***Diabetic Rethinopathy***

**Table of Contents:**

|  |  |  |
| --- | --- | --- |
| S. No | Topic | Page Number |
| 1 | Summary | 3 |
| 2 | Introduction | 3 |
| 3 | Objective | 3 |
| 4 | Dataset | 3 |
| 5 | Key Topics | 4 |
| 5 | Source Code | 24 |
| 6 | Conclusion | 34 |
| 7 | References | 34 |
| 8 | Acknowledgement | 34 |

**Summary:**

At the end of the Diabetic Rethinopathy project. We will observe how to conduct data visualization. We get hands-on experience of various packages in Python language. Hands on with different algorithms to understand the data and work on real world projects.

**Introduction**

Diabetic retinopathy is an eye condition that can cause vision loss and blindness in people who have diabetes. It affects blood vessels in the retina (the light-sensitive layer of tissue in the back of your eye).

The early stages of diabetic retinopathy usually don’t have any symptoms. Some people notice changes in their vision, like trouble reading or seeing faraway objects. These changes may come and go.

In later stages of the disease, blood vessels in the retina start to bleed into the vitreous (gel-like fluid that fills your eye). If this happens, you may see dark, floating spots or streaks that look like cobwebs. Sometimes, the spots clear up on their own — but it’s important to get treatment right away. Without treatment, scars can form in the back of the eye. Blood vessels may also start to bleed again, or the bleeding may get worse.

Anyone with any kind of diabetes can get diabetic retinopathy including people with type 1, type 2, and gestational diabetes (a type of diabetes that can develop during pregnancy).

Your risk increases the longer you have diabetes. Over time, more than half of people with diabetes will develop diabetic retinopathy.

**Objective**

Use of CNN in image classification. Its built-in convolutional layer reduces the high dimensionality of images without losing its information. That is why CNNs are especially suited for this use case.

**Dataset:**

The dataset has been taken from Kaggle

***Key Topics of the Project:***

***Batch Normalization:***

**What is Batch Normalization?**

Batch Normalization is a [supervised learning](https://deepai.org/machine-learning-glossary-and-terms/supervised-learning) technique that converts interlayer outputs into of a [neural network](https://deepai.org/machine-learning-glossary-and-terms/neural-network) into a standard format, called normalizing.  This effectively 'resets' the distribution of the output of the previous layer to be more efficiently processed by the subsequent layer.

## 2.Normalization

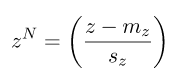
To fully understand how Batch Norm works and why it is important, let’s start by talking about normalization.

**Normalization is a pre-processing technique used to standardize data**. In other words, having different sources of data inside the same range. Not normalizing the data before training can cause problems in our network, making it drastically harder to train and decrease its learning speed.

## 3. Batch Normalization

**Batch Norm is a normalization technique done between the layers of a Neural Network instead of in the raw data**. It is done along mini-batches instead of the full data set. It serves to speed up training and use higher learning rates, making learning easier.

Following the technique explained in the previous section, we can define the normalization formula of Batch Norm as:



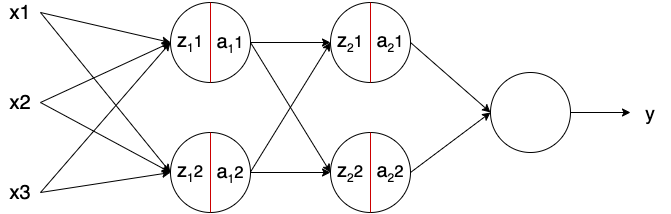
being  the mean of the neurons’ output and  the standard deviation of the neurons’ output.

# **What are the advantages of Batch Normalisation?**

* The model is less delicate to **hyperparameter tuning**. That is, though bigger **learning rates** prompted non-valuable models already, bigger LRs are satisfactory at this point
* Shrinks **internal covariant** shift
* Diminishes the reliance of **gradients** on the scale of the parameters or their underlying values
* **Weight initialization** is a smidgen less significant at this point
* **Dropout** can be evacuated for regularisation

### 3.1. How Is It Applied?

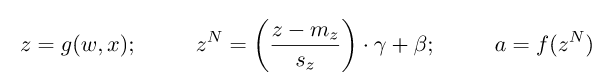
In the following image, we can see a regular feed-forward Neural Network:  are the inputs,  the output of the neurons,  the output of the activation functions, and  the output of the network:

[](https://www.baeldung.com/wp-content/uploads/sites/4/2020/10/neural-network.png)

Batch Norm – in the image represented with a red line – is applied to the neurons’ output just before applying the activation function. Usually, a neuron without Batch Norm would be computed as follows:



Being   the linear transformation of the neuron,   the weights of the neuron,   the bias of the neurons, and   the activation function. The model learns the parameters  and . Adding Batch Norm, it looks as:



Being   the output of Batch Norm,   the mean of the neurons’ output ,  the standard deviation of the output of the neurons, and  and  learning parameters of Batch Norm. Note that the bias of the neurons (b) is removed. This is because as we subtract the mean  , any constant over the values of z – such as b – can be ignored as it will be subtracted by itself.

**The parameters** **and  shift the mean and standard deviation, respectively**. Thus, the outputs of Batch Norm over a layer results in a distribution with a mean  and a standard deviation of . These values are learned over epochs and the other learning parameters, such as the weights of the neurons, aiming to decrease the loss of the model.

### 3.2. Implementation in Python

Implementing Batch Norm is quite straightforward when using modern Machine Learning frameworks such as Keras, Tensorflow, or Pytorch. They come with the most commonly used methods and are generally up to date with state of the art.

With Keras, we can implement a really simple feed-forward Neural Network with Batch Norm as:

**from** tensorflow.keras.models **import** Sequential

**from** tensorflow.keras.layers **import** Dense, BatchNormalization

model = Sequential([

Dense(16, input\_shape=(1,5), activation='relu'),

BatchNormalization(),

Dense(32, activation='relu'),

BatchNormalization(),

Dense(2, activation='softmax')

])

### 3.3. Why Does Batch Normalization Work?

Now that we know how to apply and implement Batch Norm, why does it work? How can it speed up training and make learning easier?

There are different reasons why it is believed that Batch Norm affects all of that. Here we will expose the intuitions of the most important reasons.

Firstly, we can see how normalizing the inputs to take on a similar range of values can speed up learning. One simple intuition is that Batch Norm is doing a similar thing with the values in the layers of the network, not only in the inputs.

Secondly, in their original paper [Sergey et al.](https://arxiv.org/pdf/1502.03167.pdf) claim that **Batch Norm reduces the internal covariate shift of the network**. The covariate shift is a change in data distribution. For example, going back to our example on the car rental service, imagine we want to include other motorbikes. If we only look at our previous data set, containing only cars, our model will likely fail to predict motorbikes’ price. This change in data (now containing motorbikes) is named covariate shift, and it is gaining attention as it is a common issue in real-world problems.

The internal covariate shift is a change in the input distribution of an internal layer of a Neural Network. For the neurons in an internal layer, the inputs received (from the previous layer) are constantly changing. This is due to the multiple computations done before it and the weights over the training process.

Applying Batch Norm ensures that the mean and standard deviation of the layer inputs will always remain the same;  and , respectively. Thus, the amount of change in the distribution of the input of layers is reduced. The deeper layers have a more robust ground on what the input values are going to be, which helps during the learning process.

Lastly, it seems that **Batch Norm has a regularization effect**.  Because it is computed over mini-batches and not the entire data set, the model’s data distribution sees each time has some noise. This can act as a regularizer, which can help overcome overfitting and help learn better. However, the noise added is quite small. Thus, it generally is not enough to properly regularize on its own and is normally used along with Dropout.

## 4. Batch Normalization in Convolutional Neural Networks

Batch Norm works in a very similar way in Convolutional Neural Networks. Although we could do it in the same way as before, we have to follow the convolutional property.

In convolutions, we have shared filters that go along the feature maps of the input (in images, the feature map is generally the height and width). These filters are the same on every feature map. It is then reasonable to normalize the output, in the same way, sharing it over the feature maps.

In other words, this means that the parameters used to normalize are calculated along with each entire feature map. In a regular Batch Norm, each feature would have a different mean and standard deviation. Here,**each feature map will have a single mean and standard deviation**, used on all the features it contains.

### 4.1. Implementation in Python

Again, implementing Batch Norm in Convolutional Neural Networks is really easy using modern frameworks. It does the operation in the background, using the same function we used before:

**from** tensorflow.keras.models **import** Sequential

**from** tensorflow.keras.layers **import** Dense, BatchNormalization, Conv2D, MaxPooling2D

model = Sequential([

Conv2D(32, (3,3), input\_shape=(28, 28, 3) activation='relu'),

BatchNormalization(),

Conv2D(32, (3,3), activation='relu'),

BatchNormalization(),

MaxPooling2D(),

Dense(2, activation='softmax')

])

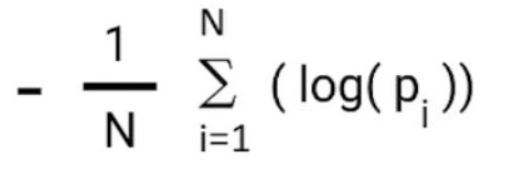
***Entropy:***

In physics, entropy is measurement of randomness in an isolated system. It's quite similar when it comes to machine learning. Here, Entropy is also a measure of randomness. Here, you measure the disorder of the information processed in your ML project.

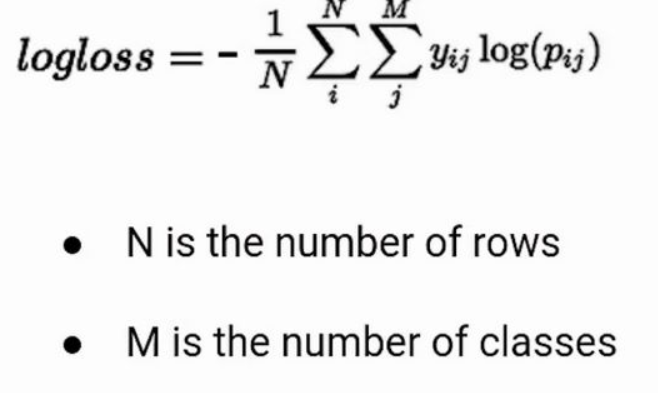
Binary cross entropy:

Binary Cross Entropy is the negative average of the log of corrected predicted probabilities. Binary Classification is a problem where we have to segregate our observations in any of the two labels on the basis of the features. Suppose you have some images now you have to put each of them in a stack one for Dogs and the other for the Cats.If a machine learning model is identifying mails as ham or spam, it is performing binary classification as it is dividing the objects into two classes.

Binary cross entropy:

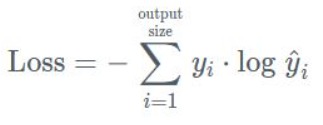


For multi-class classification:



Categorical cross entropy:

Categorical cross entropy is a loss function that is used in multi-class classification tasks. These are tasks where an example can only belong to one out of many possible categories, and the model must decide which one. The categorical cross entropy loss function calculates the loss of an example by computing the following sum:



***Regularization:***

# Machine Learning Regularization uses Ridge regression, which is a model tuning method used for analyzing data with multicollinearity. In Lasso regression, the model is penalized by the sum of absolute values of the weights, whereas in Ridge regression the model is penalized for the sum of squared values of the weights of coefficient. The least-squares are unbiased when there are multicollinearity issues and hence improve the accuracy of the prediction. Ridge regression is a method of estimating co-efficient where the linearly independent variables are highly correlated. While both Ridge and Lasso are variations of linear regression, Bias and Variance trade-off plays a major role in Ridge regression.

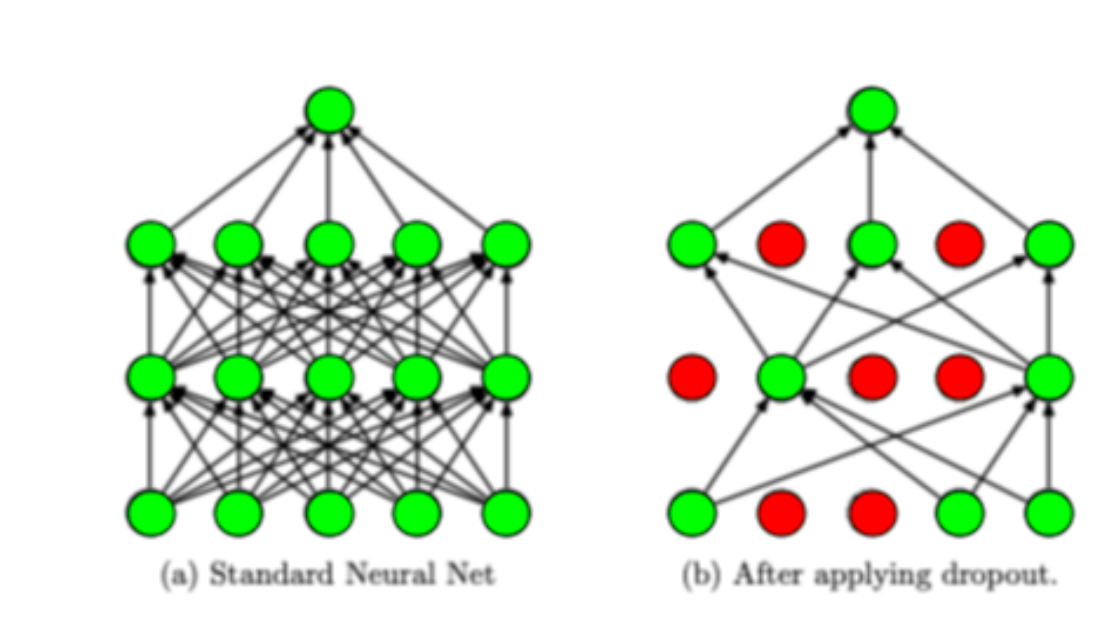
If you have been working on business case problems using Machine or Deep Learning techniques or participating in some of the data science hackathons, you would certainly have faced the situation where your training accuracy is very high, and the testing accuracy is not.

The reason this happens is because of the inevitable trade-off between bias and variance error. If you want to make functional machine learning or deep learning models, you cannot escape this trade-off. Therefore, to understand what regularization is in machine learning and why it is needed, it is crucial to understand the concept of the bias-variance trade-off and its impact on the model

**Dropout Machine Learning Regularization**

one of the most commonly used techniques for **Deep Learning Systems**. Deep Neural Nets are powerful Machine Learning Systems. And overfitting could be a serious problem to counter in these large Neural Nets.

Dropout is a **Machine Learning Regularization** technique that approximates training a large number of neural networks with different architectures in parallel. It is achieved by blocking or dropping randomly selected neurons during training.



Dropout can be easily implemented in input as well as hidden data. In this regularization technique, the neurons are randomly omitted, and the existing neurons on different levels lead to compensate for reduced capacity for the prediction. This forces the network to learn complex internal representation. The network becomes insensitive to certain neurons and makes better generalizations for the overall training data.

The main advantage of the Dropout technique is that it prevents all the neurons in the network from converging towards the same goal and working synchronously. With the Dropout technique, you can de-correlate the weights and make the **Deep Learning Model** perform better generalization tasks and Predictions.

Artificial intelligence and machine learning jobs have jumped almost by 75% over the past four years. With the number of funds pouring in for research and development in this field, it is expected to grow at an even faster pace.

If you feel this is a challenge for you and like to learn more about machine learning, consider a Master’s Program in Artificial Intelligence and Machine Learning. We are a globally recognized ATO [An accredited training organization] called Sprintzeal.

#### How will dropout help with overfitting?

Dropout regularization will ensure the following:

* The neurons can’t rely on one input because it might be dropped out at random. This reduces bias due to over-relying on one input, bias is a major cause of overfitting.
* Neurons will not learn redundant details of inputs. This ensures only important information is stored by the neurons. This enables the neural network to gain useful knowledge which it uses to make predictions.

From our program, you can learn the latest AI technologies like Machine Learning, Deep Learning, Speech Recognition, Language Processing, and much more. Our program will equip you with all the necessary knowledge and resources to take on the competitive world of machine learning and will ensure your success in the field.

***Adam Optimizer***

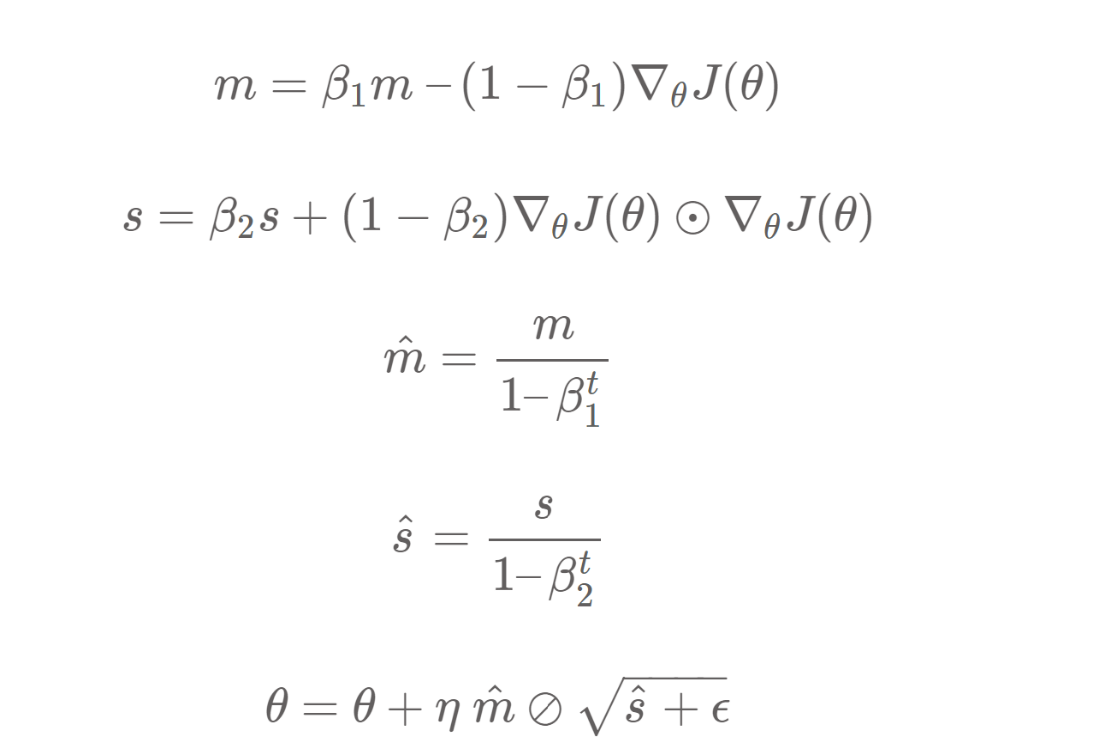
Adam, derived from Adaptive Moment Estimation, is an optimization algorithm.

The Adam optimizer makes use of a combination of ideas from other optimizers.

Similar to the momentum optimizer, Adam makes use of an exponentially decaying average of past gradients. Thus, the direction of parameter updates is calculated in a manner similar to that of the momentum optimizer.

Adam also employs an exponentially decaying average of past squared gradients in order to provide an adaptive learning rate. Thus, the scale of the learning rate for each dimension is calculated in a manner similar to that of the RMSProp optimizer.

The similarity of the Adam optimizer to the momentum and RMSProp optimizers is immediately clear upon examining the equations defining the Adam optimizer.



Arguments

* learning\_rate: A Tensor, floating point value, or a schedule that is a callable that takes no arguments and returns the actual value to use, The learning rate. Defaults to 0.001.
* beta\_1: A float value or a constant float tensor, or a callable that takes no arguments and returns the actual value to use. The exponential decay rate for the 1st moment estimates. Defaults to 0.9.
* beta\_2: A float value or a constant float tensor, or a callable that takes no arguments and returns the actual value to use, The exponential decay rate for the 2nd moment estimates. Defaults to 0.999.
* epsilon: A small constant for numerical stability. This epsilon is "epsilon hat" in the Kingma and Ba paper (in the formula just before Section 2.1), not the epsilon in Algorithm 1 of the paper. Defaults to 1e-7.
* amsgrad: Boolean. Whether to apply AMSGrad variant of this algorithm from the paper "On the Convergence of Adam and beyond". Defaults to False.
* name: Optional name for the operations created when applying gradients. Defaults to "Adam".
* \*\*kwargs: keyword arguments. Allowed arguments are clipvalue, clipnorm, global\_clipnorm. If clipvalue (float) is set, the gradient of each weight is clipped to be no higher than this value. If clipnorm (float) is set, the gradient of each weight is individually clipped so that its norm is no higher than this value. If global\_clipnorm (float) is set the gradient of all weights is clipped so that their global norm is no higher than this value.

***Convolution Neural network:***

It is form of neural network built on top of the Ann. In this neural network before going to distribute pixels to neurons in dense network we apply filters to images which give n\*n pixels of features.

There are two types of filters

1.horizontal filter

2.vertical filter

**Horizontal filters** which give n\*n pixels of features in horizontal direction

**Vertical filters** which give n\*n pixel of features in vertical direction

***Convolution:***

When applying filters sum of pixels take for every filter which is obtained as one pixel value in resultant pixels these process continues for every filter in n\*n pixels.

Ex: If there are n\*n pixels after applying filters of r\*r which gives (n-r+1)\*(n-r+1)\*3 pixels of features where each pixel in (n-r+1)\*(n-r+1) is sum of pixels in n\*n pixels

Here 3 means r,g,b values

***Stride:***

It depends of how we apply filters to pixels. When we want to apply filter with one pixel gap then

stride is 1

When we want to apply filters with two pixel gaps then

stride is 2

If we apply filters of r\*r\*3 to n\*n\*3 pixels with stride s

Then resultant is (n-r+1/s)\*n-r+1/s)\*3

***Padding:***

When applying filters to pixels, some pixels got more filters compare to some.

These resultant in imbalance of features in resultant pixels after applying filters.

To avoid these padding is best solution By applying padding to filters all pixels got balance apply of filters.

It is represented by 2p where p=1,2,3….

If we apply r\*r filters to n\*n pixels with stride s and padding is 2p then (n+2p-r+1/s)\*(n+2p-r+1/s) pixels is the

resultant.

***Maxpooling:***

When applying filters to n\*n pixels then there is a max pixel value taken for each filter in n\*n pixels which give pixel with Max pixel value

Actually maxpooling will apply after convolution whichgives large pixel value in resultant .By applying maxpooling We got features with high pixel values which is more helpful in easily detection of any picture after training.

After applying convolution- maxpooling number of times Pixels of n\*n\*3 convert into one dimensional pixels before going to dense network.

Why do we need cnn:

Generally when applying pixels to neurons in Ann it require large number of neurons which makes our model more complex.

But due to cnn we get pixels with better features by applying convolution-cnn before going into dense network which makes our model simple in easily classify

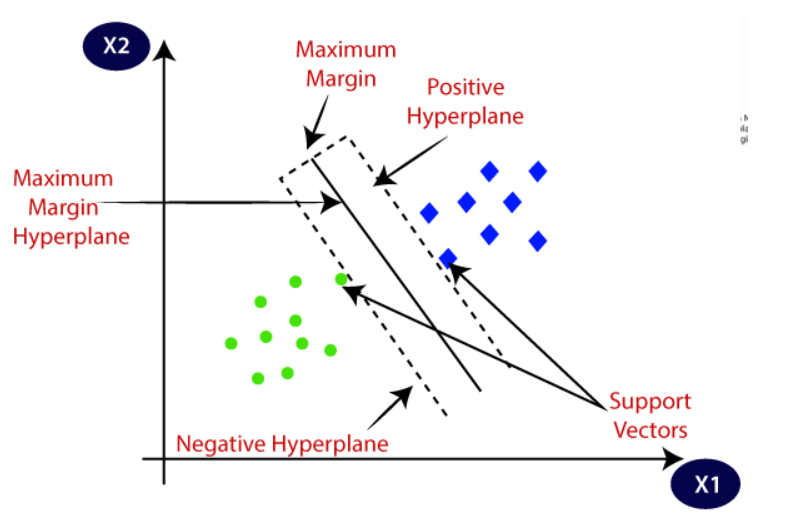
of features to neurons and in case of this the resultant pixels are also less before going into dense network which is not in case of Ann.

***Support Vector Machine Algorithm***

Support Vector Machine or SVM is one of the most popular Supervised Learning algorithms, which is used for Classification as well as Regression problems. However, primarily, it is used for Classification problems in Machine Learning.

The goal of the SVM algorithm is to create the best line or decision boundary that can segregate n-dimensional space into classes so that we can easily put the new data point in the correct category in the future. This best decision boundary is called a hyperplane.

SVM chooses the extreme points/vectors that help in creating the hyperplane. These extreme cases are called as support vectors, and hence algorithm is termed as Support Vector Machine. Consider the below diagram in which there are two different categories that are classified using a decision boundary or hyperplane:



## Types of SVM

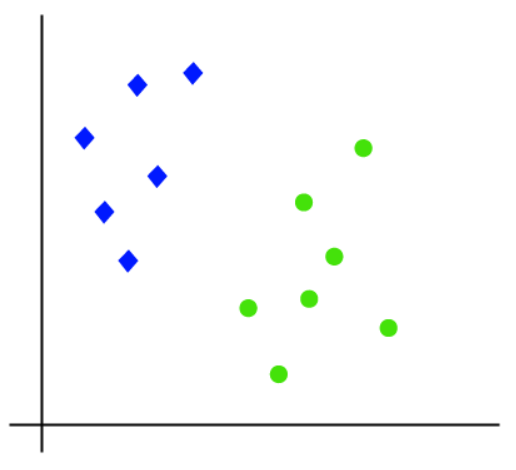
**SVM can be of two types:**

* **Linear SVM:** Linear SVM is used for linearly separable data, which means if a dataset can be classified into two classes by using a single straight line, then such data is termed as linearly separable data, and classifier is used called as Linear SVM classifier.
* **Non-linear SVM:** Non-Linear SVM is used for non-linearly separated data, which means if a dataset cannot be classified by using a straight line, then such data is termed as non-linear data and classifier used is called as Non-linear SVM classifier.

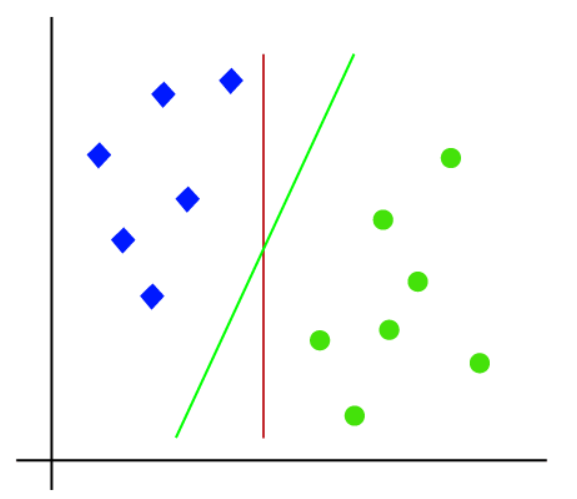
## How does SVM works?

**Linear SVM:**

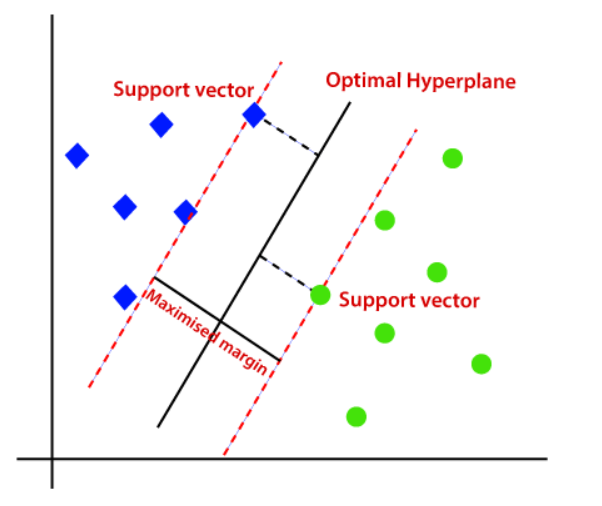
The working of the SVM algorithm can be understood by using an example. Suppose we have a dataset that has two tags (green and blue), and the dataset has two features x1 and x2. We want a classifier that can classify the pair(x1, x2) of coordinates in either green or blue. Consider the below image:



So as it is 2-d space so by just using a straight line, we can easily separate these two classes. But there can be multiple lines that can separate these classes. Consider the below image:



Hence, the SVM algorithm helps to find the best line or decision boundary; this best boundary or region is called as a **hyperplane**. SVM algorithm finds the closest point of the lines from both the classes. These points are called support vectors. The distance between the vectors and the hyperplane is called as **margin**. And the goal of SVM is to maximize this margin. The **hyperplane** with maximum margin is called the **optimal hyperplane**.



***Decision Tree***

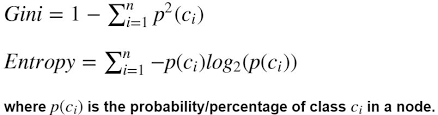
**How does the Decision Tree work?**

As we can get some idea from the name itself, it makes a tree for making a decision. The decision tree uses features for splitting data at each node. There can be a number of options to select from features, so for choosing the best-split Decision tree calculates Gini impurity or entropy.

**Gini Impurity and Entropy**

In the Decision tree, we split the dataset at a node based on a feature. There can be many choices in selecting a feature for a node to split data based on the features a dataset has. To calculate the purity of split on a specific node we use methods such as Gini impurity and entropy.

It is very easy to calculate Gini impurity. It is the summation of the square of the ratio of each class count in that node to total instances in that node and then subtracting by 1.

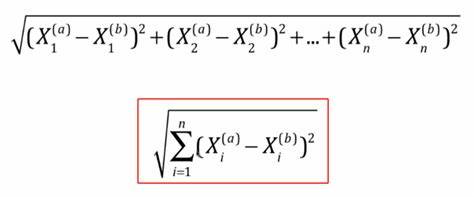


**K- NEAREST NEIGHBOUR CLASSIFICATION**

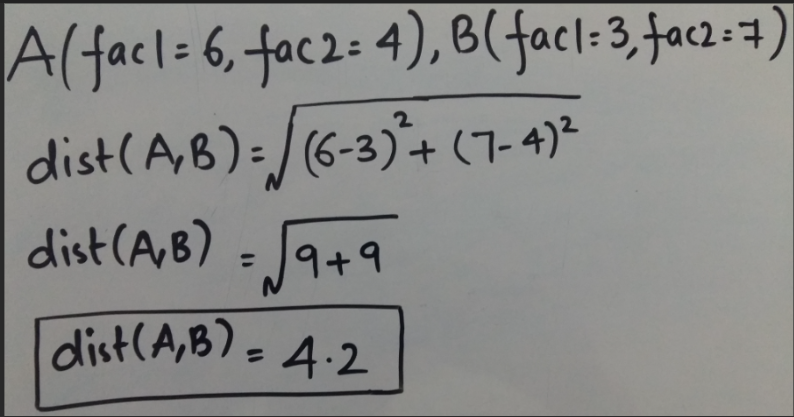
An example of a supervised machine learning (ML) technique that may be applied to both classification and regression issues is the K-nearest (KNN) approach. But in the industry, classification issues are where it is most frequently used. With k-NN, all computation is postponed until after the function has been evaluated and the function is only locally approximated. Since this technique relies on distance for classification, normalizing the training data can significantly increase accuracy if the features reflect several physical units or have distinct sizes.

So how is KNN implemented? Let’s go over how it is now:

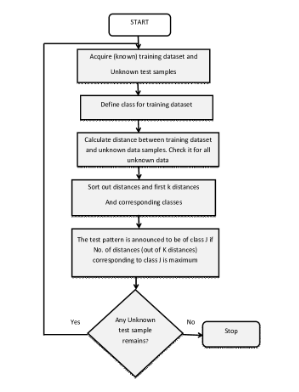
1. Set an initial value for K, which will represent all of the nearby data points to which we will attempt to compare the new prediction. One thing to be aware of is how the value of K is chosen. There isn't a one-size-fits-all solution here, but 5 is typically the conventional and ideal value. One or two could be too low and not adequately capture the context of a novel or unknown data point, or they could be too high and point to an outlier or an overly specific data point, which could result in an overly generic model.
2. The distance between points p and q, which is also the distance from q to p, is shown by this equation.



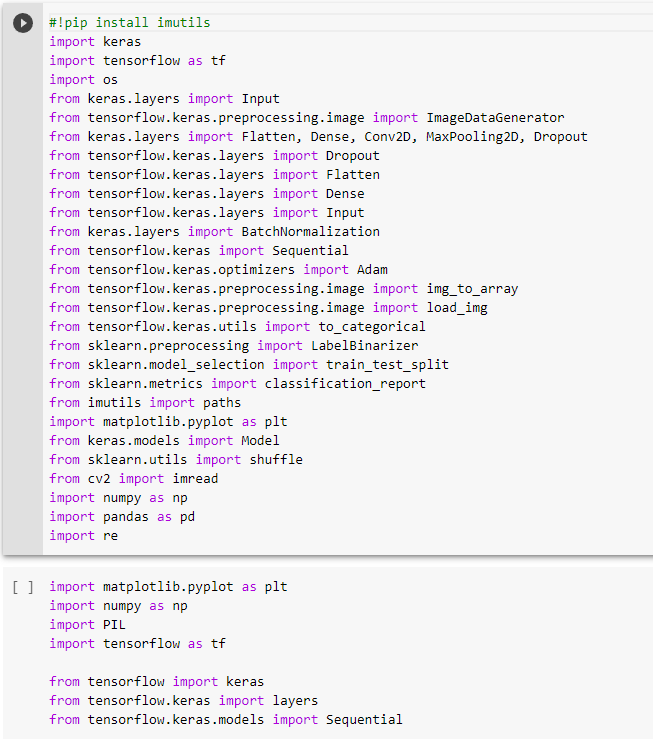
In order to calculate the distance, we multiply the size of each coordinate by one another, add them all up, use the power of 2, and then square the result. Here's an illustration: Finding the distance between points (3,4) and (9) is our goal (11, 7). Notably, you will typically find the separation between multiple data points with significantly more than two. However, let's go on. The formula need to appear something like this:



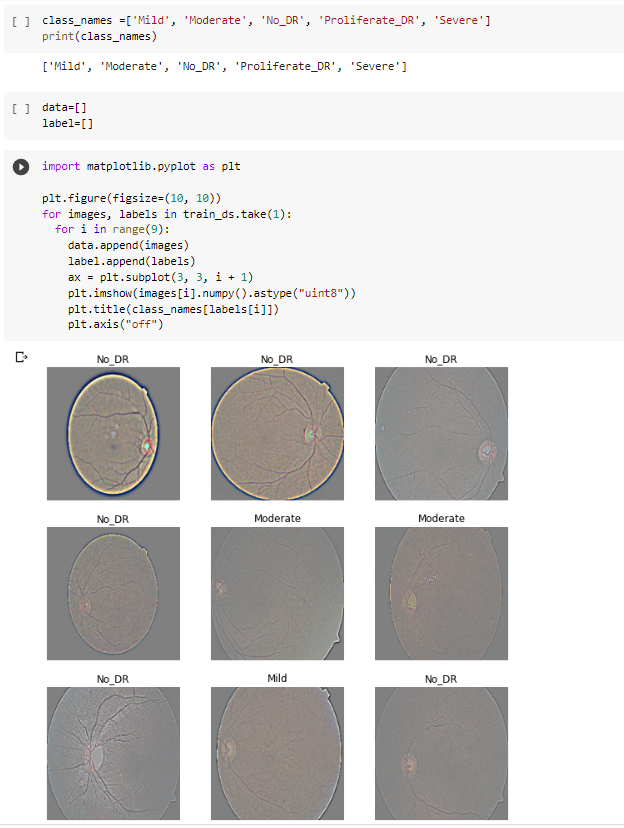
1. Select the nearest K-amount of observations in the training data using the aforementioned Euclidean distance.
2. Using the most well-liked response as the next data point's prediction. Now, in order to determine which K is appropriate for the dataset, we repeatedly run the KNN algorithm using different values. For instance, we might run the method four times on a range between 3 and 6. In order to prevent a tie and achieve a tiebreaker, it is a good idea to try to choose an odd value for K.



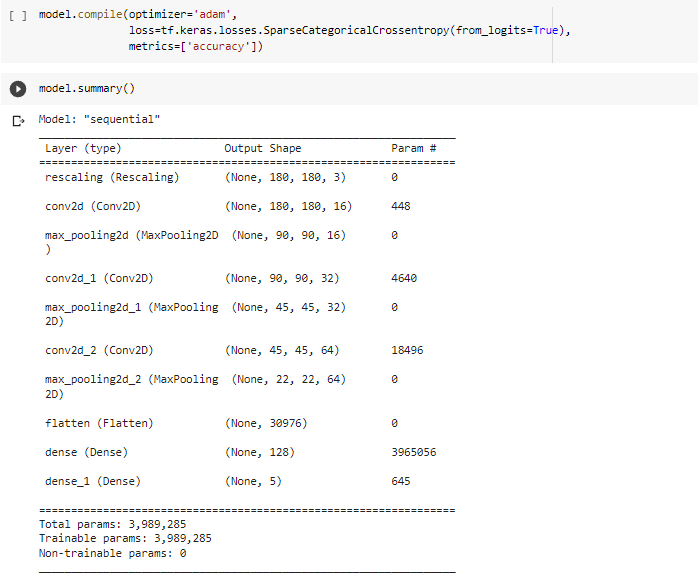
***Source Code & Output:***

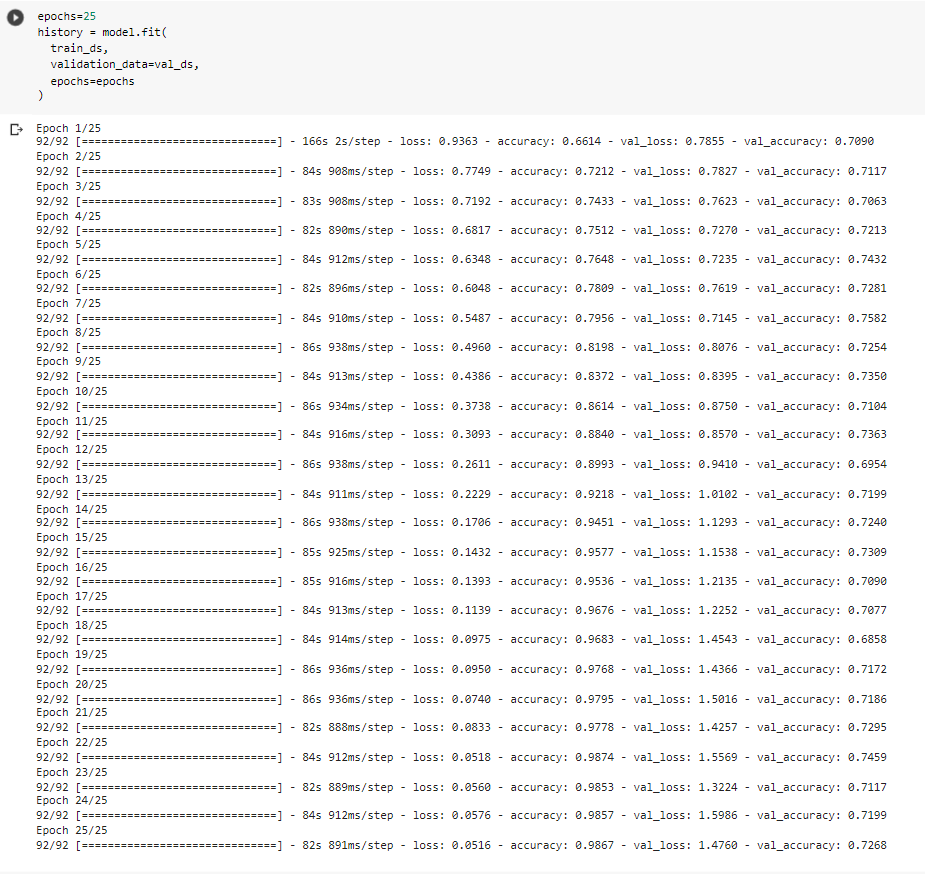
****

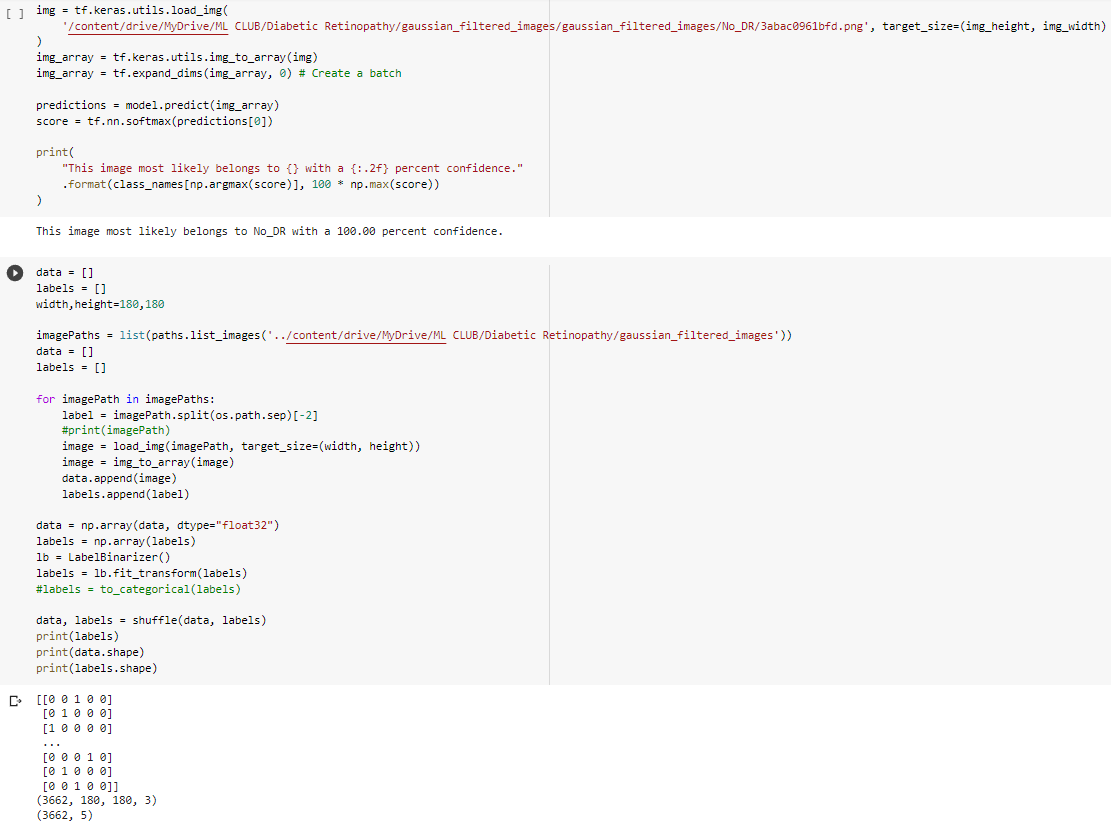
****

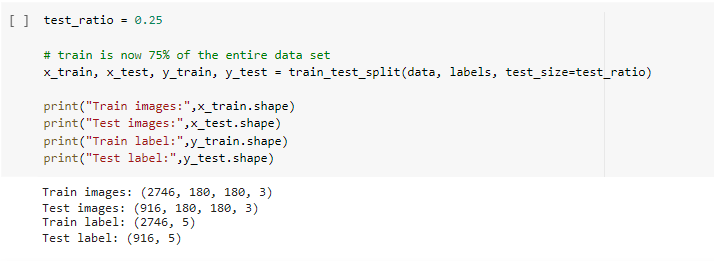
****

****

****

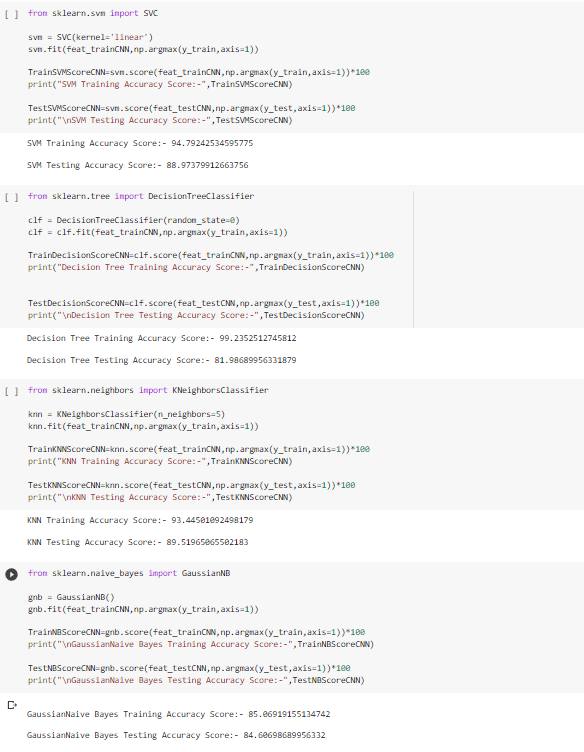
****

****

****

****

****

****

****

***Conclusion***

So, we can conclude by saying that this project “Diabetic Retinopathy” is extremely useful for detecting diabetic retinopathy at a very early age. These findings suggest that a machine learning process(use of CNN in image classification) is a reliable method for detection of diabetic retinopathy outside of clinical settings. Final results are statistically limited and will benefit from future clinical studies to extend the sample size. The conclusion which we found is that who are likely to have diabetic retinopathy by comparing various aspects.

***References:***

<https://www.geeksforgeeks.org/convolutional-neuralnetwork-cnn-in-machine-learning/>

<https://www.simplilearn.com/tutorials/deep-learningtutorial/convolutional-neural-network>