Master Thesis

MASTER IN INDUSTRIAL ENGINEERING

CORRUPTION IDENTIFICATION AND CLASIFICATION IN STANDARD PLANE PRENATAL ULTRASOUNDS

REPORT

Author:	Lluís Cierco Corominas
Director:	Raúl Benítez Iglésias
Course:	July 2020

Escola Tècnica Superior
d'Enginyeria Industrial de Barcelona

ETSEIB

Abstract

Artificial Intelligence (AI) has proven in the recent years to be a powerful tool to automate simple and complex tasks. One of the main fields where it has been applied is medical science where especially computer vision algorithms are gaining popularity.

This project aims to be useful both for medical staff and future projects on the field, easing the process of data cleaning. Nowadays medical personnel have to invest time discarding corrupted images to clean the data from the ultrasounds. It is also an impediment for other machine learning projects as corrupted images can negatively affect the results.

Our goal is to develop a Deep Learning (DL) algorithm not only able to identify the corrupted images but also to classify them by type of distortion. With this we will be able to clean the dataset and bring an opportunity to future projects to apply artifact cleaning.

The proposed strategy has three main parts. First, we have designed a code to create semi random artifacts on correct images to generate a dataset big enough for our experiments.

Secondly, we have studied the behavior under corrupted images of a previous deep learning model, designed by Vicent at [6]. With this we determined the boundary between corruption and correct images.

Finally, we successfully created and compared the result of two models based on a Convolutional Neural Networks (CNNs) to identify and classify the corrupted images.

Summary

ETSEIB

1. Glossary

Neuron	Node that acts as computational unit. It is a function with an input and an output. Usually refereed as part of a larger network.
Neural Network	Structure of related neurons with an input and an output.
Layer (context: Neural Networks)	Group of neurons that are related by common inputs and outputs. It can be understood as a neurons part of a phase of the neural network.
Artifact	Feature appearing in an image not present in the original object.
Distortion	A modification or degradation something that makes it appear unnatural.
Corrupted image	An image with a distortion or an artifact that makes it defer from the original object.
Degree or Intensity of Distortion	Level of degradation
Structural Similarity Index (SSIM)	Comparison between structural information carried by two images. [1, -1] If equal, the result is 1.
Noise	Type of image degradation by the appearance of random intensity pixels
Blur	Degradation of the image edges, creating diffuse shapes
Scale up	Zoom, or crop like effect. When the image is not correctly centered and the fetus appears partially.

2. Preface

2.1. Project Origins

This project is result of a fruitful collaboration between Doctor Joan Sabrià Bach from the obstretrics department of the hospital Sant Joan de Déu and the Lassie Lab from Univeristat Politècnica de Catalunya.

After the previous successful projects of Safae [14] and Vicent [6] on classification of fetal plane ultrasounds, it become clear a necessity for an automation process to discard corrupted images form data bases.

From here, and thanks the images provided by Doctor Joan Sabrià, we decided to create a Deep Learning algorithm to identify and classify corrupted images from fetal ultrasounds.

3. Introduction

Artificial Intelligence (AI) has come a long way to arrive at where we are today. Thanks to the increase in computer power and research on the field AI is revolutionizing the way we work.

Medicine has always been a focal point of development for new technologies and AI is not an exception. From machine learning models to deep learning, AI is already been implemented in several applications for medicine.

One of the most promising AI branches is visual recognition, with the arise of Convolutional Neural Networks (CNNs) image classification algorithms have become a reality. It has already proved its power in multiple applications like cancer detection [18] or ultrasound recognition.

On this project we will contribute on the efforts done to implement Artificial Intelligence on medicine, and more specifically on fetal ultrasounds.

3.1. Project Objective

The aim of the project is to identify and classify distorted images of three standard fetal biometry ultrasounds: BPD (Biparietal diameter), CRL (Crown-rump length) and NT (Nuchal translucency).

The classification process has to be able to differentiate between correct images and three different distortions:

- Original: correct unmodified ultrasound images, without difference by type.
- Scale up: the fetus is not correctly centered and appears zoomed in relation to its shape.
- Blur: the outlines are diffused on different degrees obscuring details.
- Noise: the arbitrary alteration of brightness or color in an image by irregular pixels, misrepresenting the luminance and tonality.

Comparison of different degree of distortion in ultrasound images:

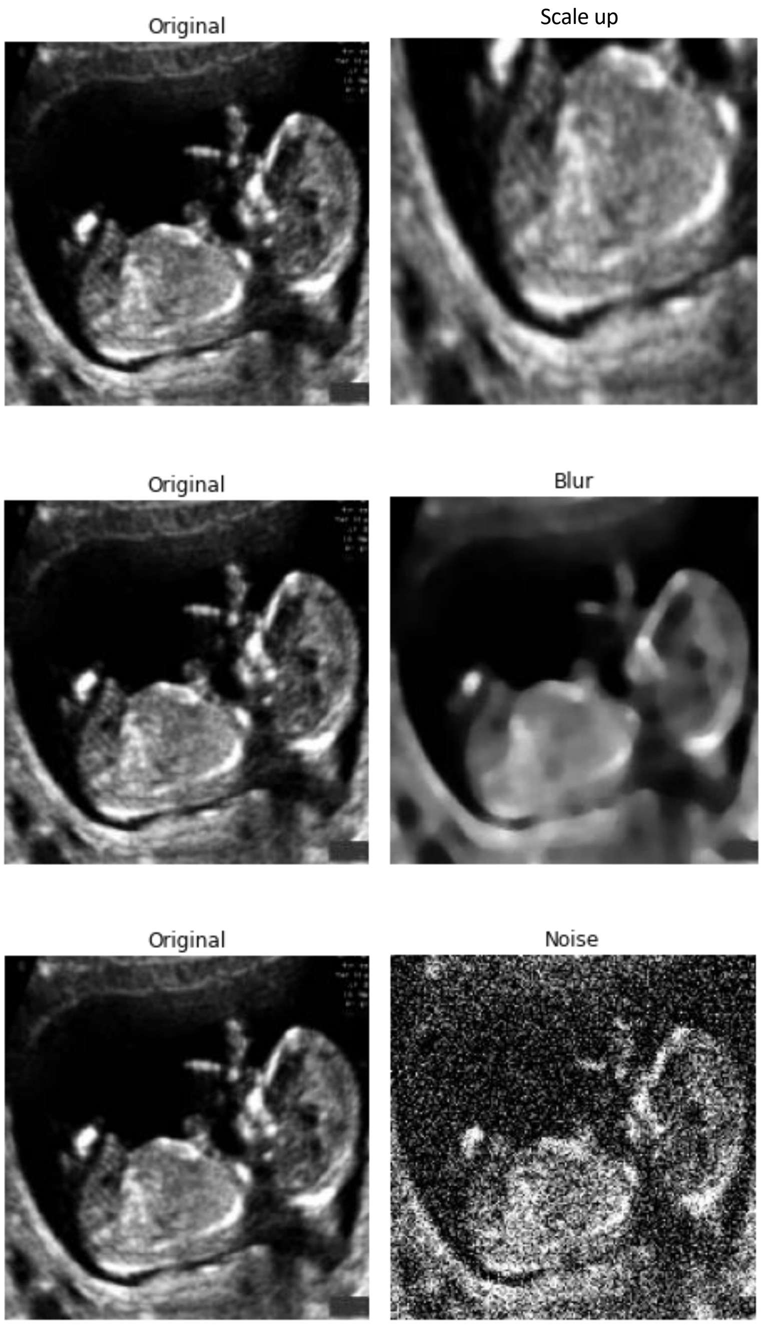


Figure 1. Comparison of the three corruptions and the correct image. (Source: own)

3.2. Project Pipeline

The project is structured in four major steps all of equal importance and specific sequential order:

1. **Define Project Objectives:** Set a goal to be able to define the success of the whole

project. As seen in the previous section the goal is to classify distorted ultrasound images.

2. Obtain Raw Dataset: Collect a set of the desired ultrasound images big enough for the project objectives. Most of the data has been provided by Doctor Joan Sabrià.

3. Inclusion of distortion and artifacts: we affect the images by introducing distortions and artifacts inspired in those observed in actual ultrasound images from the third trimester. Typically affected by different levels of blur, noise or scale like effect. We produce three copies for each image, simulating a distortion. As these images are artificial it is important to define the correct degree distortion applied.

4. Study of the impact of distortions in deep learning classification algorithms: We start by the deep learning classification model previously created by Vicent [6] to recognize ultrasound planed. In this section we study how the performance of the classifier is affected by different levels and types of distortions. The goal is to identify what should be considered a distortion and, as so, the current model cannot correctly classify.

5. Model Selection and creation: investigate the different machine learning models, specifically the ones using Convolutional Neural Networks (or CNN). In this project the library explored has been "Keras", comparing the models VGG16 and Xception pretrained on ImageNet. We can differentiate two major layers: the CNN as feature extraction and a dense connected layer as classifier.

6. Results: once the model is trained the performance is tested to check for the threshold values of identifiable distortion. Similarly, to the previous exploration the goal is to verify what can be identified as distortion and, as so, the model can correctly classify. To confirm the accuracy several visualization tools have been used to determine what the model is "looking at".

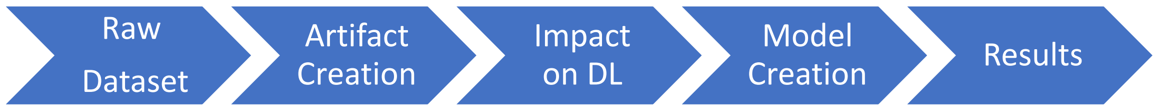


Figure 2. Project pipeline structure. (Source: own)

3.3. Programming Framework

The language used for the whole project is Python, a high-level and open programming language created by Guido van Rossum in 1991. It is, currently, the most popular tool for machine learning as stated in GitHub at [4]. There are several libraries created by the community oriented to machine learning, the following are used in this project:

- NumPy: multidimensional data operations and scientific computing.

- **SciPy:** algebraic functions, it uses NumPy to create more complex functions.

- Scikit-Image (skimage): image processing package

- Scikit-learn (Sklearn): machine learning algorithms and functions

- Keras / TensorFlow: library for developing and evaluating deep learning models. It also provides several popular pretrained models.

- **Matplotlib:** comprehensive library to create and edit visualizations.

The environment used is Google Colaboratory (Colab), a free cloud service based on Jupyter Notebooks for machine learning. It provides a fully configured linux virtual machine for machine learning projects with free CPU, GPU or TPU use for 12h a day. As part of google it has a direct connection to google drive storage. To do this we simply have to mount drive into Colab:

```
from google.colab import drive
drive.mount('/content/drive')
```

This code enables an identity check after which we can navigate through our google drive folders and load or save content.

4. Dataset

The ultrasound images used in the project are divided in two major categories: uncorrupted original images and artificial distorted images with three different kinds of alterations (scale up, noise or blur).

4.1. Raw Dataset

All the original images come from the previous project of Vicent [6]. Most of them were provided by the collaboration between Doctor Joan Sabrià Bach from the obstetrics department of the hospital Sant Joan de Déu and the Lassie Lab from the Univeristat Politècnica de Catalunya.

There is a total of 542 images classified by type (BPD, CRL or NT) and already clean of annotations. All images have been normalized to the size 337x227 and are colored (while the ultrasound is grayscale the untreated images have color tags).

The dataset was given in three different folders each with one class of ultrasound. All the images were also labeled under the class name and a sequence number as identifier but without any relevance. The set was merged in a single folder to ease the data loading process. These images are the only external input to the model, a part from the data imported from python libraries.

To load the dataset into the notebook, first we specify the folder path and list all the files (images). We import the package *"os"* to use the command functions of the operating system.

```
#----------------------PREPROCESS AND DATA CREATION IMATGES---------
#Get the the Image folders
import os

directori1='/content/drive/My Drive/TFM Lluis Cierco - DeepLearning
/Agrupades'
y1=os.listdir(directori1)
```

4.2. Generation of image artifacts and corruption

As said, we will need a dataset of not only correct ultrasound images but also corrupted ones. For this we could have manually selected and labeled a dataset of naturally distorted images. However, this would have been and expensive process and without control on the corruption itself. A better solution and the one we use in this project is to generate all the distortions by code.

To do this, we have to assume that all the images from the original dataset are correct and, as so, none present any kind of corruption. Because of this, we will often label the correct images as original.

The process to create the distorted (also refereed as modified) images consist of making three copies of all the original images and apply a different effect to each of them. With this we will triplicate the dataset and have 542 images of every class.

As result we have four classified datasets each with one class of image that we want to identify (correct, scale up, noise and blur).

It is important to keep in mind that the dataset has to be class balanced. This means that for optimal results we must have an equal number of images for each class. Luckily, as we are generating three of the four classes, we can easily maintain the balance. It is easy to see that the number of original images is the bottleneck in the process. Whichever is the size of our original dataset after the data generation process, we will have four times that size.

CLASS	**IMAGES**
ORIGINAL	542
SCALED UP	542
BLUR	542
NOISE	542
TOTAL	**2.168**

Table 1. Number of images by class and total number after the corruption generation process. (Source: own)

We have divided the corruption generation in multiple steps. Below we explain in depth the whole process:

First, we import several libraries to load and transform the images. All these packages are already preinstalled in Colab so they are ready to use.

```
#----------------------IMPORT----------------------
import skimage as sk
import skimage.io
import numpy as np
import cv2

from skimage.morphology import disk
from skimage.filters import median
from skimage.util import random_noise
import random
```

Now we have to prepare some variables to use in the transformation process. We use two groups of lists, one for the original (correct) images and one for the modified (transformed) images. Both of them have one list to store the images and another one to store the class of each image.

We also define the size of the images that we will load into the model by the width and height. In this case we decided to use 244x244 instead of the original shape to normalize all images after the transformation process, this will be further explained below.

```
#VARIABLES
#Create empty lists to fill with loaded and modified images
images_orig=[]
y_orig=[]

images_mod=[]
y_mod=[]

width=224 #Original shape 227
height=224 #Original shape 337
```

Now we start by looping through all the files in the folder previously specified. Every image is loaded in grayscale on the variable “im_orig” and resized to the defined shape.

```
for image in y1:

#Load image in grayscale and resize it to a normalised square shape
  im_orig=sk.io.imread(os.path.join(directori1,image),as_gray=True)
  im_orig=cv2.resize(im_orig, (width,height))
```

From this point we start the transformation process.

4.2.1. Scale up

First, we copy the original image and perform a crop. The size of the crop is defined by the four sides of the image with a random range that can accumulate up to 100 and at minimum is 75 per axis.

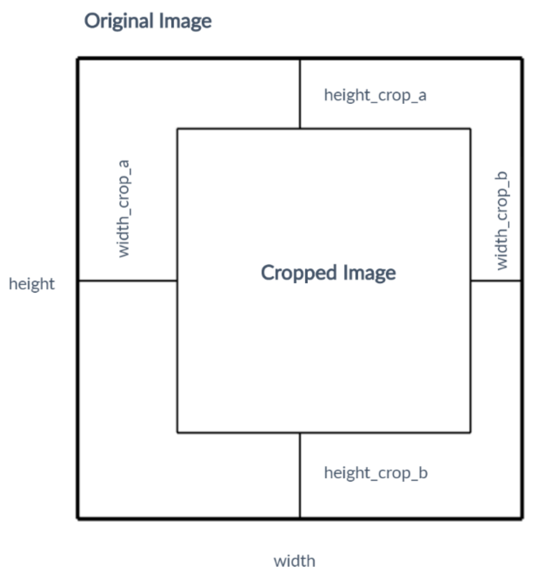


Figure 3. Representation of the cropping parameters applied to an image to produce the scale up effect. (Source: own)

After the crop the image size is drastically reduced, in the most extreme case (100 cropping per axis) the resulting shape is 124x124. Now we must normalize all image sizes, to do this we could scale up back to 224x224 but this would reduce the resolution of the scaled images. On the other hand, we could scale down all images to 114x114 but this would also produce higher differences in resolution. To keep it as neutral as possible we decided to scale all images to the average 174x174.

```
  # 1. Occlusion / Crop
  # Generate a random crop size for each side

  width_crop_a=random.randint(25,75)
  width_crop_b=random.randint(75,100)-width_crop_a
  height_crop_a=random.randint(25,75)
  height_crop_b=random.randint(75,100)-width_crop_a

  #Apply the crop
  im_crop=sk.util.crop(im_orig, ((width_crop_a,  width_crop_b), (he
ight_crop_a,height_crop_b)))
  #Resize to normalised shape
  im_orig=cv2.resize(im_orig, (width-50,height-50))
  im_crop=cv2.resize(im_crop, (width-50,height-50))
```

4.2.2. Blur

The second transformation is the blur, as said it consist in diffusing the image making the outlines harder to differentiate. The method used is the median filter, a non-linear digital filter that replaces each entry with the median of its neighbors.

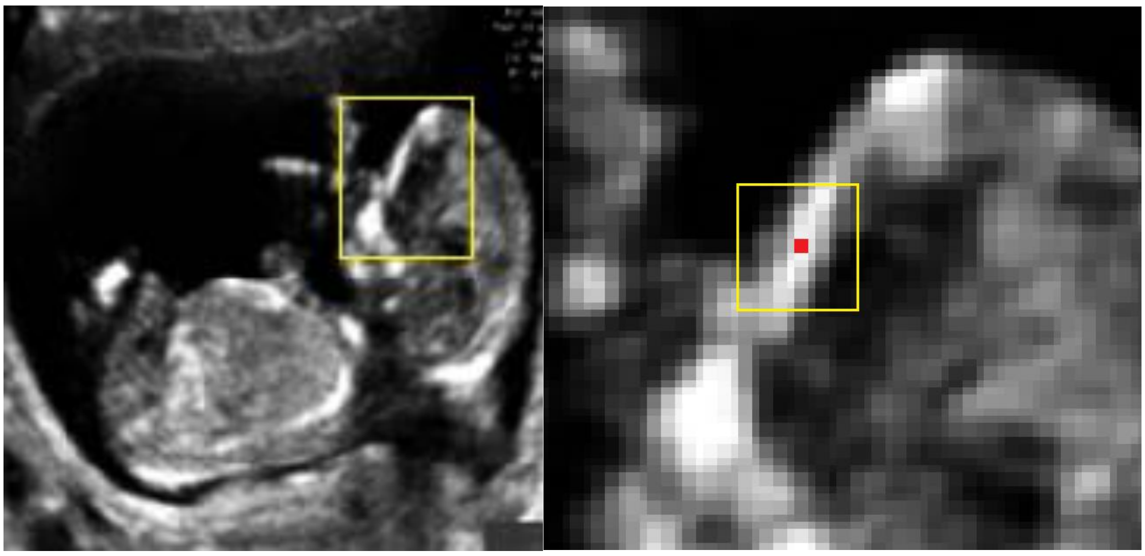


Figure 4. Example of the "window" used in median blur (left: original image, right: window example). The central point, in red, is calculated in relation all the neighbor pixels inside the yellow square. (Source: own)

The window is just a matrix where each position is the value of a pixel. As example we use a window of 3x3 and calculate its median (sort the values and take the middle one), this result is the new value of the central pixel. With this we replace each pixel in the image by the median of its neighbors. It is then clear that the highest impact will occur in the edges, where the pixel values have higher differences.

15	*132*	*195*
19	*120*	*201*
24	*145*	*192*

15	19	24	129	132	145	192	195	201

Figure 5. Numerical example of the median blur. (Top: Matrix, bottom: median selection). (Source: own)

The window of neighbors taken into account is defined, in this case, by a disk of random radius between 1 and 10. The disk nevertheless, is constant for a single image but can vary between images.

```
# 2. Blurr -- GAUSSIAN FILTER (MEDIAN)
# Create a random disk and appy the median

sel25 = disk(random.randint(1,10))
img_med25x25 = median(im_orig, sel25)
```

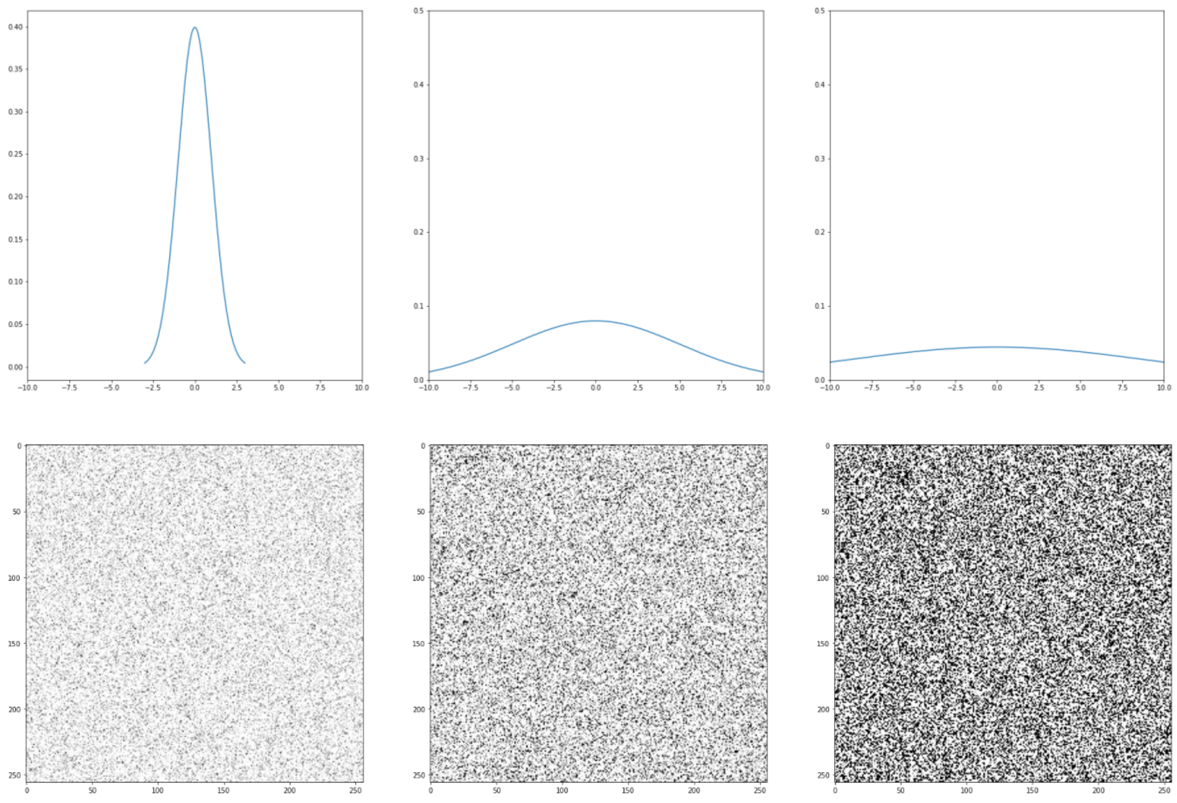
4.2.3. Noise

Lastly, to create the noise we add a pseudorandom number to some pixels in the image. This changes the brightness of each pixel by an intensity determined with a random distribution, in this case we use the gaussian distribution. This is defined by the mean, which represents the most probable value, we have set it to zero to maintain equal most of the pixels. To this, we add a variance that translates to how easy is to deviate from the mean. The higher the variance the more scattered the result.

```
# 3. Gaussian Noise
#Create random sigma and apply normal noise

sigma = random.randint(1,5)/10.0
noisy_image_normal = random_noise(im_orig, var=sigma**2)
```

The limit to this method comes when the image is saturated with noise, meaning that almost all pixels have been modified to some point. As the mean is set to zero this situation depends on the variance. As show in the Figure 6 a higher variance means a flattened distribution curve what makes all pixels affected.



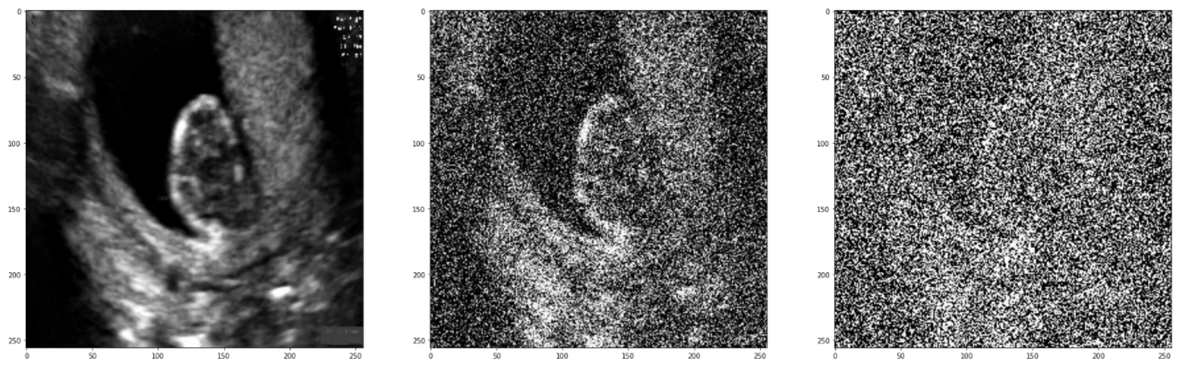


Figure 6. Three examples of gaussian distributions, and below the noise they create in a white image and the noise created in the ultrasound images (read in vertical). The variance increases from left to right as sigma=0.001, 0.1, 1.0 (variance=sigma^2). The first row of gaussian distributions shows in the y axis the probability to return the value in the x axis. (Source: own)

To conclude the process, we triplicate the array representing each image to create a colored (rgb) version. Images are represented by matrixes of numbers that set the brightness of each pixel. Color images are just an aggrupation of three matrixes: red, green and blue (rgb). As the color image that we create has the same value for each matrix the result does not modify the information that the image brings. This whole conversion is done because the CNN models that we will use require color images.

Then we store each image and its class into the lists that we created earlier, outside the loop. The lists are not directly related so it is important to keep the sequence in order as the items are related by the index. We can affirm that the "n" image of the list "images_orig" is the source image for the transformed "n, n+1, n+2" images of the list "images_mod". Equally we can classify the "n" item of the list "images_mod" by the "n" item of the list "y_mod".

```
  #CNN model needs color images
  im_orig=cv2.merge((im_orig,im_orig,im_orig))
  im_crop=cv2.merge((im_crop,im_crop,im_crop))
  img_med25x25=cv2.merge((img_med25x25,img_med25x25,img_med25x25))
  noisy_image_normal=cv2.merge((noisy_image_normal,noisy_image_norm
al,noisy_image_normal))

  #Store image + 3 distortions to the lists
  images_orig.append(im_orig)
  images_mod.append(im_crop)
  images_mod.append(img_med25x25)
  images_mod.append(noisy_image_normal)

  #Save the class of the iamges
  y_orig.append(0)
  y_mod.append(1) #Crop
  y_mod.append(2) #Blurr
  y_mod.append(3) #Noise
```

This lists store all the dataset we prepared, as we did not store the images in the memory everything is loaded in ram. While this is ram consuming it is much faster to operate because we do not need to store and load every single image we create.

5. Artificial Intelligence

Artificial intelligence (AI) is a branch of computer science with the objective to build smart machines able to perform some tasks "thinking" like humans. This field englobes multiple subfields but, in recent years machine learning has been specially rising in popularity.

Machine Learning is a branch of Artificial intelligence where the algorithms can learn from themselves without human intervention. Deep Learning is a subfield of Machine Learning based on the idea to mimic biological neuronal structures to create nonlinear algorithms.

In traditional machine learning algorithms are feed with features, important characteristics chosen and extracted by the programmer, from which the machine learns. Deep Learning uses instead an end to end learning, and it is able to extract the features by itself.

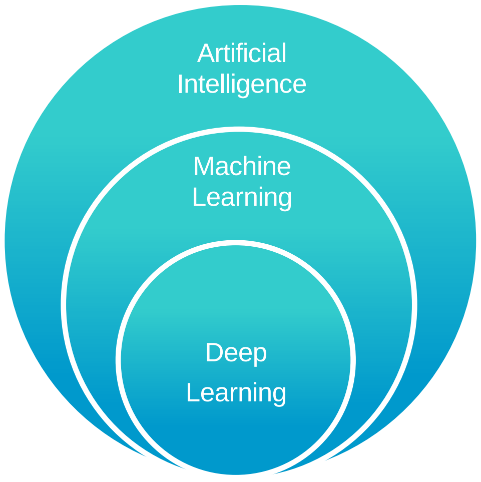


Figure 7. Artificial Intelligence englobes Machine Learning which englobes Deep Learning. (Source: Own)

5.1. History

The history of artificial intelligence began in antiquity with stories and myth of craftsman's that created artificial intelligent beings. It followed by philosophers reasoning about the structure of human reasoning and thinking. Its foundations started to take with mathematics and calculus but without the technology to make it possible AI was still far.

On 1940 some research was done and the first algorithms and intelligent machines where build. Later, on 1950 when Alan Turing set up the question "Can machines think?" [16]. In his article he created the "Turing Test" designed to identify whether a machine is intelligent by testing its ability to trick a judge into thinking it is a human. It was not until 1956 that all started to come together, a team of leading researchers met at the Dartmouth Conference where they discussed of machine intelligence and “created” the term artificial intelligence.

On 1959 Hubel and Wiesel studied the cortex of a cat and determined that some neurons activated to recognize simple structures as lines and other more complex combinations [7], this was the foundations of computer vision.

From the Dartmouth conference up to 1966 a lot of investment propelled ai research and major advancement where made however, the lack of computer power was a massive roof for its true capabilities and at one point most funding stopped. Thus, started the called AI Winter for roughly 30 years.

On this time AI kept developing by those who believed in it and started to generate more specialized branches. On 1997 Deep Blue was the first computer to beat a chess game champion. The recent advances in hardware explained by Moore’s Law, co-founder of Intel, he observed (predicted) that the quantity of micro transistors to fit in an integrated circuit doubles every year.

On 2012 the CNN AlexNet by Alex Krizhevsky and his team accomplishes a first position in the ImageNet classification challenge, thanks in part to the use of GPUs as computation power. From there major advancements have occurred and Artificial Intelligence is now commercially available on many sectors. The increase of both computer power and data have proven to propel artificial intelligence to where we are today.

5.2. Machine Learning

Machine Learning is based in the learning of the machines themself as opposite to the traditional programming where the rules are defined by the programmer. Thus, in machine learning the program is feed with data and it is able to automatically change its algorithms to achieve the result without the intervention of the programmer. This process is understood as the learning done by the machine.

Traditional machine learning needs the intervention of an engineer to extract and select the most appropriate features for the data. In computer vision, for example, we could use edge detection to extract features and then feed it to the algorithm. Then the machine would learn what is known as “bag of words”, the characteristic features that define each class. From here it can identify image of the same class by searching for this bag of words.

It is divided in three main fields:

- Supervised: the input data is correctly labeled and the algorithm learns the features that define each label. At the end, the machine is able to predict the label of an input data.

- Unsupervised: the input data is not labeled. The purpose of the algorithm is to find a pattern in the structure and define clusters (groups).
- Reinforcement: the input data is not labeled and the learning process is based in the experience. When the algorithm perform well it is reinforced with positive feedback while when it is incorrect it is penalized.

In our case we will be focusing in supervised learning as our data is labeled and we need the algorithm to be able to predict the label of new data. All this process is based in the idea of regression.

5.2.1. Regression

In statistics regression is an approach to define that a model that represents the relationship between one or more variable (dependents) and another one (explanatory). It is the algorithm created by the machine learning process.

There are several models but the simplest case is linear regression. In a two-dimensional situation we could have time (x) and the value of some stocks (y). Our intention is to know how much these stocks will be worth next year. As data we have several data points of the time and the value at that moment, obviously each one is more or less different. A linear regression will create a line to represent the relation between time and value and with it, we will be able to predict the value (y) at a given time (x). However, there is always some error between the prediction and the real data.

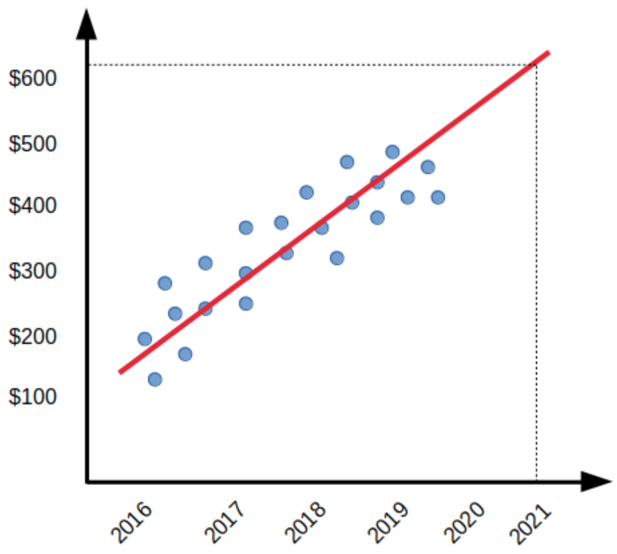


Figure 8. Simple example of linear regression. We have some datapoints and a linear regression that represents the relation between x and y. [13]

The linear model result of the regression is defined as:

$$h(x) = \theta_0 + \theta_1 \cdot x \qquad \text{(Eq.1)}$$

Where h(x) is the hypothesis function that fits the data, x is the feature and θ are the parameters or weights, these are the variables that the algorithm will adapt to fit the relation x/h(x).

To adapt the parameters to the data we use a loss or cost function. This function can be defined in several ways but what it does is measuring how off the current hypothesis (h(x)) is from the real data. From here the goal is to tune the weights (θ) to reduce the loss function making the model to better fit the data and as close as possible to the real data.

A basic example of cost function is the squared difference between the hypothesis result and the real data:

$$(h(x) - y_{real})^2 \qquad \text{(Eq.2)}$$

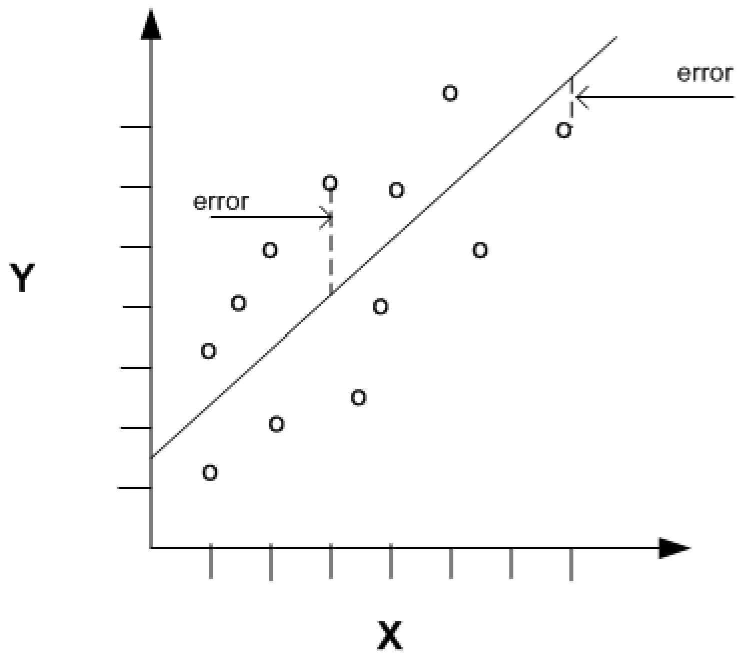


Figure 9. Visual example of linear regression and the error or loss, between the predicted result (line) and the real data (point). [13]

What we need is to minimize the cost for every data sample (pair of x/y) we have. Then, as x and y are defined all it remains is the parameters, which we will tune to reduce the cost.

There are again several methodologies to reduce the cost function, a general approach is to use gradient descent. This focuses on variating each parameter by a small value that accomplishes to reduce the cost function. It is usually compared to taking small steps in the steepest direction until we reach the bottom. It is important to keep in mind that when we arrive to the bottom we cannot assure if it is a global or local minimum of the function.

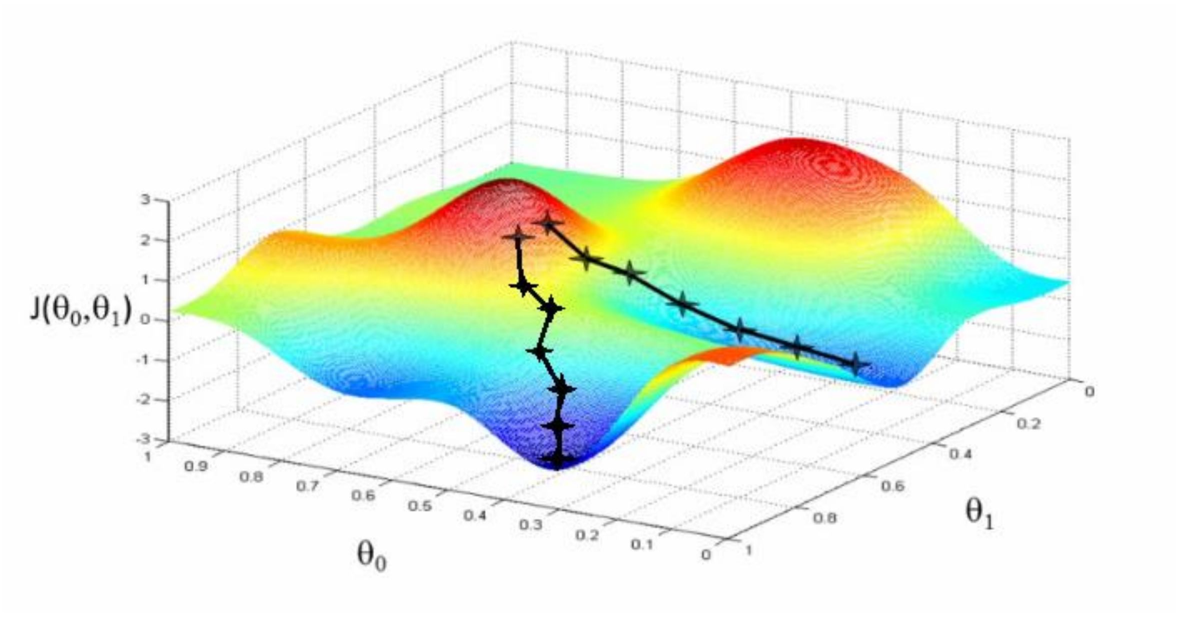


Figure 10. Cost function representation with two paths of gradient descent arriving at two different local minimums. [1]

To mathematically define this, we calculate the partial derivatives of the cost function for each parameter to find the “steepest” direction, the direction that will reduce more the cost. We repeat the process until converging into a local (or global) minimum.

The important concept here is generalization, we need the model to be able to predict the result of an input without having seen it before. If we train the model with insufficient data it will “memorize” the result of the training inputs but, as it has not learnt all possible variations it will give a wrong result on new inputs. As example we could take the data at Figure 9, if we would create a regression line from only two points it is very unlikely that the result would have been the same, instead the more points we take the more fitted is the line.

The selection of all these functions has a key role in the training and performance of all machine learning models. More advanced models have substituted the simpler linear regression as for example the sigmoid, tanh, ReLu (used in this project) or ReLu leaked to say some. These models perform better and are faster to train. However, the underling idea remains the same, optimizing the cost function to fit the data.

5.3. Artificial Neural Networks

Artificial neural networks (ANN) where first inspired by the brain, and they try to mimic the very same behavior of neurons, thus the name.

5.3.1. Deep Learning

In a traditional structure, the network is composed by multiple neurons usually grouped by layers. There are basically three types of layers: input, which is directly the input data. Second, hidden (or middle) layers and lastly the output layer. ANNs with many hidden layers are called Deep Learning.

Each neuron represents a computational unit, an operation similar to the regression defined by the activation function. The input of each neuron comes from all the outputs of the previous layer. The relation is modeled with weights (θ) which as before, the training process will adapt.

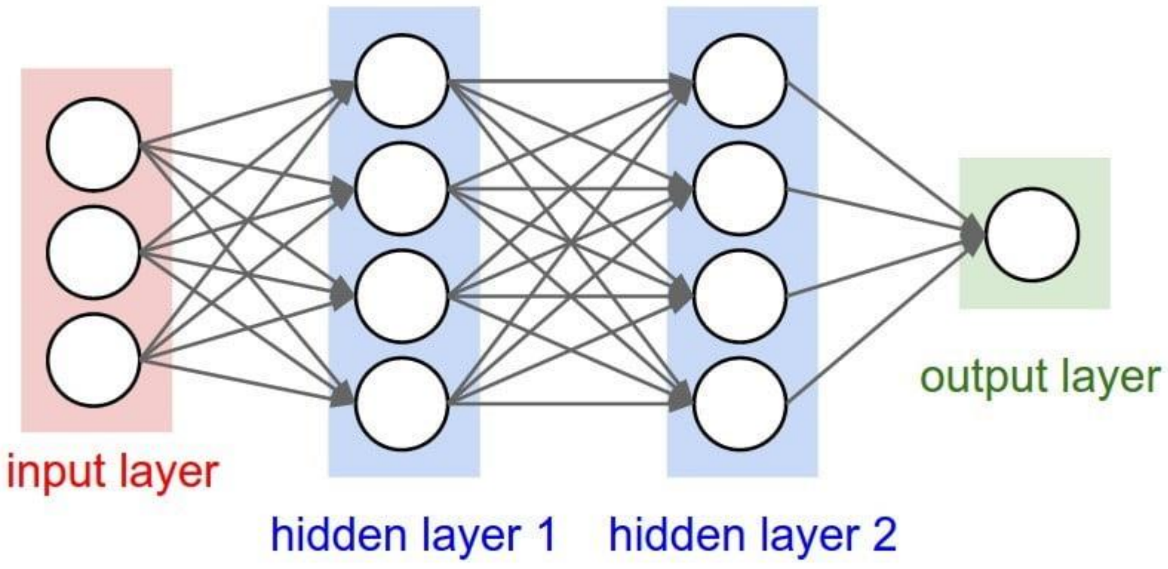


Figure 11. Artificial Neural Network Architecture. [5]

This combination of neurons permits automatic extraction of features and also generates a nonlinear algorithm, fitted for more complex problems.

To help understand how this works we have prepared a visual example with three neurons (one as bias, a constant value) and one output. With this structure we can easily create and AND logic function.

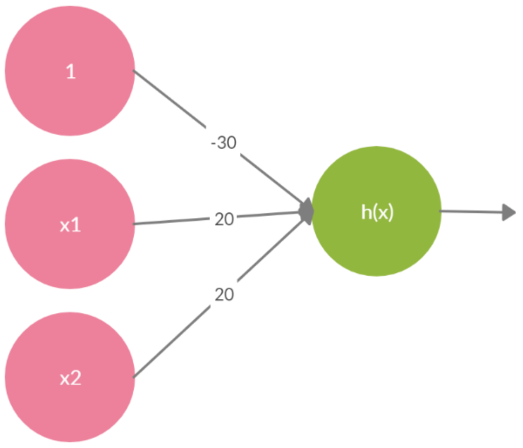


Figure 12. Example of neurons acting as AND logic function. (Source: own)

The output of the neurons is weighted and summed as input of the next neuron. The weights have been adjusted in the training and now, the output of the first layer (x1, x2) will define our output.

The activation function on the last neuron is the sigmoid function, shown below at Figure 13:

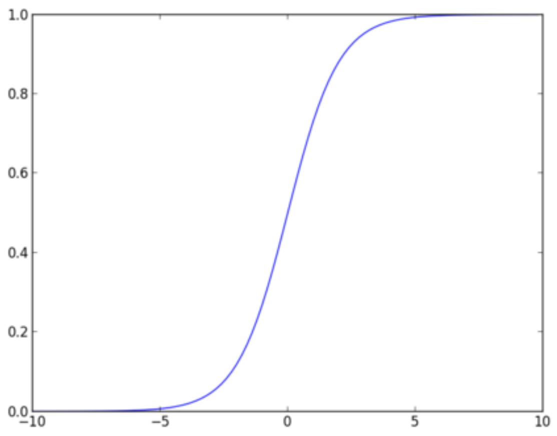


Figure 13. Sigmoid function plot, used as activation of the last layer neuron. (Source: own)

The sigmoid function returns an output limited between 0 and 1. Large x values will return 1 while large negative values will return 0.

With this we can now create a decision table of the possible solutions based in the input values:

x1	x2	h(x)
0	0	h(-30) → 0
0	1	h(-10) → 0
1	0	h(-10) → 0
1	1	h(10) → 1

Table 2. Logic table representing AND function created by the ANN example. (Source: own)

Combining multiple neurons and layers permits more advanced and complex algorithms.

A common approach in classification algorithms is to create the output layer with as many neurons as classes we have. Then, while activation functions like ReLu or ReLu leaked are usually better at hidden layers we set the output neurons with sigmoid functions. Now, each neuron represents a class and the output of the sigmoid function, from zero to one, is the predicted probability of the input data to belong to that class.

5.3.2. Back-propagation

To train a neural net we follow the same principles as before. Now we have an algorithm result of combining multiple activation functions by weights. To correct the output, we will again use the cost function and gradient descend. In this case however, we will use it from the last layer and use the partial derivatives to "propagate" the correction thought the previous layers and the network.

This process is highly resource consuming, and due this it is very important to choose the correct activation functions. The shape of the derivative of the cost function will determine the "velocity" of the training.

5.3.3. Convolutional Neural Networks

Convolutional neural networks or CNNs are a branch of artificial neural networks specifically designed for image inputs. This specialization allows to adapt certain features into the architecture that greatly enhance performance.

CNNs are much better than traditional ANNs at extracting features from images, as we will see, the use of convolutions takes into account spatially related pixels rather than trying to connect all pixels. This characteristic is what makes CNNs much better and faster.

The basic idea is the same as in ANNs where several neuronal layers are connected between each other. In the case of CNNs however, the layers are organized in 3 dimensional volumes: height, width and depth. This structure represents an image itself and are usually referred as feature maps.

Another important difference is that neurons are not fully connected to previous layers, instead each neuron is only connected to a small region of neurons.

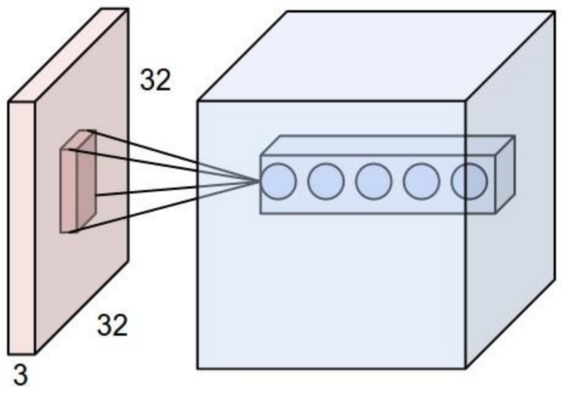


Figure 14. Convolutional neural network layer example. The red cube represents an input image of 32x32x3, the neurons from the next layer only look at a small region. [5]

All neurons from a certain depth of a layer share the same input size and parameters, while they are “looking” at different regions all regionals are of equal shape and the weights are equally distributed. This permits to drastically reduce the number of parameters as we have unified all the neurons per depth. These specifics characteristics are referred as filter or kernel.

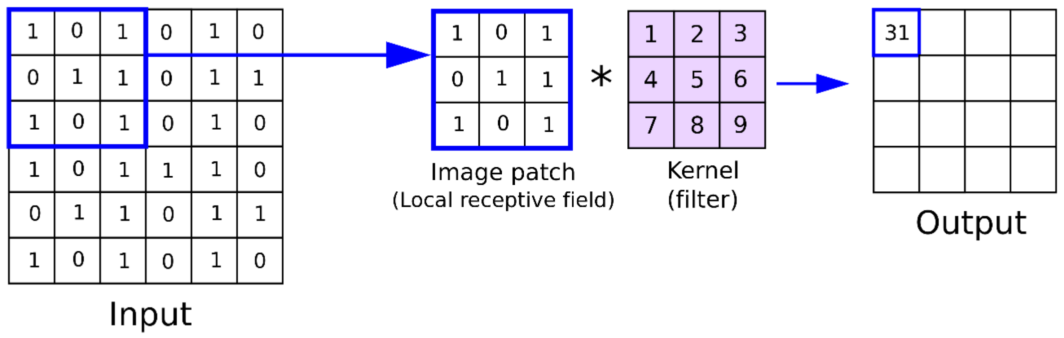


Figure 15. Visual representation of a convolution calculus. Notice that the kernel represents the weights of each connection of the input to the neuron in the output. [5]

Adjusting the parameters of each filter we can tune the filters to identify certain structures. The first layer would, for example, identify straight lines oriented in different directions. Then combining these results in the next layer might identify shapes as squares or connected lines at a certain degree. As the process dives deeper in the network the CNN is able to identify more complex shapes.

Depending on the volum of the layers it is interesting to reduce the shape of the feature map to control the ammount of parametrs and increase efficency (speed up the learning). To do it, architecures usually include polling layers to reduce the size of the feature maps. The pooling simply takes a region of the previous layer and merges it into one neuron using a max or avergage operation. On the opposite of convolutional layers, the depth of the feature map remains unchanged what means that the resulting shape is smaller than the input.

Combining convolutoion with pooling we can create a CNN and train it the same way than an ANN using back propagation. Nevertheless, neither convolutions nor pooling are fitted to classify the input images. Thus we have to add a traditional ANN on top (at the end) of the CNN. To do this we must flatten the last layer of the CNN so we can feed it to a fully connected network. This combination uses the feature extraction of convolutions and the classification of ANNs.

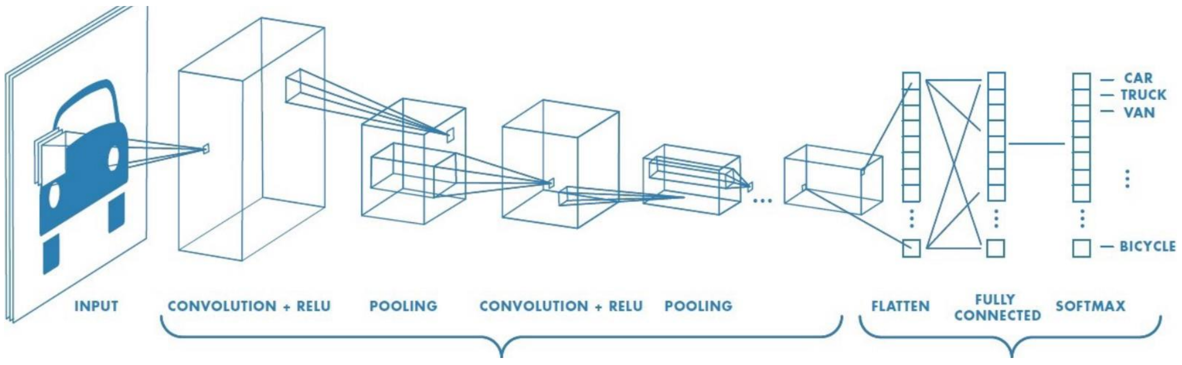


Figure 16. Convolutional Neural Network architecture example. [9]

5.3.4. Predefined and custom

There are an infinite number of ANN architectures to create, we can regulate layers, activation functions and many more parameters to define a network. The problem relies on selecting the right combination that performs best in our scenario.

Luckily, an architecture fitted for a specific case is usually also fitted for another case more or less similar. Thus, we can reuse predefined architectures that have already proved to be good. Because of this is much faster and straight forward to create models as we can simply import an already existing.

On the other hand, we can opt to create a model from scratch to specially fit our requirements. There are several libraries as Keras tensorflow, Caffe or Theano that provide both predefined models and layer structures. We can easily combine these structures to create our own model.

Another option and the one that we will be using in this project is to use a predefined CNN model and remove the top classification layer to create our own classification. If needed, we can also modify specific layers and parameters of the network.

5.3.5. Transfer learning with pretrained networks

When we have defined our model whether is predefined or custom, we have to train it to fit our data. As explained, this means using backpropagation to tune our model weights so it reduces the loss function. Similar to the architecture we have two options: train from scratch or use transfer learning with a pretrained model.

If we have a predefined model, we can opt for transfer learning which it relies on the concept of general-purpose models. In this we case, we will import not only the model architecture but also its parameters.

It is especially relevant in CNNs where the features to solve one problem are usually similar for other problems. If we remember how convolutions work, we can understand that the basic shapes that define one structure (lines, squares…) are the same for other structures.

Thus, we can save these feature extraction learning from one model and use it on another. By recycling the learning, our model will already be able to extract some features from the data, for example lines and curves from images.

A common approach and the one that we will use in this project, is to use a pretrained CNN and add a custom classifier. Then we feed our data to the whole model (CNN + classifier) and train it but, instead of starting to train the CNN from random weights we do it from the imported ones.

This method accelerates the learning process and allow us to train models with less data, as the model will "only" need to slightly adjust the weights instead of modify them completely. The biggest improvement in weights is done in the classification layer as it is usually utterly specific to each case.

If we train from scratch, we start by randomly initializing the weights of the model then, we feed our data and the learning process adjusts step by step the weights to the optimal values. For this, we need a relatively large amount of data to allow the learning process do enough modifications.

5.3.6. Data augmentation

A useful resource in machine learning is use functions to increase the dataset. A common approach is to perform slight modifications to the image dataset to create more images but equally valid. Some of the main modifications are:

- Flipping: mirror the image in the vertical or horizontal axis to create its opposite
- Rotation: rotate images between some certain degrees, usually a defined range.
- Brightness: modify the intensity of each pixel to change the brightness, usually a range.
- Zoom: zoom in a particular area of the image.

This methodology is based on the concept that the more situations we show our model, the better it will perform. Even if it is the same image changing some characteristics will improve the learning.

Nonetheless it is important to keep in mind that the root image is the same, thus it can produce the undesired effect of overfitting. As we feed our model many variations of few images the learning process memorizes these data rather than generalizing the solution.

5.3.7. Model validation

When the model has been defined and trained, we must do a validation test to ensure that the predictions are correct and generalized to the problem and not only fitted to the training data. Due to this, it is common to part the dataset between training and test data.

Once trained the test data will be feed into the model for first time to check if the predictions are correct. There are several metrics that we can use to do so:

- **Accuracy:** proportion of correct predictions in relation to the quantity of predictions done. It is the most straight forward way to evaluate a model.
- **Sensitivity or Recall:** it is the proportion of correct positive predictions in relation all real positive cases (positive refers to the input belonging to the class). Proportion of true positive.
- **Specificity:** as opposite to the sensitivity it refers to the proportion of negative predictions in relation all real negative cases, inputs that do not belong to a certain class. Proportion of true negative.
- **Precision:** proportion of true positive in relation all positive predictions.
- **F1 Score:** a unique metric to sum up precision and recall, it usually uses a harmonic average that shows if one of the metrics is especially low.

To present a more visual representation of the results we can use the confusion matrix: Table 3. We see on the diagonal (top left to bottom right) the correctly predicted inputs. The rest of the matrix shows the wrongly classified (False) proportions. The simplest version is a binary classification but the same principles apply to multiple class.

		ACTUAL		
		POSITIVE	NEGATIVE	**METRICS**
PREDICTIED	POSITIVE	TP	FP	Precision $\frac{TP}{TP+FP}$
	NEGATIVE	FN	TN	
	METRICS	Sensitivity $\frac{TP}{TP+FN}$	Specificity $\frac{TN}{TN+FP}$	Accuracy $\frac{TP+TN}{TP+FP+TN+FN}$

Table 3. Confusion Matrix and formulas for performance metrics. (Source: own)

6. Deep Learning with Distorted Images

The main goal of this project is to build a model that is able to detect and classify distorted ultrasounds from correct ones. For this we must first define what is considered a distortion and explore the limits between correct and corrupted images.

6.1. Methodology

As previously explained, one option would be to use a dataset with correct and distorted images labeled by an expert in the field. The model could then learn from these data without us having to define an explicit limit at which an image is no longer considered correct. Nevertheless, this process has some drawbacks, as the database is quite hard to obtain and we have no control over the corruptions.

Instead we opted to create artificial artifacts on correct images. This methodology is far more flexible than the previous as we can easily control the degree of corruption and the size of the dataset. As we will see this will allow us to study in detail each distortion.

Then, we used the model designed by Vicent at [6] to experiment its behavior classifying Fetal Biometric Planes under different degrees of image corruption.

For the experiment, we used the structure with the best results, a pretrained Xception model from Keras plus a densely connected neural network as classifier, and trained it with the data we had (keeping a 20% of the images as test). With this we can accquire a fairly high accuracy, over 90% under only correct images.

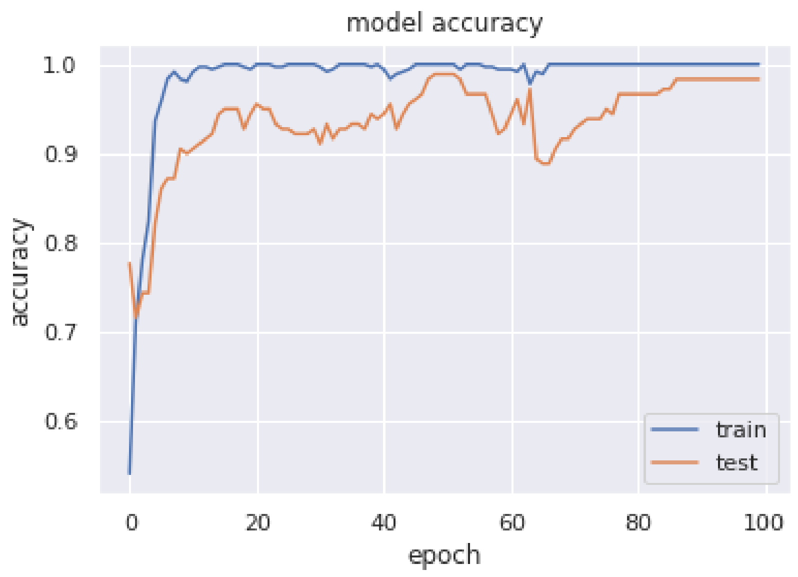
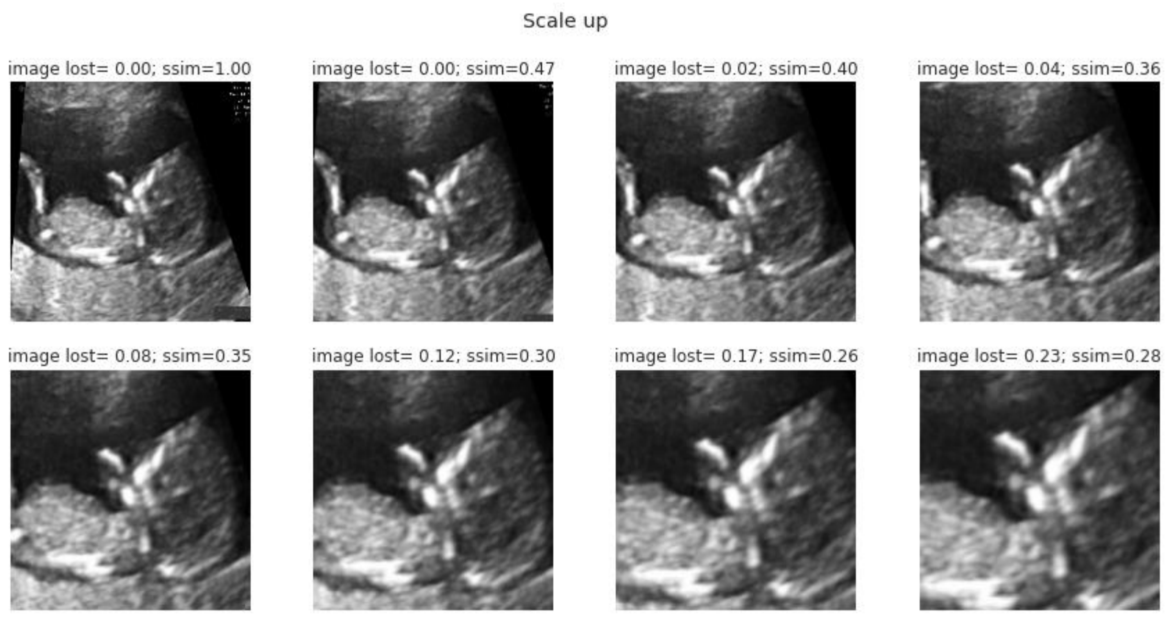


Figure 17. Model training and testing for a classifier (CNN + dense layers) of Medical Biometric Planes. The result is over 90% on accuracy. (Source: own)

From here we apply the transformations to create the distortions onto the test dataset. We apply the same intensity of the effect to all the test images and use a loop to successively scale up the intensity on each iteration.

This means that on a first iteration we will apply scale up to the full test dataset cutting out all the borders 4pixels. On a next iteration we increase the intensity to 8pixels. We keep this process until we have done 50 iterations of the same transformation and then repeat it onto the next distortion, starting again at the "lowest" intensity of the effect and scaling it up.


Scale up
image lost= 0.00; ssim=1.00
image lost= 0.00; ssim=0.47
image lost= 0.02; ssim=0.40
image lost= 0.04; ssim=0.36
image lost= 0.08; ssim=0.35
image lost= 0.12; ssim=0.30
image lost= 0.17; ssim=0.26
image lost= 0.23; ssim=0.28

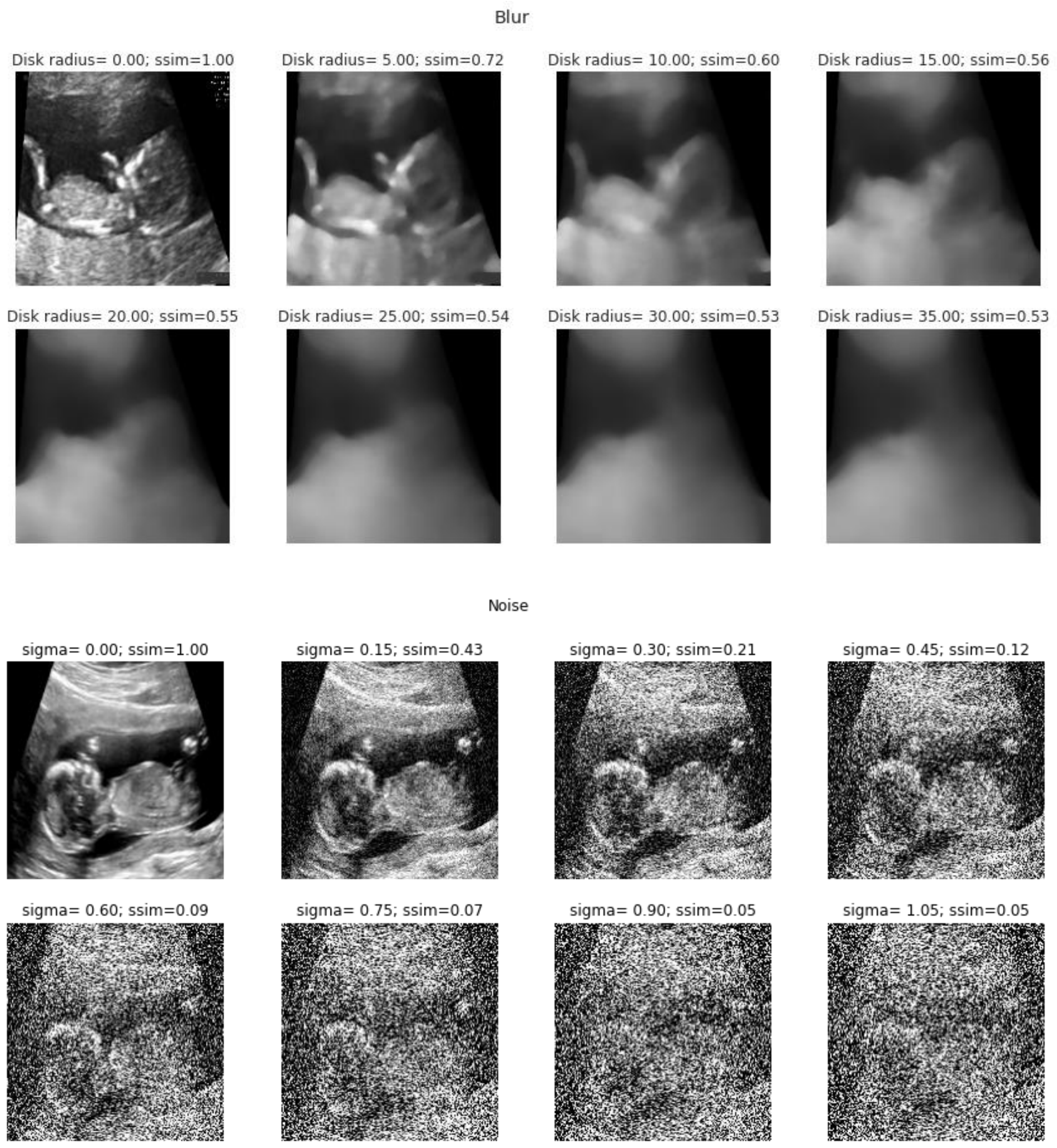

Blur
Disk radius= 0.00; ssim=1.00
Disk radius= 5.00; ssim=0.72
Disk radius= 10.00; ssim=0.60
Disk radius= 15.00; ssim=0.56
Disk radius= 20.00; ssim=0.55
Disk radius= 25.00; ssim=0.54
Disk radius= 30.00; ssim=0.53
Disk radius= 35.00; ssim=0.53
Noise
sigma= 0.00; ssim=1.00
sigma= 0.15; ssim=0.43
sigma= 0.30; ssim=0.21
sigma= 0.45; ssim=0.12
sigma= 0.60; ssim=0.09
sigma= 0.75; ssim=0.07
sigma= 0.90; ssim=0.05
sigma= 1.05; ssim=0.05

Figure 18. Different degrees of distortion for each corruption. The labels indicate the value of the variable used to create the effect and the structural similarity index (ssim). (Source: own)

With each iteration we create a new dataset of test images transformed a certain degree by a certain distortion. We then use this dataset to test the model and get its accuracy by distortion intensity.

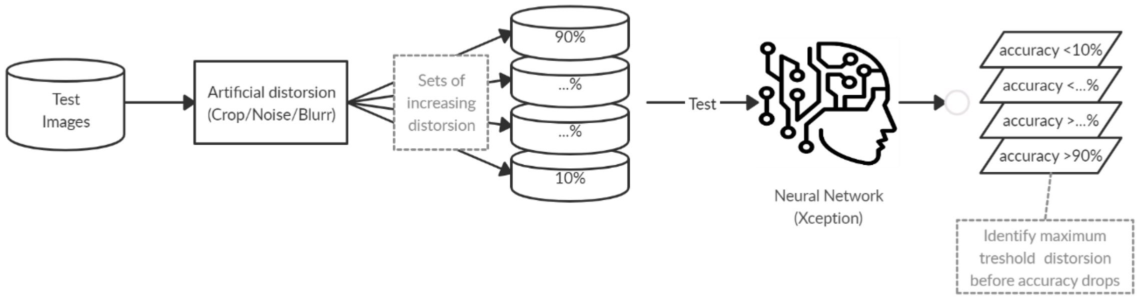

Test Images
Artificial distorsion (Crop/Noise/Blurr)
Sets of increasing distorsion
90%
...%
...%
10%
Test
Neural Network (Xception)
accuracy <10%
accuracy <...%
accuracy >...%
accuracy >90%
Identify maximum treshold distorsion before accuracy drops

Figure 19. Structure of the experiment to define the distortion limits. (Source: own)

6.2. Results

6.2.1. Accuracy vs intensity

To visualize the results, we have plotted three graphs (one for each distortion) that relate the accuracy of the model (y-axis) with the intensity of the corruption (x-axis). Each intensity represents the defining parameter of the corruption. The scale up is defined by the proportion of image "lost" in relation of the original image. The noise is defined by the value of sigma and the blur by the radius of the disk used.

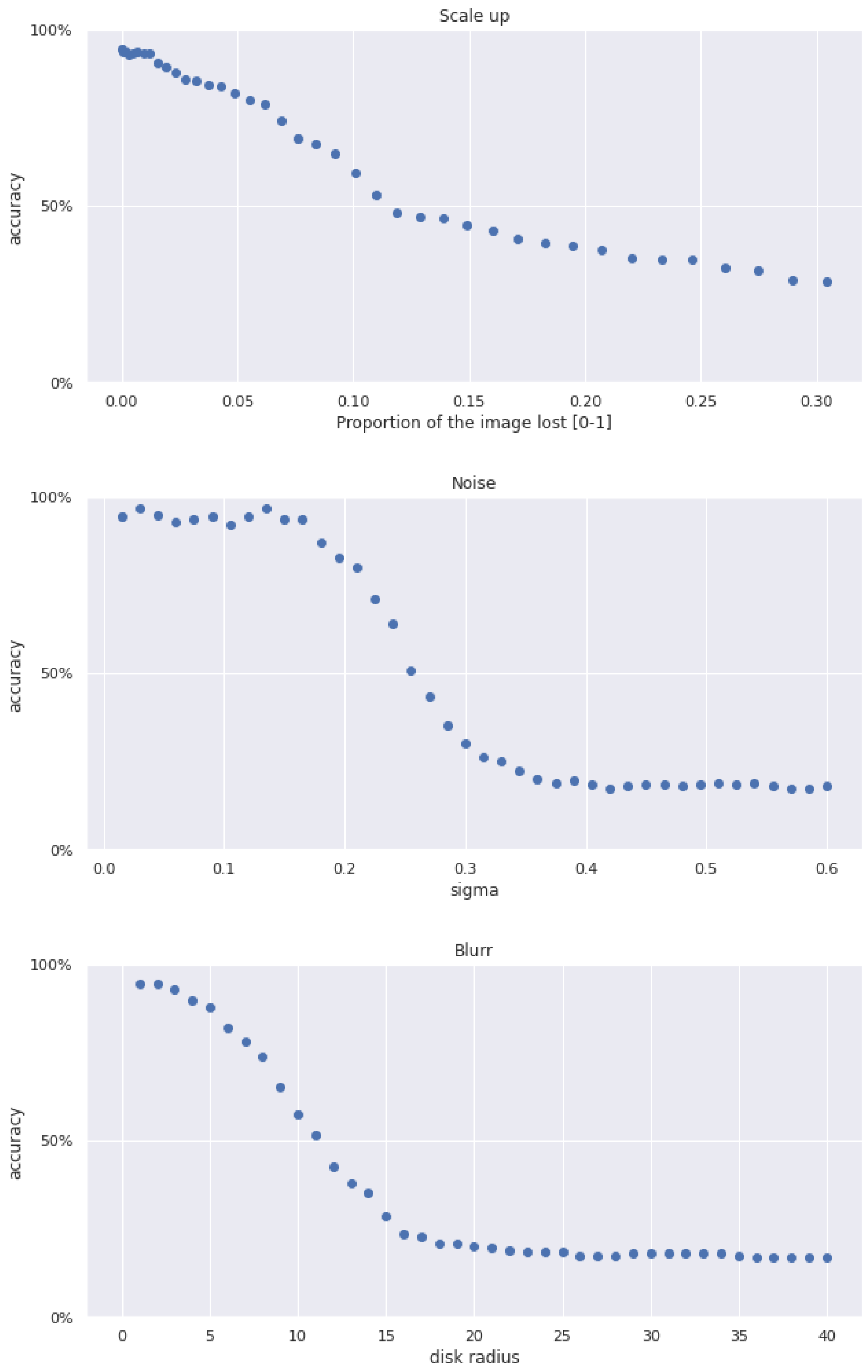


Figure 20. Relation of accuracy and intensity for each distortion. (Source: own)

As it is predictable the more the intensity of distortion the lower the accuracy of the model. We observe, nonetheless, an inflation points where more distortion does not translate into accuracy loss Figure 20. We clearly see that all graphics have a minimum accuracy level, this is because at some point even if the model is randomly classifying it will sometimes predict the correct class.

Combining all the results we can set a threshold value of what the model considers a distortion. We set a limit accuracy of 90% or an equivalent accuracy loss of around 5%. When the intensity of the corruption makes the accuracy drop under this value, we will consider the effect a distortion. The results of this experiments are:

- Scale up: proportion of the image lost 0.015 (1,5%)

- Noise: the sigma that defines the variance is sigma = 0.18

- Blur: the pixel size of the disk radius is r = 4 pixels

6.2.2. Accuracy vs structural similarity (ssim)

To further understand the results, we decided to measure the distortion by a standardized method. On the one hand, this will let us have an idea of how much the image is distorted in relation to the original. And on the other hand, it will permit a comparison between effects.

The method used is the structural similarity index [20]. It defines a value of similarity between a reference image and a distorted one, comparing both images and the structure of the information they carry. This approach is inspired on how humans see, as it does not compare pixel by pixel but instead it focuses in structures.

The key relies in the idea that pixels have strong interrelations specially between neighbors. To measure this, the algorithm compares sets of spatially close pixels (windows) of both images in multiple ways, to finally determine an index between 1 and -1 where 1 means identical images, 0 no similarity at all and -1 opposite structure. For our problem we only have to take into account 1 or 0.

Now we can represent the effect of the distortion based on the real impact on the image. Because the structural similarity index is based in how humans see it will be easier to understand why the model fails to correctly classify.

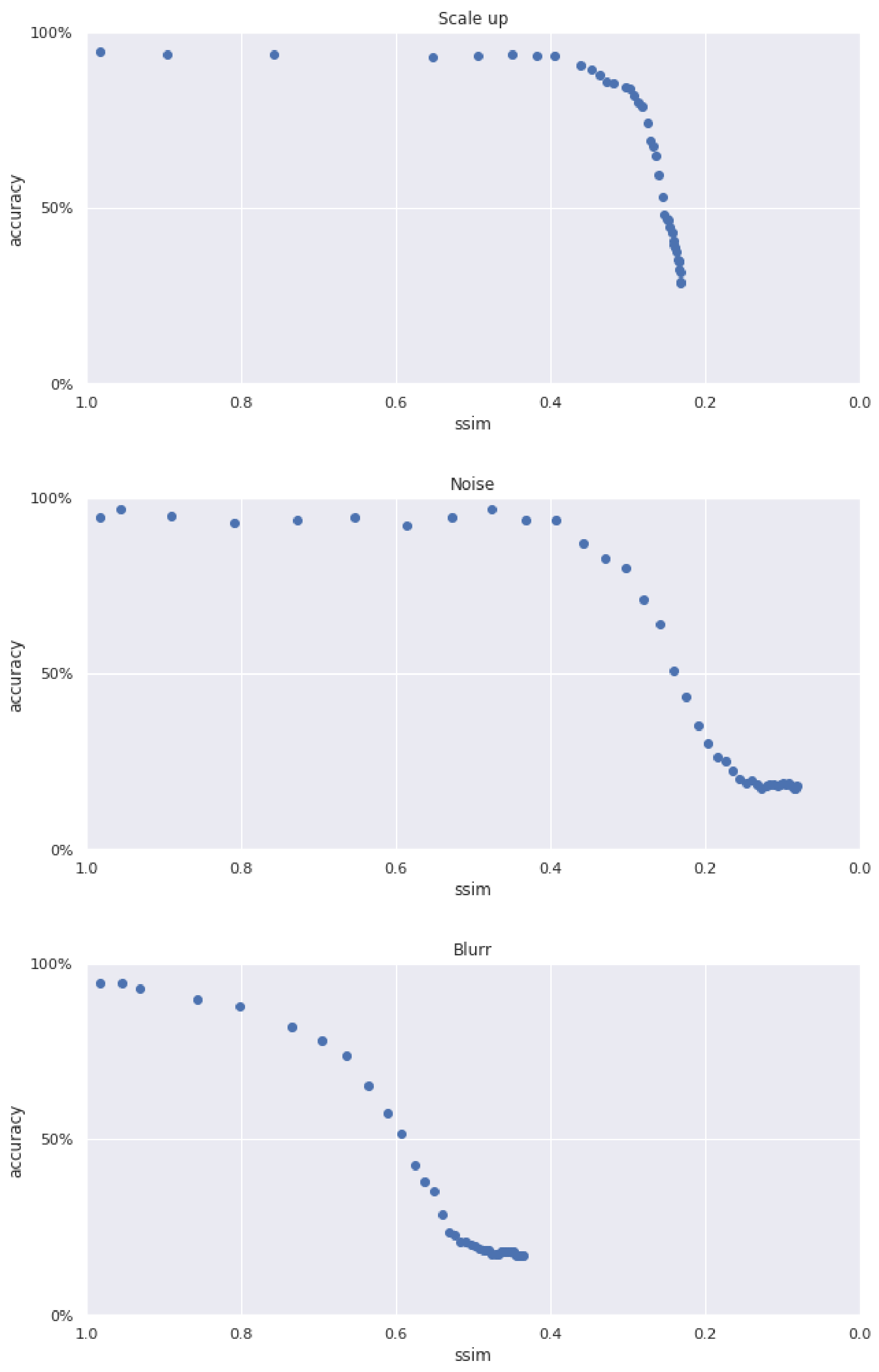


Figure 21. Relation of accuracy and structural similarity index (ssim) for each distortion. (Source: own)

Now we can extract a lot of information from how the corruptions affect the images by studying the Figure 21.

First, we can see that the shape of the curve is similar in the three cases, they have a first phase where small changes of ssim do not affect the accuracy, especially short in the blur effect.

Then in a second phase the slope decreases in concave shape. At this point the accuracy drastically drops in relation to the ssim. It is interesting to see that not in all cases this occurs on the same moment. For scale up and noise it seems to be close to 0.4 while for blur it is around 0.8 and 0.6.

Another difference is the steep of the slope, for noise and blur is somewhat similar but for scale up is almost vertical. We cannot explain for sure the reason of this and it is outside the scope of the current project, however, we estimate that the moment we lose a part of the image with relevant information for the classification is when the accuracy drops, as there is no longer any way to classify the image. Relevant parts of the image might be the shapes surrounding the fetus or parts of the fetus itself.

The third and last phase starts when the curve loses the slope and ends with the last point. In the case of noise and blur the slope completely flattens while on the scale up we see that both accuracy and ssim stop decreasing. We clearly see that the points are much more grouped almost with no more drop in accuracy and ssim. It is easy to understand the lower limit of accuracy but harder to explain why the ssim seems to have a limit too. We will now explore further this situation.

6.2.3. Intensity vs structural similarity (ssim)

To understand how the behavior of the structural similarity index we studied its relation with the corruption intensity. It is obvious that the ssim will go down as the intensity increases but it is interesting to understand how it evolves.

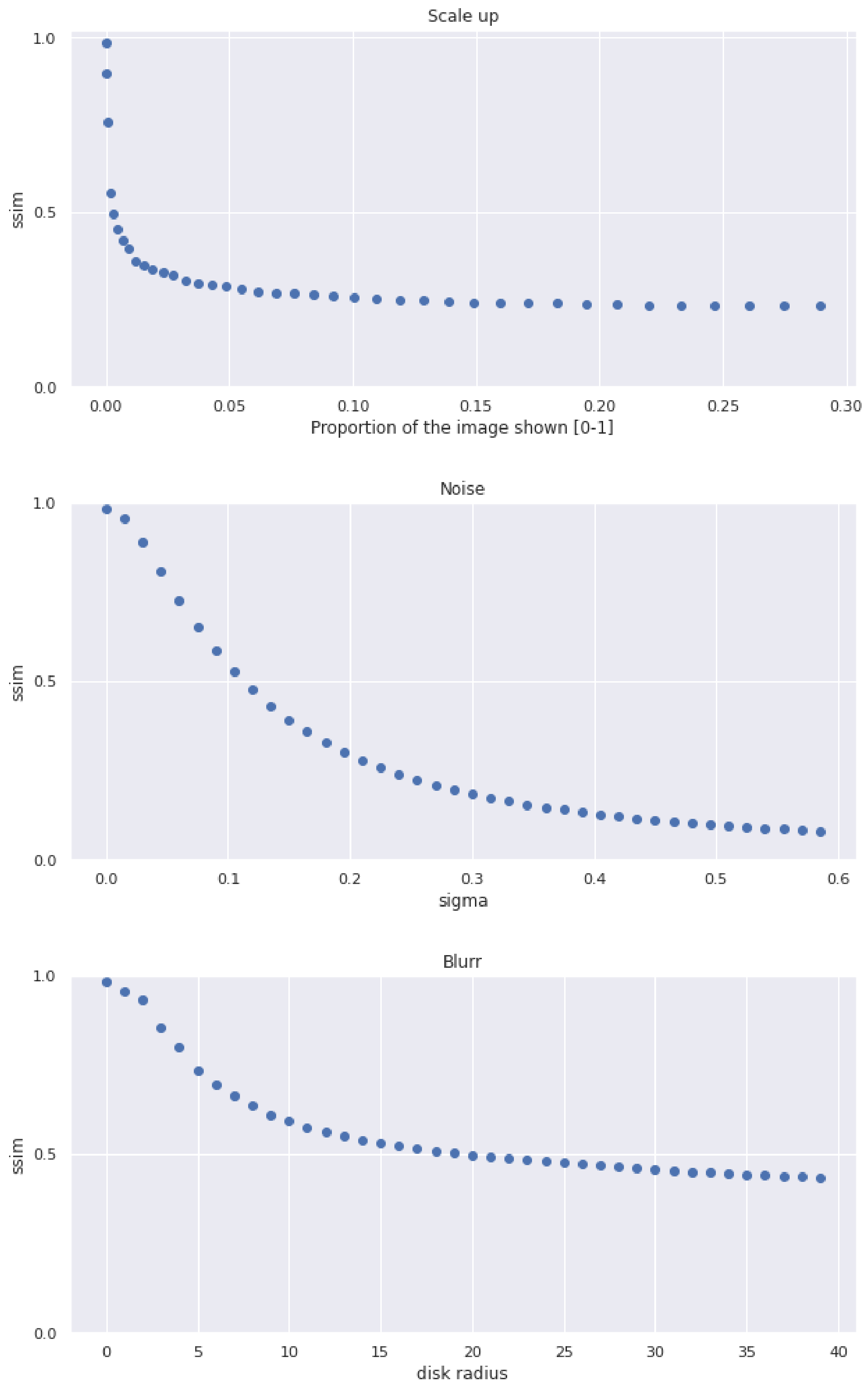


Figure 22. Relation between structural similarity index and corruption intensity. (Source: own)

We clearly observe that at some point while the intensity increases the ssim remains equal.

This explains as well why the ssim drop has a lower limit the image is already “saturated” in distortion. We observe a different behavior for each distortion, below we will explain each of them:

- Scale up: convex shaped curve with steep slope at the beginning. We rapidly get to the point of ssim saturation, meaning the image has lost most of its relation with the original but still keeps some. This is because while the shapes and content are different, they still possess some relation as both images are ultrasounds and show, although partially, the same image. It arrives so fast to the saturation that we cannot even see the flattening of accuracy and ssim in the Figure 21.

- Noise: convex shaped curve that reaches zero ssim. We stated that the noise is created with a gaussian distribution of mean equal to 0 that affects all pixels. As we increase the variance (or sigma) the resulting ssim keeps dropping as we modify all the pixels of the image by some degree until losing all relation to the original. This effect is clearly seen at the most extreme cases in Figure 6 and Figure 20.

- Blur: convex shaped curve that flattens just under 0.5 ssim. This premature flattening is due to the function used to create blur and the definition of the corruption itself. As what we are doing id dissolving the edges the more general shapes remain more or less equal thus, maintaining the relation with the original image.

7. Deep Learning for Image Distortion Recognition

In this section we will use machine learning tools to create a model able to classify distorted images and accomplish our goal. We will take advantage of the learning done with the previous datasets, for example Image Net, to recognize patterns and transfer it to our model. Then we will feed these patterns to predict the class of the input image.

Before going further, we will recap all the steps done and define where we are:

- We have gathered all medical images used by Vicent [6] and provided by the collaboration between the medical staff at Hospital Sant Joan de Déu and the Lassie Lab from UPC.

- Defined the limits between what is considered distortion and correct images.

- Expanded the original dataset with generated distorted images of blur, noise and zoom or scale up.

7.1. Image Preprocessing

From here we can now proceed to train our own machine learning model but first, we must prepare the dataset to fit it. This is a required step because the models we will use are predefined. While we can modify some parts of the model, they do require some specific formatting in the input data.

First of all, we merge the original and modified lists that we previously prepared to have two unique lists: one for images and another one for labels. As we generated this lists in an ordered sequence all the data has a specific order. It will be interesting to avoid this in order to properly train our model.

With the two list we then, use a Keras functionality to split the train and test data. By default, it is set 80% for training and 20% for test. This is actually a pretty common and also good approach so we will not modify it. The build in function also shuffles the dataset so we avoid having a certain order in the training that may cause adverse effects in the result.

```
#---------------------PREPARE DATA FOR CNN----------------------
#Merge lists of images and merge lists of labels
all_images = images_orig + images_mod
all_y = y_orig+y_mod


#Split train and test images
x_train, x_test, y_train, y_test = train_test_split(all_images,all_
y)
```

The result of this process are two groups, one for train and one for test, with a list of images and another of labels. The sizes are quite important to understand the validity of the model. Again, the balance of the number of classes is important and the function automatically balances the result (when possible).

CLASS	**TRAIN (80%)**	**TEST (20%)**
ORIGINAL	434	108
SCALED UP	433	109
BLUR	434	108
NOISE	433	109
TOTAL	**1.734**	**434**

Table 4. Sum of train and test data by class type. The dataset is split 80% for training and 20% for test. (Source: own)

Finally, we reshape the arrays into the format taken by the model. For the images we will convert our list into a numpy array of 174x174x3. While the ultrasounds are greyscale the models are built for colored images so we have to adapt our data to the requirements, as explained previously. The labels, on the other hand, are converted into an array of 4 positions per image each corresponding to a class. As before, we use a function that automatically converts each label that we have (0,1,2,3) to a 1 in a concrete position of the array.

```
#Reshape into np array
x_train=np.array(x_train).reshape(np.shape(x_train)[0],width-
50,height-50,3)
x_test=np.array(x_test).reshape(np.shape(x_test)[0],width-
50,height-50,3)

# Code classes on shape (4,) arrays
y_train = utils.to_categorical(y_train)
y_test= utils.to_categorical(y_test)
```

Now, the data is ready to be used by our model. The next step is creating a model and adapt it to our data so it meets our requirements.

As explained, there are many strategies to implement our model so we will explore and compare some to get the best results. Nevertheless, we cannot assure whether there is a model that can outperform ours.

7.2. VGG16

7.2.1. Introduction

The VGG16 is a convolutional neural network model proposed by Oxford's university Visual Geometry Group (VGG) at [15]. The model became famous for securing the first position at the Image Net Challenge 2014 achieving a 92,7% top 5 test accuracy. It is also known for the good performance when generalizing to other datasets using transfer learning. The model is trained under the ImageNet dataset with 14 million images of 1000 different classes.

Figure 23. Small sample of ImageNet dataset. [8]

The model consists of multiple convolutional layers connected by max pooling filters. The output is then flattened and classified by a fully connected network. The default input image is

244x244 colored image.

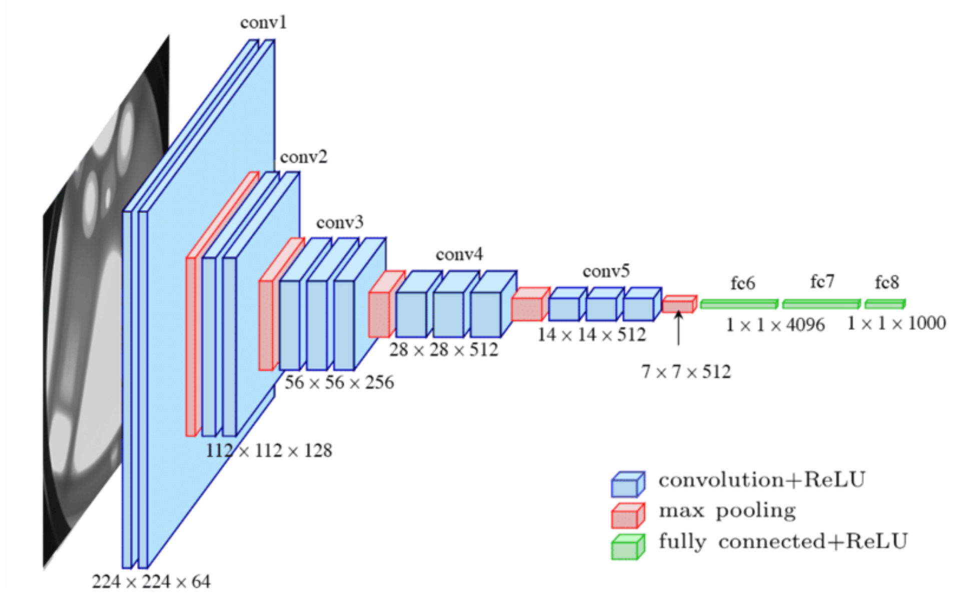


Figure 24. Graphic representation of VGG16 architecture. From the input image to the classification output, left to right.[9]

7.2.2. Model Implementation

With everything set up we can now proceed to create our own model. The model will be a combination of the predefined convolutional neural network VGG16 and a densely connected network. The CNN will extract the patterns and the neural network will predict the class from these.

To begin with, we define a sequential model. This means that each layer that we add goes after the last one and so, all the layers are connected. The output of the previous layer is the input of the new layer. The whole process has an inherent order, from the CNN to the final classifier layer.

The first layer is the predefined VGG16 convolutional neural network. While this is a model itself with its own layers, we can use it as a unique layer with one input and one output. This CNN has its own dense classifier layer however, we will use our own classifier as the shape it has to classify are fairly different from the ones it was made for.

We also have to define the input shape of the images, while we can slightly modify it, we must respect the three-color input.

Finally, the VGG16 CNN returns an output array of 7x7x512. From here the output is flattened in one layer so, as the input is 7x7x512 the flat result is size 1x25.088. Then we create the neural network, with a first layer of 1.024 nodes fully connected to the flattened input. We set the activation to ReLu (Rectified Linear Unit) as it is relatively fast to train and with good performance.

To end with, we connect the 1.024 nodes to 4 output nodes that will serve as classifiers. The activation function in this case is the softmax as we want to have a probabilistic prediction.

```
# Function to create secuential model CNN + Dense classifier
def baseline_model():
    from tensorflow.keras.applications.vgg16 import VGG16
    model=Sequential()
    model.add(VGG16(include_top=False,
input_shape=(width-50,height-50,3)))
    model.add(Flatten())
    model.add(Dense(1024, activation='relu'))
    model.add(Dense(4, activation='softmax'))
    return model
```

Now, from any input image the model will calculate the value of each classification node and give a number from zero to one on the probability of that image to belong to each class.

7.2.3. Train and test

The model is now able to extract relevant patterns from the images but, it is still unable to correctly relate these features to a class. To do so, we will use the train data to feed the model with input images and their correct class, with this the model will learn which patterns define each class adapting the weights of each layer to match the data with the prediction.

To train the model we can tune some hyperparameters that will define the learning process. These include multiple parameters that help to adapt the process to the circumstances, in our case we will modify two basic factors: epochs and batch size. We have set the epochs to one hundred because, for the data we have, it is a safe number that guarantees the model will achieve the maximum training. On the other hand, we set the batch size to a standard measure of 64 (recommended values are 32, 64, 128).

```
# Define model
model = baseline_model()
model.compile(optimizer='adam', loss='categorical_crossentropy', me
trics=['accuracy'])

# Train the model to fit the data and validate the result with the
test
history=model.fit(x_train, y_train, validation_data=(x_test, y_test
), epochs=100, batch_size=64, shuffle=True)
```

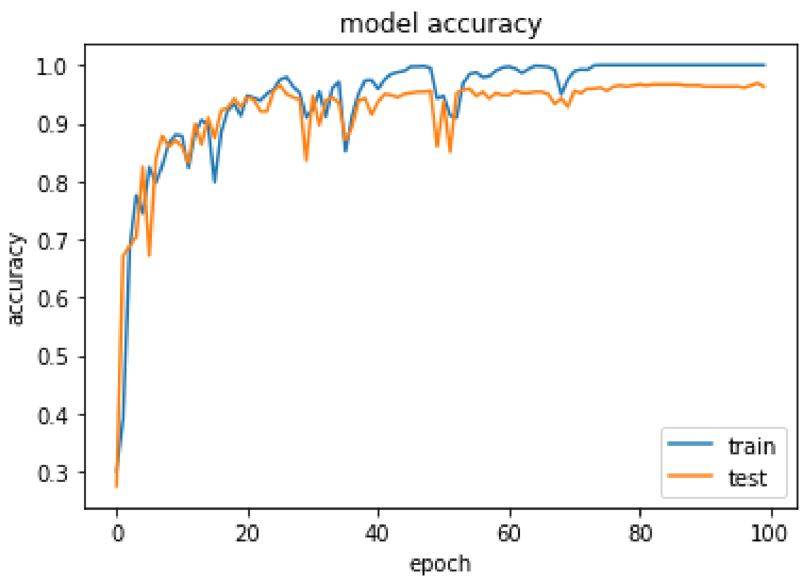


Figure 25. Model with vgg16 CNN training and testing per epoch. (Source: own)

After the training process we can see that the accuracy arrives to its maximum without overfitting. The accuracy is over 90% which we can estimate it is a fairly good result.

7.2.4. Results

To better visualize the results, we plot the confusion matrix for the test data. This will allow us to understand how the model performs for each class.

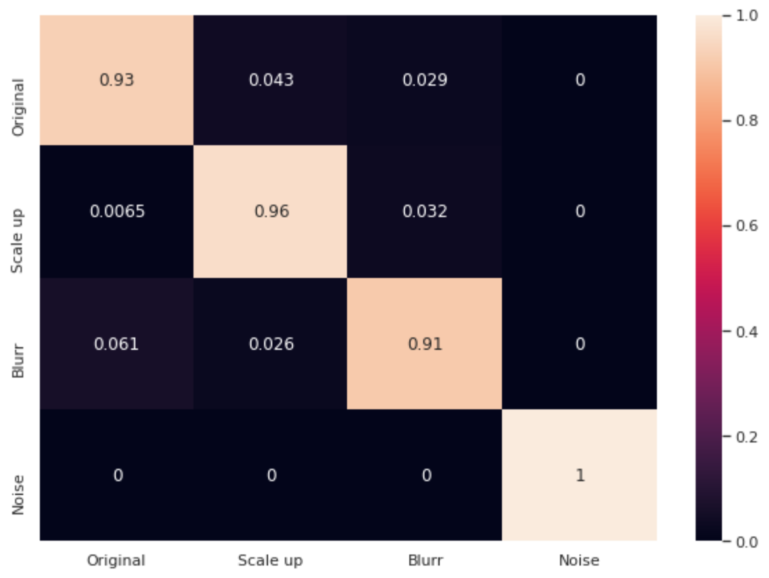


Figure 26. Confusion Matrix by proportions, of the VGG16 model. Y axis represents the predicted class and the X axis the true class. (Source: own)

The results are quite good as the diagonal (top left to bottom right) which represents the correct predictions has the highest success in the four classes. Moreover, the accuracy in all classes is over 90%. Moreover, the noise has 100% accuracy and, as shown on Table 5, it actually has perfect precision and recall. This means that it is a differentiable corruption and the model has never predicted it wrong.

Now we can calculate more metrics to sum up the confusion matrix in a few key performance indicators (KPIs).

```
              precision    recall  f1-score   support

    Original       0.94      0.93      0.94       140
    Scale up       0.94      0.96      0.95       155
        Blur       0.92      0.91      0.92       115
       Noise       1.00      1.00      1.00       132

    accuracy                           0.95       542
   macro avg       0.95      0.95      0.95       542

weighted avg       0.95      0.95      0.95       542
```

Table 5. VGG16 performance summary. (Source: own)

All metrics are above 90% and have good support, we can define the results as good.

7.3. Xception

7.3.1. Introduction

Xception is a convolutional neural network that stands for extreme inception. The name come from one of the previous versions of the network in which is based, the winner of the Image Net Challenge: Inception.

With 71 layers, the model is much deeper than the VGG architecture and is able to achieve better results [3]. It also uses a different methodology called depth wise separable convolutions that dramatically reduces training time by enhancing efficiency rather than reducing parameters or accuracy.

Depth wise convolution divides a normal convolution in two steps. Actually, it is more correct to call it a separable convolution as it divides the operation in depth wise and pointwise convolution.

Depth wise uses a kernel size that operates one channel at a time to create one image with the same channels as the original one.

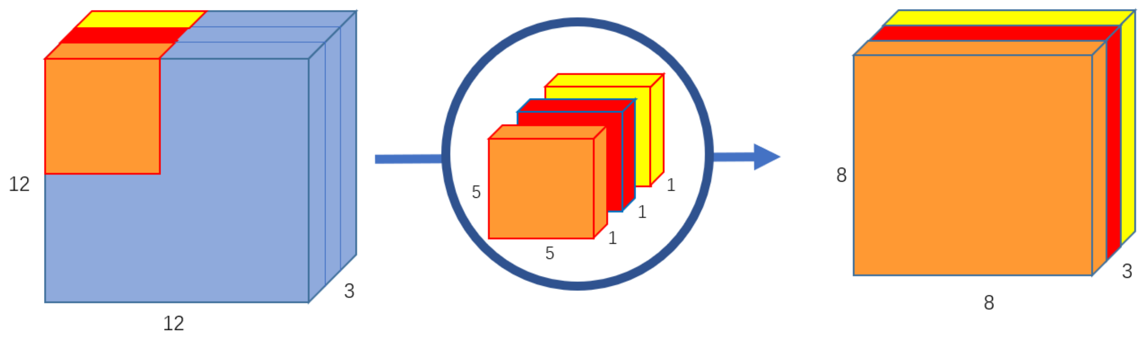


Figure 27. Depth wise convolution, the kernel iterates through each channel and then the result is stacked together. [12]

The pointwise convolution uses a Keras size of one pixel and the number of channels the original image has (a normal rgb would be 1x1x3).

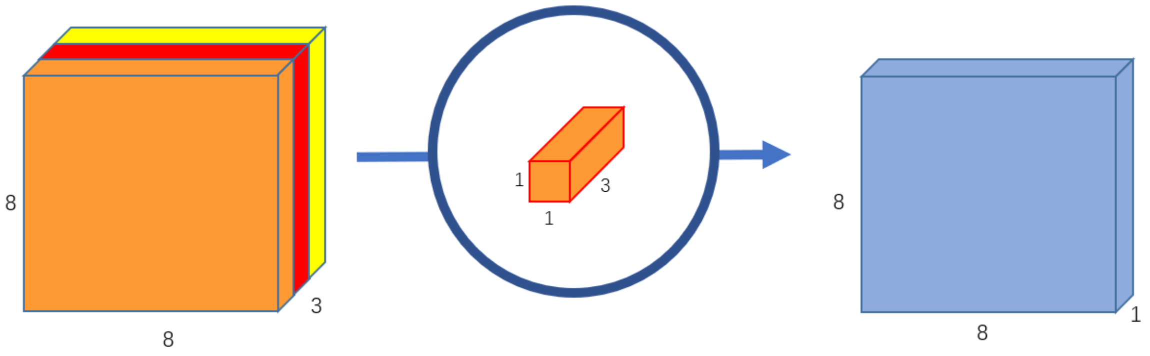


Figure 28. Pointwise convolution, single pixel kernels of depth equal to the input channels. [12]

Then we simply create as many kernels as output channels we need to create the final image.

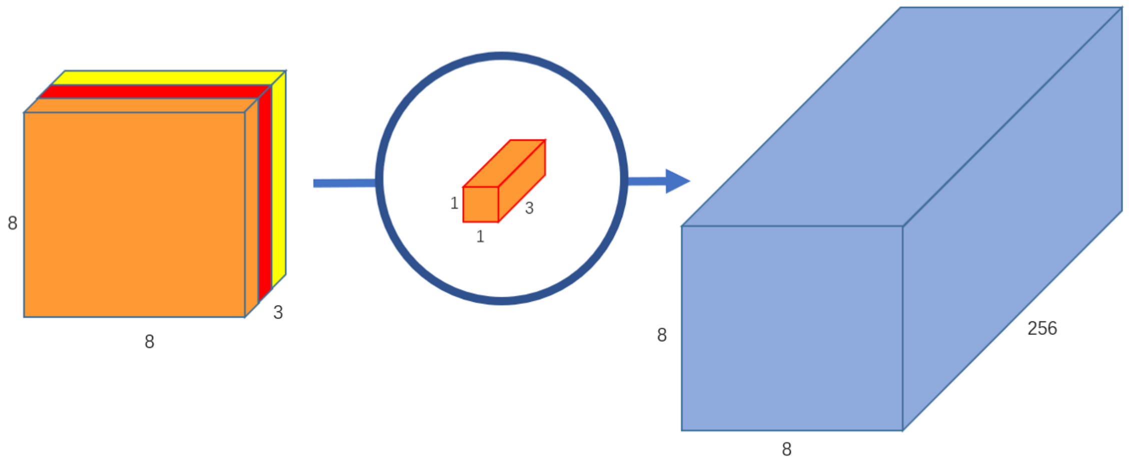


Figure 29. Pointwise convolution with as many kernels as output channels. [12]

One way to see it is that it transforms the image once (depth wise) and then elongates it to the output channels with the pointwise convolution. Thus, this process is much more efficient than the traditional convolution that would transform the image as many times as output channels we have.

The predefined model is accessible at Keras also pretrained with Image Net. The model takes as default input a 299x299 colored image and includes a fully connected layer as classifier (removable).

The 71 layers mix some traditional convolutions with separable convolutions and pooling. It uses the ReLu function to learn and softmax for the classification layers. The full architecture can be divided into three parts: entry, middle, exit as shown in the Figure 30.

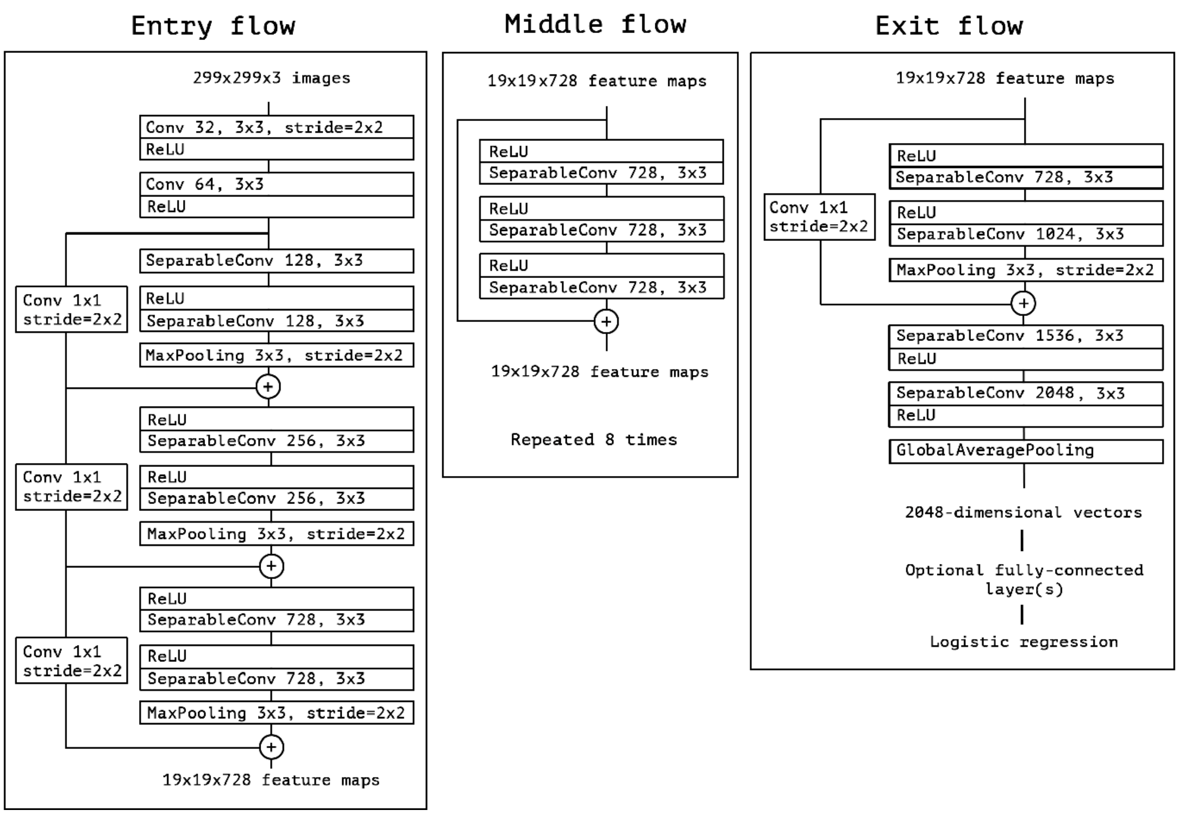


Figure 30. Xception model architecture divided in three parts [3]

7.3.2. Model Implementation

We can reuse almost all the structure prepared for the VGG16. As before, the model will be a combination of the predefined convolutional neural network Xception and a densely connected network. Again, the CNN will extract the patterns and the neural network will predict the class from these.

We will use the same sequential model as before but we will change the first layer so instead of a VGG16 we will use the Xception CNN.

We also have to define the input shape of the images, as said the Xception has a predefined input of 299x299 colored image but, this would require a resizing to increase the shape. As said, this might cause some blurring due to the scalation. Because of this we will keep the same image size.

Now, from any input image the model will calculate the value of each classification node and give a number from zero to one on the probability of that image to belong to each class.

7.3.3. Train and test

We will again reuse most of the code used before to train and validate the model. We maintain the same hyperparameters and train our model.

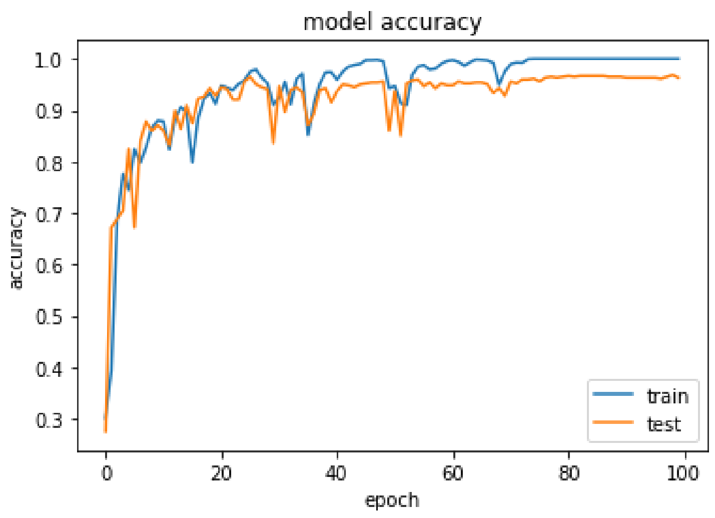


Figure 31. Model with vgg16 CNN training and testing per epoch. (Source: own)

After the training process we can see that the accuracy arrives to its maximum without overfitting. The accuracy is over 90% which we can estimate it is a fairly good result. The learning curve as well as the general accuracy is quite similar to the VGG16.

7.3.4. Results

As before, we plot the confusion matrix for the test data to better visualize the results.

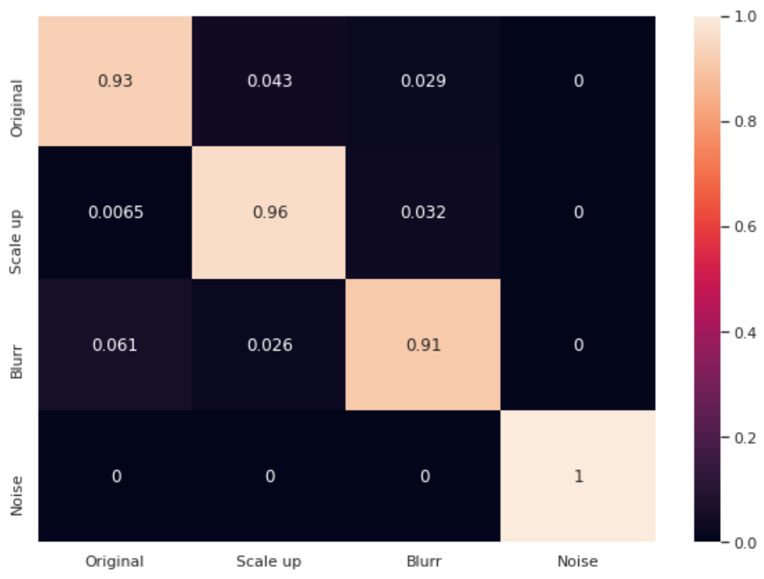


Figure 32. Confusion Matrix by proportions, of the Xception model. Y axis represents the predicted class and the X axis the true class. (Source: own)

Similarly, to before the results are quite good as the diagonal (top left to bottom right) has the highest success in the four classes. Moreover, the accuracy in all classes is over 90%.

Now we can calculate more metrics to sum up the confusion matrix in a few key performance indicators (KPIs).

```
              precision    recall  f1-score   support

    Original       0.94      0.93      0.94       140
    Scale up       0.94      0.96      0.95       155
        Blur       0.92      0.91      0.92       115
       Noise       1.00      1.00      1.00       132

    accuracy                           0.95       542
   macro avg       0.95      0.95      0.95       542

weighted avg       0.95      0.95      0.95       542
```

Table 6. Xception performance summary. (Source: own)

Again, the results are quite similar to the VGG16 for what we can say that both of our models perform good enough for our requirements.

7.4. Results Comparison

From exploring both models we have acquired two valid models with accuracy over 90%. We will prepare and experiment to select one of the models and better understand the results.

One of the advantages of creating the corrupted images is that we have full control on thus corruptions. As so we will now increase our test dataset using a similar methodology of that we used in the previous performance experiment at 6. *Deep Learning with Distorted Images.* With this we will be able to validate our models on a more specific way exploring all possible scenarios.

The only difference with the previous experiment is that we will filter only the original images from the test and from these create all the corruptions to experiment. This is because corruptions are easily created and controlled from original images.

This experiment has two main objectives: check which is the best model and check how does the model perform in each situation. Both will be approached by the same methodology.

7.4.1. Overall accuracy

To expand the results obtained while validating the models we will first test the overall accuracy of the models but with a larger and more detailed test set. As said, we will use the original images from the test and corrupt them in different degrees to obtain a full range of corruptions and validate the model with it. However, this process has a limitation on testing original images as we can only have as many as there are in the first test dataset.

It also important to notice that the images are corrupted from the first value we consider a corruption until the ssim starts to flatten, refer to Figure 22 for relation between intensity and ssim.

Below we compare the two models under the generated test datasets:

```
              precision    recall  f1-score   support

    Original       0.98      0.97      0.98      2622
    Scale up       0.85      0.95      0.90       966
        Blur       1.00      0.87      0.93       690
       Noise       1.00      1.00      1.00       966

    accuracy                           0.96      5244
   macro avg       0.96      0.95      0.95      5244

weighted avg       0.96      0.96      0.96      5244
```

Table 7. VGG16 performance summary under full range of image corruption. (Source: own)

```
              precision    recall    f1-score   support

    Original       0.98      1.00      0.99      2622
    Scale up       1.00      0.96      0.98       966
        Blur       1.00      1.00      1.00       690
       Noise       1.00      1.00      1.00       966

    accuracy                           0.99      5244
   macro avg       1.00      0.99      0.99      5244

weighted avg       0.99      0.99      0.99      5244
```

Table 8. Xception performance summary under full range of image corruption. (Source: own)

Again, we see that while close to one another the Xception outperforms the VGG16by 3% in accuracy. It is also remarkable the high level of accuracy but explained, as we see below in Figure 33, by the extreme cases where the corruptions are at its peak and easily identifiable. While we would probably never have these cases in real life ultrasounds it is interesting for the experiment to understand the whole range.

It is important to notice that at Figure 33 the accuracy is closer to zero when the ssim is close to one. To understand the reason behind we must remember the methodology with which it was created. We start from the original images and start apply increasing corruption to the images, however to see how the accuracy evolves we have labeled each image as corrupted. Obviously the first created images with low corruption should be considered originals, thus the low accuracy.

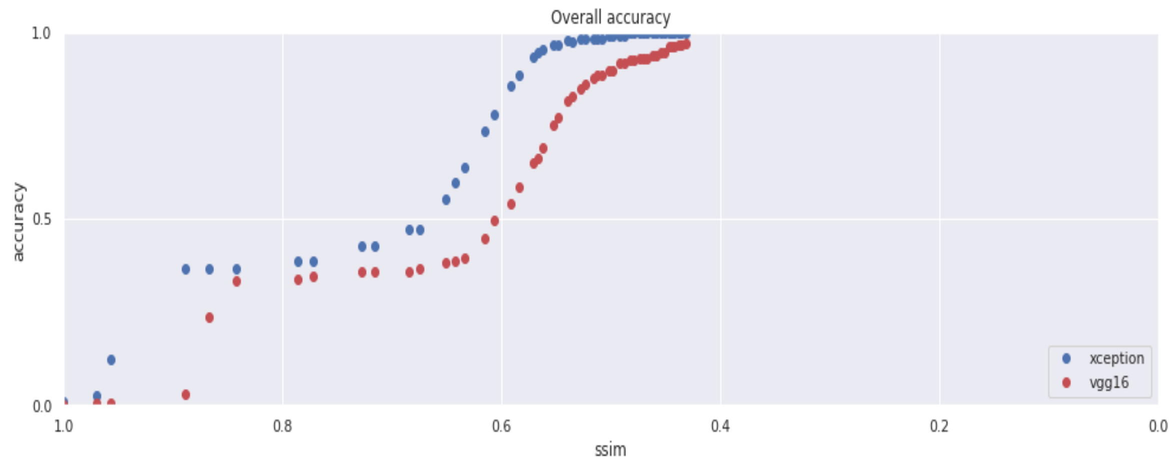


Figure 33. Average accuracy and ssim, all images are label as corrupted. (Source: own)

With Figure 33 we can now confirm that the Xception model performs better overall than the VGG16. Nevertheless, we have to validate that the model is good enough for each corruption.

7.4.2. Accuracy by distortion class

Now we will study the behavior with each corruption. As we cannot define the range of usual corruption in ultrasounds, we will have to study the full range and then pay especial attention to the limit values that we defined to distinguish corrupted images from originals.

We will now plot the result of accuracy for the three corruptions but taking into account the limit values of each distortion. The ideal result would be, obviously, a straight line of 100% accuracy. However, we can expect the accuracy to drop near the limit as the algorithm may not be able to distinguish the thin line of difference between corruption and unmodified image. This is due to the fact that we set a single value as limit instead of a range.

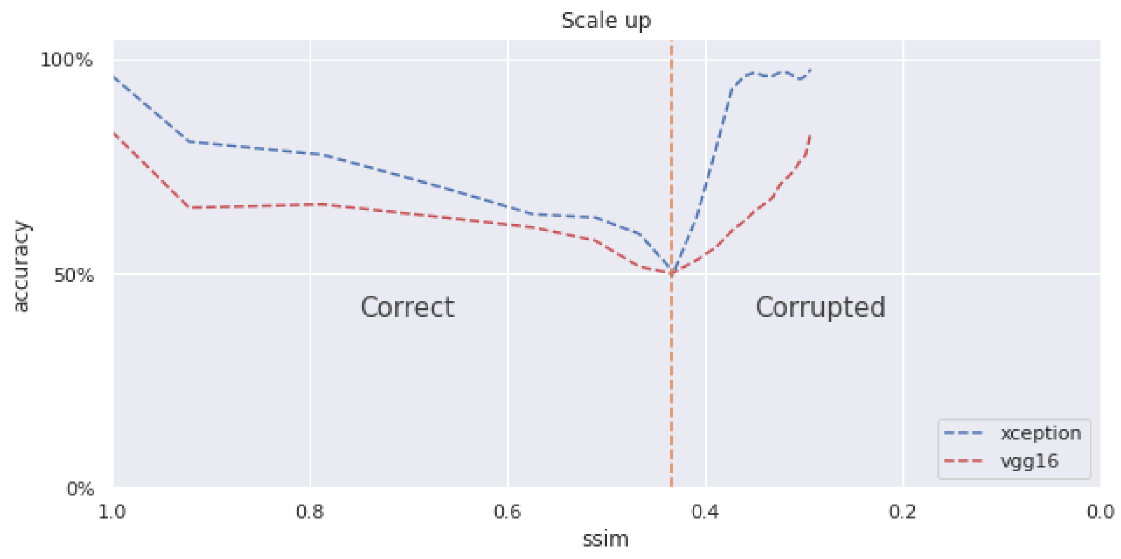
Looking the results at Figure 34 we confirm our expectations on the output. In every corruption we distinguish three major phases:

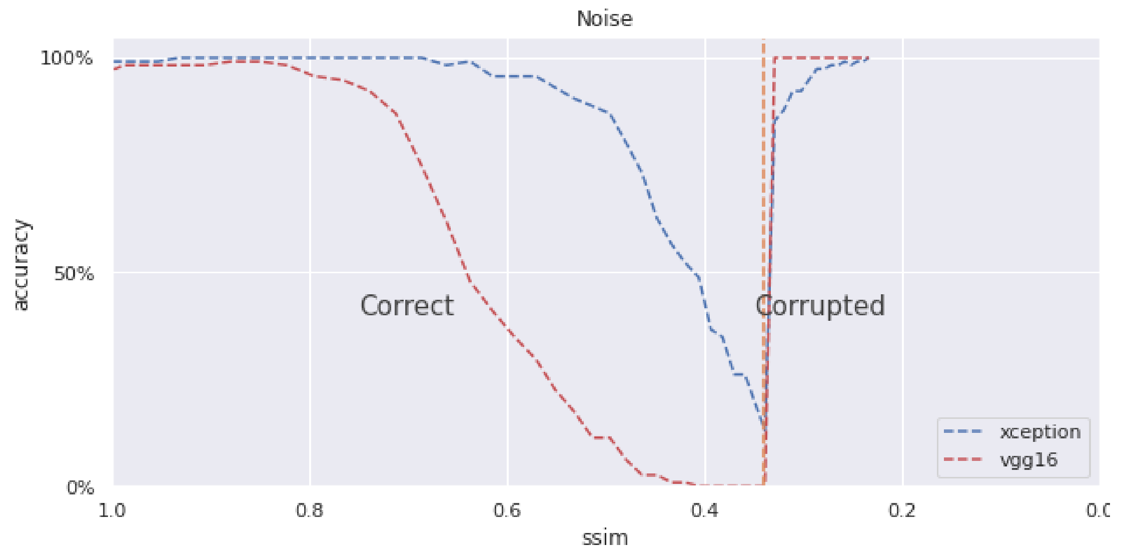
A first stage at which the accuracy is fairly high, both noise and blur have a 100% accuracy while the scale starts at 100% and drops slowly to 85% by the end of the phase. This phase is reason of the low intensity corruptions where the images are quite similar to the originals. The algorithm can identify most of the images as correct but start to fail in some of them.

A second phase where the accuracy has a major drop. Close to the limit threshold value the algorithm identifies all distortions but they are still labeled as correct. As we expected, this is because the value that we set as limit is not perfectly aligned with the model capabilities. A minor change in our decision boundaries might have a huge impact on the results.

When the corruption arrives to the classification limit the result have a drastic nonlinear change as all images are now labeled with distortion. Again, this is because the limit is a single value not perfectly aligned with the model. We can observe that this change is mirrored like, if the model was at 20% accuracy it suddenly passes to 80%. This is inherent to the model as all images are different and setting an exact point to swap from one class to another is practically unachievable.

The last phase the accuracy it is again at its peak as all images are highly corrupted and thus, easily identifiable by the algorithm. This phase could continue until the maximum value of distortion but it would be highly impractical and complicated due to computer power limitations.





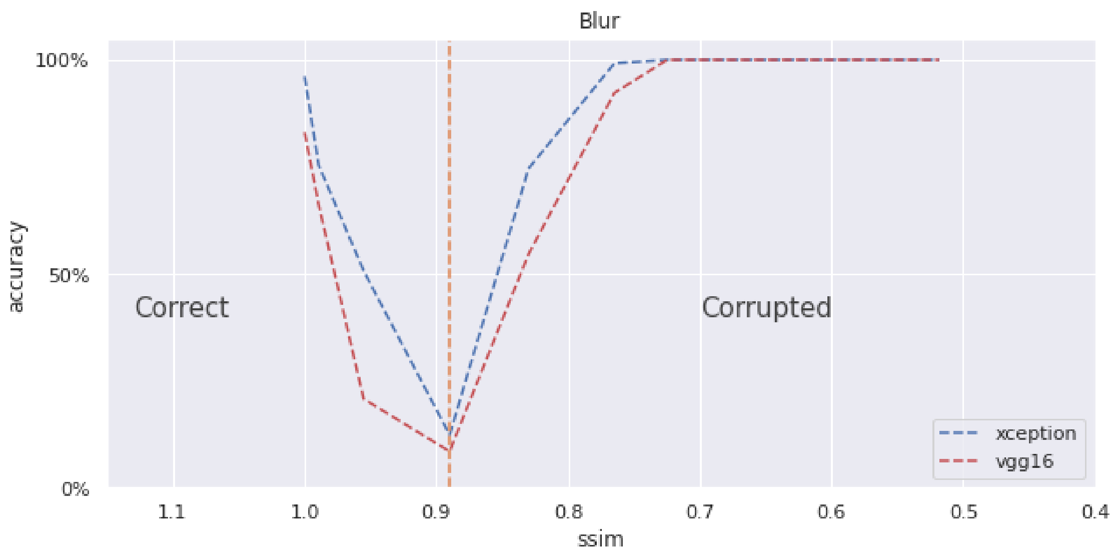


Figure 34. Comparison of accuracy evolution vs ssim of both models. (Source: own)

Comparting one model to the other we observe again similar results yet some differentiable characteristics.

First of all, we see that the accuracy of the Xception model is superior at the first phase on all cases.

On the second phase there is a huge different in the noise as it starts far before on the VGG16 thus, the Xception model is far more accurate and closer to the limit for the noise artifact.

The third and final phase, is again quite similar with both models giving good results.

We conclude that the Xception model is slightly better and have a fairly good accuracy. As the limit of corruption is a single point it produces a drastic drop in accuracy and a mirror like effect.

Nevertheless, we estimate that this drop is of minor importance as the classification close to the limit could easily be interchangeable meaning that an image classified as corrupted could probably be interpreted as correct without major consequences. The real problem are images further from the boundary that present serious corruption, as seen, these are easily identified by our model.

Overall, we conclude that the model is successful and the goal of identifying and classifying corruption is accomplished.

8. Environmental Impact

The environmental impact of this project is fairly low. There are two possible sources to track: one is the hardware equipment used and the other is the electricity usage to run the program.

The hardware is a full workstation of multiple components all of which have a useful life far of roughly 10 years, far larger than the project itself. The most hazardous part is the battery, with potentially toxic chemicals inside as lithium, nickel or cadmium to say some.

When not recycled these substances can pollute land fields, rivers, lakes, oceans and even air by combustion as some are highly volatile. Correctly recycling the battery and all computer parts will drastically reduce the pollution footprint of the workstation.

On the other hand, the electricity usage is quite low. The workstation has a power of 70W roughly 30 times less than an oven. We cannot estimate the power consumption of Google Collaboratory servers as the data is confidential but, as Google is buying all of its energy from renewable sources (solar and wind farms mainly) we can expect lesser environmental impact.

9. Conclusion

This project has been structured in several sub projects, following we list its goals and results:

- First of all, we required to define a **limit value to differentiate corrupted images** from correct ones. By representing the effect of the distortions on the previous classification model created by Vicent at [6] we determined the boundary value for each corruption.

- To understand the project own limitations, we need to **understand how corruptions behave and its extreme values**. We represented the relation between the structural similarity index (ssim) and the intensity degree of corruption applied.

- The main objective was to identify and **classify corruptions in fetal ultrasound images**. By combining a convolutional neural network and a fully connected artificial neural network we managed to obtain over 90% of accuracy on the three classes of distortion.

- Additionally, to the main objective we wanted to **obtain the best results by comparing different models**. We created both models and train – test them in a full range of possible corrupted images to obtain the best model.

We believe that a next step to the project could be to reuse the corruption production code to create a Generative Adversarial Network (GAN) to remove the corruption from images. Research on this fairly new technology have achieved impressing results removing rain or snow from images [19] and filling gaps (inpainting) [2]. With this we would not only detect the corruptions but also fix them.

From now on, this project can serve as cleaning tool for medical databases, as it can discard corrupted images. It can be specially interesting when combined with other classification models to correctly classify a full data base. Our model can prepare the database to assure the next classification is done only over correct data.

With all this, we state that the goal of the project has been achieved. It can now be implemented to clean databases or help in the selection of ultrasound fetal images. Further work can be done to combine this project with a classifier to automate a full classification process. With this a complete algorithm could help sanitary professionals and researchers on their tasks.

Budget

Reason	Description	Qty	Unit Price	Price
Programming	Data Generation	50 h	50 €/h	2.500,00 €
	Corruption Analysis	100h	50 €/h	5.000,00 €
	Model Development	180h	50 €/h	9.000,00 €
	Result Analysis	60 h	50 €/h	3.000,00 €
	Total	**390h**	**50 €/h**	**19.500,00 €**
Documentation	Report	120 h	50 €/h	6.000,00 €
	Total	**120**	**50 €/h**	**6.000,00 €**
Equipment	Workstation	1	1.000 €/un	1.000,00 €
	Programing	-	Free	- €
	Windows and Office	1	150 €/un	150,00 €
	Total	**-**	**-**	**1.150,00 €**
TOTAL				**26.650,00 €**

Bibliography

References

[1] Andrew, Ng, *Seminar 2: Linear regression with one variable*, lecture notes, Machine Learning, Stanford University, 21 May 2020.

[2] D. Pathak, P. Krähenbühl, J. Donahue, T. Darrell and A. A. Efros, "*Context Encoders: Feature Learning by Inpainting*," 2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR), Las Vegas, NV, 2016, pp. 2536-2544, doi: 10.1109/CVPR.2016.278.

[3] F. Chollet, "*Xception: Deep Learning with Depthwise Separable Convolutions*," *2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, Honolulu, HI, 2017, pp. 1800-1807, doi: 10.1109/CVPR.2017.195.

[4] GitHub [online]. GitHub Blog. *The State of the Octoverse: machine learning*, 2019. [https://github.blog/2019-01-24-the-state-of-the-octoverse-machine-learning/, 24th may 2020].

[5] GitHub [online]. Stanford Course: *Convolutional Neural Networks for Visual Recognition*, 2020. [https://cs231n.github.io/convolutional-networks/, 2nd june 2020].

[6] Gregori. Vicent, "*Classificació Automàtica de plans ecogràfics en ecografies prenatals*", Final Degree Thesis on Biomedic Engineering, Barcelona, UPC, 2018

[7] HUBEL, D H, and T N WIESEL. "*Receptive fields, binocular interaction and functional architecture in the cat's visual cortex*." *The Journal of physiology* vol. 160,1 (1962): 106-54. doi:10.1113/jphysiol. 1962.sp006837

[8] Image-net [online]. [https://image-net.org/, 10 june 2020]

[9] Kamruzzaman, Abu. (2018). *A Comparative Study of Convolutional Neural Network Models with Rosenblatt's Brain Model*.

[10] Krizhevsky, Alex & Sutskever, Ilya & Hinton, Geoffrey. (2012). *ImageNet Classification with Deep Convolutional Neural Networks*. Neural Information Processing Systems. 25. 10.1145/3065386.

[11] MathWorks [online]. Deep Learning. Convolution Neural Networks, [https://www.mathworks.com/solutions/deep-learning/convolutional-neural-network.html, 12 june 2020]

[12] Medium [online]. Towards Data Science. *A basic introduction to separable convolutions*, 2018. [https://towardsdatascience.com/a-basic-introduction-to-separable-convolutions-b99ec3102728, 3 june 2020]

[13] Medium [online]. Towards Data Science. *Linear Regression for machine learning*, 2018. [https://towardsdatascience.com/how-does-linear-regression-actually-work-3297021970dd, 5 june 2020]

[14] S. Bendali, “Biometric Plane Classification in Fetal Ultrasound Scan”, Final Degree Thesis on Biomedic Engineering, Barcelona, UPC, 2017

[15] Simonyan, Karen & Zisserman, Andrew. *Very Deep Convolutional Networks for Large-Scale Image Recognition*. arXiv 1409.1556. . (2014).

[16] Turing A.M. (2009) *Computing Machinery and Intelligence*. In: Epstein R., Roberts G., Beber G. (eds) Parsing the Turing Test. Springer, Dordrecht

[17] University of Oxford [online]. Department of Engineering Science Page. Visual Geometry Group [http://www.robots.ox.ac.uk/~vgg/, 29th may 2020].

[18] Yamashita R, Nishio M, Do RKG, Togashi K. *Convolutional neural networks: an overview and application in radiology*. Insights Imaging. 2018;9(4):611-629. doi:10.1007/s13244-018-0639-9

[19] Zhang, He & Sindagi, Vishwanath & Patel, Vishal. (2017). *Image De-raining Using a Conditional Generative Adversarial Network*. IEEE Transactions on Circuits and Systems for Video Technology. PP. 10.1109/TCSVT.2019.2920407.

[20] Zhou Wang, A. C. Bovik, H. R. Sheikh and E. P. Simoncelli, "*Image quality assessment: from error visibility to structural similarity*" in IEEE Transactions on Image Processing, vol. 13, no. 4, pp. 600-612, April 2004, doi: 10.1109/TIP.2003.819861.