

Instituto Tecnológico y de Estudios Superiores de Occidente

Reconocimiento de validez oficial de estudios de nivel superior según acuerdo secretarial 15018, publicado en el Diario Oficial de la Federación del 29 de noviembre de 1976.

Departamento de Electrónica, Sistemas e Informática
Maestría en Sistemas Computacionales



Performance comparison of Deep Learning Models applied for Satellite Image Classification

TRABAJO RECEPCIONAL que para obtener el **GRADO** de
MAESTRO EN SISTEMAS COMPUTACIONALES

Presenta: **CARLOS ALBERTO CORDERO ROBLES**

Director: **DR. IVÁN ESTEBAN VILLALÓN TURRUBIATES**

Co-Director: **VÍCTOR HUGO MARTÍNEZ SÁNCHEZ**

Tlaquepaque, Jalisco. Agosto de 2020.

Acknowledgments

I would like to acknowledge:

My thesis advisor Dr. Ivan Villalón, who suggested me the topic for this document and always tried to find the way to unblock me with any blocker that appeared during the investigation.

The Mtro. Victor Martinez who provided good guidance in many aspects to improve the quality of this effort.

Luxoft that is the enterprise where I work and provided resources and flexibility to let me continue with my professional development.

The "Instituto Tecnológico y de Estudios Superiores de Occidente (ITESO)" for the resources provided for the development of this research. Additionally, to the "Consejo Nacional de Ciencia y Tecnología (CONACYT)" for the financial support received through the grant number 498325.

AGRADECIMIENTOS

El autor desea dar las gracias a:

Mi asesor de tesis Dr. Ivan Villaón, por sugerirme el tema de tesis para este documento y siempre busco maneras de desbloquearme en todos los contratiempos que surgieron durante la investigación.

Al Mtro. Víctor Martínez por todos sus consejos y guía en muchos aspectos para mejorar la calidad de este esfuerzo.

A la empresa Luxoft en la cual laboro por los recursos y flexibilidad que me permitió continuar con mi desarrollo profesional.

Al "Instituto Tecnológico y de Estudios Superiores de Occidente (ITESO)" por los recursos provistos para el desarrollo de esta investigación. Adicionalmente, al "Consejo Nacional de Ciencia y Tecnología (CONACYT)" por el soporte financiero recibido a través del número de autorización 498325.

Dedication

I would like to dedicate this project to:

My great family that has always supported my ideas and projects.

Dedicatoria

Me gustaría dedicar este proyecto a:

Mi gran familia que siempre ha apoyado mis ideas y proyectos.

Abstract

Satellite images classification is important for applications that involve the distribution of the human activities. Such distribution helps the governments to determine the best places to expand cities avoiding problems related to natural disasters or legal constrains. Currently, existing few agencies in charge of image classification and the area to cover is enormous. Therefor an automation of this process is necessary for this task otherwise, it will take an eternity to perform this task manually. On the other hand, detection and classification algorithms used before Machine Learning (ML) have not shown good result classifying this specific sort of images. However, latest approaches for image classification using Convolutional Neural Networks (CNN) have shown quite accurate results. In this research, we analyses the performance in four different CNN architectures used for satellite image classification. We use a dataset provided in 2017 by IARPA names IARPA fMoW. It contains more than two thousand images belonging to 62 classes already separated in train and validation.

The solution was implemented in Python using the Keras and Tensorflow libraries. The research was divided in two parts: Hyperparameters optimization and architectures results evaluation. For the first part we used only seven classes from a sample of the dataset (The sample is three hundred times smaller than the complete dataset). The architectures are trained using these seven classes of this small dataset to determine the best hyperparameters. After having selected the hyperparameters the architectures are trained with the complete sample. The evaluation is based on visual examination with the help of the tool Tensorboard and SKLearn metrics.

All the architectures showed accuracies near to 90% over the training dataset sample. The architecture with the best accuracy result was Resnet-152 with one accuracy of 99% over the training dataset Sample. The accuracy over the validation dataset will become important after training the architectures with the complete dataset. The training with the complete dataset will be performed in future works.

Resumen

Las imágenes satelitales y su clasificación son importantes en diversas aplicaciones que involucran la distribución de las actividades humanas. Dicha distribución ayuda a los gobiernos a determinar la mejor ubicación para la construcción en áreas específicas para evitar problemas relacionados con desastres naturales o conflictos legales. Actualmente no existen muchas agencias destinadas a este propósito y considerando lo enorme que es el área por cubrir se llega a la conclusión que es necesario automatizar el proceso para esta tarea. Esta tarea sería eterna si se realiza manualmente. Por otra parte, los algoritmos de detección y clasificación usados antes de “Machine Learning” no han mostrado buenos resultados en la clasificación de este tipo de imágenes. Un método que ha mostrado ser bastante preciso en tareas de clasificación son las Redes Neuronales Convolucionales (CNN). En esta investigación analizo el desempeño de cuatro diferentes modelos de CNN para esta tarea específica de clasificación de imágenes satelitales. El “dataset” utilizado es uno provisto el 2017 por IARPA fMoW. Este “dataset” contiene más de doscientas mil imágenes pertenecientes a 62 clases y ya previamente separadas en Entrenamiento y Validación.

La solución fue implementada en Python usando la librería de Keras ya integrada a Tensorflow. La investigación se divide en dos partes: optimización de hiperparametros y evaluación de los resultados de las arquitecturas. Para la primera parte usamos solo siete clases de una muestra del “dataset” original (La muestra es trescientas veces más pequeño que el “dataset” original). Las arquitecturas son entrenadas con estas siete clases de la muestra del dataset para así determinar los mejores hiperparametros. Después de haber seleccionado los mejores hiperparametros las arquitecturas son entrenadas con la muestra completa del “dataset”. y los métricos de cada modelo, con la ayuda de la herramienta Tensorboard. Después de tener los resultados, La evaluación se realizo de manera visual en base a la herramienta Tensorboard y usando las librerías SKLearn para generar los métricos.

Todas las arquitecturas mostraron una eficiencia cercana al 90% sobre el “dataset” de entrenamiento de la muestra usada. Resnet-152 fue la arquitectura con la mejor eficiencia la cual fue de 99% sobre el “dataset” de entrenamiento de la muestra usada. La eficiencia sobre el “dataset” de validación será importante después de entrenar las arquitecturas con el “dataset” completo. El entrenamiento usando el “dataset” completo será realizado en trabajos futuros.

Table of Contentents

MAESTRÍA EN SISTEMAS COMPUTACIONALES	1
1. INTRODUCTION.....	16
1.1. BACKGROUND.....	17
1.2. JUSTIFICATION	17
1.3. PROBLEM.....	18
1.4. OBJECTIVE.....	18
1.4.1. General Objective:.....	18
1.4.2. Specific Objective:.....	18
1.5. SCIENTIFIC OR TECHNOLOGIC SHARE/INNOVATION.....	19
2. STATE OF ART OR THE TECHNIQUE.....	20
2.1. SATELLITE IMAGE CLASSIFICATION BASED ON UNSUPERVISED METHODS	21
2.1.1. A NOVEL TWO-TIER PARADIGM FOR LABELING WATER BODIES	21
2.2. SATELLITE IMAGE CLASSIFICATION BASED ON SUPERVISED MACHINE LEARNING	22
2.2.1. PATTERN RECOGNITION SCHEME FOR LARGE-SCALE CLOUD	22
2.2.2. IMPROVED MACHINE LEARNING METHODOLOGY FOR HIGH PRECISION AGRICULTURE	23
2.3. SATELLITE IMAGE CLASSIFICATION BASED ON DEEP LEARNING	24
DEEP LEARNING NEURAL NETWORKS FOR LAND USE LAND COVER MAPPING.....	24
DENSE CLOUD CLASSIFICATION ON MULTISPECTRAL	26
2.3.1. SATELLITE IMAGE CLASSIFICATION WITH DEEP LEARNING	27
2.3.2. DEEP LEARNING FOR CLOUD DETECTION	28
3. THEORIC/CONCEPTUAL FRAMEWORK.....	30
3.1. ARTIFICIAL INTELLIGENCE (AI)	30
3.1.1. MACHINE LEARNING	31
3.1.2. DEEP LEARNING.....	33
3.2. CONVOLUTIONAL NEURAL NETWORK (CNN).....	33
3.2.1. INTRODUCTION	33
3.2.2. CONVOLUTIONAL NETWORK	34
3.2.2.1. LOCAL RECEPTIVE FIELDS AND SHARED WEIGHTS	34
3.2.2.2. SUB-SAMPLING	35
3.2.2.3. FULLY CONNECTED LAYER	36
3.2.2.4. CNN LEARNING.....	36
3.2.3. CNN ARCHITECTURES	36
3.2.3.1. RESNET-152	36
3.2.3.2. INCEPTIONV3.....	39
3.2.3.3. XCEPTION	41
3.2.3.4. DENSENET-161	43
3.3. CLASSIFICATION METRICS.....	44
3.3.1. ACCURACY	44
3.3.2. PRECISION	44
3.3.3. RECALL	44

3.3.4.	F1-SCORE	44
3.3.4.1.	MACRO-F1	45
3.3.5.	HAMMING LOSS	45
3.3.6.	JACCARD SCORE	45
3.3.7.	LOG LOSS.....	45
3.4.	MACHINE AND DEEP LEARNING LIBRARIES	45
3.4.1.	NUMPY.....	45
3.4.1.	TENSORFLOW AND TENSORFLOW-GPU	46
3.4.2.	KERAS	46
3.4.3.	TENSORBOARD.....	46
3.4.4.	PANDAS	46
3.4.5.	SKLEARN	47
4.	DEVELOPMENT METHODOLOGY	48
4.1.	REQUIREMENTS.....	49
4.2.	DATABASE PREPROCESSING	50
4.3.	DATASET SAMPLE	51
4.4.	MODELS CREATION/ADAPTATION.....	51
4.5.	MINIBATCH GENERATION	52
4.6.	HYPERPARAMETERS OPTIMIZATION	53
4.7.	DATASET SAMPLE TRAINING	57
4.8.	DATASET SAMPLE EVALUATION	58
4.9.	FULL DATASET TRAINING.....	72
5.	RESULTS AND DISCUSION.....	73
5.1.	RESULTS	74
5.2.	DISCUSSION	75
6.	CONCLUSIONS.....	77
6.1.	CONCLUSIONES	78
6.2.	FUTURE WORKS	78

LIST OF FIGURES

Figure 1	Tow-tier image labeling scheme.....	21
Figure 2	Vineyard split with tiles of 30x30px	24
Figure 3	Province of Manitoba constructed with snippets of satellite images	25
Figure 4	Tiling process 2224x224px with an overlap of 1/2 ($224/2 = 122px$).....	25
Figure 5	Deep Learning NN Produced LULC Map.....	25
Figure 6	Examples of 32x32 image patches of 13 classes	26
Figure 7	Architecture of the CloudNet, pretrained kernels are blue	26
Figure 8	Window classifier applied to thunder cells (Xanxi province). Left: Original image, Middle: Label map CNN, Right: Label map SVM.....	27
Figure 9	The deep earning system for classifying satellite imagery.	28
Figure 10	Artificial intelligence disciplines map	31
Figure 11	Neural network topology	32
Figure 12	Representation of a convolution over a gray representation of an image	35
Figure 13	CNN sequence proposed by LeCun.....	35
Figure 14	Residual learning block	37
Figure 15	34-layer Resnet architecture	38
Figure 16	Left side, factorization of a 5x5 conv grid into two 3x3 conv grid. Right side: factorization of a 3x3 conv grid into one 1x3 conv grid followed by a 3x1 conv grid	39
Figure 17	Inception block	40
Figure 18	Left side: Inception block 1. Center: Inception block 2. Right side: Inception block 3.	41
Figure 19	Xception architecture.....	42
Figure 20	Image that contains an airport. Bounding box in yellow. Center point in red. Image crop square 299x299 pixels in blue.....	50
Figure 21	Accuracy of the architecture ResnNet-152, with learning rate variations of 5×10^{-10} every 20 epochs.	53
Figure 22	Accuracy of the architecture InceptionV3, with learning rate variations of 5×10^{-10} every 20 epochs.	54
Figure 23	Accuracy of the architecture Xception, with learning rate variations of 5×10^{-10} every 20 epochs.	54
Figure 24	Accuracy of the architecture DenseNet-161, with learning rate variations of 5×10^{-10} every 40 epochs.	54
Figure 25	Accuracy over the training dataset of 7 classes using the ResNet-152 architecture	55
Figure 26	Accuracy over the training dataset of 7 classes using the InceptionV3 architecture	55
Figure 27	Accuracy over the training dataset of 7 classes using the Xception architecture	56
Figure 28	Accuracy over the training dataset of 7 classes using the DenseNet-161 architecture	56
Figure 29	Accuracy vs epoch results overlapped for the training of the four architectures using the dataset sample	57
Figure 30	First five results reported in CSV for architecture metrics	58
Figure 31	Metrics reported for the training dataset.....	58
Figure 32	Correct and incorrect predictions count by the architecture ResNet-152 for the train dataset of the dataset sample	59

Figure 33 Correct and incorrect predictions count by the architecture ResNet-152 for the validation dataset of the dataset sample	59
Figure 34 Correct and incorrect predictions count by the architecture InceptionV3 for the train dataset of the dataset sample	59
Figure 35 Correct and incorrect predictions count by the architecture InceptionV3 for the validation dataset of the dataset sample	60
Figure 36 Correct and incorrect predictions count by the architecture Xception for the train dataset of the dataset sample	60
Figure 37 Correct and incorrect predictions count by the architecture Xception for the validation dataset of the dataset sample	60
Figure 38 Correct and incorrect predictions count by the architecture DenseNet-161 for the train dataset of the dataset sample	61
Figure 39 Correct and incorrect predictions count by the architecture DenseNet-161 for the validation dataset of the dataset sample	61
Figure 40 Confusion matrix using Resnet152 as classifier over the training dataset.....	64
Figure 41 Confusion matrix using Resnet152 as classifier over the validation dataset	65
Figure 42 Confusion matrix using InceptionV3 as classifier over the training dataset.....	66
Figure 43 Confusion matrix using InceptionV3 as classifier over the validation dataset	67
Figure 44 Confusion matrix using Xception as classifier over the training dataset	68
Figure 45 Confusion matrix using Xception as classifier over the validation dataset	69
Figure 46 Confusion matrix using DenseNet161 as classifier over the training dataset.....	70
Figure 47 Confusion matrix using DenseNet161 as classifier over the validation dataset	71
Figure 48 SageMaker service achitecture	84
Figure 49 Administrator access selected.....	85
Figure 50 SageMaker access selected.....	85
Figure 51 Credential of user with SageMaker enabled	85
Figure 52 Notebook instances option.....	86
Figure 53 Basic configuration for SageMaker	86
Figure 54 Instance role creation.....	86
Figure 55 S3 access enabled for the instance	87
Figure 56 Open jupyter option	87
Figure 57 Backend selection	87
Figure 58 CUDA toolkit 10.1	88
Figure 59 Platform and version for CUDA toolkit	88
Figure 60 Download cuDNN selection	89
Figure 61 cuDNN v7.6 for CUDA toolkit 10.1	89
Figure 62 cuDNN added to toolkit.....	89
Figure 63 CUDA folders added to path environment variable	89
Figure 64 Virtual environment with name example and python 3.6 created	90
Figure 65 Opening virtual environment.....	90
Figure 66 Tensorflow installation	90
Figure 67 Keras installation	90
Figure 68 Install libraries based on requirements	91

LIST OF TABLES

Table 1 Selected features for Pixel-Based classification	22
Table 2 Kappa Statistics and Overall Accuracy [k (OA%)] for the Selected Landmarks.....	23
Table 3 Comparative Results table with the previous Study and local Vineyards application	24
Table 4 Pixel accuracy for the different networks	29
Table 5 Resnet architectures	37
Table 6 InceptionV3 architecture.....	40
Table 7 Accuracy results using different input dimensions	41
Table 8 DenseNet architectures	44
Table 9 Params and Memory characteristics of the architectures	52
Table 10 LR ranges that showed accuracy improvement	55
Table 11 Best learning rate found using Tensorboard	56
Table 12 Time spent per epoch.....	58
Table 13 Average accuracy for the dataset sample	61
Table 14 Max accuracy for the dataset sample	61
Table 15 Min accuracy for the dataset sample.....	61
Table 16 Training Dataset accuracy per class per architecture	62
Table 17 Validation Dataset accuracy per class per architecture	63
Table 18 F1-Score for the dataset sample	71
Table 19 Hamming Loss for the dataset sample	71
Table 20 Jaccard Score for the dataset sample.....	72
Table 21 Log loss for the dataset sample.....	72

LISTA OF ACRONYMS AND ABBREVIATIONS

IARPA fMoW		Functional Map of the World
ML		Machine Learning
DL		Deep Learning
CNN		Convolutional Neural Networks
SVM		Support Vector Machine
PCA		Principal Component Analysis
LDA		Linear Discrimination Analysis
SURF		Speeded Up Robust Features
SIFT		Scale Invariant Feature Transform
HOG		Histogram of Gradients
POP		Province of Manitoba
OLI		Operation Land Imager
TIRS		Thermal Infrared Sensor
IR		Infra-Red
UAV		Unmanned Aerial Vehicle
MSG		Meteosat Second Generation
NN		Neural Network
AI		Artificial Intelligence
TF		Tensorflow
LULC		Land use/Land cover
TIFF		Tagged Image File Format

1. INTRODUCTION

Several applications for the satellite images are related to public works. They contribute to the development of the civilization and its life quality, although they are extremely expensive. One construction can cost easily thousands of millions of dollars. A well planification can be de difference between a good usage of the building or a waste of money that can be lost because the accessibility or a natural disaster.

To plan and determine correct strategy for the public works construction it is required a classification of the current public works and “land use” distribution.

The methodologies used before ML have not shown good results classifying this kind of images. Based on those ML results, in this research we are going to use and compare the behavior of four different DL models in order to find the best accuracy possible.

The models used the dataset provided for the challenge fMoW in 2017 that contains 62 classes already labeled and separated in training and validation. This is a multispectral dataset, however we will work only with the classic RGB bands. Nevertheless, a dataset preparation is required. As a future work the use of more bands is expected.

1.1. Background

This research is based mainly on 2 previous researches. The first one is entitled “Satellite Image Classification with Deep Learning” [7]. In this paper Mark Pritt and Gary Chern used the dataset provided from IARPA fMoW. They got an accuracy of 83% using a hybrid model using four models Resnet-152 [8], InceptionV3[9], Xception [10] and DenseNet-121[11]. The models were trained using one epoch. Data augmentation was required because satellite images don’t follow any alignment. The dataset was expanded by eight by flipping the image horizontal a vertically and rotating the image 90°, 180° and 270° degrees. The output of the model was attached to the model’s outputs to create a hybrid model that categorize the images.

The second paper entitled “Functional Map of the World” [12] is related with the dataset itself. This paper describes the characteristics of the dataset. As it is explained, this is a multispectral dataset and the images goes from three to eight bands. In addition, the dataset contains a metadata with valuable geographic information as well as the sensor information.

1.2. Justification

Satellite images and its classification is important for many applications that involve the distribution of the human activities. Public works is one of the most representative and expensive responsibilities of the governments and in some cases for some investors. At the same time, they represent an important factor for the population development and distribution. Unfortunately, these sorts of investments are overwhelmingly expensive. Just here in México two constructions are in progress and together will cost more than 150,000 million of Mexican pesos (more than 7.5 hundreds of millions of dollars) [1][2]. This is just to give an example of the cost that can take a public work of this magnitude. Although, the public works have a purpose and if they are well planned the benefits for the enclosed population and the life quality tends to improve. On the other hand, if the public work is not well planned and geographically well distributed it will directly impact the enclosed population as well as the economy of the country.

Another controversial example is the “Nuevo Aeropuerto Internacional de México (NAIM)” that was recently cancelled because of floods probabilities. We can even confirm this information and many other environment hazardous impacts in the document resolute analysis SGPA/DGIRA/DG/09965 [3] that shows that in some periods of the year the airport remains covered by water, in addition the airport would be near to areas were endemic and extinction endangered species lives. The cost for the cancellation of this airport was 120 thousand millions of Mexican pesos (6 thousand millions of dollars approximately) [4]. Those are the kind of mistakes related with constructions allocation that can be avoid with good planification and distribution of public works.

Nevertheless, this situation is not only limited to government public works. Many private constructions focused on recreation are also involved. One example is the Mercedes-Benz stadium in New Orleans also called the superdome. In 2005, this city suffered of floods caused by the hurricane Katrina and that was granted with a renovation that will cost 450 millions of dollars [5].

This take us to other important point when a construction is planned. As we have seen these investments costs millions of dollars and well applied they will retrieve a lot of incomes to the population

and the nation in general. On the other hand, if they are not well located or geographical distributed or if they are endangered for any natural disaster it will result in one enormous catastrophe for the stakeholders and the people that inhabit in the area.

In order to plan it is required to map the current distribution of the public works and constructions. One powerful resource that can be used are the satellite images. Although the task of classification is most of the time performed manually and in order to categorize all the entire earth surface will be quite exhausting. The importance of this task has triggered many challenges [6] that are not limited to any technology and the goal is to obtain the higher performance in detection and classification of many kind of images including satellite images.

The latest years neural networks technologies have shown good results categorizing images, specifically Deep Learning models that involve many layers of neurons. This technology is costly and involve hours or even weeks of computing. That is why it is highly valuable to determine if this technology has good result with a specific dataset. In this case, a dataset of satellite images.

1.3. Problem

The problem faced in this research is a classic classification problem related with satellite images and the method of classification is DL. The dataset taken as reference is IARPA fMoW that provides 62 classes. The CNN architectures used are Resnet-152, InceptionV3, Xception and DensNet-161 [18]. Classification of satellite images is quite important for the civilization distribution. The latest years DL is a state-of-the-art technology that has shown outstanding results in image classification. Currently, several DL architectures and models are accessible, and the goal of this research is to compare the behavior of four different architectures to determine the most appropriated for this satellite image datasets.

1.4. Objective

1.4.1. General Objective:

To find the highest accuracy using four different CNN architectures and working with a labeled satellite images dataset with 3 bands.

1.4.2. Specific Objective:

To accomplish the general objective, are required the following specific objectives:

- * To obtain a labeled dataset to be able to stimulate the CNN models with such information.
- * To preprocess the dataset to accomplish the specifications of every Model.
- * To build or get every model and adapt it to the number of classes that are handled in our dataset.
- * To define a strategy to manage the dataset. Maybe it's required to separate the images in tiles or maybe it is a good idea to start working with a small batch of images.
- * To find the best hyperparameters for every CNN model.

* Evaluate the behavior of every CNN using metrics as F1-Score, Hamming Loss, Jaccard Score and Log loss.

1.5. Scientific or technologic share/innovation

This document provides the results of the overall behavior of four different CNN architecture using a satellite images dataset.

2. STATE OF ART OR THE TECHNIQUE

We can find in recently researches related to the classification of satellite images using DL. Some of them are focused in a specific area of study like land usage or bodies of water. Some of them are even more specific and would focus their work in a specific topic like areas for agriculture and cropping. On the other hand, we can find paper related with a more general detection of human construction that not only detect land usage areas but also public works.

One particularity that have the satellite images is that some images contain multispectral information, that means that the image is not having only the classic RGB layers but also near and far infrared that have shown good result making visible bodies of water. Some other images have eight layers and some of them have the capacity to trespass some centimeters the surface of the earth, nevertheless not all the investigations are using such layers, most of them are still using only the classic RGB layers.

One common problem that exist for the satellite images that only work with the classic RGB layers are the clouds. It is something that cannot be removed and is always going to be a problem of the classic layers. Some papers have even focused their investigation on the detection of clouds and categorizing its shape and texture.

Another aspect that we can find in previous works is the use of hybrid models. Sometimes it is possible to option more accuracy if we add to the system not only the image but also the geographical information or any other post processing that can be performed with dense layers or SVM.

2.1. Satellite Image classification based on unsupervised methods

2.1.1. A novel two-tier paradigm for labeling water bodies

In 2017 Janice Aroma and Kumudha Raimond from the University of Karunya described the steps required to label satellite images. Labeling is an unsupervised method of clustering required to perform supervised training. The study doesn't end just with the labeling, it also applies a supervised classifier to recognize seasonal water bodies [13].

In this paper is explained the necessity of intelligent models to determine and measure the damage and post hazard because the surveys and field works insitu represent a heavy time consumption. Satellite images are the opposite problem, they contain a lot of information at every band and the task now is to extract the features with accuracy and in the shortest time possible.

Currently the traditional methods for classification PCA and LCA are dependent of statistical color features although the satellite images commonly come from different satellites that have different sensors then using local discriminators instead of global discriminators have shown been more robust. The local discriminators used for image classification are SURF, SIFT and HOG.

Unfortunately, the process to obtain the features is still not efficient. In general, the most highly used labeling methods are pixel based that are laborious and complex. Then they proposes a method named Two-tier labeling scheme. In Figure 1 we can see an example where one specific water body was taken for this study and the first tier was the Local monsoon pattern. The second one was the water body area. Obtaining descriptors provided from these two tiers they were able to label different states of the water body including disasters or hazardous events.

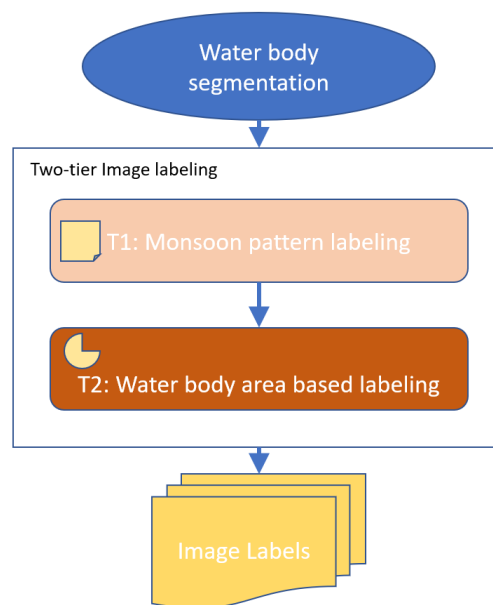


Figure 1 Two-tier image labeling scheme

2.2. Satellite Image classification based on supervised Machine learning

2.2.1. Pattern Recognition Scheme for Large-Scale Cloud

In 2018 Adrian Pérez-Suay and Jordi Muñoz-Marí used machine learning for cloud detection over landmarks using MSG satellites images. MSG SEVIRI takes an image every 15 min with 12 spectral channels and the size of the images is 3712x3712. The dataset used was from the year 2010 with 200 landmarks and 7 million images [17].

The labels used for this experiment were Space/no data (0), water (50), land (100), cloud (200) and the ML classifier method was SVM.

The features extracted were related with reflectance, brightness and temperature. IR channels provide information about the temperature of clouds, surface, and land. Some NIR channels help distinguish between cloud and land and some IR channels help to detect fog and log clouds. At the end only 16 features were used, we can see them listed in Table 1.

Table 1 Selected features for Pixel-Based classification

NUMBER	FEATURES	DAY	NIGHT
1	R1 VIS 0.6 μm	Yes	No
2	R2 VIS 0.8 μm	Yes	No
3	R3 NIR 1.6 μm	Yes	No
4	R4 IR 3.9 μm	Yes	Yes
5	BT7 IR 8.7 μm	Yes	Yes
6	BT9 IR 10.8 μm	Yes	Yes
7	BT10 IR 12.0 μm	Yes	Yes
8	Cloud Test: R2/R1	Yes	No
9	Snow Test: (R1-R3)/(R1+R3)	Yes	No
10	NDVI: (R2-R1)/(R2+R1)	Yes	No
11	$mean_{3 \times 3}(R1)$	Yes	No
12	$std_{3 \times 3}(R1)$	Yes	No
13	$mean_{5 \times 5}(R1)$	Yes	No
14	$std_{5 \times 5}(R1)$	Yes	No
15	$mean_{3 \times 3}(BT9)$	Yes	Yes
16	$std_{3 \times 3}(BT9)$	Yes	Yes

In order to give more accuracy to the problem it was divided in different scenarios depending on the time of the day and the zone. The day was divided in four ranges and the landmarks were assigned to twelve zones. The results are presented below in Table 2 where we can see that twelve zones were selected (Name) and four days divisions are evaluated according to the light intensity of the day: High Light, Medium Light, low light and Night.

Table 2 Kappa Statistics and Overall Accuracy [k (OA%)] for the Selected Landmarks

NAME	#LM	HIGH LIGHT SZA<=SZA _M	MEDIUM LIGHT SZA _M <=SZA<=80	LOW LIGHT 80<=SZA<=90	NIGHT SZA>=90	GLOBAL
AD DAKHLA (MOROCCO)	0	0.62 (83.24)	0.71 (87.03)	0.63 (83.51)	0.65 (85.78)	0.66 (85.35)
AQABA2 (SAUDI ARABIA)	14	0.50 (82.74)	0.56 (83.34)	0.57 (84.95)	0.65 (90.06)	0.59 (86.65)
AZORES5 (PORTUGAL)	17	0.72 (87.61)	0.64 (86.20)	0.56 (80.98)	0.56 (79.86)	0.61 (82.94)
CHAD2 (CHAD)	48	0.76 (87.94)	0.74 (87.09)	0.64 (81.97)	0.58 (78.75)	0.65 (82.83)
DANGER (SOUTH AFRICA)	63	0.82 (90.89)	0.81 (90.36)	0.68 (84.22)	0.63 (81.57)	0.71 (85.64)
GRAMPIAN (SCOTLAND)	83	0.70 (89.99)	0.69 (88.90)	0.57 (81.92)	0.48 (78.32)	0.57 (82.95)
LIBREVILLE (GABON)	107	0.69 (87.93)	0.73 (89.52)	0.69 (87.34)	0.68 (88.25)	0.69 (88.40)
MESSINA (SICILIA)	120	0.80 (90.09)	0.80 (89.92)	0.73 (86.47)	0.71 (85.73)	0.75 (87.61)
NASSER2 (EGYPT)	131	0.57 (89.16)	0.59 (88.33)	0.63 (90.08)	0.71 (94.17)	0.64 (91.52)
RHODES (GREECE)	154	0.80 (91.54)	0.77 (88.73)	0.72 (86.44)	0.72 (86.50)	0.75 (88.05)
TENERIFE (SPAIN)	177	0.77 (88.46)	0.71 (85.41)	0.63 (81.30)	0.67 (83.18)	0.69 (84.68)
VALENCIA (SPAIN)	190	0.83 (91.59)	0.84 (92.18)	0.76 (87.88)	0.73 (86.72)	0.78 (89.01)

Best Results are highlighted in bold.

2.2.2. Improved Machine Learning Methodology for High Precision Agriculture

In 2018 Jérôme Treboux and Dominique Genoud members of the Institute of Information Systems of the University of Applied Sciences, Valais Sierre, Switzerland developed a method using ML to improve from 89.6% of accuracy (using methods based in color analysis) to 94.27% in classification of vineyards and roads. These sorts of analysis are important to increase the productivity in agriculture and determine the correct amount of inputs (water, fertilizer, etc.) in the correct place [16].

The research used infrared images taken from a UAV that in this case was a drone that can fly over the fields carrying treatment products.

The dataset were five images of five different vineyards in Valais, Switzerland taken by a drone. An expansion of the dataset was performed dividing the images in tiles of 30x33 pixels (Figure 2) to end up with 13, 005 images manually labeled. The categories to classify were: Road, Vineyard or Other. We can notice in Figure 2 that using tiles of 30x33px some objects are mixed but rarely.

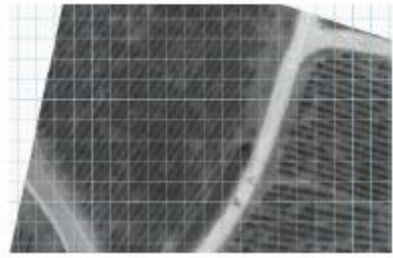


Figure 2 Vineyard split with tiles of 30x30px

The dataset was divided 90% for training and 10% for validation. Only 16 features were taken from the images from the following three categories, First Order Statistics (Min, max, mean, geometric mean, sum, variance, etc.), Tamura (Granularity, Contrast, kurtosis of directional, etc.) and Haralick (Statistical features based on gray-level).

The final overall accuracy of the algorithm is of 94.275% that shows an improvement from the previous studies (Table 3) that showed accuracy near to 90%.

Table 3 Comparative Results table with the previous Study and local Vineyards application

	BEST DETECTION WITHIN THE STUDY	LOCAL VINEYAR 1	LOCAL VINEYAR 2
ACCURACY ± STD ERR	96.06% ± N/D	90.02% ± 1.17%	89.6% ± 1.01%

2.3. Satellite Image classification based on Deep Learning

DEEP LEARNING NEURAL NETWORKS FOR LAND USE LAND COVER MAPPING

This research performed in 2018 by Christopher D. Storie and Chrisotpeh J. Henry from the University of Winnipeg, had the objective to classify the land use/land cover of Manitoba Canada. GeoManitoba that is department of POM requested such classification and continue doing it human bases semi-automated showed to be an unsustainable task (as much as 4800 work hours) because it has to be performed yearly [14].

The interest of the stakeholders relay on the possibility to get information related to flood forecasting, urban and rural land use planning, resource management, disaster management and planning.

POP provided a multispectral (6 bands) Landsat database provided by GeoManitoba of 19,039 images belonging to 18 classes for the years 1993, 2000 and 2004. The requirement was to use the model to tag all the untagged regions of those years.

The model used was VGG-16 that has an input layer of 224x244.

The first step was to construct the region of Manitoba using satellite images and matching them to construct an image with the entire map as we can see in Figure 3.

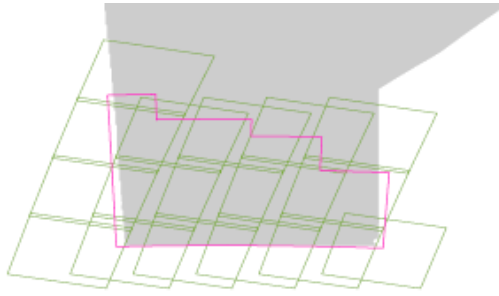


Figure 3 Province of Manitoba constructed with snippets of satellite images

After that the map was divided in tiles of the size of the input layer of the model (224x224) but overlapped the half size of the window (112 pixels) as we can see in Figure 4.

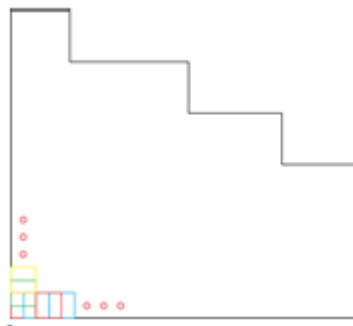


Figure 4 Tiling process 2224x224px with an overlap of 1/2 ($224/2 = 122px$)

The dataset was divided in 18,054 images for training and 958 for validation. Since the dataset was too small it was required to apply transfer learning. The training plus the mapping process took near to 10 days and the accuracy was the following: 82.5% (1993), 81.2% (2000) and 79.5% (2004). Using the classification with Deep Learning NN ever tile was classified and colored to obtain a complete LULC Map as illustrated in Figure 5.

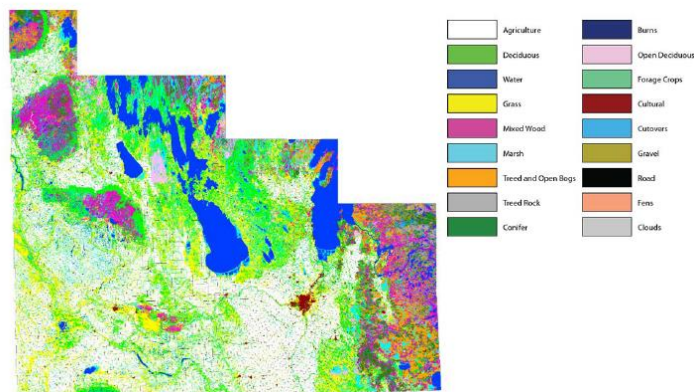


Figure 5 Deep Learning NN Produced LULC Map

Dense Cloud Classification on Multispectral

In this research by the University of Dortmund by the Image Analysis Group a methodology to categorize 13 cloud classes (also called genera) was performed based on a small dataset (147 images) of Landsat 8 satellite images. The images provided by the dataset used two different sensors OLI that has nine band from 435nm to 2294nm and TIRS that has two bands from 1060nm to 1251nm [15].

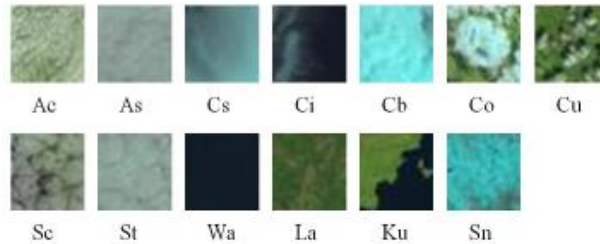


Figure 6 Examples of 32x32 image patches of 13 classes

The dataset of 147 images (every image with a range between 6000 and 8000 pixels) was expanded cropping it in squares of 320x320 pixels and down sampling the squares to 32x32 pixels to end up with an expanded dataset of 9578 images. This expanded dataset was augmented with rotations and mirroring to obtain one thousand images per class, thirteen images classes in total that were distributed 90% for training and 10% for validation. The labeling was performed manually in few sessions, in Figure 6 we can see examples of the labeled images.

For the CNN they used a customized Alex-Net. The customization consisted in removing all the layers excepting the first two Convolutional layers and their pooling. Transfer learning was applied for these two layers from imageNet database. Finally, they added an untrained Convolutional Layer and a fully connected layer and used Adagrad optimizer for the training. They called this model CloudNet (Figure 7) About training time, they reported that the training of 10,000 epochs took near to ten minutes.

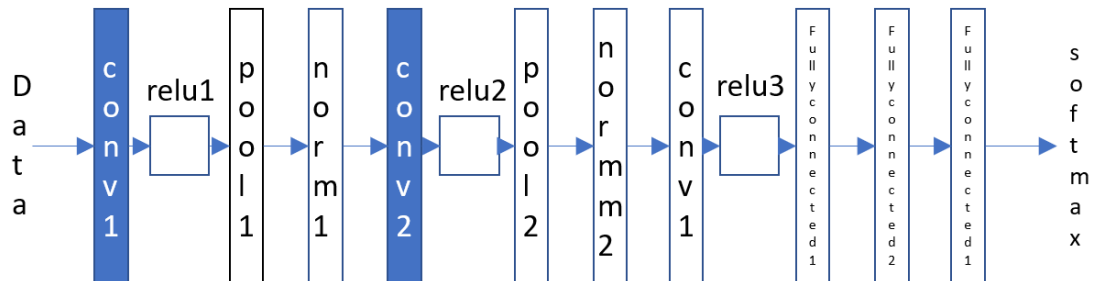


Figure 7 Architecture of the CloudNet, pretrained kernels are blue

For the SVM they used SURF and one additional input for the SVM was the color histogram of the image, this final training took between 30 and 60 min.

The accuracy using RGB layers was 85.8% after adding the IR layer the accuracy slightly raised to 86.1% and finally adding the SVM classification the final accuracy was 95.4%. In Figure 8 we can see

an example of labeling using CloudNet CNN and CloudNet CNN plus SVM over one satellite image of the province of Xanxi.

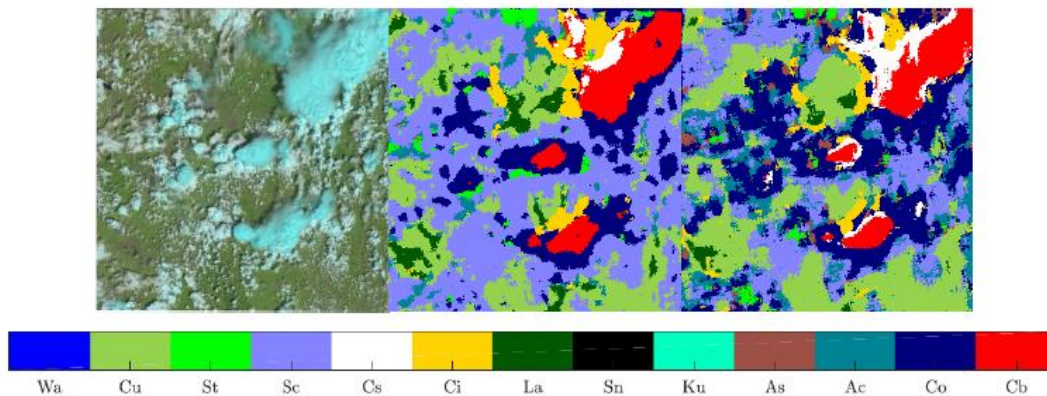


Figure 8 Window classifier applied to thunder cells (Xanxi province). Left: Original image, Middle: Label map CNN, Right: Label map SVM

2.3.1. Satellite Image Classification with Deep Learning

In 2017 Mark Pritt and Gary Chern from Lockheed Martin Co. developed a DL classification system in python using Keras and Tensorflow libraries and a GPU NVIDIA Titan X. They are classifying satellite images, because the geographic expanses to be covered are great and the analyst available to conduct the searches reduced it is required to automate the process [18].

The dataset used was the one provided by the Intelligence Advanced Research Projects Agency (IARPA), the dataset is named the Functional Map of the World (fMoW) that contain 62 classes already labeled. The images contain from 3 to 8 bands and include metadata, although for this experiment only 3 bands were used. The dataset already comes divided in training and validation. For training the dataset was augmented eightfold by flips and 90°, 180° and 270° degree rotations.

The system that they proposed is an ensemble of CNNs that receive a processed (preprocessed) image and followed by a classic NN at the end the maximum probability determines de classification (Figure 9).

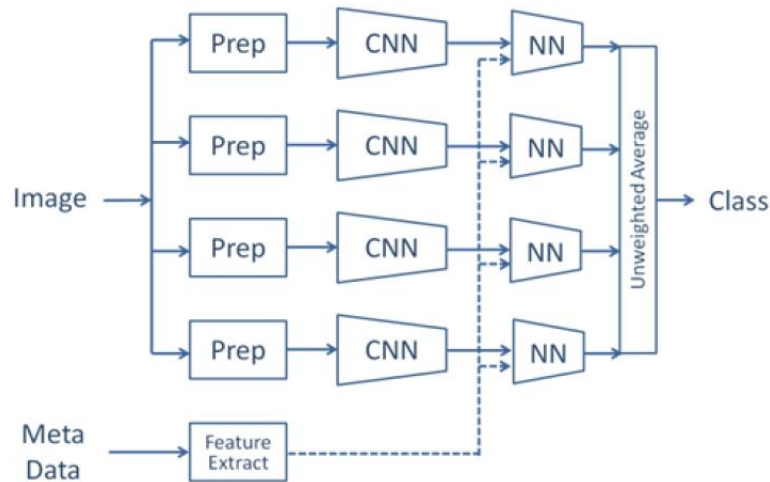


Figure 9 The deep learning system for classifying satellite imagery.

The preprocessing is needed because the satellite images in general don't match the CNN input size of 224x224 or 299x299 then they must be cropped and adjusted. The bounding box is part of the metadata information and it is used to perform the cropping.

The CNN models selected were DenseNet-161, ResNet-152, Inception-v3 and Xception. The metadata was normalized and used as an input for the NN models. Transfer learning was applied from ImageNet.

For training, the CNN models were trained only one epoch and for the NN twenty epochs were needed. The results showed an accuracy of 83% and F1 score of 0.797.

2.3.2. Deep Learning for Cloud Detection

In this research performed in the University of Toulouse a comparison between DL methods used with classical handcrafted features and classical CNN is performed for cloud detection [19].

The detection consists in labeling every pixel of an image indicating if that pixel belongs to a cloud or not. The first detectors took the descriptors from the morphology of the shadow or from dedicated spectral bands, although they showed a lack of generalization and low robustness, the next generation of detectors used handcraft features that the engineer itself select for DL this is not required and the results are even better.

The dataset is 10,000 images from SPOT 6 satellite that provide 4 band (RGB and NIR) granted by the Airbus Defense and Space.

The comparison was made between the CNN solution against four classic methods: RGBI raw pixel values, band ratios, Gabor coefficient and discrete cosine transform (DCT) coefficients.

The selection of features critical form the classic methods. Some of these features are noise sensitive, require neighboring information and many times need physical correction, nevertheless band ratios have shown good results. Band-ratios were processed at three different spatial resolutions (60m,

120m, 240m). Gabor is mainly used for textual features of the image, for this study 48 features were taken and for DCT 192 features that were the input for a classic NN to determine de classification. For the CNN this is not required but instead of having features it is required to take an input of 32x32 (patch) were the pixel to analyze is just at the center. The CNN used was like the one used in CIFAR-10. The results are shown below.

Table 4 Pixel accuracy for the different networks

INPUT TYPE	NETWORK	ACCURACY	RECALL	PRECISION
GABOR	ANN	77%	43%	66%
RAW PIXELS	ANN	83%	68%	77%
FEATRUES	ANN	81%	68%	80%
SUEPRPIXELS	ANN	83%	69%	80%
DCT	ANN	83%	75%	80%
PATHCHES	CNN	86%	75%	81%

In general, the CNN (patches) got slightly better results 86%, we shall consider that the work to identify the features is not required when working with CNN.

3. THEORIC/CONCEPTUAL FRAMEWORK

This chapter describe some concepts that were used during the research. Some of them are methods or techniques followed to create and train the architectures. Some others are metrics used to evaluate such architectures. Many topics mentioned are related to Artificial Intelligence, in particular CNN architectures and the platforms used to train and evaluate its performance.

3.1. Artificial Intelligence (AI)

Artificial intelligence is a discipline found in 1956 by John McCarthy. The objective of this discipline is to mimic the way how humans solve problems or to solve problems that at this time only humans were able to solve and therefore were related with intelligence [20].

Even before this discipline was formalized, we can find many examples in the history talking about automatons and early concepts of robots that had specific task [21]. Many of them were simply mechanic devices with human or animal shapes that had the task to answer a specific question performed by a user or to execute a movement depending on a user action.

John McCarthy defined IA as a branch of computer science which deals with the study and design of intelligent agents that perceives its environment and takes actions which maximize its chances of success [22]. Many of the problems related with intelligence included situations that required experience, reasoning to define a solution or inferring and quick response. In addition, these systems can take decisions based on priorities to overcome complexity and ambiguity in problems. The goal of this discipline is to build computer programs that exhibit the mentioned intelligence behavior.

Normally, the languages used for IA are not oriented to handle numeric information because the input for these systems is qualitative information.

AI is divided in four main components:

- * Expert Systems
- * Heuristic Problem Solving

*Natural Language Processing

* Vision

Expert systems optimized the solutions to handle it as an expert and obtain the best possible performance. Heuristic problem solving find the solution among a group of possible solutions evaluating them and applying a sort of heuristic to find the closes option to the optimal. Natural language processing handles the communication between the machine and the human using human languages. Vision is dedicated to shapes, features, etc. recognition. [22][23]

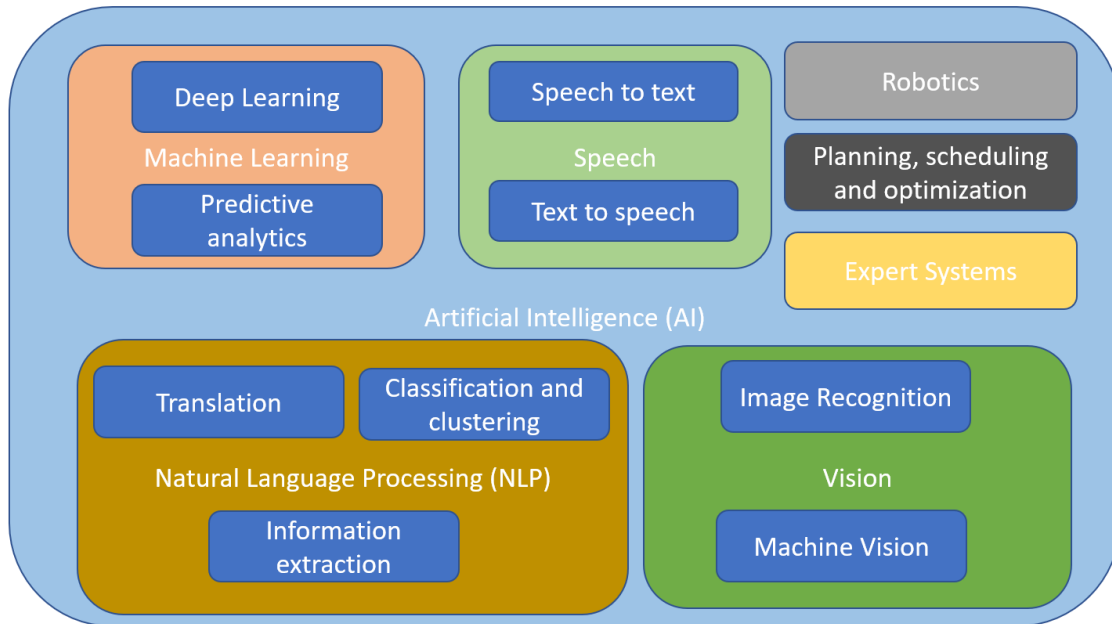


Figure 10 Artificial intelligence disciplines map

AI, Machine Learning and Deep Learning are many times confused each other. Although, Deep Learning is a branch of Machine Learning and Machine learning is a branch of Artificial Intelligence as we can see in Figure 10 [20]. Artificial intelligence is the discipline that develop programs that achieve solutions analogically to the humans. On the other hand, Machine and Deep Learning are methods that solve problems that need to replicate the process of human learning.

3.1.1. Machine Learning

The first formal development of neural networks started in 1943 by McCulloch and Pitts that created models of biological neurons using circuits that can perform computational tasks [24]. Continuing with his work in 1969 Minsky and Paper published a book called *Perceptrons* where they described this model but also showed some disadvantages that demotivated the effort. In the early eighties a new method to train the neurons called back-propagation was discovered and the new development of more powerful computers renewed the interest for such topic.

Artificial Neural Networks is one of the mightiest models to learn, to generalize and to cluster. They have also the capacity to be computed in parallel. Nevertheless, the lack of knowledge that we still have about the neurons in the biological systems limits the development of artificial neural networks to an oversimplified version of the biological counterpart [24]. We can understand a neural network as a bunch of processing units called neurons that send signals each other over weighted connections. Then each unit receives a weighted input signal from a neighbor's units or external sources and processes all these inputs to compute one output that can be an input to another unit.

All the units have the same structure and functionality, but we can divide them in three types. The first type are the input units that takes the external data. We also have the output units that send signals outside the system and between those two types of units we have layers of hidden units (Figure 11).

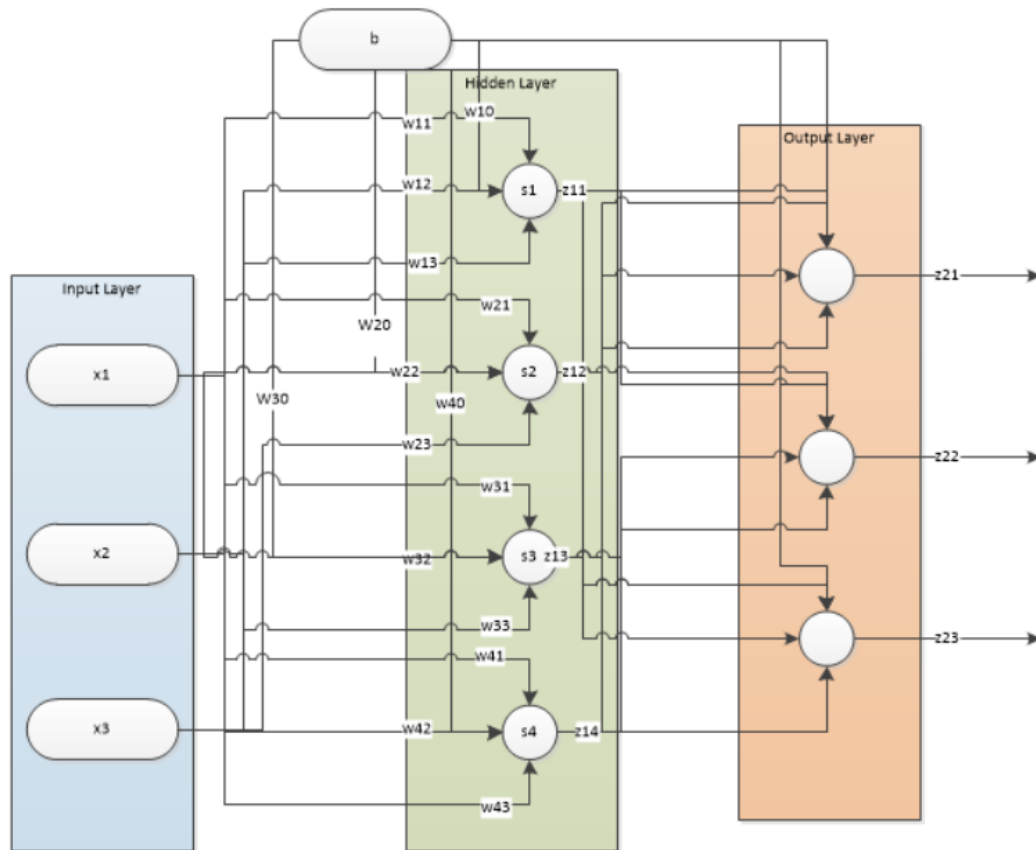


Figure 11 Neural network topology

We can consider that every input for the neuron contribute plus an offset that is called bias. Although the output of the unit is not only the addition of this weighted signals, most of the neurons have an activation function that receive this addition as input to process an output.

To train the system we need a table with the value of the desired outputs for specific inputs. The goal is to adjust the weights in order to make the predicted outputs match the desired outputs. The

difference will be considered as the error. Therefore, the weights must be adjusted in a process called *training*.

3.1.2. Deep Learning

Deep Learning is a branch of Machine learning. ML and DL are methods that achieve a solution based on learning procedures.

The basis of DL is the same than ML although we shall take in consideration that DL handles multiple hidden layers [25]. ML has achieved many impressive results the last decades but the last years some DL architectures have shown better result in some areas in specific. These results have a lot of sense if we take in consideration that IA goal is to mimic the way how humans solve problems. To solve problems the human process the information with the brain that has billions of neurons, then it is not a surprise that some problems requires more neurons to be solved [20].

When we are trying to make a trainable system to obtain a binary deduction according to a bunch of inputs, then ML is enough to compute the solution. But many of the activities that realize the human are quite more complex than that. Our brain is a marvel and even when we are doing something that seems to be easy for us, like catching a ball, it is a very complex task if we analyze all the small task that the brain had to perform. In this example, the brain first shall identify that the thing that is approaching is a ball and determine how far is the objective from the user position. In addition, it must infer that it is approaching and send signals to the actuators to place the hand in a place that cross the trajectory of the ball. This for sure is an activity that we can perform without suffering a headache but behind this simple task the brain is stimulating many neurons to make this happen and that is basically the philosophy of DL.

Deep Learning then can be used for task that cannot be achieved with simple architectures of ML and that are elementary for a human like: image recognition, emotions predictions, process of natural language and regression of complex functions.

3.2. Convolutional Neural Network (CNN)

3.2.1. Introduction

When talking about image recognition prior works were based on hand-designed features that gather important information form the image. This method ignored many unnecessary information that comes with the input itself and make the task easier to process [26]. After having selected manually the features a NN fully connected is trained to categorize the image into the corresponding class. The CNN is a system that avoid the necessity to gather some hand-designed features. The CNNs let the backpropagation training generate the feature extractor that takes the features from the images.

CNNs are part of the DL methodologies and follows the same logic that ML but including more hidden layers. Then in theory we can achieve this task of obtaining the feature extractor just creating a highly dense fully connected network. Nevertheless, the complexity of some problems makes this not possible or quite complicated just using a fully connected architecture. For example, images and spoken words are heavy inputs with many variables (every channel of every pixel is an input for example). If we consider a hidden layer of hundreds of neurons that will mean thousands of weights to be processed just

for one layer and we have established that many layers are going to be needed. This will impact directly in the memory of our system [26].

Another important aspect is that it is complicated to normalize the information to make the inputs homogeneous for the NN. For example, if we consider written characters or a spoken word, we will easily understand that it is difficult to find a centered point and a normalization value. We can imagine that many variations between an input and other will appear. Although the NN will be adapted according to the inputs by training, nevertheless this will require to have almost infinite dataset with all the possible variations to have an acceptable accuracy. CNNs are not affected by this phenomenon because the CNNs looks for the same patterns along the entire image then it doesn't matter if it is not perfectly centered [26].

Inputs like images or speeches are sequences of information that are strongly correlated in space or time. That means that we cannot change the order of the variables, otherwise the information will not have sense. The fully connected architectures don't handle this correlation, on the contrary they are fixed and expect the input to be passed always in the same order. This take us to other important factor that is the possibility to take local features instead of variable features that are the ones that are going to be provided by a fully connected network. In this case CNN works getting local features of the entire input [26].

3.2.2. Convolutional Network

The CNN architecture is constructed based in three principles:

- * Local receptive fields.
- * Shared weights.
- * Sub-sampling.

3.2.2.1. Local receptive fields and Shared weights

Local receptive fields help the neurons to extract features as oriented edges, end points and corners. These features are going to be combined along the layers [26]. The weights are part of the features extractor and one feature that can be useful in one region of the image can be useful in other. Then sharing the same weights to extract the same feature from other part of the image is also useful in other. In addition, it has the advantage that the process can be parallelized. The output of this feature extractor is going to be a set of feature maps. In other words, the neurons take a specific local receptive field determined by the weights and place the output into a feature map sequentially. This operation of multiplying the input by a kernel (weights) to obtain an output can be represented as a convolution and that is why this architecture has that name.

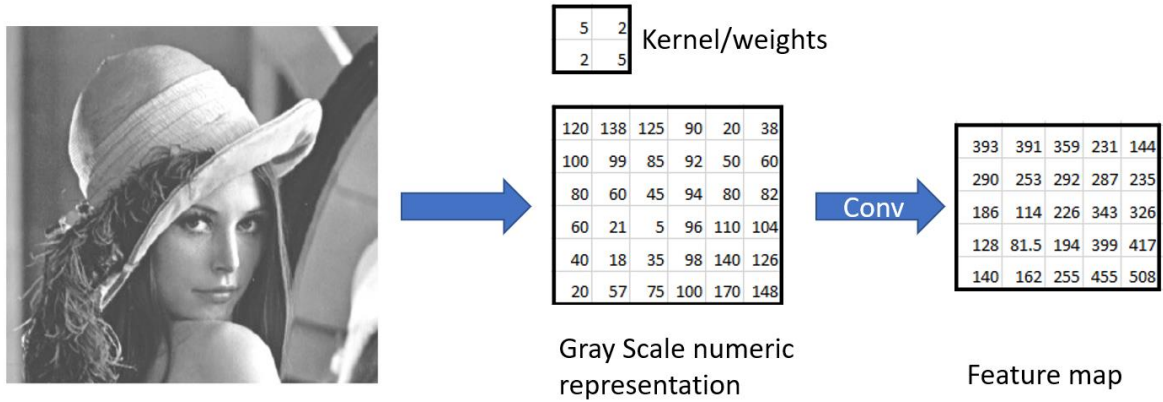


Figure 12 Representation of a convolution over a gray representation of an image

As we can see in Figure 12, the same kernel or weights are applied over all the image, in other words the weights are shared at every location of 4 variables. We can also notice that the local field is a group of four variables all together and after the operation we obtain only one feature for that location. All the features together complete the feature map. In the Figure 12 we can see one example of convolution layer with only one feature extractor (kernel) that generates one feature map but normally a convolution layer has many feature extractors that generate several feature maps.

3.2.2.2. Sub-sampling

After having the features map, we will notice that near locations features represent near input locations as well, then an average of near features can be performed and a subsampling as well. This will reduce the resolution of the feature map but will also reduce the sensibility to shifts and distortions [26]. If we continue adding this set of layers convolution plus average-subsampling the effect that we will see is that the number feature maps will keep growing and the spatial resolution will decrease until we end up with a spatial resolution of 1x1. In the following example (Figure 13) proposed by LeCun in 1990 we can see this behavior [26]. Note that the feature is extracted locally from the *same location of all the input feature maps or input* and not only one, then that means that we can extend this example if we use 3 or more channels as input.

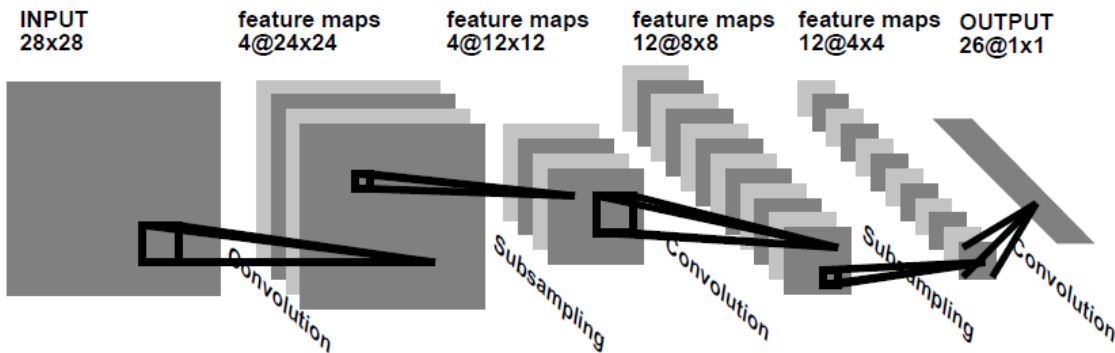


Figure 13 CNN sequence proposed by LeCun

3.2.2.3. Fully connected layer

In ML the neural networks layers are stacked one over the other. The layers are named input, output and hidden layers. The output of one layer is the result of an activation function applied over linear product of the previous layer output(\mathbf{x}) [19]. Its weights(\mathbf{W}) are a matrix of size number_of_inputs_units x number_of_output_units, then we can represent the output of one layer as:

$$\text{output}(\mathbf{x}) = f(\mathbf{x}) = f(\mathbf{W}\mathbf{x}) \quad [\text{Formula 1}]$$

Generally, the output layer doesn't have any activation function [19] because we expect that layer to throw the score that we expect without any modification. Although, if we expect a probabilistic value then a softmax or a sigmoid activation function can be used [18].

Typically, the CNN architecture needs a fixed grid-structure input and will compute features but in order to achieve the classification we still need to use a classic fully connected NN. The fully connected NN will be fed then with the flattened representation 1x1 of the extracted features by the CNN [19].

3.2.2.4. CNN Learning

In the same way how the NN are trained it is expected that all the neurons get trained using backpropagation, this is how the CNNs can synthesize their own feature extractor. The characteristic of weight sharing reduces the memory requirements and add the possibility to parallelize the process [26].

3.2.3. CNN architectures

3.2.3.1. ResNet-152

In 2015 Haiming He, Ziangyu Zhang Shaoqing Ren and Jian Sun part of the Microsoft Research team reformulated the layers of the CNNs to refer to the input layers and called it residual. This change showed more accuracy when used with depth architectures. Their design won first place in many contests like ImageNet and COCO detection [8].

Taking the basic theory, stacking more and more layers will provide more features and information. Unfortunately, it was discovered that some features started to vanish. A partial solution for this problem is the normalization of the layers. Another problem is related with deep CNN architectures and is that the accuracy gets saturated and not because overfitting reasons. This is the problem that the Microsoft Research team addressed with their residual learning block (Figure 14).

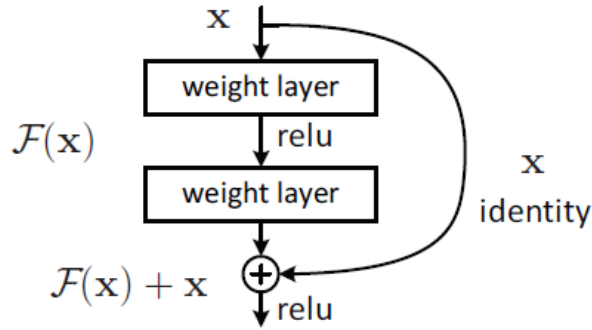


Figure 14 Residual learning block

This residual block is generated basically creating a shortcut skipping one or more layers and adding the features maps. In the Figure 14 we can see one example where the $F(x)$ is the set Conv-Relu-Conv and the output of this operation is added to the input itself. This doesn't require more parameters and the computation added is not a lot. Two observations appeared. First, training complexity doesn't increase a lot, second, the accuracy increases when the depth increased in addition the results were visible in many datasets not only one.

To make the addition it is required that $F(x)$ and x have the same dimensions and number filter, if it is not the case then we need to make a linear projection to match the dimensions. About the shortcuts we can see that the jump is over 2 conv layers, in general, they didn't saw improvement jumping only one conv layer.

To construct the architecture most of the conv layers are 3x3 filters. Two design principles were taken from VGG. First, all the feature maps with the same dimensions shall have the same number of filters, second, if the feature map size is halved then the number of filters is doubled. In this way the linear projection mentioned before is achieved with a stride of 2. The final layer is a global average pooling. To attend the vanishing a batch normalization (BN) is applied after every Conv layer.

Table 5 Resnet architectures

Layer name	Output size	18-layer	34-layer	50-layer	101-layer	152-layer
Conv1	112x112	7x7, 64, stride 2				
Conv2_x	56x56	3x3 max pool, stride 2				
		[3x3,64 3x3,64] x2	[3x3,64 3x3,64] x3	[1x1,64 3x3,64 1x1,256] x3	[1x1,64 3x3,64 1x1,256] x3	[1x1,64 3x3,64 1x1,256] x3
Conv3_x	28x28	[3x3,128 3x3,128] x2	[3x3,128 3x3,128] x4	[1x1,128 3x3,128 1x1,512] x4	[1x1,128 3x3,128 1x1,512] x4	[1x1,128 3x3,128 1x1,512] x8
Conv4_x	14x14	[3x3,256 3x3,256] x2	[3x3,256 3x3,256] x6	[1x1,256 3x3,256 1x1,1024] x6	[1x1,256 3x3,256 1x1,1024] x23	[1x1,256 3x3,256 1x1,1024] x36
Conv5_x	7x7	[3x3,512 3x3,512] x2	[3x3,512 3x3,512] x3	[1x1,512 3x3,512 1x1,2048] x3	[1x1,512 3x3,512 1x1,2048] x3	[1x1,512 3x3,512 1x1,2048] x3
	1x1	Average pool, 100-d fc, softmax				
	FLOPs	1.8x10 ⁹	3.5 x10 ⁹	3.8 x10 ⁹	7.6 x10 ⁹	11.3 x10 ⁹

34-layer residual

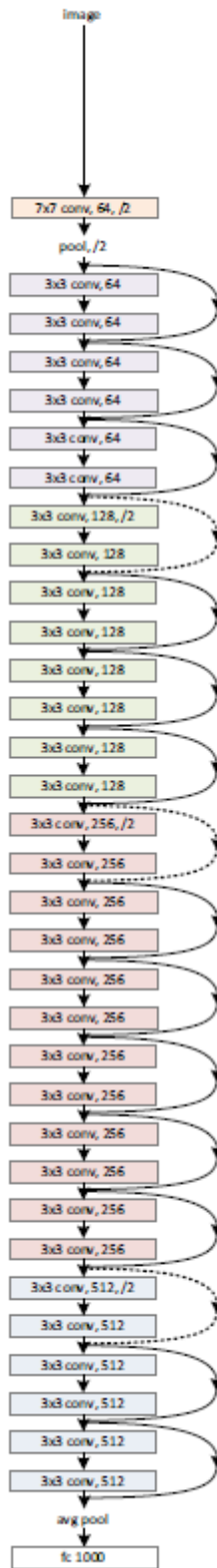


Figure 15 34-layer Resnet architecture

We can notice that for deeper architectures a sequence $1 \times 1 - 3 \times 3 - 1 \times 1$ is applied, this is called a bottleneck architecture, the first 1×1 reduces the number of feature maps to make the 3×3 less heavy and the 1×1 restore the number of feature maps.

3.2.3.2. InceptionV3

In 2016 Christian Szegedy, Vincent Vanhoucke, Sergey Ioffe, Hon Schlens and Zbigniew Wojna in a collaboration between Google Research team and University College London did a research to optimize the resources in CNN architectures. Previous works like Resnet in 2015 was achieved to increase the quality of the architectures making them deeper. Nevertheless, there are areas like mobile and big data where the memory and speed are still an important factor, then they explored a way to keep the efficiency and reduce the parameters and computation cost by factorizing the convolutional layers and aggressive regularization [9].

This new architecture was called inception and its computational cost is much lower than a VGG architecture for example. This benefit is achieved optimizing certain operations, although this adds more complexity to the architecture. Because the complexity of the architecture it is difficult to adapt or adjust.

Some design principles were used for this architecture. The first design principle is to avoid bottlenecks that extremely compress the information. It can be basically understood that we cannot lose information before a conv layer. Maybe we will think that the Resnet bottleneck block is violating this principle, but it isn't. They just exchange dimension with number of feature maps before the conv filter. Then another exchange is applied to take the feature maps to the expected quantity.

The second is the essence of the CNN and is a gradual reduction of the feature's maps size from layer to layer and the increasing of filters every layer to have more disentangled features. The third principle is reducing the feature maps before a $N \times N$ convolution. One example is the bottle neck block presented in Resnet. The last principle is to keep a balance between depth and width of the architecture, the optimal improvement of a constant amount of computation can be reached if both are increased in parallel.

The biggest modification applied in Inception architectures was the factorization of large size filters. For example, we can divide a 5×5 filter into two 3×3 filters, but we can even go further and divide the 3×3 into one 1×3 and one 3×1 filter in this way we will have 12 parameters instead of 25 that is less than the half.

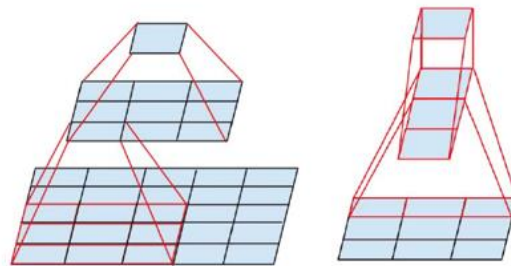


Figure 16 Left side, factorization of a 5×5 conv grid into two 3×3 conv grid. Right side: factorization of a 3×3 conv grid into one 1×3 conv grid followed by a 3×1 conv grid

In summary we can replace any $N \times N$ filter with combinations of $1 \times n$ and $n \times 1$ filters although the research saw that it has good result in medium grid layers with dimension $M \times M$ when M is between 12 and 20.

Another reduction that can be performed is to reduce the computational cost. For example, when we have k features maps as inputs with dimension d and we want to use a conv-average pool layer to reduce the dimension of the output to the half ($d/2$) and double the number of features maps ($2k$). The first step is to duplicate k with the conv layer one stride. That means that to get one output feature map it is needed to compute every single variable of the input (kd^2) and we want $2k$ number of feature maps then in total the operations for the conv layer will be $2k^2d^2$. For the average pooling the number of operations is going to be using a 2×2 window with stride of 2 and applied every feature map then the computation is $2k(d/2)^2$. If we invert the order (average pool -conv layer) we first apply the average pooling that requires $k(d/2)$ operations plus $2k^2(d/2)^2$ because the dimension has already been halved then we can see that the computation cost is almost a quarter than the former proposal. Although we are violating the first principle of this section that is avoid bottlenecks.

To avoid this mentioned loss of information produced by violating the bottleneck principle, they proposed to perform the avg pooling and the conv (with stride of 2 instead of 1) in parallel and then concatenate the results.

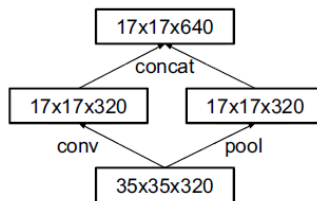


Figure 17 Inception block

Following all these design principles the architecture InceptionV3 was constructed as we can see in the Table 6

Table 6 InceptionV3 architecture

type	Patch size/stride or remarks	Input size
conv	3x3/2	299x299x3
conv	3x3/1	149x149x32
Conv padded	3x3/1	147x147x32
pool	3x3/2	147x147x64
conv	3x3/1	73x73x64
conv	3x3/2	71x71x80
conv	3x3/1	35x35x192
3xInception	Inception block 1	35x35x288
5xInception	Inception block 2	17x17x768
2xInception	Inception block 3	8x8x1280
pool	8x8	8x9x2048
Linear	Logits	1x1x2048
softmax	classifier	1x1x1000

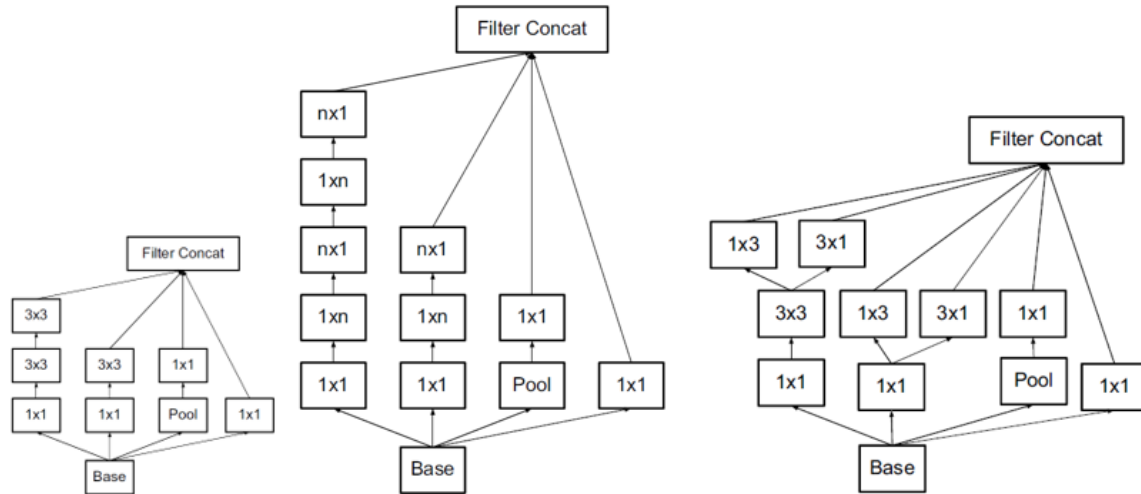


Figure 18 Left side: Inception block 1. Center: Inception block 2. Right side: Inception block 3.

As we can see in Table 6 the classical 7x7 conv layer at the input was replaced with three 3x3 layers. The rest of the layers followed the design principles explained in this section. About the size of the input experiments were done with 299x299, 251x251 and 79x79 and the best results were achieved with the input 299x299.

Table 7 Accuracy results using different input dimensions

RECEPTIVE FIELD SIZE	TOP-1 ACCURACY (SINGLE FRAME)
79X79	75.2%
151X151	76.4%
299X299	76.6%

3.2.3.3. Xception

François Chollet that is the developer of Keras inspired by the Inception architecture designed the Xception architecture. The change was basically replacing the inception modules by depth wise separable convolutions. The architecture has the same number of parameter and apparently use them more efficiently [10].

To understand the Xception hypothesis first we shall understand two concepts, cross-channel and spatial correlation. The dimensions of the filters are not assigned randomly, we rather use odd dimensions to center the result in one point, having even dimension will make difficult to assign the result of the computation of the input and map it to one output. Having a dimension over 7 will mean a costly computation and they are normally not used.

Although, there is a special dimension 1x1. This is the cross-channel correlation because the filter of dimension 1x1xk (considering k the number of input feature maps or channels) generates a linear combination of the features in one point or pixel (if we are talking about the input). For example, we can take from an image just the channel red if we apply the filter [1,0,0] and we can convert from RGB to gray

scale with the proportion $[0.2121 \cdot \text{red}, 0.7152 \cdot \text{green}, 0.722 \cdot \text{blue}]$. This two different filters or feature extractors can generate two different features maps at the output with cross-channel correlation.

On the other hand, we have the spatial correlation that is achieved with any $N \times N$ filter when N is bigger than 1. We call this spatial correlation because these filters extract spatial features like horizontal or vertical lines. Although if we apply a 1×1 immediately after a spatial correlation then we are projecting the features into a new channel space this is called depth wise separable convolutions.

The main Inception hypothesis is that cross-channel and spatial correlation are enough decoupled that is better not map them together. We can visualize this hypothesis in the Figure 18 were the last two branches only work with the 1×1 filter on the other hand the first two branches additionally execute a $N \times N$ or its fractioned version $N \times 1$ plus $1 \times N$.

Then two main changes were proposed by François. First, to use depth wise separable convolutions instead the first two branches of the Inception blocks of the Figure 18 and remove the ReLU operation that is following every spatial and cross-channel operation.

With the first change François took the Inception hypothesis even further because spatial and cross-channel correlations are now totally decoupled.

The Xception architecture then has 36 convolutional layers that have separable convolutions all with residual connections as used in Resnet excepting the first and last convolution.

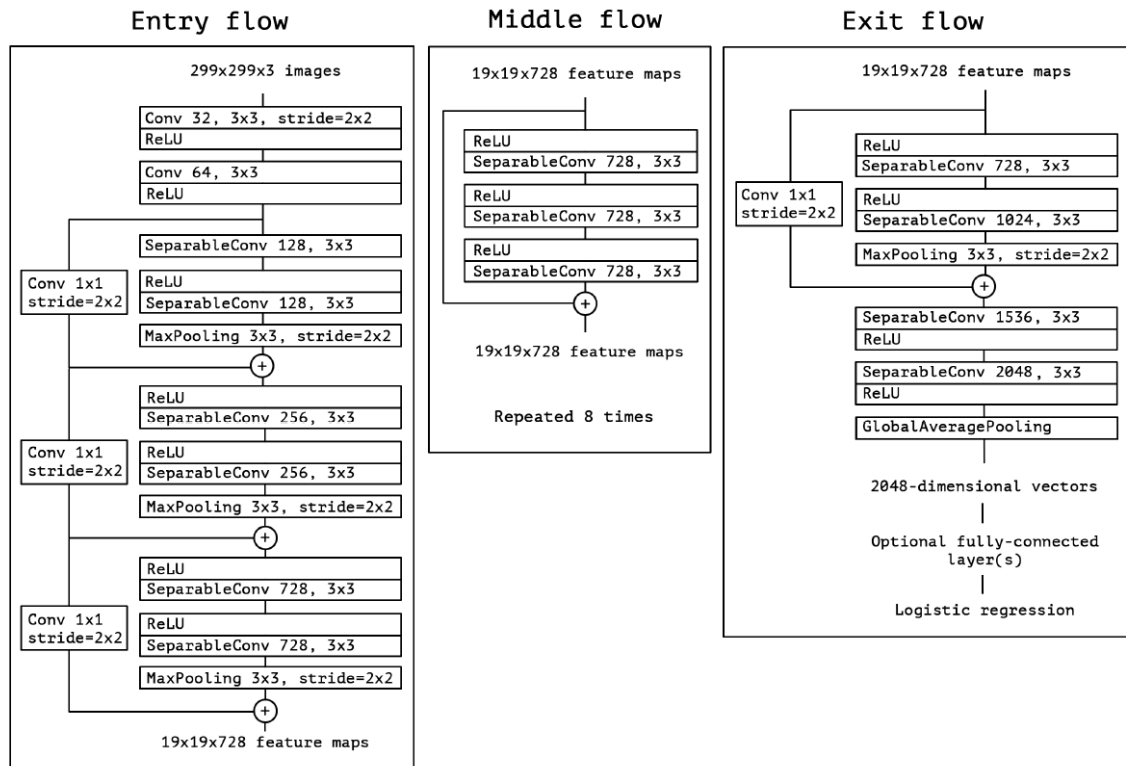


Figure 19 Xception architecture

Residual connections seem to be quite important. An efficiency analysis over imageNet dataset showed a and improvement from 60% to 80%. Removing ReLU resulted in faster convergence a better performance.

3.2.3.4. DenseNet-161

In 2017 Gao Huang, Zhuang Liu, Laurens van der Maaten and Kian Q. Weinberger as part of a collaboration between Facebook AI Research and Cornell University designed the architecture DenseNet. It has the particularity to use the feature maps of all the preceding layers as inputs with the goal to reduce the vanishing problem, strengthen feature propagation and reuse. In addition to reduce the number of parameters [11].

The vanishing problem is quite common in deep architectures. Many researches have been related to this problem and all of them share the characteristic to create short paths from previous layers. In this architecture they decided basically to connect all the layers. The difference in the shortcuts is that for example ResNet adds the previous feature map on the other hand DenseNet concatenates all the previous ones.

It is expected to require less parameters because there is no need to relearn redundant feature maps. In ResNet the information is preserved by adding although many feature maps contribute very little. This can be confirmed because in their investigation they improved the training applying dropout [8]. ResNet is also quite heavier than DenseNet. DenseNet is easy to train and reduces the overfitting with small train datasets.

Since ResNet, it was assumed that increasing the depth will increase the accuracy. There is another solution that is increasing the width adding more feature maps although DenseNet focus on the reuse. About complexity, DenseNet is less complex than Inception.

To construct the architecture in the layer L we are going to apply a transformation H_L over the concatenation of all the previous layers $x_L = H_L([x_0, x_1, \dots, x_{L-1}])$. In this case H will be a combination BN-ReLU-Conv(3x3). Nevertheless, this operation requires that all the layers have the same size or dimension. DenseNet design principle violates the main principle of the CNNs that is reduce the size of the feature maps gradually. Then the proposal solution is to construct Dense blocks that follows the described formula and between these blocks a transition block will be applied to reduce the features maps size. The transition layer will have the format BN-Conv(1x1)-average pooling (2x2). At the transition layer a data compression ($\theta = 0.5$) is also applied to reduce the number of feature maps. Finally, the growth rate, every H_L , k features map will be added.

A bottleneck layers 1x1 is applied like in ResNet and Xception. Before every H_L operation a reduction of feature maps is applied using the same format as H_L that is BN-ReLU-Conv(1x1).

Following this design principles, the DenseNet is constructed and we can see its variants in the Table 8.

Table 8 DenseNet architectures

Layers	Output size	DenseNet-121(k=32)	DenseNet-169(k=32)	DenseNet-201(k=32)	DenseNet-161(k=48)
Convolution	112x112	7x7 conv, stride 2			
Pooling	56x56	3x3 max pool, stride 2			
Dense Block (1)	56x56	[1x1 conv 3x3 conv] x 6	[1x1 conv 3x3 conv] x 6	[1x1 conv 3x3 conv] x 6	[1x1 conv 3x3 conv] x 6
Transition Block (1)	56x56	1x1 conv			
	28x28	2x2 average pool, stride 2			
Dense Block (2)	28x28	[1x1 conv 3x3 conv] x 12	[1x1 conv 3x3 conv] x 12	[1x1 conv 3x3 conv] x 12	[1x1 conv 3x3 conv] x 12
Transition Block (2)	28x28	1x1 conv			
	14x14	2x2 average pool, stride 2			
Dense Block (3)	14x14	[1x1 conv 3x3 conv] x 24	[1x1 conv 3x3 conv] x 32	[1x1 conv 3x3 conv] x 48	[1x1 conv 3x3 conv] x 36
Transition Block (3)	14x14	1x1 conv			
	7x7	2x2 average pool, stride 2			
Dense Block (4)	7x7	[1x1 conv 3x3 conv] x 16	[1x1 conv 3x3 conv] x 32	[1x1 conv 3x3 conv] x 32	[1x1 conv 3x3 conv] x 24
Classification layer	1x1	7x7 global average pool			
		1000D fully connected, softmax			

3.3. Classification Metrics

3.3.1. Accuracy

In general, accuracy is the number of items well classified over the total number of inputs. If we want to evaluate the accuracy over a class, we need to evaluate the number of items of such class well classified over the total of inputs of such class [17].

3.3.2. Precision

The precision is evaluated over the class and it is the number of items that were well classified of such class over the total of predictions that denoted this class [19][27].

3.3.3. Recall

The Recall is evaluated over the class and it is the number of items that were well classified of such class over the real number of inputs of this class [19][27].

3.3.4. F1-Score

Normally a model in the way how it gets more precision it starts reducing the recall the ideal is to have a balance. F1-Score follows the following formula:

$$\text{F1-Score} = 2 * (\text{precision} * \text{recall}) / (\text{precision} + \text{recall}) \quad [\text{Formula 2}]$$

The highest value will be F1-Score = precision = recall. In other words, F1-Score measure the valance between precision and recalls of a class prediction [28][29].

3.3.4.1. Macro-F1

Macro-F1 is the average value of all the F1-Score values of all the classes to predict [28][29].

3.3.5. Hamming Loss

Hamming Loss is the number of items wrong predicted over the total of inputs [30].

3.3.6. Jaccard Score

Jaccard Score or Jaccard similarity coefficient score is the relationship for a class between the times that it was well predicted against all the times that class appears in the classifications. For example if we have one input like {0,1,2,2} and the prediction is the following {0,2,1,2} for the class number 2 the correct predictions is just one, and the times that appear in the classification as input or as prediction is 3 then the Jaccard score for the class 2 is 1/3. We can get the average Jaccard score for all the classes and get a value for the entire model called Jaccard Score macro. Understanding this we can understand Jaccard score as the similarity between the inputs and the predictions considering 0% the minimum when inputs and predictions are totally different and 100% when the accuracy is 100% [31].

3.3.7. Log loss

Log loss or cross-entropy is a metric quite useful when the prediction is based in probabilities. In general, log loss is the error between the input and the output. Even if the accuracy is 100% if we use prediction based on probabilities there will always be a gap between the input and the output for example if have one image that is dog and a classifier that differs between cat and dogs and it says that the image is 90% dog and 10% cat, then it will be well classified but the error is of 10%. It is applied a log function to avoid handling too small numbers [32][33].

3.4. Machine and Deep Learning libraries

3.4.1. NumPy

NumPy was created in 2005 as a numerical computing library. It is an open source library that is integrated to python and the releases are approved by the scientific Python community that ensure the long-term wellbeing of the project [34].

NumPy has optimized libraries focused in vectoral and matrixial operations therefore many other libraries are based on NumPy such as TensorFlow and SKlearn. In this project we are going to be widely using such libraries then implicitly NumPy is going to be used.

3.4.1. TensorFlow and TensorFlow-GPU

TensorFlow (TF) is an opensource library released by Google in 2015 designed to create ML models [35]. TF was developed to enable the developers to easily design from a small model to complex hybrid models. It already has methods to compile and train designed models. The steps of the model can be executed one after the other without problem, this flexibility enable an intuitive debugging that is complicated to achieve in ML. If the addressed model has thousands of variables it is possible to use the parallelization capabilities of the GPU using TensorFlow-GPU, in general the models are hardware independent [36]. TF is mainly supported over Python, but it also has support over javascript(web) and mobile devices.

The compatibility with the GPU requires some drivers and libraries. To begin it is important to have a computer with NVIDIA GPU and its drivers installed. This NVIDIA GPU shall have support to CUDA 3.5 or higher [37][38].

3.4.2. Keras

Keras was developer in 2015 by Francois Chollet developer and researcher in Google AI. It is the most used Deep Learning framework. Just as TF, it is a framework focused in ML and DL although the level of abstraction is higher and more human related that TF. Initially, Chollet developed Keras for personal purposes but because its practicality and user-friendly API [39][40]. Before Keras was developed many other libraries were used for DL like Torch, Theano and Café. Nevertheless, those previous in libraries the code was a mix between assemble/C++ not easy to use, time consuming and inefficient [39]. Because those differences we can find Keras as the tool used by many winners in competitions of image recognition [40]. Originally, Keras was developed over Theano, although in the same year TF was released then the tool was migrated to work over TF and continue in this way until now [39]. Even when Keras is a complete separate development than TF some stable releases are incorporated to tensorflow libraries and we can find them under the packages *tf.keras* that ensure a complete compatibility.

The advantage that has Keras over TF is that it is a high-level API that simplifies de design of complex models that sometimes have hundreds of layers and in using TF a developer can easily get lost.

3.4.3. Tensorboard

Tensorboard is a visualization tool focused in the analysis of ML. Using this tool is possible to visualize the metric values of a model as accuracy and loss. We can even visualize the model and probabilistic analysis across the time [41]. To use Tensorboard it is required to create a callback that is going to be logging events and results along the training [42].

3.4.4. Pandas

Pandas was developed first by AQR Capital Management that is a financial enterprise in 2008. Pandas is a library designed for the data frame handling. This tool makes easy to read from some common formats as CSV, excel, SQL databases and creates data frames [43]. It even handles the read data to avoid missing data and alignment. Just as NumPy this library is optimized because critical parts of the code were developed in C. Pandas is developed in Python as a high-level API for data analysis [43].

3.4.5. SKLearn

SKLearn that is an acronym science Kit Learn was released in 2010 by Fabian Pedregosa, Gael Varoquax, Alexandre Gramfort and Vincent Michel. SKLearn is a high-level API designed over Python that has APIs related with Clustering, Composite and Covariance Estimators, metrics and other ones all related to data science [32]. The libraries have dependency of NumPy.

4. DEVELOPMENT METHODOLOGY

The investigation is focused in the comparison of four architectures proposed by Mark Pritt and Gary Chern in [18]. These four architectures are DL state of the art architectures that have shown recently good results. Mark Pritt and Gary Chern in [18] used a system that use these four architectures to classify the dataset IARPA fMoW getting an accuracy of 83%. Although, the optimization and analysis of the CNN architectures is not deeply analyzed. They just trained the architectures with one epoch of the complete dataset.

In our case we are going to evaluate the behavior and characteristics of these four architectures over the IARPA fMoW dataset. The dataset contains more than three thousand images coming from many satellites. The dataset can be downloaded in multi-spectral and RGB format, nevertheless in this investigation we used the RGB as performed in [12].

The images of the dataset come already labeled and separated in training and validation dataset, although the image sizes and different for every image. The object to classify not centered in the dataset but every image comes with a metadata that brings the bounding box.

The first part of the research is to prepare the dataset. To prepare the dataset first it is required to define a size of image that is going to be used. This size will be limited to the input layers of the CNN architectures. Then using the information that comes in the metadata the images shall be cropped to obtain centered images on the object to classify.

The complete dataset contains more than three hundred images. Working with the complete dataset will require a lot of resources and will take a long time then a good practice is to start working with a sample of the dataset. The size of the sample will be limited to the personal resources. The sample will be randomly selected. The same sample will be used over the four architectures.

The sample selected will be used to optimize the hyperparameter per architecture. Once the hyperparameters are found the architectures are going to be trained and evaluated. The metrics that are going to be obtained are: accuracy, F1-Score, Jaccard score, Hamming loss and log loss.

4.1. Requirements

System/SW Requirement	Description
SRS-1	The dataset that the Classification System shall use is the IARPA fMoW 2017 dataset.
SRS-2	The RGB version of the IARPA fMoW 2017 dataset shall be used by the Classification System.
SRS-3	The Classification System shall crop the images in squares of 299x299 pixels.
SRS-4	The Classification System shall place the object to classify centered in the cropped square.
SRS-5	The Classification System shall place all the images of the same class in the same folder.
SRS-6	The Classification System shall perform the image preprocessing for the training and validation datasets.
SRS-7	The Classification System shall generate a sample of the complete dataset.
SRS-8	The Classification System dataset sample shall contain 15 ± 1 image per class for the training dataset.
SRS-9	The Classification System dataset sample shall contain 5 ± 1 image per class for the validation dataset.
SRS-10	The Classification System dataset sample shall be selected randomly.
SRS-11	The Classification System dataset sample shall use the same dataset sample to stimulate all the architectures.
SRS-12	The Classification System shall classify the 62 classes included in the IARPA fMoW 2017 dataset.
SRS-13	The Classification System shall not classify the false detection class of IARPA fMoW 2017 dataset (This is part of the future works).
SRS-14	The Classification System shall use the architectures Resnet-152, InceptionV3, Xception and DenseNet-161 for the classification.
SRS-15	The Classification System shall use the architectures ResNet-152, InceptionV3 and Xception already implemented in the application layer of Keras library.
SRS-16	ResNet-152, InceptionV3 and Xception architecture implementations shall be adapted to handle 62 classes.
SRS-17	DenseNet-161 architecture implementation shall be constructed following the Table 8.
SRS-18	The Classification System shall document the number of features, number of params, weight and input layer size in pixels of the four architectures (Resnet-152, InceptionV3, Xception and DenseNet-161).
SRS-19	The Classification System shall use an image generator to create minibatches.
SRS-20	The Classification System image generator shall be configured to enable horizontal and vertical flip.
SRS-21	The Classification System image generator shall be configured to enable a maximum rotation of 270° degrees.
SRS-22	The Classification System image generator shall be configured to normalize the images.
SRS-23	The Classification System image generator shall be configured to enable a maximum zoom of 25%.
SRS-24	The Classification System image generator shall be configured to fill all the gaps with ceros.
SRS-25	When using the graphic board NVIDIA GTX 1650 the Classification System image generator shall be configured to use a minibatch of 10 images.
SRS-26	When using the Amazon SageMaker instance ml.p3.2xlarge the Classification System image generator shall be configured to use a minibatch of 20 images.
SRS-27	The Classification System shall determine an optimizer that shows improvements.
SRS-28	The Classification System shall find the optimal LR per architecture (Resnet-152, InceptionV3, Xception and DenseNet-161).
SRS-29	The architectures (Resnet-152, InceptionV3, Xception and DenseNet-161) shall be trained with the optimized hyperparameters.
SRS-30	The architectures (Resnet-152, InceptionV3, Xception and DenseNet-161) shall apply transfer learning from ImageNet.
SRS-31	The architectures (Resnet-152, InceptionV3, Xception and DenseNet-161) shall be trained until they reach plateau.
SRS-32	The Classification System shall report the time spent per epoch per architecture.
SRS-33	The trained architectures (Resnet-152, InceptionV3, Xception and DenseNet-161) shall be used to classify sample dataset images.
SRS-34	The Classification System shall generate a CSV for the train dataset sample that includes the: <ul style="list-style-type: none"> * Name of the file * Desired class * Predicted class * Prediction status * Percentage for the classification prediction
SRS-35	The Classification System shall generate a CSV for the validation dataset sample that includes the: <ul style="list-style-type: none"> * Name of the file * Desired class * Predicted class * Prediction status * Percentage for the classification prediction
SRS-36	The Classification System shall calculate F1-Score, Hamming Loss, Jaccard Score and log loss per architecture for the train dataset sample.
SRS-37	The Classification System shall calculate F1-Score, Hamming Loss, Jaccard Score and log loss per architecture for the validation dataset sample.

4.2. Database preprocessing

The dataset that is going to be used is IARPA fMoW that was provided as part of a contest in 2017 for satellite image classification. Many of the pictures provided by fMoW are multispectral but in this research, we will be using only three channels (RGB). It is specter to use more band as part of future researches.

The dataset can be found into an S3 public bucket [45] or it can be downloaded using torrents [46]. In general, multispectral images comes in TIFF format on the other hand RGB images comes in JPG format for this dataset. The bucket as well as the torrents are already divided in multispectral and RGB in this way we can download just the information that we want. It is important to mention that the dataset contains more than three hundred thousand of images then even the RGB dataset weights more than one hundred gigabytes.

The dataset contains more than one image of the same place. All those images of the same place are taken at different times of the day and year. In every image we can find an object of a specific class. This object is not necessarily centered in the image and every image has different size. In order to train the model, we are expecting to use images with size 299x299 pixels with the object centered in this square then an image preprocessing is needed. The input layers for the architectures that are going to be used are 224x224 pixels or 299x299 pixels, in this case we are going to use the size 299x299 pixels for all the training dataset. The models that require an input with different size are going to be handled with the minibatch generator that is going to be explained latter.

The image preprocessing will consist on centering and cropping. Every image comes with a metadata that contains general information about the image. Part of the information of the metadata are the upper left and lower right corners of the bounding box of the object inside the image. Taking this information, a centered point of the object is obtained. The cropping is performed making a square with 149 pixels at every side (left, right, up and down) from the centered point (Figure 20).



Figure 20 Image that contains an airport. Bounding box in yellow. Center point in red. Image crop square 299x299 pixels in blue.

After the image is cropped it must be placed in a folder of the class that it belongs to. This process must be repeated with every single image of the dataset to end up with 62 folders (one per class) containing images centered and cropped with the dimension 299x299 pixels. All this process was performed using python scripts. This same process is repeated with the validation dataset.

4.3. Dataset Sample

The complete dataset has more than three hundred thousand of images. Depending the hardware, the training using the entire dataset will take days or even weeks. Then a pretraining is going to be performed first using a small dataset taken from the cropped dataset. The purpose of the pretraining is to stimulate the networks using a smaller dataset to identify milestones and the best hyper-parameters to use for this project. It is also a good practice to at least check if the project is feasible and in what conditions.

For this experiment, I will use a dataset of 15 images for training and 5 for validation per class (62 classes in total). The quantity is determined by the hardware limitations. This part of the research will be performed using a graphic card NVIDIA GeForce GTX 1650 and an Amazon SageMaker instance ml.p3.2xlarge.

Once we have demonstrated that the project is feasible then it will be again executed using a framework with more capabilities in GPU RAM memory that can be provided by Amazon Web Services called SageMaker or another provider. The complete dataset is near to two-hundred times larger than we need this kind of support.

The sample was selected randomly. This same sample dataset will be used to train the four architectures.

4.4. Models creation/adaptation

The architectures that are going to be used are ResNet-152, InceptionV3, Xception and DensNet-161. The number of features that are going to be extracted with the architectures can be appreciated in Table 9. It is visible that the number of parameters is quite similar therefore the research will evaluate witch architecture is more efficient extracting valuable features.

On the other hand, the number of parameters of these architectures is not the same (Figure 9). We can notice that the number of params of the architecture ResNet-152 is significantly higher than the other three architectures. This will directly impact the size of the model and the quantity of memory that it consumes to be stored and to be trained.

ResNet-152, InceptionV3 and Xception can be found implemented as part of Keras application libraries. They are implemented canonically, that means that the output layer is designed to classify one thousand different classes. That means that the output layers must be adapted for this specific dataset that has 62 classes only.

On the other hand, there is no implementation of DenseNet-161 into Keras application library then it was created from scratch. For this DenseNet-161 creation first it was needed the sequence for conv layers mentioned in section 3.2.3.4 (BN-ReLU-Conv). This sequence is built with a batch normalization followed by a ReLU operation and finally a convolution. Having this basic function performed necessary for all the DenseNets architectures we proceed with the dense block and transition block functions creation. These two blocks are the main elements for the DenseNet architecture construction. The dense block is a repeated sequence of layers BN-ReLU-Conv(1x1) followed by BN-ReLU-Conv(3x3). The first BN-ReLU-Conv operation of the dense block expand the number of feature maps by 4 and the second one takes them back to the initial number of feature maps passed to the dense block. Every repetition the feature maps are concatenated. The first executes a sequence BN-ReLU-Conv(1x1) that will reduce the number of feature maps then performs an average pool with stride of 2 to reduce the size of the feature maps to the half. After having this main two blocks represented in functions the sequence described in Table 8 is programmed.

Table 9 Params and Memory characteristics of the architectures

ARCHITECTURE	NO. OF FEATURES	TOTAL OF PARAMS	SIZE (MB)	INPUT LAYER
RESNET-152	2048	58,497,982	229.7	224x224x3
INCEPTIONV3	2048	21,929,822	86.352	299x299x3
EXCEPTION	2048	20,875,823	82.293	299x299x3
DENSENET-161	2208	26,820,062	106.05	224x224x3

4.5. Minibatch generation

The training is executed using the “fit” method that is already provided by the Keras library. This method provided the flexibility to manage the quantity of images that are used to train every epoch. This small portion of images taken per epoch is called minibatch. The minibatch will be generated using the method ImageDataGenerator that is also provided by Keras library. This method not only provide the possibility to generate minibatches taking images randomly from a dataset, it also has the capability to scale, rotate and flip the images. The configuration of this image generated will be based on the dataset augmentation described in section 2.3.3. In this section is was explained that the dataset was augmented by eight by rotating and flipping the image. In our case instead of augmenting the dataset we are going to let the generator perform random horizontal and vertical flipping of the image as well as a random rotation range of 270 degrees.

The images come in JPG format, that means that every pixel has three values (RGB) that goes from 0 to 255. Nevertheless, the models are expecting to be normalized (from 0 to 1) information at the input, then the image generator is also used to scale the data to normalize the values. In addition, a random zoom range of 25% will be enables and all the gaps generated will be filled in with ceros.

The size of the minibatch is limited by the hardware. When working with the graphics board NVIDIA GTX 1650 the batch size was limited to 10 images. It is because when the GPU is used, we get the full capability to parallelize the process. Parallelize means that we can execute the model over every one of the images of the minibatch at the same time, but the pay back is in memory. We are basically creating one model per image in the minibatch. The heaviest architecture is ResNet-152, it weights near to 230 MBs (Figure 9), using a minibatch of 10 images will require near to 2.3GBs of RAM memory. Every epoch the model stores another 2MBs for metrics and logs and we would like to have the capability to

execute at least 200 epochs, that adds another 400MBs. The graphic board NVIDIA GTX 1650 has only 4GBs of RAM memory and it is only allowed to use the 75% of the memory (3GBs). Using a minibatch size of 10 lets only near to 300MBs Free for the GPU. The Amazon SageMaker instance ml.p3.2xlarge has 8GBs of memory then to simplify the calculus when using this instance we will use a minibatch size of 20 Images.

4.6. Hyperparameters optimization

The hyperparameter optimization was obtained using the dataset sample in to reduce the time and computation. The optimizer used was Adam. We tried to use descendent gradient, but it was not too efficient. Then, there is just one hyper-parameters to find, it is the learning rate. At this point we are going to avoid other hyperparameters as dropout or modify the learning rate during the training.

The learning rate optimization was performed using the library Tensorboard and the Tensorboard callback provided by Keras. First a code was developed to train the models two hundred epochs starting from a learning rate of 0.5 and divide such learning rate by ten every twenty epochs to analyze the learning rate window from 0.5 to 5×10^{-10} . The result will be analyzed to check the window were the model shows learning to then focus the analysis in such area. In the case of DenseNet-161 it was required to modify the learning rate every forty epochs because any visible change appeared using just 20 epochs.

Fit function provided by Keras has the possibility to iterate the training over the same minibatch many times before considering the epoch finished. This iteration number is configured as steps in the Fit function. In our case we configured 4 steps. Smaller steps made the accuracy to increase slower and more than 4 steps did not show to much difference.

All this analysis will be performed using only seven classes of the dataset sample because at this point the purpose is not the classification itself, we just want to find the best learning rate per architecture.

After having the results, they are analyzed using Tensorboard and smoothing the graph as much as possible to check were the accuracy increasing is clear and not noisy.

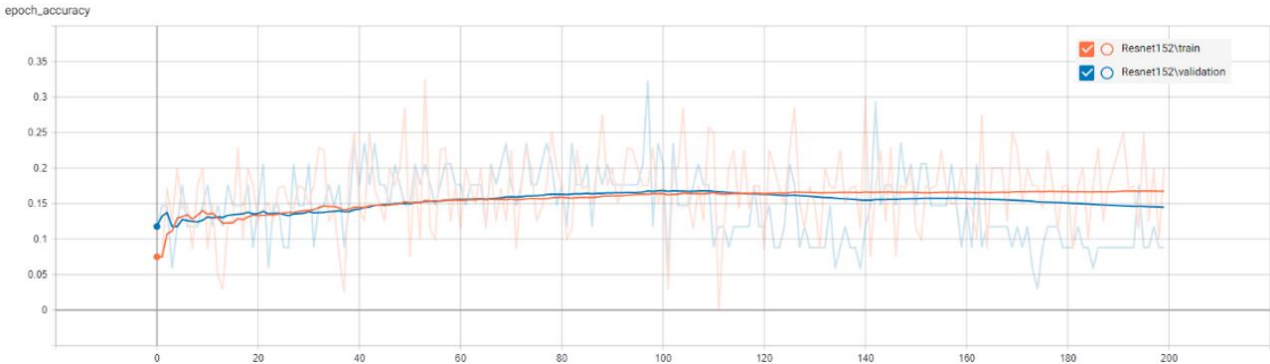


Figure 21 Accuracy of the architecture ResNet-152, with learning rate variations of 5×10^{-10} every 20 epochs.

The Figure 21 shows that the window of improvement goes from the epoch 40 to the epoch 100 that represents the learning rate 0.005 to 0.000005 for Resnet-152 architecture.

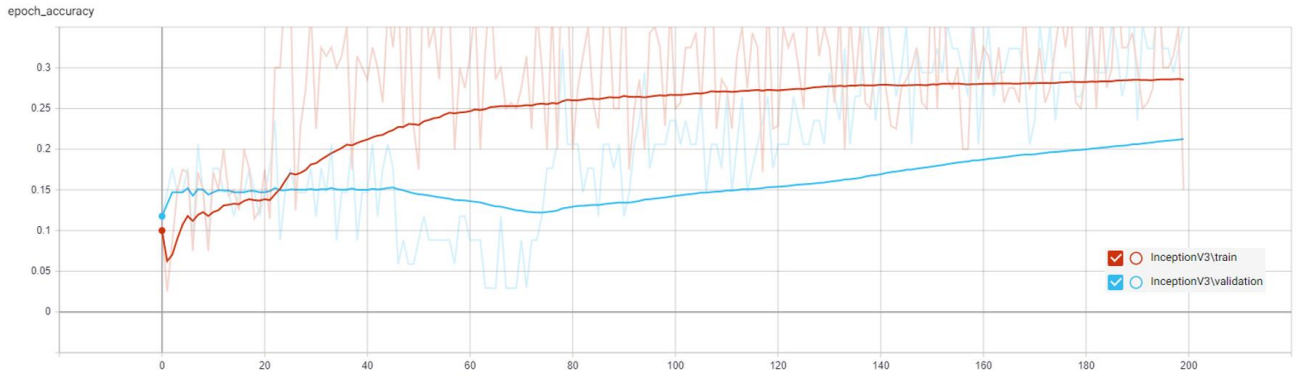


Figure 22 Accuracy of the architecture InceptionV3, with learning rate variations of 5×10^{-10} every 20 epochs.

The Figure 22 shows that the window of improvement goes from the epoch 20 to the epoch 100 that represents the learning rate 0.05 to 0.000005 for InceptionV3 architecture.

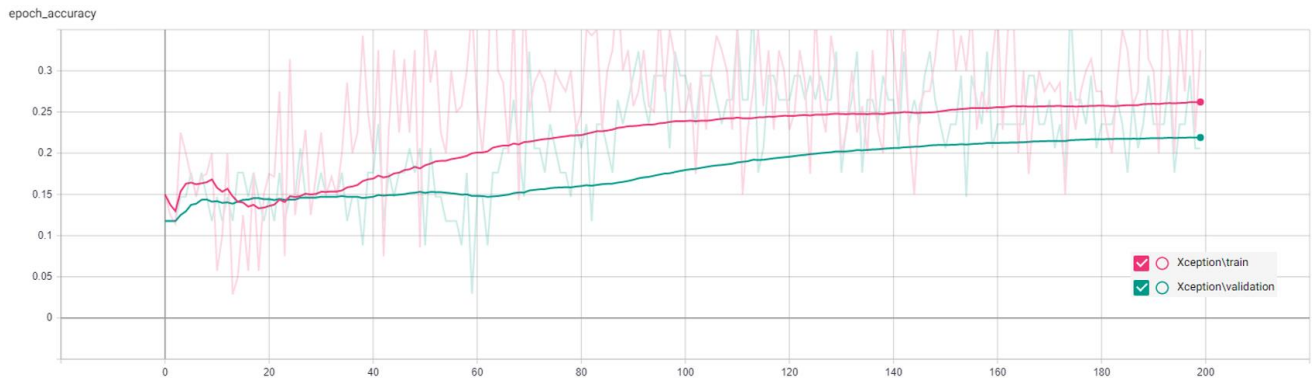


Figure 23 Accuracy of the architecture Xception, with learning rate variations of 5×10^{-10} every 20 epochs.

The Figure 23 shows that the window of improvement goes from the epoch 40 to the epoch 100 that represents the learning rate 0.005 to 0.000005 for Xception architecture.

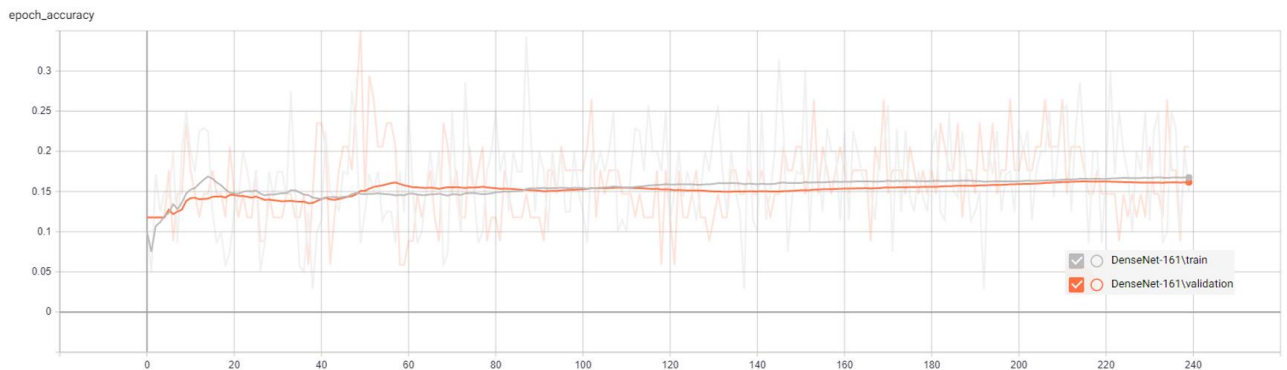


Figure 24 Accuracy of the architecture DenseNet-161, with learning rate variations of 5×10^{-10} every 40 epochs.

The Figure 24 shows that the window of improvement goes from the epoch 40 to the epoch 80 that represents the learning rate 0.005 to 0.000005 for DenseNet-161 architecture.

Table 10 LR ranges that showed accuracy improvement

ARCHITECTURE	LR RANGE FOR ACCURACY IMPROVEMENT
RESNET-152	0.005 - 0.000005
INCEPTIONV3	0.05 - 0.000005
XCEPTION	0.005 - 0.000005
DENSENET-161	0.005 - 0.00005

In the Table 10 we have found LR ranges were the CNN architectures showed some increase in accuracy. Now, instead of changing the LR we are going to perform the complete training using values inside the ranges described. After having the results, we are going to use Tensorboard and smooth the results as much as possible to identify the LR that provided the best results. At this point we are not showing accuracies over the validation dataset. The validation dataset will become meaning full when using the complete dataset.

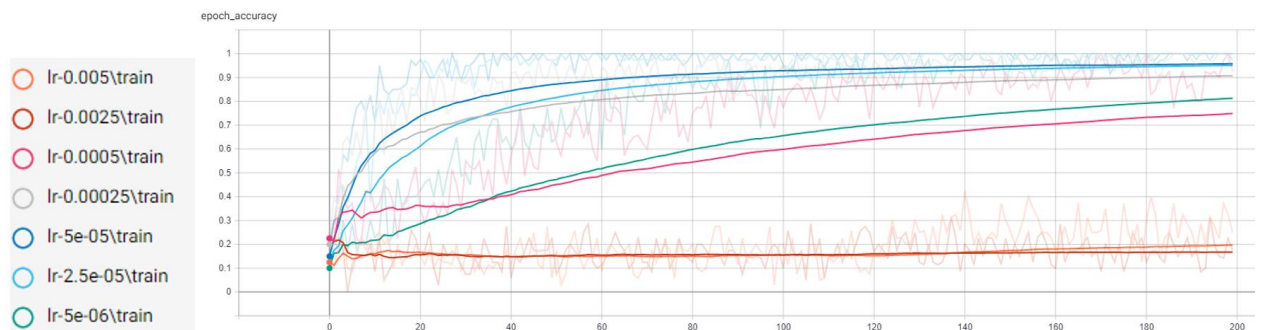


Figure 25 Accuracy over the training dataset of 7 classes using the ResNet-152 architecture

In Figure 25 we can see the accuracy of the architecture ResNet-152. The training was using only 7 classes of the dataset sample. The results are overlapped one over the other. We can check graphically that the best result was achieved with a learning rate of 0.00005.

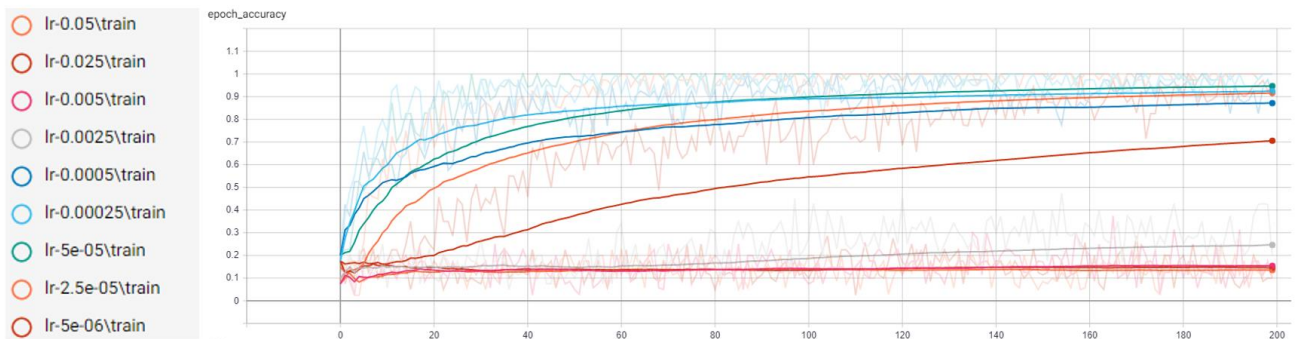


Figure 26 Accuracy over the training dataset of 7 classes using the InceptionV3 architecture

In Figure 26 we can see the accuracy of the architecture InceptionV3. The training was using only 7 classes of the dataset sample. The results are overlapped one over the other. We can check graphically that the best result was achieved with a learning rate of 0.00005.

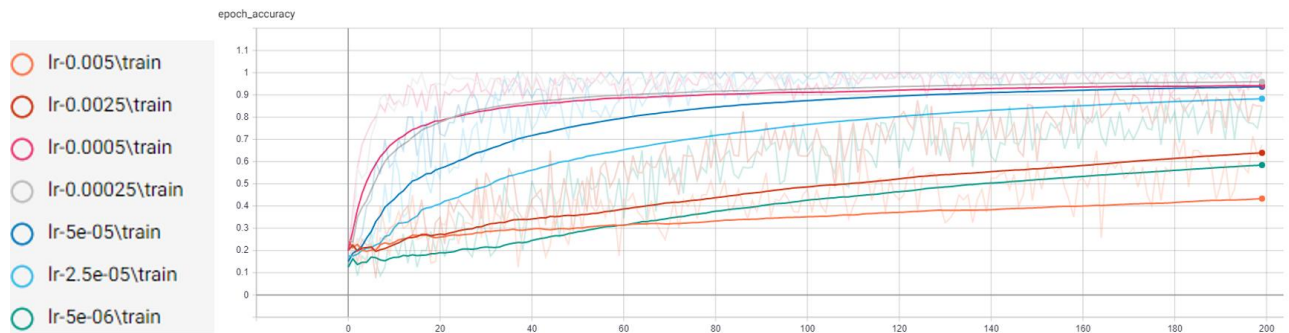


Figure 27 Accuracy over the training dataset of 7 classes using the Xception architecture

In Figure 27 we can see the accuracy of the architecture Xception. The training was using only 7 classes of the dataset sample. The results are overlapped one over the other. We can check graphically that the best result was achieved with a learning rate of 0.00025.

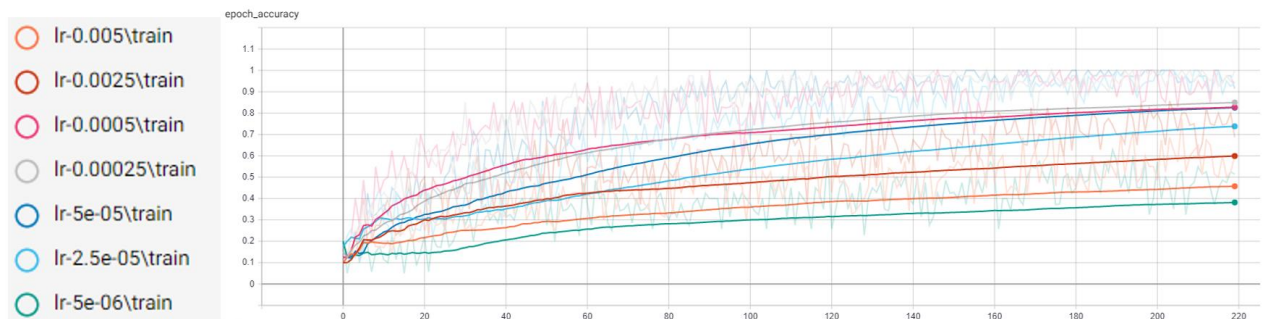


Figure 28 Accuracy over the training dataset of 7 classes using the DenseNet-161 architecture

In Figure 27 we can see the accuracy of the architecture DenseNet-161. The training was using only 7 classes of the dataset sample. The results are overlapped one over the other. We can check graphically that the best result was achieved with a learning rate of 0.00025.

Table 11 Best learning rate found using Tensorboard

ARCHITECTURE	BEST LR
RESNET-152	0.00005
INCEPTIONV3	0.00005
XCEPTION	0.00025
DENSENET-161	0.00025

4.7. Dataset Sample Training

The dataset sample contains fifteen images per class for training and five images per class for validation. The previous analysis was performed using only 7 classes that means that only 10% of the dataset sample was used. We can see in Figure 28 that two hundred epochs were not enough to reach plateau for all the architectures. Therefore, the training using the dataset sample requires more computation power. For the dataset sample training we are going to be using the Amazon SageMaker instance ml.p3.2xlarge.

To accelerate the training a minibatch of 20 images will be used. As is mentioned in [18], when the minibatch is changed some adjustments have to be used in the learning rate. In this case we are going to follow a rule to apply the same proportion added to the minibatch to the learning rate. That means that all the values of the Table 11 will be doubled for this training.

The same transfers learn strategy was applied than the one used in [18]. In such research the models received a transfer learn from ImageNet. Normally transfer learn reduces the learning process.

In Figures 25, 26 and 27 we can see that the plateau is reached almost at the epoch 120. The dataset is ten times bigger, but we are doubling the minibatch size then we are going to consider for ResNet-152, InceptionV3 and Xception seven hundred epochs. In Figure 28 we are not able to see when is reached plateau although at the epoch two hundred we can see that the accuracy increase rate is small. We are considering one thousand and two hundred epochs for the architecture DenseNet-161 that is almost the double used for the other three architectures.

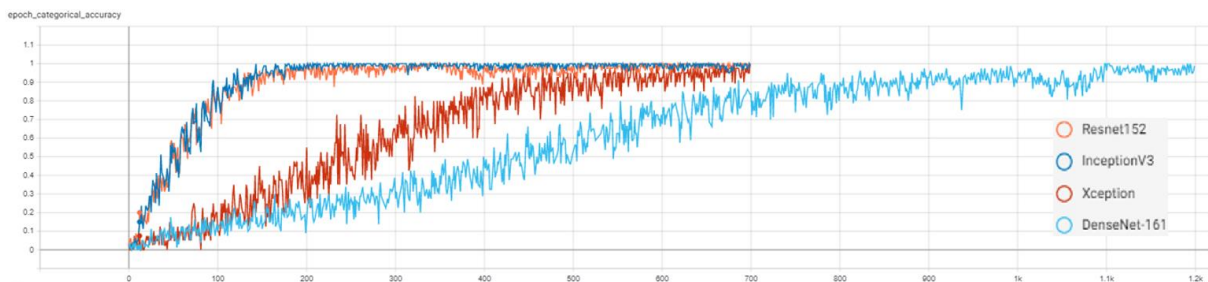


Figure 29 Accuracy vs epoch results overlapped for the training of the four architectures using the dataset sample

In the Figure 29 we can overlapped the train dataset accuracy at every epoch of the four architectures (ResNet-152, InceptionV3, Xception, DenseNet-161). We can notice that all the architectures performed outstandingly well, all of them reached accuracies higher that 90%. We can also see that the relationship epochs/accuracy is almost the same for ResNet-152 and InceptionV3 architectures. On the other hand, Xception and DenseNet-161 are taken more than three times the number of epochs that need ResNet-152 or InceptionV3.

To measure the time that takes the training is not enough knowing the number of epochs need we also need to know the time spent per epoch. For example, in Figure 29 we can notice that ResNet-152 and InceptionV3 are taking almost the same number of epochs to be trained. Although the time to train the architecture ResNet-152 will be the double than InceptionV3 because it takes the double to train every epoch (Table 12).

Table 12 Time spent per epoch

ARCHITECTURE	TIME SPENT PER EPOCH (SECONDS)
RESNET-152	8
INCEPTIONV3	4
XCEPTION	8
DENSENET-161	7

4.8. Dataset sample Evaluation

After having all the architectures trained with the dataset sample a script was developed to predict the class of the training and validation dataset. The prediction doesn't consume a lot of computer processing of the GPU and the RAM consumed is just the weight of the architecture then it can be easily performed using the NVIDIA graphic board 1650.

A script was created to perform the evaluation of the Model. This script load one trained model and generates four CSV files, two for the training dataset and two for the validation dataset. The first CSV reports the file name, its class, its predicted class its prediction status and the percentage of prediction (Figure 30).

	File	InputClass	Prediction	PredictionStatus	Percentage
0	airport_118_2_msrgb.jpg	airport	port	Incorrect	0.883416
1	airport_118_5_msrgb.jpg	airport	airport	Correct	0.999989
2	airport_118_7_msrgb.jpg	airport	airport	Correct	0.933649
3	airport_139_1_msrgb.jpg	airport	airport	Correct	0.956992
4	airport_166_8_msrgb.jpg	airport	port	Incorrect	0.566607

Figure 30 First five results reported in CSV for architecture metrics

The second CSV reports the class and its accuracy. At the end of the CSV generations the metrics

F1-Score, Hamming Loss, Jaccard Score and log loss of the architecture of both datasets (training and validation) are printed (Figure 31).

```

F1 Score: 0.8827789013863031
Hamming Loss: 0.1186623516720604
Jaccard Score: 0.7984909354221321
log loss: 0.4880079954175008
    
```

Figure 31 Metrics reported for the training dataset

Using this CSV and metrics, we can make some graphics and tables to analyze the behavior of the architectures.

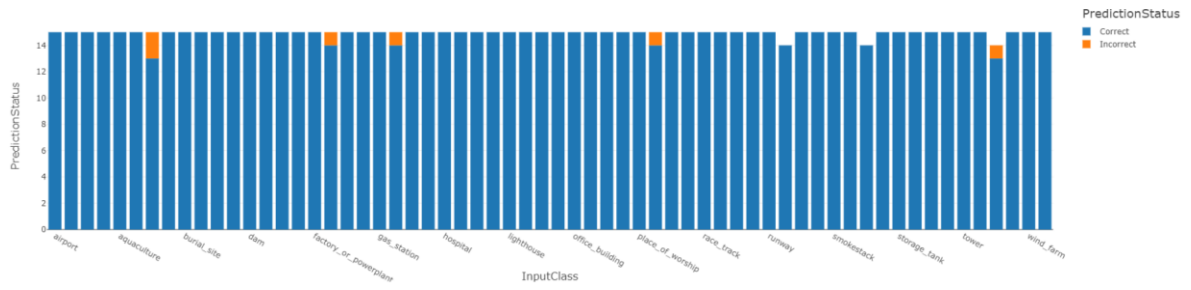


Figure 32 Correct and incorrect predictions count by the architecture ResNet-152 for the train dataset of the dataset sample

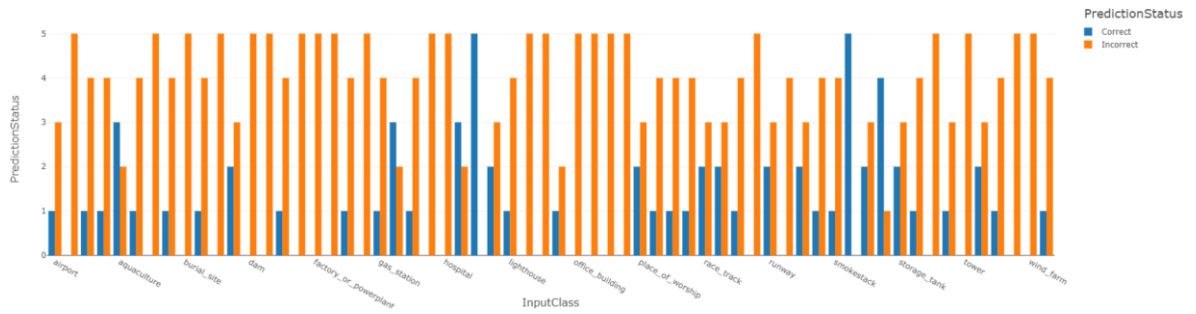


Figure 33 Correct and incorrect predictions count by the architecture ResNet-152 for the validation dataset of the dataset sample

In Figure 32 and 33 we can see the accuracy of the architecture ResNet-152 over the train and validation datasets of the dataset sample.

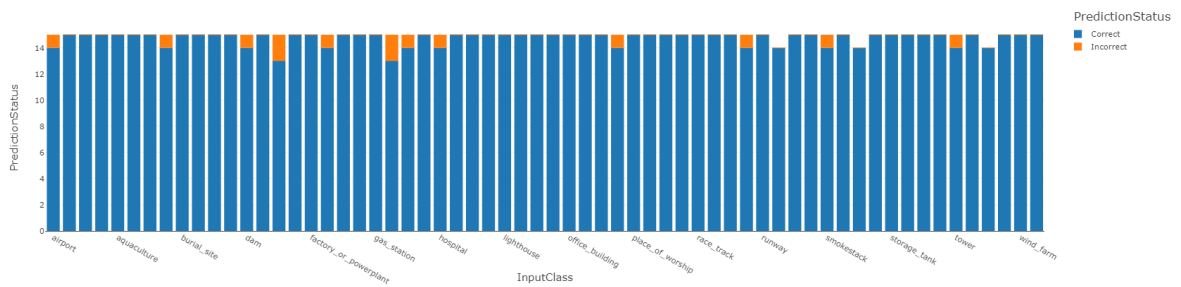


Figure 34 Correct and incorrect predictions count by the architecture InceptionV3 for the train dataset of the dataset sample

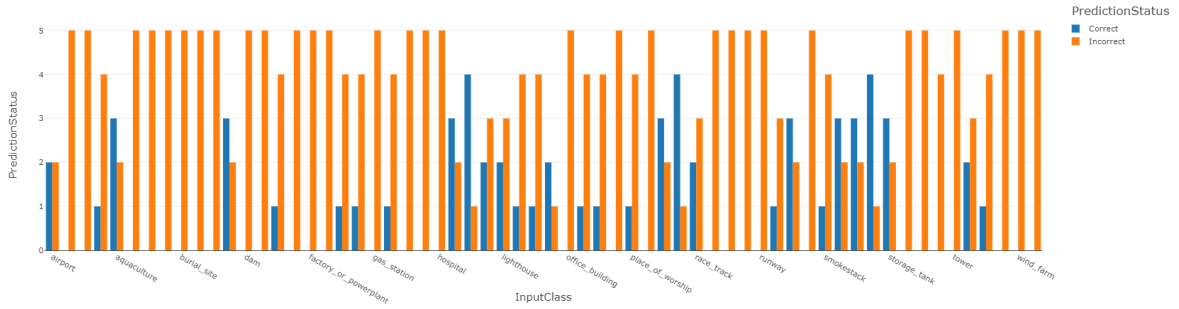


Figure 35 Correct and incorrect predictions count by the architecture InceptionV3 for the validation dataset of the dataset sample

In Figure 34 and 35 we can see the accuracy of the architecture InceptionV3 over the train and validation datasets of the dataset sample.

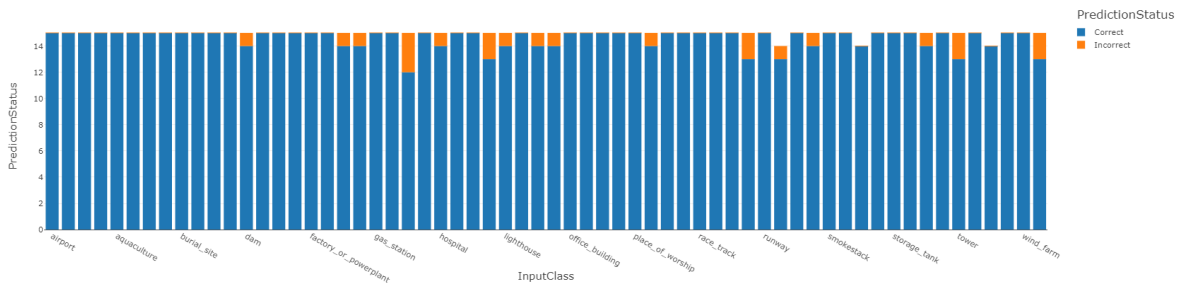


Figure 36 Correct and incorrect predictions count by the architecture Xception for the train dataset of the dataset sample

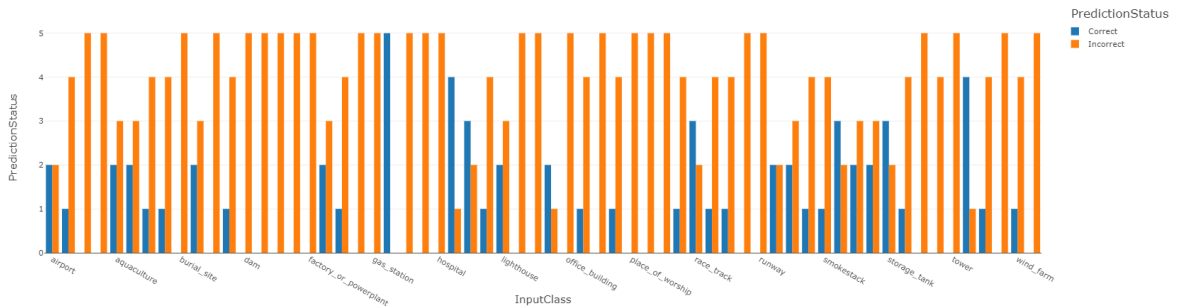


Figure 37 Correct and incorrect predictions count by the architecture Xception for the validation dataset of the dataset sample

In Figure 36 and 37 we can see the accuracy of the architecture Xception over the train and validation datasets of the dataset sample.

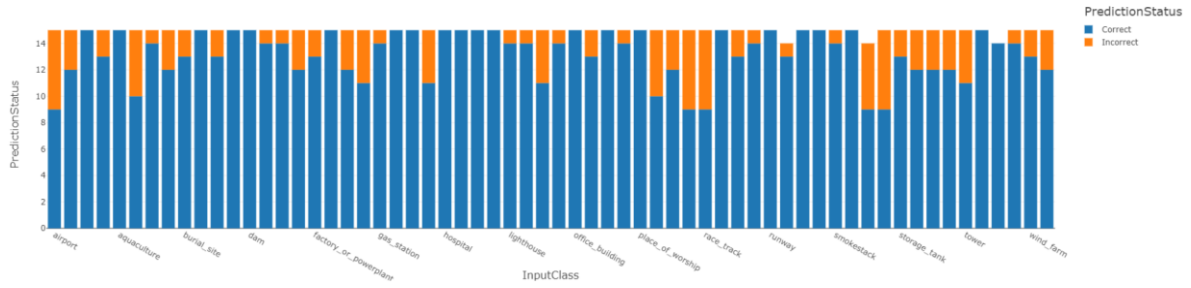


Figure 38 Correct and incorrect predictions count by the architecture DenseNet-161 for the train dataset of the dataset sample

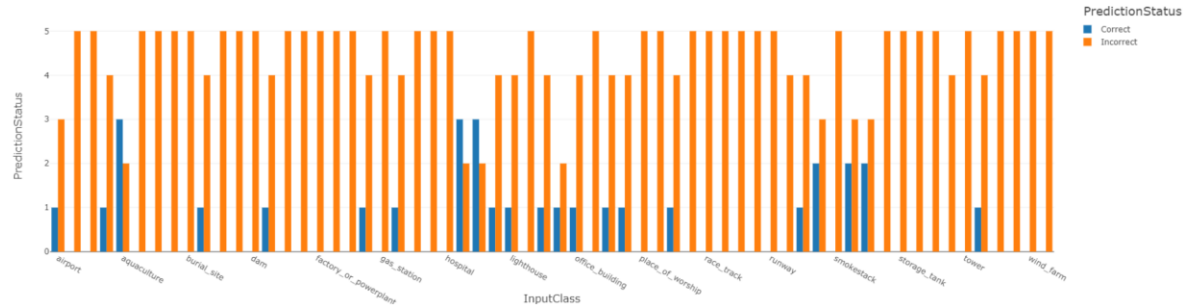


Figure 39 Correct and incorrect predictions count by the architecture DenseNet-161 for the validation dataset of the dataset sample

In Figure 38 and 39 we can see the accuracy of the architecture DenseNet-161 over the train and validation datasets of the dataset sample.

In general, we can notice that the predictions are mostly correct for the training dataset for all the architectures. For the validation dataset the predictions are mostly wrong, but it is expected to reduce this error using the complete dataset that is three hundred times larger than the sample dataset.

Table 13 Average accuracy for the dataset sample

MODEL	AVERAGE_ACCURACY (TRAINING) (HIGHER IS BETTER)	AVERAGE_ACCURACY (VALIDATION) (HIGHER IS BETTER)
RESNET152	0.9934715821812596	0.2134408602150538
INCEPTIONV3	0.9849462365591396	0.20349462365591398
EXCEPTION	0.9762672811059907	0.21075268817204296
DENSENET161	0.8812596006144391	0.10295698924731186

Table 14 Max accuracy for the dataset sample

MODEL	MAX_ACCURACY (TRAINING) (HIGHER IS BETTER)	MAX_ACCURACY (VALIDATION) (HIGHER IS BETTER)
RESNET152	1	1
INCEPTIONV3	1	0.8
EXCEPTION	1	1
DENSENET161	1	0.6

Table 15 Min accuracy for the dataset sample

MODEL	MIN_ACCURACY (TRAINING) (HIGHER IS BETTER)	MIN_ACCURACY (VALIDATION) (HIGHER IS BETTER)
RESNET152	0.8666666666666667	0
INCEPTIONV3	0.8666666666666667	0
EXCEPTION	0.8	0
DENSENET161	0.6	0

Table 16 Training Dataset accuracy per class per architecture

CLASS	RESNET-152	INCEPTIONV3	XCEPTION	DENSENET161
AIRPORT	1	0.93333333	1	0.6
AIRPORT_HANGAR	1	1	1	0.8
AIRPORT_TERMINAL	1	1	1	1
AMUSEMENT_PARK	1	1	1	0.86666667
AQUACULTURE	1	1	1	1
ARCHAEOLOGICAL_SITE	1	1	1	0.66666667
BARN	0.86666667	1	1	0.93333333
BORDER_CHECKPOINT	1	0.93333333	1	0.8
BURIAL_SITE	1	1	1	0.86666667
CAR_DEALERSHIP	1	1	1	1
CONSTRUCTION_SITE	1	1	1	0.86666667
CROP_FIELD	1	1	1	1
DAM	1	0.93333333	0.93333333	1
DEBRIS_OR_RUBBLE	1	1	1	0.93333333
EDUCATIONAL_INSTITUTION	1	0.86666667	1	0.93333333
ELECTRIC_SUBSTATION	1	1	1	0.8
FACTORY_OR_POWERPLANT	1	1	1	0.86666667
FIRE_STATION	0.93333333	0.93333333	1	1
FLOODED_ROAD	1	1	0.93333333	0.8
FOUNTAIN	1	1	0.93333333	0.73333333
GAS_STATION	1	1	1	0.93333333
GOLF_COURSE	0.93333333	0.86666667	1	1
GROUND_TRANSPORTATION_STATION	1	0.93333333	0.8	1
HELIPAD	1	1	1	0.73333333
HOSPITAL	1	0.93333333	0.93333333	1
IMPOVERISHED_SETTLEMENT	1	1	1	1
INTERCHANGE	1	1	1	1
LAKE_OR_POND	1	1	0.86666667	1
LIGHTHOUSE	1	1	0.93333333	0.93333333
MILITARY_FACILITY	1	1	1	0.93333333
MULTI-UNIT_RESIDENTIAL	1	1	0.93333333	0.73333333
NUCLEAR_POWERPLANT	1	1	0.93333333	0.93333333
OFFICE_BUILDING	1	1	1	1
OIL_OR_GAS_FACILITY	1	1	1	0.86666667
PARK	1	1	1	1
PARKING_LOT_OR_GARAGE	1	0.93333333	1	0.93333333
PLACE_OF_WORSHIP	1	1	1	1
POLICE_STATION	0.93333333	1	0.93333333	0.66666667
PORT	1	1	1	0.8
PRISON	1	1	1	0.6
RACE_TRACK	1	1	1	0.6
RAILWAY_BRIDGE	1	1	1	1
RECREATIONAL_FACILITY	1	1	1	0.86666667
ROAD_BRIDGE	1	0.93333333	0.86666667	0.93333333
RUNWAY	1	1	1	1
SHIPYARD	1	1	0.92857143	0.92857143
SHOPPING_MALL	1	1	1	1
SINGLE-UNIT_RESIDENTIAL	1	1	0.93333333	1
SMOKESTACK	1	0.93333333	1	0.93333333
SOLAR_FARM	1	1	1	1
SPACE_FACILITY	1	1	1	0.64285714
STADIUM	1	1	1	0.6
STORAGE_TANK	1	1	1	0.86666667
SURFACE_MINE	1	1	1	0.8
SWIMMING_POOL	1	1	0.93333333	0.8
TOLL_BOOTH	1	1	1	0.8
TOWER	1	0.93333333	0.86666667	0.73333333
TUNNEL_OPENING	1	1	1	1
WASTE_DISPOSAL	0.92857143	1	1	1
WATER_TREATMENT_FACILITY	1	1	1	0.93333333
WIND_FARM	1	1	1	0.86666667
ZOO	1	1	0.86666667	0.8

Table 17 Validation Dataset accuracy per class per architecture

CLASS	RESNET-152	INCEPTIONV3	XCEPTION	DENSENET161
AIRPORT	0.25	0.5	0.5	0.25
AIRPORT_HANGAR	0	0	0.2	0
AIRPORT_TERMINAL	0.2	0	0	0
AMUSEMENT_PARK	0.2	0.2	0	0.2
AQUACULTURE	0.6	0.6	0.4	0.6
ARCHAEOLOGICAL_SITE	0.2	0	0.4	0
BARN	0	0	0.2	0
BORDER_CHECKPOINT	0.2	0	0.2	0
BURIAL_SITE	0	0	0	0
CAR_DEALERSHIP	0.2	0	0.4	0.2
CONSTRUCTION_SITE	0	0	0	0
CROP_FIELD	0.4	0.6	0.2	0
DAM	0	0	0	0
DEBRIS_OR_RUBBLE	0	0	0	0.2
EDUCATIONAL_INSTITUTION	0.2	0.2	0	0
ELECTRIC_SUBSTATION	0	0	0	0
FACTORY_OR_POWERPLANT	0	0	0	0
FIRE_STATION	0	0	0.4	0
FLOODED_ROAD	0.2	0.2	0.2	0
FOUNTAIN	0	0.2	0	0.2
GAS_STATION	0.2	0	0	0
GOLF_COURSE	0.6	0.2	1	0.2
GROUND_TRANSPORTATION_STATION	0.2	0	0	0
HELIPAD	0	0	0	0
HOSPITAL	0	0	0	0
IMPOVERISHED_SETTLEMENT	0.6	0.6	0.8	0.6
INTERCHANGE	1	0.8	0.6	0.6
LAKE_OR_POND	0.4	0.4	0.2	0.2
LIGHTHOUSE	0.2	0.4	0.4	0.2
MILITARY_FACILITY	0	0.2	0	0
MULTI-UNIT_RESIDENTIAL	0	0.2	0	0.2
NUCLEAR_POWERPLANT	0.33333333	0.66666667	0.66666667	0.33333333
OFFICE_BUILDING	0	0	0	0.2
OIL_OR_GAS_FACILITY	0	0.2	0.2	0
PARK	0	0.2	0	0.2
PARKING_LOT_OR_GARAGE	0	0	0.2	0.2
PLACE_OF_WORSHIP	0.4	0.2	0	0
POLICE_STATION	0.2	0	0	0
PORT	0.2	0.6	0	0.2
PRISON	0.2	0.8	0.2	0
RACE_TRACK	0.4	0.4	0.6	0
RAILWAY_BRIDGE	0.4	0	0.2	0
RECREATIONAL_FACILITY	0.2	0	0.2	0
ROAD_BRIDGE	0	0	0	0
RUNWAY	0.4	0	0	0
SHIPYARD	0	0.25	0.5	0
SHOPPING_MALL	0.4	0.6	0.4	0.2
SINGLE-UNIT_RESIDENTIAL	0.2	0	0.2	0.4
SMOKESTACK	0.2	0.2	0.2	0
SOLAR_FARM	1	0.6	0.6	0.4
SPACE_FACILITY	0.4	0.6	0.4	0.4
STADIUM	0.8	0.8	0.4	0
STORAGE_TANK	0.4	0.6	0.6	0
SURFACE_MINE	0.2	0	0.2	0
SWIMMING_POOL	0	0	0	0
TOLL_BOOTH	0.25	0	0	0
TOWER	0	0	0	0
TUNNEL_OPENING	0.4	0.4	0.8	0.2
WASTE_DISPOSAL	0.2	0.2	0.2	0
WATER_TREATMENT_FACILITY	0	0	0	0
WIND_FARM	0	0	0.2	0
ZOO	0.2	0	0	0

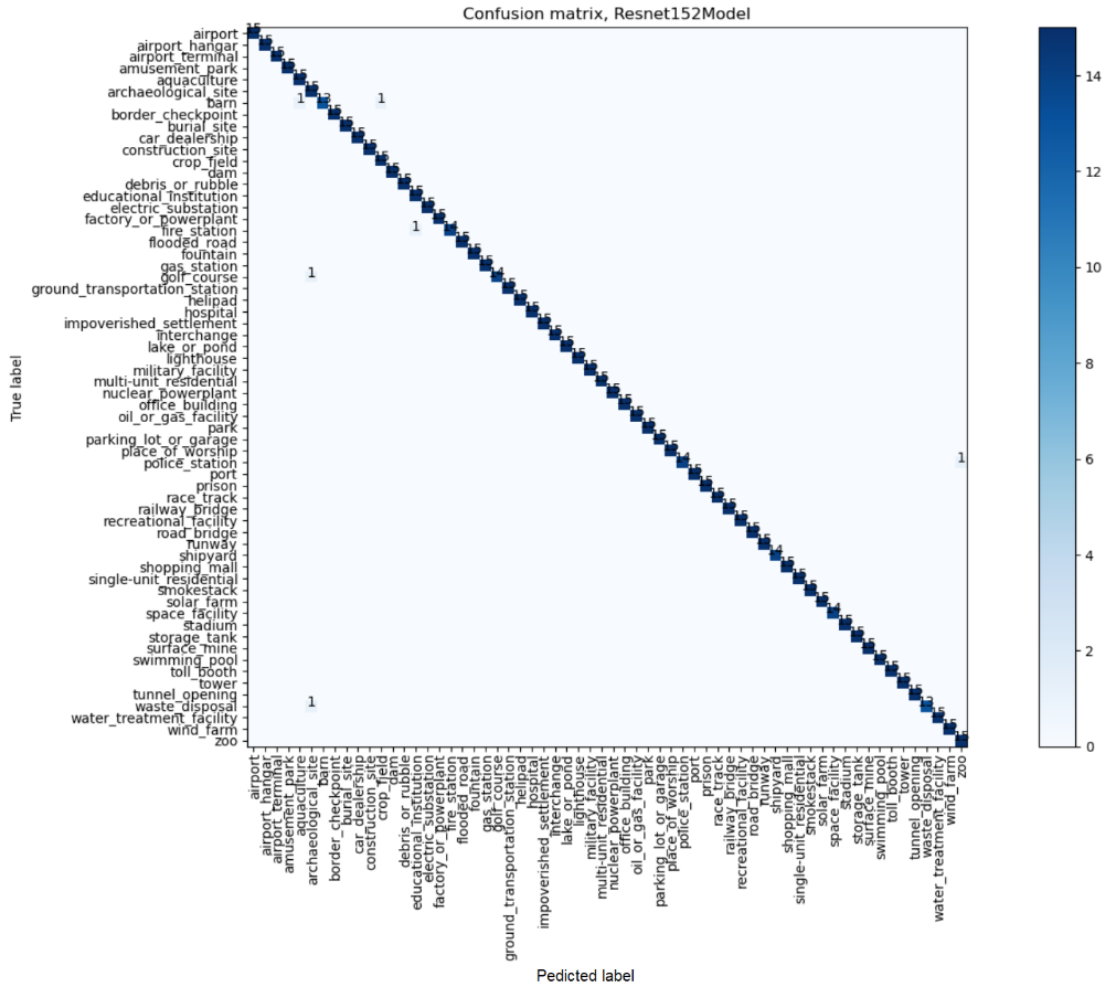


Figure 40 Confusion matrix using Resnet152 as classifier over the training dataset

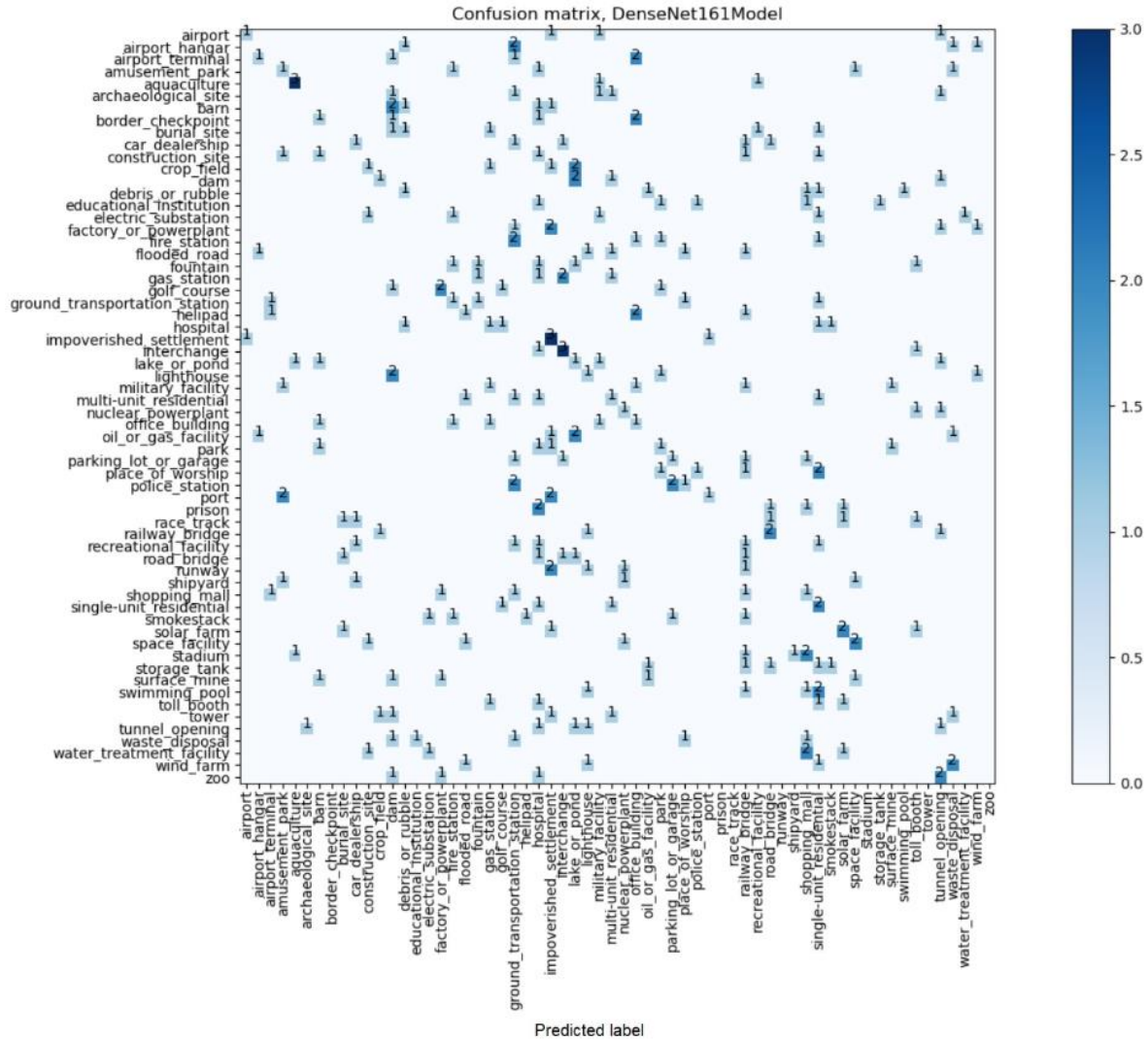


Figure 47 Confusion matrix using DenseNet161 as classifier over the validation dataset

Table 18 F1-Score for the dataset sample

MODEL	F1_SCORE (TRAINING) (HIGHER IS BETTER)	F1_SCORE (VALIDATION) (HIGHER IS BETTER)
RESNET152	0.9934928041549645	0.20485367075515398
INCEPTIONV3	0.985161983146659	0.18157992969113512
EXCEPTION	0.9763987243508819	0.1858011875326866
DENSENET161	0.8827789013863031	0.08976616145269721

Table 19 Hamming Loss for the dataset sample

MODEL	HAMMING_LOSS (TRAINING) (LOWER IS BETTER)	HAMMING_LOSS (VALIDATION) (LOWER IS BETTER)
RESNET152	0.006472491909385114	0.7868852459016393
INCEPTIONV3	0.015102481121898598	0.8
EXCEPTION	0.023732470334412083	0.7934426229508197
DENSENET161	0.1186623516720604	0.898360655737705

Table 20 Jaccard Score for the dataset sample

MODEL	JACCARD_SCORE (TRAINING) (HIGHER IS BETTER)	JACCARD_SCORE (VALIDATION) (HIGHER IS BETTER)
RESNET152	0.9875417909099123	0.13806384520081713
INCEPTIONV3	0.971991827291188	0.12111989712796163
EXCEPTION	0.9557125736542496	0.11799703827692441
DENSENET161	0.7984909354221321	0.05340519695358405

Table 21 Log loss for the dataset sample

MODEL	LOG_LOSS (TRAINING) (LOWER IS BETTER)	LOG_LOSS (VALIDATION) (LOWER IS BETTER)
RESNET152	0.024156280111023325	5.692915327122016
INCEPTIONV3	0.06519975679543337	4.413419244144481
EXCEPTION	0.1006710358155096	5.024169145817829
DENSENET161	0.4880079954175008	11.916597437710259

Some metrics are presented from Table 13 to Table 19. The metrics were obtained from the trained architectures. The metrics are values related with the accuracy and some other metrics that are widely used in the classification evaluation. We can notice that in general ResNet-152 architecture provided slightly better results than the other architectures. Although, all the architectures behave well considering the training dataset of the dataset sample. The results for the training dataset are not relevant at this point of the research, they will become important after training with the complete dataset.

4.9. Full dataset training

The full dataset contains more than three hundred thousand of images. It is quite important to train using the entire dataset to make the architectures more robust. It is expected that the gap between train average accuracy and validation average accuracy (Table 13) is reduced after training the architectures with the entire dataset. When this research was performed, the required resources to execute this training were not available then it will be performed in future works.

5. RESULTS AND DISCUSSION

The main objective of this research is to make a comparison of four architectures (ResNet-152, InceptionV3, Xception and DenseNet-161) when working with satellite images. The dataset used in this research is IARPA fMoW 2017. In section 4 the methodology to train the architectures with optimized hyperparameters is described. In addition, the evaluation of every architecture is presented based on many metrics. In this section, not only the evaluation will be analyzed but also the different characteristics that every architecture has as memory and training time.

5.1. Results

CNN architectures Resnet-152, InceptionV3, Xception and DenseNet-161 extract almost the same number of features as shown in Table 9. Although, every architecture is different, then the features extracted are going to be different.

Although the architectures extract similar numbers of features, the number of parameters used by each one is different, as shown in Table 9. Resnet-152 requires almost as twice as many parameters required by DenseNet-161 and almost three times more than the required parameters by InceptionV3 and Xception architectures. Thus, the same relationship will apply for the memory required.

As explained in section 3.2. a balance between the depth of the CNN architecture and the size of the input layer is required. Having a bigger input layer translates into a wider window to find the object to classify. But also means more compute processing. In this case, the inputs layers for InceptionV3 and Xception are nearly 30% bigger than Resnet-152 and DenseNet-161 (Table 9).

The LR range where the architectures showed improvement is not the same. A big window for LR improvement will provide more flexibility to work. Out of these four architectures, InceptionV3 has the biggest window (Table 10).

The training results were different for each architecture. As shown in figure 29, Resnet-152 and InceptionV3 architectures required a smaller number of epochs to reach plateau. They reached plateau after two hundred epochs. On the other hand, Xception and DenseNet-161 required many more epochs. Xception required nearly six hundred epochs to reach plateau and DenseNet-161 required almost one thousand and five hundred epochs.

The time spent per epoch is directly affected by the parameters to train and the complexity of the operations per layer. In this case we can see in Table 12 that InceptionV3 takes only 4 times per epoch which is almost half of the other three architectures.

We can see in Figures 32, 34, 36 and 38 that the predictions of the architectures over the training dataset sample are good overall. It is also evident that the results for Resnet-152 are outstanding. Resnet-152 missed only 6 out of 927 classifications.

In Figures 33, 35, 37 and 39 it is evident that the predictions over the validation dataset sample are quite poor. We expect improvements after training with the complete dataset.

Tables 13, 14, 15 and 17 show information related to the overall accuracy. We can notice that all the architectures had an average accuracy close to 90%. However, Resnet152 markedly showed the best accuracy, which was up to 99%.

Tables 16 and 17 provides the accuracy per class. Once again, the result for training dataset are significantly better than the ones for the validation dataset. We found that for the validation dataset many accuracies of 100% appeared for many classes.

Jaccard Score (Table 18) will tell us how similar the inputs against the predictions are. This provides a rough idea of how good the classification was. We can notice that all the architectures developed well. On the other hand, F1-Score provides us information about the distribution of the class not well classified.

In general, we can see that all the architectures got good results in such distribution. We can confirm this analyzing the confusion matrices (Figures 40 to 47). The confusion matrices show the distribution of the failures where we found that for the training dataset the distribution is quite homogeneous.

Taking in consideration that log loss (Table 21) is smaller than the other architectures, we found that the best results were the ones provided by the architecture Resnet152. Nevertheless, we cannot conclude that the other architectures developed wrong.

5.2. Discussion

Some interesting situations appeared in this research. The first thing that we can notice is that the dataset fMoW is quite complete. Satellite images datasets have been quite limited and having this kind of dataset enable the possibility to perform researches related to the classification of satellite images. The downside of this huge datasets is that more resources are required to be processed.

Having the dataset labeled helps a lot in the classification process. Although the images still need a preprocessing to adapt the images to the requirements of the CNN architectures.

Some architectures can be found as part of Keras application libraries. This is a big help because we explained in sections 3.2.3.2. and 3.2.3.3. that InceptionV3 and Xception are quite difficult to build and any adaptation is almost impossible. Resnets and DenseNets architectures are easy to build, escalate and adapt. In this research we built the DenseNet161 architecture because it was not part of the application library for Keras, but the same code can be easily adapted to create any other of the Densenet architectures.

In some development areas, like embedded systems, the memory is a limited resource. In this sort of technologies InceptionV3 and Xception architecture seems to be more suitable.

Tensorboard showed to be a power tool when optimizing the parameters. It not only stores the results of the trainings in a log that can be visualized and analyzed, it can also overlap the results of other trainings to make a more confident comparison of the results. It is also possible to smooth the graphs to reduce the noise and even make an easier comparison.

The number of epochs is not the only factor that will impact the time spent on training. The time taken by every epoch affects too. In this case we can see in Figure 29 that Resnet152 and InceptionV3 have almost the same training graph. But if we take in consideration that InceptionV3 spent half of the time per epoch (Table 12) then we can infer that the training will take the double for Resnet152.

In general, all the architectures provided better results in this research executed with a sample of the complete dataset than the ones reported in [18] that reported an accuracy of 83%. Then we cannot exclude any one of them in future works where we are expecting to use the entire dataset.

6. CONCLUSIONS

6.1. Conclusiones

- IARPA fMoW is a labeled dataset that contains enough images to stimulate the CNN architectures in order to perform classification tasks.
- The dataset IARPA fMoW provided all the information required in the metadata to be able to crop and adapt it to the CNN architectures.
- Keras libraries provided Resnet152, InceptionV3 and Xception architectures as part of its application libraries.
- Keras is flexible enough to be able to adapt the architectures.
- Keras provide also all the elements required in case that an CNN architecture must be created from scratch.
- If the dataset is too big, it is a good strategy to start working with a sample of the dataset well distributed and randomly selected.
- Tensorboard is a handy tool when optimizing the hyperparameters of architectures.
- SKLearn libraries contained the methods required to obtain the metrics F1-Score, Hamming loss, Jaccard Score and log loss.
- The accuracy was obtained with a simple code that counts the correct and incorrect predictions.
- The metrics provided a clear view of the behavior of the CNN architectures.
- The most accurate architecture was Resnet152.
- The heaviest architecture is Resnet152.
- The architectures that consume less memory are InceptionV3 and Xception.
- DenseNet architecture is easy to build and adapt.
- InceptionV3 was the architecture that was faster in training.
- In general, all the architectures had good accuracy (close to 90%) over the train sample dataset.
- In general, all the architectures had bad accuracy (close to 20%) over the validation sample dataset.
- Training over the complete dataset is required to improve the validation classification accuracy.
- The number of samples is quite small. This research provides one approach to the results that we can use using the entire dataset. It is quite important to complete the process using the entire dataset.
- When using the complete dataset all the steps performed in the section 4. Development methodology shall be executed again.

6.2. Future Works

- 1) The process followed in section 4.6. can be automated.
- 2) This investigation was performed using only the three basic bands RGB as suggested by [7]. On the other hand, [13][14] used multispectral datasets with good results then as part of future works, we are planning to use more of the bands provided by the dataset IARPA fMoW.

3) [18] is adding a class of false detection. The dataset IARPA fMoW provide this additional class in a separated dataset. We are planning to add this class to the classification process.

4) All the architectures showed better results on the training dataset sample that [18] then when having the resources enabled, we need to train all the architectures using the complete dataset.

BIBLIOGRAPHY

- [1] “Fonatur Establece El Costo Del ‘Rentable’ Tren Maya En 139,072 Millones de Pesos.” <https://www.eleconomista.com.mx/estados/Fonatur-establece-el-costo-del-rentable-Tren-Maya-en-139072-millones-de-pesos-20200108-0081.html>.
- [2] “Otorgarían 41,300 millones de pesos a Dos Bocas.” <https://www.eleconomista.com.mx/empresas/Otorgarian-41300-millones--de-pesos-a-Dos-Bocas-20190909-0020.html>.
- [3] “Análisis Del Resolutivo SGPA/DGIRA/DG/09965 Del Proyecto ‘Nuevo Aeropuerto Internacional de La Ciudad de México, S. A. de C. V.’ MIA15EM2014V0044.” https://www.uccs.mx/images/library/analisis_resolutivo_aeropuerto_uccs_2015.pdf.
- [4] “NUEVO AEROPUERTO INTERNACIONAL DE MÉXICO, UN PROYECTO INDISPENSABLE: RIESGOS Y OPORTUNIDADES.” https://imco.org.mx/nuevo-aeropuerto-internacional-mexico-proyecto-indispensable-riesgos-opportunidades/?gclid=Cj0KCQjwoub3BRC6ARIsABGhnyaN_vz4zxgkBnO6F3yAd4wnMyOKJSef3mj3_6oBdoxmFo8RlrTva7EaAk6JEALw_wcB.
- [5] “Construction Firm for Renovations to Mercedes-Benz Superdome Approved, Renderings Released, NOLA.Com Reports.” <https://www.neworleanssaints.com/news/construction-firm-for-renovations-to-mercedes-benz-superdome-approved-renderings>.
- [6] “Large Scale Visual Recognition Challenge (ILSVRC).” <http://www.image-net.org/challenges/LSVRC/>.
- [7] M. Pritt and G. Chern, “Satellite Image Classification with Deep Learning,” in *2017 IEEE Applied Imagery Pattern Recognition Workshop (AIPR)*, Washington, DC, USA, Oct. 2017, pp. 1–7, doi: [10.1109/AIPR.2017.8457969](https://doi.org/10.1109/AIPR.2017.8457969).
- [8] K. He, X. Zhang, S. Ren, and J. Sun, “Deep Residual Learning for Image Recognition,” *arXiv:1512.03385 [cs]*, Dec. 2015, Accessed: Aug. 01, 2020. [Online]. Available: <http://arxiv.org/abs/1512.03385>.
- [9] C. Szegedy, V. Vanhoucke, S. Ioffe, J. Shlens, and Z. Wojna, “Rethinking the Inception Architecture for Computer Vision,” in *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, Las Vegas, NV, USA, Jun. 2016, pp. 2818–2826, doi: [10.1109/CVPR.2016.308](https://doi.org/10.1109/CVPR.2016.308).
- [10] F. Chollet, “Xception: Deep Learning with Depthwise Separable Convolutions,” *arXiv:1610.02357 [cs]*, Apr. 2017, Accessed: Aug. 01, 2020. [Online]. Available: <http://arxiv.org/abs/1610.02357>.
- [11] G. Huang, Z. Liu, L. Van Der Maaten, and K. Q. Weinberger, “Densely Connected Convolutional Networks,” in *2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, Honolulu, HI, Jul. 2017, pp. 2261–2269, doi: [10.1109/CVPR.2017.243](https://doi.org/10.1109/CVPR.2017.243).

- [12] Christie, N. Fendley, J. Wilson, and R. Mukherjee, "Functional Map of the World," *arXiv:1711.07846 [cs]*, Apr. 2018, Accessed: Aug. 01, 2020. [Online]. Available: <http://arxiv.org/abs/1711.07846>.
- [13] R. J. Aroma and K. Raimond, "A novel two-tier paradigm for labeling water bodies in supervised satellite image classification," in *2017 International Conference on Signal Processing and Communication (ICSPC)*, Coimbatore, Jul. 2017, pp. 384–388, doi: [10.1109/CSPC.2017.8305875](https://doi.org/10.1109/CSPC.2017.8305875).
- [14] C. D. Storie and C. J. Henry, "Deep Learning Neural Networks for Land Use Land Cover Mapping," in *IGARSS 2018 - 2018 IEEE International Geoscience and Remote Sensing Symposium*, Valencia, Jul. 2018, pp. 3445–3448, doi: [10.1109/IGARSS.2018.8518619](https://doi.org/10.1109/IGARSS.2018.8518619).
- [15] K. Wohlfarth *et al.*, "Dense Cloud Classification on Multispectral Satellite Imagery," in *2018 10th IAPR Workshop on Pattern Recognition in Remote Sensing (PRRS)*, Beijing, Aug. 2018, pp. 1–6, doi: [10.1109/PRRS.2018.8486379](https://doi.org/10.1109/PRRS.2018.8486379).
- [16] J. Treboux and D. Genoud, "Improved Machine Learning Methodology for High Precision Agriculture," in *2018 Global Internet of Things Summit (GIoTS)*, Bilbao, Jun. 2018, pp. 1–6, doi: [10.1109/GIOTS.2018.8534558](https://doi.org/10.1109/GIOTS.2018.8534558).
- [17] A. Perez-Suay, J. Amoros-Lopez, L. Gomez-Chova, J. Munoz-Mari, D. Just, and G. Camps-Valls, "Pattern Recognition Scheme for Large-Scale Cloud Detection Over Landmarks," *IEEE J. Sel. Top. Appl. Earth Observations Remote Sensing*, vol. 11, no. 11, pp. 3977–3987, Nov. 2018, doi: [10.1109/JSTARS.2018.2863383](https://doi.org/10.1109/JSTARS.2018.2863383).
- [18] M. Pritt and G. Chern, "Satellite Image Classification with Deep Learning," in *2017 IEEE Applied Imagery Pattern Recognition Workshop (AIPR)*, Washington, DC, USA, Oct. 2017, pp. 1–7, doi: [10.1109/AIPR.2017.8457969](https://doi.org/10.1109/AIPR.2017.8457969).
- [19] M. Le Goff, J.-Y. Tournet, H. Wendt, M. Ortner, and M. Spigai, "Deep Learning for Cloud Detection," in *8th International Conference of Pattern Recognition Systems (ICPRS 2017)*, Madrid, Spain, 2017, p. 10 (6.)-10 (6.), doi: [10.1049/cp.2017.0139](https://doi.org/10.1049/cp.2017.0139).
- [20] "What's the Difference Between AI, Machine Learning, and Deep Learning?" <https://blogs.oracle.com/bigdata/difference-ai-machine-learning-deep-learning>.
- [21] "Early Robots and Automaton." <https://www.history.com/news/7-early-robots-and-automatons>.
- [22] G. Singh, A. Mishra, and D. Sagar, "SBIT JOURNAL OF SCIENCES AND TECHNOLOGY ISSN 2277-8764 VOL-2, ISSUE 1, 2013.," p. 4.
- [23] "AI Mind Map." <https://medium.com/ml-ai-study-group/ai-mind-map-a70dafcf5a48>.
- [24] "An introduction to Neural Networks." <https://www.infor.uva.es/~teodoro/neuro-intro.pdf>.
- [25] "The Next Era: Deep Learning in Pharmaceutical Research." <https://link.springer.com/article/10.1007/s11095-016-2029-7>.

- [26] “Convolutional Networks for Images, Speech, and Time-Series.” <http://yann.lecun.com/exdb/publis/pdf/lecun-bengio-95a.pdf>.
- [27] “Multi-Class Metrics Made Simple, Part I: Precision and Recall.” <https://towardsdatascience.com/multi-class-metrics-made-simple-part-i-precision-and-recall-9250280bddc2>.
- [28] “Multi-Class Metrics Made Simple, Part II: The F1-Score.” <https://towardsdatascience.com/multi-class-metrics-made-simple-part-ii-the-f1-score-ebe8b2c2ca1>.
- [29] “Sklearn.Metrics.F1_score.” https://scikit-learn.org/stable/modules/generated/sklearn.metrics.f1_score.html#sklearn.metrics.f1_score.
- [30] “Sklearn.Metrics.Hamming_loss.” https://scikit-learn.org/stable/modules/generated/sklearn.metrics.hamming_loss.html#sklearn.metrics.hamming_loss.
- [31] “Sklearn.Metrics.Jaccard_score.” https://scikit-learn.org/stable/modules/generated/sklearn.metrics.jaccard_score.html#sklearn.metrics.jaccard_score.
- [32] “Sklearn about Us.” <https://scikit-learn.org/stable/about.html>.
- [33] “Sklearn.Metrics.Log_loss.” [https://scikit-learn.org/stable/modules/generated/sklearn.metrics.log_loss.html#sklearn.metrics-log-loss](https://scikit-learn.org/stable/modules/generated/sklearn.metrics.log_loss.html#sklearn.metrics.log_loss).
- [34] “Numpy about Us.” <https://numpy.org/about/>.
- [35] “TensorFlow Is an End-to-End Open Source Platform for Machine Learning.” <https://www.tensorflow.org/overview>.
- [36] “Why TensorFlow.” <https://www.tensorflow.org/about>.
- [37] “TensorFlow GPU Support.” <https://www.tensorflow.org/install/gpu>.
- [38] “How to Run TensorFlow with GPU on Windows 10 in a Jupyter Notebook.” <http://bailiwick.io/2017/11/05/tensorflow-gpu-windows-and-jupyter/>.
- [39] “Keras vs. Tf.Keras: What’s the Difference in TensorFlow 2.0?” <https://www.pyimagesearch.com/2019/10/21/keras-vs-tf-keras-whats-the-difference-in-tensorflow-2-0/>.
- [40] “Keras Simple. Flexible. Powerful.” <https://keras.io/>.
- [41] “TensorBoard: TensorFlow’s Visualization Toolkit.” <https://www.tensorflow.org/tensorboard>.
- [42] “Tf.Keras.Callbacks.TensorBoard.” https://www.tensorflow.org/api_docs/python/tf/keras/callbacks/TensorBoard.
- [43] “About Pandas.” <https://pandas.pydata.org/about/index.html>.
- [44] “Earth on AWS: Functional Map of the World,” Amazon.com, <https://aws.amazon.com/earth/>.
- [45] “Functional Map of the World (FMoW) Dataset.” <https://github.com/fMoW/dataset>.

APPENDIX A. SageMaker Configuration

SageMaker is an AWS service designed to the creation, training and implementation of ML architectures. With this service the developer can save the time that takes to create the develop environment. At this moment the creation the creation of develop environment for ML is a complex task because there is not any integrated tool, we can only find dedicated libraries. AWS SageMaker provide stacks already configured and ready to be used.

The precondition to use SageMaker is to have an account created in AWS.

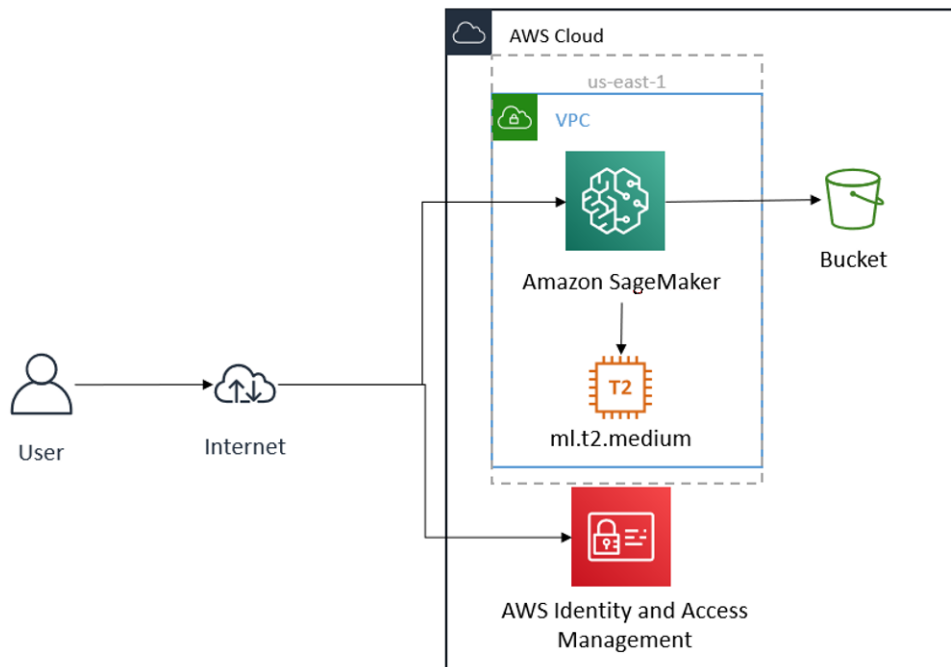


Figure 48 SageMaker service architecture

1 Sagemaker environment creation

1.1 Create a user with SageMaker access

First, we need to create a user that has access to use SageMaker service.

We move to Services -> IAM -> Users, and we press add user. It is needed to provide a name and press continue.

In access we press create a group, then we select all the administrator access. We also need to add all the SageMaker access.

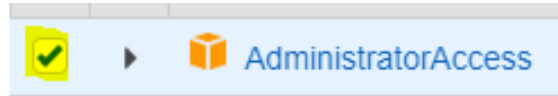


Figure 49 Administrator access selected

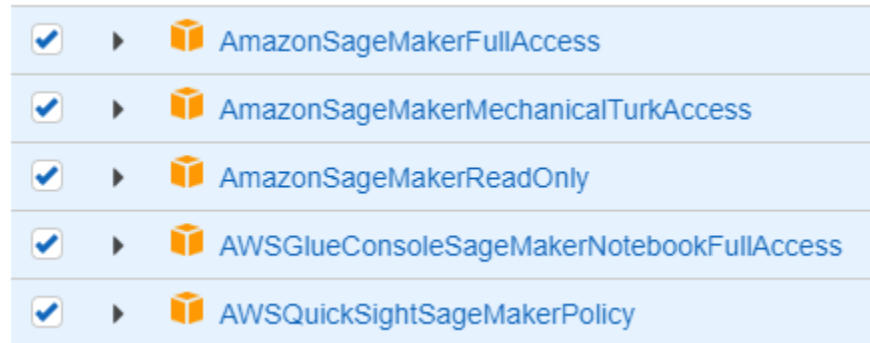


Figure 50 SageMaker access selected

Then, we need to press create user. Do not forget to save the user credentials.



Figure 51 Credential of user with SageMaker enabled

Sing in with the credentials of the created user.

1.2 Create a SageMaker Notebook

We need to select now Services -> SageMaker. At the left side panel, we will find the option Notebook instances. We need to press it and then we create one.

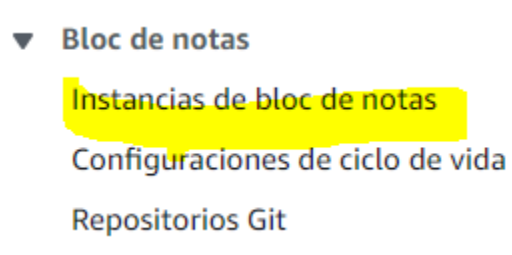


Figure 52 Notebook instances option

Then we provide a name to the instance and select an instance type.

Figure 53 Basic configuration for SageMaker

It is a good practice to pass the dataset through S3. We can give the instance access to S3 by creating a role.

Figure 54 Instance role creation

And we select S3.

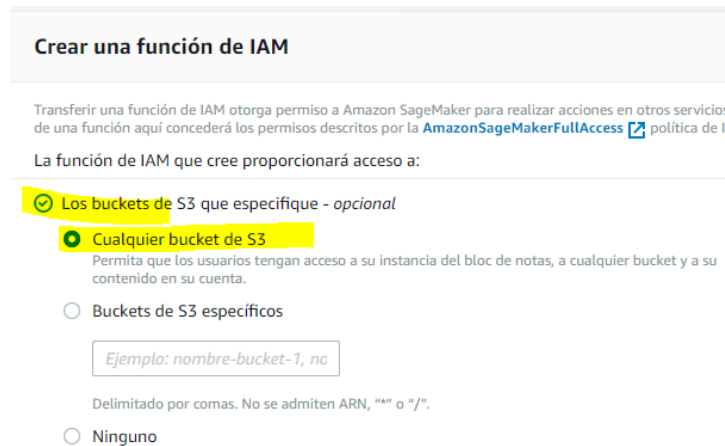


Figure 55 S3 access enabled for the instance

Finally, we just need to press create to finish the creation of the instance.

Once the instance is “InService” state we can press Open Jupyter and select the backend that we want to use.

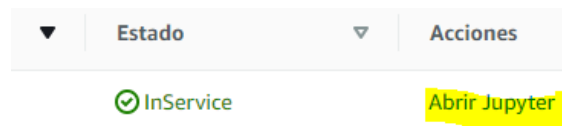


Figure 56 Open jupyter option

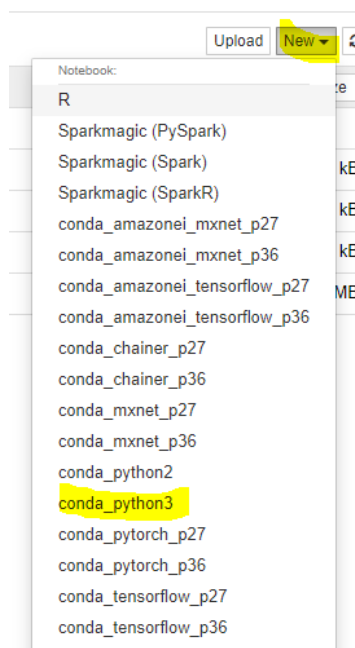


Figure 57 Backend selection

APPENDIX B. PC Tensorflow GPU backend configuration

This process is only possible if you have a computer with a NVIDIA board that supports CUDA libraries.

1 Anaconda, Visual Studio and CUDA installation

First it is needed to install VS 2017. Download from <https://visualstudio.microsoft.com/es/vs/older-downloads/> and install it.

Then it is required to download CUDA toolkit. Open the page <https://developer.nvidia.com/cuda-toolkit-archive> and select CUDA toolkit 10.1.

CUDA Toolkit 10.1 update2 (Aug 2019), [Versioned Online Documentation](#)
CUDA Toolkit 10.1 update1 (May 2019), [Versioned Online Documentation](#)
CUDA Toolkit 10.1 (Feb 2019), [Online Documentation](#)

Figure 58 CUDA toolkit 10.1

Then select the platform and version.

Select Target Platform

Click on the green buttons that describe your target platform. Only supported platforms will be shown.

Operating System

Windows Linux Mac OSX

Architecture

x86_64

Version

10 8.1 7 Server 2019 Server 2016 Server 2012 R2

Figure 59 Platform and version for CUDA toolkit

Download the installer and follow the steps to install CUDA Toolkit.

Now we need to add the CNN libraries to the CUDA toolkit. Open the page <https://developer.nvidia.com/cudnn> and select Download cuDNN.

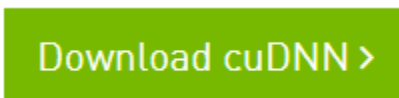


Figure 60 Download cuDNN selection

Download cuDNN 7.6.X and the one compatible with toolkit 10.1 that is the one that we have already installed.

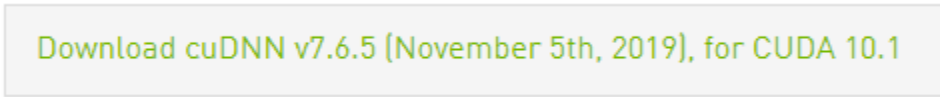


Figure 61 cuDNN v7.6 for CUDA toolkit 10.1

The download zip will contain three folders bin, include and lib. We need to add the content of these folders into our CUDA toolkit installation.

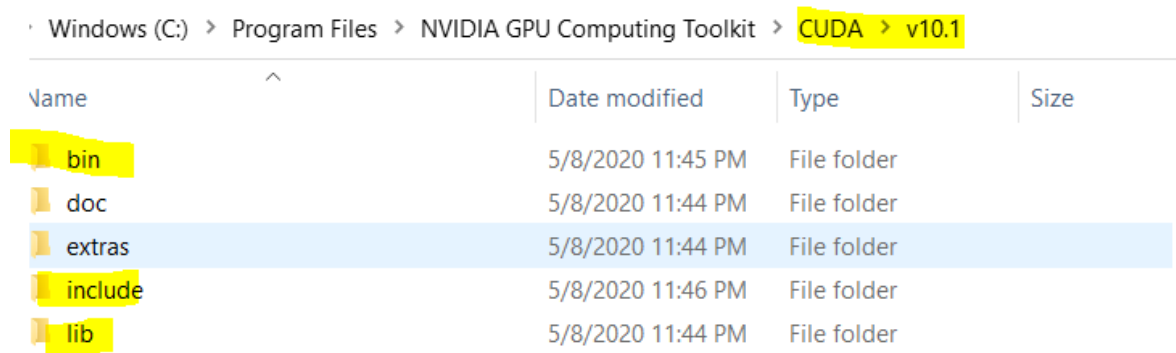


Figure 62 cuDNN added to toolkit

Finally, we need to add two environment variables to path. We need to add the location of bin folder and libnvp folder.

Move to System and properties in your computer -> Environment Variables, Select Path. Then press add and add the folders mentioned before.

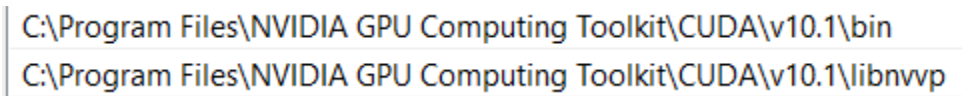
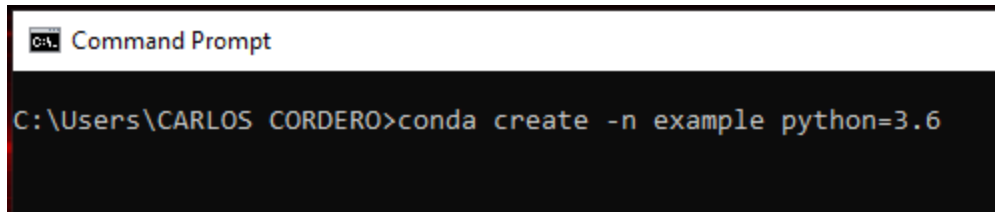


Figure 63 CUDA folders added to path environment variable

To install Anaconda refer to this page <https://www.anaconda.com/products/individual> and press Download and install. Do not forget to enable the option to add anaconda to path when installing.

2 Virtual environment creation with Anaconda

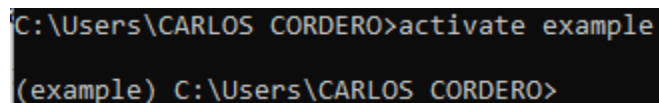
Open Command Prompt and write `conda create -n <Name of the environment> python=<python version>`. Tensorflow is not supported on all python versions we suggest using 3.6 because we are going to use TensorFlow 2.1.



```
C:\Users\CARLOS CORDERO>conda create -n example python=3.6
```

Figure 64 Virtual environment with name example and python 3.6 created

Now if we want to enter to such virtual environment, we just need to type `activate <Name of the environment>`

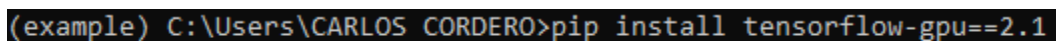


```
C:\Users\CARLOS CORDERO>activate example  
(example) C:\Users\CARLOS CORDERO>
```

Figure 65 Opening virtual environment

3 Tensorflow GPU/keras backend configuration

For this step we are going to use the PIP installer. Then after opening the virtual environment type, `pip install tensorflow-gpu==2.1`.

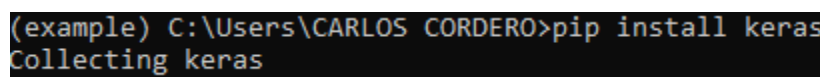


```
(example) C:\Users\CARLOS CORDERO>pip install tensorflow-gpu==2.1
```

Figure 66 Tensorflow installation

Avoid installing classic tensorflow otherwise Keras will not be well stacked.

Finally, we need to install Keras. We are going to use pip again to install Keras. Type again, `pip install keras`.



```
(example) C:\Users\CARLOS CORDERO>pip install keras  
Collecting keras
```

Figure 67 Keras installation

The complete backend that is used for this research can be found in <https://github.com/CARLOSALBERTOCORDERO/ToGDocumentation/blob/master/requirements.txt> and can be installed just typing: `pip install -r requirements.txt`.

```
(example) C:\Users\CARLOS CORDERO>pip install -r requirements.txt
```

Figure 68 Install libraries based on requirements

Take in consideration that this environment will weight near to 2GB.