REPUBLIQUE ALGERIENNE DEMOCRATIQUE ET POPULAIRE

MINISTERE DE L'ENSEIGNEMENT SUPERIEURE ET DE LA RECHERCHE SCIENTIFIQUE

UNIVERSITE IBN KHALDOUN - TIARET

MEMOIRE

Présenté à :

FACULTÉ MATHEMATIQUES ET INFORMATIQUE
DÉPARTEMENT D'INFORMATIQUE

Pour l'obtention du diplôme de :

MASTER

Spécialité : Génie Logiciel

Par :

ZERROUKI Sofiane

Sur le thème

Proposition et Evaluation d'un Modèle Deep Learning pour la Classification d'Images Basé sur les Espaces de Couleurs

Soutenu publiquement le 27/09/2020 à Tiaret devant le jury composé de :

Mr ALEM Abdelkader	MAA	Université Ibn-Khaldoun Tiaret	Président
Mr CHENINE Abdelkader	MAA	Université Ibn-Khaldoun Tiaret	Encadreur
Mr MOKHTARI Ahmed	MAA	Université Ibn-Khaldoun Tiaret	Examinateur

2019-2020

ABSTRACT

Deep learning and image classification are used in different research areas (medicine, social networks, etc.) and in various ways (image / video, audio and text). The classification of images can be considered among the essential and necessary things in the field of artificial intelligence, with progress in this field Deep learning.

ResNet20 architecture has brought out the latest in technology to perform image classification tasks, they mostly take data sets as inputs in RGB format images, though many other color spaces are available.

In this research we evaluate deep convolutional neural network to classify the 60 thousand images in the Cifar100 into the 100 different classes with five color spaces (RGB, HSV, LUV, LAB, YUV) and trained each one of them using deep learning architecture models namely ResNet20. The results obtained show a minor change in accuracy but we are focus at the difference between different color-spaces when the object is to understand the impact of image color-space on the performance of CNN models in Image classification.

Keywords: Deep Learning, image classification, ResNet20, Color Spaces, Cifar100, CNNs Models.

RESUME

L'apprentissage profond et la classification d'images sont utilisés dans différents domaines de recherche (médecine, réseaux sociaux, etc.) et de diverses manières (image / vidéo, audio et texte). La classification des images peut être considérée parmi les choses essentielles et nécessaires dans le domaine de l'intelligence artificielle, avec des progrès dans ce domaine d'apprentissage profond.

L'architecture ResNet20 a mis au point la dernière technologie pour effectuer des tâches de classification d'images, ils prennent principalement des ensembles de données comme entrées dans un format d'images en RGB bien que de nombreux autres espaces colorimétriques soient disponibles.

Dans cette recherche, nous évaluons le réseau de neurones à convolution profond pour classer les 60000 images du Cifar100 dans 100 classes différentes avec cinq espaces colorimétriques (RGB, HSV, LUV, LAB, YUV) et avons entrainer chacune d'elles en utilisent l'architectures ResNet20.

Les résultats obtenus montrent un changement mineur de précision, mais nous concentrons sur la différence entre les différents espaces de couleur lorsque l'objet est de comprendre l'impact de l'espace couleur de l'image sur les performances des modèles CNN dans la classification d'image.

Mots clés : apprentissage profond, classification des images, ResNet20, espaces colorimétriques, Cifar100, modèles CNNs.

ACKNOWLEDGMENT

First of all, " الحمد لله"

All thanks to Allah who create me when I was nothing, all thanks to Allah who give everything.

I would like to express my special thanks of gratitude to my supervisor "Mr. Chenine Abdelkader" for their able guidance and support in completing my project.

I would like to express my gratitude to my mam "Mrs. Derrak khayra" and my father "Mr. Zerrouki m'hamed" and all my family, they always supported me in both of incorporeal and physically.

Thanks to all my teacher from whom we learned a lot during my period of study at Ibn

Khaldun Tiaret University especially in computer science department.

Thanks to all my friend that's encourage me to prepare this research: Djamel Zerrouki, Oussama Djerboua, Khaled awad, Ilyes Aroussi,Ilyes Makboul, Houari, Djilali , Karim , Nasser , Ibreahim and all my friend.

A good thank you to my college at the youth center.

Finally, Thanks to everyone who loves science and learning .

"وَمَا تَوْفِيقِي إِلَّا بِاللَّهِ ۚ عَلَيْهِ تَوَكَّلْتُ وَإِلَيْهِ أُنِيبُ"

Contents

LIST OF FIGURES

LIST OF TABLES

GENERAL INTRODUCTION

Deep learning has been in the forefront of solving quite a few real-world problems. A machine is made to learn from data and is expected to improve its performance over a period of time.

Deep learning represents an exciting intersection of machine learning and artificial intelligence, and a very significant disruption to society and industry. It transfers the logical burden from an application developer, who develops and scripts a rules-based algorithm, to an engineer training the system. It also opens a new range of possibilities to solve applications that have never been attempted without a human. In this way, deep learning makes machine vision easier to work with, while expanding the limits of what a computer and camera can accurately inspect.

Convolutional Neural Networks (CNNs) is the most popular neural network model being used for image classification problem which sometimes exceeds the human vision. There are many techniques for improving deep development like data augmentation, batch normalization , regularization and dropout.

At the end of this research we can response at this question: Can we use different image color-space than the usually used RGB to get better performance? This what we shall show in this thesis by converting CIFAR100 datasets into five color-spaces namely RGB, HSV, LUV, LAB, YUV and train each one of them in ResNet20 architecture.

I. BACKGROUND

This work is a continuation of the previous thesis, where they demystified the Impact of color space on the Convolutional Neural Network following the paper [1]. They said that ResNet20 and CapsulNet have showing minor change in accuracy in ResNet20 and also in CapsulNet but even a small percentage may make the difference, in the other hand LUV is good alternative.

To our knowledge there is two other controversy published papers, the authors claim that image color space have impact over CNN performance, they said that LUV color space is a best alternative to work with CNN model while YUV color space is the worst one. A more recent paper shows that image color space by itself have no effect over CNN performance, they said that the accuracy did not vary too much using different color spaces but it can be used as a sort of data augmentation by combined different image color-spaces from the original data-set using dense-net model, this will help to deal with common issues such as the vanishing gradient problem and the problem of over-fitting and get better performance.

II. THE SCOPE OF THIS RESEARCH

This thesis focusses on study of the impact of color-space in image classification by deep learning model, especially Deep Residual Learning (ResNet) won the first place in the ILSVRC 2015 classification competition with top-5 error rate of 3.57% by experimenting using Keras [2] which is a high-level neural networks API, written in Python and capable of running on top of TensorFlow. There are other architectures like AlexNet, VGG, GoogLeNet and framework like Caffe, Pytorch. We also focus in GPU computing.

III. RESEARCH GOALS AND QUESTIONS

Our goal is to answer a simple question: The RGB color space is the most color space used for datasets in image classification using deep learning model, can the performance of deep learning model be improved by using different color spaces?

IV. STRUCTURE OF THE MANUSCRIPT

CHAPTER 1:

The first chapter addresses an overview of the field of artificial intelligence and in particular deep learning techniques which is a branch of the art of machine learning.

We'll also see the basics of neural networks in deep learning. It also contains some of the techniques used in machine learning such as the loss function, batch normalization and gradient descendent.

CHAPTER 2:

In this chapter we show what is the convolutional neural network and how classify images with it. we show many techniques for optimization of our model. Also, we show how the architectures used in this research namely ResNet20.

CHAPTER 3:

This chapter describes the image color spaces in particular RGB, HSV, YUV, LUV, LAB, and how to obtain them from RGB color spaces.

CHAPTER 4:

Describes the tool, library, and CIFAR100 dataset used in the experiment. The process of converting a data set to different color spaces, as briefly described in Implementation.

Finally, we will analyze the results and come up with a summary.

CHAPTER 1

DEEP LEARNING AND ITS APPLICATION

I. INTRODUCTION

This chapter describes what is deep learning! what is it used for? And in answer to the second question is pretty much everywhere. recent application includes things such as detecting spams and Enums, recognition images and pictures, and even diagnostic sum times with more precision than doctors and of course one of the most celebrity application of deep learning is self-driving cars and where is the hard of deep learning this wonderful object called Neural network.

Neural networks like the process of how the brain operates with neurons that few are bit information.

II. 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 subfields of Computer Science (figure 1).

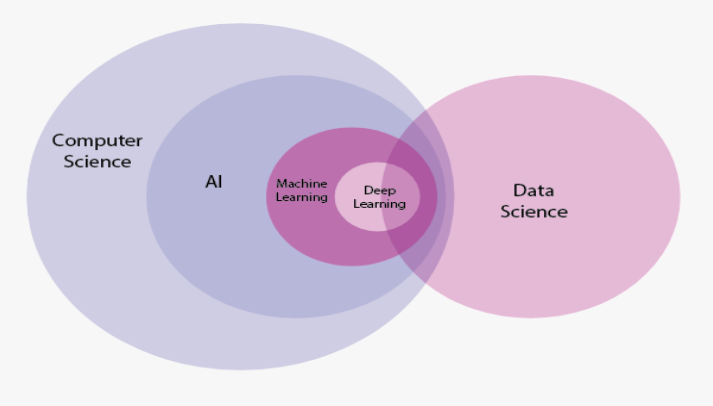


Figure 1 : Machine Learning is a subfield of Computer Science.

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?"

This question opens the door to a new programming paradigm. In classical programming, the paradigm of symbolic AI, humans input rules (a program) and data to be processed according to these rules, and outcome answers, With machine learning, humans input data as well as the answers expected from the data, and outcome the rules (a program). These rules can then be applied to new data to produce original answers (see Figure 2).

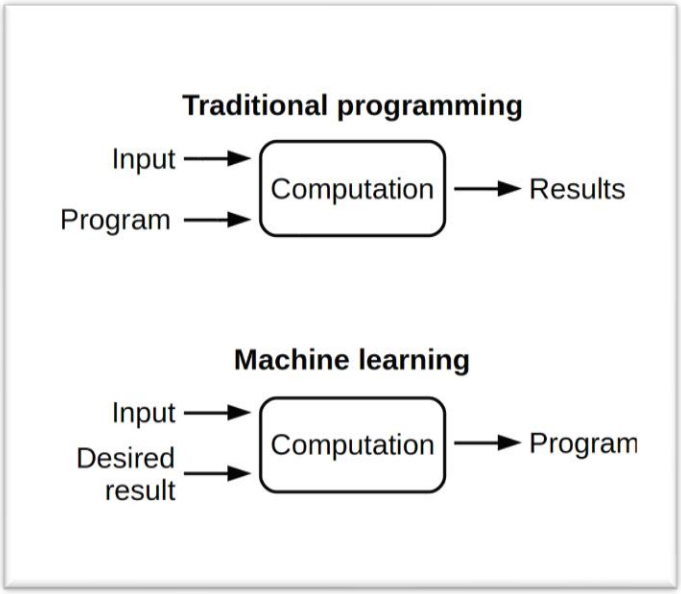


Figure 2: Traditional programming vs machine learning

1.2.1 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:

1.2.1.1 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.

1.2.1.2 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. 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.

1.2.1.3 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.

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.

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. 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 DeepMind has created a reinforcement learning program that plays old Atari's video games. When watching

the video, notice how the program is initially clumsy and unskilled but steadily improves with training until it becomes a champion.

1.2.1.4 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.

1.2.2 CATEGORIZING ON THE BASIS OF REQUIRED OUTPUT

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

1.2.2.1 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”.

1.2.2.2 REGRESSION:

Which is also a supervised problem, A case when the outputs are continuous rather than discrete.

1.2.2.3 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.

III. DEEP LEARNING

Deep learning is a branch of machine learning which is completely based on artificial neural network as neural network is going to mimic the human brain so deep learning is also a kind of mimic of human brain. In deep learning, we don’t need to explicitly program everything. The concept of deep learning is not new. It has been around for a couple of years now. It’s on hype nowadays because earlier we did not have that much processing power and a lot of data. As in the last 20 years, the processing power increases exponentially, deep learning and machine learning

came in the picture.

A formal definition of deep learning is- neurons.

"Deep learning is a particular kind of machine learning that achieves great power and flexibility by learning to represent the world as a nested hierarchy of concepts, with each concept defined in relation to simpler concepts, and more abstract representations computed in terms of less abstract ones."

1.3.1 THE NEURON

The foundational unit of the human brain is the neuron. A tiny piece of the brain, about the size of grain of rice, contains over 10,000 neurons, each of which forms an average of 6,000 connections with other neurons, it's this massive biological network that enables us to experience the world around us.

At its core, the neuron is optimized to receive information from other neurons, process this information in a unique way, and send its result to other cells. This process is summarized in Figure 3.

The neuron receives its inputs along antennae-like structures called dendrites. Each of these incoming connections is dynamically strengthened or weakened based on how often it is used (this is how we learn new concepts!), and it's the strength of each connection that determines the contribution of the input to the neuron's output. After being weighted by the strength of their respective connections, the inputs are summed together in the cell body. This sum is then transformed into a new signal that's propagated along the cell's axon and sent off to other neurons [3].

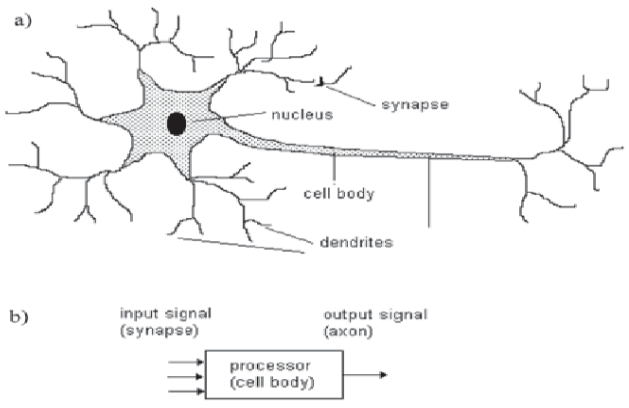


Figure 3: Structure of individual neuron with input and output process

1.3.1.1 ARTIFICIAL NEURON

The basic unit of computation in a neural net work is the neuron, often called a **node** or **unit**. It receives input from some other nodes, or from an external source and computes an output. Each input has an associated **weight** (w), which is assigned on the basis of its relative importance to other inputs. The node applies a function *f* (defined below) to the weighted sum of its inputs as shown in Figure 4 below:

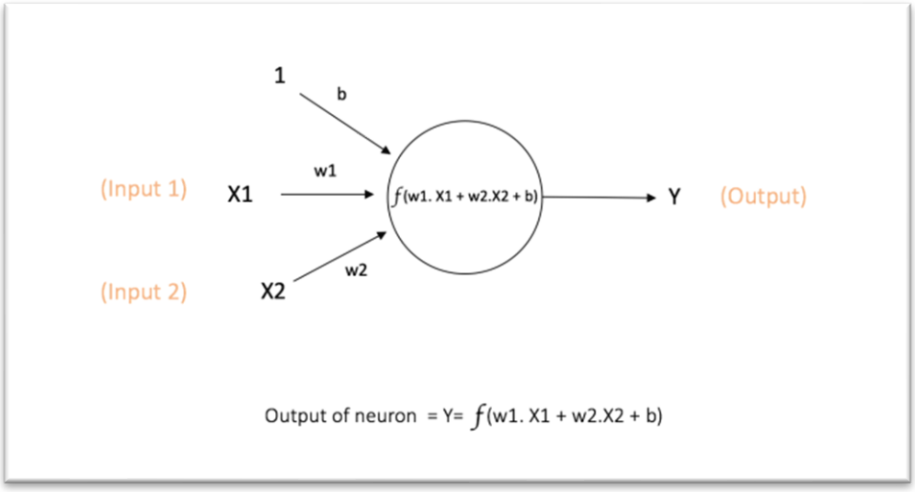


Figure 4: Artificial Neuron

The above network takes numerical inputs **X1** and **X2** and has weights **w1** and **w2** associated with those inputs. Additionally, there is another input **1** with weight **b** (called the **Bias**) associated with it. We will learn more details about role of the bias later.

The output **Y** from the neuron is computed as shown in the Figure 1. The function ***f*** is non-linear and is called the **Activation Function**. The purpose of the activation function is to introduce non-linearity into the output of a neuron. This is important because most real-world data are non linear and we want neurons to learn these non-linear representations.

1.3.1.2 ARTIFICIAL NEURAL NETWORK

Artificial neural network (ANN) is a popular machine learning algorithm that attempts to mimic how the human brain processes information (Rumelhart and McClelland, 1986). It provides a flexible way to handle regression and classification problems without the need to explicitly specify

any relationships between the input and output variables. Generally, neural networks are arranged in three layers: one input layer, one or more hidden layers, and one output layer—as shown in Figure 5 In this example, the inputs are estimated attributes from various well logs, and the output is an indicator referring to the specific lithofacies assigned to that depth. For a regression problem, the output could be a numerical value (e.g., corresponding log- or core-derived permeability).

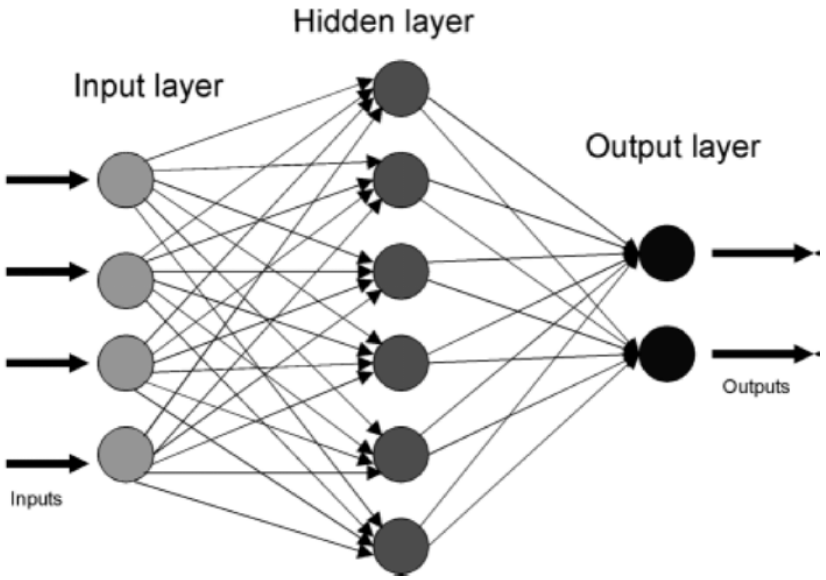


Figure 5: Schematic architecture of an ANN algorithm where input layers correspond to different well logs and output refers to different lithofacies

Artificial Neural Networks (ANN) are comprised of a large number of simple elements, called neurons, each of which makes simple decisions. Together, the neurons can provide accurate answers to some complex problems, such as natural language processing, computer vision, and AI.

A neural network can be “shallow”, meaning it has an input layer of neurons, only one “hidden layer” that processes the inputs, and an output layer that provides the final output of the model. A Deep Neural Network (DNN) commonly has between 2-8 additional layers of neurons. Research from Goodfellow, Bengio and Courville and other experts suggests that neural networks increase in accuracy with the number of hidden layers.

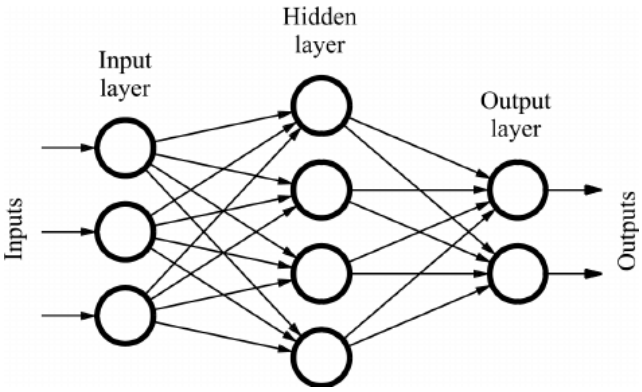


Figure 6: "Non-deep" feedforward neural network

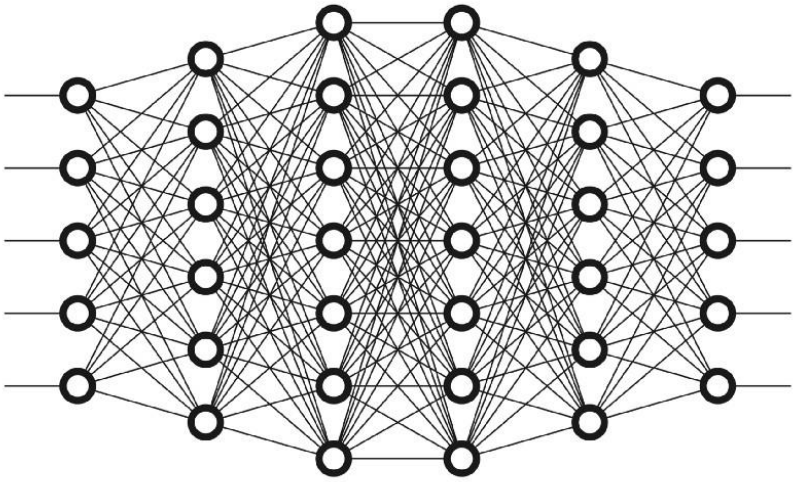


Figure 7: Deep neural network

1.3.1.3 FEEDFORWARD NEURAL NETWORK

The feedforward neural network was the first and simplest type of artificial neural network devised. It contains multiple neurons (nodes) arranged in layers. Nodes from adjacent layers have connections or edges between them. All these connections have weights associated with them.

An example of a feedforward neural network is shown in Figure 8.

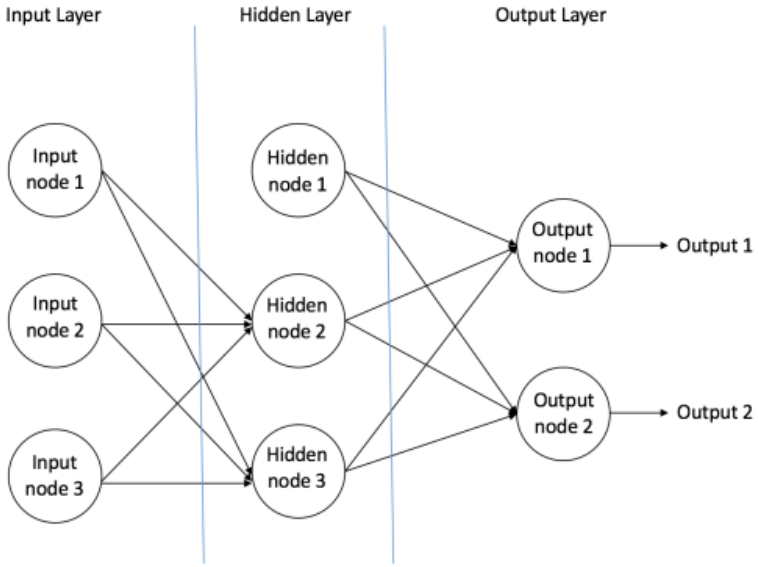


Figure 8: an example of feedforward neural network

A feedforward neural network can consist of three types of nodes:

A. INPUT NODES

The Input nodes provide information from the outside world to the network and are together referred to as the "Input Layer". No computation is performed in any of the Input nodes – they just pass on the information to the hidden nodes.

B. HIDDEN NODES

The Hidden nodes have no direct connection with the outside world (hence the name "hidden"). They perform computations and transfer information from the input nodes to the output nodes. A collection of hidden nodes forms a "Hidden Layer". While a feedforward network will only have a single input layer and a single output layer, it can have zero or multiple Hidden Layers.

C. OUTPUT NODES

The Output nodes are collectively referred to as the "Output Layer" and are responsible for computations and transferring information from the network to the outside world

In a feedforward network, the information moves in only one direction – forward – from the input nodes, through the hidden nodes (if any) and to the output nodes. There are no cycles or loops in the network.

Two examples of feedforward networks are given below:

1. SINGLE LAYER PERCEPTRON

This is the simplest feedforward neural network and does not contain any hidden layer. You can learn more about Single Layer perceptions.

2. MULTI LAYER PERCEPTRON

A Multi Layer Perceptron has one or more hidden layers. We will only discuss Multi Layer perceptron below since they are more useful than Single Layer perceptron for practical applications today.

1.3.2 ACTIVATION FUNCTIONS

Activation functions are mathematical equations that determine the output of a neural network. The function is attached to each neuron in the network, and determines whether it should be activated ("fired") or not, based on whether each neuron's input is relevant for the model's prediction. Activation functions also help normalize the output of each neuron to a range between 1 and 0 or between -1 and 1.

An additional aspect of activation functions is that they must be computationally efficient because they are calculated across thousands or even millions of neurons for each data sample. Modern neural networks use a technique called backpropagation to train the model, which places an increased computational strain on the activation function, and its derivative function

The activation function is a mathematical "gate" in between the input feeding the current neuron and its output going to the next layer. It can be as simple as a step function that turns the neuron

output on and off, depending on a rule or threshold. Or it can be a transformation that maps the input signals into output signals that are needed for the neural network to function.[Figure 9]

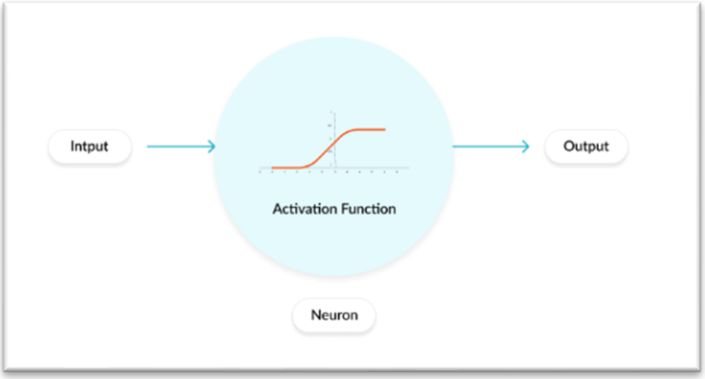


Figure 9: Activation Function

There are 3 Types of Activation Functions

1.3.2.1 BINARY STEP FUNCTION

A binary step function [Figure10] is a threshold-based activation function. If the input value is above or below a certain threshold, the neuron is activated and sends exactly the same signal to the next layer.

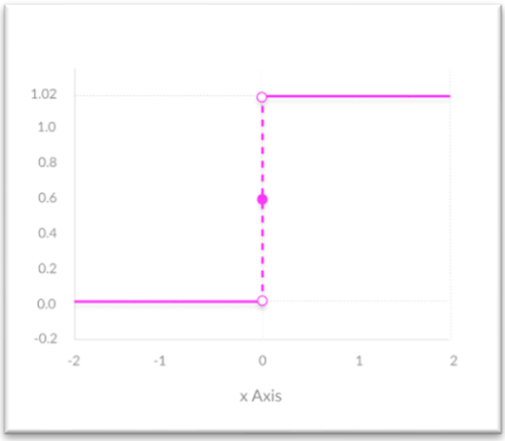


Figure 10: Binary Step Function

The problem with a step function is that it does not allow multi-value outputs—for example, it cannot support classifying the inputs into one of several categories.

1.3.2.2 LINEAR ACTIVATION FUNCTION

A linear activation [Figure 11] function takes the form: **A = cx**

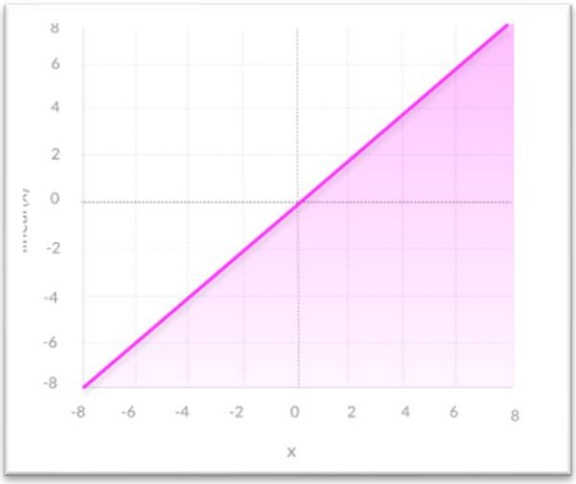


Figure 11: linear activation Function

It takes the inputs, multiplied by the weights for each neuron, and creates an output signal proportional to the input. In one sense, a linear function is better than a step function because it allows multiple outputs, not just yes and no.

1.3.2.3 NON-LINEAR ACTIVATION FUNCTIONS

Modern neural network models use non-linear activation functions. They allow the model to create complex mappings between the network's inputs and outputs, which are essential for learning and modeling complex data, such as images, video, audio, and data sets which are non-linear or have high dimensionality.

Almost any process imaginable can be represented as a functional computation in a neural network, provided that the activation function is non-linear.

Non-linear functions address the problems of a linear activation function:

They allow backpropagation because they have a derivative function which is related to the inputs.

They allow "stacking" of multiple layers of neurons to create a deep neural network. Multiple hidden layers of neurons are needed to learn complex data sets with high levels of accuracy.

1.3.2.3.1 Sigomid

Sigmoid neuron, which uses the function:

$$\mathbf{f(z) = 1/1+e^{-z} \quad (1)}$$

Intuitively, this means that when the logit is very small, the output of a logistic neuron is very close to 0. When the logit is very large, the output of the logistic neuron is close to 1. In-between these two extremes, the neuron assumes an S-shape, as shown in Figure 12

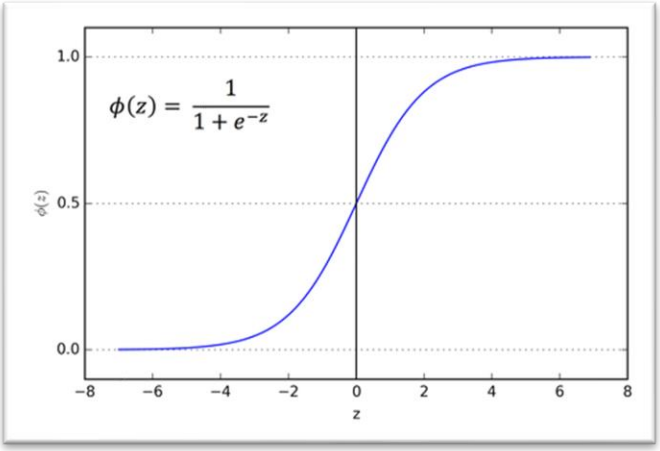


Figure 12: Sigmoid Activation Function

1.3.2.3.2 Tanh

neurons use a similar kind of S-shaped nonlinearity, but instead of ranging from 0 to 1, the output of tanh neurons range from −1 to 1, they use f (z)=tanh(z) , tanh represents the ratio of the hyperbolic sine to the hyperbolic cosine:

$$\mathbf{\tanh(x) = \sinh(x) / \cosh(x)}$$

The resulting relationship between the output y and the logit z is described by Figure 13 When S-shaped nonlinearities are used, the tanh neuron is often preferred over the sigmoid neuron because it is zero-centered

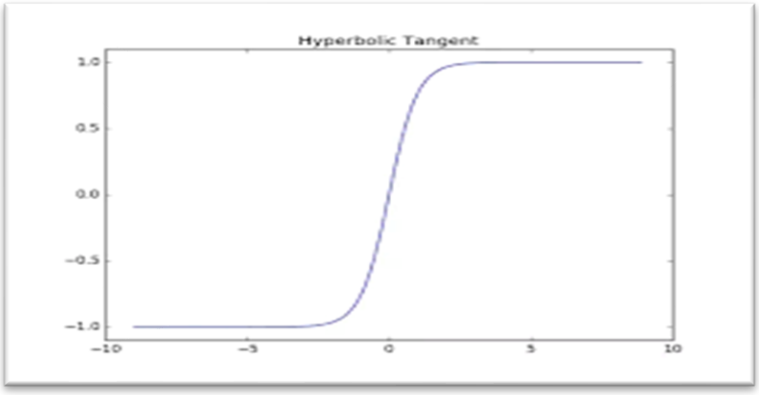


Figure 13: Hyperbolic Tangent

1.3.2.3.3 ReLU

a different kind of nonlinearity is used by the restricted linear unit neuron. It uses the function

$$\mathbf{f(z)=max(0,z)}$$

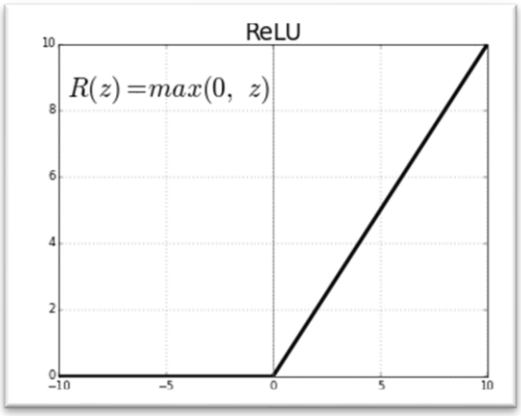


Figure 14 : Relu Activation Function

The ReLU [Figure 14] has recently become the neuron of choice for many tasks (especially in computer vision) for a number of reasons, one of it as strategies to combat the potential pitfalls , because the gradient of a ReLU is either zero or a constant, it is possible to reign in the vanishing exploding gradient issue. ReLU activation functions have shown to train better in practice than sigmoid activation functions.

1.3.2.3.4 Softmax

Softmax is a generalization of logistic regression inasmuch as it can be applied to continuous data (rather than classifying binary) and can contain multiple decision boundaries. It handles multinomial labeling systems. Softmax is the function often find at the output layer of a classifier.(see Figure15)

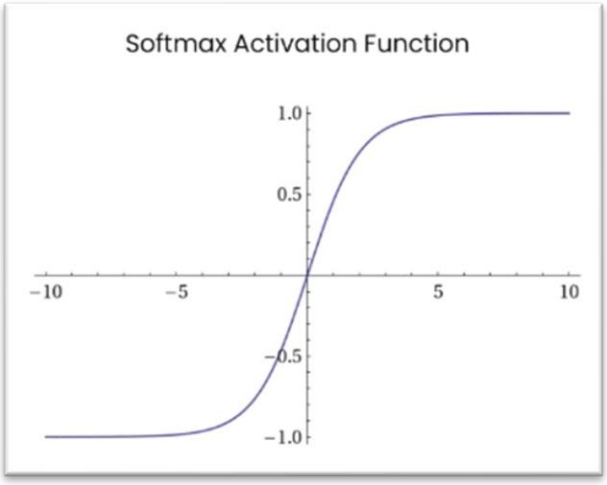


Figure 15: Softmax Activation Function

1.3.3 LOSS FUNCTIONS

Loss functions quantify how close a given neural network is to the ideal toward which it is training. The idea is simple. We calculate a metric based on the error we observe in the network's predictions. We then aggregate these errors over the entire dataset and average them and now we have a single number representative of how close the neural network is to its ideal.

Looking for this ideal state is equivalent to finding the parameters (weights and biases) that will minimize the "loss" incurred from the errors. In this way, loss functions help re-frame training neural networks as an optimization problem.

We want to train the neuron so that we pick the optimal weights possible the weights that minimize the errors we make on the training examples.

In most cases, these parameters cannot be solved for analytically, but, more often than not, they can be approximated well with iterative optimization algorithms like gradient descent.

In this case, let's say we want to minimize the square error over all of the training examples that we encounter. More formally, if we know that t (i) is the true answer for the i (th) training example and y (i) is the value computed by the neural network, we want to minimize the value of the error function E:

$$E=\frac{1}{2}\sum_{i}\left(t^{(i)}-y^{(i)}\right)^2$$

IV. GRADIENT DESCENT

Let's visualize how we might minimize the squared error over all of the training examples by simplifying the problem. Let's say our linear neuron only has two inputs (and thus only two weights, w 1 and w 2). Then we can imagine a three-dimensional space where the horizontal dimensions correspond to the weights w 1 and w 2 , and the vertical dimension corresponds to the value of the error function E . In this space, points in the horizontal plane correspond to different settings of the weights, and the height at those points corresponds to the incurred error. If we consider the errors we make over all possible weights, we get a surface in this three-dimensional space, in particular, a quadratic bowl as shown in Figure 16.

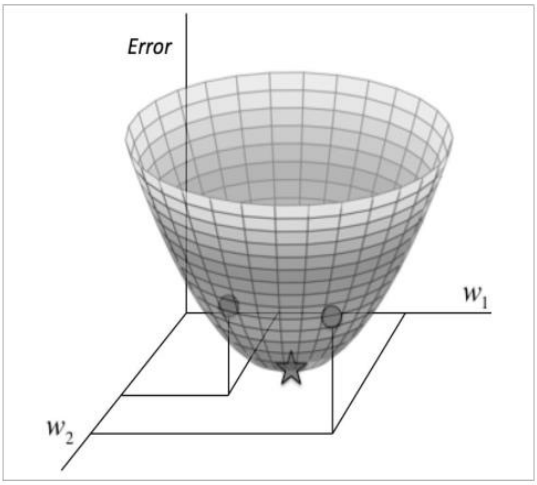


Figure 16: The quadratic error surface for a linear neuron

We can also conveniently visualize this surface as a set of elliptical contours, where the minimum error is at the center of the ellipses. In this setup, we are working in a two-dimensional plane where the dimensions correspond to the two weights. Contours correspond to settings of w 1 and w 2 that

evaluate to the same value of E . The closer the contours are to each other, the steeper the slope. In fact, it turns out that the direction of the steepest descent is always perpendicular to the contours. This direction is expressed as a vector known as the gradient.

Now we can develop a high-level strategy for how to find the values of the weights that minimizes the error function. Suppose we randomly initialize the weights of our network so we find ourselves somewhere on the horizontal plane. By evaluating the gradient at our current position, we can find the direction of steepest descent, and we can take a step in that direction. Then we'll find ourselves at a new position that's closer to the minimum than we were before. We can reevaluate the direction of steepest descent by taking the gradient at this new position and taking a step in this new direction. It's easy to see that, as shown in Figure 17, following this strategy will eventually get us to the point of minimum error. This algorithm is known as gradient descent, and we'll use it to tackle the problem of training individual neurons and the more general challenge of training entire networks [4].

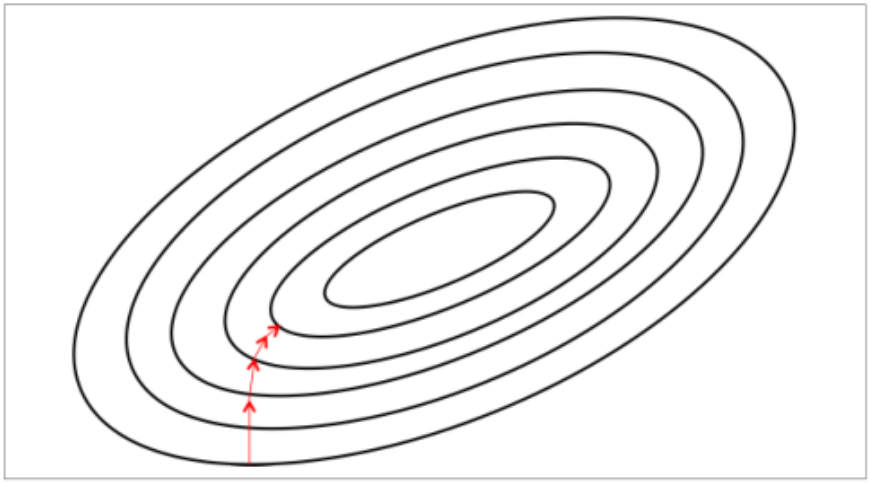


Figure 17: Visualizing the error surface as a set of contours

V. BACKPROPAGATION

A neural network propagates the signal of the input data forward through its parameters towards the moment of decision, and then backpropagates information about the error, in reverse through the network, so that it can alter the parameters. This happens step by step:

- The network makes a guess about data, using its parameters

- The network's is measured with a loss function
- The error is backpropagated to adjust the wrong-headed parameters

You could compare a neural network to a large piece of artillery that is attempting to strike a distant object with a shell. When the neural network makes a guess about an instance of data, it fires, a cloud of dust rises on the horizon, and the gunner tries to make out where the shell struck, and how far it was from the target. That distance from the target is the measure of error. The measure of error is then applied to the angle and direction of the gun (parameters), before it takes another shot.

Backpropagation takes the error associated with a wrong guess by a neural network, and uses that error to adjust the neural network's parameters in the direction of less error. How does it know the direction of less error?

VI. CONCLUSION

In this chapter, we discussed the general concept of machine learning, especially deep learning, as well as basic concepts such as neural network, loss function, and gradient regression.

In the next chapter, we'll explain well the convolutional neural network (CNN) and its relationship to image recognition as well as image classification, and we talk about the architecture used in this research like ResNet20 model.

CHAPITRE 2

CONVOLUTIONAL NEURAL NETWORK

I. INTRODUCTION

In this chapter we show what is the convolutional neural network and how classify images with it. we show many techniques for optimization of our model like data augmentation. we discuss about filters and polling operations. also, we show how many architectures like ResNet20 [5].

II. CONVOLUTIONAL NEURAL NETWORKS (CNNS / CONVNETS)

Convolutional Neural Network also known as ConvNet or CNN [6] are inspired by the biological visual cortex. The visual cortex has small regions of cells that are sensitive to specific regions of the visual field. Different neurons in the brain respond to different features. For example, certain neurons fire only in the presence of lines of a certain orientation, some neurons fire when exposed to vertical edges and some when shown horizontal or diagonal edges, this idea of certain neurons having a specific task is the basis behind ConvNets.

ConvNets have shown excellent performance on several applications such as image classification, object detection, speech recognition, natural language processing, and medical image analysis. Convolutional neural networks are powering core of computer vision that has many applications which include self-driving cars, robotics, and treatments for the visually impaired. The main concept of ConvNets is to obtain local features from input (usually an image) at higher layers and combine them into more complex features at the lower layers.

2.1.1 CONVOLUTION OPERATION

Convolution is a mathematical operation performed on two functions and is written as (f * g), where f and g are two functions.

The output of the convolution operation for domain n is defined as

$$(f*g)(n)=\sum_{m} f(m)g(n-m)$$

For time-domain functions, n is replaced by t. The convolution operation is commutative in nature, so it can also be written as

$$(f*g)(n)=\sum_m f(n-m)\,g(m)$$

Convolution operation is one of the important operations used in digital signal processing and is used in many areas which includes statistics, probability, natural language processing, computer vision, and image processing and can be applied to higher dimensional functions as well.

It can be applied to a two-dimensional function by sliding one function on top of another, multiplying and adding. Convolution operation can be applied to images (treated as two-dimensional functions) to perform various transformations.

An example of a two-dimensional filter, a two-dimensional input, and a two-dimensional feature map is shown in Figure 18 Let the 2D input (i.e., 2D image) be denoted by A, the 2D filter of size m × n be denoted by K, and the 2D feature map be denoted by F. Here, the image A is convolved with the filter K and produces the feature map F. This convolution operation is denoted by A*K and is mathematically given as

$$f(i,j)=(A*k)(i,j)=\sum_m\sum_n A(m,n)k(i-m,j-n)$$

The convolution operation [Figure 1] is commutative in nature, so we can write

$$f(i,j)=(A*k)(i,j)=\sum_m\sum_n A(i-m,j-n)k(m,n)$$

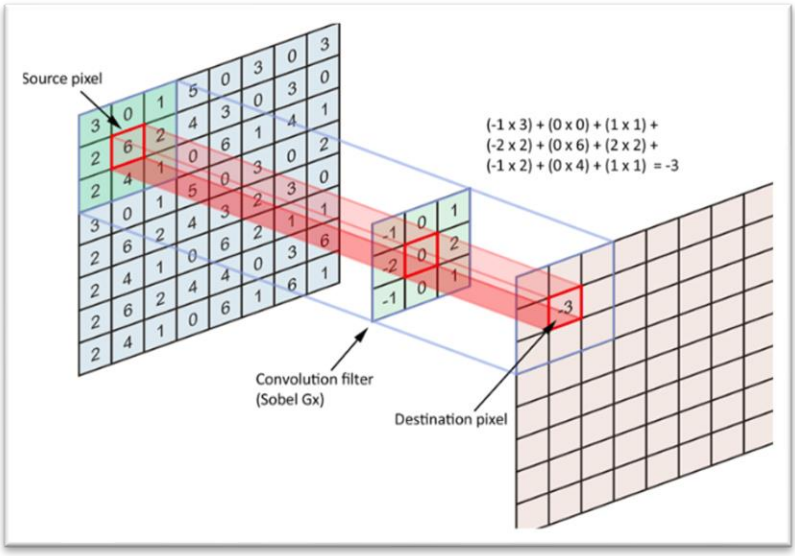


Figure 18: Convolutional Operation

The kernel K is flipped relative to the input. If the kernel is not flipped, then convolution operation will be same as cross-correlation operation that is given below:

$$f(i,j)=(A*k)(i,j)=\sum_m \sum_n A(i+m,j+n)k(m,n)$$

Many CNN libraries use cross-correlation function as convolution function because cross correlation is more convenient to implement than convolution operation itself.

2.1.2 ARCHITECTURE OF CNN

In a traditional neural network, each hidden layer is made up of a number of neurons, where each neuron is fully connected to all neurons in the preceding layer. The problem with the fully connected neural network is that its densely connected network architecture does not scale well to large images. For large images, the most preferred approach is to use convolutional neural network.

Convolutional neural network is a deep neural network architecture designed to process data that has a known, grid-like topology, for example, 1D time-series data, 2D or 3D data such as images and speech signal, and 4D data such as videos.

ConvNets have three key features: local receptive field, weight sharing, and subsampling (pooling) [7].

2.1.2.1 LOCAL RECEPTIVE FIELD

In a traditional neural network, each neuron or hidden unit is connected to every neuron in previous layer or every input unit. Convolutional neural networks, however have local receptive field architecture, each hidden unit can only connect to a small region of the input called local receptive field. This is accomplished by making the filter/weight matrix smaller than the input.

With local receptive field, neurons can extract elementary visual features like edges, corners, end points, etc.

2.1.2.2 WEIGHT SHARING

Weight sharing refers to using the same filter/weights for all receptive fields in a layer. In ConvNet, since the filters are smaller than the input, each filter is applied at every position of the input, same filter is used for all local receptive fields.

2.1.2.3 SUBSAMPLING (POOLING)

Subsampling reduces the spatial size of the input, thus reducing the parameters in the network.

There are few subsampling techniques available, and the most common subsampling technique is max-pooling.

ConvNet consists of a sequence of different types of layers [Figure 19] to achieve different tasks, a typical convolutional neural network consists of the following layers:

- Convolutional layer,
- Activation function layer (ReLU),
- Pooling layer
- Fully connected layer

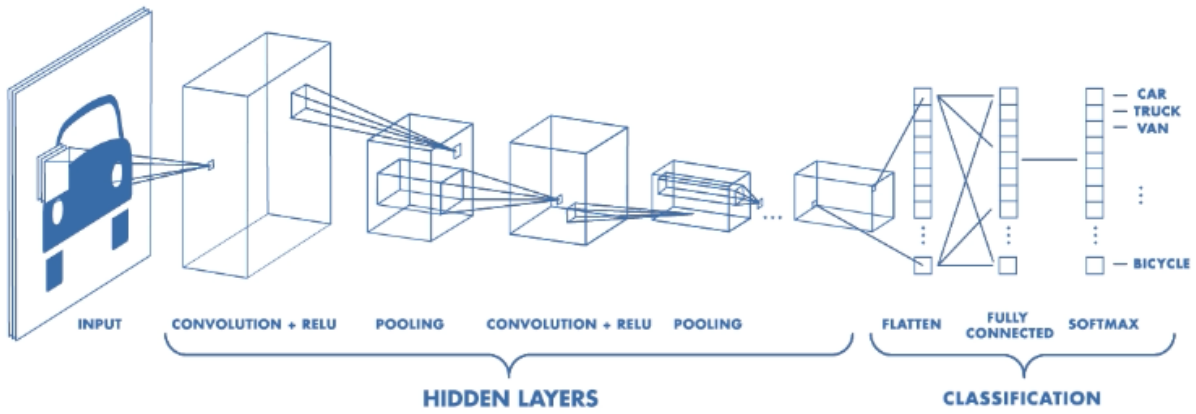


Figure 19: Convolutional Neural Network with Different Layer

These layers are stacked up to make a full ConvNet architecture. Convolutional and activation function layers are usually stacked together followed by an optional pooling layer. Fully connected layer makes up the last layer of the network, and the output of the last fully connected layer produces the class scores of the input image as showing in figure 19.

In addition to these main layers mentioned above, ConvNet may include optional layers like batch normalization layer to improve the training time and dropout layer to address the overfitting issue.

2.1.3 CONVOLUTION LAYER

Convolution layer is the core building block of a convolutional neural network which uses convolution operation (represented by *) in place of general matrix multiplication. Its parameters consist of a set of learnable filters also known as kernels. The main task of the convolutional layer is to detect features found within local regions of the input image that are common throughout the dataset and mapping their appearance to a feature map.

A feature map is obtained for each filter in the layer by repeated application of the filter across subregions of the complete image, convolving the filter with the input image, adding a bias term, and then applying an activation function.

The input area on which a filter is applied is called local receptive field.

The size of the receptive field is same as the size of the filter, (Figure 3) shows how a filter (T-shaped) is convolved with the input to get the feature map.

Feature map is obtained after adding a bias term and then applying a nonlinear function to the output of the convolution operation. The purpose of nonlinearity function is to introduce nonlinearity in the ConvNet model.

2.1.3.1 FILTERS/KERNELS

The weights in each convolutional layer specify the convolution filters and there may be multiple filters in each convolutional layer. Every filter contains some feature like edge, corner, etc. and during forward pass, each filter is slid across the width and height of the input generating feature map of that filter(figure 20).

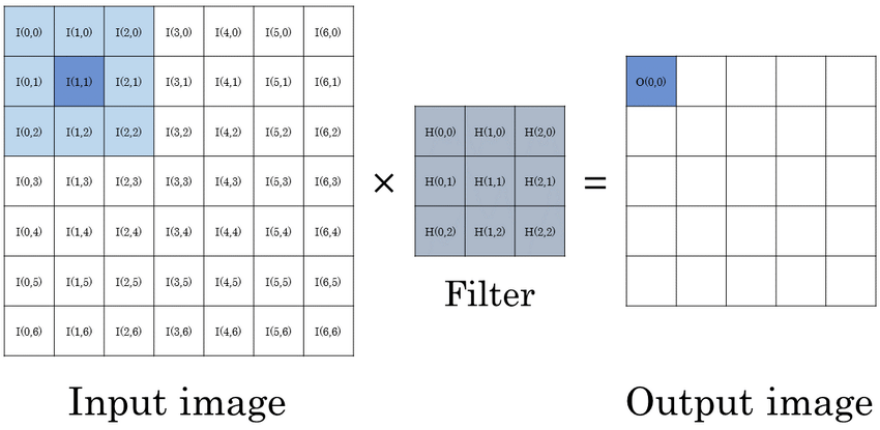


Figure 20: Example of convolution operation

2.1.3.2 HYPERPARAMETERS

Convolutional neural network architecture has many hyperparameters that are used to control the behavior of the model. Some of these hyperparameters control the size of the output while some are used to tune the running time and memory cost of the model, the four important hyperparameters in the convolution layer of the ConvNet are given below:

2.1.3.2.1 Filter Size:

Filters can be of any size greater than 2 × 2 and less than the size of the input but the conventional size varies from 11 × 11 to 3 × 3. The size of a filter is independent of the size of input.

2.1.3.2.2 Number of Filters:

There can be any reasonable number of filters. AlexNet used 96 filters of size 11 × 11 in the first convolution layer. VGGNet used 96 filters of size 7 × 7, and another variant of VGGNet used 64 filters of size 11 × 11 in first convolution layer.

2.1.3.2.3 Stride:

It is the number of pixels to move at a time to define the local receptive field for a filter. Stride of one means to move across and down a single pixel. The value of stride should not be too small or too large. Too small stride will lead to heavily overlapping receptive fields and too large value will overlap less and the resulting output volume will have smaller dimensions spatially.

2.1.3.2.4 Zero Padding:

This hyperparameter describes the number of pixels to pad the input image with zeros. Zero padding is used to control the spatial size of the output volume.

Each filter in the convolution layer produces a feature map of size ([A − K + 2P]/S)+ 1 where A is the input volume size, K is the size of the filter, P is the number of padding applied and S is the stride.

Suppose the input image has size 128 × 128, and 5 filters of size 5 × 5 are applied, with single stride and zero padding, i.e., A # 128, F # 5, P # 0 and S # 1. The number of feature maps produced will be equal to the number of filters applied, i.e., 5 and the size of each feature map will be ([128 − 5 + 0]/1) + 1 # 124. Therefore, the output volume will be 124 × 124 × 5.

III. POOLING LAYER

In ConvNets, the sequence of convolution layer and activation function layer is followed by an optional pooling or down-sampling layer to reduce the spatial size of the input and thus reducing the number of parameters in the network.

A pooling layer takes each feature map output from the convolutional layer and down-samples it, pooling layer summarizes a region of neurons in the convolution layer. There are few pooling techniques available and the most common pooling technique is max-pooling.

Max-pooling simply outputs the maximum value in the input region. [figure 21]

The input region is a subset of input (usually 2 × 2). For example, if input region is of size 2 × 2, the max-pooling unit will output the maximum of the four values as shown in Figure 21 Other options for pooling layers are average pooling and L2-norm pooling.

Pooling layer operation discards less significant data but preserves the detected features in a smaller representation. The intuitive reasoning behind pooling operation is that feature detection is more important than feature's exact location. This strategy works well for simple and basic problems but it has its own limitations and does not work well for some problems.

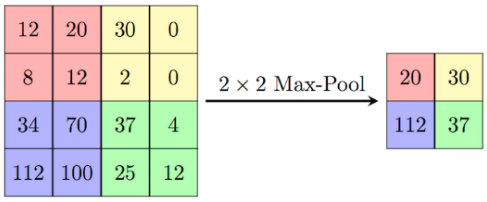


Figure 21: Example of Max Polling Operation

IV. REGULARIZATION FOR DEEP LEARNING

"A central problem in machine learning especially in deep learning is how to make an algorithm that will perform well not just on the training data, but also on new inputs. Many strategies used in machine learning are explicitly designed to reduce the test error, possibly at the expense of increased training error. These strategies are known collectively as regularization." [8]

Regularization is often done by putting some extra constraints on a machine learning model, such as adding restrictions on the parameter values or by adding extra terms in the objective function (penalizes the weight matrices) that can be thought of as corresponding to a soft constraint on the parameter values. If chosen correctly these can lead to a reduced testing error.

An effective regularization is said to be the one that makes a profitable trade by reducing variance significantly while not overly increasing the bias.

Regularization helps us control our model capacity, ensuring that our models are better at making (correct) classifications on data points that they were not trained on, which we call the ability to generalize. If we don't apply regularization, our classifiers can easily become too complex and overfit to our training data, in which case we lose the ability to generalize to our testing data.

However, too much regularization can be a bad thing. We can run the risk of underfitting, in which case our model performs poorly on the training data and is not able to model the relationship between the input data and output class labels (because we limited model capacity too much).

Our goal when building deep learning classifiers is to obtain a model that fit our training data nicely, but avoid overfitting. Regularization can help us obtain this type of desired fit (Figure 22)

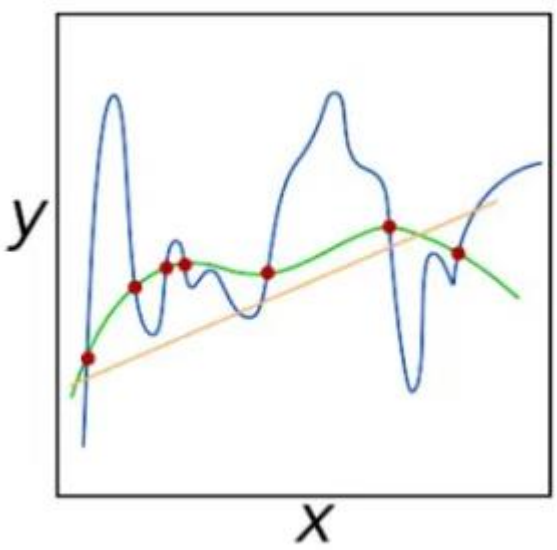


Figure 22: example of underfitting (orange line) , overfitting (blue line) , and generalizing (green line)

Overfitting and underfitting are detected by looking to the training and test error behave (Figure 23) At the left end of the graph, training error and generalization error are both high. This is the underfitting regime As we increase capacity, training error decreases, but the gap between training and generalization error increases. Eventually, the size of this gap outweighs the decrease in training error, and we enter the overfitting regime, where capacity is too large, above the optimal capacity.

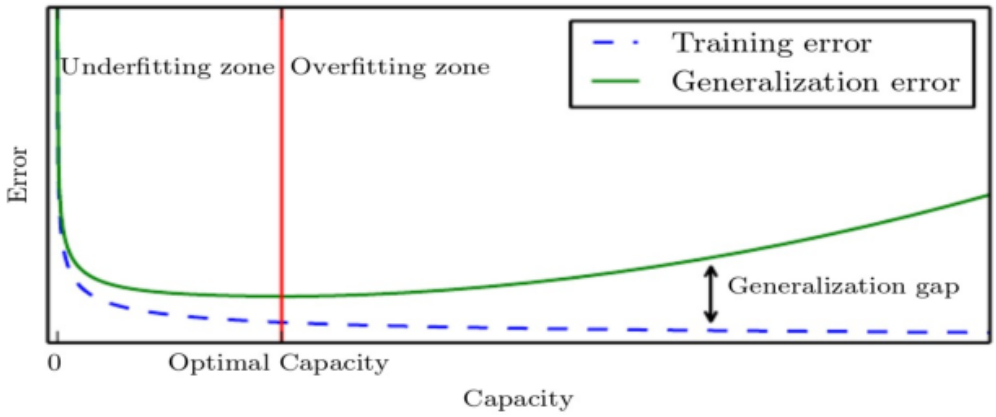


Figure 23: Typical relationship between capacity and error

V. DATA-SET AUGMENTATION

The best way to make a machine learning model generalize better is to train it on more data. Of course, in practice, the amount of data we have is limited. One way to get around this problem is to create fake data and add it to the training set.

Dataset augmentation is a common practice to virtually increase the size of training dataset, and is also used as a regularization technique, making the model more robust to slight changes in the input data has been a particularly effective technique for a specific classification problem: object f recognition. Images are high dimensional and include an enormous range of factors of variation, many of which can be easily simulated.

Operations like translating the training images a few pixels in each direction can often greatly improve generalization, even if the model has already been designed to be partially translation invariant by using the convolution and pooling techniques.

Many other operations, such as rotating the image or scaling the image, have also proved quite effective (see Figure 24).

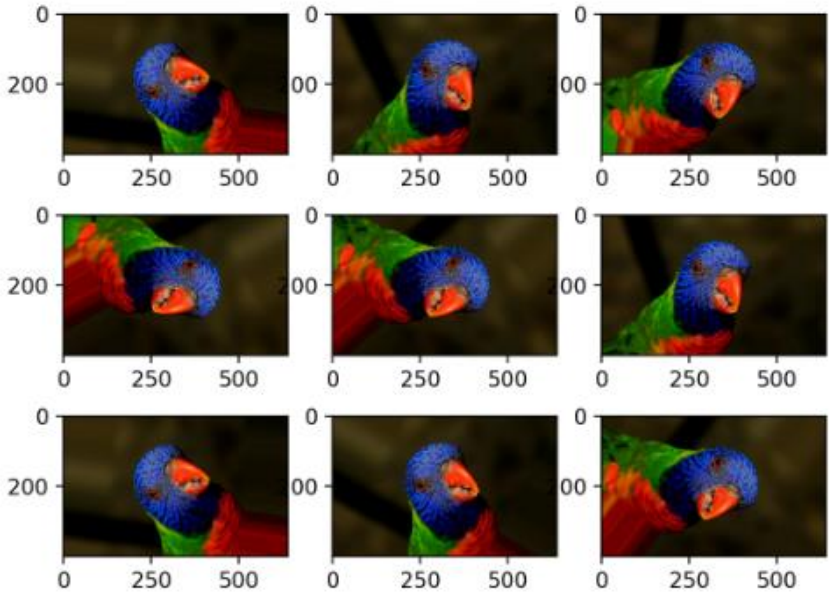


Figure 24: Images Generated With a Random Rotation Augmentation

One must be careful not to apply transformations that would change the correct class. For example, optical character recognition tasks require recognizing the difference between “b” and f “d” and the difference between “6” and “9,” so horizontal flips and 180° rotations are not f appropriate ways of augmenting datasets for these tasks [9].

2.5.1 DROPOUT [10]

Dropout is an effective way of regularizing neural networks to avoid the overfitting of ANN. During training, the dropout layer cripples the neural network by removing hidden units stochastically as shown in the following image

(figure 25,26).

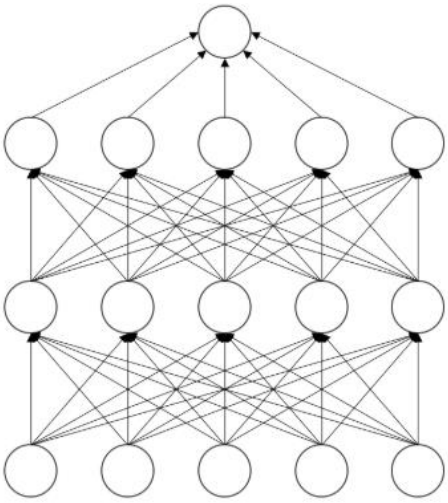


Figure 25: Standard Neural network

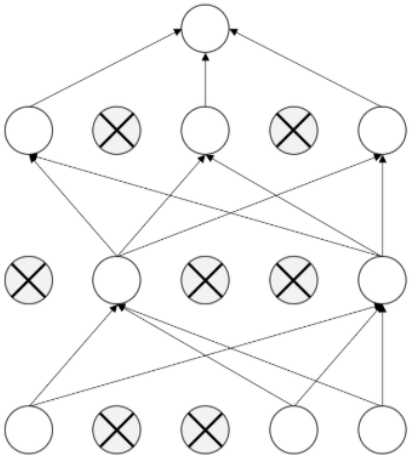


Figure 26: After Applying Dropout

Note how the neurons are randomly trained. Dropout is also an efficient way of combining several neural networks. For each training case, we randomly select a few hidden units so that we end up with different architectures for each case. This is an extreme case of bagging and model averaging. Dropout layer should not be used during the inference as it is not necessary.

VI. DEEP LEARNING CONVOLUTIONAL NEURAL NETWORKS ARCHITECTURES

Many supervised deep learning architectures have evolved over the last few years, achieving top scores on many tasks. In this paragraph we will discussed briefly one of the most recent supervised CNN architectures proposed by researchers ResNet.

2.6.1 RESNET

As the number of layers of deep networks increases, its accuracy improves and the accuracy saturates once the network has converged. However, if the depth is further increased, then the performance starts getting degraded rapidly. This degradation is caused by adding more layers to an already converged deep model which results in higher training error. Thus, there is a need for a strategy that obtains an optimal deep network for a given application.

ResNet [5] was proposed with a residual learning framework that lets new layers to fit a residual mapping. It is easier to push the residual to zero when a model has converged than to fit the mapping by a stack of nonlinear layers.

Given an underlying mapping H(x) to be fit by a few stacked layers, where x is the input to these layers, the residual learning uses the residual function.

$$F(x)=H(x)-x$$

It is easier to optimize the residual mapping than to optimize the original, and it can be realized by a feedforward neural network with shortcut connection as shown in (Figure 27) The shortcut link simply accomplishes identity mapping, and the output of the shortcut link is added to the outcomes of the stacked layers.

The identity shortcut link does not add calculation complexity or parameters.

The residual function F uses a stack of 2 or 3 layers (more layers are also possible) as shown in (Figure 27), the building block is defined by

$$y=F(x,\{W_i\})+x$$

where x and y represent the input and output vectors of layers considered.

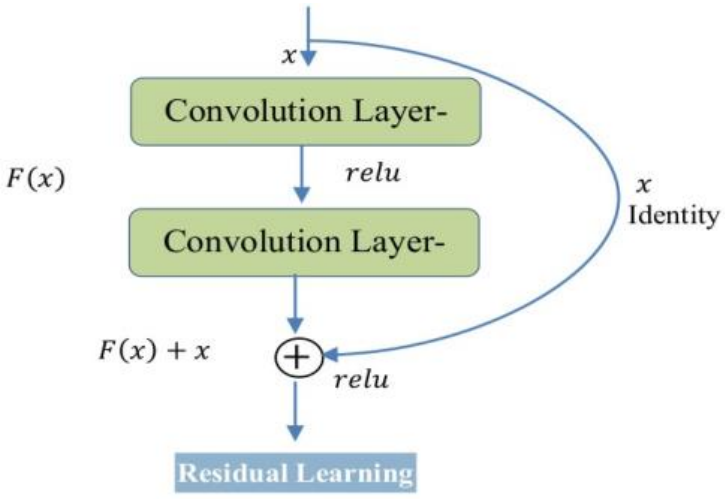


Figure 27: ResNet residual learning building block

The function F(x, {W i }) represents the residual mapping which is to be learnt. The linear projection W s is performed by a shortcut link to match the dimensions as in figure 28.

$$y = F(x, \{W_i\}) + W_s x$$

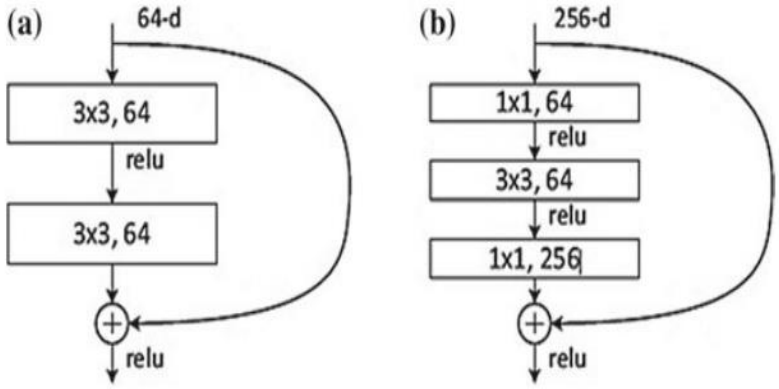


Figure 28: a ResNet building block, b "Bottleneck" ResNet building block

The architecture diagram of ResNet-20 is shown in (Figure 29)

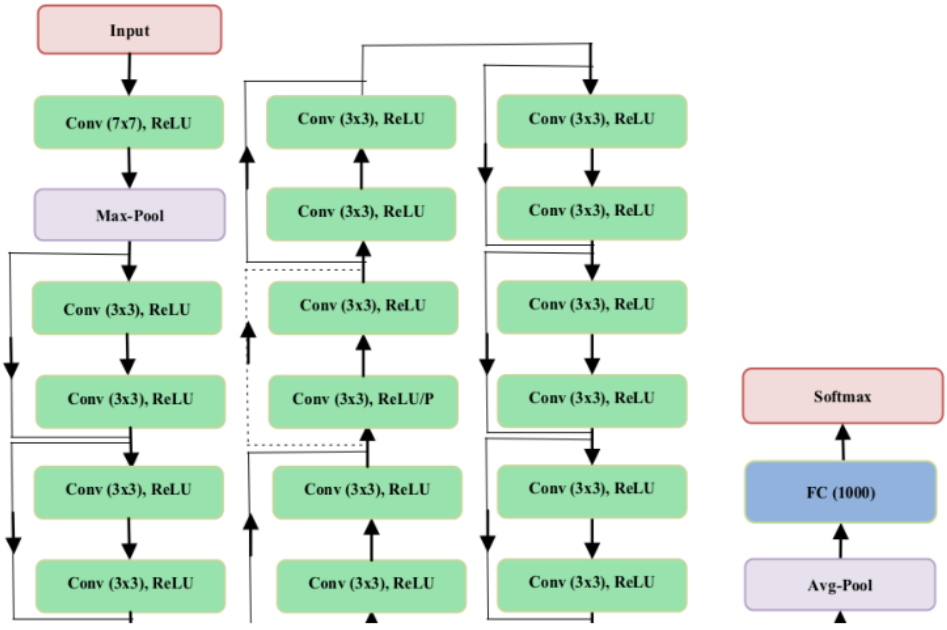


Figure 29: Architecture diagram of ResNet-34 layers

VII. CONCLUSION

In this chapter, we show the general concept of the convolutional neural network, in addition to basic concepts such as the convolutional operation and data augmentation technique, and we present the ResNet architecture present in our research.

In the next chapter, we will well explain the color spaces and how to convert to different other color space such as HSV, LUV, LAB and YUV.

CHAPTER 3

COLOR SPACES

I. INTRODUCTION

In this chapter we will review the color spaces used in our research and we show how to convert from RGB color space to other like HSV, LUV, LAB, YUV.

We'll also show some basic concepts related to color spaces.

II. COLOR SPACES AND COLOR SPACE CONVERSION

The Image Processing Toolbox software typically represents colors as red, green, and blue (RGB) numeric values. However, there are other models besides RGB for representing colors numerically. The various models are referred to as *color spaces* because most of them can be mapped into a 2-D, 3-D, or 4-D coordinate system.

The various color spaces exist because they present color information in ways that make certain calculations more convenient or because they provide a way to identify colors that is more intuitive. For example, the RGB color space defines a color as the percentages of red, green, and blue hues mixed together. Other color models describe colors by their hue (shade of color), saturation (amount of gray or pure color), and luminance (intensity, or overall brightness).

3.2.1 RGB COLOR SPACE

The RGB color space [11] represents images as an m-by-n-by-3 numeric array whose elements specify the intensity values of the red, green, and blue color channels. The range of numeric values depends on the data type of the image.

For single or double arrays, RGB values range from [0, 1].

For uint8 arrays, RGB values range from [0, 255].

For uint16 arrays, RGB values range from [0, 65535].

The toolbox supports variations of the RGB color space.

RGB Color Space	Description
Linear RGB	Linear RGB values are raw data obtained from a camera sensor. The value of R, G, and B are directly proportional to the amount of light that illuminates the sensor. Preprocessing of raw image data, such as white balance, color balance, and chromatic aberration compensation, are performed on linear RGB values.
sRGB	sRGB values apply a nonlinear function, called gamma correction, to linear RGB values. Images are frequently displayed in the sRGB color space because they appear brighter and colors are easier to distinguish. The parametric curve used to transform linear RGB values to the sRGB color space is: $f(u) = -f(-u), \quad u < 0$ $f(u) = c \cdot u, \quad 0 \le u < d$ $f(u) = a \cdot u^{\gamma} + b, \quad u \ge d,$ where u represents one of the R, G, or B color values with these parameters: $a = 1.055$ $b = -0.055$ $c = 12.92$ $d = 0.0031308$ $\gamma = 1/2.4$

Adobe RGB (1998)	Adobe RGB (1998) RGB values apply gamma correction to linear RGB values using a simple power function: $v = u^{\gamma}, \quad u \geq 0$ $v = -(-u)^{\gamma}, \quad u < 0,$ with $\gamma = 1/2\ 19921875$

Table 1: The toolbox supports variations of the RGB color space

3.2.2 HSV COLOR SPACE

The HSV (Hue, Saturation, Value) [11] color space corresponds better to how people experience color than the RGB color space does. For example, this color space is often used by people who are selecting colors, such as paint or ink color, from a color wheel or palette.

Attribute	Description
H	Hue, which corresponds to the color's position on a color wheel. *H* is in the range [0, 1]. As *H* increases, colors transition from red to orange, yellow, green, cyan, blue, magenta, and finally back to red. Both 0 and 1 indicate red.
S	Saturation, which is the amount of hue or departure from neutral. *S* is in the range [0, 1]. As *S* increases, colors vary from unsaturated (shades of gray) to fully saturated (no white component).
V	Value, which is the maximum value among the red, green, and blue components of a specific color. *V* is in the range [0, 1]. As *V* increases, the corresponding colors become increasingly brighter.

Table 2: description of HSV color space

We can show an Illustration of the HSV Color Space in the following figure (figure 30).

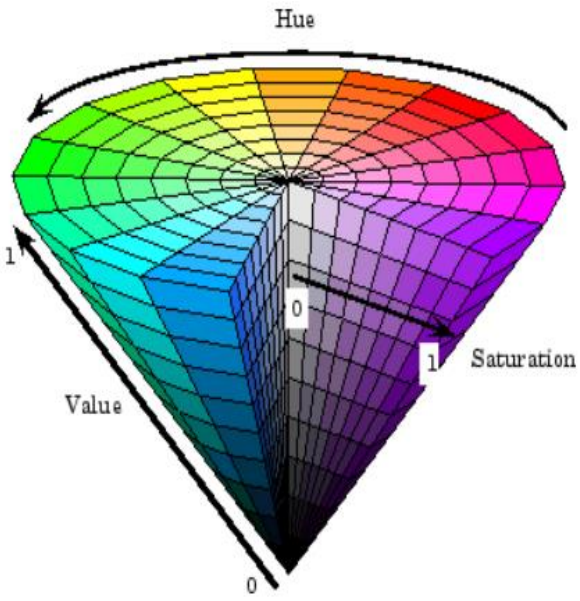


Figure 30: Illustration of the HSV Color Spase

3.2.3 COLOR SPACES YUV

YUV [12] is the basis for the color encoding used in analog television in both the North American NTSC and the European PAL systems. The luminance component Y is computed, just as in Eqn.

from the RGB components as

$$\mathbf{Y=0.299} \times \mathbf{R+0.587} \times \mathbf{G+0.114} \times \mathbf{B}$$

under the assumption that the RGB values have already been gamma corrected according to the TV encoding standard (γ NTSC = 2.2 and γ PAL = 2.8, see Sec. 4.7) for playback.

The UV components are computed from a weighted difference between the luminance and the

blue or red components as

$$\mathbf{U=0.492} \times \mathbf{(B-Y)} \text{ and } \mathbf{V=0.877} \times \mathbf{(R-Y)}$$

The Y′UV model defines a color space in terms of one luma component (Y′) and two chrominance components, called U (blue projection) and V (red projection) respectively. The Y′UV color model is used in the PAL composite color video (excluding PAL-N) standard. Previous black-and-white systems used only luma (Y′) information. Color information (U and V) was added separately via a subcarrier so that a black-and-white receiver would still be able to receive and display a color picture transmission in the receiver's native black-and-white format.

Y′ stands for the luma component (the brightness) and U and V are the chrominance (color) components; luminance is denoted by Y and luma by Y′ – the prime symbols (') denote gamma compression,[1] with "luminance" meaning physical linear-space brightness, while "luma" is (nonlinear) perceptual brightness.

The scope of the terms Y′UV, YUV, YCbCr, YPbPr, etc., is sometimes ambiguous and overlapping. Historically, the terms YUV and Y′UV were used for a specific *analog encoding* of color information in television systems, while YCbCr was used for *digital encoding* of color information suited for video *and* still-image compression and transmission such as MPEG and JPEG.[2] Today, the term YUV is commonly used in the computer industry to describe *file-formats* that are encoded using YCbCr.

The YPbPr color model used in analog component video and its digital version YCbCr used in digital video are more or less derived from it, and are sometimes called Y′UV. (C_B/P_B and C_R/P_R are deviations from grey on blue–yellow and red–cyan axes, whereas U and V are blue–luminance and red–luminance differences respectively.) The Y′IQ color space used in the analog NTSC television broadcasting system is related to it, although in a more complex way. The YDbDr color space used in the analog SECAM and PAL-N television broadcasting systems, are also related.

As for etymology, Y, Y′, U, and V are not abbreviations. The use of the letter Y for luminance can be traced back to the choice of XYZ primaries. This lends itself naturally to the usage of the same letter in luma (Y′), which approximates a perceptually uniform correlate of luminance. Likewise, U and V were chosen to differentiate the U and V axes from those in other spaces, such as the x and y chromaticity space. See the equations below or compare the historical development of the math.

A. CONVERSION TO/FROM RGB

a. SDTV with BT.601

Y′UV signals are typically created from RGB (red, green and blue) source. Weighted values of R, G, and B are summed to produce Y′, a measure of overall brightness or luminance. U and V are computed as scaled differences between Y′ and the B and R values.

BT.601 defines the following constants:

$$
\begin{aligned}
W_R &= 0.299, \\
W_G &= 1 - W_R - W_B = 0.587, \\
W_B &= 0.114, \\
U_{\max} &= 0.436, \\
V_{\max} &= 0.615.
\end{aligned}
$$

Y′UV is computed from RGB (linear RGB, not gamma corrected RGB or sRGB for example) as follows:

$$
\begin{aligned}
Y' &= W_R R + W_G G + W_B B = 0.299R + 0.587G + 0.114B, \\
U &= U_{\max} \frac{B - Y'}{1 - W_B} \approx 0.492(B - Y'), \\
V &= V_{\max} \frac{R - Y'}{1 - W_R} \approx 0.877(R - Y').
\end{aligned}
$$

The resulting ranges of Y′, U, and V respectively are [0, 1], [−Umax, Umax], and [−Vmax, Vmax].

Inverting the above transformation converts Y′UV to RGB:

$$
\begin{aligned}
R &= Y' + V \frac{1 - W_R}{V_{\max}} = Y' + \frac{V}{0.877} = Y' + 1.14V, \\
G &= Y' - U \frac{W_B(1 - W_B)}{U_{\max} W_G} - V \frac{W_R(1 - W_R)}{V_{\max} W_G} \\
&= Y' - \frac{0.232U}{0.587} - \frac{0.341V}{0.587} = Y' - 0.395U - 0.581V, \\
B &= Y' + U \frac{1 - W_B}{U_{\max}} = Y' + \frac{U}{0.492} = Y' + 2.033U.
\end{aligned}
$$

Equivalently, substituting values for the constants and expressing them as matrices gives these formulas for BT.601:

$$
\begin{aligned}
\begin{bmatrix} Y' \\ U \\ V \end{bmatrix} &= \begin{bmatrix} 0.299 & 0.587 & 0.114 \\ -0.14713 & -0.28886 & 0.436 \\ 0.615 & -0.51499 & -0.10001 \end{bmatrix} \begin{bmatrix} R \\ G \\ B \end{bmatrix}, \\
\begin{bmatrix} R \\ G \\ B \end{bmatrix} &= \begin{bmatrix} 1 & 0 & 1.13983 \\ 1 & -0.39465 & -0.58060 \\ 1 & 2.03211 & 0 \end{bmatrix} \begin{bmatrix} Y' \\ U \\ V \end{bmatrix}.
\end{aligned}
$$

Note that for small values of Y' it is possible to get R, G, or B values that are negative so in practice we clamp the RGB results to the interval [0,1].

In the following figure [Figure 31] we show the decomposition of an image in YUV color space.

$$
\begin{aligned}
R &= Y' + V\frac{1-W_R}{V_{\max}} = Y' + \frac{V}{0.877} = Y' + 1.14V, \\
G &= Y' - U\frac{W_B(1-W_B)}{U_{\max}W_G} - V\frac{W_R(1-W_R)}{V_{\max}W_G} \\
&= Y' - \frac{0.232U}{0.587} - \frac{0.341V}{0.587} = Y' - 0.395U - 0.581V, \\
B &= Y' + U\frac{1-W_B}{U_{\max}} = Y' + \frac{U}{0.492} = Y' + 2.033U.
\end{aligned}
$$

Equivalently, substituting values for the constants and expressing them as matrices gives these formulas for BT.601:

$$
\begin{bmatrix} Y' \\ U \\ V \end{bmatrix} = \begin{bmatrix} 0.299 & 0.587 & 0.114 \\ -0.14713 & -0.28886 & 0.436 \\ 0.615 & -0.51499 & -0.10001 \end{bmatrix} \begin{bmatrix} R \\ G \\ B \end{bmatrix},
$$

$$
\begin{bmatrix} R \\ G \\ B \end{bmatrix} = \begin{bmatrix} 1 & 0 & 1.13983 \\ 1 & -0.39465 & -0.58060 \\ 1 & 2.03211 & 0 \end{bmatrix} \begin{bmatrix} Y' \\ U \\ V \end{bmatrix}.
$$

Note that for small values of Y' it is possible to get R, G, or B values that are negative so in practice we clamp the RGB results to the interval [0,1].

In the following figure [Figure 29] we show the Decomposition of an image in YUV color space.

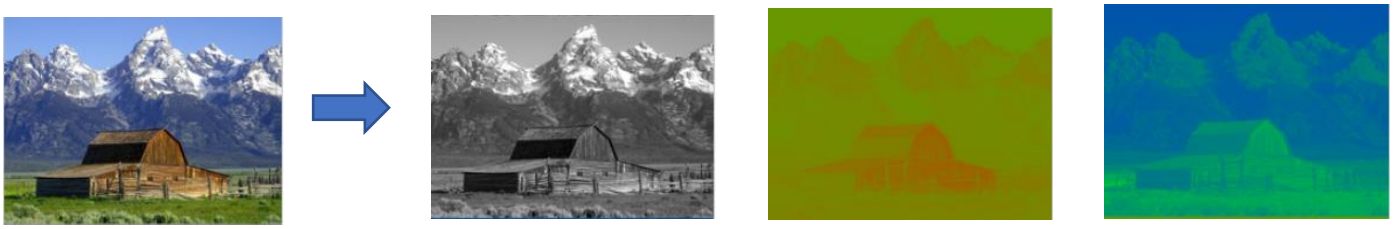


Figure 31: Decomposition of an image in YUV color space.

3.2.4 CIELAB COLOR SPACE:

CIE L*a*b* (CIELAB) is a color space specified by the International Commission on Illumination (French Commission international de eclarite, hence its CIE initialism). It describes

all the colors visible to the human eye and was created to serve as a device-independent model to be used as a reference.

The three coordinates of CIELAB represent the lightness of the color (L* = 0 yields black and L* = 100 indicates diffuse white; specular white may be higher), its position between red/magenta and green (a*, negative values indicate green while positive values indicate magenta) and its position between yellow and blue (b*, negative values indicate blue and positive values indicate yellow). The asterisk (*) after L, a and b are pronounced star and are part of the full name, since they represent L*, a* and b*, to distinguish them from Hunter's L, a, and b, described below.

Since the L*a*b* model is a three-dimensional model, it can be represented properly only in a three-dimensional space.[10] Two-dimensional depictions include chromaticity diagrams: sections of the color solid with a fixed lightness. It is crucial to realize that the visual representations of the full gamut of colors in this model are never accurate; they are there just to help in understanding the concept.

Because the red-green and yellow-blue opponent channels are computed as differences of lightness transformations of (putative) cone responses, CIELAB is a chromatic value color space.

A related color space, the CIE 1976 (L*, u*, v*) color space (a.k.a. CIELUV), preserves the same L* as L*a*b* but has a different representation of the chromaticity components. CIELAB and CIELUV can also be expressed in cylindrical form (CIELCH[11] and CIELCHuv, respectively), with the chromaticity components replaced by correlates of chroma and hue.

Since CIELAB and CIELUV, the CIE has been incorporating an increasing number of color appearance phenomena into their models, to better model color vision. These color appearance models, of which CIELAB is a simple example,[12] culminated with CIECAM02.

A. PERCEPTUAL DIFFERENCES

The nonlinear relations for L*, a*, and b* are intended to mimic the nonlinear response of the eye. Furthermore, uniform changes of components in the L*a*b* color space aim to correspond to uniform changes in perceived color, so the relative perceptual differences between any two colors in L*a*b* can be approximated by treating each color as a point in a three-dimensional space (with three components: L*, a*, b*) and taking the Euclidean distance between them.

3.2.4.1 RGB AND CMYK CONVERSIONS

There are no formulas for conversion between RGB or CMYK values and L*a*b*, because the RGB and CMYK color models are device-dependent. The RGB or CMYK values first must be transformed to a specific absolute color space, such as sRGB or Adobe RGB. This adjustment will be device-dependent, but the resulting data from the transform will be device-independent, allowing data to be transformed to the CIE 1931 color space and then transformed into L*a*b*.

3.2.4.2 RANGE OF COORDINATES

As mentioned previously, the L* coordinate ranges from 0 to 100. The possible range of a* and b* coordinates is independent of the color space that one is converting from, since the conversion below uses X and Z, which come from RGB.

3.2.4.3 CIELAB–CIEXYZ CONVERSIONS

3.2.4.3.1 Forward transformation

$$
\begin{aligned}
L^\star &= 116\, f\left(\frac{Y}{Y_\mathrm{n}}\right) - 16 \\
a^\star &= 500 \left(f\left(\frac{X}{X_\mathrm{n}}\right) - f\left(\frac{Y}{Y_\mathrm{n}}\right)\right) \\
b^\star &= 200 \left(f\left(\frac{Y}{Y_\mathrm{n}}\right) - f\left(\frac{Z}{Z_\mathrm{n}}\right)\right)
\end{aligned}
$$

Where

$$
\begin{aligned}
f(t) &= \begin{cases} \sqrt[3]{t} & \text{if } t > \delta^3 \\ \frac{t}{3\delta^2} + \frac{4}{29} & \text{otherwise} \end{cases} \\
\delta &= \frac{6}{29}
\end{aligned}
$$

Here, Xn, Yn and Zn are the CIE XYZ tristimulus values of the reference white point (the subscript n suggests "normalized").

Under Illuminant D65 with normalization Y = 100, the values are

$$
\begin{aligned}
X_\mathrm{n} &= 95.0489, \\
Y_\mathrm{n} &= 100, \\
Z_\mathrm{n} &= 108.8840
\end{aligned}
$$

Values for illuminant D50 are

$$X_\mathrm{n} = 96.4212,$$
$$Y_\mathrm{n} = 100,$$
$$Z_\mathrm{n} = 82.5188$$

The division of the domain of the f function into two parts was done to prevent an infinite slope at t = 0. The function f was assumed to be linear below some t = t0, and was assumed to match the t1/3 part of the function at t0 in both value and slope. In other words:

$$\begin{aligned} t_0^{1/3} &= mt_0 + c && \text{(match in value)} \\ \frac{1}{3}t_0^{-2/3} &= m && \text{(match in slope)} \end{aligned}$$

The intercept f(0) = c was chosen so that L* would be 0 for Y = 0: c = 16/116 = 4/29. The above two equations can be solved for m and t0:

$$\begin{aligned} m &= \frac{1}{3}\delta^{-2} && = 7.787037\ldots \\ t_0 &= \delta^3 && = 0.008856\ldots \end{aligned}$$

Where

$$\delta = 6/29.$$

3.2.4.3.2 Reverse transformation

The reverse transformation is most easily expressed using the inverse of the function f above:

$$\begin{aligned} X &= X_\mathrm{n} f^{-1}\left(\frac{L^\star + 16}{116} + \frac{a^\star}{500}\right) \\ Y &= Y_\mathrm{n} f^{-1}\left(\frac{L^\star + 16}{116}\right) \\ Z &= Z_\mathrm{n} f^{-1}\left(\frac{L^\star + 16}{116} - \frac{b^\star}{200}\right) \end{aligned}$$

Where

$$f^{-1}(t) = \begin{cases} t^3 & \text{if } t > \delta \\ 3\delta^2\left(t - \frac{4}{29}\right) & \text{otherwise} \end{cases}$$

and where

$$\delta = 6/29.$$

III. CONCLUSION

In this chapter we got to know some color spaces and how to convert from RGB to various other areas in order to understand the goal of this project.

In the next chapter we will show the tools used, datasets, the model, and a brief description of the implementation, followed by a section containing the experimental results and performance evaluation.

CHAPTER 4

EXPERIMENTATION AND RESULTS

I. INTRODUCTION

In this research, we used the architecture ResNet20 as our deep learning model for image classification.

The main goal in this study is to show what is the ideal color space to train the models, otherwise what is the impact of color spaces in the classification of images and in deep learning models, this result is shown by converting the cifar100 dataset into five color spaces.

We define the data set used in this chapter, the library as tensor-flow, and we give a brief implementation. At the end, we analyze the result, followed by a conclusion.

II. THE CIFAR-100 DATASET [13]

This dataset is just like the CIFAR-10, except it has 100 classes containing 600 images each. There are 500 training images and 100 testing images per class. The 100 classes in the CIFAR-100 are grouped into 20 super classes. Each image comes with a "fine" label (the class to which it belongs) and a "coarse" label (the superclass to which it belongs).

Here is the list of classes in the CIFAR-100:

N	Superclass	Classes
01	aquatic mammals	beaver, dolphin, otter, seal, whale
02	Fish	aquarium fish, flatfish, ray, shark, trout
03	Flowers	orchids, poppies, roses, sunflowers, tulips
04	food containers	bottles, bowls, cans, cups, plates
05	fruit and vegetables	apples, mushrooms, oranges, pears, sweet peppers
06	household electrical devices	clock, computer keyboard, lamp, telephone, television

07	household furniture	bed, chair, couch, table, wardrobe
08	Insects	bee, beetle, butterfly, caterpillar, cockroach
09	large carnivores	bear, leopard, lion, tiger, wolf
10	large man-made outdoor things	bridge, castle, house, road, skyscraper
11	large natural outdoor scenes	cloud, forest, mountain, plain, sea
12	large omnivores and herbivores	camel, cattle, chimpanzee, elephant, kangaroo
13	medium-sized mammals	fox, porcupine, possum, raccoon, skunk
14	non-insect invertebrates	crab, lobster, snail, spider, worm
15	People	baby, boy, girl, man, woman
16	Reptiles	crocodile, dinosaur, lizard, snake, turtle
17	small mammals	hamster, mouse, rabbit, shrew, squirrel
18	Trees	maple, oak, palm, pine, willow
19	vehicles 1	bicycle, bus, motorcycle, pickup truck, train
20	vehicles 2	lawn-mower, rocket, streetcar, tank, tractor

Table 3: The cifar_100 classes

III. LIBRARIES

1. KERAS [2]

Keras is a deep learning API written in Python, running on top of the machine learning platform TensorFlow. It was developed with a focus on enabling fast experimentation. Being able to go from idea to result as fast as possible is key to doing good research.

2. TENSORFLOW [14]

It is an open source artificial intelligence library, using data flow graphs to build models. It allows developers to create large-scale neural networks with many layers. TensorFlow is mainly used for: Classification, Perception, Understanding, Discovering, Prediction and Creation.

3. OPENCV [15]

OpenCV (Open Source Computer Vision Library) is an open source computer vision and machine learning software library. OpenCV was built to provide a common infrastructure for computer vision applications and to accelerate the use of machine perception in the commercial products. Being a BSD-licensed product, OpenCV makes it easy for businesses to utilize and modify the code.

The library has more than 2500 optimized algorithms, which includes a comprehensive set of both classic and state-of-the-art computer vision and machine learning algorithms.

4. SKIMAGE [16]

scikit-image (a.k.a. skimage) is a collection of algorithms for image processing and computer vision.

The main package of skimage only provides a few utilities for converting between image data types.

IV. IMPLEMENTATION

At the beginning, we convert the cifar100 data to different color spaces as shown in the following figure.

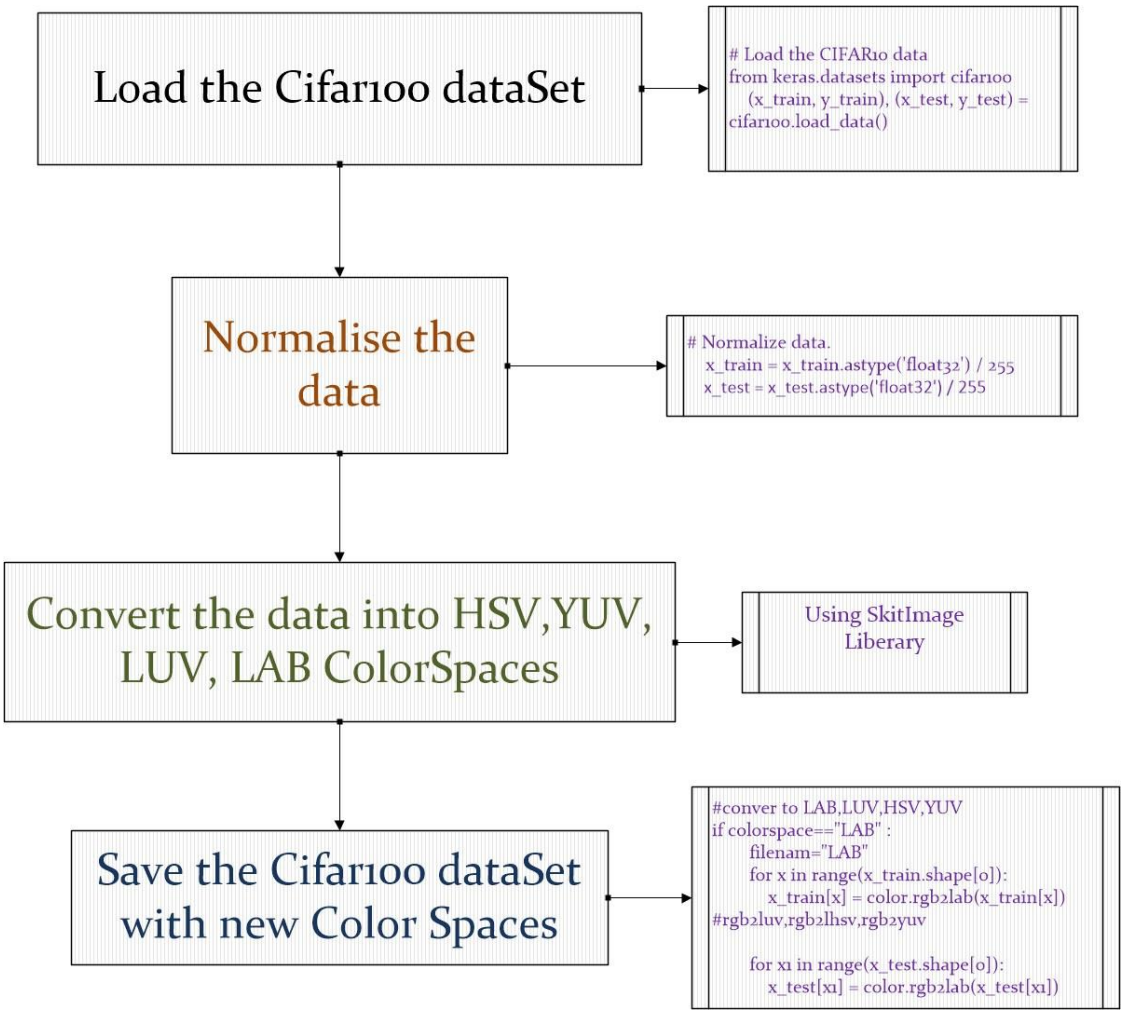


Figure 32: the process of converting Cifar 100 into HSV, LUV, LAB, YUV colors spaces

After we transformed the dataset, we train each model with data augmentation and without augmentation data as we explain in the experiments section.

V. EXPERIMENTS

In the part of experiments, we train the model ResNet20 after converting the dataset into five Color spaces: RGB, HSV, LUV, LAB, YUV.

For each color space using data augmentation and not using data augmentation, in each train we set the batch size to 128 and with 200 epochs.

We are using google Collaboratory because we have limited resources (CPU and no GPU) and the goal for this study is to show the color spaces effect into the convolutional neural network.

	RGB	HSV	LAB	LUV	YUV
apple	**0.90**	0.90	*0.89*	0.93	**0.94**
aquarium_fish	0.77	0.81	0.72	0.73	0.71
baby	0.57	0.50	0.53	0.54	0.59
bear	**0.58**	0.46	0.40	0.52	0.46
beaver	0.51	0.45	0.48	0.47	0.46
bed	0.68	0.67	0.65	0.65	0.68
bee	0.60	0.68	0.70	0.69	0.72
beetle	0.60	0.57	0.59	0.58	0.58
bicycle	0.75	0.72	0.77	0.72	0.69
bottle	**0.82**	**0.82**	**0.83**	**0.81**	**0.81**
bowl	0.47	0.49	0.53	0.53	0.46
boy	0.43	0.48	0.48	0.51	0.47
bridge	0.66	0.60	0.64	0.63	0.61
bus	0.64	0.66	0.64	0.60	0.60
butterfly	0.50	0.55	0.46	0.46	0.45
camel	0.71	0.67	0.69	0.68	0.66
can	0.64	0.63	0.61	0.64	0.69
castle	0.84	0.86	0.92	0.80	0.85
caterpillar	0.58	0.55	0.47	0.46	0.56

cattle	0.64	0.61	0.64	0.65	0.63
chair	0.83	0.78	0.81	0.82	0.75
chimpanzee	0.81	0.79	0.72	0.80	0.77
clock	0.57	0.48	0.51	0.55	0.53
cloud	0.86	0.76	0.85	0.86	0.78
cockroach	0.84	0.84	0.85	0.74	0.85
couch	0.57	0.55	0.53	0.48	0.51
crab	0.44	0.39	0.49	0.45	0.48
crocodile	0.43	0.40	0.42	0.45	0.51
cup	0.79	**0.85**	0.74	**0.84**	0.83
dinosaur	0.65	0.57	0.67	0.60	0.63
dolphin	0.65	0.64	0.72	0.58	0.64
elephant	0.80	0.74	0.77	0.87	0.81
flatfish	0.57	0.67	0.59	0.65	0.65
forest	0.70	0.60	0.64	0.64	0.57
fox	0.77	0.73	0.72	0.68	0.66
girl	0.46	0.51	0.41	0.44	0.43
hamster	0.80	0.83	0.87	0.81	0.77
house	0.63	0.64	0.69	0.67	0.62
kangaroo	0.55	0.50	0.55	0.54	0.58
keyboard	0.74	0.75	0.68	0.69	0.67

lamp	0.58	0.62	0.60	0.61	0.62
lawn_mower	0.77	0.67	0.70	0.70	0.85
leopard	0.47	0.43	0.42	0.47	0.49
lion	0.77	0.81	0.75	0.75	0.69
lizard	0.33	0.28	0.34	0.29	0.31
lobster	0.51	0.48	0.40	0.44	0.45
man	0.47	0.50	0.48	0.47	0.55
maple_tree	0.65	0.59	0.62	0.60	0.55
motorcycle	0.71	0.75	0.72	0.66	0.77
mountain	0.72	0.62	0.77	0.71	0.71
mouse	0.47	0.53	0.49	0.47	0.51
mushroom	0.68	0.65	0.66	0.64	0.67
oak_tree	0.57	0.57	0.56	0.56	0.59
orange	0.84	0.83	0.78	0.80	0.83
orchid	0.72	0.76	0.77	0.74	0.72
otter	0.44	0.35	0.41	0.36	0.41
palm_tree	0.89	0.83	**0.91**	0.82	0.90
pear	0.78	0.72	0.73	0.73	0.71
pickup_truck	0.83	0.82	0.85	0.81	0.79
pine_tree	0.57	0.66	0.55	0.58	0.55
plain	0.86	0.81	0.85	0.85	0.78

plate	0.69	0.75	0.65	0.61	0.66
poppy	0.65	0.61	0.72	0.72	0.65
porcupine	0.54	0.55	0.55	0.59	0.54
possum	0.51	0.46	0.47	0.51	0.56
rabbit	0.51	0.49	0.49	0.58	0.50
raccoon	0.68	0.64	0.62	0.57	0.58
ray	0.49	0.48	0.51	0.51	0.52
road	0.90	0.92	0.92	0.87	0.89
rocket	0.81	0.81	0.75	0.73	0.81
rose	0.70	0.69	0.67	0.72	0.77
sea	0.73	0.79	0.76	0.81	0.76
seal	0.37	0.34	0.31	0.38	0.32
shark	0.59	0.54	0.56	0.51	0.53
shrew	0.34	0.40	0.44	0.48	0.39
skunk	0.82	0.78	0.84	0.83	0.79
skyscraper	0.86	0.82	0.86	0.91	0.86
snail	0.66	0.60	0.69	0.58	0.68
snake	0.39	0.39	0.41	0.42	0.43
spider	0.66	0.62	0.65	0.73	0.65
squirrel	0.59	0.51	0.57	0.57	0.67
streetcar	0.65	0.59	0.56	0.63	0.62

sunflower	0.91	0.88	0.87	0.86	**0.94**
sweet_pepper	0.66	0.60	0.73	0.59	0.61
table	0.58	0.62	0.70	0.61	0.60
tank	0.74	0.72	0.72	0.79	0.79
telephone	0.67	0.58	0.66	0.59	0.58
television	0.64	0.70	0.71	0.63	0.68
tiger	0.70	0.68	0.80	0.71	0.74
tractor	0.72	0.66	0.68	0.67	0.66
train	0.67	0.60	0.69	0.61	0.63
trout	0.77	0.78	0.85	0.69	0.70
tulip	0.61	0.63	0.65	0.66	0.50
turtle	0.48	0.41	0.50	0.49	0.46
wardrobe	**0.87**	0.87	**0.91**	0.84	0.80
whale	0.68	0.63	0.67	0.63	0.60
willow_tree	0.62	0.64	0.67	0.65	0.69
wolf	0.71	0.76	0.74	0.69	0.69
woman	0.40	0.47	0.43	0.50	0.40
worm	0.62	0.65	0.64	0.65	0.57

Table 4: Accuracy per class of the Resnet20in differente color spaces

The number of parameters in each model is 280292 when the number of trainable parameters is 278916 and number of non-trainable parameter is 1376.

The time to train each model is 2 hours with colaboratory GPU [17].

VI. DISCUSSION AND RESULTS

After training the model ResNet with different color spaces we obtained the results listed in table below.

If we show the results, we infer that there is a difference between each color space and another,

this difference is not major in global terms.

We note that we have a 1.1% improvement in YUV color space accuracy compared to th RGB.

The less accuracy obtained is related to the color spaces of the HSV when the accuracy is 1.25% lower than the main color spaces.

By training with augmenting the data, we note that YUV and RGB have the same accuracy.

Color Space	ResNet20 Without Data Augmentation		ResNet20 With Data Augmentation	
	Test Loss	Test Accuracy	Test Loss	Test Accuracy
RGB	**4.74**	**43.66%**	1.66	**64.52%**
LUV	4.79	43.05%	1.68	63.99%
LAB	4.76	43.52%	1.66	64.20%
YUV	**4.67**	**44.76%**	1.67	**64.42%**
HSV	4.92	42.41%	1.68	63.51%

Table 5: Comparison of CNNs Models on the Different color spaces with cifar100

By looking at the table 5 below, which shows the value of the accuracy corresponding to each class of the data, Notice that the result change in the same class with the change of the color spaces.

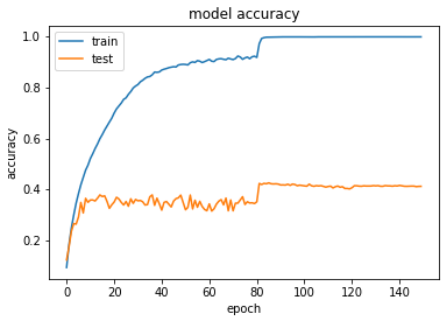
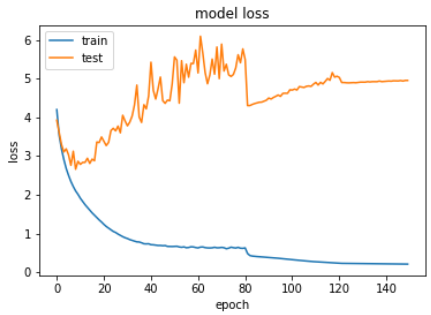
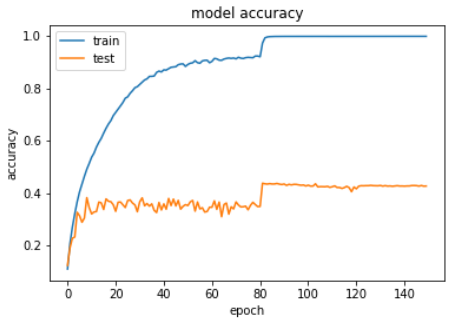
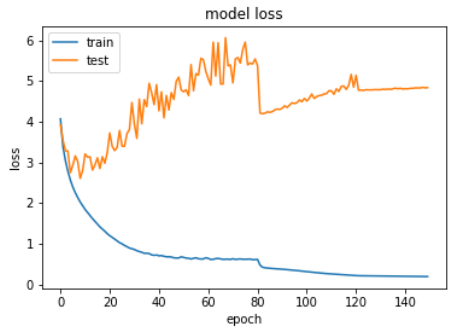
We observe that the accuracy are good for some classes and not well for some classes.

For example, looking to the class apple when the accuracy in RGB is 0.90 and it is 0.04 better with the YUV Color space. Another example takes with the class cup when the best accuracy is related with the HSV and the worst related with the LAB.

In the class wardrobe we observe a difference of 0.11 between the LAB when accuracy is 0.91 with HSV and the accuracy is 0.80 with LAB color space.

Graphs Accuracy and Loss of models:

- **RESNET WITHOUT DATA AUGMENTATION**

ResNet20 without Data Augmentation		
Colors Spaces	**Model Accuracy**	**Model Loss**
HSV		
LAB		

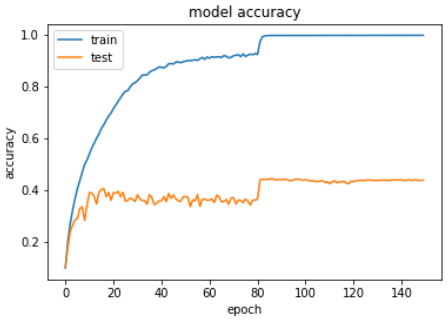
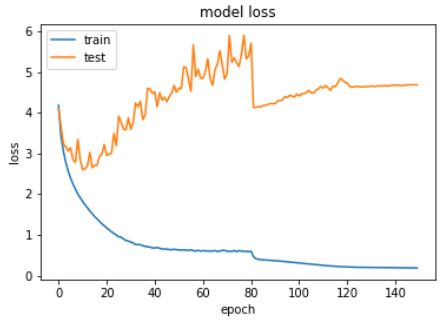
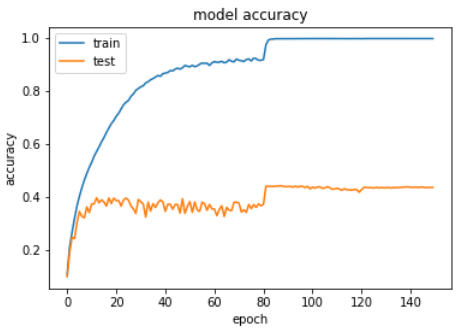
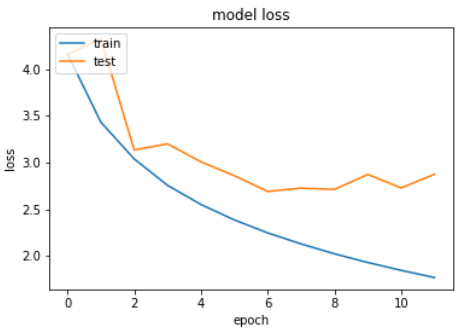
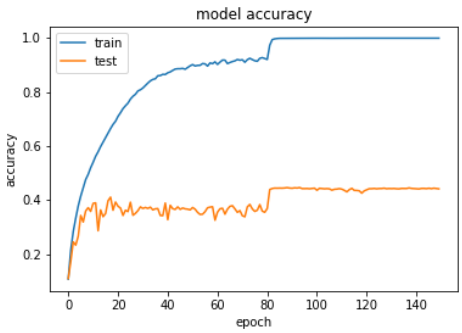
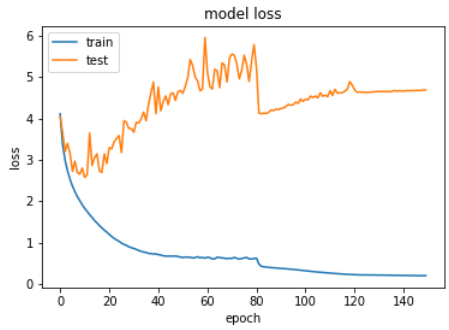
LUV		
RGB		
YUV		

Table 6: Graphs of accuracy and loss of ResNet20 without Data Augmentation

- **RESNET WITHOUT DATA AUGMENTATION**

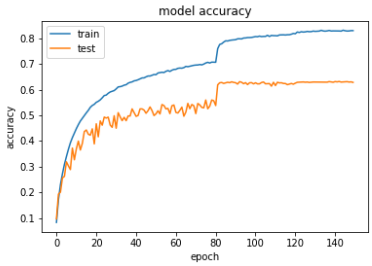
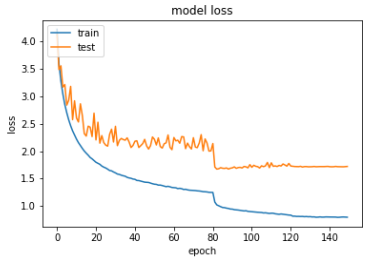
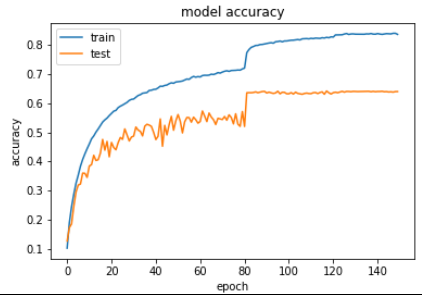
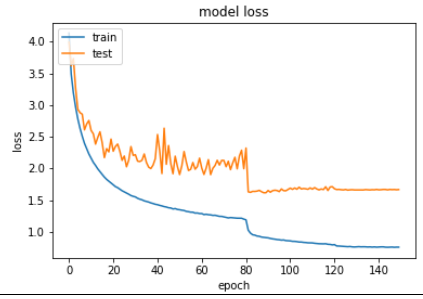
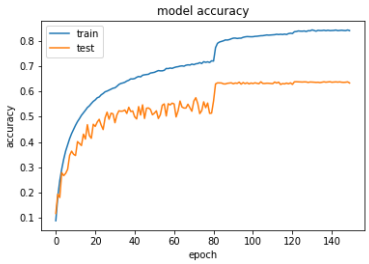
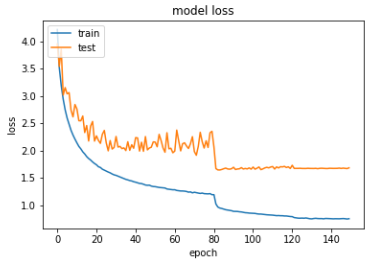
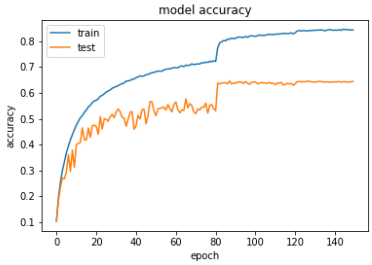
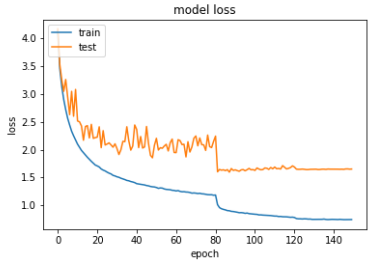
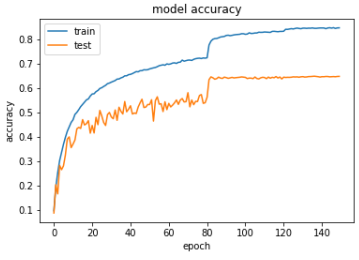
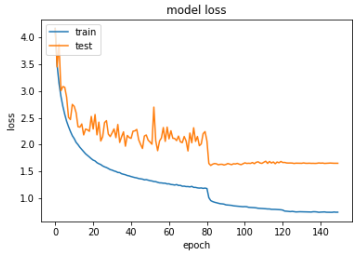
ResNet20 with Data Augmentation		
Color Spaces	**Model Accuracy**	**Model Loss**
HSV		
LAB		
LUV		
RGB		
YUV		

Table 7: Graphs of accuracy and loss of ResNet20 with Data Augmentation

CONCLUSION

The main focus in this research is to evaluate the effect of color spaces for the image classification task using deep learning. After converting the dataset to different color spaces and training with the ResNet20 architecture, we observe that there is no major change.

We obtain an improvement of 1.10 % in accuracy with the LUV color space comparing to the RGB and we observe that the HSV has a minor accuracy.

We didn't get the highest classification accuracy on cifar_100 because there are many classes but the number of training samples for each class is very small. In particular, the dataset comprises only 500 training images and 100 testing images per class.

We suggest in future research to train other new architectures (using a custom model) with a big dataset (data augmentation using image transformations) to get better results.

BIBLIOGRAPHY

[1] A. e. a. Chenine, "Colour Spaces Impact On Convolutional Neural Networks Performance.," 2019.

[2] Keras, "keras Libarery," 2019. [Online]. Available: https://keras.io/.

[3] N. Buduma, Fundamentals of Deep Learning, O'Reilly, 2017.

[4] N. Buduma, "Fundamentals of deep learning : designing next-generation machine," Media, 2017.

[5] K. Z. X. R. S. &. S. J. He, Deep residual learning for image recognition., In Proceedings of the IEEE conference on computer vision and pattern recognition (pp. 770-778), (2016)...

[6] F. Chollet, Deep learning with Python, Shelter Island: Manning Publications Co, 2018.

[7] "convolutional networks," 2020. [Online]. Available: https://cs231n.github.io/convolutional-networks/.

[8] Y. B. a. A. I. Goodfellow, "Deep Learning.," Courville, , 2016.

[9] Jason Brownlee, "how-to-configure-image-data-augmentation-when-training-deep-learning-neural-networks," 2020. [Online]. Available: https://machinelearningmastery.com.

[10] Jason Brownlee, "A Gentle Introduction to Dropout for Regularizing Deep Neural Networks," 2020. [Online]. Available: https://machinelearningmastery.com/dropout-for-regularizing-deep-neural-networks/. [Accessed 2020].

[11] The MathWorks, "Understanding Color Spaces and Color Space Conversion," The MathWorks, 2020. [Online]. Available: https://se.mathworks.com/help/images/understanding-color-spaces-and-color-space-conversion.html. [Accessed 2020].

[12] W. Burger, Principles of digital image processing : core algorithms, London: Springer, 2009.

[13] A. Krizhevsky, "the cifar100 dataset," Mar 2013-Sep 2017. [Online]. Available: https://www.cs.toronto.edu/~kriz/cifar.html.

[14] "TOP FIVE USE CASES OF TENSORFLOW," exastax, 2017. [Online]. Available: https://www.exastax.com/deep-learning/top-five-use-cases-of-tensorflow/. [Accessed 2019].

[15] opencv.org, 2020. [Online]. Available: https://opencv.org/.

[16] skimage, "skimage," 2020. [Online]. Available: https://scikit-image.org/.

[17] "colaboratory," 2020. [Online]. Available: https://research.google.com/colaboratory/faq.html.

[18] "Classification of Machine Learning," geeksforgeeks.org, [Online]. Available: https://www.geeksforgeeks.org/introduction-machine-learning/. [Accessed 2020].

[19] M. A. W. Singapore, , Advances in deep learning., Springer, 2020.