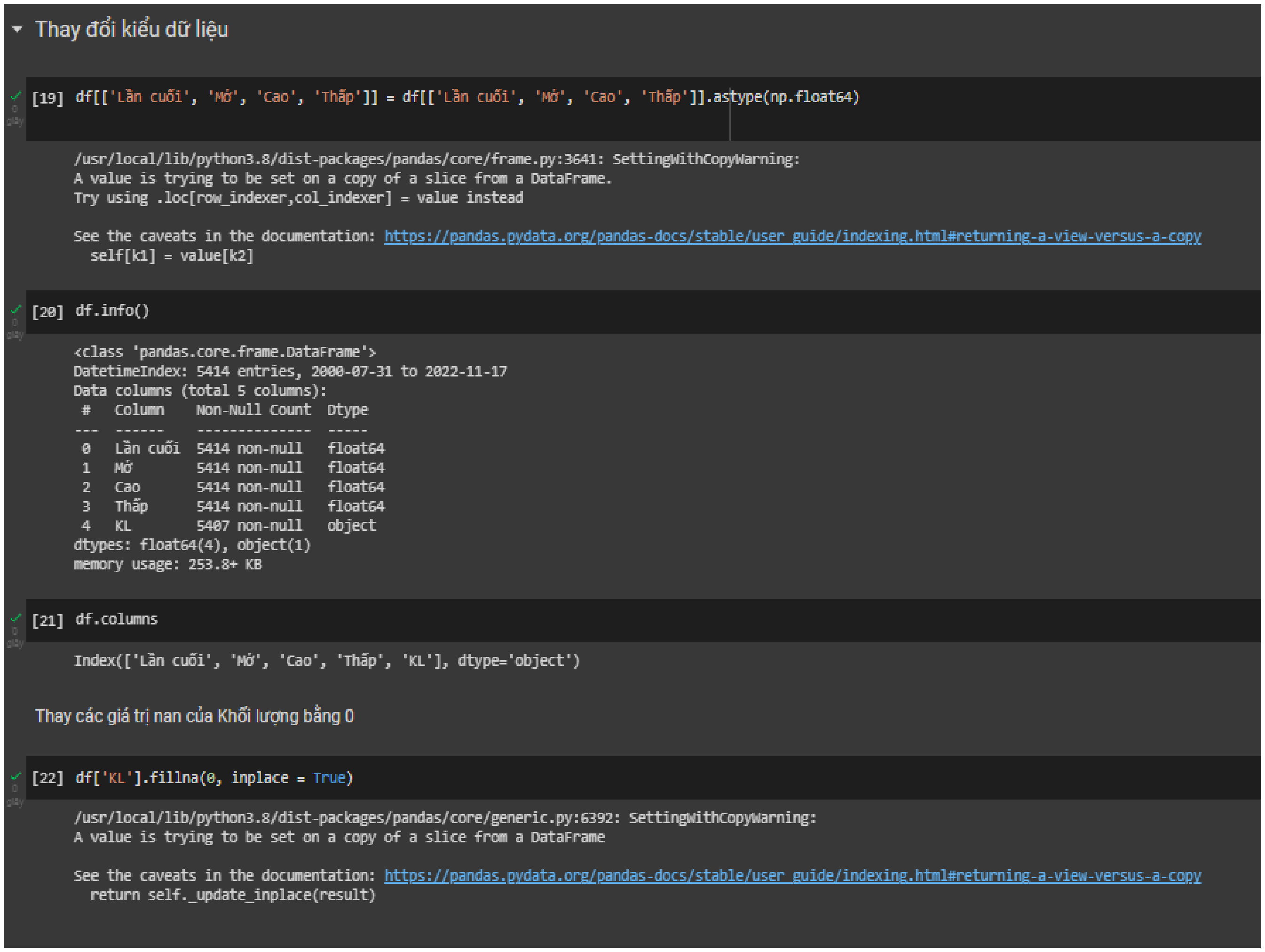
5. Normalization



```
[17] df = df.iloc[:, df.columns != '% Thay đổi']
```

```
[18] df.head()
```

Ngày	Lần cuối	Mở	Cao	Thấp	KL
2000-07-31	101.55	101.55	101.55	101.55	0.01K
2000-08-02	103.38	103.38	103.38	103.38	NaN
2000-08-04	105.20	105.20	105.20	105.20	0.00K
2000-08-07	106.92	106.92	106.92	106.92	0.01K
2000-08-09	108.64	108.64	108.64	108.64	0.02K



Thay đổi kiểu dữ liệu

```
[19] df[['Lần cuối', 'Mở', 'Cao', 'Thấp']] = df[['Lần cuối', 'Mở', 'Cao', 'Thấp']].astype(np.float64)

     /usr/local/lib/python3.8/dist-packages/pandas/core/frame.py:3641: SettingWithCopyWarning:
     A value is trying to be set on a copy of a slice from a DataFrame.
     Try using .loc[row_indexer,col_indexer] = value instead

     See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy
       self[k1] = value[k2]
```

```
[20] df.info()

     <class 'pandas.core.frame.DataFrame'>
     DatetimeIndex: 5414 entries, 2000-07-31 to 2022-11-17
     Data columns (total 5 columns):
      #   Column    Non-Null Count  Dtype
     ---  ------    --------------  -----
      0   Lần cuối  5414 non-null   float64
      1   Mở        5414 non-null   float64
      2   Cao       5414 non-null   float64
      3   Thấp      5414 non-null   float64
      4   KL        5407 non-null   object
     dtypes: float64(4), object(1)
     memory usage: 253.8+ KB
```

```
[21] df.columns

     Index(['Lần cuối', 'Mở', 'Cao', 'Thấp', 'KL'], dtype='object')
```

Thay các giá trị nan của Khối lượng bằng 0

```
[22] df['KL'].fillna(0, inplace = True)

     /usr/local/lib/python3.8/dist-packages/pandas/core/generic.py:6392: SettingWithCopyWarning:
     A value is trying to be set on a copy of a slice from a DataFrame

     See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy
       return self._update_inplace(result)
```

PROBLEM 3

I. Introduction of three models

1. Recurrent Neural Network (RNN)

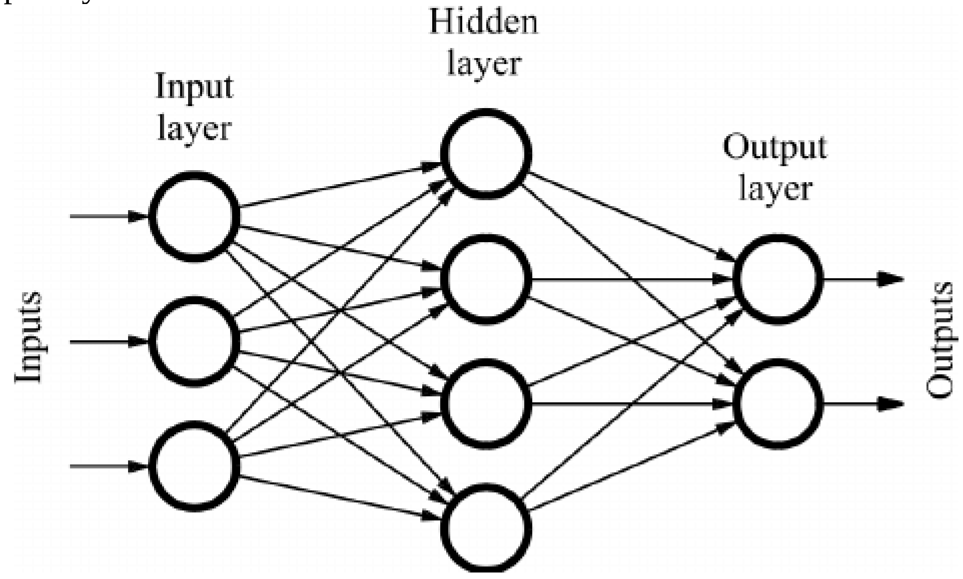
1.1.What are recurrent neural networks?

Recurrent neural networks (RNNs) are the state of the art algorithm for sequential data and are used by Apple's Siri and Google's voice search. It is the first algorithm that remembers its input, due to an internal memory, which makes it perfectly suited for machine learning problems that involve sequential data. It is one of the algorithms behind the scenes of the amazing achievements seen in deep learning over the past few years. In this post, we'll cover the basic concepts of how recurrent neural networks work, what the biggest issues are and how to solve them. One of the most frequent types of artificial neural networks is called a recurrent neural network. It is commonly used for automatic voice recognition and machine translation (RNN). It is possible to forecast the most likely future situation utilizing patterns in sequential data by employing recurrent neural networks. Recurrent neural networks (RNN) make it easier to model sequence data. RNNs, which come from feedforward networks, act in a way that is similar to how human brains do. In other words, recurrent neural networks can predict sequential data in ways that other algorithms can't.

1.2.Recurrent Neural Network vs. Feedforward Neural Network

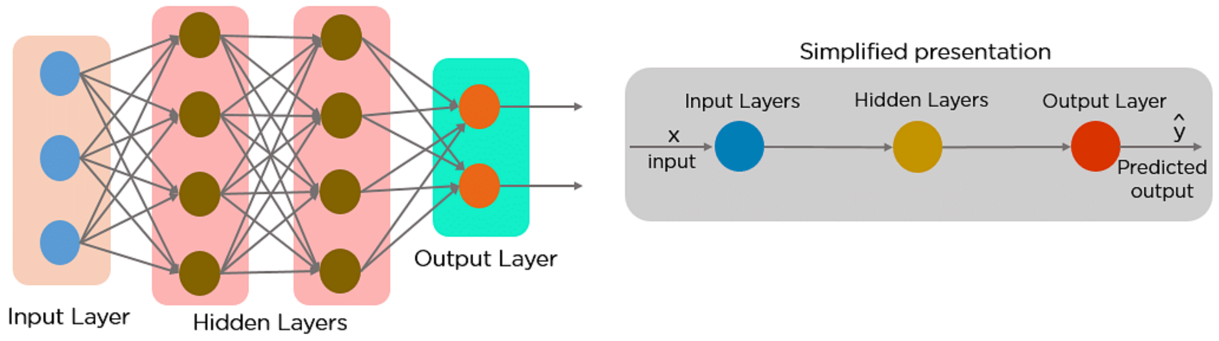
The Feedforward Neural Network

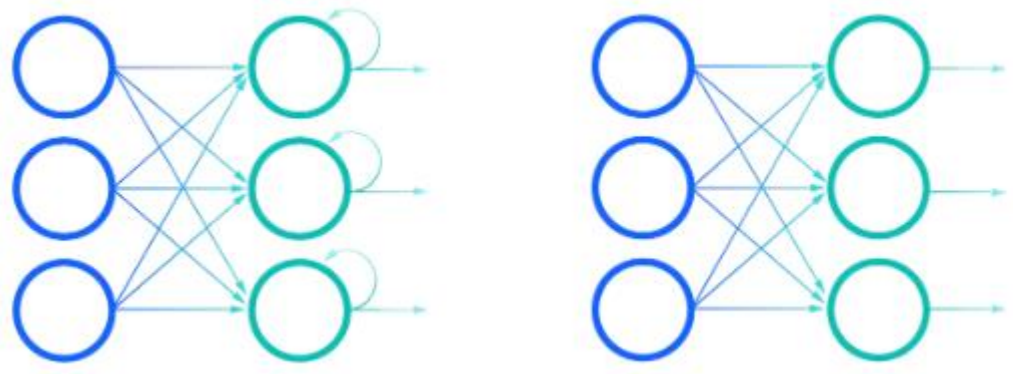
A feed-forward neural network (FFN) is a single-layer perceptron in its most fundamental form. Components of this network include the hidden layer, output layer, and input layer.



In the above image, the neural network has input nodes, output nodes, and hidden layers. Due to the absence of connections, information leaving the output node cannot return to the network. As the name of the network suggests, the information goes in only one direction.

- The input layer comprises neurons that receive input.
- The hidden layer contains a large number of neurons that modify the inputs and interact with the output layer.
- The output layer contains the result of the computation.

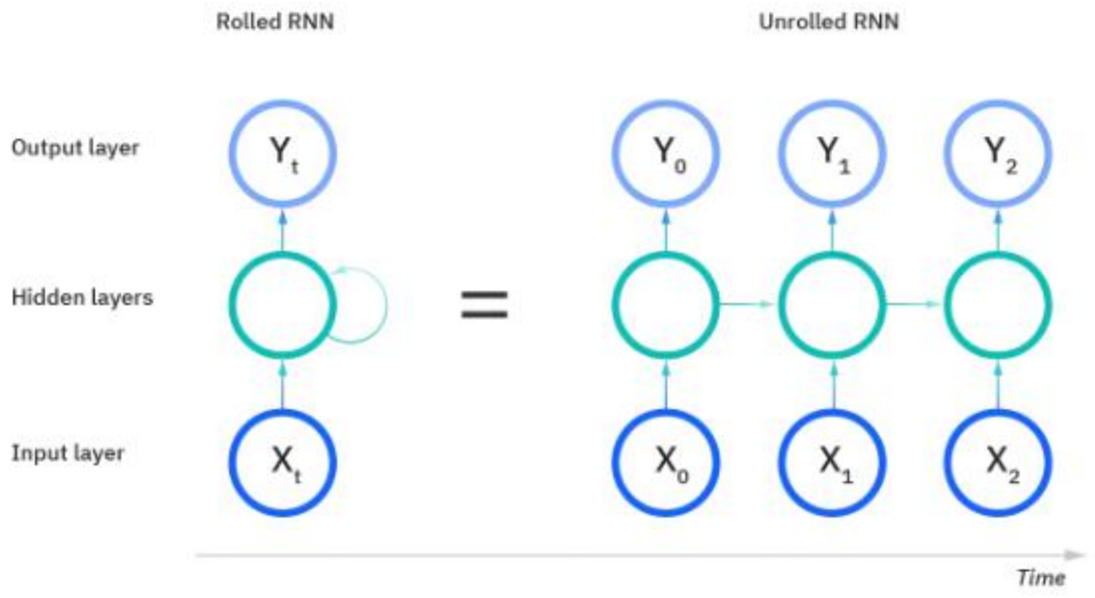




In a feed-forward neural network, the decisions are based on the current input. It doesn’t memorize the past data, and there’s no future scope. Feed-forward neural networks are used in general regression and classification problems.
Comparison of Recurrent Neural Networks (on the left) and Feedforward Neural Networks (on the right)

- Let us use an idiom, such as "feeling under the weather," which is commonly used when someone is sick, to help us understand RNNs. The idiom must be expressed in that specific order for it to make sense. As a result, recurrent networks must account for the position of each word in the idiom before predicting the next word in the sequence.
- Looking at the image below, the RNN's "rolled" visual represents the entire neural network, or rather the entire predicted phrase, such as "feeling under the weather." The "unrolled" visual represents the neural network's individual layers, or time steps. Each layer corresponds to a single word in that phrase, for

example, "weather." In the third timestep, prior inputs such as "feeling" and "under" would be represented as a hidden state to predict the output in the sequence, "the."

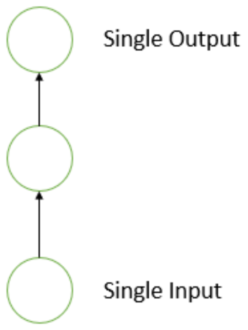


- Another distinguishing feature of recurrent networks is that they share parameters across all network layers. While feedforward networks have different weights for each node, recurrent neural networks share the same weight parameter across all layers. However, these weights are still adjusted in the backpropagation and gradient descent processes to facilitate reinforcement learning.
- To determine the gradients, recurrent neural networks use the backpropagation through time (BPTT) algorithm, which is slightly different from traditional backpropagation because it is specific to sequence data. BPTT works on the same principles as traditional backpropagation, in which the model trains itself by calculating errors from its output layer to its input layer. These calculations allow us to appropriately adjust and fit the model's parameters. BPTT differs from the traditional approach in that it sums errors at each time step, whereas feedforward networks do not have to sum errors because parameters are not shared across layers.
- RNNs frequently encounter two issues during this process: exploding gradients and vanishing gradients. The size of the gradient, which is the slope of the loss function along the error curve, defines these issues. When the gradient is too small, it continues to shrink, updating the weight parameters until they become insignificant—that is, 0—and then stops. When this happens, the algorithm stops learning. When the gradient is too large, it explodes, resulting in an unstable model. In this case, the model weights will become too large and will be represented as NaN. One approach to addressing these issues is to reduce the

number of hidden layers within the neural network, thereby removing some of the complexity in the RNN model.

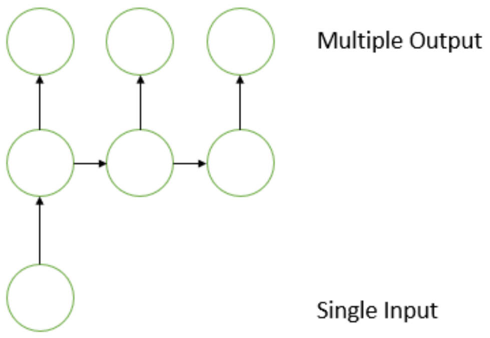
1.3.Types of recurrent neural networks

1.3. 1. One-to-One RNN:



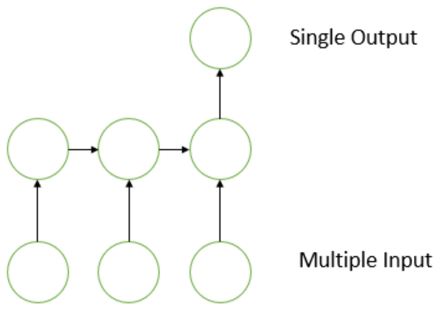
The above diagram represents the structure of the Vanilla Neural Network. It is used to solve general machine learning problems that have only one input and output.

1.3. 2. One-to-Many RNN:



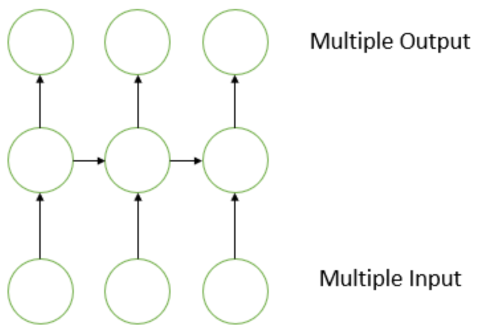
A single input and several outputs describe a one-to-one Recurrent Neural Network. The above diagram is an example of this.

1.3 3. Many-to-One RNN:



This RNN creates a single output from the given series of inputs.

1.3. 4. Many-to-Many RNN:

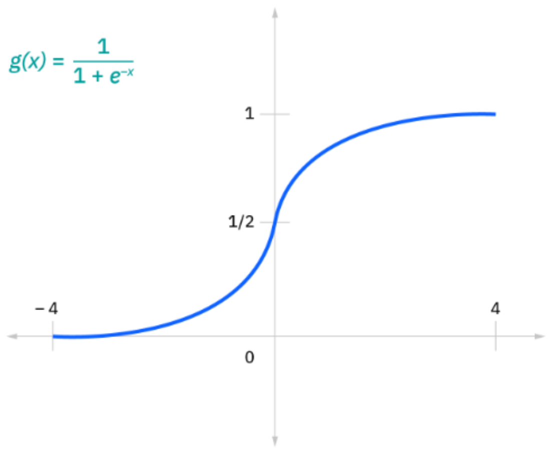


This RNN receives a set of inputs and produces a set of outputs.

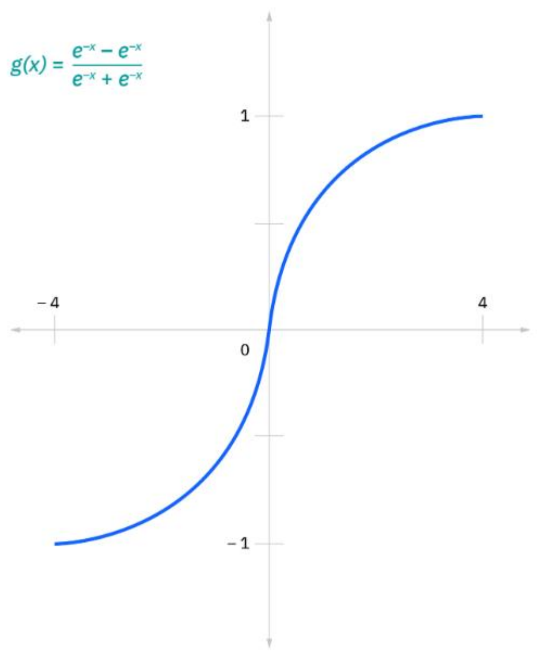
1.4.Common activation functions

As discussed in the Learn article on Neural Networks, an activation function determines whether a neuron should be activated. The nonlinear functions typically convert the output of a given neuron to a value between 0 and 1 or -1 and 1. Some of the most commonly used functions are defined as follows:

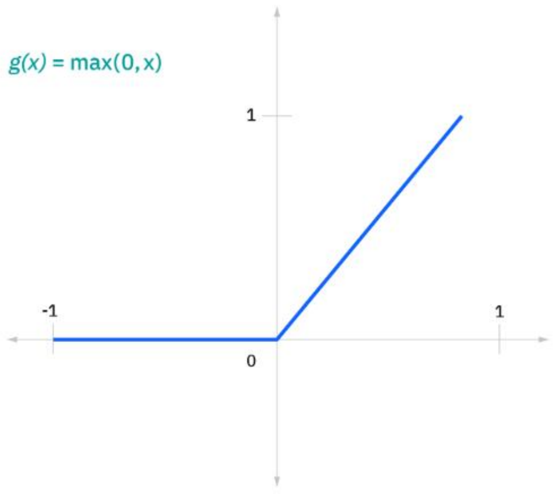
- **Sigmoid**: This is represented with the formula g(x) = 1/(1 + e^-x).



- **Tanh**: This is represented with the formula g(x) = (e^-x - e^-x)/(e^-x + e^-x).



- **Relu**: This is represented with the formula g(x) = max(0 , x)



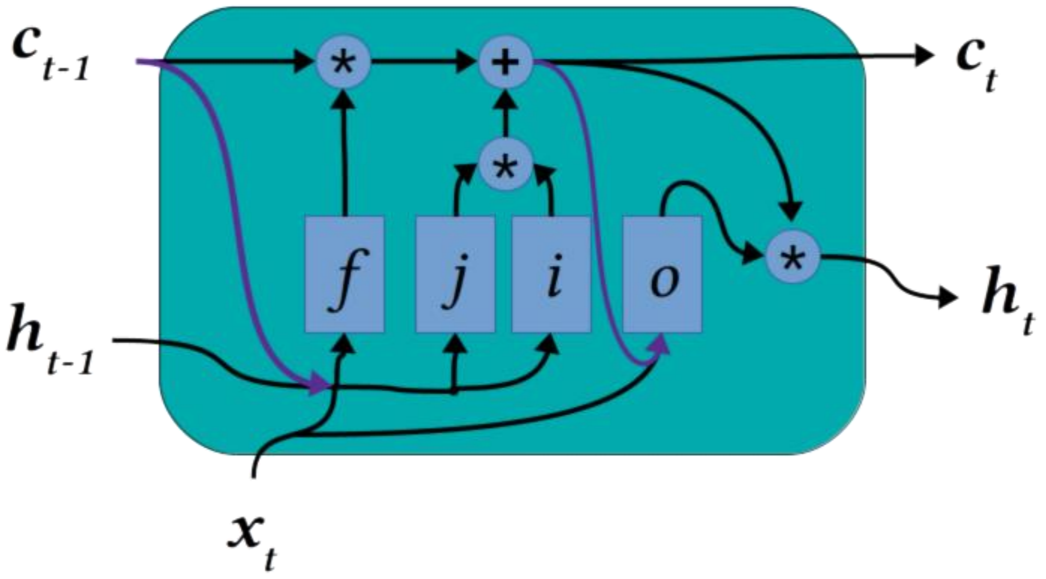
1.5.Variant RNN architectures

1.5. 1. LSTM Classic

The classic LSTM architecture is characterized by a persistent linear cell state surrounded by non-linear layers feeding input and parsing output from it. Concretely the cell state works in concert with 4 gating layers, these are often called the forget, (2x) input, and output gates.

The forget gate chooses what values of the old cell state to get rid of, based on the current input data. The two input gates (often denoted i and j) work together to decide what to add to the cell state depending on the input. i and j typically have different activation functions, which we intuitively expect to be used to suggest a scaling vector and candidate values to add to the cell state.
Finally, the output gate determines what parts of the cell state should be passed on to the output. Note that in the case of classic LSTMs, the output h consists of hidden layer activations (these can be subjected to further layers for classification, for example) and the input consists of the previously hidden state output and any new data x provided at the current time step.
Classic LSTM illustration.
The original LSTM immediately improved upon the state of the art on a set of synthetic experiments with long time lags between relevant pieces of data. Fast forward to today, and we still see the classic LSTM forming a core element of state-of-the-art reinforcement learning breakthroughs like the Dota 2 playing team OpenAI Five. Examining the policy architecture in more detail (pdf), you can see that while each agent employs a number of dense ReLU layers for feature extraction and final decision classification, a 1024-unit LSTM forms the core representation of each agent's experience of the game. A similar arrangement was used by OpenAI to train a Shadow robotic hand from scratch to manipulate a colored cube to achieve arbitrary rotations.

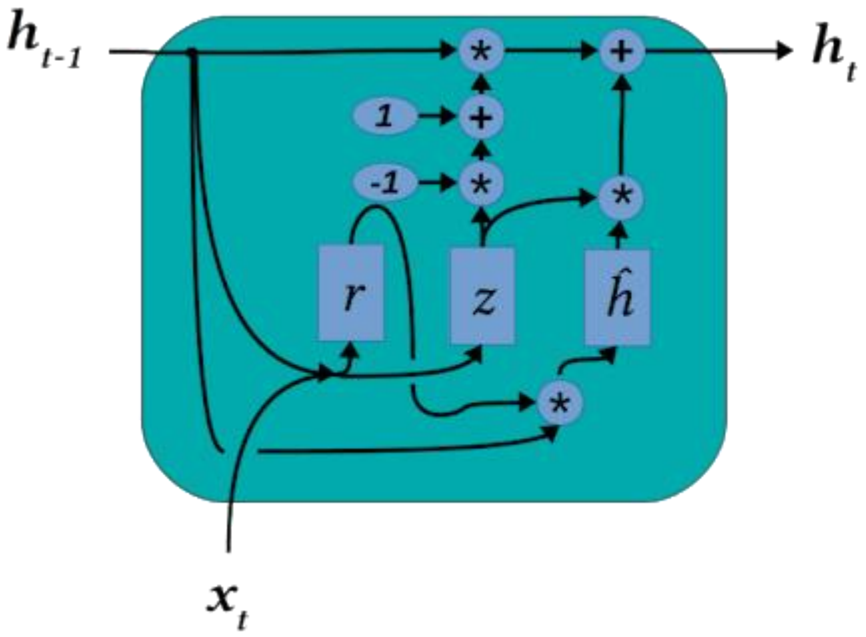
1.5. 2. Peephole Connections



LSTM Peephole Connections illustration.

The classic LSTM overcomes the problem of gradients vanishing in a recurrent neural network unrolled in time by connecting all-time points via a persistent cell state (often called a "constant error carousel" in early papers describing LSTMs). However, the gating layers that determine what to forget, what to add, and even what to take from the cell state as output doesn't take into account the contents of the cell itself.
Intuitively, it makes sense that an agent or model would want to know the memories it already has in place before replacing them with new. Enter LSTM peephole connections. This modification (shown in dark purple in the figure above) simply concatenates the cell state contents to the gating layer inputs. In particular, this configuration was shown to offer an improved ability to count and time distances between rare events when this variant was originally introduced. Providing some cell-state connections to the layers in an LSTM remains a common practice, although specific variants differ in exactly which layers are provided access.

1.5. 3. Gated Recurrent Unit



LSTM Gated Recurrent Unit illustration.
Diagrammatically, a Gated Recurrent Unit (GRU) looks more complicated than a classical LSTM. In fact, it's a bit simpler, and due to its relative simplicity trains a little faster than the traditional LSTM. GRUs combine the gating functions of the input gate j and the forget gate f into a single update gate z.
Practically that means that cell state positions earmarked for forgetting will be matched by entry points for new data. Another key difference of the GRU is that the cell state and hidden output h have been combined into a single hidden state layer, while the unit also contains an intermediate, internal hidden state.

Gated Recurrent Units (GRUs) have been used for the basis for demonstrating exotic concepts like Neural GPUs as well as a simpler model for the sequence to sequence learning in general, such as machine translation. GRUs are a capable LSTM variant and they have been fairly popular since their inception. While they can learn quickly on tasks like music or text generation, they have been described as ultimately less powerful than classic LSTMs due to their limitations in counting.

1.5. 4. Multiplicative LSTM (2017)

Multiplicative LSTMs (mLSTMs) was introduced by Krause et al 2016. Since then, this complicated variant has been the centerpiece of a number of high-profile, state of the art achievements in natural language processing. Perhaps the most well-known of these is OpenAI's unsupervised sentiment neuron.

Researchers on the project that by pre-training a big mLSTM model on unsupervised text prediction it became much more capable and could perform at a high level on a battery of NLP tasks with minimal fine-tuning. A number of interesting features in the text (such as sentiment) were emergently mapped to specific neurons.

Remarkably, the same phenomenon of interpretable classification neurons emerging from unsupervised learning has been reported in end-to-end protein sequences learning. On next-residue prediction tasks of protein sequences, multiplicative LSTM models apparently learn internal representations corresponding to fundamental secondary structural motifs like alpha helices and beta sheets. Protein sequence and structure is an area ripe for major breakthroughs from unsupervised and semi-supervised sequence learning models.

Although the amount of sequence data has been increasing exponentially for the last few years, available protein structure data increases at a much more leisurely pace. Therefore, the next big AI upset in the protein folding field will probably involve some degree of unsupervised learning on pure sequences, and may even eclipse Deepmind's upset at the CASP13 protein folding challenge.

1.5. 5. LSTMs With Attention

Finally, we arrive at what is probably the most transformative innovation in sequence models in recent memory*. Attention in machine learning refers to a model's ability to focus on specific elements in data, in our case the hidden state outputs of LSTMs. Wu et al. at Google used an architecture consisting of an attention network sandwiched between encoding and decoding LSTM layers to achieve state of the art Neural Machine Translation.

This likely continues to power Google Translate to this day. OpenAI's demonstration of tool use in a hide-and-seek reinforcement learning environment is a recent example of the capability of LSTMs with attention on a complex, unstructured task.

The significant successes of LSTMs with attention in natural language processing foreshadowed the decline of LSTMs in the best language models. With increasingly powerful computational resources available for NLP research, state-of-the-art models

now routinely make use of a memory-hungry architectural style known as the transformer.
Transformers do away with LSTMs in favor of feed-forward encoder/decoders with attention. Attention transformers obviate the need for cell-state memory by picking and choosing from an entire sequence fragment at once, using attention to focus on the most important parts. BERT, ELMO, GPT-2, and other major language models all follow this approach.
On the other hand, state-of-the-art NLP models incur a significant economic and environmental impact to train from scratch, requiring resources available mainly to research labs associated with wealthy tech companies. The massive energy requirements for these big transformer models make transfer learning all the more important, but it also leaves plenty of room for LSTM-based sequence-to-sequence models to make meaningful contributions to tasks sufficiently different from those the big language transformers are trained for.

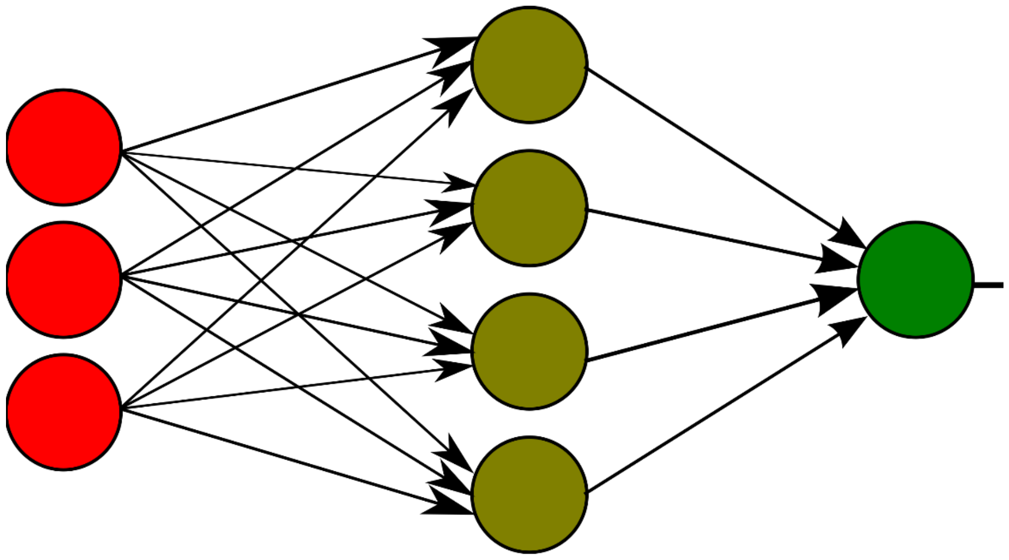
1.6.Recurrent neural networks and IBM Cloud

- IBM has been a pioneer in the development of AI technologies and neural networks for decades, as evidenced by the development and evolution of IBM Watson. Watson is now a trusted solution for businesses looking to apply advanced natural language processing and deep learning techniques to their systems while adhering to a tried-and-true tiered approach to AI adoption and implementation.
- IBM Watson Machine Learning, for example, supports popular Python libraries used in recurrent neural networks, such as TensorFlow, Keras, and PyTorch. Your enterprise can seamlessly bring your open-source AI projects into production while deploying and running your models on any cloud by utilizing tools such as IBM Watson Studio and Watson Machine Learning.

2. MultiLayer Perceptron (MLP)

2.1.What are multilayer perceptrons?

A multilayer perceptron (MLP) is a feedforward artificial neural network that generates a set of outputs from a set of inputs. An MLP is characterized by several layers of input nodes connected as a directed graph between the input and output layers. MLP uses backpropogation for training the network. MLP is a deep learning method.



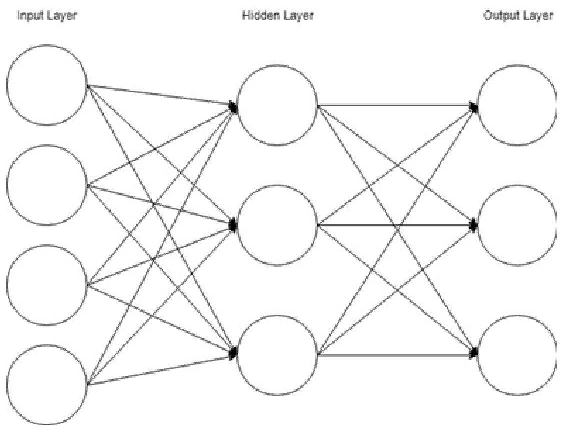
Techopedia Explains Multilayer Perceptron (MLP)

A multilayer perceptron is a neural network connecting multiple layers in a directed graph, which means that the signal path through the nodes only goes one way. Each node, apart from the input nodes, has a nonlinear activation function. An MLP uses backpropagation as a supervised learning technique. Since there are multiple layers of neurons, MLP is a deep learning technique.

MLP is widely used for solving problems that require supervised learning as well as research into computational neuroscience and parallel distributed processing. Applications include speech recognition, image recognition and machine translation.

2.2 How does a multilayer perceptron work?

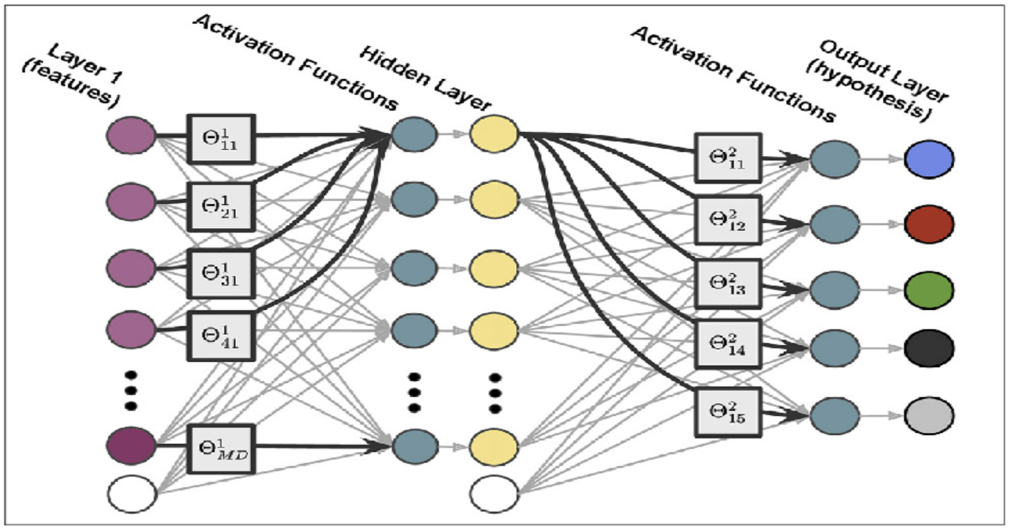
The Perceptron is made up of two fully connected layers: input and output. MLPs have the same input and output layers but may have multiple hidden layers in between, as shown below.



The algorithm for the MLP is as follows:

1. The MLP, like the perceptron, pushes inputs forward by taking the dot product of the input and the weights that exist between the input layer and the hidden layer (W---H). At the hidden layer, this dot product produces a value. We do not, however, advance this value as we would with a perceptron.
2. MLPs utilize activation functions at each of their calculated layers. There are many activation functions to discuss: rectified linear units (ReLU), sigmoid function, tanh. Push the calculated output at the current layer through any of these activation functions.
3. Once the calculated output at the hidden layer has been pushed through the activation function, push it to the next layer in the MLP by taking the dot product with the corresponding weights.
4. Repeat steps 2 and 3 until you reach the output layer.
5. The calculations will be used at the output layer for either a backpropagation algorithm that corresponds to the activation function that was chosen for the MLP (in the case of training) or a decision will be made based on the output.

➔ *MLPs serve as the foundation for all neural networks and have significantly increased computer power when applied to classification and regression problems. Because of the multilayer perceptron, computers are no longer limited by XOR cases and can learn rich and complex models.*



II. Comparison and evaluations of 2 models:

MLP :

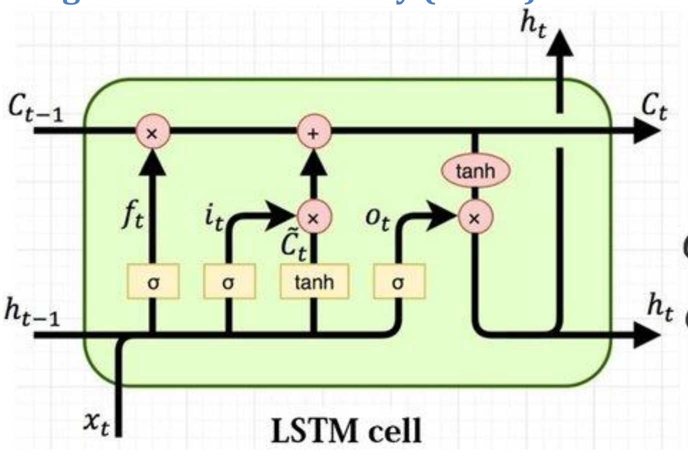
```
from keras.layers import Dense, Dropout, Activation
from tensorflow.keras import regularizers, optimizers
from keras.models import Sequential
import time

mlp = Sequential()

mlp.add(Dense(units=3000, activation='relu', input_dim=x_train.shape[1])
)
mlp.add(Dense(units=3000, activation='relu'))
mlp.add(Dense(units=3000, activation='relu'))
mlp.add(Dense(units=3000, activation='relu'))
mlp.add(Dense(units=3000, activation='relu'))

mlp.add(Dense(1))


opt = optimizers.RMSprop(lr=0.001)

mlp.compile(optimizer=opt, loss='mse', metrics=['mean_squared_error'])

start = time.time()
mlp.fit(x_train, y_train, epochs=30)
end = time.time()

print(f"Thời gian training mạng MLP là: {(end - start)} s")
```

RNN:

```
from keras.layers import Dense, SimpleRNN

# Build the LSTM model
rnn = Sequential()
rnn.add(SimpleRNN(128, return_sequences=True, input_shape= (x_train.shap
e[1], 1)))
rnn.add(SimpleRNN(64, return_sequences=False))
rnn.add(Dense(25))
rnn.add(Dense(1))

# Compile the model
rnn.compile(optimizer='adam', loss='mean_squared_error')

# Train the model
start = time.time()
rnn.fit(x_train, y_train, batch_size=32, epochs=30)
end = time.time()

print(f"Thời gian training mạng MLP là: {(end - start)} s")
```

PROBLEM 6

I. Long Short-Term Memory (LSTM)



$$i_t = \sigma\left(x_t U^i + h_{t-1} W^i\right)$$
$$f_t = \sigma\left(x_t U^f + h_{t-1} W^f\right)$$
$$o_t = \sigma\left(x_t U^o + h_{t-1} W^o\right)$$
$$\tilde{C}_t = \tanh\left(x_t U^g + h_{t-1} W^g\right)$$
$$C_t = \sigma\left(f_t * C_{t-1} + i_t * \tilde{C}_t\right)$$
$$h_t = \tanh(C_t) * o_t$$

1. What Does Long Short-Term Memory (LSTM) Mean?

Long Short Term Memory Networks is an advanced RNN, a sequential network, that allows information to persist. It is capable of handling the vanishing gradient problem faced by RNN. A recurrent neural network is also known as RNN is used for persistent memory.

Similarly RNNs work, they remember the previous information and use it for processing the current input. The shortcoming of RNN is, they can not remember Long term dependencies due to vanishing gradient. LSTMs are explicitly designed to avoid long-term dependency problems.

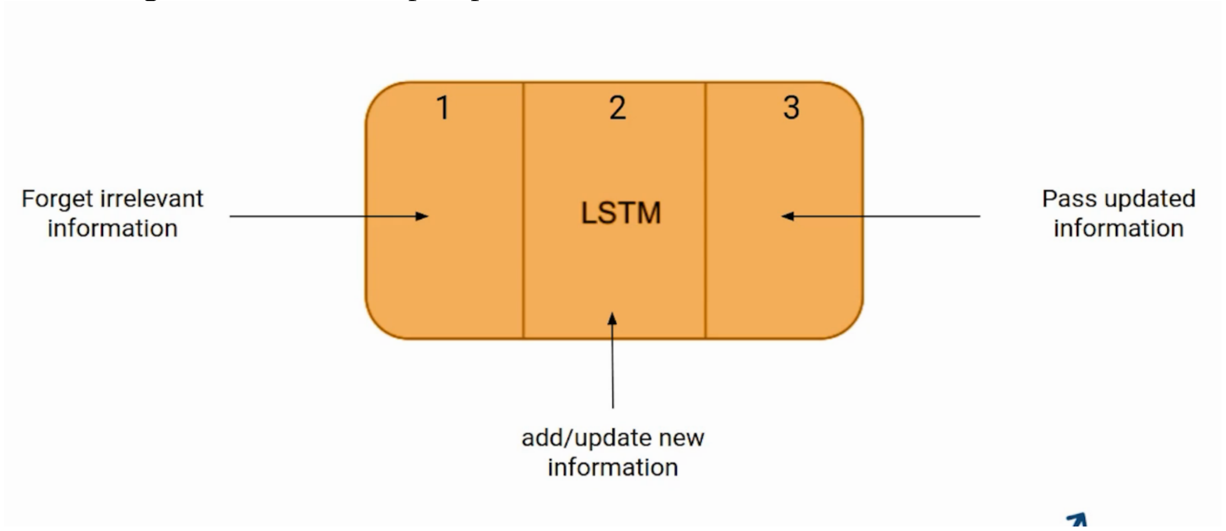
2. Explains Long Short-Term Memory (LSTM)

- The recurrent neural network uses long short-term memory blocks to offer context for how the software receives inputs and produces outputs. A complex unit with weighted inputs, activation functions, inputs from earlier blocks, and potential outputs is the long short-term memory block.
- The unit is referred to as a long short-term memory block because the software employs a structure built on short-term memory processes to produce longer-term memory. These systems are frequently employed, for instance, in natural language processing. The recurrent neural network uses long short-term memory blocks, which are effective for sorting and categorizing these kinds of inputs, to evaluate a single word or phoneme in the context of other words in a string. In general, LSTM is an established and popular idea that helped develop recurrent neural networks.
- LSTM stands for long short-term memory networks, used in the field of Deep Learning. It is a variety of recurrent neural networks (RNNs) that are capable of learning long-term dependencies, especially in sequence prediction problems. LSTM has feedback connections, i.e., it is capable of processing the entire sequence of data, apart from single data points such as images. This finds application in speech recognition, machine translation, etc. LSTM is a special

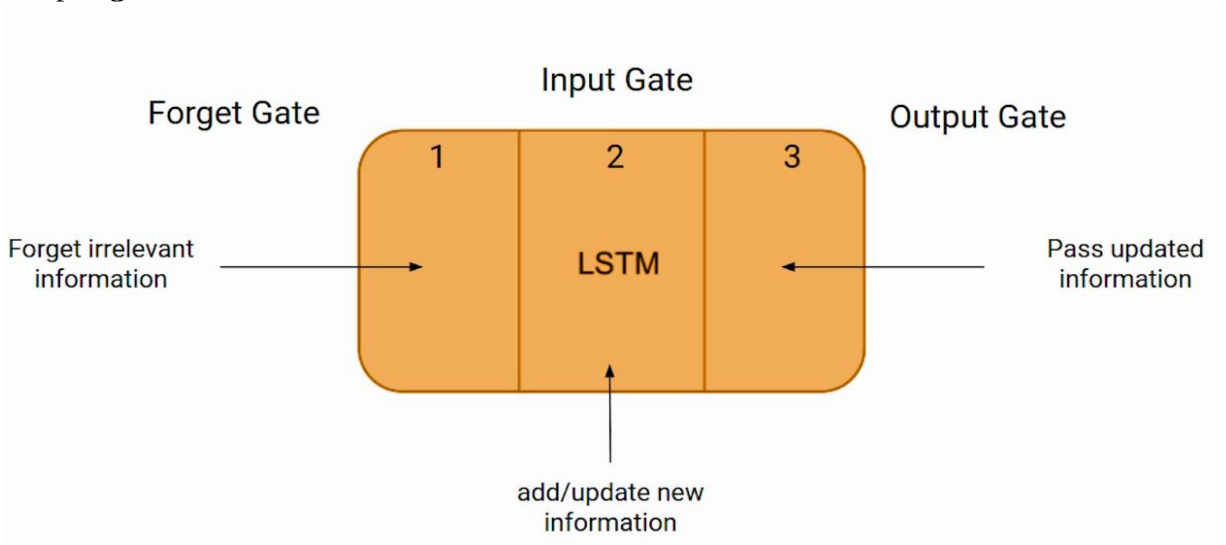
kind of RNN, which shows outstanding performance on a large variety of problems.

3. LSTM Architecture

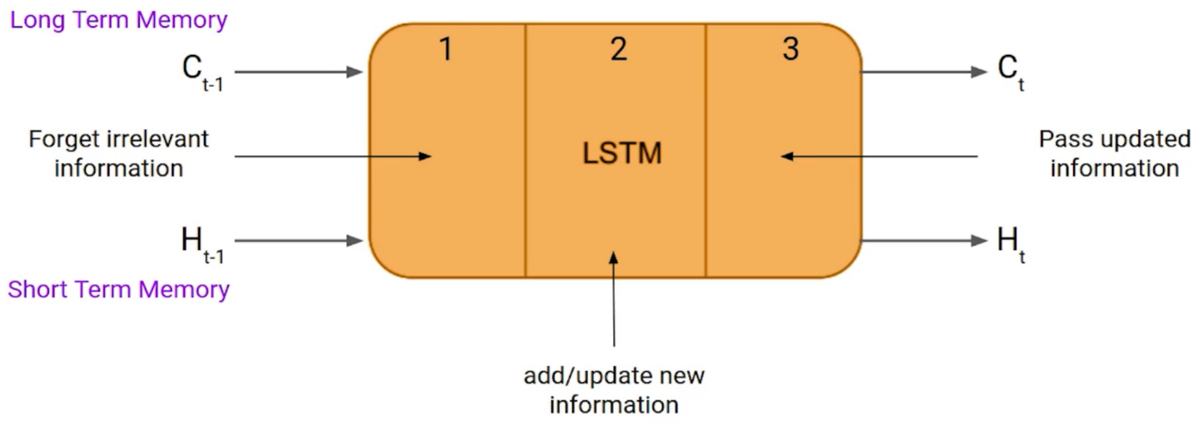
At a high-level LSTM works very much like an RNN cell. Here is the internal functioning of the LSTM network. The LSTM consists of three parts, as shown in the image below and each part performs an individual function.



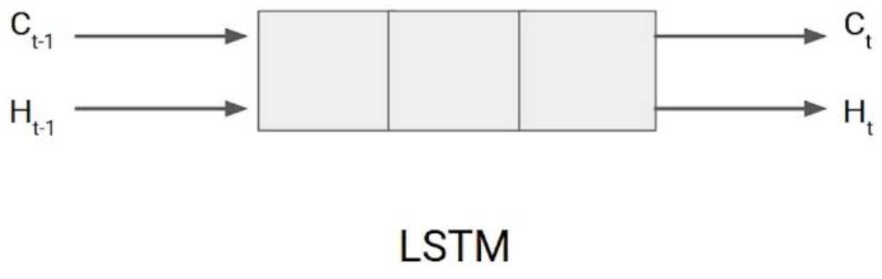
The first part chooses whether the information coming from the previous timestamp is to be remembered or is irrelevant and can be forgotten. In the second part, the cell tries to learn new information from the input to this cell. At last, in the third part, the cell passes the updated information from the current timestamp to the next timestamp.
These three parts of an LSTM cell are known as gates. The first part is called Forget gate, the second part is known as the Input gate and the last one is the Output gate.



Just like a simple RNN, an LSTM also has a hidden state where H(t-1) represents the hidden state of the previous timestamp and Ht is the hidden state of the current timestamp. In addition to that LSTM also have a cell state represented by C(t-1) and C(t) for previous and current timestamp respectively. Here the hidden state is known as Short term memory and the cell state is known as Long term memory. Refer to the following image.



It is interesting to note that the cell state carries the information along with all the timestamps.



Bob is a nice person. Dan on the other hand is evil.

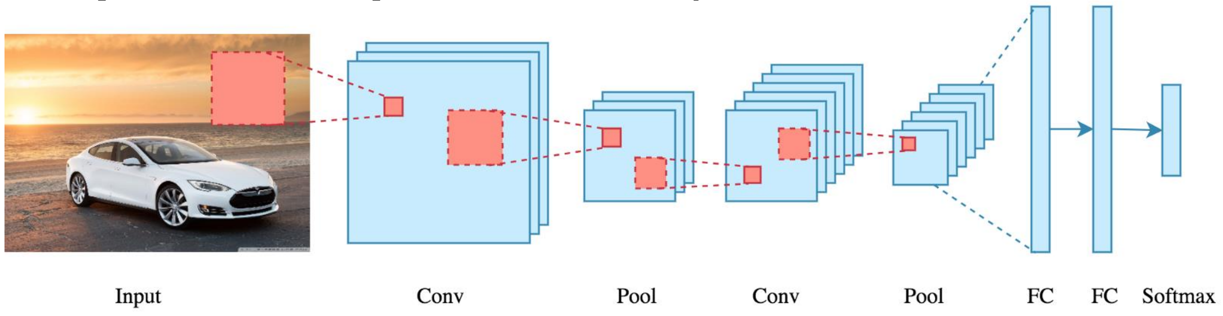
Let’s take an example to understand how LSTM works. Here we have two sentences separated by a full stop. The first sentence is “Bob is a nice person” and the second sentence is “Dan, on the Other hand, is evil”. It is very clear, in the first sentence we are talking about Bob and as soon as we encounter the full stop(.) we started talking about Dan.
As we move from the first sentence to the second sentence, our network should realize that we are no more talking about Bob. Now our subject is Dan. Here,

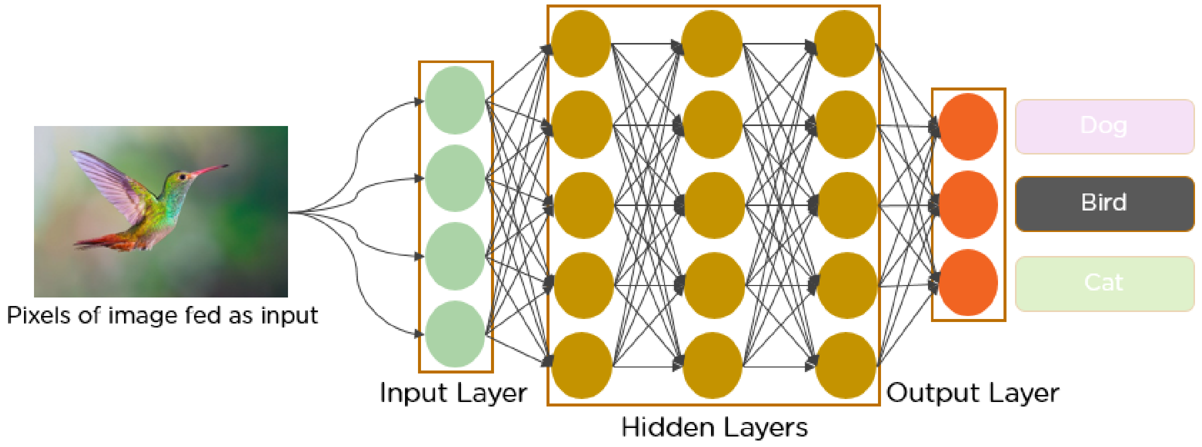
the Forget gate of the network allows it to forget about it. Let's understand the roles played by these gates in LSTM architecture.

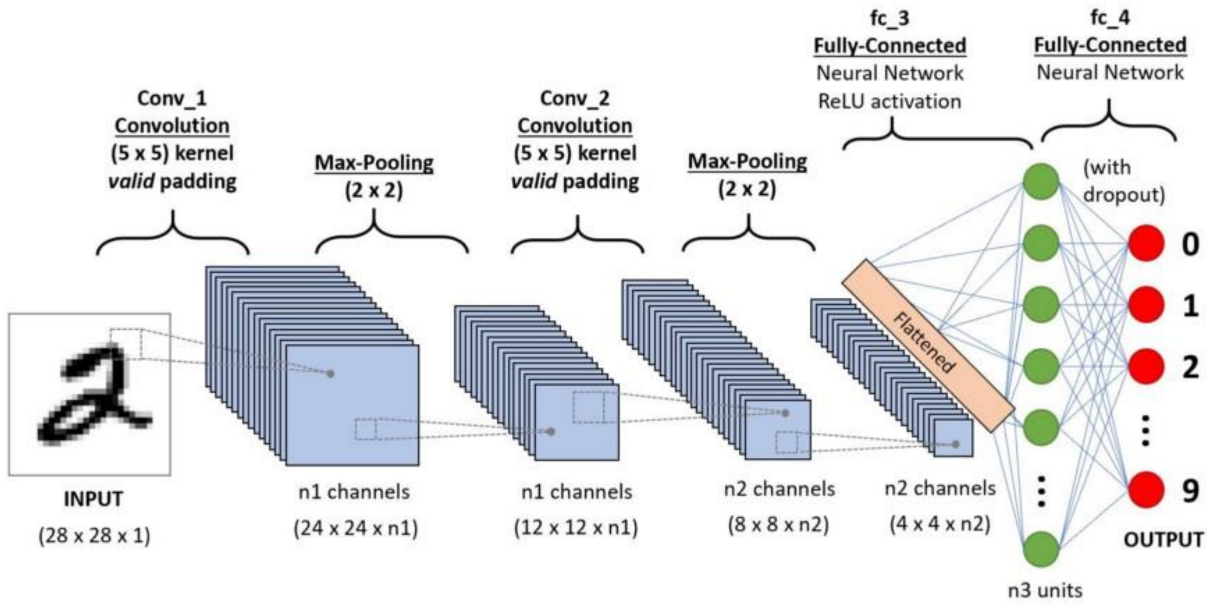
II. Convolutional Neural Network (CNN)

1. What is Convolutional Neural Network?

A convolutional neural network (CNN/ConvNet) is a class of deep neural networks, most commonly applied to analyze visual imagery. Now when we think of a neural network we think about matrix multiplications but that is not the case with ConvNet. It uses a special technique called Convolution. Now in mathematics convolution is a mathematical operation on two functions that produces a third function that expresses how the shape of one is modified by the other.

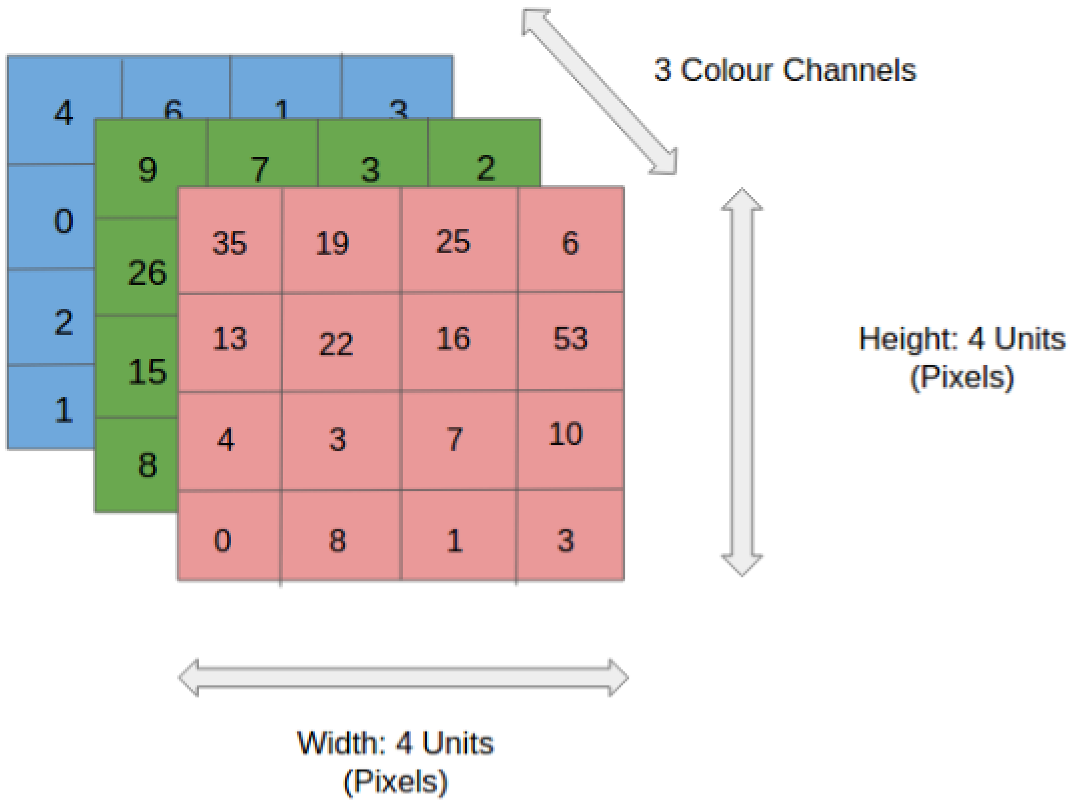




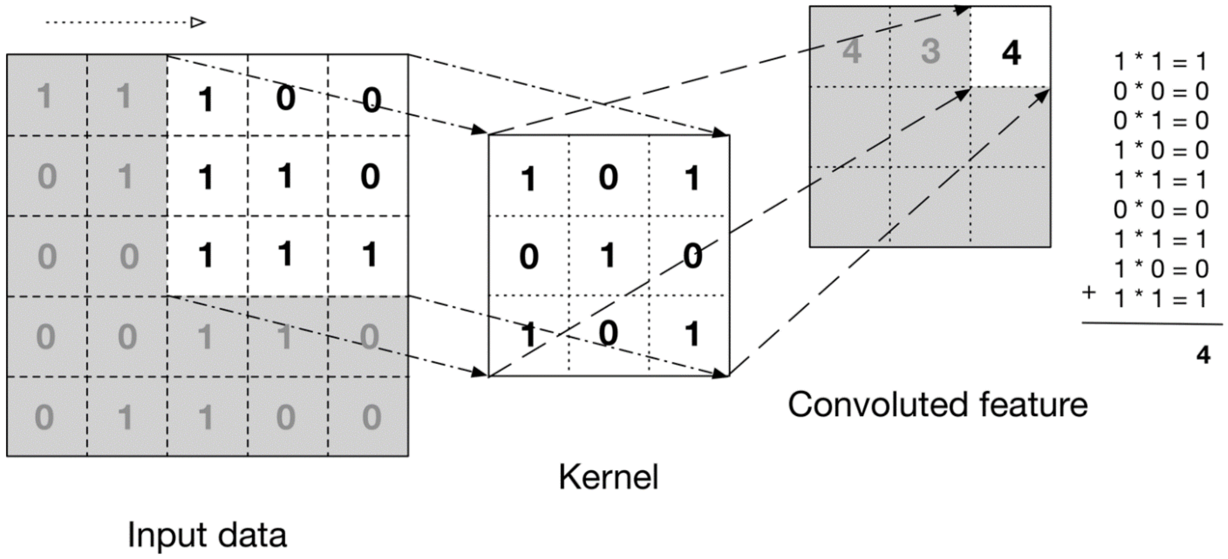


- Convolutional Neural Networks (CNNs) can learn complicated objects and patterns because they have an input layer, an output layer, numerous hidden layers, and millions of parameters. It subsamples the given input using convolution and pooling processes before applying an activation function, where all of them are hidden layers that are partially connected, with the fully connected layer at the end resulting in the output layer. The size of the output image is similar to the size of the input image.
- The process of combining two functions to produce the output of the other function is known as convolution. The input image is convoluted by the use of filters in CNNs, yielding a Feature map. Filters are weights and biases that are generated at random in the network. CNN uses the same weights and biases for all neurons rather than having individual weights and biases for each neuron. Many filters can be created, each capturing a different aspect of the input. Filters are also known as kernels.

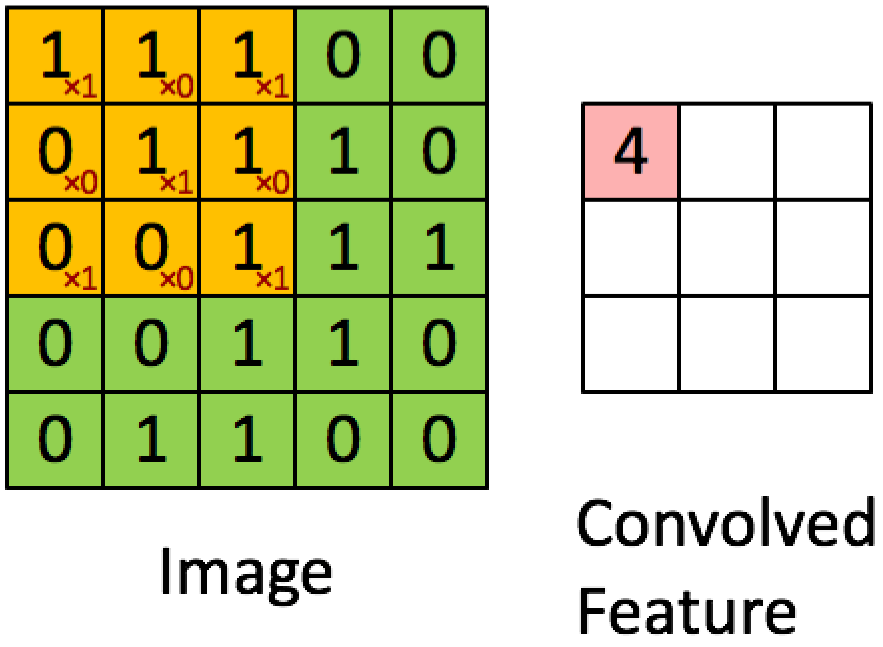
2. How does it work?
   Before we go to the working of CNN's let's cover the basics such as what is an image and how is it represented. An RGB image is nothing but a matrix of pixel values having three planes whereas a grayscale image is the same but it has a single plane. Take a look at this image to understand more.

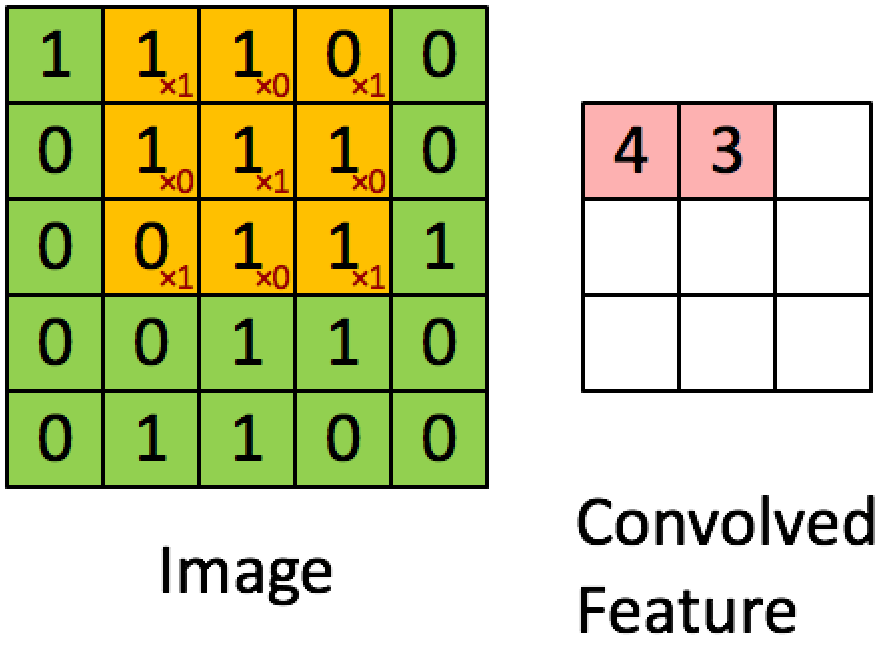


For simplicity, let's stick with grayscale images as we try to understand how CNNs work.



The above image shows what a convolution is. We take a filter/kernel(3×3 matrix) and apply it to the input image to get the convolved feature. This convolved feature is passed on to the next layer.


1 ×1 | 1 ×0 | 1 ×1 | 0 | 0
0 ×0 | 1 ×1 | 1 ×0 | 1 | 0
0 ×1 | 0 ×0 | 1 ×1 | 1 | 1
0 | 0 | 1 | 1 | 0
0 | 1 | 1 | 0 | 0
Image
4
Convolved Feature


1 | 1 ×1 | 1 ×0 | 0 ×1 | 0
0 | 1 ×0 | 1 ×1 | 1 ×0 | 0
0 | 0 ×1 | 1 ×0 | 1 ×1 | 1
0 | 0 | 1 | 1 | 0
0 | 1 | 1 | 0 | 0
Image
4 | 3
Convolved Feature

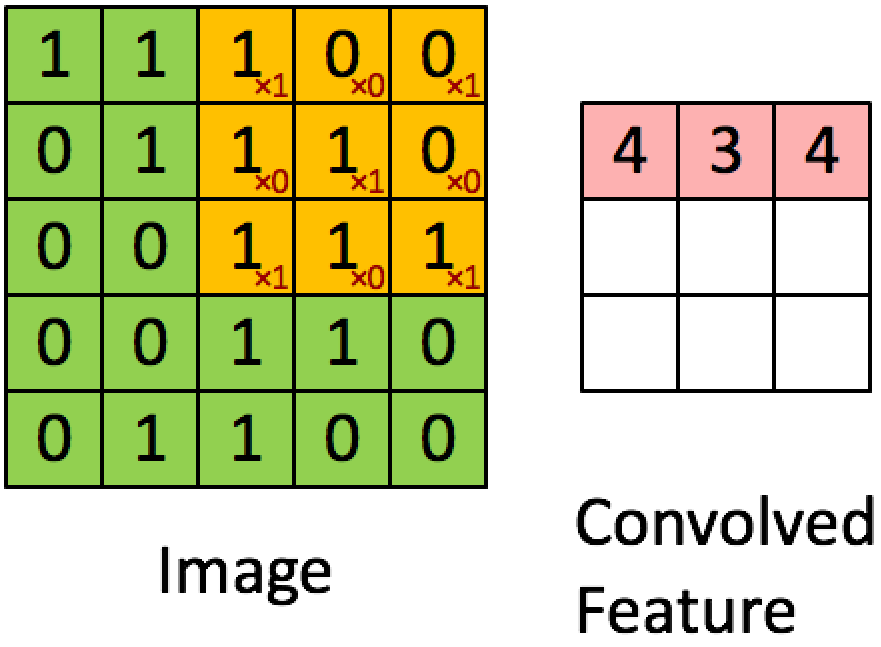


Image
Convolved Feature
4 3 4

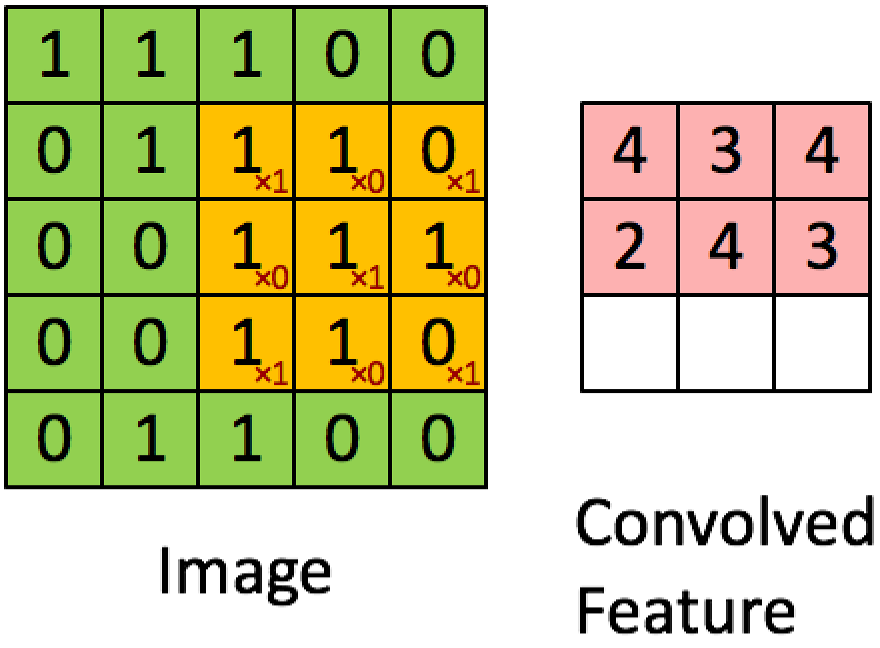
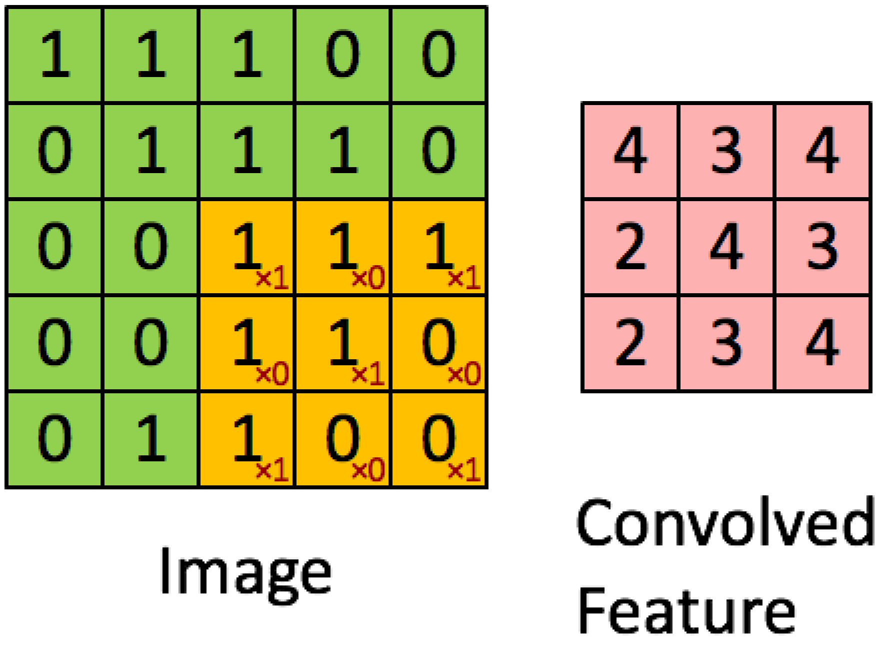
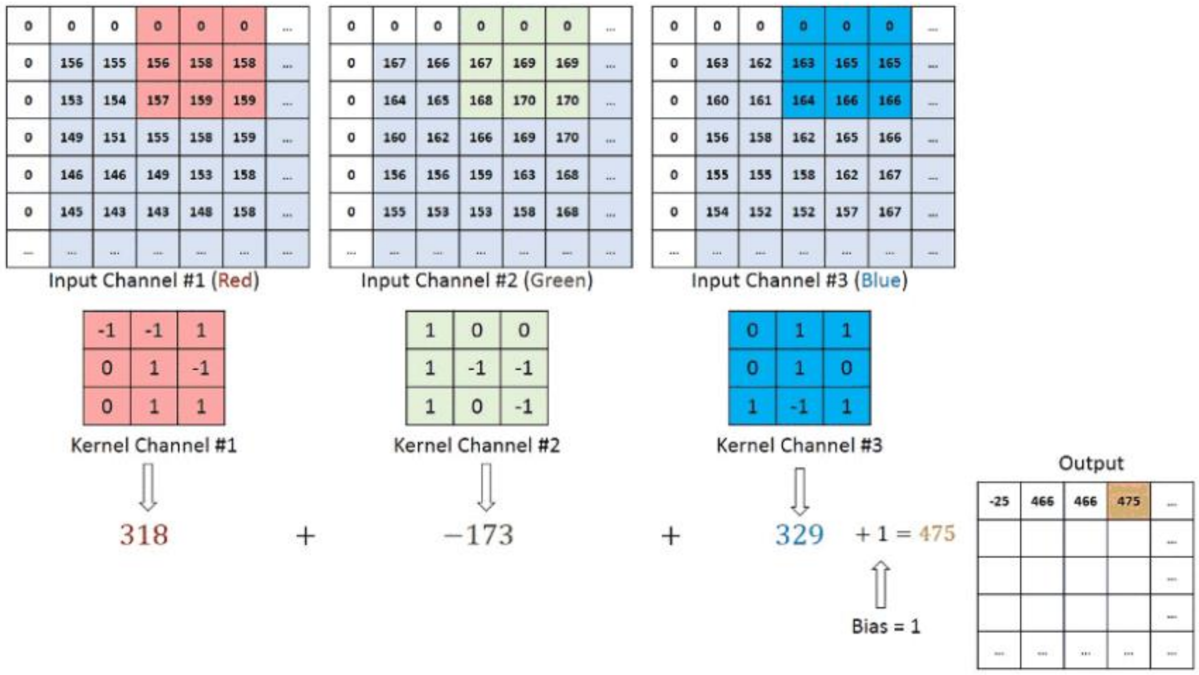


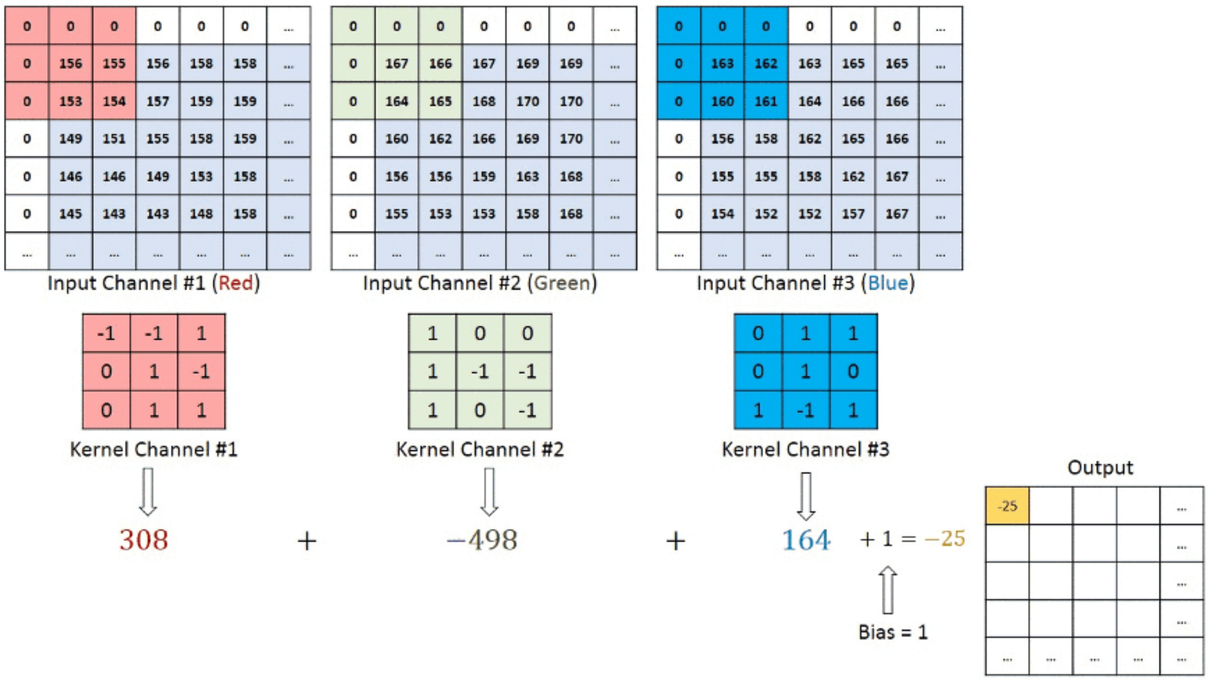
Image
Convolved Feature
4 3 4
2 4 3

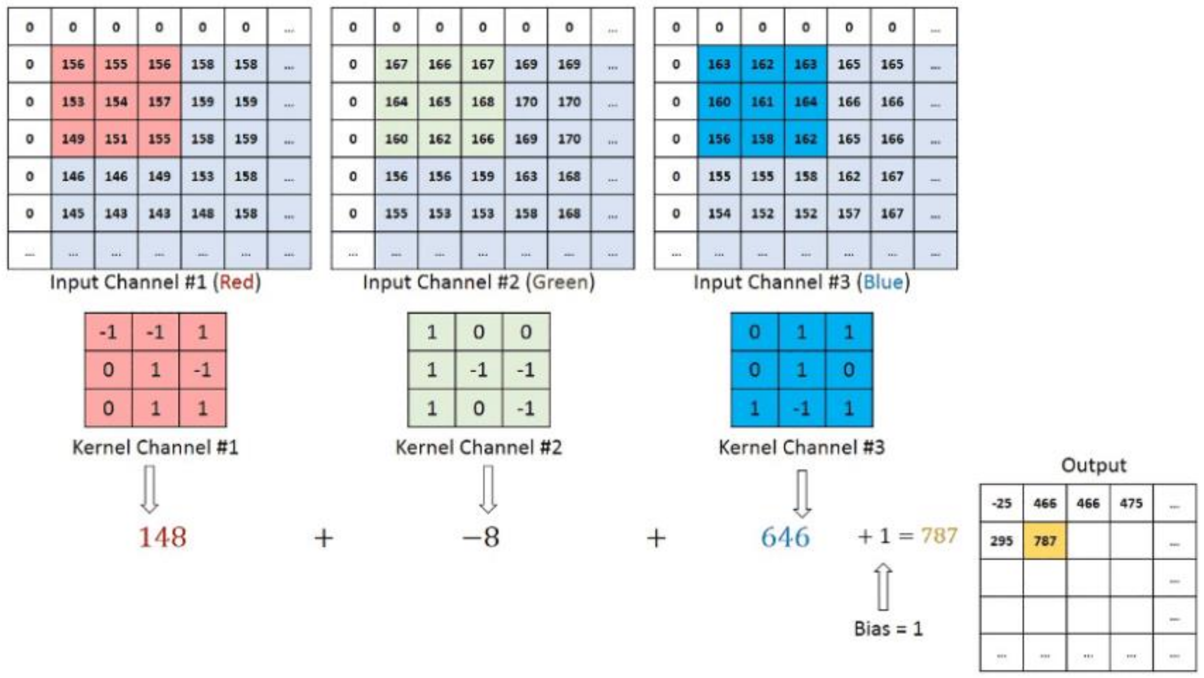


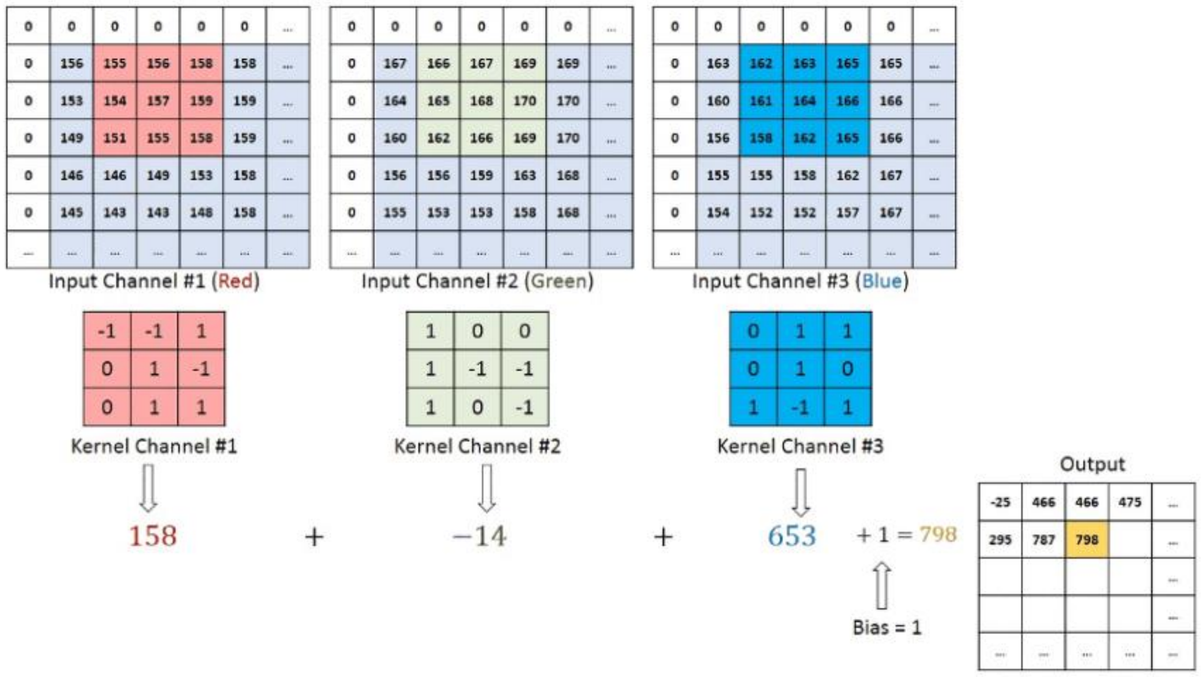
In the case of RGB color, channel take a look at this animation to understand its working

In the case of RGB color, channel take a look at this animation to understand its working



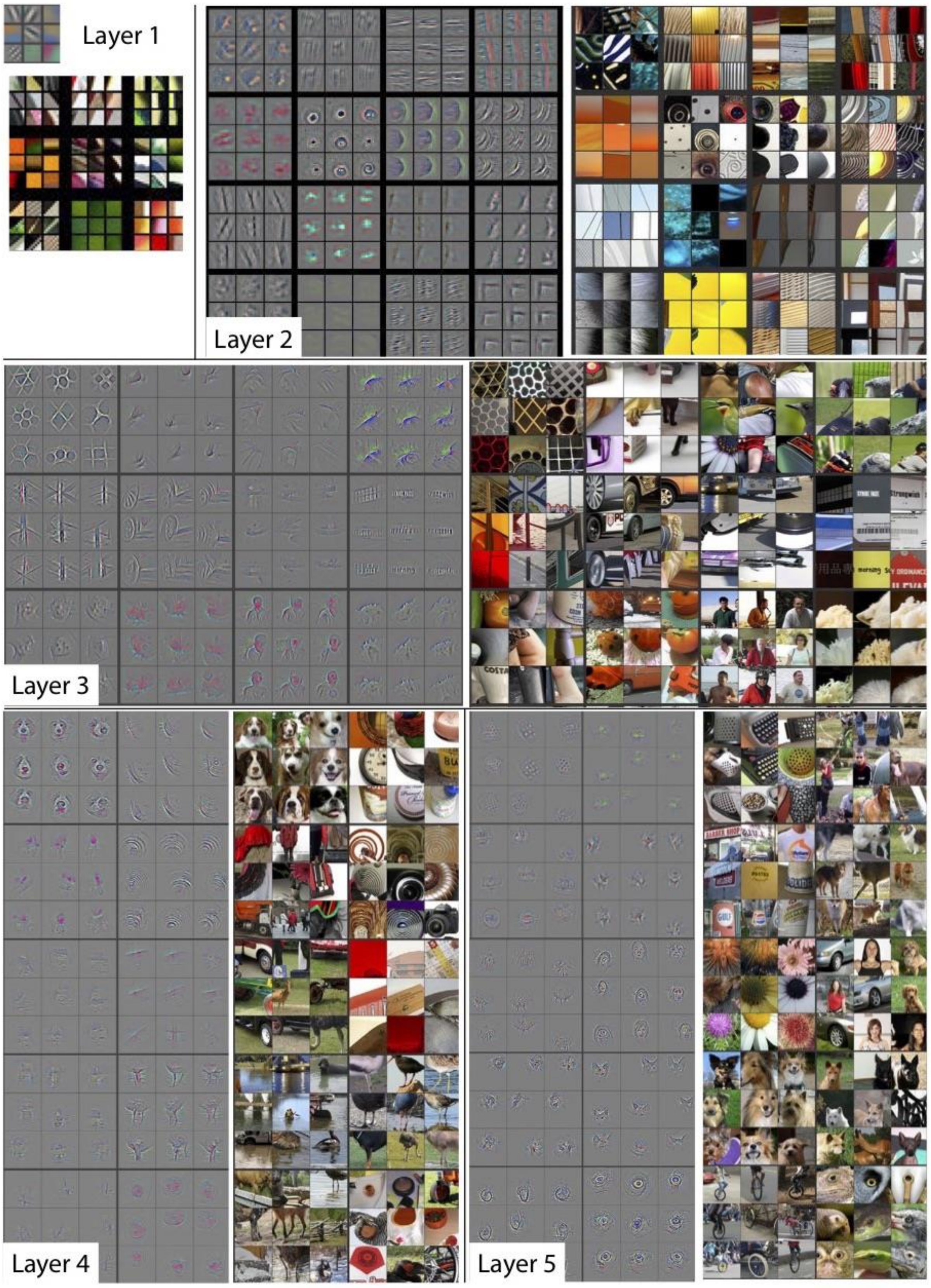



Input Channel #1 (Red)
Input Channel #2 (Green)
Input Channel #3 (Blue)
Kernel Channel #1
Kernel Channel #2
Kernel Channel #3
148
+
−8
+
646
+ 1 = 787
Bias = 1
Output

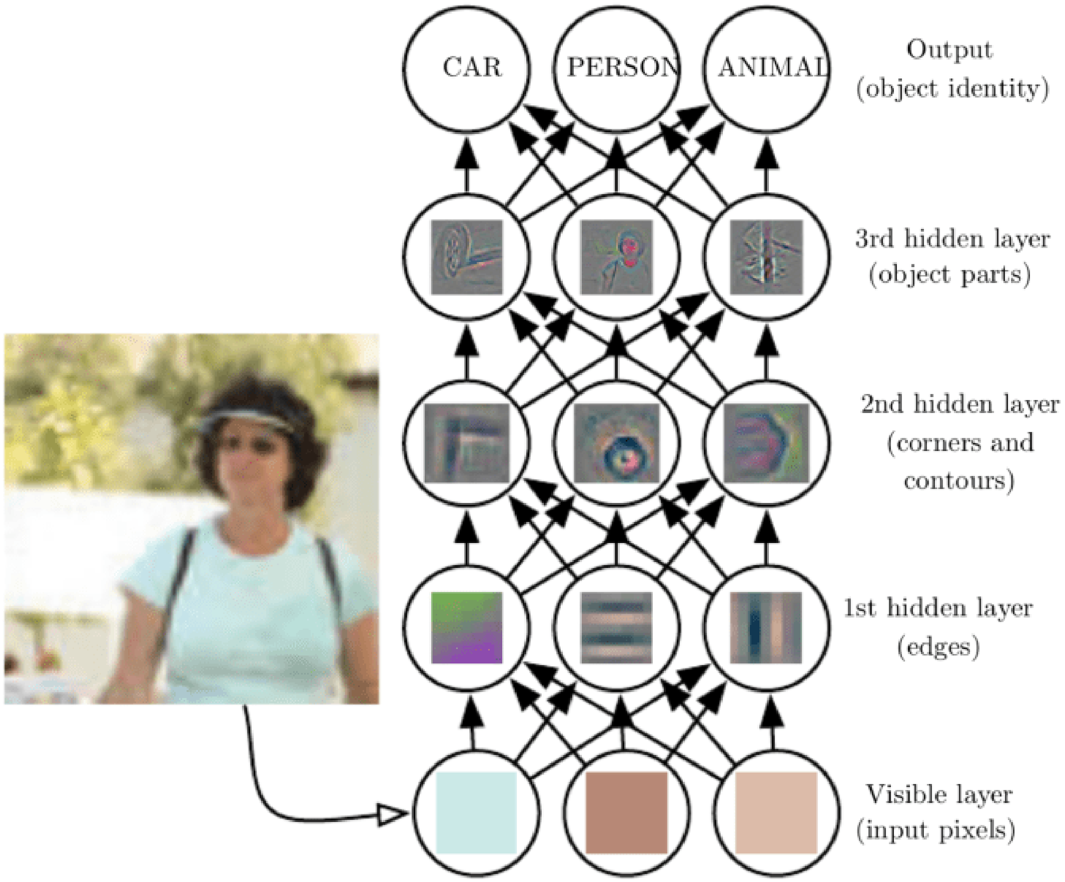

Input Channel #1 (Red)
Input Channel #2 (Green)
Input Channel #3 (Blue)
Kernel Channel #1
Kernel Channel #2
Kernel Channel #3
158
+
−14
+
653
+ 1 = 798
Bias = 1
Output

Convolutional neural networks are composed of multiple layers of artificial neurons. Artificial neurons, a rough imitation of their biological counterparts, are mathematical functions that calculate the weighted sum of multiple inputs and outputs an activation value. When you input an image in a ConvNet, each layer generates several activation functions that are passed on to the next layer.

The first layer usually extracts basic features such as horizontal or diagonal edges. This output is passed on to the next layer which detects more complex features such as corners or combinational edges. As we move deeper into the network it can identify even more complex features such as objects, faces, etc.


Layer 1
Layer 2
Layer 3
Layer 4
Layer 5

Based on the activation map of the final convolution layer, the classification layer outputs a set of confidence scores (values between 0 and 1) that specify how likely the image is to belong to a "class." For instance, if you have a ConvNet that detects cats, dogs, and horses, the output of the final layer is the possibility that the input image contains any of those animals.


CAR
PERSON
ANIMAL
Output (object identity)
3rd hidden layer (object parts)
2nd hidden layer (corners and contours)
1st hidden layer (edges)
Visible layer (input pixels)

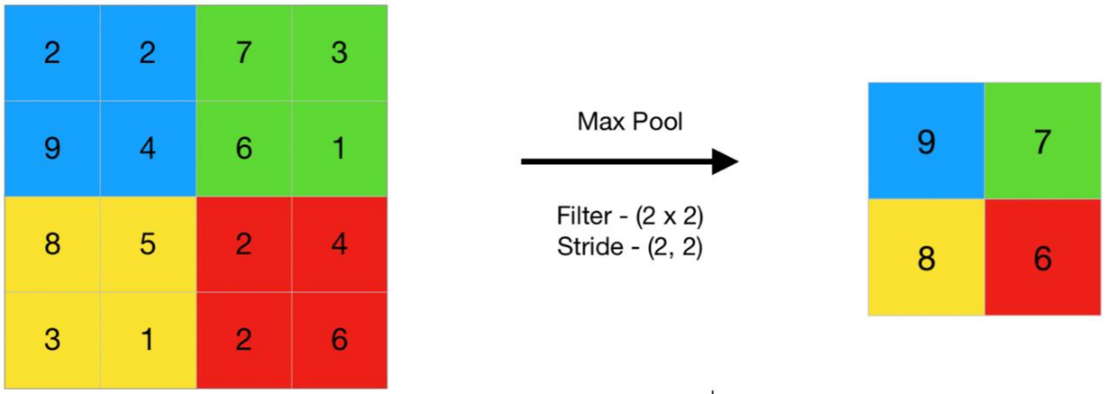
- ## What's a pooling layer?

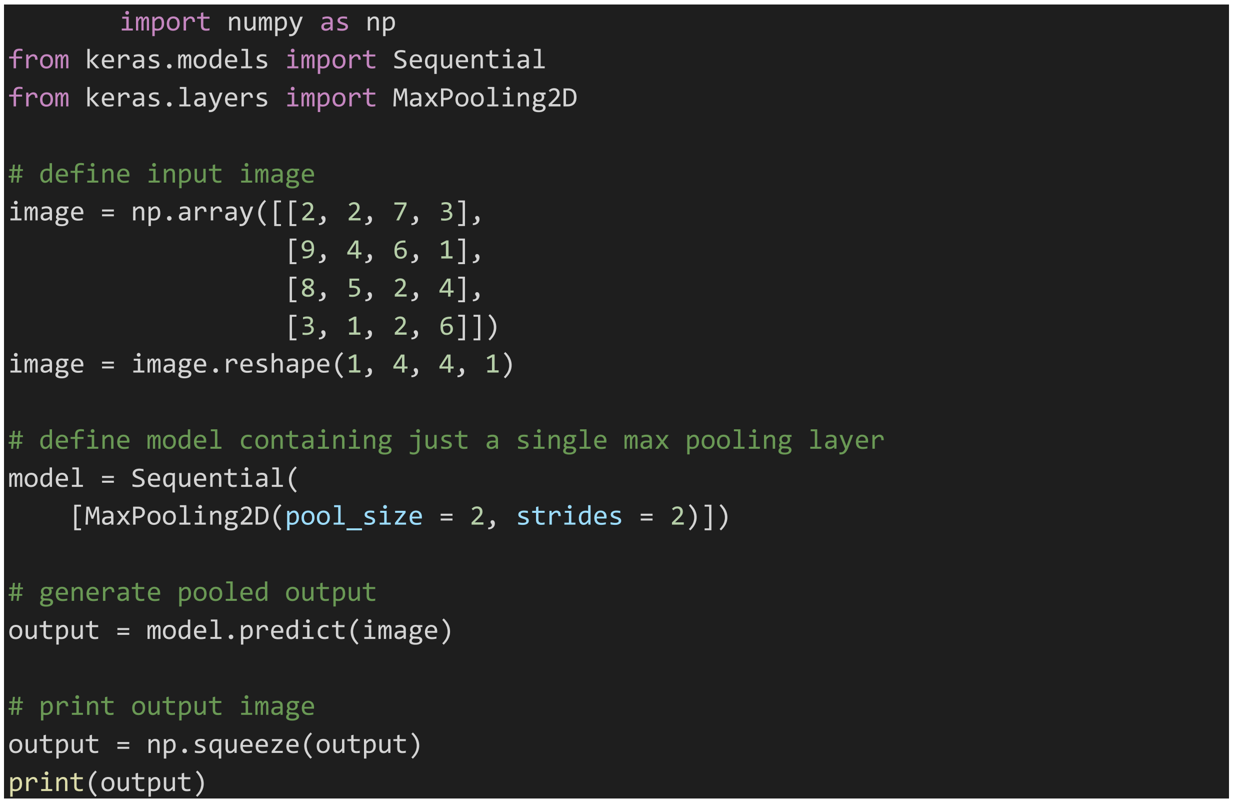
Similar to the Convolutional Layer, the Pooling layer is responsible for reducing the spatial size of the Convolved Feature. This is to decrease the computational power required to process the data by reducing the dimensions. There are two types of pooling average pooling and max pooling. I've only had experience with Max Pooling so far I haven't faced any difficulties.

So what we do in Max Pooling is we find the maximum value of a pixel from a portion of the image covered by the kernel. Max Pooling also performs as a Noise Suppressant. It discards the noisy activations altogether and also performs de-noising along with dimensionality reduction. On the other hand, Average Pooling returns the average of all the values from the portion of the image covered by the Kernel. Average Pooling simply performs dimensionality reduction as a noise suppressing mechanism. Hence, we can say that Max Pooling performs a lot better than Average Pooling.

1. Max Pooling

Max pooling is a pooling operation that selects the maximum element from the region of the feature map covered by the filter. Thus, the output after max-pooling layer would be a feature map containing the most prominent features of the previous feature map.

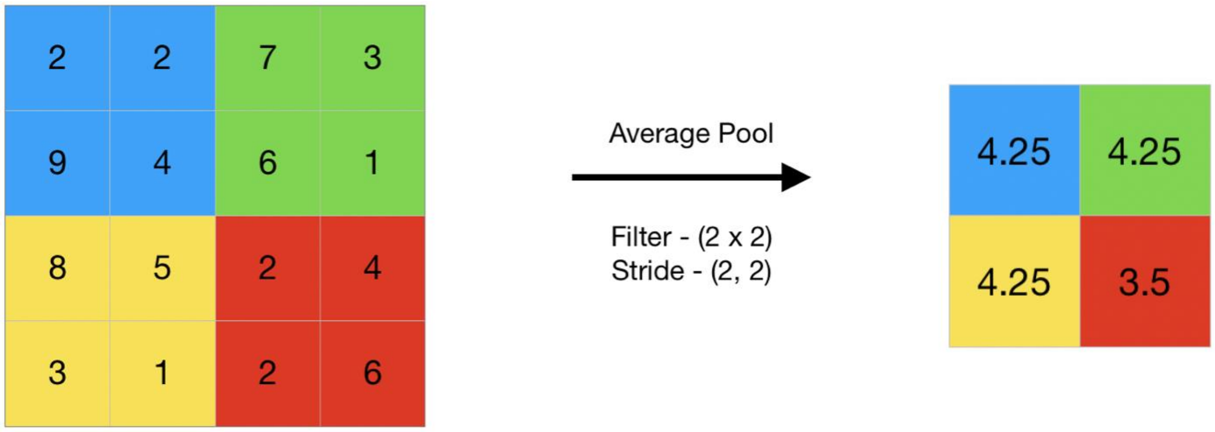


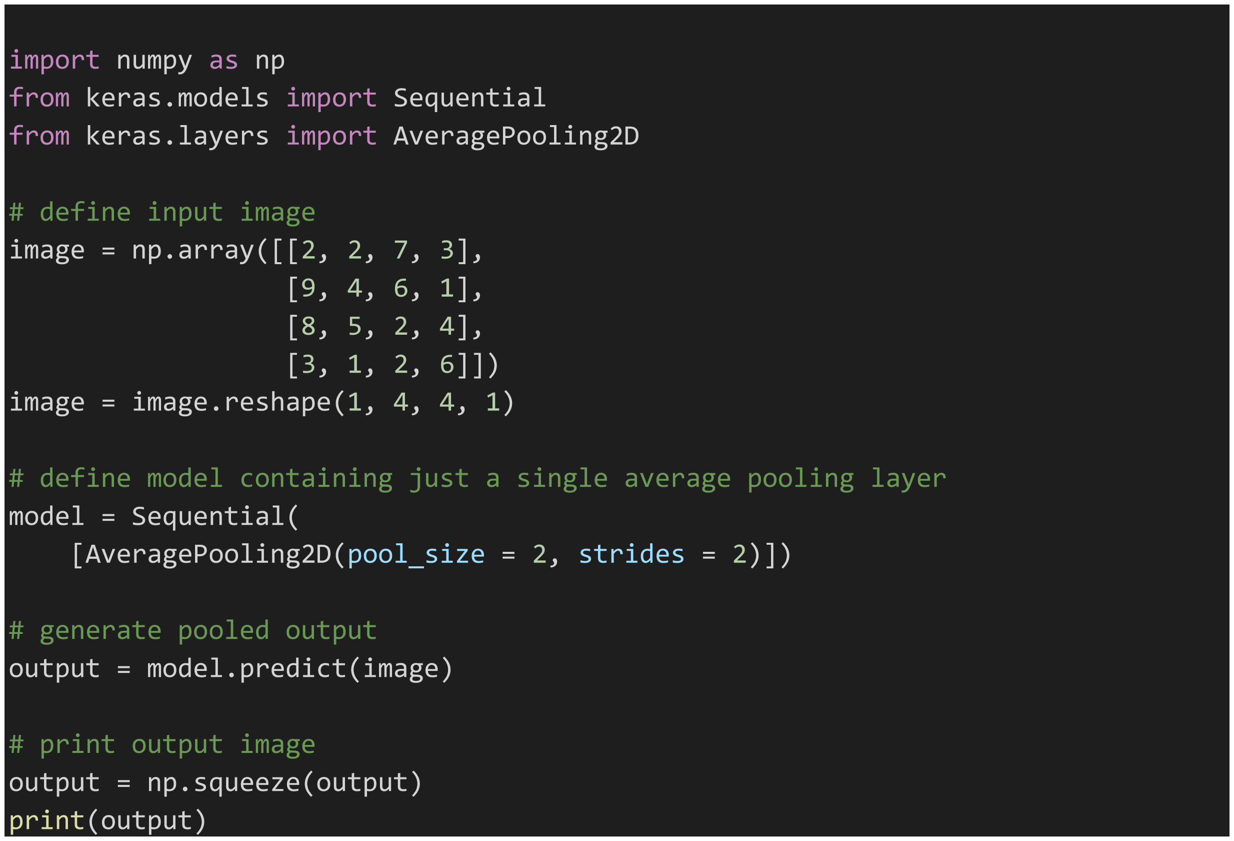


```
        import numpy as np
from keras.models import Sequential
from keras.layers import MaxPooling2D

# define input image
image = np.array([[2, 2, 7, 3],
                  [9, 4, 6, 1],
                  [8, 5, 2, 4],
                  [3, 1, 2, 6]])
image = image.reshape(1, 4, 4, 1)

# define model containing just a single max pooling layer
model = Sequential(
    [MaxPooling2D(pool_size = 2, strides = 2)])

# generate pooled output
output = model.predict(image)

# print output image
output = np.squeeze(output)
print(output)
```

2.Average Pooling

Average pooling computes the average of the elements present in the region of feature map covered by the filter. Thus, while max pooling gives the most prominent feature in a particular patch of the feature map, average pooling gives the average of features present in a patch.





```python
import numpy as np
from keras.models import Sequential
from keras.layers import AveragePooling2D

# define input image
image = np.array([[2, 2, 7, 3],
                  [9, 4, 6, 1],
                  [8, 5, 2, 4],
                  [3, 1, 2, 6]])
image = image.reshape(1, 4, 4, 1)

# define model containing just a single average pooling layer
model = Sequential(
    [AveragePooling2D(pool_size = 2, strides = 2)])

# generate pooled output
output = model.predict(image)

# print output image
output = np.squeeze(output)
print(output)
```

3. Global Pooling
   Global pooling reduces each channel in the feature map to a single value. Thus, an nh x nw x nc feature map is reduced to 1 x 1 x nc feature map. This is equivalent to using a filter of dimensions nh x nw i.e. the dimensions of the feature map. Further, it can be either global max pooling or global average pooling.

REFERENCES

https://www.ibm.com/cloud/learn/recurrent-neural-networks

https://www.simplilearn.com/tutorials/deep-learning-tutorial/rnn

https://www.analyticsvidhya.com/blog/2021/03/introduction-to-long-short-term-memory-lstm/

https://www.exxactcorp.com/blog/Deep-Learning/5-types-of-lstm-recurrent-neural-networks-and-what-to-do-with-them

https://databasecamp.de/en/ml/lstms

https://intellipaat.com/blog/what-is-lstm/

https://www.analyticsvidhya.com/blog/2021/05/convolutional-neural-networks-cnn/