Performance Analysis of Machine Learning Techniques for Smart Agriculture: Comparison of Supervised Classification Approaches

Rhafal Mouhssine[1], Abdoun Otman[2], El khatir Haimoudi[3]
Computer Science Department, Laboratory of Advanced
Science and Technologies
Polydisciplinary faculty, University UAE, Larache, Morocoo

***Abstract*—Agriculture form one of the most important aspects of life necessities, it is responsible to feed 7.7 billion person for the time being, and it is expected to supply more than 9.6 billion individual in 2050, the thing that made classical farming insufficient, and give birth to the notion of smart farming, and the race has begun toward using the latest technologies in the field. They integrate the Internet of Things (IoT), automation, Artificial Intelligence (AI), etc. And as researchers from a country that highly depends on agriculture, we have decided to also contribute to this evolution, and we chose Machine learning (ML) as our entrance to the field to satisfy the need for automated classification of the different products produced by a farm. In this work, we wanted to solve the problem of automatic classification of agricultural products, without the need of any human intervention, and we concentrate on the classification of red fruits, due to our proximity to a location that its product is red fruits. In other words, we are doing a comparative study among the well-known approaches that are used in image classification, and we are applying the best-found method to correctly classify the pictures of red fruits. And this empirically leads us to achieve great results as shown in the numerical result area.**

Keywords*—Support vector machine; K-nearest neighbor; deep neural networks; convolutional neural networks; smart agriculture; Cifar10

I. Introduction

The agriculture plays an important role in the economic systems of several countries, and one of these is our country, the Kingdom of Morocco, the agriculture forms one of the most important incomes to the country. Thus, increasing the effectiveness of the farming would also affect positively the economy of the kingdom, and develop somethings means to integrate the latest existing technologies in the field. And After the last revolution of the AI appears a term called smart farming, which directly affect the field of agriculture. But this short term assembles many intelligent technologies, and some of them already in use. But in this work, we chose to enter this world by using computer vision and image classification and use it to automatically classify the different species by the means of images.

Image classification is the ability to choose a unique correct label to the input image from a predefined set of categories, and it's considered one of the core problems in computer vision which resides in the intersection of several fields of studies: Mathematics, image processing, data mining, etc. Image classification has a large variety of applications such as object detection, segmentation, facial recognition, etc. and those applications can be used in larger practical applications like Surveillance Autonomous vehicles. Its complexity and its effectiveness highly depend on the method used to solve the problem since there are several methods that can be used to solve that problem (image classification).

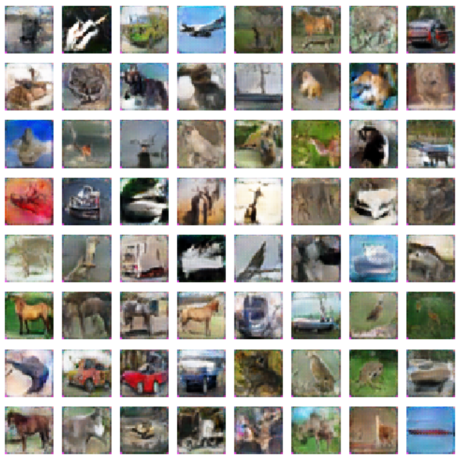


Fig. 1. Cifar10 dataset.

There have been several attempts to automate the process of image classification, but have chosen the closest papers to our work. In this area, Y. Abouelnaga and al tried on their work to work on an assembled model that use several CNN models and combine it with a KNN approach optimized by PCA (principal component analysis), and they have achieved good results on classifying the CIFAR-10 (Fig. 1) dataset [2]. In the same area, L. H. Thai et al. have used SVM together with artificial Neural networks to construct their model. They use feature-based sub-images and feed them to neural networks, and they use the SVM as the last layer that receives the results of the neural nets. And this approach made them reach a precision of 86% by applying their model on classifying human numerals [3]. As one of the first attempts at using convolutional neural networks Y. leCun et al. are one of the first ones who use convolutional layers and subsampling in order to extract the right features from images, even if the shape of the object inside the image has a large range of variance. Such as handwriting, and this made them achieve great results and inspire all the later CNN users on both image classification and NLP [4]. And to optimize the speed of training a neural Net and its variants Sergey Ioffe et al. make the normalization of each layer, the thing that made each layer learn independently. This addition reduces the overfitting, enabled the use of a higher learning rate and consequently makes the training faster and also enables the use of a larger number of layers [13]. Also in the agricultural area, Horea Muresan and Mihai Oltean have collected a new high-quality dataset, concentrated only on fruits named fruit-360, and to prove the quality of their dataset they apply a CNN based classifier, and they have got great results [19].

In this work, our objective will be to make a comparative study between the well know methods that attempted to solve the problem of image classification and to be more specific we will use K-nearest neighbor, Support vector machine classifier, Deep neural networks, and Convolutional neural networks, and after each implementation, we will mention the strengths and weaknesses of each method. and it's worth mentioning that all our tests will depend on the well known Cifar10 dataset, since their images have small dimensions (32X32X3), and it will let us experiment with our tests without the need for the clouds and expensive hardware. Despite the fact that it's hard to achieve good results with such highly pixelated images. And after choosing the right classifier and prove it by results we will apply it on our main problem which is the classification of red fruits, seeing their our importance to our country and especially to our country.

The rest of this paper will be organized as follows: we will begin by a study case section and, in the coming section, we will describe the K-Nearest neighbor its formal implementations, its applications on image classification and the results achieved with it as well as its weaknesses. Then we will devote the next section to the shallow learning method (Support vector machine) and its performance on the cifar10 dataset. After that, we will study the uniform neural networks, the difficulties to build a robust deep neural network and its performance in the same dataset. And we will leave the last section to the strongest method which is the Convolutional neural network and its performance of cifar10, and finally, we will conclude by a global conclusion which summarizes our work and gives an idea about our future perspectives.

II. Study Case

The kingdom of morocco depends largely on agriculture and it's one of the principal incomes of the country, according to Wikipedia, the agricultural sector in morocco accounts for approximately 13-15% of GDP (gross domestic product) as shown in (Fig. 2) and employs about 40% of the national workforce, and if we take the year 2011 as an example, we find that Morocco's GDP is 221 billion dollars and the agriculture has contributed to it by 15% [1].

Thus, improving the Quality or Quantity of agriculture directly affect the GDP of the country. For this reason, farming and agriculture, in general, is a strong power that can effectively ameliorate the income of the country. And for this work, we are trying to enter this interesting sector by the gate of smart farming, and we are trying to take advantage of our proximity to an agricultural area that takes a special care of the red fruits, and the possibility that we can enough information about the subject, to orient our objective

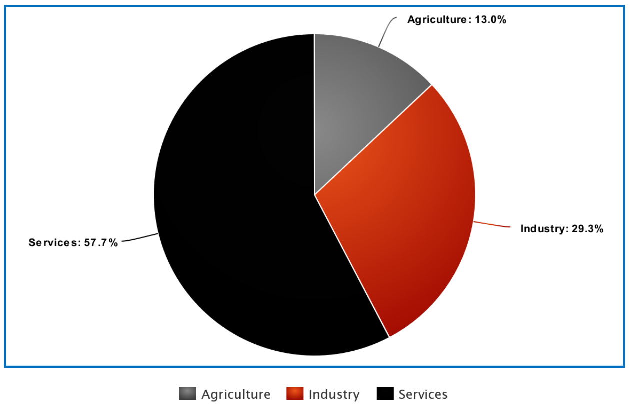
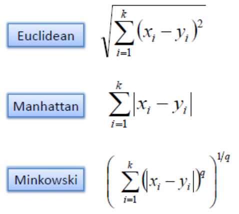


Fig. 2. GDP of Morocco.

to classify the different species of the red fruits after studying and analyzing the different existing approaches, and dedicate the best-found method to our case study.

III. K-Nearest Neighbor Classifier

K-nearest neighbor is a simple algorithm that stores all available cases and classifies new cases based on a similarity measure (e.g., distance functions, Fig. 3). A case is classified by a majority vote of its neighbors, with the case being assigned to the class most common among its K nearest neighbors measured by a distance function.



$$\sqrt{\sum_{i=1}^{k}(x_i-y_i)^2}$$

$$\sum_{i=1}^{k}|x_i-y_i|$$

$$\left(\sum_{i=1}^{k}(|x_i-y_i|)^q\right)^{1/q}$$

Fig. 3. Distance functions.

if K = 1, then the case is simply assigned to the class of its nearest neighbor. In general, a large K value is more precise as it reduces the overall noise. To choose the optimal value for K is best done by first inspecting the data, and cross-validation is another way to retrospectively determine a good K value by using an independent dataset to validate the K value, but historically, the optimal k for most datasets has been between 3-10. that produces much better results than 1NN.

A. K-Nearest Neighbor and Image Classification

The K-Nearest neighbor classifier is by far the most simple machine learning classifier used in image classification. This machine-learning algorithm doesn't actually learn anything. To use it we simply flatten the images and turn them into vectors before passing them to the kNN classifier, which simply keeps them in memory without any processing. In other words, it keeps all the hard work to the prediction step. When we ask the classifier to predict the class of a new image, it calculates the distance between that new image and all the datasets which already kept in memory using one of the distance functions. Then it chooses the most k similar ones to the image. Next, it decides which class is more suited for this image.

B. Experiments and Results

In the beginning, it's hard to decide the most suited value of k to our problem nor the most effective distance function. So in our experiments, we used one of the most well-known methods which is cross-validation to decide the most effective hyperparameters to our problem (image classification), but we didn't focus that much on the distance function we just used the most known one (euclidean distance). After running several tests and experimenting several values of k we found approximately the same results for k between 3-10, but the best accuracy we have got is approximately 29%. But according to the best-known results, K-Nearest neighbor can reach 35% accuracy if it's used with the right distance function and right value of K. and there are also other ways such as principal component analysis which could improve its performance furthermore. Moreover, KNN nearest neighbor could be used in combination with convolutional neural networks to increase its accuracy[2].

C. Limitations

This approach has several flaws. Apart from its low accuracy, it also suffers from the extensive memory usage, which means that with a large dataset we will have problems to store the dataset, and also it has another major flaw, it has to do all the work in the prediction time so that the user must wait for the classifier to compare its image to all the dataset and calculate the distance between them and give it the most k nearest classes to the image, and this kind of behavior is not acceptable in real-time applications.

IV. Parameterized Classifiers

Using parameterized classifiers (Fig. 4) helps us overcome the major flaw of K-Nearest Neighbor because in this case all the time-consuming tasks are done in the training stage. Once the training is complete, we can discard all the training dataset and free the memory, we just preserve the learned parameters W and b. And since we can have these parameters (w and b), we quickly predict the new test data since all we have to do is a simple linear transformation:

$$f(x_i, W, b) = Wx + b$$

A. Train a Linear Classifier

To train this type of models we only need to adjust the parameters W and b in a way that helps us achieve the best possible accuracy, and we do accomplish so with help of a loss function (quantifies how well our prediction agree with the ground-truth label) which we try to minimize using Stochastic gradient descent or one of if its variants.

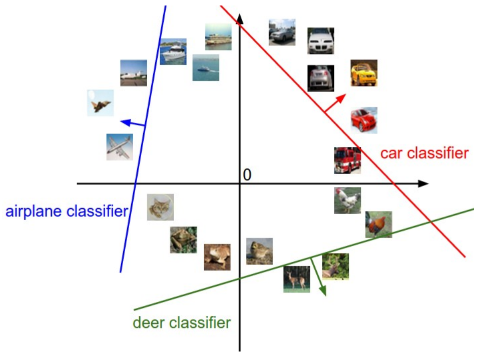


Fig. 4. Linear classfier.

B. Loss Function

The loss function is one of the most important pieces of all parameter based classifiers, and as we have previously mentioned, the loss function tells us how good our prediction compared to the ground truth label [6]. For linear classifiers, we can use multiple loss functions, but the most commonly used are this two:

- Multi-class Support vector machine, also known as hinge loss: inspired from the famous support vector machine classifier [5]:

$$s_j = Wx_i + b$$
$$L_i = \sum_{j \neq y_i} max(0, s_j + s_{y_i} + 1)$$

- cross-entropy: which used with softmax (Fig. 5) classifier that uses probabilities to describe the confidence of each class:

$$s_j = Wx_i + b$$
$$L_i = -\log(\frac{e^{s_{y_i}}}{\sum_j e^{s_j}})$$

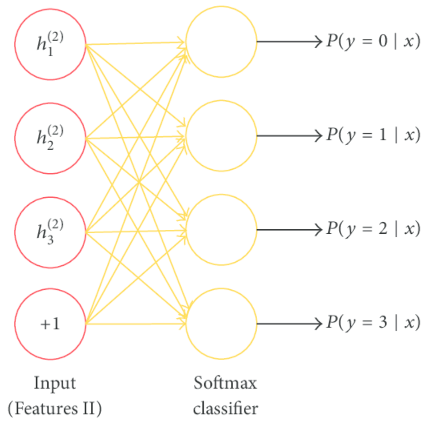


Fig. 5. Softmax classifier.

After calculating the loss function L_i of each example in the batch (or minibatch), we do calculate its mean $L = \frac{1}{n}\sum_{i=1}^{n} L_i$, to get a more global view, and converge in the direction of all the training examples, instead of zigzagging in the direction of each training example at each iteration and consequently slow down the training.

C. Limitations

Although linear classifiers are much better than K-nearest neighbor method and overcomes most of its flaws, in terms of accuracy they still suffer, in our case we have tried the two versions of linear classifiers (the svm based, and softmax classifier), we didn't exceed 45% in both cases, despite the all the efforts we did to tune parameters and experiment with the combinations of hyperparameters.

V. Neural Networks

Neural networks are the most effective machine learning algorithm, and it can easily outperform almost any other machine learning algorithm in any task that involves learning, and its architectures has a wide range of variants (DNNs, CNNs, RNNs, AutoEncoders, GANs...), which make it makes it capable to perform a large variety of tasks such as Object detection, Image recognition, Regression, compression ..., and it's used in almost any modern applications that require some sort of intelligence. Even the most simple form of a Neural network (shallow Network) which consists of only two layers (hidden, output), is considered as a universal function, and in theory, it could approximate any existing mathematical function.

Neural networks share a lot of the common notions of the classical methods of machine learning (especially the ones that uses trainable parameters) such as normalization, loss functions, activation functions, optimization techniques (gradient descent, stochastic gradient descent), but Neural Nets are characterized by another type of notions that are specific to them like the fact that they could contain a large number of layers, and that they use backpropagation to train an arbitrary number of layers, which make them special and give them high flexibility that enables them to adjust to any kind of data. The powerful architectures of Neural Nets made them prove their effectiveness and attract the curiosity of the researchers which consequently made them one of the most active research

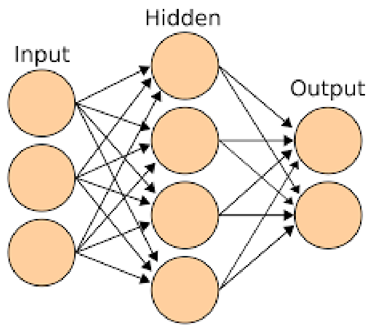


Fig. 6. Simple shalow network.

areas. They have focused on every detail of Neural Nets. There are researches in weight initialization, activation functions, regularization, normalization, and even in the right number of layers. Thus, in this work, we have included the most recent terms and tried to use the latest studies and the best choices to construct our own neural network and use it to classify the Cifar10 dataset, and the following subsections describe the elements that we have used in our implementations.

A. Regularization

Neural networks are considered the most flexible machine learning algorithms and can adapt with any type of data as discussed previously, but this flexibility comes with a cost: Overfitting (Fig. 7), In other words, they memorize the training data which make them unable to generalize and recognize new data, and that's where the term of regularization could help to prevent this Phenomenon.

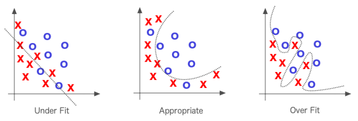


Fig. 7. Overfitting phenomenon.

There two major types of regularization L2 regularization and Dropout, there's also L1 regularization but its not preferable.

1) L2 regularization: is the most known, and it's not exclusive to neural networks, it's also used with a large variety of machine learning algorithms, it's simply an addition of the Frobenius norm of the weight matrix to the loss functions, which decays the weights and consequently encourages the simple version of the neural network model, the thing that prevents overfitting to some extent. and since the L2 regularization encourage small weights, it also does another important job that serves positively some non-linearities such as sigmoid and tanh, because it confines the weights in the small portions where it can make use of the linear area of the non-linearities as shown in (Fig. 8), this thing accelerates the learning process because the gradient isn't dead, in contrast to areas where $|w|$ are large.

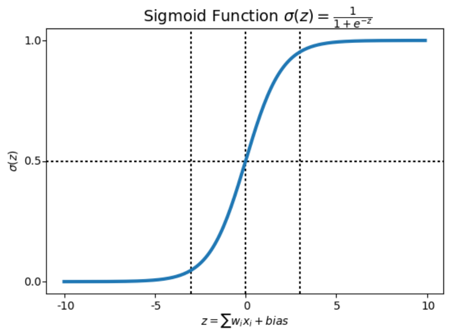


Fig. 8. Linear portion of the sigmoid

2) Dropout: does also a similar job and make each time a simple version of the neural network, by deactivating a number of neurons on each layer according to so some pre-specified probability (Fig. 9) and it does the work because it prevents the model to rely on any specific feature, and make it take different paths each time so that it can finally generalize well [10].

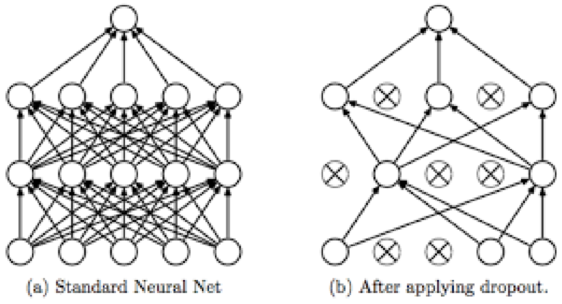


Fig. 9. Dropout

B. Input Normalization

Input normalization/standardization is a simple preprocess of the input data, that can be summarized by the following equations:

$$\mu = \frac{1}{m}\sum_{i=1}^{m} x_i \quad (1)$$

$$\sigma^2 = x_i \quad (2)$$

$$\hat{x_i} = \frac{x_i - \mu}{\sqrt{\sigma^2 + \epsilon}} \quad (3)$$

Input normalization is an important process for a large variety of machine learning algorithms. It does work because it prevents the large variance between input features, which could cause the data to be sensible to the parameter update and risk to make the gradient overshoot in some directions. Input normalization also makes the input data zero mean, which inhibits the parameter update to be in the same direction.

C. Optimization

The optimization is the most important building stone of the learning process of a neural network. Almost always meant by optimization gradient descent and its variants in the context of the literature of neural network training, we could train our model using another kind of optimizers such as nature-inspired algorithms(meta-heuristics), but by far the gradient descent and its variants are the most suited to train neural nets. The original version of gradient descent is considered to be too slow because at each iteration it needs to explore all the training data to make one step. As a major successor of gradient descent is the mini-batch stochastic version of gradient descent, which doesn't need to traverse all the data, it just takes a random prefixed amount of training data from the training set and evaluate the loss and then take a step. Although, SGD make a good job in replacing Gradient descent, it has some drawbacks. SGD makes a parameter update with just a subset of the training set, which makes the direction of the update has some variances, and thus, the path taken by SGD, will oscillate toward convergence, and those oscillations forces us to use a small learning rate which consequently slows down the learning process. For this reason, there were several attempts to solve this problem, the most famous ones are: sgd+momentum [7], RMSprop [8], and Adam [9].

D. SGD + Momentum

The idea behind SGD+momentum is that it adds a little momentum to the gradient, by adding a new term called velocity, which is simply the exponentially weighted average of the gradient. In one side it helps us escape from the critical points where the gradient could die, and in the other side it tends to average out the oscillations in the directions that aren't towards minima, which make by making them smooth, and since the motion, toward the minima, is stable it doesn't affect the velocity toward convergence, instead, it accelerates the learning process and it allows us to use larger learning rate. The following equations are the simple modification made to the update when we use SGD+momentum:

$$V_{\partial W} = \beta V_{\partial w} + (1-\beta)\partial W \quad (4)$$

$$V_{\partial b} = \beta V_{\partial b} + (1-\beta)\partial b \quad (5)$$

$$W = W - \alpha V_{\partial W} \quad (6)$$

$$b = b - \alpha V_{\partial b} \quad (7)$$

E. RMSprop

RMSprop is from the family of adagrad (adaptive gradient) optimizers. This type of optimizers tries to adjust the learning rate of each parameter independently, by performing smaller updates to the frequently occurring features, and larger updates for parameters associated with infrequent features, the thing that make them able to handle sparse data very well, but given the cumulative nature of the term that tries to adapt the learning rate, it creates the problem of continuously decreasing the learning rate, which leads to halting the learning process. That's why RMSprop along with other algorithms come as extensions to the original adagrad algorithm, as an attempt to fix this disadvantage, in the case of the RMSprop, it simply tries to replace the accumulative term by a running average which makes it decay with time, and forget about the old values as shown in following equations:

$$S_{\partial W} = \beta S_{\partial W} + (1-\beta)\partial W^2 \quad (8)$$
$$S_{\partial b} = \beta S_{\partial b} + (1-\beta)\partial b^2 \quad (9)$$
$$W = W - \alpha \frac{\partial W}{\sqrt{S_{\partial W} + \epsilon}} \quad (10)$$
$$b = b - \alpha \frac{\partial b}{\sqrt{S_{\partial b} + \epsilon}} \quad (11)$$

F. Adam

Adaptive momentum is one of the most effective algorithms that used to optimize NNs, and recently it becomes the standard. Adam optimizer doesn't reinvent the wheel, instead, it's simply a combination of the concepts of the two previously discussed optimizers, it takes advantage of both of them, and it almost always outperforms them in practice. and the following equations show how it combines the two set of equations of the SGD+momentum and RMSprop:

$$V_{\partial W} = \beta_1 V_{\partial w} + (1-\beta_1)\partial W \quad (12)$$
$$S_{\partial W} = \beta_2 S_{\partial W} + (1-\beta_2)\partial W^2 \quad (13)$$
$$\hat{V}_{\partial W} = \frac{V_{\partial W}}{(1-\beta_1^t)} \quad (14)$$
$$\hat{S}_{\partial W} = \frac{S_{\partial W}}{(1-\beta_2^t)} \quad (15)$$
$$W = W - \alpha \frac{\hat{S}_{\partial W}}{\sqrt{\hat{S}_{\partial W} + \epsilon}} \quad (16)$$

There are more optimizer and more alternatives, that we haven't discussed here such as nestrove algorithm which is an extension to sgd+momentum, and we didn't mention them because they aren't used in practice, but as shown in (Fig. 10), the Adam optimizer is the most powerful.

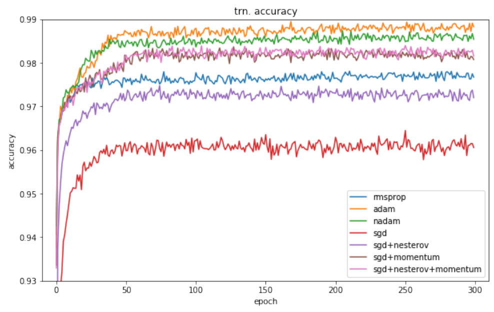


Fig. 10. Optimizers comparison

G. Deep Neural Networks

Deep neural networks are simply a version of neural networks (Fig. 6) with more than one hidden layer (Fig. 11). In principle, you don't need a deep neural network. And given enough training data, a large neural net with only one hidden layer can approximate any mathematical function. But the problem with extremely large single hidden layered neural networks is the lack of generalization, they could memorize but this is not enough. If we test a super-wide shallow network with new data, it won't do well, even if it could memorize all the training data. and this is not useful in a real-world scenario.

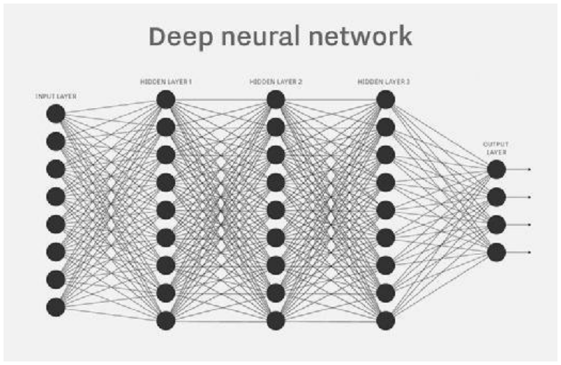


Fig. 11. Deep Neural Network

In the other hand, a deep neural net with multiple hidden layers learns in a different way, the first layers learn to recognize basic things such as edges in the case of pictures, and the deeper layers learn more complicated things that are constructed from combinations of the things learned in the earlier layers, and this gives multi-layered neural nets the ability to generalize better, and this serves better a practical application. Deep neural networks are extremely useful, they generalize well, learn better and achieve better results, but the complexity in there architecture comes with a cost, they are hard to train in comparison to the other simple shallow networks, deep neural networks use the same principle as the other regular networks, but in case of DNN, there are some other things that should be considered, like the weight initialization and batch normalization to simplify the learning process.

H. Weight Initialization

One of the starting points to take care of while building your network is to initialize your weight matrix correctly. Weight initialization also plays an important role in training Deep Neural networks, it might seem evident, and we might think that we could initialize weights with just some random values or just initialize them with zero. But it's not that simple, if we do initialize them with zero, we will get the same output results, and eventually get the same results which will lead us to update the weights with the same values, and also if we think to initialize them with the extremely small values we will risk having weights decays in deeper layers as shown in (Fig. 12), and if we initialize them with large numbers, we will suffer from having vanishing gradients especially with some non-linearities such as sigmoid and tanh.

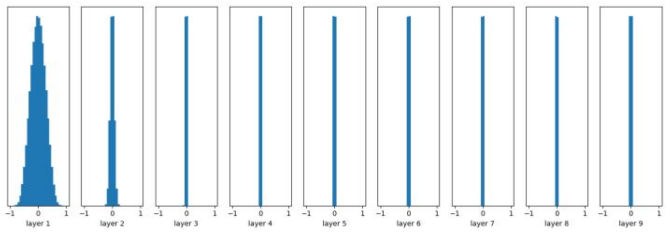


Fig. 12. Weight decay in deep layers

That's why weight initialization is considered to be an important task and gets the attention of many researchers, and it's one of the widest areas of researches that concerns neural networks, but the most known two methods are the one called xavier initialization [11], and another extension [12] to it that works better with relu activation variants. And the initialization equation of those two methods are as follows:

$$W^{[l]} = random(W)(from - normal - distribution) * \frac{1}{\sqrt{n^{l-1}}}$$
$$W^{[l]} = random(W)(from - normal - distribution) * \frac{2}{\sqrt{n^{l-1}}}$$

where n^{l-1} is the size of the input from the previous layer.

I. Batch Normalization

Training Deep Neural Networks is complicated Due to the differences in distributions of the inputs of each layer caused by the constant changes of the parameters of previous layers, and this makes training process becomes too hard, and to make progress we have to lower the learning rate and be too careful when we initialize the parameters, the thing that make the training too slow. We refer to this phenomenon as an internal covariate shift and address the problem by normalizing each layer's inputs (Fig. 13). When we introduce normalization and normalize each training mini-batch, we can use a larger training rate and be less cautious about the initialization process. And it also acts as a regularizer. Also, batch normalization allows each layer of a network to learn by itself a little bit more independently of other layers.

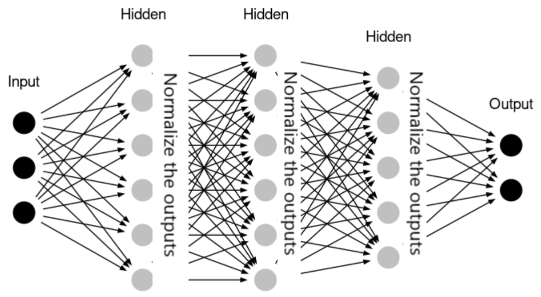


Fig. 13. Batch Normalization

Batch normalization normalizes the output of a previous activation layer by subtracting the batch mean and dividing by the batch standard deviation. However, after this shift/scale of activation outputs by some randomly initialized parameters, the weights in the next layer are no longer optimal. SGD (Stochastic gradient descent) undoes this normalization if it's a way for it to minimize the loss function. Consequently, batch normalization adds two trainable parameters to each layer (Fig. 14), so the normalized output is multiplied by a "standard deviation" parameter (gamma) and add a "mean" parameter (beta). In other words, batch normalization lets SGD and its variants do the denormalization by changing only these two weights for each activation, instead of losing the stability of the network by changing all the weights.

The Batch normalization operation is simply governed by the following equations [13]:

Input: Values of x over a mini-batch: $\mathcal{B} = \{x_{1...m}\}$;
Parameters to be learned: γ, β
Output: $\{y_i = \mathrm{BN}_{\gamma,\beta}(x_i)\}$

$$\mu_{\mathcal{B}} \leftarrow \frac{1}{m}\sum_{i=1}^{m} x_i \qquad \text{// mini-batch mean}$$
$$\sigma_{\mathcal{B}}^2 \leftarrow \frac{1}{m}\sum_{i=1}^{m}(x_i - \mu_{\mathcal{B}})^2 \qquad \text{// mini-batch variance}$$
$$\widehat{x}_i \leftarrow \frac{x_i - \mu_{\mathcal{B}}}{\sqrt{\sigma_{\mathcal{B}}^2 + \epsilon}} \qquad \text{// normalize}$$
$$y_i \leftarrow \gamma\widehat{x}_i + \beta \equiv \mathrm{BN}_{\gamma,\beta}(x_i) \qquad \text{// scale and shift}$$

Fig. 14. Batch Normalization equations

Where γ and β are trainable variables.

We have implemented this notion in our own version of the neural network, and experiment with different sizes of batches to see the differences that batch normalizes make and how it accelerates the training phase, and we have summarized our experiments in (Fig. 15).

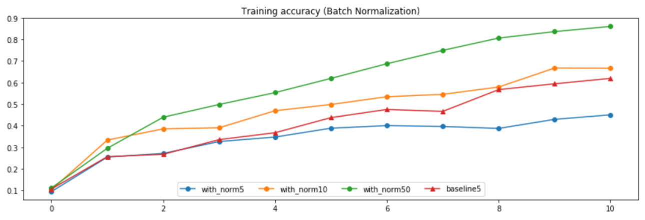


Fig. 15. Batch Normalization Effect

As shown in the figure above, despite the clear acceleration of the training, it's clear that the size of the batch has a huge effect on the effectiveness of batch normalization which could make a problem, that's why there where several attempts to solve it, such as layer normalization [14] and also recently appears an activation function that's his author claims that it eliminates the need of the batch normalization [15], but all those claims are still under test and the batch Norm still prove its effectiveness for the time being.

J. Deep Neural Networks and Cifar10

After implementing our own version of the neural network, that we have tried to insert all the above-discussed concepts in it, and we've tried to experiment with the best possible options, we have adjusted our network to classify Cifar10 dataset, that we chose it to be our criterion of the performance of our classifiers. Then after building the most convenient version and after a series of hyperparameters tuning to find the best combinations of hyperparameters, we test and we get an accuracy that exceeds 55%. And that's kind of disappointing after all this work, but that's happened because we've ignored the fact that the dataset is images. And here comes the role of another type of Neural Network called Convontional neural networks, which more suited to this kind of dataset (images).

VI. Convolutional Neural Network

Convolutional Neural network is just a deep Neural network with a different structure, and it has been proven empirically that CNN is by far the most effective Neural Network architectures for image classification, it even outperforms the human performance on classifying the imageNet dataset on 2015 [12]. This kind of powerful performance on image recognition tasks enabled the CNNs to achieve great results in bigger use cases such as object detection, and segmentation. The Convnets benefit from all the features and ideas of the usual deep neural network, they use all the techniques explained in the previous section. But they use two additional layers (convolution and pooling), which are the real cause behind the outstanding performance of CNNS.

A. CNN History

The concept that has lighted the idea of Convolutional Neural Networks has begun decades ago with the conclusions of the two famous research papers titled: "Receptive Fields of Single Neurons In The Cat's striate cortex" in 1959 and "Receptive fields, binocular interaction and functional architecture in the cat's visual cortex" in 1962, made by Hubel and Wiesel. they observed that the neurons have a hierarchical organization, and that earlier layers response to light orientations, and later layers response to light orientations and movements, and the last set of layers (that contains the most complex types of neurons) responds to movements and endpoints. Then in 1980, Fukushima [16] has built the first example of a network architecture model that has this idea of simple and complex cells. afterward, in 1998 they built the first model that applies backpropagation and gradient-based learning to train a CNN and they were able to do a good job on document recognition [17], and also did well on digit recognition, but it wasn't able to scale to more complex data until the appearance of the AlexNet [18] in 2012, which has been able to achieve great results, scale to larger and complex data, and make use of the latest hardware.

B. Convolutional Layer

The convolutional layer is the most important part of a convolutional network and it's the one that does contain the most valuable set of parameters. The Convolutional layer's parameters consist of a set of learnable filters (also called kernels or feature detectors); every filter contains a small set of weights spread vertically and horizontally and through a specified depth (Fig. 16). A usual size of the first layer is 5X5X3 or 3X3X3.

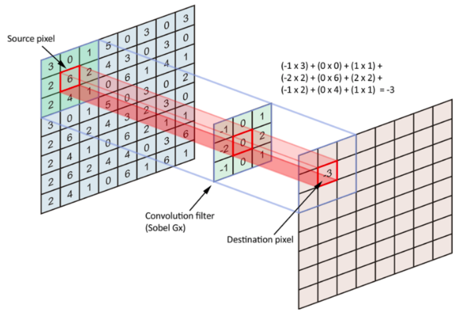

Source pixel
(-1 x 3) + (0 x 0) + (1 x 1) +
(-2 x 2) + (0 x 6) + (2 x 2) +
(-1 x 2) + (0 x 4) + (1 x 1) = -3
Convolution filter
(Sobel Gx)
Destination pixel

Fig. 16. Convolution operation

To perform the convolution operation, we slide each filter across the width and height of the input volume and compute the dot products between the entries of the filter and the input at any position, and this operation will produce a 2-dimensional activation map that gives the response of that filter at every spatial position, and eventually, through training process the network will learn filters that activate when they see some type of visual feature.

C. Pooling Layer

The pooling layer reduces the number of parameters and calculations in the network. Thus, it improves the efficiency of the network and avoids over-learning. The pooling layer receives several feature maps and applies to each of them a pooling operation, which used to reduce the size of the images while preserving their important characteristics. For this, the image is cut into regular cells, then the maximum value is kept within each cell (Fig. 17). In practice, small square cells are often used to avoid losing too much information. The most common choices are adjacent cells of size 2X2 pixels that do not overlap. The same number of feature maps is preserved, but these feature maps are much smaller.

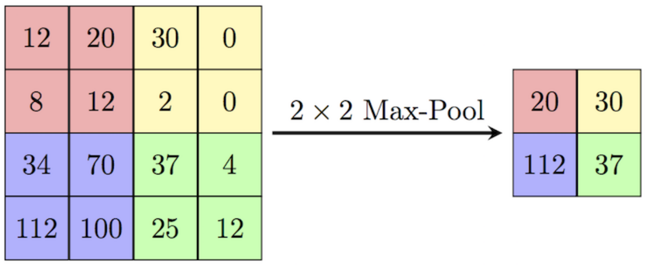


Fig. 17. Pooling operation

Thus, the pooling layer makes the network less sensitive to the position of features: the fact that a feature is a little higher or lower, or even that it has a slightly different orientation should not cause a radical change in the classification of the image.

D. CNN Architectures

CNN has proven that it is the most efficient image classifier since the 2012 imageNet international competition using AlexNet architecture [18], which is very similar to LeNet architecture which used in 1998 for digit recognition, But AlexNet after 14 years was able to take advantage of the computational power of GPUs, and consequently could be used for more powerful and realistic datasets, such as ImageNet. And after the great performance achieved at that time by AlexNet, the world's attention has turned again toward ConvNets, and all the subsequent winners used one of the variants of CNN.

Until now, dozens of CNN architectures have appeared, But the top architectures that we're able to positively affect the evolution of CNN were 3: VGGNet 2014, GoogLeNet 2014, ResNet 2015.

1) VGGNet (Visual Geometry Group Net): was ranked second in 2014 competition, but it used a distinctive interesting idea with regard to the receptive field of the filters used in convolutional layers, so instead of using 5X5, 7X7 or 11X11 like in the case of AlexNet, VGG uses only two 3X3 receptive fields to replace the 5X5 filter, and five 3X3 to replace the 11X11 receptive field (Fig. 18). This way it could effectively replace the large filters without hurting the performance, in the case of 11X11 filter we get 121 parameters, and VGG achieves the same results with only 3X3X5 = 45 parameters[20].

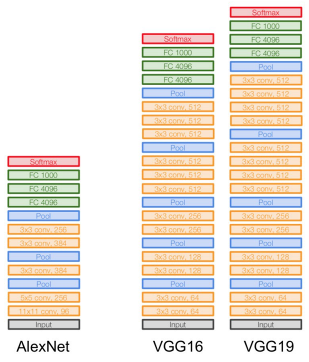


Fig. 18. AlexNet vs VGG

2) GoogLeNet: also known by the name of **Inception V1**, and it is the winner of 2014 competition, GoogLeNet is formed from a number of inception modules (Fig. 19), and each one of this modules contains a number of parallel convolutional layers, that uses filters with different receptive fields and a pooling layer in addition to a concatenation layer which sums up the output of the parallel layers depth-wise [21].

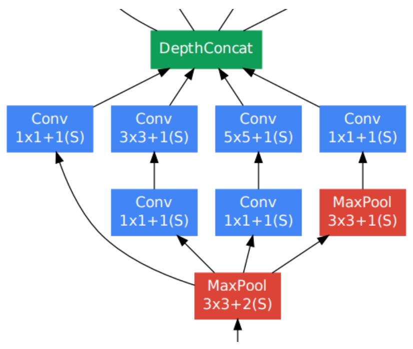


Fig. 19. Inception module

This way googleNet was able to increase the number of layers to 22, with 12 times less number of parameters in comparison to AlexNet.

3) ResNet: winner of 2015 competition, and the first one who outperform the human capability of classifying imageNet dataset (Fig. 20), with only 3.57 error.

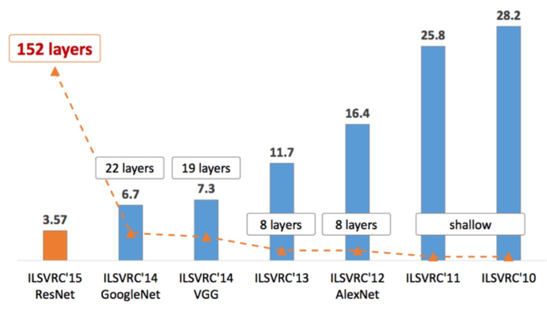


Fig. 20. Evolution of CNNs.

ResNet was able to dramatically increase the depth of the neural networks with an innovative idea which simplifies the $f(x)$ that needed to be learned by each layer (Fig. 21), this happens by adding an identity function to the residual $f(x)$, which means that the layer only needs to learn a $\Delta = H(x) - x$ [12].

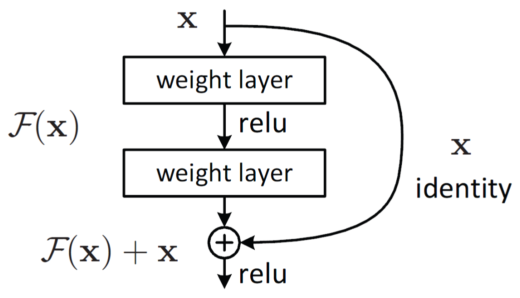

x
weight layer
relu
weight layer
$\mathcal{F}(\mathbf{x})$
x identity
$\mathcal{F}(\mathbf{x}) + \mathbf{x}$
relu

Fig. 21. Residual.

This way resNet was able to go very deep and use 152 layers. ResNet was also able to affect the normal deep neural networks and made them able to attain 1000 layers.

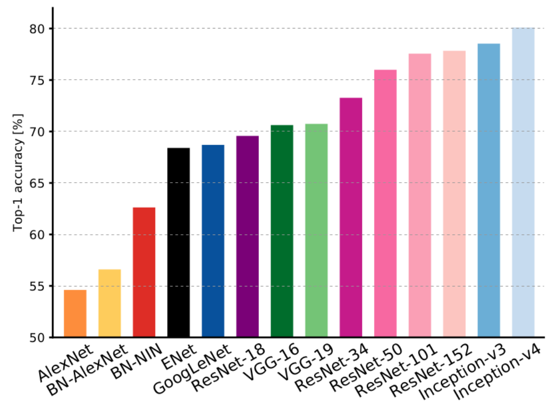


Fig. 22. Comparison of accuracy [22].

In the subsequent years, the architectures that could win the famous ILSVRC competition were only some kind of a hybridisation or reformulation of this architecture, and Inception v4 is an example of a hybridisation of resNet and googLeNet that gives the best performance in term of accuracy. And (Fig. 22 and 23) are an overview of the performance of this architectures.

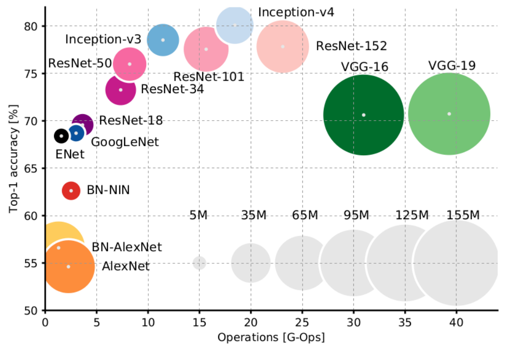


Fig. 23. General comparison of CNN architectures [22].

E. Experimental Results

After studying the theory behind convolutional neural networks, we have made an application that took advantage of all the notions discussed in the usual neural networks, and additional features of the convolution neural network, and after spending a fair amount of time searching we have found the hyperparameters and structure that would best suit our test dataset (cifar10) and give as a precision that exceeds 90% as shown in Table I.

TABLE I. CLASSFIER PERFORMANCES

Method	KNN	Linear Classifier	Neural Network	CNN
accuracy	28%	45%	55%	>90%

Despite our knowledge about the important types of architectures, and the way to achieve good results in image classification, we weren't able to experiment with these architectures and use them with Bigger and more realistic datasets such as imageNet, because of the lack of the adequate hardware. and we were forced to use the simple form of the convolutional neural network, but it was sufficient to get great results in the Cifar10 dataSet, and we also thought to apply these notions on a real use case that we could benefit from. So we chose to use it in agriculture and classify a dataset of red fruits since our nearby area (Larache city) is suitable for planting red fruits. Thus, we chose a subset (only red fruits Fig. 24) of a famous dataset that classifies fruits [19].

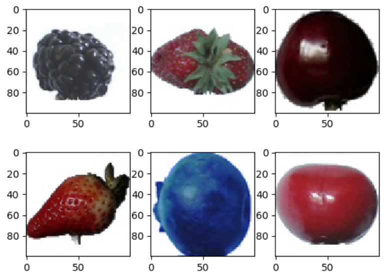


Fig. 24. Sample from red fruits set.

To accomplish this task, we adapt our convolutional neural net model to classify this subset by adjusting hyperparameters and preprocessing the raw images of the dataset. By doing so we have achieved a precision that reaches 99.9% because of the simple nature of the data set. And as presented in (Fig. 25) all the guesses of the model are correct.

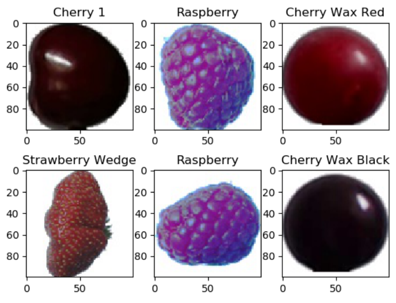


Fig. 25. Cnn model predections.

VII. CONCLUSION

In this work, we have experimented and tested a fair amount of classifiers that are used to recognize images, and we have known the strengths and weaknesses of each one of them, we have also studied in depth neural networks and convolutional neural networks and achieved good results in classifying our chosen datasets. And we are looking forward to do more, we want to explore all the variations of neural nets, and also apply them in more interesting fields of study like object detection and segmentation, and also use them in a real applications that could directly affect our everyday life.

REFERENCES

[1] Fanack, *Independent online media organization committed to publishing and disseminating balanced and informed analysis about the Middle East and North Africa*, consulted on 22-19-2019.

[2] Y. Abouelnaga, S. Ali, H. Rady, and M. Moustafa. *CIFAR-10: KNN-based Ensemble of Classifiers*. 2016.

[3] L. H. Thai, T. S. Hai, N. T. Thuy, *Image Classification using Support Vector Machine and Artificial Neural Network*, 2012.

[4] Y. LeCun, P. Haffner, L. Bottou, and O. Bengio, *Object Recognition with Gradient-Based Learning*, 1998.

[5] C. Cortes and V. Vapnik. *Support-Vector Networks*. In: Mach. Learn. pages 273-297. Sept. 1995.

[6] A. rosebrock. *Deep learning for computer vision with python*. 1st Edition., 2017. [online]. Available: https://www.pyimagesearch.com/deep-learning-computer-vision- python-book.

[7] N. Qian, *On the Momentum Term in Gradient Descent Learning Algorithms.*, In: Neural Netw. 12.1 , pages 145-151, Jan. 1999.

[8] S. Ruder. *An overview of gradient descent optimization algorithms*. 2017.

[9] P. Kingma, J. L. Ba, *ADAM: A method for stochastic optimization*. 2017.

[10] N. Srivastava, G. Hinton, A. Krizhevsky, I. Sutskever, R. Salakhutdinov, *Dropout: A Simple Way to Prevent Neural Networks from Overfitting*. 2014.

[11] X. Glorot, Y. Bengio, *Understanding the difficulty of training deep feedforward neural networks*. 2010.

[12] K. He et al., *Delving Deep into Rectifiers: Surpassing Human-Level Performance on ImageNet Classification*. 2015.

[13] S. Ioffe, C. Szegedy, *Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift*. 2015.

[14] J. L. Ba, J. R. Kiros, Geoffrey E. Hinton, *Layer Normalization*. 2016.

[15] G. Klambauer, T. Unterthiner, A. Mayr, *Self-Normalizing Neural Networks*. 2017

[16] K. Fukushima, *Neocognitron, A Self-organizing Neural Network Model for a Mechanism of Pattern Recognition Unaffected by Shift in Position*. 1980.

[17] Y. LeCun, L. Bottou, Y. Bengio and P. Haffner, *Gradient Based Learning Applied to Document Recognition*. 1998.

[18] A. Krizhevsky, I. Sutskever and G. E. Hinton, *ImageNet Classification with Deep Convolutional Neural Networks*. 2012.

[19] H. Mureşan, M. Oltean, *Fruit recognition from images using deep learning*. 2018.

[20] K. Simonyan, A. Zisserman, *Very deep convolutional networks for large-scale image recognition*. 2015.

[21] C. Szegedy et al., *Going deeper with convolutions*. 2014.

[22] A. Canziani, E. Culurciello, A. Paszke. *An analysis of deep neural network models for practical applications*. 2017.