PROJECT REPORT

ON

"Classification of SEM using machine learning "

Submitted in partial fulfilment of Degree of

Master of Science
(Physics)

Submitted by

Mr. Darshan Vijay Satone
(M.Sc 2nd year Physics)

Under the guidance of

Dr. Prashant .M. Gade
Professor,
Department Of Physics

Post Graduate Teaching Department of Physics
Rashtrasant Tukadoji Maharaj Nagpur University
,Nagpur

[2018-19]

PROJECT REPORT

ON

"Classification of SEM using machine learning "

Submitted in partial fulfilment of Degree of

Master of Science

(Physics)

Submitted by

Mr. Darshan Vijay Satone

(M.Sc 2nd year Physics)

Under the guidance of

Dr. Prashant .M. Gade

Professor,

Department Of Physics

Post Graduate Teaching Department of Physics

Rashtrasant Tukadoji Maharaj Nagpur University

,Nagpur

[2018-19]

CERTIFICATE

It is certify that the project entitled **"Classification of SEM images using machine learning"** which is being submitted **Mr. Darshan Vijay Satone** in the partial fulfilment towards attaining the Degree of " **MASTER OF SCIENCE** " in **Physics** (faculty of science) . Post Graduate Teaching Department of the Physics , Rashtrashant Tukadoji Maharaj Nagpur University , Nagpur . It is the record of her work carried out by her under my guidance . I found that the work is comprehensive and completed in all respect and ready for the submission to the University.

25/04/19

Head Deptt. of Physics
Rashtrasant Tukadoji Maharaj
Nagpur University, Nagpur

Approved By

Dr. Prashant .M.Gade
Professor & Head ,
Department Of Physics ,
Rashtrasant Tukadoji Maharaj Nagpur University,
Nagpur – 440033,

DECLARATION

I hereby, declare that the piece of work presented in this project report entitled, **"Classification of SEM images using machine learning"** has been carried out by me under the guidance of
Prof .**P.M.Gade.**
The work has been carried out at the **Post Graduate Teaching Department of the Physics , Rashtrashant Tukadoji Maharaj Nagpur University , Nagpur** it has been submitted as the project work in Degree of Master of Science in Physics . The result and conculsion given in the project work has been arrived as a consequence of the research work carried out by me.

Place : Nagpur
Date : 25/04/19

(Mr. Darshan Vijay Satone)

CERTIFIED BY :
Dr. Prashant .M.Gade
Professor & Head ,
Department Of Physics ,
Rashtrasant Tukadoji Maharaj Nagpur University,Nagpur .

ACKNOWLEDGMENT

I express may deep gratitude to **Dr. P.M.Gade**, Post Graduate Teaching Department of the Physics , Rashtrashant Tukadoji Maharaj Nagpur University , Nagpur for initially suggesting the problem and guidance throughout the progress of work. The critical observation, encouragement and suggestion given by him from time to time cannot be expressed in word.Also Pankaj sir,Pratik sir amd Priyanka mam for there constant support.

It is great pleasure to acknowledge the cooperation extended to me by all the teaching and non – teaching staff of the Department of Physics , **Rashtrashant Tukadoji Maharaj Nagpur University , Nagpur** .

I am very thankful to my dear friends **Ashwini Gadge , Sourabh Khamdelwal, Trupti Sharma ,Sarah, Ashwini , Bhagyashri** and **Prerana and** all my collegues for their support and help.

Last but not the least , I express my feeling to my family . They always supported me during the course of my life .

Darshan Satone
M.Sc II(Physics)

INDEX

Chapters	Page Number
Chapter 1 Preliminary Background	
1.1) An Introduction to machine learning	7
Chapter 2 Classification Of machine learning	
2.0) Classification	10
2.1) Regression and Classification: Supervised Learning	11
2.2) Regression	12
2.3) Classification	13
2.4) Confusion matrix	14
Chapter 3 What are artificial neural network	
3.0)Artificial Neural Network	16
3.1)Basic structure of ANN	16
3.2)Convolusion neural network	20
Chapter 4 The 7 steps of machine learning	
4.1)The 7 steps of machine learning	24
Project Work	34
Results	50
References	55

AN INTRODUCTION
TO MACHINE LEARNING

Chapter :1
An introduction to Machine Learning

The term Machine Learning was coined by Arthur Samuel in 1959,an American pioneer in the field of computer gaming and artificial intelligence and stated that "it gives computers the ability to learn without being explicitly programmed".
And in 1997, Tom Mitchell gave a "well-posed" mathematical and relational definition that "A computer program is said to learn from experience E with respect to some task T and some performance measure P, if its performance on T, as measured by P, improves with experience E.

Machine Learning is a latest buzzword floating around. It deserves to, as it is one of the most interesting subfield of Computer Science.

So what does Machine Learning really mean?
Let's try to understand Machine Learning in layman terms. Consider you are trying to toss a paper to a dustbin.

After first attempt, you realize that you have put too much force in it. After second attempt, you realize you are closer to target but you need to increase your throw angle. What is happening here is basically after every throw we are learning something and improving the end result. We are programmed to learn from our experience.

This implies that the tasks in which machine learning is concerned offers a fundamentally operational definition rather than defining the field in cognitive terms. This follows Alan Turing's proposal in his paper "Computing Machinery and Intelligence", in which the question "Can machines think?" is replaced with

the question "Can machines do what we (as thinking entities) can do?"

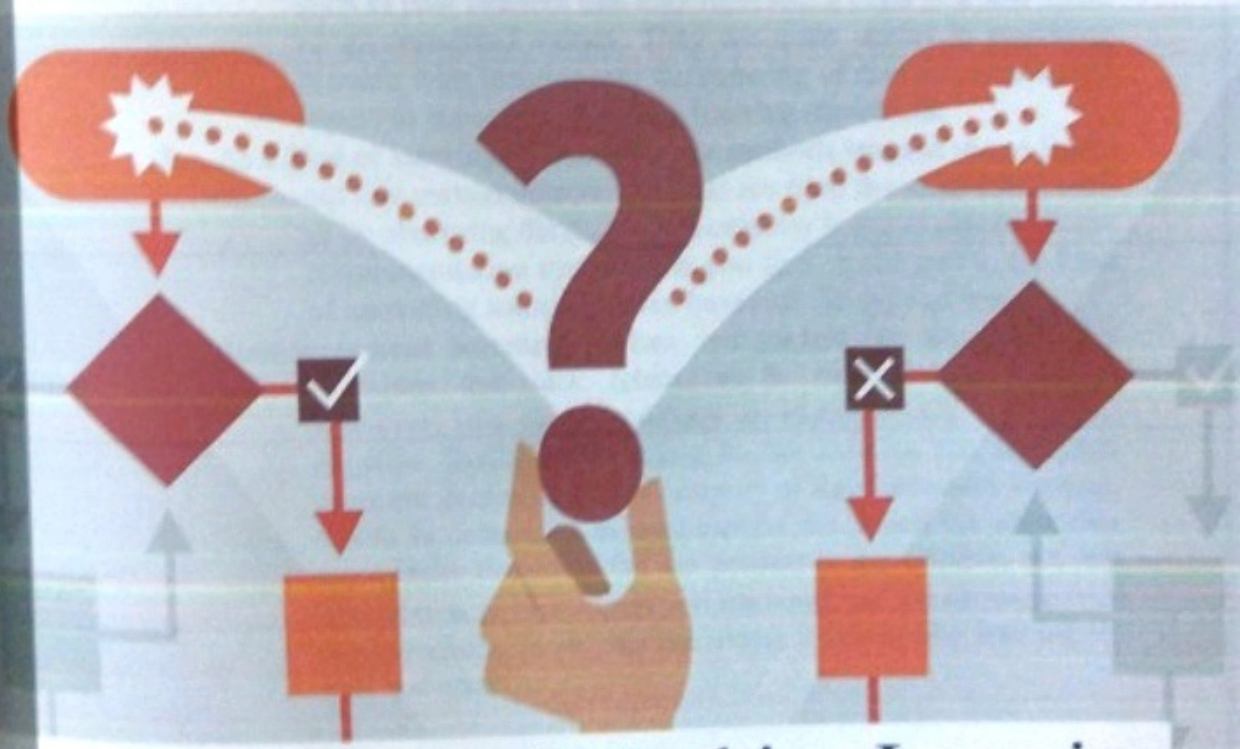
Within the field of data analytics, machine learning is used to devise complex models and algorithms that lend themselves to prediction; in commercial use, this is known as predictive analytics.

These analytical models allow researchers, data scientists, engineers, and analysts to "produce reliable, repeatable decisions and results" and uncover "hidden insights" through learning from historical relationships and trends in the data set(input).

Suppose that you decide to check out that offer for a vacation . You browse through the travel agency website and search for a hotel. When you look at a specific hotel, just below the hotel description there is a section titled "You might also like these hotels". This is a common use case of Machine Learning called "Recommendation Engine". Again, many data points were used to train a model in order to predict what will be the best hotels to show you under that section, based on a lot of information they already know about you.

So if you want your program to predict, for example, traffic patterns at a busy intersection (task T), you can run it through a machine learning algorithm with data about past traffic patterns (experience E) and, if it has successfully "learned", it will then do better at predicting future traffic patterns (performance measure P).

The highly complex nature of many real-world problems, though, often means that inventing specialized algorithms that will solve them perfectly every time is impractical, if not impossible. Examples of machine learning problems include, "Is this cancer?", "Which of these people are good friends with each other?", "Will this person like this movie?" such problems are excellent targets for Machine Learning, and in fact machine learning has been applied such problems with great success.



Classification of Machine Learning

Chapter :2
Classification of Machine Learning

Machine learning implementations are classified into three major categories, depending on the nature of the learning "signal" or "response" available to a learning system which are as follows:-

1. **Supervised learning :** When an algorithm learns from example data and associated target responses that can consist of numeric values or string labels, such as classes or tags, in order to later predict the correct response when posed with new examples comes under the category of Supervised learning. This approach is indeed similar to human learning under the supervision of a teacher. The teacher provides good examples for the student to memorize, and the student then derives general rules from these specific examples.
2. **Unsupervised learning :**Whereas when an algorithm learns from plain examples without any associated response, leaving to the algorithm to determine the data patterns on its own. This type of algorithm tends to restructure the data into something else, such as new features that may represent a class or a new series of un-correlated values. They are quite useful in providing humans with insights into the meaning of data and new useful inputs to supervised machine learning algorithms.
3. As a kind of learning, it resembles the methods humans use to figure out that certain objects or events are from the same class, such as by observing the degree of similarity between objects. Some recommendation systems that you find on the web in the form of marketing automation are based on this type of learning.
4. **Reinforcement learning :** When you present the algorithm with examples that lack labels, as in unsupervised learning. However, you can accompany an example with positive or negative feedback according to the solution the algorithm proposes comes under the category of Reinforcement learning, which is connected to applications for which the algorithm must make decisions (so the product is prescriptive, not just descriptive, as in unsupervised learning), and the decisions bear consequences. In the human world, it is just like learning by trial and error.

5. Errors help you learn because they have a penalty added (cost, loss of time, regret, pain, and so on), teaching you that a certain course of action is less likely to succeed than others. An interesting example of reinforcement learning occurs when computers learn to play video games by themselves.
6. In this case, an application presents the algorithm with examples of specific situations, such as having the gamer stuck in a maze while avoiding an enemy.

7. The application lets the algorithm know the outcome of actions it takes, and learning occurs while trying to avoid what it discovers to be dangerous and to pursue survival. You can have a look at how the company Google Deep Mind has created a reinforcement learning program that plays old Atari's videogames. When watching the video, notice how the program is initially clumsy and unskilled but steadily improves with training until it becomes a champion.

8. **Semi-supervised learning :** where an incomplete training signal is given: a training set with some (often many) of the target outputs missing. There is a special case of this principle known as Transduction where the entire set of problem instances is known at learning time, except that part of the targets are missing.

Another categorization of machine learning tasks arises when one considers the desired output of a machine-learned system:

1. **Classification :** When inputs are divided into two or more classes, and the learner must produce a model that assigns unseen inputs to one or more (multi-label classification) of these classes. This is typically tackled in a supervised way. Spam filtering is an example of classification, where the inputs are email (or other) messages and the classes are "spam" and "not spam".
2. **Regression :** Which is also a supervised problem, A case when the outputs are continuous rather than discrete.
3. **Clustering :** When a set of inputs is to be divided into groups. Unlike in classification, the groups are not known beforehand, making this typically an unsupervised task.

2.1 Regression and Classification | Supervised Machine Learning

What is Regression and Classification in Machine Learning?

Data scientists use many different kinds of machine learning algorithms to discover patterns in big data that lead to actionable insights. At a high level, these different algorithms can be classified into two groups based on the way they "learn" about data to make predictions: supervised and unsupervised learning.

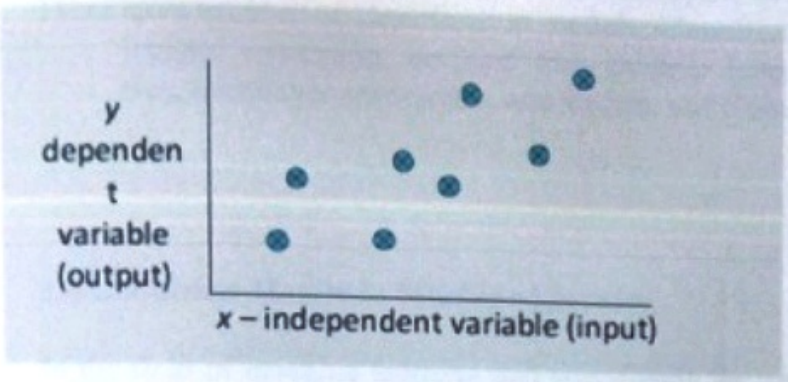
Supervised Machine Learning: The majority of practical machine learning uses supervised learning. Supervised learning is where you have input variables (x) and an output variable (Y) and you use an algorithm to learn the mapping function from the input to the output **Y = f(X)** . The goal is to approximate the mapping function so well that when you have new input data (x) that you can predict the output variables (Y) for that data.

Techniques of Supervised Machine Learning algorithms include **linear** and **logistic regression**, **multi-class classification**, **Decision Trees** and **support vector machines**. Supervised learning requires that the data used to train the algorithm is already labeled with correct answers. For example, a classification algorithm will learn to identify animals after being trained on
a dataset of images that are properly labeled with the species of the animal and some identifying characteristics.

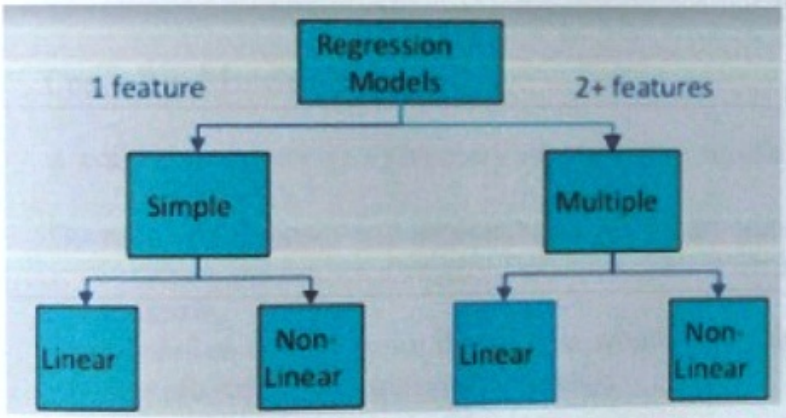
Supervised learning problems can be further grouped into **Regression** and **Classification** problems. Both problems have as goal the construction of a succinct model that can predict the value of the dependent attribute from the attribute variables. The difference between the two tasks is the fact that the dependent attribute is numerical for regression and categorical for classification.

2.2 Regression

A regression problem is when the output variable is a real or continuous value, such as "salary" or "weight". Many different models can be used, the simplest is the linear regression. It tries to fit data with the best hyper-plane which goes through the points.



Types of Regression Models:



2.3 Classification

A classification problem is when the output variable is a category, such as "red" or "blue" or "disease" and "no disease". A classification model attempts to draw some conclusion from observed values. Given one or more inputs a classification model will try to predict the value of one or more outcomes.

For example, when filtering emails "spam" or "not spam", when looking at transaction data, "fraudulent", or "authorized". In short Classification

either predicts categorical class labels or classifies data (construct a model) based on the training set and the values (class labels) in classifying attributes and uses it in classifying new data.

There are a number of classification models. Classification models include logistic regression, decision tree, random forest, gradient-boosted tree, multilayer perceptron, one-vs-rest, and Naive Bayes.

2.4 Confusion Matrix in Machine Learning

In the field of machine learning and specifically the problem of statistical classification, a confusion matrix, also known as an error matrix.

A confusion matrix is a table that is often used to describe the performance of a classification model (or "classifier") on a set of test data for which the true values are known. It allows the visualization of the performance of an algorithm.

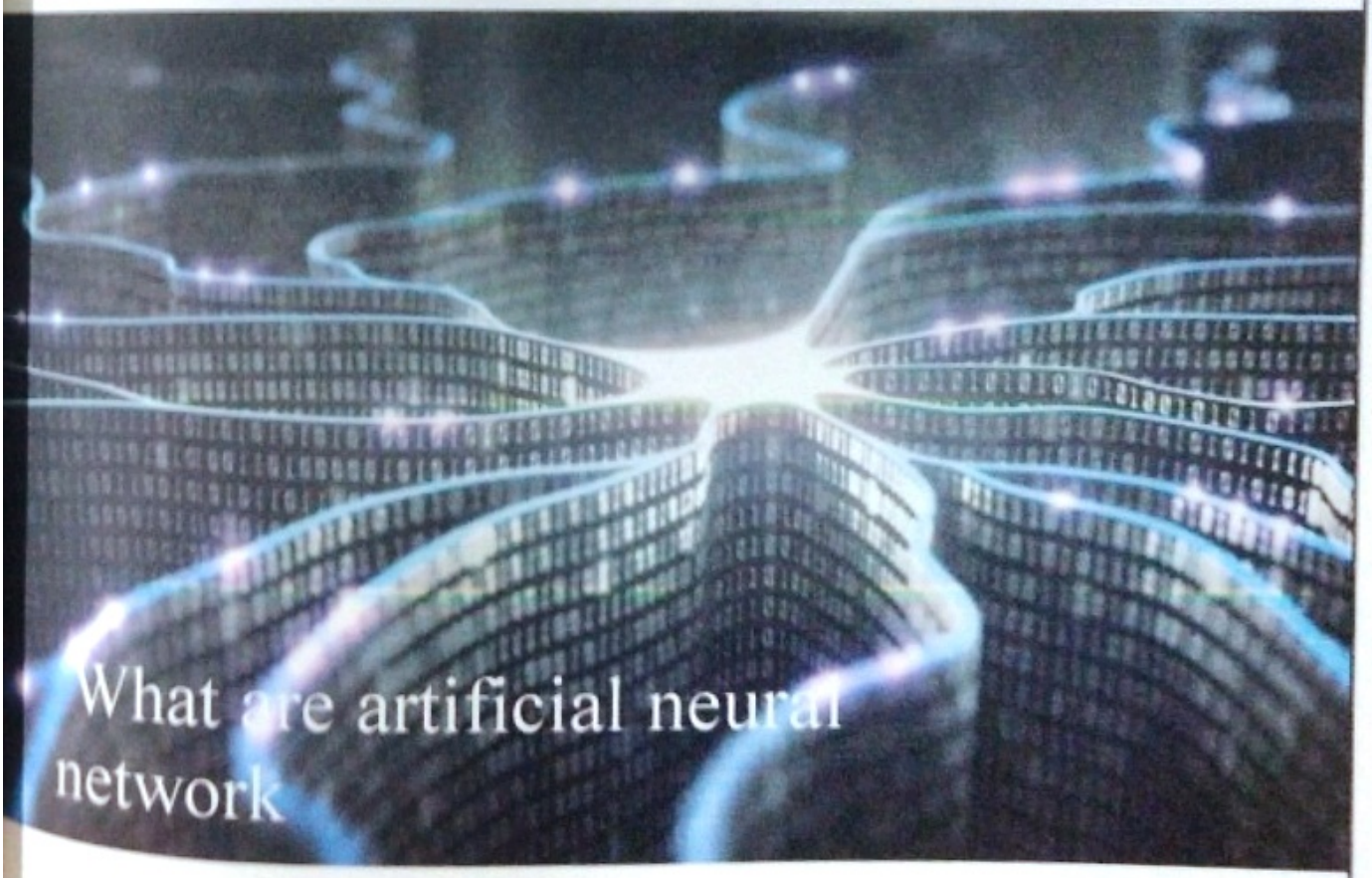
It allows easy identification of confusion between classes e.g. one class is commonly mislabeled as the other. Most performance measures are computed from the confusion matrix.

Confusion Matrix:

A confusion matrix is a summary of prediction results on a classification problem.

The number of correct and incorrect predictions are summarized with count values and broken down by each class. This is the key to the confusion matrix.

The confusion matrix shows the ways in which your classification model is confused when it makes predictions.



What are artificial neural network

The inventor of the first neurocomputer, Dr. Robert Hecht-Nielsen, defines a neural network as =

"...a computing system made up of a number of simple, highly interconnected processing elements, which process information by their dynamic state response to external inputs."

Neural networks are artificial systems that were inspired by biological neural networks. These systems learn to perform tasks by being exposed to various datasets and examples without any task-specific rules. The idea is that the system generates identifying characteristics from the data they have been passed without being programmed with a pre-programmed understanding of these datasets.

Neural networks are based on computational models for threshold logic. Threshold logic is a combination of algorithms and mathematics. Neural networks are based either on the study of the brain or on the application of neural networks to artificial intelligence. The work has led to improvements in finite automata theory.

Components of a typical neural network involve neurons, connections, weights, biases, propagation function, and a learning rule. Neurons will receive an input from predecessor neurons that have an activation , threshold , an activation function f, and an output function . Connections consist of connections, weights and biases which rules how neuron transfers output to neuron . Propagation computes the input and outputs the output and sums the predecessor neurons function with the weight. The learning rule modifies the weights and thresholds of the variables in the network.

3.1Basic structure of ANN:

The idea of ANNs is based on the belief that working of human brain by making the right connections, can be imitated using silicon and wires as living **neurons** and **dendrites**.

The human brain is composed of 86 billion nerve cells called **neurons**. They are connected to other thousand cells by **Axons**. Stimuli from external environment or inputs from sensory organs are accepted by dendrites.

Chapter :3

What are artificial neural network

The inventor of the first neurocomputer, Dr. Robert Hecht-Nielsen, defines a neural network as –

"...a computing system made up of a number of simple, highly interconnected processing elements, which process information by their dynamic state response to external inputs."

Neural networks are artificial systems that were inspired by biological neural networks. These systems learn to perform tasks by being exposed to various datasets and examples without any task-specific rules. The idea is that the system generates identifying characteristics from the data they have been passed without being programmed with a pre-programmed understanding of these datasets.

Neural networks are based on computational models for threshold logic. Threshold logic is a combination of algorithms and mathematics. Neural networks are based either on the study of the brain or on the application of neural networks to artificial intelligence. The work has led to improvements in finite automata theory.

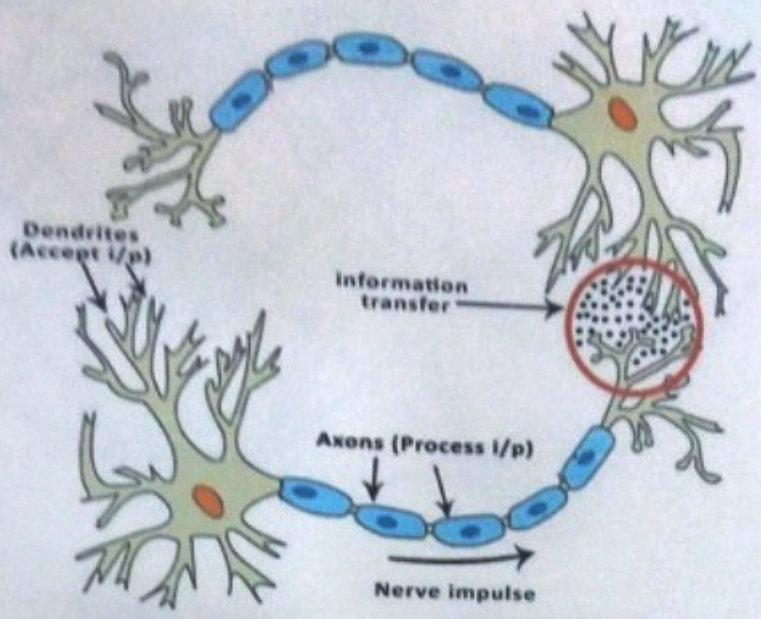
Components of a typical neural network involve neurons, connections, weights, biases, propagation function, and a learning rule. Neurons will receive an input from predecessor neurons that have an activation , threshold , an activation function f, and an output function . Connections consist of connections, weights and biases which rules how neuron transfers output to neuron . Propagation computes the input and outputs the output and sums the predecessor neurons function with the weight. The learning rule modifies the weights and thresholds of the variables in the network.

3.1Basic structure of ANN:

The idea of ANNs is based on the belief that working of human brain by making the right connections, can be imitated using silicon and wires as living **neurons** and **dendrites**.

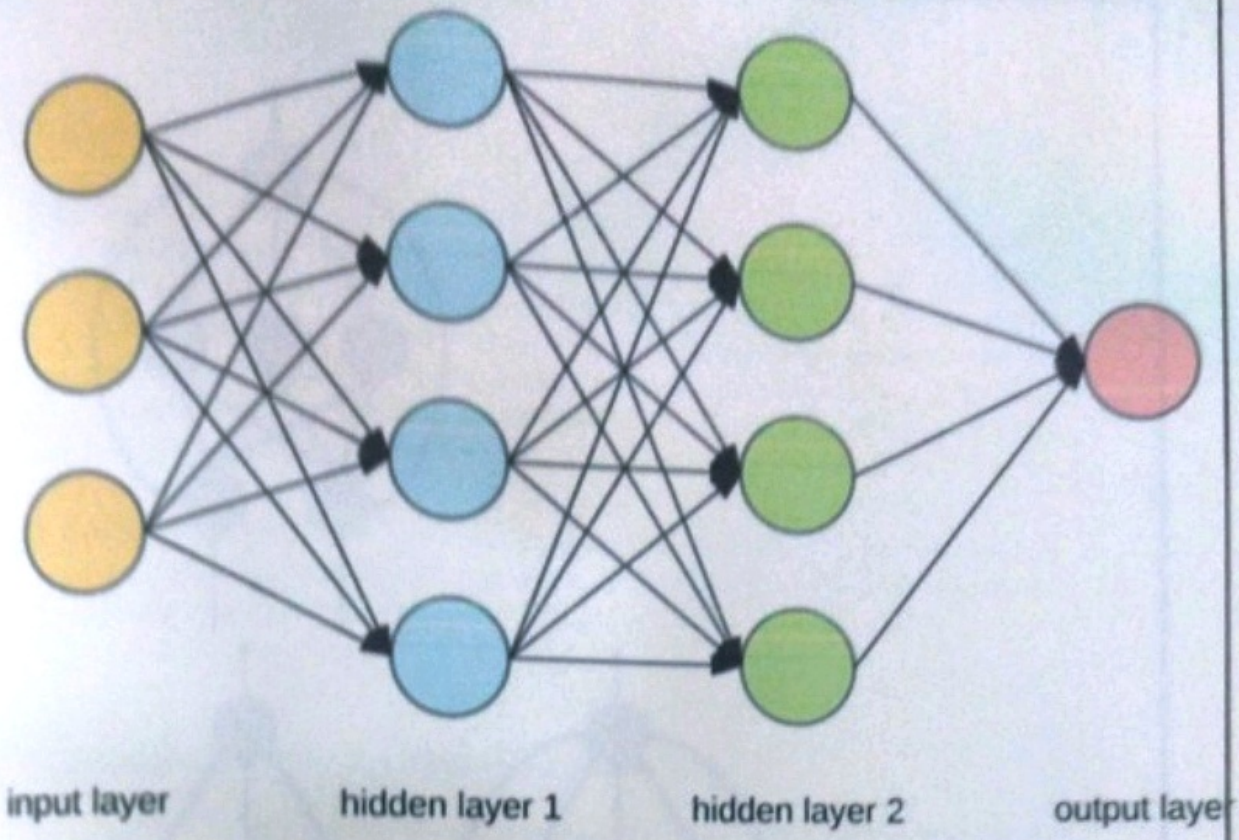
The human brain is composed of 86 billion nerve cells called **neurons.** They are connected to other thousand cells by **Axons.** Stimuli from external environment or inputs from sensory organs are accepted by dendrites.

These inputs create electric impulses, which quickly travel through the neural network. A neuron can then send the message to other neuron to handle the issue or does not send it forward.



ANNs are composed of multiple **nodes**, which imitate biological **neurons** of human brain. The neurons are connected by links and they interact with each other. The nodes can take input data and perform simple operations on the data. The result of these operations is passed to other neurons. The output at each node is called its **activation** or **node value.**

Each link is associated with **weight.** ANNs are capable of learning, which takes place by altering weight values. The following illustration shows a simple ANN –



Types of Artificial Neural Networks

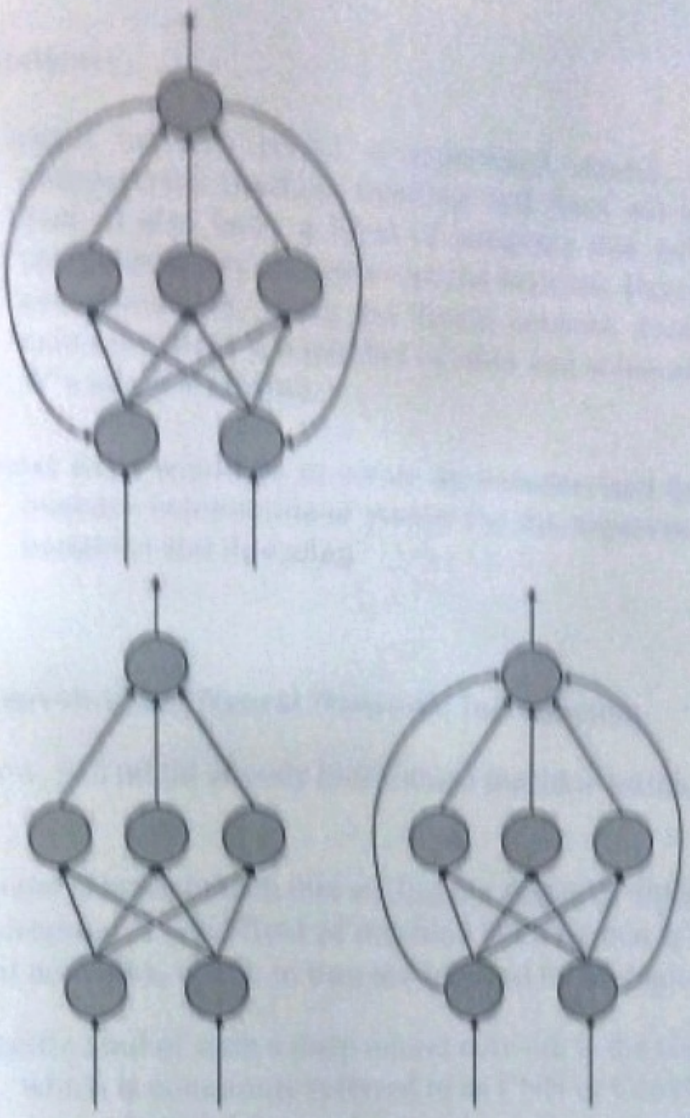
There are two Artificial Neural Network topologies – **FeedForward** and **Feedback.**

FeedForward ANN

In this ANN, the information flow is unidirectional. A unit sends information to other unit from which it does not receive any information. There are no feedback loops. They are used in pattern generation/recognition/classification. They have fixed inputs and outputs.

FeedBack ANN

Here, feedback loops are allowed. They are used in content addressable memories.



Working of ANNs

In the topology diagrams shown, each arrow represents a connection between two neurons and indicates the pathway for the flow of information. Each connection has a weight, an integer number that controls the signal between the two neurons.

If the network generates a "good or desired" output, there is no need to adjust the weights. However, if the network generates a "poor or undesired" output or an error, then the system alters the weights in order to improve subsequent results.

Limitations:

The neural network is for a supervised model. It does not handle unsupervised machine learning and does not cluster and associate data. It also lacks a level of accuracy that will be found in more computationally expensive neural network. Based on Andrew Trask's neural network. Also, the neural network does not work with any matrices where X's number of rows and columns do not match Y and W's number of rows.

The next steps would be to create an unsupervised neural network and to increase computational power for the supervised model with more iterations and threading.

3.2Convolutional Neural Network: Introduction

By now, you might already know about machine learning and deep learning, a

computer science branch that studies the design of algorithms that can learn. Deeplearning is a subfield of machine learning that is inspired by artificial neural networks, which in turn are inspired by biological neural networks.

A specific kind of such a deep neural network is the convolutional network, which is commonly referred to as CNN or ConvNet. It's a deep, feed-forward artificial neural network.

Remember that feed-forward neural networks are also called multi-layer perceptrons(MLPs), which are the quintessential deep learning models. The models are called "feed-forward" because information flows right through the model. There are no feedback connections in which outputs of the model are fed back into itself.

CNNs specifically are inspired by the biological visual cortex. The cortex has small regions of cells that are sensitive to the specific areas of the visual field. This idea was expanded by a captivating experiment done by Hubel and Wiesel in 1962 . In this experiment, the researchers showed that some individual neurons in the brain activated or fired only in the presence of edges of a particular orientation like vertical or horizontal edges.

For example, some neurons fired when exposed to vertical sides and some when shown a horizontal edge.

Hubel and Wiesel found that all of these neurons were well ordered in a columnar fashion and that together they were able to produce visual perception. This idea of specialized components inside of a system having specific tasks is one that machines use as well and one that you can also find back in CNNs.

Convolutional neural networks have been one of the most influential innovations in the field of computer vision. They have performed a lot better than traditional computer vision and have produced state-of-the-art results. These neural networks have proven to be successful in many different real-life case studies and applications, like:

- Image classification, object detection, segmentation, face recognition;
- Self driving cars that leverage CNN based vision systems;
- Classification of crystal structure using a convolutional neural network;
- And many more, of course!

To understand this success, you'll have to go back to 2012, the year in which Alex Krizhevsky used convolutional neural networks to win that year's ImageNet Competition, reducing the classification error from 26% to 15%.

Note that ImageNet Large Scale Visual Recognition Challenge (ILSVRC) began in the year 2010 is an annual competition where research teams assess their algorithms on the given data set and compete to achieve higher accuracy on several visual recognition tasks.

This was the time when neural networks regained prominence after quite some time. This is often called the "third wave of neural networks". The other two waves were in the 1940s until the 1960s and in the 1970s to 1980s.

Alright, you know that you'll be working with feed-forward networks that are inspired by the biological visual cortex, but what does that actually mean?

Take a look at the picture below:

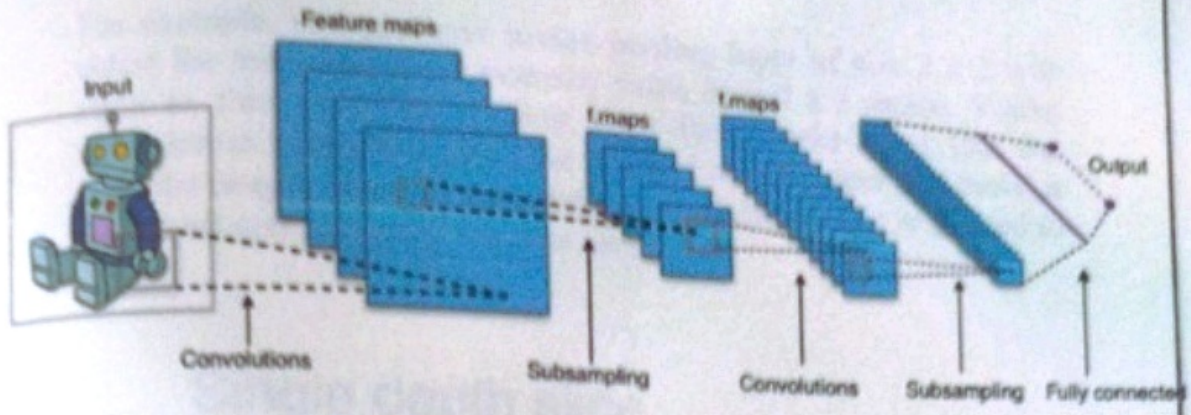


Figure: Convolutional Neural Network from Wikimedia

The image shows you that you feed an image as an input to the network, which goes through multiple convolutions, subsampling, a fully connected layer and finally outputs something.

But what are all these concepts?

1. The convolution layer computes the output of neurons that are connected to local regions or receptive fields in the input, each computing a dot product between their weights and a small receptive field to which they are connected to in the input volume. Each computation leads to extraction of a feature map from the input image. In other words, imagine you have an image represented as a 5x5 matrix of values, and you take a 3x3 matrix and slide that 3x3 window or kernel around the image. At each position of that matrix, you multiply the values of your 3x3 window by the values in the image that are currently being covered by the window. As a result, you'll get a single number that represents all the values in that window of the images. You use this layer to filtering: as the window moves over the image, you check for patterns in that section of the image. This works because of filters, which are multiplied by the values outputted by the convolution.

2. The objective of subsampling is to get an input representation by reducing its dimensions, which helps in reducing overfitting. One of the techniques of subsampling is max pooling.

3. With this technique, you select the highest pixel value from a region depending on its size. In other words, max pooling takes the largest value from the window of the image currently covered by the kernel.

For example, you can have a max-pooling layer of size 2 x 2 will select the maximum pixel intensity value from 2 x 2 region. You're right to think that the pooling layer then works a lot like the convolution layer! You also take a kernel or a window and move it over the image; The only difference is the function that is applied to the kernel and the image window isn't linear.

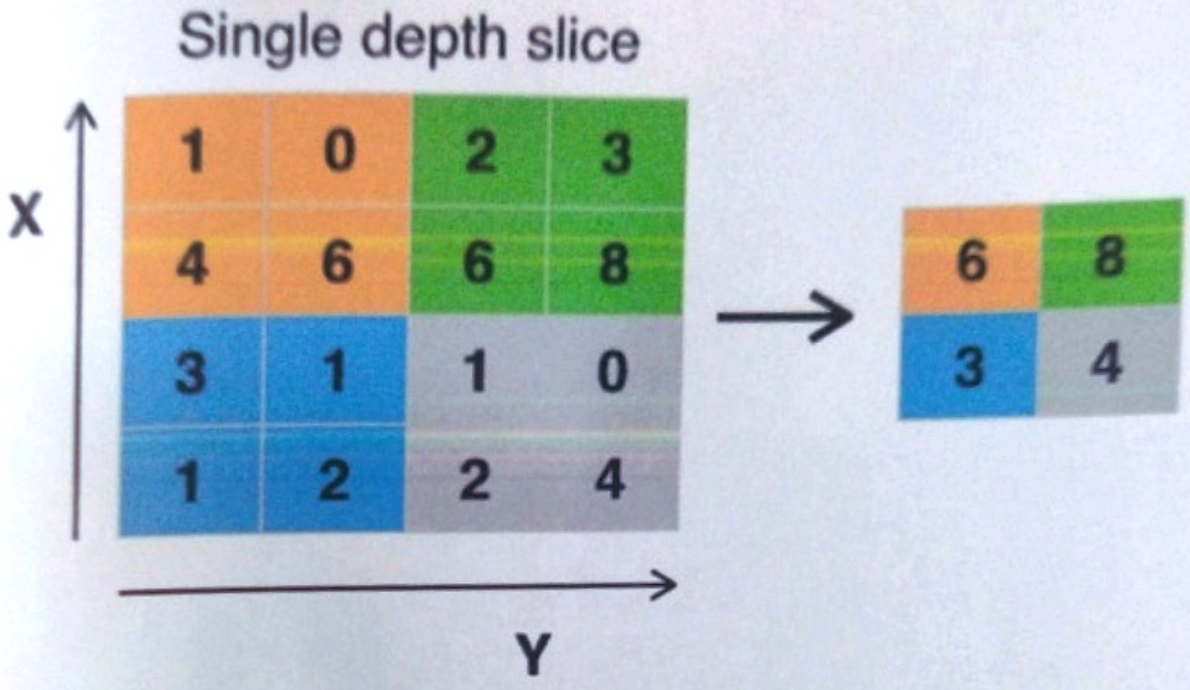


Figure: Max-Pooling from Wikipedia

4. The objective of the fully connected layer is to flatten the high-level features that are learned by convolutional layers and combining all the features. It passes the flattened output to the output layer where you use a softmax classifier or a sigmoid to predict the input class label.

7 Steps of Machine Learning

Chapter: 4
The 7 Steps of Machine Learning

From detecting skin cancer, to sorting cucumbers, to detecting escalators in need of repairs, machine learning has granted computer systems entirely new abilities.

But how does it really work under the hood? Let's walk through a basic example, and use it as an excuse talk about the process of getting answers from your data using machine learning.

Let's pretend that we've been asked to create a system that answers the question of whether a drink is wine or beer. This question answering system that we build is called a "model", and this model is created via a process called "training". The goal of training is to create an accurate model that answers our questions correctly most of the time. But in order to train a model, we need to collect data to train on.

Wine or Beer?

Our data will be collected from glasses of wine and beer. There are many aspects of the drinks that we *could* collect data on, everything from the amount of foam, to the shape of the glass.

Chapter: 4
The 7 Steps of Machine Learning

From detecting skin cancer, to sorting cucumbers, to detecting escalators in need of repairs, machine learning has granted computer systems entirely new abilities.

But how does it really work under the hood? Let's walk through a basic example, and use it as an excuse talk about the process of getting answers from your data using machine learning.

Let's pretend that we've been asked to create a system that answers the question of whether a drink is wine or beer. This question answering system that we build is called a "model", and this model is created via a process called "training". The goal of training is to create an accurate model that answers our questions correctly most of the time. But in order to train a model, we need to collect data to train on.

Wine or Beer?

Our data will be collected from glasses of wine and beer. There are many aspects of the drinks that we *could* collect data on, everything from the amount of foam, to the shape of the glass.

For our purposes, we'll pick just two simple ones: The color (as a wavelength of light) and the alcohol content (as a percentage). The hope is that we can split our two types of drinks along these two factors alone. We'll call these our "*features*" from now on: color, and alcohol.

The first step to our process will be to run out to the local grocery store and buy up a bunch of different beers and wine, as well as get some equipment to do our measurements—a spectrometer for measuring the color, and a hydrometer to measure the alcohol content. Our grocery store has an electronics hardware section :)

Gathering Data

Once we have our equipment and booze, it's time for our first real step of machine learning: **gathering data**. This step is very important because the quality and quantity of data that you gather will directly determine how good your predictive model can be. In this case, the data we collect will be the color and the alcohol content of each drink.

Color (nm)	Alcohol %	Beer or Wine?
610	5	Beer
599	13	Wine
693	14	Wine

This will yield a table of color, alcohol%, and whether it's beer or wine. This will be our **training data.**

Data preparation

A few hours of measurements later, we have gathered our training data. Now it's time for the next step of machine learning: **Data preparation**, where we load our data into a suitable place and prepare it for use in our machine learning training.

We'll first put all our data together, and then randomize the ordering. We don't want the order of our data to affect what we learn, since that's not part of determining whether a drink is beer or wine. In other words, we make a determination of what a drink is, independent of what drink came before or after it.

This is also a good time to do any pertinent visualizations of your data, to help you see if there are any relevant relationships between different variables you can take advantage of, as well as show you if there are any data imbalances. For example, if we collected way more data points about beer than wine, the model we train will be biased toward guessing that virtually everything that it sees is beer, since it would be right most of the time. However, in the real-world, the model may see beer and wine an equal amount, which would mean that guessing "beer" would be wrong half the time.

We'll also need to split the data in two parts. The first part, used in training our model, will be the majority of the dataset. The second part will be used for evaluating our trained model's performance. We don't want to use the same data that the model was trained on for evaluation, since it could then just memorize the "questions", just as you wouldn't use the same questions from your math homework on the exam.

Sometimes the data we collect needs other forms of adjusting and manipulation. Things like de-duping, normalization, error correction,

and more. These would all happen at the data preparation step. In our case, we don't have any further data preparation needs, so let's move forward.

Choosing a model

The next step in our workflow is choosing a model. There are many models that researchers and data scientists have created over the years. Some are very well suited for image data, others for sequences (like text, or music), some for numerical data, others for text-based data. In our case, since we only have 2 features, color and alcohol%, we can use a small linear model, which is a fairly simple one that should get the job done.

Training

Now we move onto what is often considered the bulk of machine learning—the **training**. In this step, we will use our data to incrementally improve our model's ability to predict whether a given drink is wine or beer.

In some ways, this is similar to someone first learning to drive. At first, they don't know how any of the pedals, knobs, and switches work, or when any of them should be used. However, after lots of practice and correcting for their mistakes, a licensed driver emerges. Moreover, after a year of driving, they've become quite adept. The act of driving and reacting to real-world data has adapted their driving abilities, honing their skills.

$$y = m * x + b$$

OUTPUT SLOPE INPUT Y-INTERCEPT

We will do this on a much smaller scale with our drinks. In particular, the formula for a straight line is y=m*x+b, where x is the input, m is the

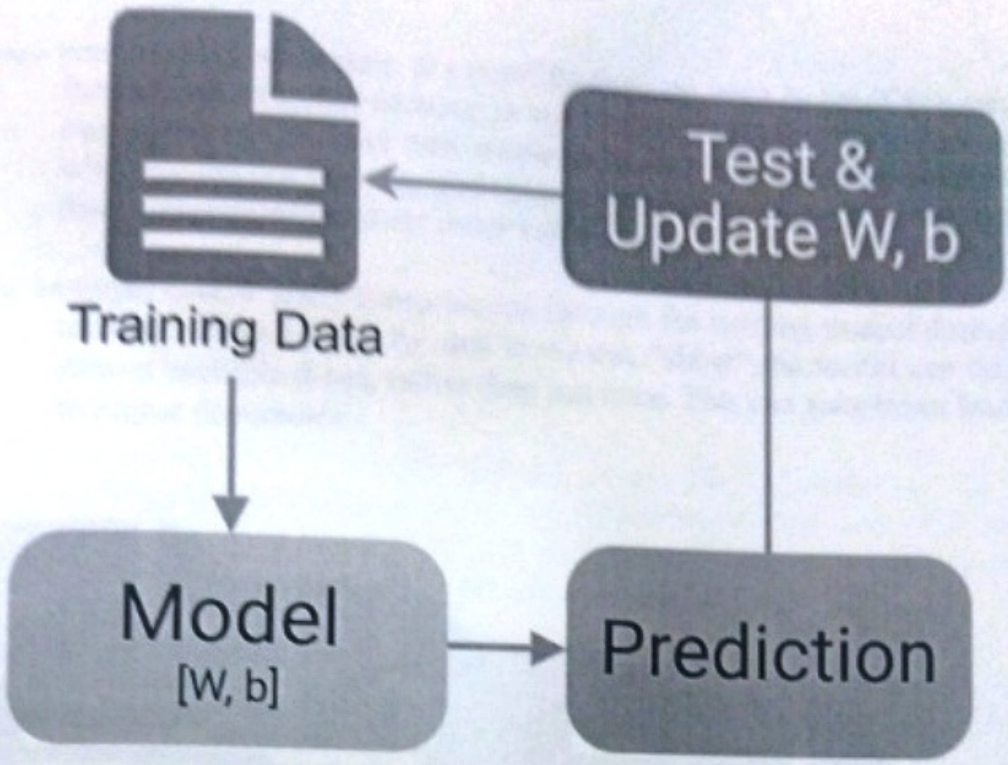
slope of that line, b is the y-intercept, and y is the value of the line at the position x. The values we have available to us for adjusting, or "training", are m and b. There is no other way to affect the position of the line, since the only other variables are x, our input, and y, our output.

$$\text{WEIGHTS} = \begin{bmatrix} m_{1,1} & m_{1,2} \\ m_{2,1} & m_{2,2} \\ m_{3,1} & m_{3,2} \end{bmatrix}$$

$$\text{BIASES} = \begin{bmatrix} b_{1,1} & b_{1,2} \\ b_{2,1} & b_{2,2} \\ b_{3,1} & b_{3,2} \end{bmatrix}$$

In machine learning, there are many m's since there may be many features. The collection of these m values is usually formed into a matrix, that we will denote W, for the "weights" matrix. Similarly for b, we arrange them together and call that the biases.

The training process involves initializing some random values for W and b and attempting to predict the output with those values. As you might imagine, it does pretty poorly. But we can compare our model's predictions with the output that it should produced, and adjust the values in W and b such that we will have more correct predictions.



This process then repeats. Each iteration or cycle of updating the weights and biases is called one training "step".

Let's look at what that means in this case, more concretely, for our dataset. When we first start the training, it's like we drew a random line through the data. Then as each step of the training progresses, the line moves, step by step, closer to an ideal separation of the wine and beer.

Evaluation

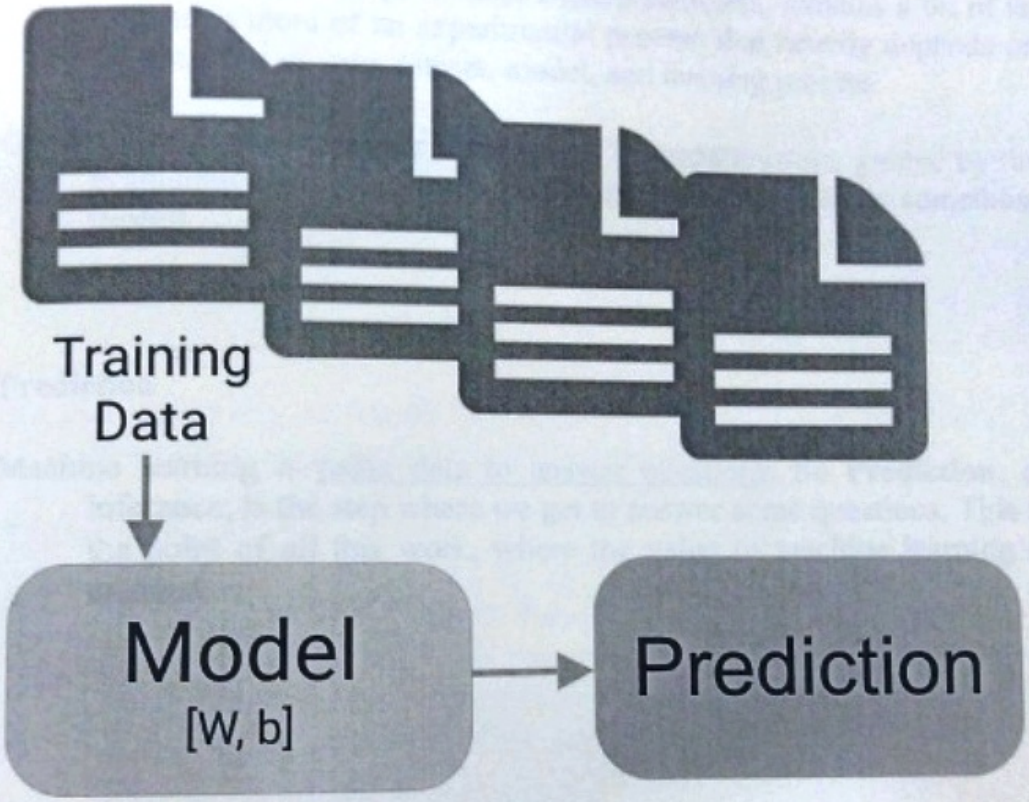
Once training is complete, it's time to see if the model is any good, using **Evaluation**. This is where that dataset that we set aside earlier comes into play. Evaluation allows us to test our model against data that has never been used for training. This metric allows us to see how the model might perform against data that it has not yet seen. This is meant to be representative of how the model might perform in the real world.

A good rule of thumb I use for a training-evaluation split somewhere on the order of 80/20 or 70/30. Much of this depends on the size of the original source dataset. If you have a lot of data, perhaps you don't need as big of a fraction for the evaluation dataset.

Parameter Tuning

Once you've done evaluation, it's possible that you want to see if you can further improve your training in any way. We can do this by **tuning our parameters**. There were a few parameters we implicitly assumed when we did our training, and now is a good time to go back and test those assumptions and try other values.

One example is how many times we run through the training dataset during training. What I mean by that is we can "show" the model our full dataset multiple times, rather than just once. This can sometimes lead to higher accuracies.



Another parameter is "*learning rate*". This defines how far we shift the line during each step, based on the information from the previous training step. These values all play a role in how accurate our model can become, and how long the training takes.

For more complex models, initial conditions can play a significant role in determining the outcome of training. Differences can be seen depending on whether a model starts off training with values initialized to zeroes versus some distribution of values, which leads to the question of which distribution to use.

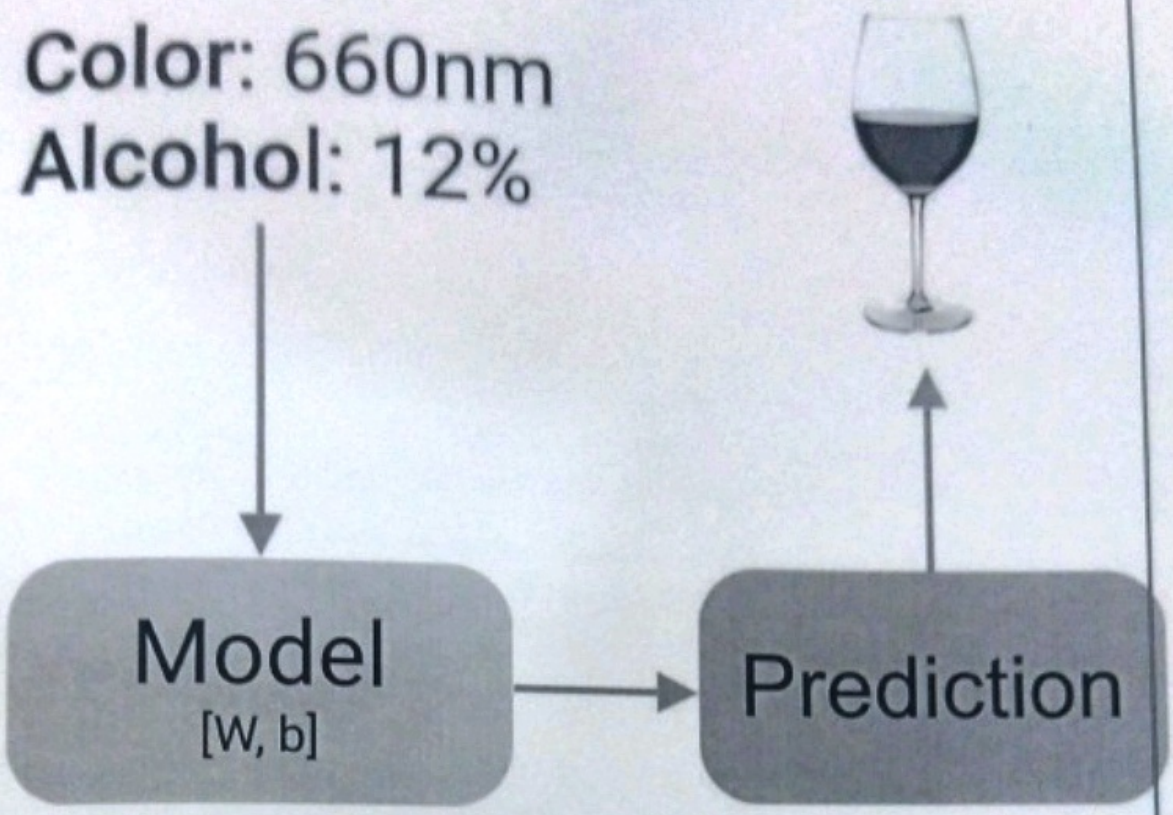
As you can see there are many considerations at this phase of training, and it's important that you define what makes a model "good enough", otherwise you might find yourself tweaking parameters for a very long time.

These parameters are typically referred to as "*hyperparameters*". The adjustment, or tuning, of these hyperparameters, remains a bit of an art, and is more of an experimental process that heavily depends on the specifics of your dataset, model, and training process.

Once you're happy with your training and hyperparameters, guided by the evaluation step, it's time to finally use your model to do something useful!

Prediction

Machine learning is <u>using data to answer questions</u>. So **Prediction**, or inference, is the step where we get to answer some questions. This is the point of all this work, where the value of machine learning is realized.



We can finally use our model to predict whether a given drink is wine or beer, given its color and alcohol percentage.

The big picture

The power of machine learning is that we were able to determine how to differentiate between wine and beer using our model, rather than using human judgement and manual rules. You can extrapolate the ideas presented today to other problem domains as well, where the same principles apply:

- Gathering data
- Preparing that data
- Choosing a model
- Training
- Evaluation
- Hyperparameter tuning
- Prediction.

Project Work

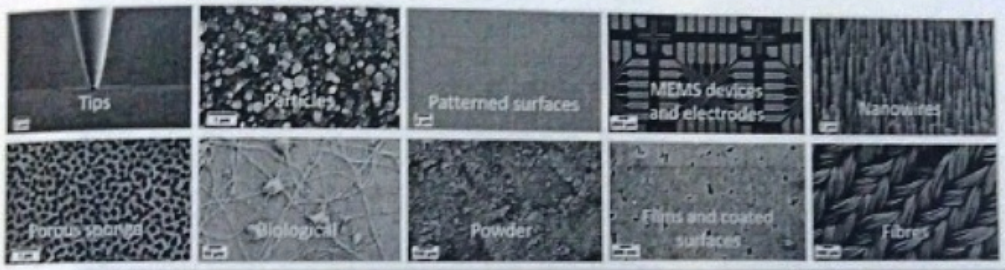
Goals

Since I'm a newcomer to this issue, I took a ready-made model from Keras blog. My goals were to understand how the model works, describe it; customise the model and teach it to recognise photos Scanning Electron Microscope of different materials.

Techniques and tools

I used Python syntax for this project. As a framework I used Keras, which is a high-level neural network API written in Python. But Keras can't work by itself, it needs a backend for low-level operations. Thus I installed a dedicated software library — Google's TensorFlow.

As a development environment I used the Anaconda Environment (Spyder). I used Matplotlib for visualisation.



Tag	Category	N images
L0	Porous_Sponge	171
L1	Patterned_surface	3310
L2	Particles	3412
L3	Films_Coated_Surface	308
L4	Powder	895
L5	Tips	1561
L6	Nanowires	3656
L7	Biological	953
L8	MEMS_devices_and_electrodes	4158
L9	Fibres	153
	TOTAL	18577

NFFA-EUROPE - SEM Dataset

by Aversa, Rossella; Modarres, Mohammad Hadi; Cozzini, Stefano; Ciancio, Regina;

Feb 19, 2018

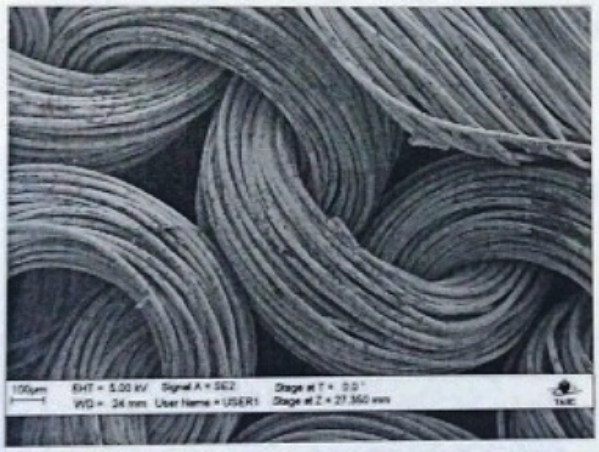
Last updated at Apr 16, 2018

For network training and tasting I used a dataset of photos Scanning Electron Microscope

Convolutional neural networks and image classification :

Convolutional neural networks (CNN) is a special architecture of arti>cial neural networks, proposed by Yann LeCun in 1988. CNN uses some features of the visual cortex. One of the most popular uses of this architecture is image classification. For example Facebook uses CNN for automatic tagging algorithms, Amazon — for generating product recommendations and Google — for search through among users' photos.

Let us consider the use of CNN for image classification in more detail. The main task of image classification is acceptance of the input image and the following definition of its class. This is a skill that people learn from their birth and are able to easily determine that the image in the picture is an elephant. But the computer sees the pictures quite differently:



01010100 01101000 01101001 01110011
00100000 01101001 01110011 00100000
01110100 01101000 01100101 00100000
01110100 01110101 01110100 01101111
01110010 01101001 01100001 01101100
00100000 01110100 01101111 00100000
01101100 01100101 01100001 01110010
01101110 00100000 01100010 01101001
01101110 01100001 01110010 01111001
00101110 00100000 01001001 00100000
01101000 01101111 01110000 01100101
00100000 01111001 01101111 01110101
00100000 01100101 01101110 01101101
01101111 01111001 00100000 01101001
01110100 00100001

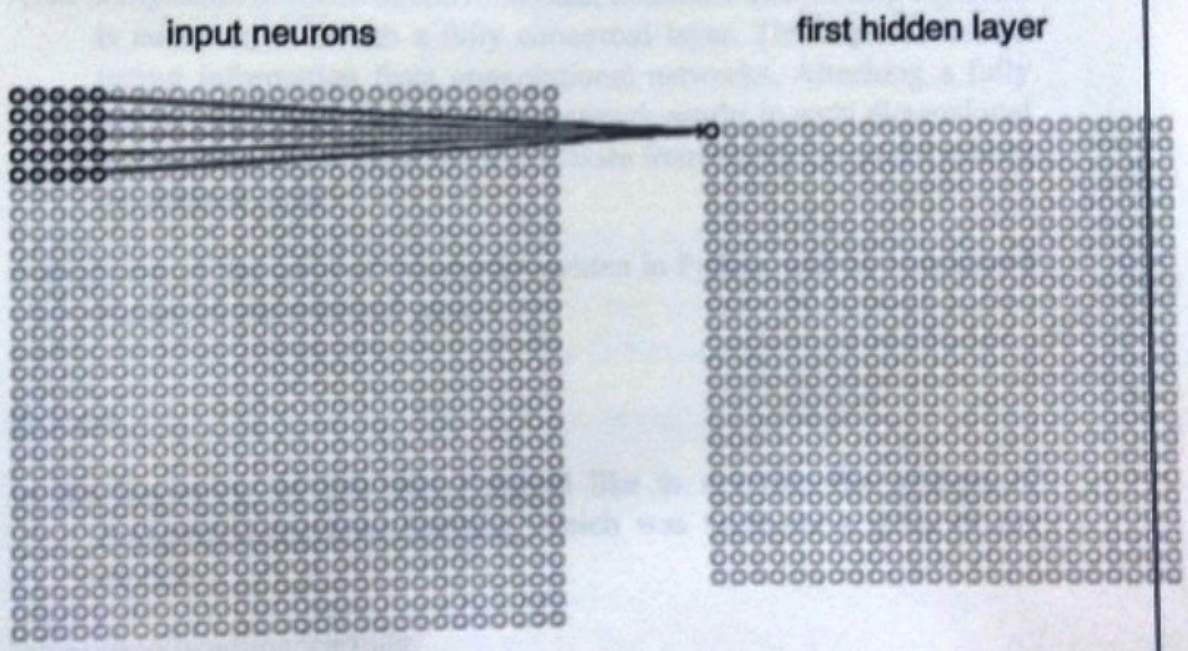
Instead of the image, the computer sees an array of pixels. For example, if image size is 300 x 300. In this case, the size of the array will be 300x300x3. Where 300 is width, next 300 is height and 3 is RGB channel values and 300x300x1 if it is black and white image. The computer is assigned a value from 0 to 255 to each of these numbers. This value describes the intensity of the pixel at each point.

To solve this problem the computer looks for the characteristics of the base level. In human understanding such characteristics are for example

the trunk or large ears. For the computer, these characteristics are boundaries or curvatures. And then through the groups of convolutional layers the computer constructs more abstract concepts.

In more detail: the image is passed through a series of convolutional, nonlinear, pooling layers and fully connected layers, and then generates the output.

The Convolution layer is always the first. The image (matrix with pixel values) is entered into it. Imagine that the reading of the input matrix begins at the top left of image. Next the software selects a smaller matrix there, which is called a @lter (or neuron, or core). Then the alter produces convolution, i.e. moves along the input image. The alter task is to multiply its values by the original pixel values. All these multiplications are summed up. One number is obtained in the end. Since the alter has read the image only in the upper left corner, it moves further and further right by 1 unit performing a similar operation. After passing the alter across all positions, a matrix is obtained, but smaller then a input matrix.



This operation, from a human perspective, is analogous to identifying boundaries and simple colours on the image. But in order to recognise the properties of a higher level such as the trunk or large ears the whole network is needed.

The network will consist of several convolutional networks mixed with nonlinear and pooling layers. When the image passes through one convolution layer, the output of the first layer becomes the input for the second layer. And this happens with every further convolutional layer.

The nonlinear layer is added after each convolution operation. It has an activation function, which brings nonlinear property. Without this property a network would not be sufficiently intense and will not be able to model the response variable (as a class label).

The pooling layer follows the nonlinear layer. It works with width and height of the image and performs a downsampling operation on them. As a result the image volume is reduced. This means that if some features (as for example boundaries) have already been identified in the previous convolution operation, then a detailed image is no longer needed for further processing, and it is compressed to less detailed pictures.

After completion of series of convolutional, nonlinear and pooling layers, it is necessary to attach a fully connected layer. This layer takes the output information from convolutional networks. Attaching a fully connected layer to the end of the network results in an N dimensional vector, where N is the amount of classes from which the model selects the desired class.

A fragment of the code of this model written in Python will be considered further in the practical part.

Process

In the beginning of this part I would like to describe the process of Supervised machine learning, which was taken as a basis of the model.

Supervised machine learning

It is one of the ways of machine learning where the model is trained by input data and expected output data.

To create such model, it is necessary to go through the following phases:

Model Construction

Model Training

Model Testing

Model Evaluation

Model construction depends on machine learning algorithms. In this projects case, it was neural networks.

Such an algorithm looks like:

1. begin with its object: model = Sequential()

2. then consist of layers with their types: model.add(type_of_layer())

3. after adding a sufficient number of layers the model is compiled. At this moment Keras communicates with TensorFlow for construction of the model. During model compilation it is important to write a loss function and an optimizer algorithm. It looks like: model.comile(loss= 'name_of_loss_function', optimizer= 'name_of_opimazer_alg') The loss function shows the accuracy of each prediction made by the model.

Before model training it is important to scale data for their further use.

After model construction it is time for model training. In this phase, the model is trained using training data and expected output for this data.

It's look this way: model.>t(training_data, expected_output).

Progress is visible on the console when the script runs. At the end it will report the final accuracy of the model.

Once the model has been trained it is possible to carry out model testing. During this phase a second set of data is loaded. This data set has never been seen by the model and therefore it's true accuracy will be veri>ed.

After the model training is complete, and it is understood that the model shows the right result, it can be saved by: model.save("name_of_>le.h5").

Finally, the saved model can be used in the real world. The name of this phase is model evaluation. This means that the model can be used to evaluate new data.

Classification Model

For my project I used only four classes i.e. fibres, biological, porous, coated .

Here I would like to describe the code that was taken as the basis of this project. It is considered that a deep learning model needs a large amount of data. But the model given in this script is excellent for training with a small amount of data. Because of that I took only available photos per class for training and divided photos of each class into 80% for training and 20% for testing .

Using little data is possible when the image is preprocessing with Keras ImageDataGenerator class. This class can create a number of random transformations, which helps to increase the number of images when it is needed.

CODE

```
#!/usr/bin/env python3
# -*- coding: utf-8 -*-
"""
Created on Wed Mar 13 13:25:55 2019

@author: darshansatone
"""

import os,cv2

import numpy as np

import matplotlib.pyplot as plt
```

```
from sklearn.utils import shuffle
from sklearn.model_selection import train_test_split


from keras import backend as K
K.set_image_dim_ordering('th')


from keras.utils import np_utils
from keras.models import Sequential
from keras.layers.core import Dense, Dropout, Activation, Flatten
from keras.layers.convolutional import Convolution2D, MaxPooling2D
from keras.optimizers import SGD,RMSprop,adam


#path of dataset
PATH = os.getcwd()
# Define data path
data_path = PATH + '/data'
data_dir_list = os.listdir(data_path)

img_rows=128
img_cols=128
```

```
num_channel=1
num_epoch=20
# Define the number of classes
num_classes = 4

img_data_list=[]

for dataset in data_dir_list:
img_list=os.listdir(data_path+'/'+ dataset)
print ('Loaded the images of dataset-'+'{}\n'.format(dataset))
for img in img_list:
input_img=cv2.imread(data_path + '/'+ dataset + '/'+ img )
input_img=cv2.cvtColor(input_img, cv2.COLOR_BGR2GRAY)
input_img_resize=cv2.resize(input_img,(128,128))
img_data_list.append(input_img_resize)


img_data = np.array(img_data_list)
img_data = img_data.astype('float32')
img_data /= 255
print (img_data.shape)


if num_channel==1:
```

```
if K.image_dim_ordering()=='th':
img_data= np.expand_dims(img_data, axis=1)
print (img_data.shape)
else:
img_data= np.expand_dims(img_data, axis=4)
print (img_data.shape)
else:
if K.image_dim_ordering()=='th':
img_data=np.rollaxis(img_data,3,1)
print (img_data.shape)
# Define the number of classes

num_classes = 4

num_of_samples = img_data.shape[0]
labels = np.ones((num_of_samples,),dtype='int64')



labels[0:202]=0
labels[202:404]=1
labels[404:606]=2
labels[606:]=3
```

```
# convert class labels to on-hot encoding

Y = np_utils.to_categorical(labels, num_classes)


#Shuffle the dataset

x,y = shuffle(img_data,Y, random_state=2)

# Split the dataset

X_train, X_test, y_train, y_test = train_test_split(x, y, test_size=0.2,
random_state=2)


# Defining the model

input_shape=img_data[0].shape


model = Sequential()


model.add(Convolution2D(32,
3,3,border_mode='same',input_shape=input_shape))

model.add(Activation('relu'))

model.add(Convolution2D(32, 3, 3))

model.add(Activation('relu'))

model.add(MaxPooling2D(pool_size=(2, 2)))

model.add(Dropout(0.5))
```

```
model.add(Convolution2D(64, 3, 3))
model.add(Activation('relu'))
#model.add(Convolution2D(64, 3, 3))
#model.add(Activation('relu'))
model.add(MaxPooling2D(pool_size=(2, 2)))
model.add(Dropout(0.5))

model.add(Flatten())
model.add(Dense(64))
model.add(Activation('relu'))
model.add(Dropout(0.5))
model.add(Dense(num_classes))
model.add(Activation('softmax')
```

Let us look at the first convolution layer Conv 2D. The number 32 shows the amount of output alter in the convolution. Numbers 3, 3 correspond to the kernel size, which determinate the width and height of the 2D convolution window. An important component of the first convolution layer is an input shape, which is the input array of pixels. Further convolution layers are constructed in the same way, but do not include the input shape.

The activation function of this model is Relu. This function setts the zero threshold and looks like: f(x) = max(0,x). If x > 0 — the volume of the array of pixels remains the same, and if x < 0 — it cuts oZ unnecessary details in the channel.

Max Pooling 2D layer is pooling operation for spatial data. Numbers 2, 2 denote the pool size, which halves the input in both spatial dimension.

After three groups of layers there are two fully connected layers. Flatten performs the input role. Next is Dense — densely connected layer with the value of the output space (64) and Relu activation function. It follows Dropout, which is preventing overfitting. Overfitting is the phenomenon when the constructed model recognises the examples from the training sample, but works relatively poorly on the examples of the test sample. Dropout takes value between 0 and 1. The last fully connected layer has 1 output and Sigmoid activation function.

```
#sgd = SGD(lr=0.01, decay=1e-6, momentum=0.9, nesterov=True)

#model.compile(loss='categorical_crossentropy',
optimizer=sgd,metrics=["accuracy"])

model.compile(loss='categorical_crossentropy',
optimizer='rmsprop',metrics=["accuracy"])
```

Next step is model compiling. It has a binary cross entropy loss function, which will show the sum of all individual losses. The optimiser algorithm is RMS drop, which is good for recurrent neural networks. The accuracy metrics shows the performance of the model.

#Viewing model_configuration

model.summary()

model.get_config()

model.layers[0].get_config()

model.layers[0].input_shape

model.layers[0].output_shape

model.layers[0].get_weights()
np.shape(model.layers[0].get_weights()[0])

model.layers[0].trainable

```
Out[21]: True

In [22]: model.summary()
    ...!

_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d_1 (Conv2D)            (None, 32, 128, 128)      320
_________________________________________________________________
activation_1 (Activation)    (None, 32, 128, 128)      0
_________________________________________________________________
conv2d_2 (Conv2D)            (None, 32, 126, 126)      9248
_________________________________________________________________
activation_2 (Activation)    (None, 32, 126, 126)      0
_________________________________________________________________
max_pooling2d_1 (MaxPooling2 (None, 32, 63, 63)        0
_________________________________________________________________
dropout_1 (Dropout)          (None, 32, 63, 63)        0
_________________________________________________________________
conv2d_3 (Conv2D)            (None, 64, 61, 61)        18496
_________________________________________________________________
activation_3 (Activation)    (None, 64, 61, 61)        0
_________________________________________________________________
max_pooling2d_2 (MaxPooling2 (None, 64, 30, 30)        0
_________________________________________________________________
dropout_2 (Dropout)          (None, 64, 30, 30)        0
_________________________________________________________________
flatten_1 (Flatten)          (None, 57600)             0
_________________________________________________________________
dense_1 (Dense)              (None, 64)                3686464
_________________________________________________________________
activation_4 (Activation)    (None, 64)                0
_________________________________________________________________
dropout_3 (Dropout)          (None, 64)                0
_________________________________________________________________
dense_2 (Dense)              (None, 4)                 260
_________________________________________________________________
activation_5 (Activation)    (None, 4)                 0
=================================================================
Total params: 3,714,788
Trainable params: 3,714,788
Non-trainable params: 0
```

Training

hist = model.fit(X_train, y_train,
batch_size=16,
nb_epoch=num_epoch,
verbose=1,
validation_data=(X_test, y_test))

#hist = model.fit(X_train, y_train, batch_size=32, nb_epoch=20,verbose=1, validation_split=0.2)

Batch size the number of training examples in one forward/backward pass (or for 1 epoch, which is expected).

Then the already described Image Data Generator is added for training and tasting samples. But it has a new transformation, which is called rescale. It multiplies the data by the given value.

The How_from_directory(directory) method is added for training and testing data. First, the path to the folders is speci>ed. Further, the target size follows. It shows width and height to which images will be resized. Next, the batch size is added. Finally binary class mode is set.

```
# Training with callbacks

from keras import callbacks

filename='model_train_new.csv'

csv_log=callbacks.CSVLogger(filename, separator=',', append=False)


early_stopping=callbacks.EarlyStopping(monitor='val_loss',  min_delta=0,
patience=0, verbose=0, mode='min')

filepath="Best-weights-my_model-{epoch:03d}-{loss:.4f}-{acc:.4f}.hdf5"


checkpoint = callbacks.ModelCheckpoint(filepath, monitor='val_loss',
verbose=1, save_best_only=True, mode='min')

callbacks_list = [csv_log,early_stopping,checkpoint]

hist = model.fit(X_train, y_train, batch_size=16, nb_epoch=num_epoch,
verbose=1, validation_data=(X_test, y_test),callbacks=callbacks_list)
```

Training is possible with the help of the @t_generator. Here it is important to indicate a number of epochs, which defines for how many times the training will repeat. 1 epoch is 1 forward pass and 1 backward pass over all the training examples.

Also, in this section steps_per_epoch and validation_steps are set. Steps_per_epoch (or number of iterations) shows total number of steps,

which is used to declare one epoch finished and begin the next. Typically this number is equal to the number of samples for training divided by the batch size (16). It means that the number of iterations: 1257/ 16 = 25. Validation_steps is total number of steps (batches of samples) to validate before stopping.

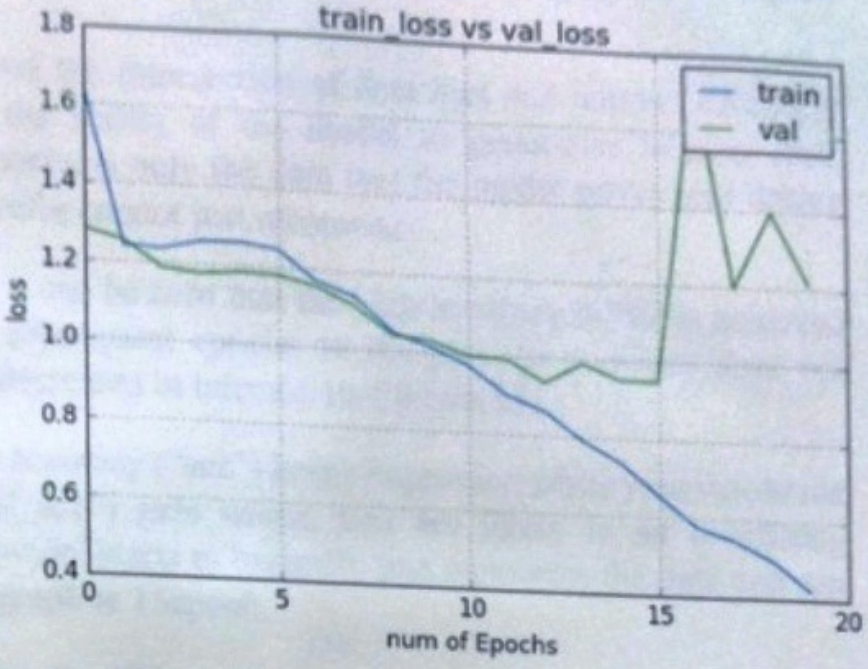
When the model is trained it should be saved with save_weights.

Best-weights-my_model-002-1.0665-0.6086.hdf5
Best-weights-my_model-001-1.0675-0.6086.hdf5
Best-weights-my_model-002-1.0885-0.6086.hdf5
Best-weights-my_model-001-1.0664-0.6086.hdf5
Best-weights-my_model-001-1.0791-0.6086.hdf5
Best-weights-my_model-001-1.0937-0.6086.hdf5

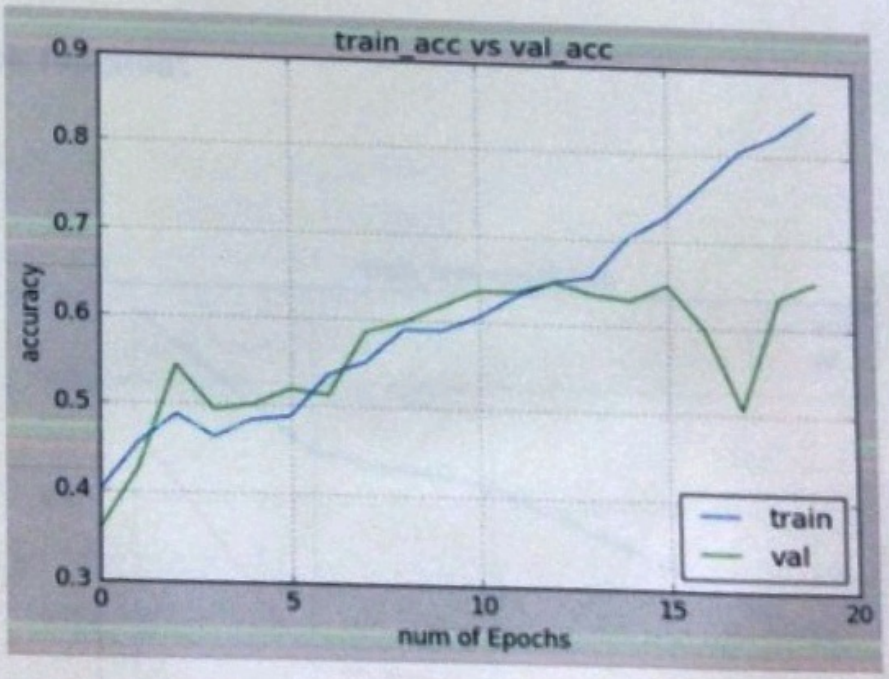
```
# visualising losses and accuracy

train_loss=hist.history['loss']

val_loss=hist.history['val_loss']

train_acc=hist.history['acc']

val_acc=hist.history['val_acc']

xc=range(3)


plt.figure(1,figsize=(7,5))

plt.plot(xc,train_loss)

plt.plot(xc,val_loss)

plt.xlabel('num of Epochs')

plt.ylabel('loss')

plt.title('train_loss vs val_loss')

plt.grid(True)
```

```
plt.legend(['train','val'])
#print plt.style.available # use bmh, classic,ggplot for big pictures
plt.style.use(['classic'])

plt.figure(2,figsize=(7,5))
plt.plot(xc,train_acc)
plt.plot(xc,val_acc)
plt.xlabel('num of Epochs')
plt.ylabel('accuracy')
plt.title('train_acc vs val_acc')
plt.grid(True)
plt.legend(['train','val'],loc=4)
#print plt.style.available # use bmh, classic,ggplot for big pictures
plt.style.use(['classic'])
```

Results



As a result of testing the model, I got good accuracy: 67% of correct classification samples after 20 epochs. The only drawback was that I had to wait about 50 minutes until 20 epochs come to the end (looking at the fact that I had a very small number of photos for training and at low system specifications).



Since we altered the parameters in our convolutional layer by reducing the size to its half there is always a chance of data loss.

Over 20 epochs the data loss was found optimum for both training and validating session.

For this, I decided to build two plots. The first shows the dependence of the data loss on the number of epochs. The data loss was calculated using

additional dataset of 1543 pictures. The second plot shows the dependence of accuracy and validation accuracy on the number of epochs during the testing.

The first graph shows the intersection of data loss and number of epoch. Data loss shows the ability of the model to generalise to new data. Validation dataset contains only the data that the model never sees during the training and therefor cannot just memorise.

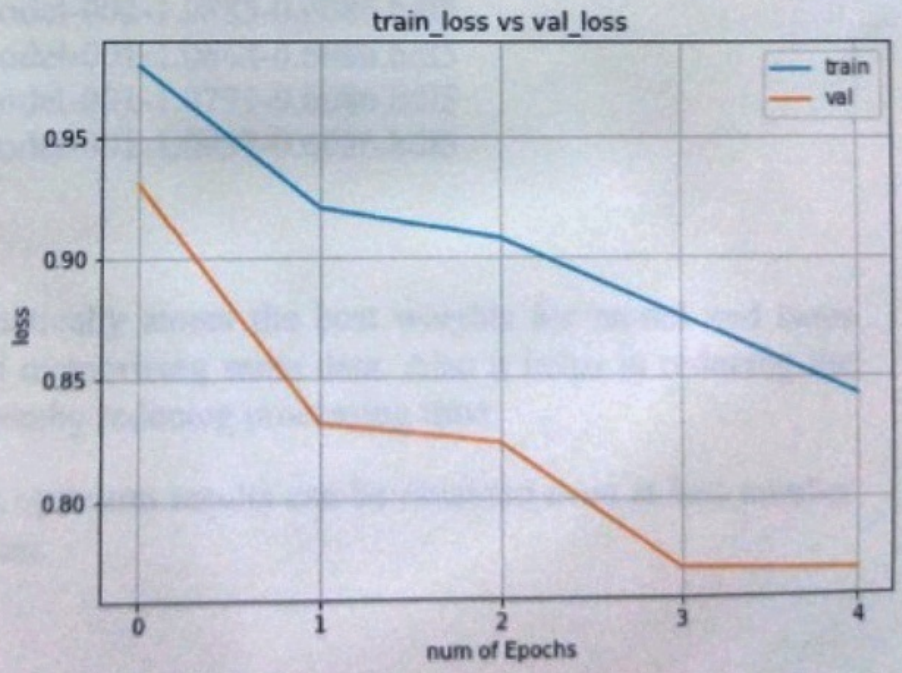
On the Second plot it can be seen that the high accuracy (67%) is achieved after 10 epoch. In subsequent epochs on the plot the accuracy does not improve (and even decreases in interval 10–20 epochs).

If your training data accuracy ("acc") keeps improving while your validation data accuracy ("val_acc") gets worse, you are likely in an overfitting situation, i.e. your model starts to basically just memorise the data and can be seen in our first graph at 15epoch.

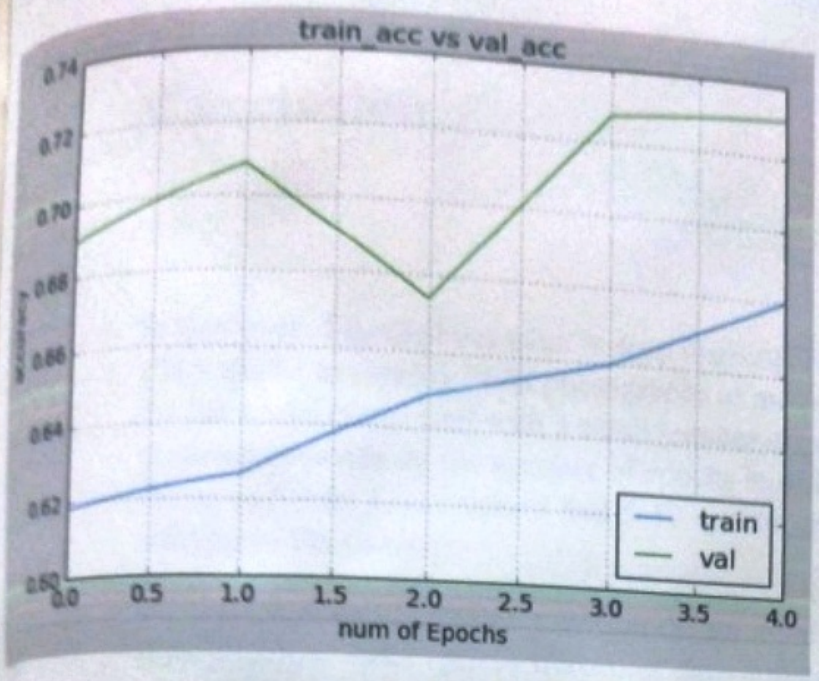
This means that after the 10th epoch the model can show the same result, but it will not be better. Consequently, this model is be sufficient to train on 10 epochs.

Using Keras callback function:

By using Keras callback function problem of overfitting and data loss can be overcome easily.



In this graph we can see the loss of data went on decreasing and model stopped at 5epochs.



The data loss is less while validating model than training session .

In this graph we can see accuracy is increased to 73% . The validation accuracy gave good results than training session.

Keras callback function stores best weights and optimise neural network in real time.

Best-weights-my_model-002-1.0665-0.6086.hdf5
Best-weights-my_model-001-1.0675-0.6086.hdf5
Best-weights-my_model-002-1.0885-0.6086.hdf5
Best-weights-my_model-001-1.0664-0.6086.hdf5
Best-weights-my_model-001-1.0791-0.6086.hdf5
Best-weights-my_model-001-1.0937-0.6086.hdf5

This function automatically stores the best weights for model and stops neural network from memorising same data. Also it helps in reducing the number of epochs thereby reducing processing time.

Using this technique optimum results can be obtained even at less number of epochs and data loss.

Conclusion

In this work, I figured out what is deep learning. I assembled and trained the CNN model to classify SEM photographs of materials. I have tested that this model works really well with a small number of photos. I measured how the accuracy depends on the number of epochs in order to detect potential over fitting problem. I determined that 5-10 epochs are enough for a successful training of the model.

My next step would be to try this model on more data sets and try to apply it to practical tasks (phase transition determination). I would also like to experiment with the neural network design in order to see how a higher efficiency can be achieved in various problems.

References

https://www.geeksforgeeks.org

https://towardsdatascience.com/the-7-steps-of-machine-learning-2877d7e5548e

http://www.datamind.cz/cz/vam-na-miru/umela-inteligence-a-strojove-uceni-ai-machine-learning

https://en.wikipedia.org/wiki/Artificial_neural_network

https://en.wikipedia.org/wiki/Deep_learning

https://en.wikipedia.org/wiki/Convolutional_neural_network

https://adeshpande3.github.io/adeshpande3.github.io/A-Beginner%27s-Guide-To-Understanding-Convolutional-Neural- Networks/

https://www.lynda.com/Google-TensorFlow-tutorials/Building-Deep-Learning-Applications-Keras-2-0/601801-2.html

https://blog.keras.io/building-powerful-image-classi>cation-models-using-very-little-data.html

NFFA-EUROPE - SEM Dataset

by Aversa, Rossella; Modarres, Mohammad Hadi; Cozzini, Stefano; Ciancio, Regina;

Feb 19, 2018

Last updated at Apr 16, 2018

https://keras.io