UNIVERSIDAD POLITÉCNICA DE MADRID

ESCUELA TÉCNICA SUPERIOR DE INGENIEROS EN TOPOGRAFÍA, GEODESIA Y CARTOGRAFÍA

TITULACIÓN DE GRADO EN INGENIERÍA DE LAS TECNOLOGÍAS DE LA INFORMACIÓN GEOESPACIAL

TRABAJO FIN DE GRADO

Influence of tile size on the quality of recognition and extraction of roads and paths in orthophotographs with deep learning

Madrid, septiembre 2021

Alumno: Víctor Fernández Valladolid

Tutores: Calimanut-Ionut Cira

Miguel Ángel Manso Callejo

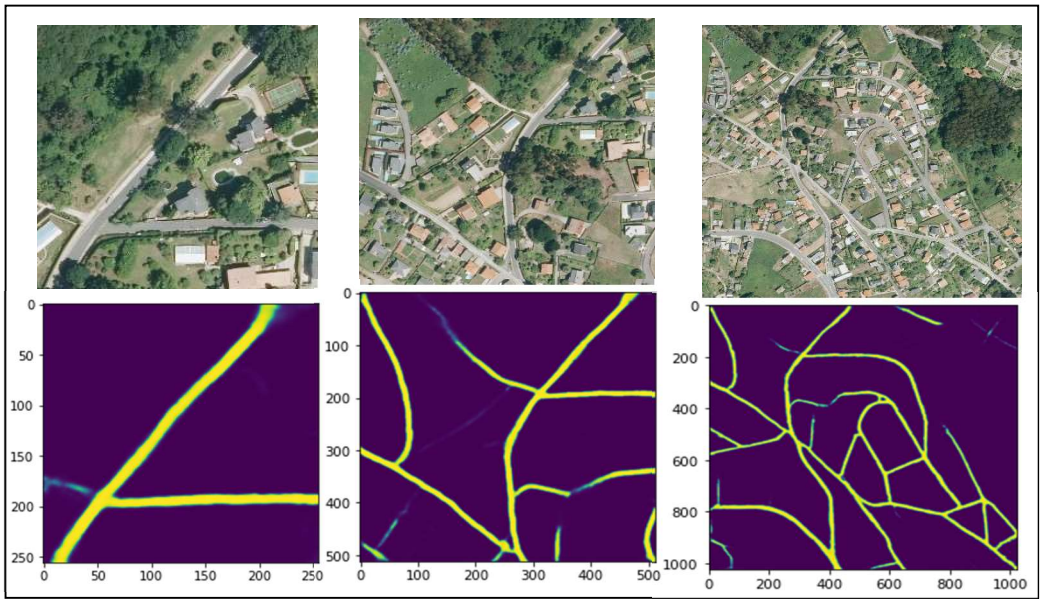
UNIVERSIDAD POLITÉCNICA DE MADRID

ESCUELA TÉCNICA SUPERIOR DE INGENIEROS EN TOPOGRAFÍA, GEODESIA Y CARTOGRAFÍA

TITULACIÓN DE GRADO EN INGENIERÍA DE LAS TECNOLOGÍAS DE LA INFORMACIÓN GEOESPACIAL

TRABAJO FIN DE GRADO

Influence of tile size on the quality of recognition and extraction of roads and paths in orthophotographs with deep learning



Madrid, septiembre 2021

Alumno: Víctor Fernández Valladolid

Tutores: Calimanut-Ionut Cira

Miguel Ángel Manso Callejo

AGRADECIMIENTOS

Estas líneas van dedicadas a las personas que me han apoyado tanto en la realización de este trabajo final y estas prácticas, como en la etapa universitaria.

En primer lugar, quería agradecer a mi familia, así como a mis amigos, por apoyarme al comienzo de esta etapa, cuando mi futuro académico era un tanto incierto.

En segundo lugar, quería destacar el fantástico trato que se me ha brindado a mi y a mis compañeros y compañeras en esta escuela, que se ha mostrado atenta y comprensiva en diversas situaciones.

Por último, quería agradecer a mis tutores de este trabajo por haberme acogido en las prácticas de la universidad y haberme dado la oportunidad de profundizar en este innovador ámbito, que ha definido mis intereses en los próximos años.

ABSTRACT

The road and highway network represent an important part in the transport network, and requires a constant update. This is not often an easy task to do as, for example, in the case of secondary road networks, the cartographic representation is a difficult task to perform, due to numerous obstacles that arises during the process: a variety of path's widths, the presence of occlusions or poorly defined edges and layouts ; for this reason, convolutional neural networks are being used which, from an input of images from different tile sizes (such as 256, 512 or 1024) of high resolution orthoimages, this challenge is solved. On the other hand, semantic segmentation networks are employed to generate the geometries from the previously mentioned different sizes of tiles. The purpose of this work is to study which tile size (256x256, 512x512, 1024x1024) is more appropriate for obtaining a more accurate cartographic representation. For this analysis, three different artificial neural networks are used for each tile size and Deep Learning task (either Image Recognition or Segmentation). For image recognition, VGG-v1, VGG-v2 and VGG-from-scratch have been trained. In the case of semantic segmentation, two architectures based on U-Net have been implemented: SeResNeXt50 and InceptionResNetv2, and one that uses LinkNet as encoder: EfficientNetb5. The optimal tile size has been selected based on the metrical comparison of the performance achieved by the models on unseen data in terms of loss, accuracy, recall, precision, F1-score, AUC-ROC or IoU score. In the classification or image recognition task, the highest results are obtained by the models trained on 512x512, with a remarkable difference with respect to the other two sizes. In the case of the semantic segmentation operation, in addition to a quantitative interpretation, a qualitative analysis by means of perceptual validation has been carried out in order to assess the significance of the computer performance metrics. Once these two points of views have been taken into consideration, it has been proven that, once again the intermediate size delivers the best results. Hence, after this work, it can be affirmed that, if a large-scale extraction and recognition of the roads is pursued, I recommend considering a tile size of 512x512.

RESUMEN

La red de caminos y carreteras tiene una gran importancia en la red de transportes, y requiere de una constante actualización. A veces, esto no es tan fácil de realizar porque, por ejemplo, en redes de carreteras secundarias, la digitalización y representación cartográfica resulta una ardua tarea debido a numerosos obstáculos que surgen durante el proceso: distintas anchuras de los caminos, presencia de oclusiones o trazados pobremente definidos; por este motivo, mediante redes convolucionales que toman de entrada imágenes de distintos tamaños de tesela (256x256, 512x512 o 1024x1024) de alta resolución, se realiza esta actualización. Por otro lado, las redes de segmentación semántica se encargan de generar las máscaras de dichos caminos a partir de las teselas previamente mencionadas. Para llevar a cabo este análisis, tres redes neuronales artificiales se usan para cada tamaño de tesela, así como dependiendo de la tarea de Aprendizaje Profundo a realizar (reconocimiento o segmentación semántica). Para reconocimiento de imágenes se usa las redes VGG-v1, VGG-v2 y unas VGG que se han construido desde cero. En cuanto a la segmentación semántica, se han usado dos arquitecturas que se apoyan en U-Net: SeResNeXt50 e InceptionResNetv2, y una que usa LinkNet como codificador: EfficientNetb5. El tamaño óptimo de tesela se ha seleccionado basado en la comparación de métricas en los resultados de cada modelo en imágenes que la red no ha visto previamente en función de la pérdida, precisión, exhaustividad, exactitud, F1-score, AUC-ROC, o IoU score. En la tarea de clasificación o reconocimiento, el tamaño de 512x512 píxeles es el que mejor ha rendido, con una diferencia notable respecto a los otros dos tamaños. Para la operación de segmentación semántica, se ha requerido de un doble análisis debido a que la diferencia de resultados no era tan amplia, una interpretación cuantitativa, y otra cualitativa. Con ambos puntos de vista, se ha comprobado que nuevamente el tamaño intermedio es el que devuelve mejores resultados. Por ello, tras este trabajo, se puede decir que, si ha de realizarse una extracción a gran escala y reconocimiento, recomendaría las orto imágenes con tamaño de tesela de 512x512.

Table of Contents

List of Tables

Acronyms and abbreviations

CNN: Convolutional Neural Network

IoU: Intersection over Union

AUC-ROC: Area under the ROC Curve

ROC: receiver operating characteristic curve

ANN: Artificial Neural Network

ReLU: Rectified Linear Unit

ILSVRC: ImageNet Large Scale Visual Recognition Challenge

ResNet: Residual Network

MTN50: Mapa Topográfico Nacional 1:50,000 scale

RAM: Random Access Memory

GB: Gigabyte

GPU: Graphics Processing Unit

IGN: Instituto Geográfico Nacional

VAT: Value Added Tax

1. INTRODUCTION

1.1. Background

The theme of this project has been decided on the basis of two lines of research carried out by members of the research group with which I have collaborated. Firstly, “A Framework Based on Nesting of Convolutional Neural Networks to Classify Secondary Roads in High Resolution Aerial Orthoimages" (Cira et al., 2020) by Calimanut-Ionut Cira, Ramón Alcarria, Miguel Ángel Manso and Francisco Serradilla. This article introduces the combination of Remote Sensing and Deep learning as a tool to interpretate images and to compute structures. The roads network and roads one is of high importance in the transport network, and it has to be updated quite often. It is proposed the used of Convolutional Neural Networks (CNN) taking as input data high-resolution images divided into 256x256 pixel tile-size.

Then, on second place, "A Deep Learning-Based Solution for Large-Scale Extraction of the Secondary Road Network from High-Resolution Aerial Orthoimagery" by Calimanut-Ionut Cira, Ramón Alcarria, Miguel Ángel Manso and Francisco Serradilla proposes the fact of the secondary roads’ networks being the widest of transport networks, and due to different reasons (occlusions, not having a proper drawing, or different widths), the digitalization process and cartographic representation turn to be complex. The use of hybrid segmentation networks trained with high resolution images and its mask is proposed as a solution to this issue.

1.2. Objectives

I consider the following starting hypothesis:

1. The quality of the results obtained after an image segmentation or image recognition task (both deep learning tasks), applied to aerial ortho-images, depends on the image tile size, being it 256x256 pixels per image, 512x512 or 1024x1024 in my case.

In order to test this hypothesis, I am pursuing a general objective, which at the same time, can be split into specific targets.

I. Compare the different metrics which are obtained by evaluating the different model architectures for the three different tile size images. Once the best

models for each task are selected, it is easier to have a visual understanding of the influence of the tile size. The procedure has been the following one:

a. Starting from a labelled dataset, I have split the data 90-5-5% amongst three categories training (90%), validation and testing. As far as image recognition is concerned, I am using the images labelled as “roads” or “no-roads”, on the other hand, for image segmentation, I am making use of the images containing roads and the mask generated from this picture.
b. Choose three different architectures for image recognition and then experimenting with the most widely-used semantic segmentation architectures (or encoder-decoder neural networks). Once the architectures are defined, they are going to be present in the three tile size images training.
c. Evaluate the performance of the models by calculating metrics such as IoU Score, F1-Score, Recall, Precision, AUC-ROC or Accuracy. This operation is followed by an interpretation of the results, selecting the best ones and deciding which ones need to be repeated, in case the metrics are really distant from the others.
d. Retrain the models displaying a bad performance by modifying hyperparameters like the batch size, learning rate, number of epochs, number of neurons or not including activation layer in case it is necessary.
e. After selecting which models fit the best with each tile-size and task, it is time to test them with unseen data from different areas: Mediterranean, dry areas, urban, rural, coastal and green spaces.

2.THEORICAL FRAMEWORK AND LITERATURE REVIEW

2.1 Artificial Intelligence, Machine Learning and Deep Learning

2.1.1 Artificial Intelligence

Artificial intelligence is a branch of computer science which is based on the computer performing tasks that usually require human intelligence. Another interesting definition, in this case given by François Chollet says: "*artificial intelligence is the effort to automate intellectual tasks normally performed by humans"* (Chollet, F. (2017). Deep learning with python. Manning Publications). At the same time, as shown in Figure 1, artificial intelligence can be divided into smaller sections.

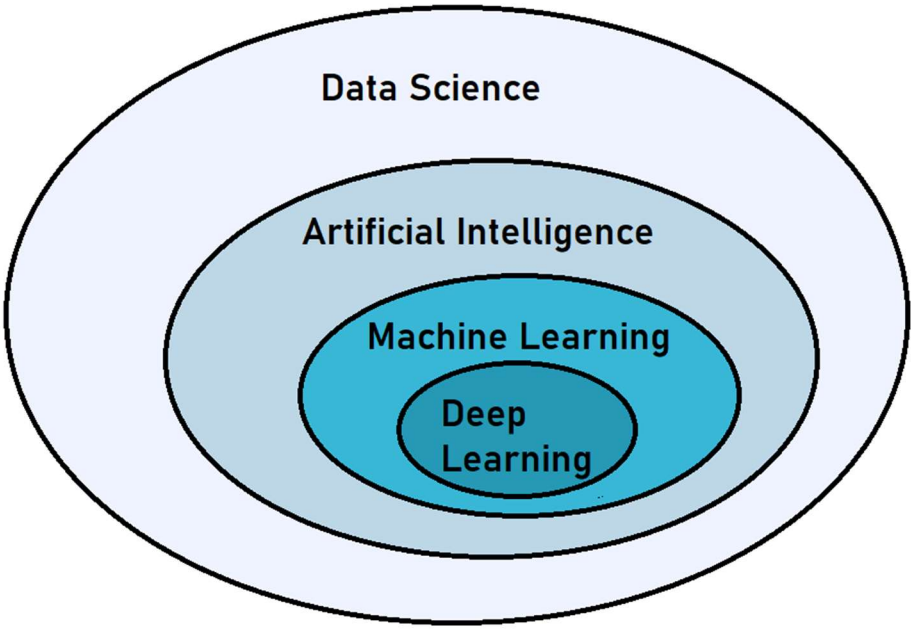


Figure 1: Data Science, Artificial Intelligence, Machine Learning and Deep Learning.

Source: Own Elaboration

2.1.2 Machine Learning

Machine learning, at the same time is a branch of Artificial Intelligence and consists of the computer being able to generate rules and recognize patterns after a training process on given data. Regarding the learning methods used in machine learning, the most popular are unsupervised learning techniques or supervised learning. Supervised learning requires labels in the data so that the machine can differentiate between different classes.

Some of the main tasks that can be solved using machine learning are regression, classification and clustering. These tasks follow a certain procedure, which is described in Figure 2.

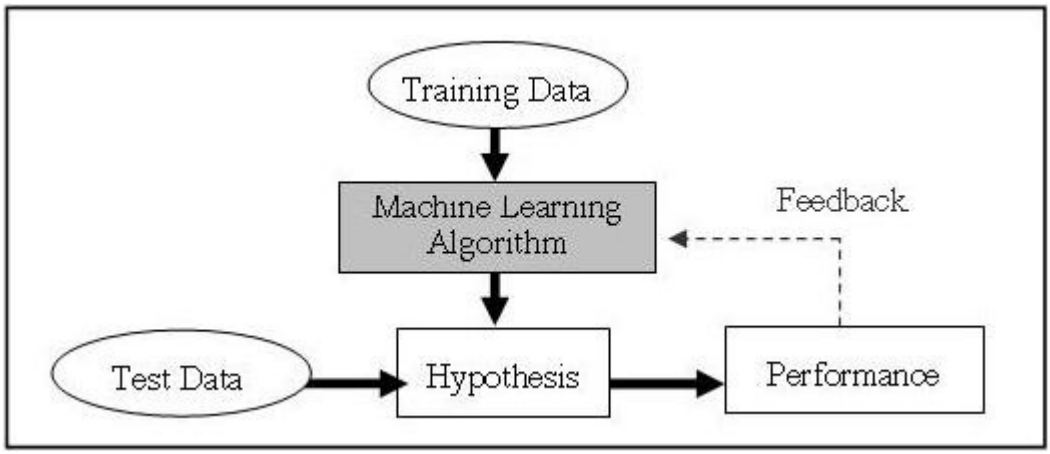


Figure 2 Training procedure of a machine Learning algorithm (Jinapattanah, 2012)

This figure basically explains how machine learning generates a hypothesis after processing the given data during training, and this hypothesis performs better or worse depending on how accurate the statement is. The quality of this assumption is calculated by testing separated data which has not been processed during training (unseen data).

2.1.3 Deep learning

Deep learning is a field of machine learning which brings the novelty of forcing the machine to learn patterns in the data through successive layers of increasingly relevant representations. Usually, each layer is going to send its output to the following layers. François Chollet defines deep learning as "*a multistage way to learn data representations"* (Chollet, 2018).

In Figure 3, there can be seen the differences between the way machine learning extracts the features and the way deep learning does it. Feature extraction on the other hand is the process of obtaining the more relevant information from a dataset. In machine learning models, the feature extraction is done outside the algorithms, whereas in deep learning it can be done automatically.

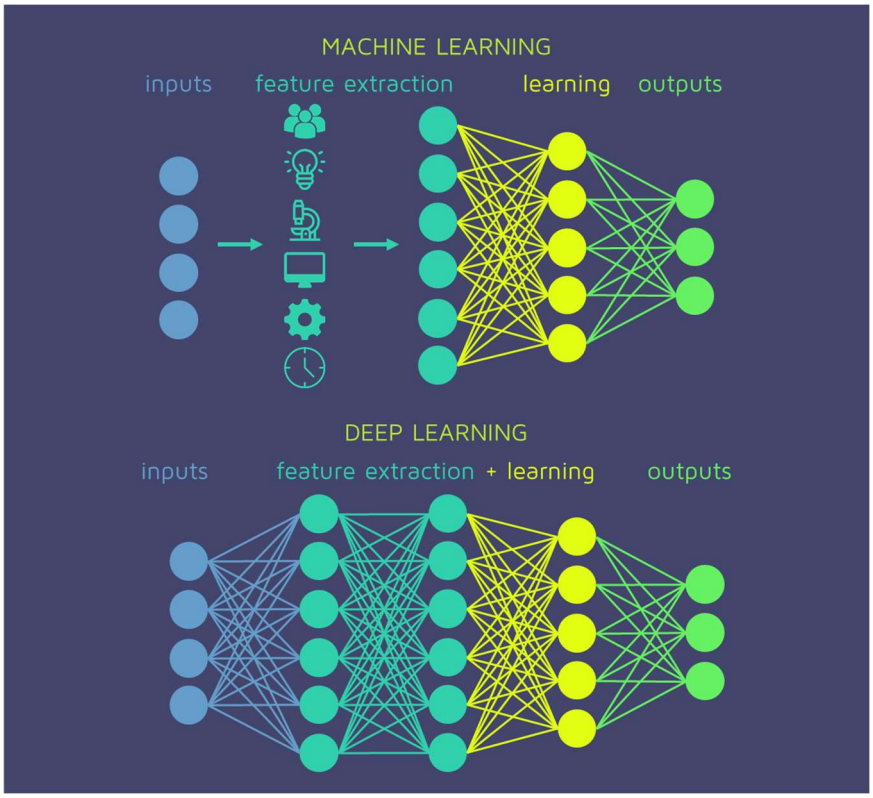


Figure 3 Difference between Machine Learning and Deep Learning ('What Is the Difference between Deep Learning and Machine Learning?', 2019)

2.2 Definition of an artificial neural network

An artificial neural network (ANN) is an engineering approach to a normal neural network. In this case, the artificial network is receiving some data that is going to be processed by the cells or neurons through mathematical operations. Then a training process will follow and, eventually, an output value will be obtained, which gives the class the input data belongs to (in case it is image recognition). As illustrated in Figure 4, an image is processed by analyzing the pixels through the neurons from each layer, trying to figure out the edges that are more significant.

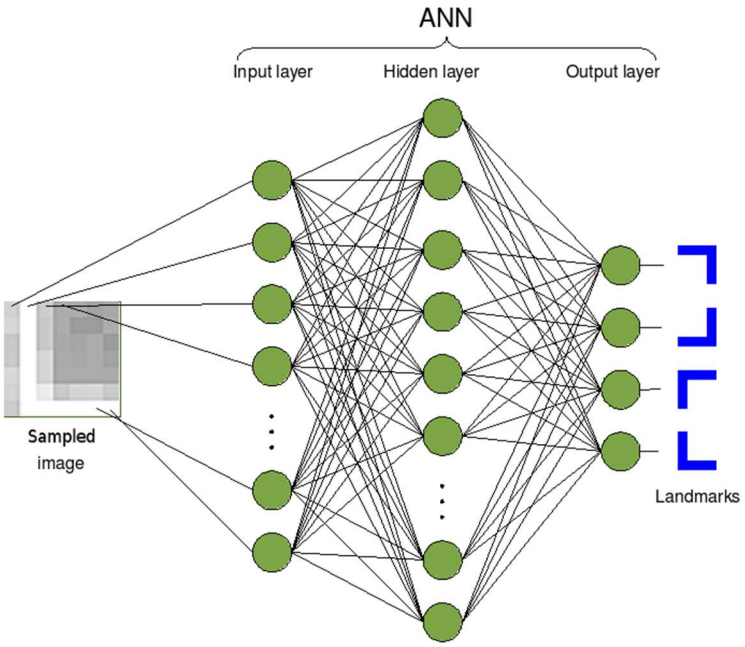


Figure 4: Example of an ANN for image recognition (Ltd, 2009).

2.3 Components of a neural network

A neural network can be described as a set of layers with which the input data is processed in order to obtain an output. Some of the layers used in the architecture of a neural network are:

- **Input layer**: layer in charge of receiving the data that will feed the network.
- **Hidden layer**: layer composed by a number of neurons responsible of doing the calculation. The output of the first layer is going to be the input for the hidden one, as all layers are connected. Each connection from a unit to another unit or neuron has its own weight which is going to be modified during the training.
- **Output layers**: this layer will often represent categories in a classification task, but it can also be a continuous value or a binary one.

These layers feature in Figure 5, where it appears one of the simplest ANN with just a hidden layer and the input and output layers.

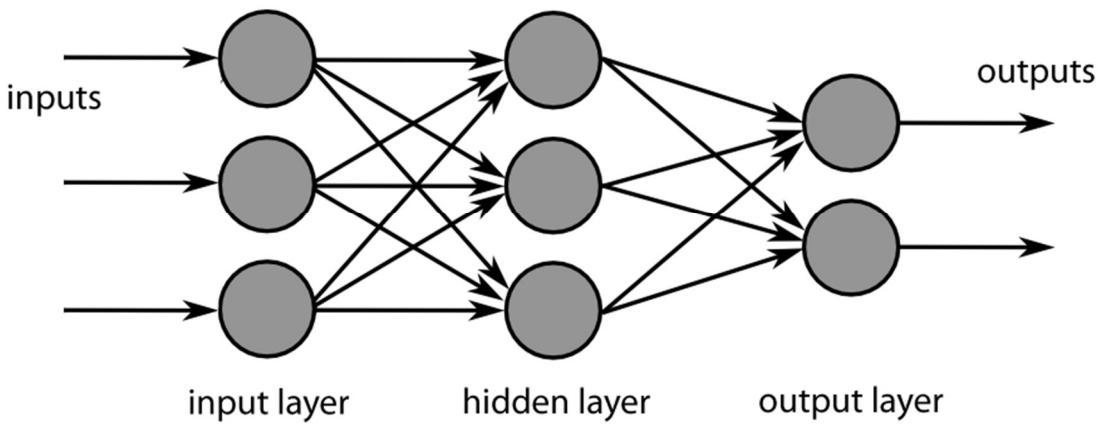


Figure 5: "Example of a neural network with a single hidden layer" (MacGregor, 2017).

2.4 How do we train a neural network?

In the previous section, I mentioned that each neuron is connected to its closest ones and this connection has its importance according to its weight, which is defined by default in the beginning of training. Due to these arbitrary values in the weights, the training will usually perform badly in the first epoch, and it can be seen as an optimization process in which the machine is adjusting these weights to get the best possible output in each iteration.

2.4.1 Activation Function

It is a function that will decide which data is going to be fired to the next neuron. In other words, it receives some input data from previous layer and stablishes some bounds, if the value obtained from earlier stages of the network is higher than certain value for example, this could be fired to the next layer, whereas the value is lower than expected or not included in the boundaries set it can be rejected or ignored.

There are many types of activation functions but some of the most important are: step function, linear function, sigmoid function and ReLU (Ph.D Kızrak, 2019). Here, I am going to talk about the ones I have used in this project, which are the sigmoid function and ReLU.

- **Sigmoid function**: the good point of the sigmoid is that it squashes the number (output) to range [0,1]. It has been quite popular since it has an easy interpretation. The main downside with this activation function is its volatility when a large positive or negative x value is inputted, which can saturate the gradient.

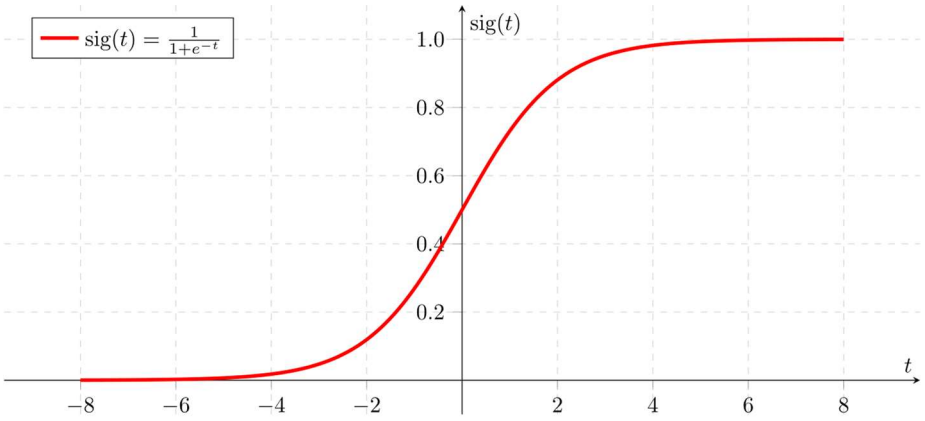


Figure 6: Example of a sigmoid function (MartinThoma, 2014).

- **ReLU**: its name comes from “Rectified Linear Unit”. It returns the value x only when x is a positive number, otherwise it gives 0. It is considered to be slightly better to the other choices due to its efficiency being computed, converges much faster than sigmoid and it does not saturate in the positive region.
  One of its disadvantages is having a non-zero centered output.

$$\mathrm{ReLU}(x) \triangleq \max(0, x)$$

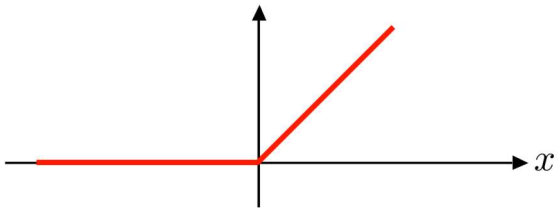


Figure 7: Example of ReLU function (Renanar2, 2018)

2.4.2 Loss Function

It is a function which represents the quantity that should be reduced during training. It shows how far from the real output the results are. The main Loss Functions are:

- Mean Squared Error: a simple metric whose procedure consists in taking the difference between the predictions and the true data, then this difference should be squared and finally the average of these squared values is obtained.
- Likelihood loss: generally used in classification problems. It is a variation of the Likelihood function adding logarithms (‘Introduction to Loss Functions’, 2018).

The formula is: $-(\boldsymbol{y} * \mathbf{log}(\boldsymbol{p}) + (\mathbf{1} - \boldsymbol{y}) * \mathbf{log}(\mathbf{1} - \boldsymbol{p}))$

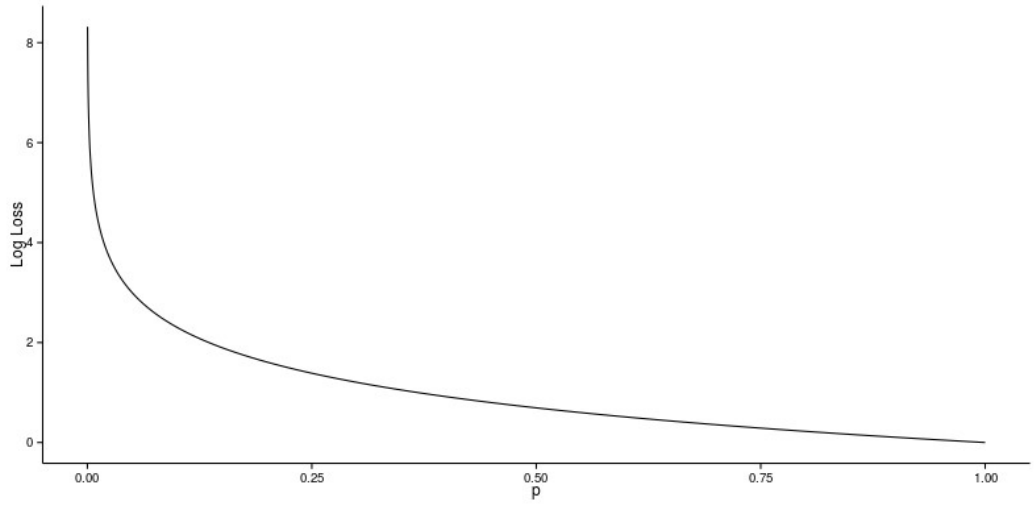


Figure 8: Log-loss function (Fortuner, B, 2016).

2.4.3 Stochastic Gradient Descent

The Stochastic Gradient Descent is an optimization algorithm used during training whose purpose is to find a set of input values which will result in a minimum value for the target function. It consists in calculating the gradient of the target function. The gradient is the "*vector of partial derivatives of specific input values with respect to a target function*" (Brownlee, 2021). Figure 11 shows a visual comparison between Stochastic Gradient Descent, and a normal gradient descent, which, based on a convex function, it tunes its parameters repeatedly to minimize a given function to its local minimum (Donges, 2021).

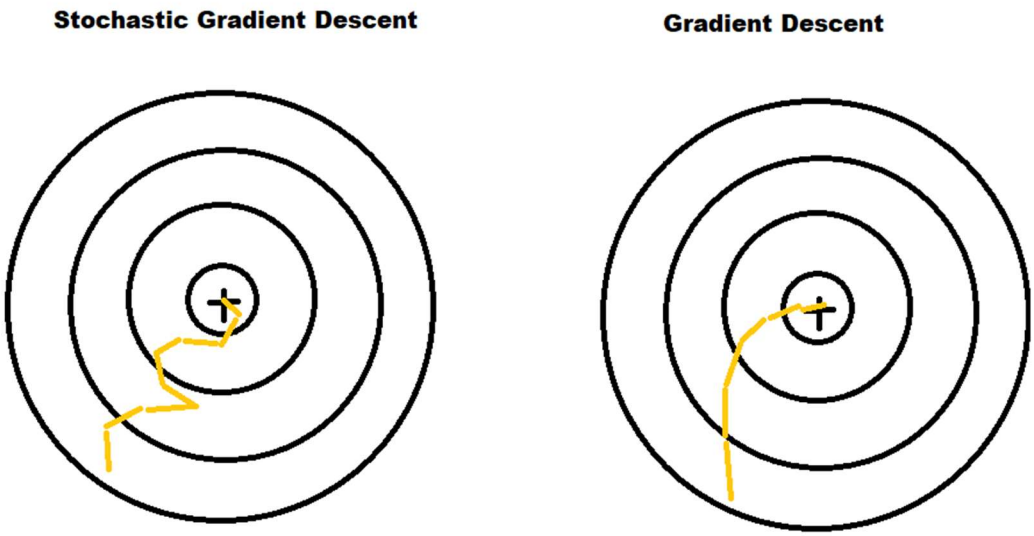


Figure 9: Difference between Stochastic Gradient Descent and normal Gradient Descent. Source: Own elaboration

2.4.4 Backpropagation

It is an algorithm for computing stochastic gradient descent. "*It determines how a single training example would like to nudges the weights and bias in terms of what relative proportions to these changes cost the most rapid decrease to the cost*" (World Circuit Records, 2017).

2.4.5 Overfitting

It is the effect of an excessive training of a certain model. If a model is trained more than it should or if the data is not homogeneous, it can lead to overfitting. The model obtained very high results at classifying data belonging to the dataset used during training, but it performs poorly when analyzing data that has not been trained before.

Despite being a problem that can occur in classification as well, it is quite common to be a challenge to face in image segmentation due to the class imbalance that exists between the narrow lines that represent a mask and the whole image which may not contain any mask inside. The under-representations of masks can cause a poor generalization that the machine could learn which would lead to losses in mask predicting (Li et al., 2021).

2.4.6 Transfer Learning

It is a training technique based on reusing information such as the weights implemented for one neural network and then being applied to other ANN to optimize the training process. Obviously, the knowledge learnt from the first task is going to be more useful if the data or problems to solve are related in both tasks (Brownlee, 2017). Transfer learning uses a model that has already been used for a specific assignment and then introduces small adjustments to make it work better for the new task. In my case, transfer learning has been used in the weights initialization process as mentioned before, as the models implement the same base architecture (in some cases it is the same network performing twice or three times). Here, at Figure 10 there can be seen an example of transfer learning applied on a dataset called Grouper Sound Data where some labels have been imported.

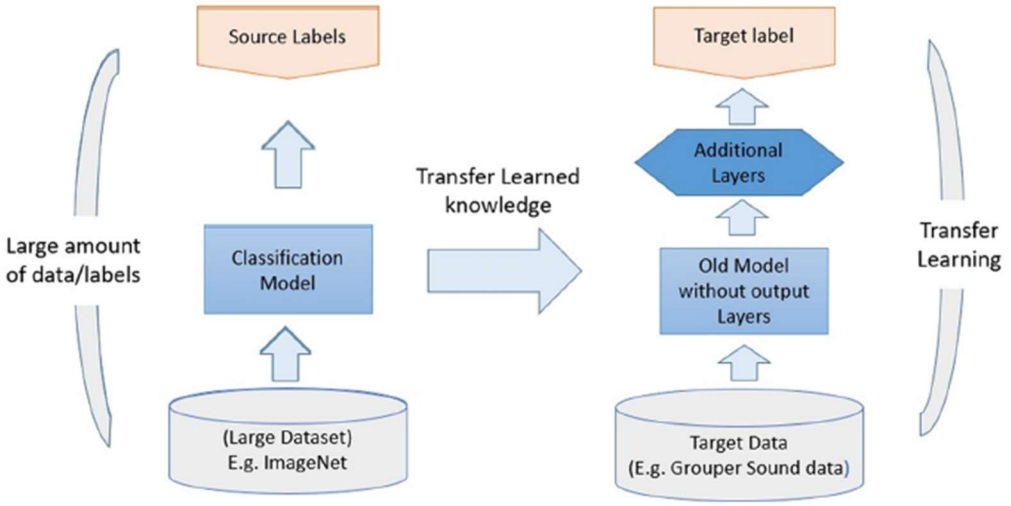


Figure 10 Schema of Transfer Learning being applied in a classification model (Ibrahim, 2020).

2.5 Convolutional Neural Networks for Image Classification

Image classification is considered as one of the cores and simplest tasks in computer vision (compared to segmentation or object detection). Image classification will input an image and output a single categorical label predicting the class.

In my case study, the computer will have to be able differentiate between pictures containing roads and pictures not containing a road element. For a human being, this might seem at first sight to be an easy task but obviously computers do not see the way we do. In the process, there may appear some challenges for the machine to recognize the labels. Some of these challenges to face are viewpoint variation (the object in the image do not necessarily have to be in the same position in every picture), scale variation (for example there can be highways occupying the whole image and tiny paths), deformations and occlusions (as an example, trees could hide part of the road).

To perform image classification the best as possible, it is recommended to follow a procedure:

Firstly, inputting the images with the correspondent distinction of labels. In this case, I am using the label “1-road” where there are roads present in the imagen, and the “2-no_roads” label for those images which do not contain any roads.

Secondly, the computer processes the input through the neural network for image classification and learn to differentiate between the two classes (in my binary classification problem, there can be multi-class problems but these ones do not fall within my area of study).

Finally, there is an evaluation process in which computer predicts the label of images that has not seen before. Comparing the results from these predictions to the labels already defined for this set I can evaluate the goodness of the model.

Convolutional Neural Networks are a type of artificial neural network used mostly for image classification. The main difference between a normal perceptron multilayer neural network (the one shown in chapter **2.3**) and a CNN is that instead of hidden layers, CNN have convolutional layers. One of the advantages of convolutional neural networks are that significantly reduces the number of parameters used in the nets. Another important advantage in CNN is that the layers are able to detect patterns regardless of the position in the given images.

2.5.1 Convolutional Layer

To understand it better, I am going to explain what a convolution is. So, for example, imagine that we have an input image of 32x32x3 dimensions. 32x32 would be height and weight in pixels whether 3 would be the depth, being 3 channels of colour (red, green and blue). Then we have the kernel, which is going to be another matrix of pixels with same depth as the input image. This kernel will do the dot product operation to every pixel in the input image and its result is going to be the weight. The output matrix resulting from the dot product is the activation map or filter. Figure 11 tries to clarify how this convolution works visually.

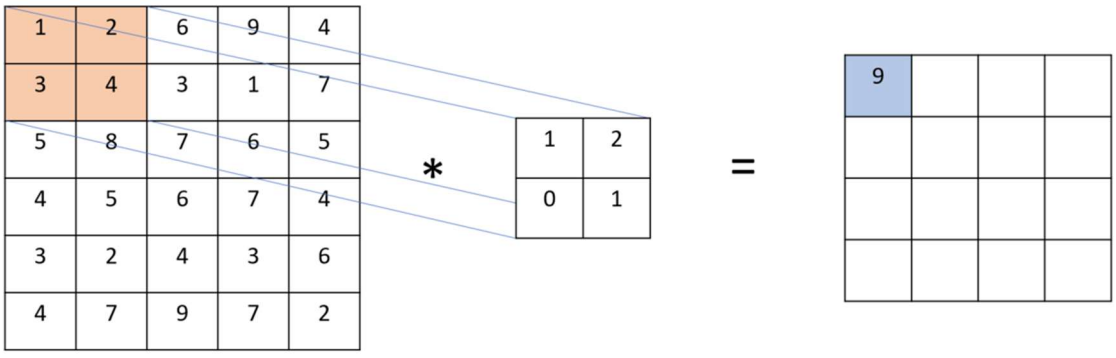


Figure 11: Example of how a kernel-convolution works. At the left there is an input image and the right matrix correspond to the output image. The (2x2) matrix in the middle is the kernel (Alzoubi36, 2021).

These set of convolutional layers stacked in a CNN enables the net to learn some hierarchical filters. The filters at the earlier layers usually represent low-level features such as edges or corners, whereas the latter ones are going to represent more complex

features. As illustrated in figure 12, these are different hierarchical features obtained from a convolution applied to a car image.

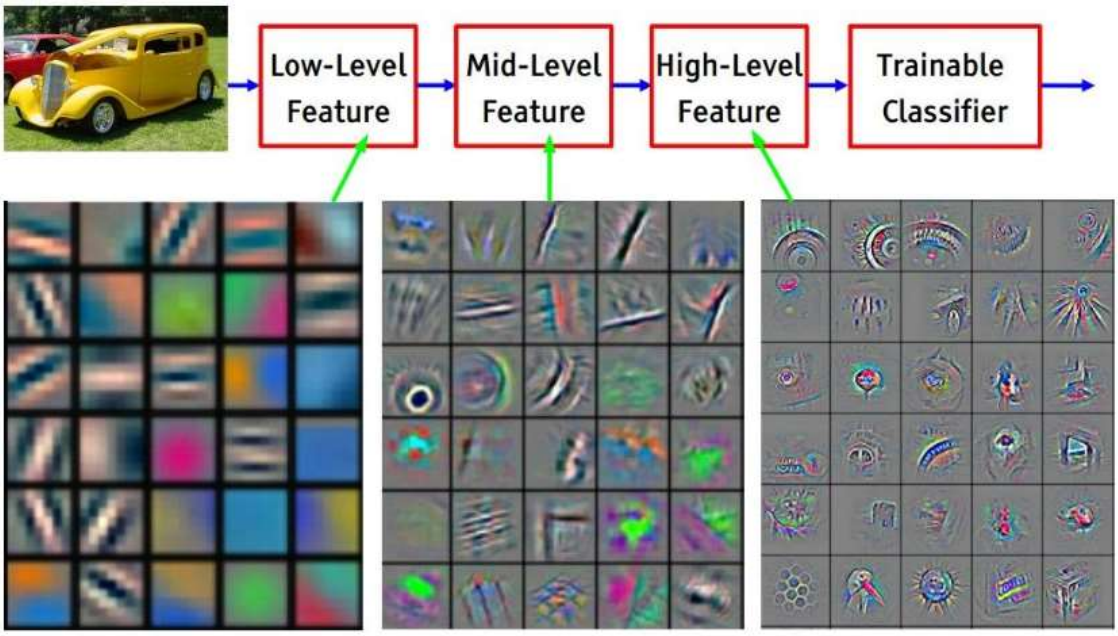


Figure 12: Different features recognized in different level layers (Gandhi, 2018).

There are two aspects as well about convolutional neural networks that are important to bear in mind. They are **local-connectivity** and **parameter-sharing**. Local connectivity defines that each value obtained in a dot product of a convolution layer is no longer linked to the input image, but a specific region of the image. Secondly, parameter-sharing states that every number from a feature map which is a result of a dot-product operation share the same weights. Usually, in a CNN, convolution layers will be followed by pooling layers, flatten layers and fully-connected layers at the end.

2.5.2 Pooling layer

Its main function is to down-sample the input volume received after a convolution operation to make the information more manageable. Another advantage of this spatial reduction is that the number of parameters is decreased as well and therefore the computation is more efficient. Despite existing two types of pooling layers, by far the most common one is the max pooling layer, which as the name says, it selects the maximum from the matrix of numbers of a feature map. The other kind of pooling layer is average pooling. An example of how pooling layer works is illustrated at Figure 13.

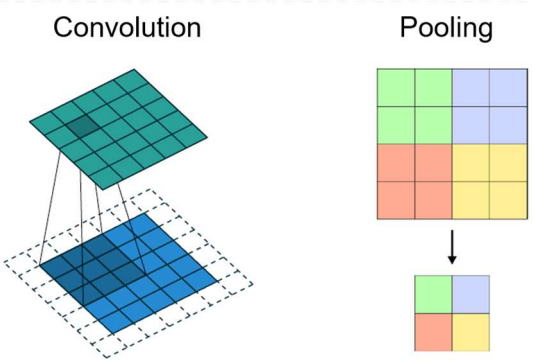


Figure 13: Example of how pooling layer works (Maier, 2019).

2.5.3 Flatten Layer

The Flatten layer will be in charge of converting the tensor into a 1-dimensional vector which will feed the next layer as a new input. *"We flatten the output of the convolutional layers to create a single long feature vector"* (Jeong, 2019). Figure 14 shows how a pooled feature map is turned into a 1-dimensional vector by applying the flattening.

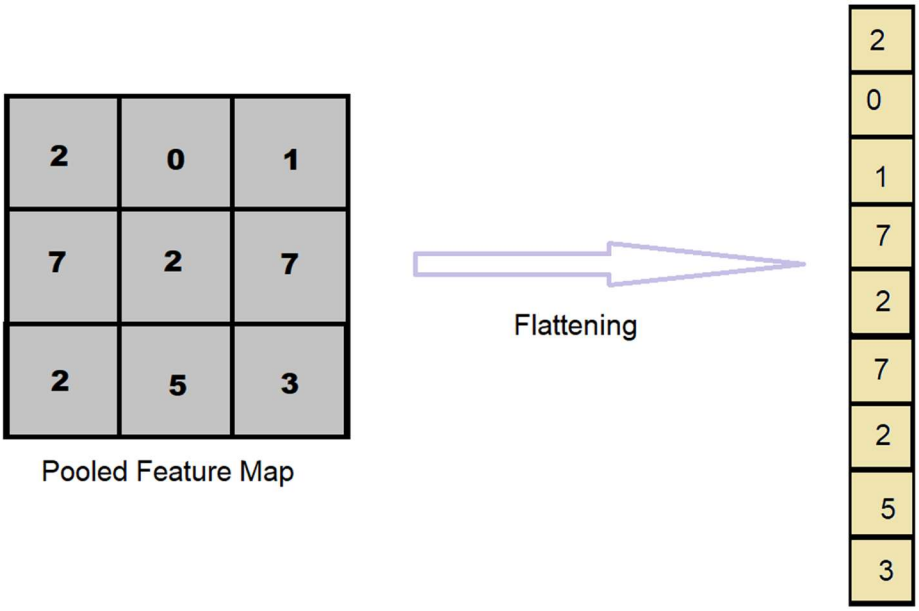


Figure 14: Example of how flatten layers work. Source: Own Elaboration.

The flatten layer always comes after a convolutional and pooling layer, so that, after all of these operations, the image is processed as a long array of numbers, and by this way, it is easier to be inputted into the artificial neural network (Kongsilp, 2019).

2.5.4 Fully-Connected layer

This is the final layer from a CNN, it is made up of "neurons" which are going to be connected to the whole input, giving the score outputs that results from ordinary neural

networks. Figure 15 illustrates the whole architecture of a CNN with the different layers seen before.

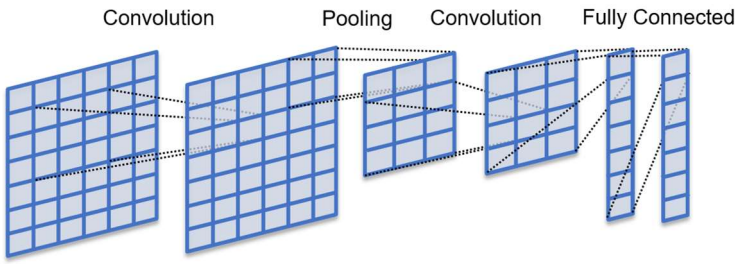


Figure 15 Example of a CNN architecture. The fully connected layer at the top right is linked to the output score's vector (Maier, 2019).

2.5.5 Main CNN architectures used in classification tasks

This section is based on (*CS231n Convolutional Neural Networks for Visual Recognition*, n.d.).

2.5.5.1 AlexNet.

AlexNet (Krizhevsky et al., 2017) was the first CNN architecture for computer vision that was popularized. It was introduced in 2012 when Alex Krizhevsky and collaborators Ilya Sutskever and Geoffrey Hinton competed in the ImageNet Large Scale Visual Recognition Challenge winning it by a margin of more than 10% difference in accuracy over the follower (top-5 error of 16%). They implemented an 8-layer architecture consisting in five convolutional layers (some followed by max-pooling layers) and then three fully-connected layers. Regarding to the activation function used in this network, ReLU was the choice (Gao, 2017).

2.5.5.2 GoogLeNet

GoogLeNet (Szegedy et al., 2015) won in 2014 the ILSVRC. It was developed by Google and they introduced the novelty of incorporating an Inception module which is able to reduce drastically the number of parameters in the architecture. The network features 4 million parameters compared to the 60 million of AlexNet. Another aspect to take into account is the use of average pooling at the end instead of fully-connected layers and a 1x1 convolution in the middle of the network (Tsang, 2018b).

2.5.5.3 VGGNet

VGGNet (Simonyan & Zisserman, 2015a) is a CNN architecture proposed by Karen Simonyan and Andrew Zisserman from the Visual Geometry Group of Oxford University. It was introduced in the 2014 edition of the Image Large Scale Visual Recognition Challenge. One of its main attributions was to highlight the depth of a network as fundamental to get good training results. Their final architecture was constituted by 16

convolutional and fully connected layers with 3x3 convolutions and 2x2 pooling layers. The main drawback of this architecture is the large number of parameters, which is around 140 million (Tsang, 2018a).

2.5.5.4 ResNet

This architecture was proposed by Kaiming He et al. It won the ILSVRC in 2015 and it introduces the possibility of skipping layers with "skip-connections". This idea is used in order to avoid the vanishing gradients problem of other architectures and to use fewer layers if needed as sometimes, with more classes, the network saturates, and the error is higher. Some others aspects to consider are the use of batch normalization after each convolution and before the activation functions, and the fact that ResNet does not use dropout (which is a technique used for avoiding overfitting by ignoring random neurons during training) (He et al., 2015).

2.5.6 Model evaluation

To test the model, there can be obtained:

- True positives: tiles containing roads that are correctly classified.
- True negatives: tiles with no-roads that are correctly differentiated.
- False positives: tiles incorrectly tagged by the model with the "Road" label.
- False negatives: tiles incorrectly tagged with the "No roads" category.

Obviously, the definitions given are centered in my study, but positives or negatives can be used for any other classification problems such as images containing cats or dogs.

After calculating these four metrics, a matrix can be created, whose name is "confusion matrix", as seen in Figure 16.

True Positive	False Positive
False negative	True Negative

Figure 16: The structure of a confusion matrix for binary classification. Source: Own Elaboration

From this matrix, it is easy to obtain the values of accuracy, precision, recall and the F1-score and AUC-ROC score. All of these are the performance's metrics.

2.5.1.1 Accuracy

It is the number of correct predictions per total entries or inputs.

$$A = \frac{TP + TN}{TP + TN + FP + FN}$$

2.5.1.2 Precision

Number of correct true positives divided by the total of positive values predicted by the model.

$$P = \frac{TP}{TP + FP}$$

2.5.1.3 Recall

It calculates the number of positive values correctly predicted by the classifier divided by the total number of inputs that should have been considered as positive.

$$R = \frac{TP}{TP + FN}$$

2.5.1.4 F1-Score

It indicates the balance between precision and recall.

$$F1 = \frac{2}{\frac{1}{P} + \frac{1}{R}}$$

2.5.1.5 AUC-ROC

AUC-ROC, or Area Under Curve, is a quite popular metric for model's quality calculation.

The AUC-ROC score is always lower than 1 and higher than 0 because it indicates the probability of classifying a random entry as positive higher rather than classifying it as negative.

2.6 Neural Networks for image segmentation

A step-forward in the field of computer vision is image segmentation. This task is a deeper approach to **image classification** and **object detection** as now the result is a map of pixels labelled with a category. *"The goal of image segmentation is to input an image and then output a decision of a category for every pixel in that image"* (Stanford University School of Engineering, 2017). The image segmentation architectures require a new approach focused on the pixels of the image, as CNN architectures process the

input at image level. The first solution in moving from classification to segmentation were the **Fully Convolutional Networks** (Long et al., 2014).

Fully convolutional Networks are able to predict labels without considering the size of the image by using convolutional and pooling layers (Tsang, 2018c). In contrast to image classification architectures, where the input image is down sampled through successive convolution layers, and then a fully connected layer outputs a label, the semantic segmentation architectures now feature an encoder and a decoder (Syed & Morris, 2019).

The encoder is the block in charge of down sampling the image. The encoder generally features convolutional layers, non-linear activations and pooling layers but fully connected layers are not present because the smallest-size spatial information of the pixel needs to be passed on to the decoder through a bottleneck. (Le, 2021).

In the decoder, it will be used up sampling layers which are in charge of increasing the size of the image again and enable this part of the architecture creates the segmentation map. The up sampling process is done thanks to transposed convolutions (Gupta, 2019).

Figure 17 illustrates the whole segmentation process, starting from an input image of a tile containing roads, then a segmentation network architecture using the encoder and decoder block with their specific layers, and finally the result obtained (the mask from semantic segmentation).

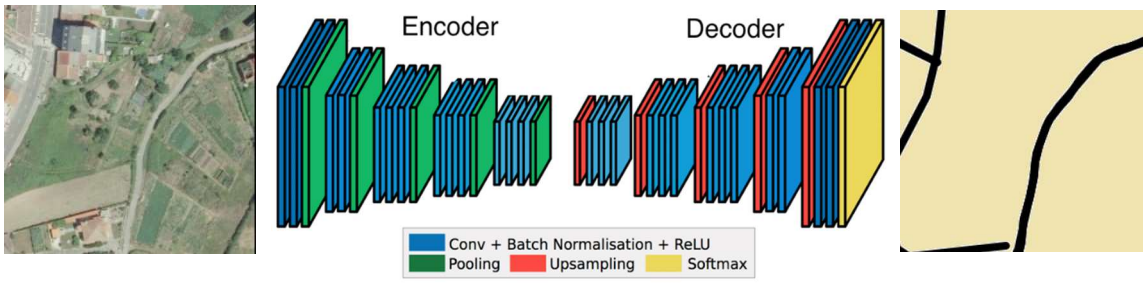


Figure 17 Encoder-Decoder architecture used for image segmentation. The image in the left is the input and the one in the right is the segmentation result, featuring the roads in black (Syed & Morris, 2019).

2.6.1 Main architectures used for image segmentation

2.6.1.1 U-Net

This architecture proposed by Olaf Ronneberger, Thomas Brox and Phillip Fischer in 2015 in the paper U-Net: Convolutional Networks for Biomedical Image Segmentation (Ronneberger et al., 2015) arises from the Fully Convolutional from the previous year.

U-Net is a NN which follows the encoder and decoder design. The path on the left of the architecture is called “contraction path”, also known as the encoder. It will down-sample the image by applying convolutions and max pooling operators. By doing this, the network can obtain accurate information about small features at different stages of the image. On the right, there is the expansion path or decoder. “The goal is to semantically project the discriminative features (lower resolution) learnt by the encoder onto the pixel space (higher resolution) to get a dense classification. The decoder consists of **upsampling** and **concatenation** followed by regular convolution operations” (*How U-Net Works?* | *ArcGIS for Developers*, n.d.).

This contraction and expansion in the same architecture gives the net the shape of an ‘U’, which is the reason for it being named U-Net. As mentioned in Fully-Convolutional Networks (**2.6**), this architecture is not going to use any fully connected layer and the procedure to follow in order to predict pixels in the edges of the image is to extrapolate the values by mirroring the input image. Figure 18 illustrates the U-Net architecture.

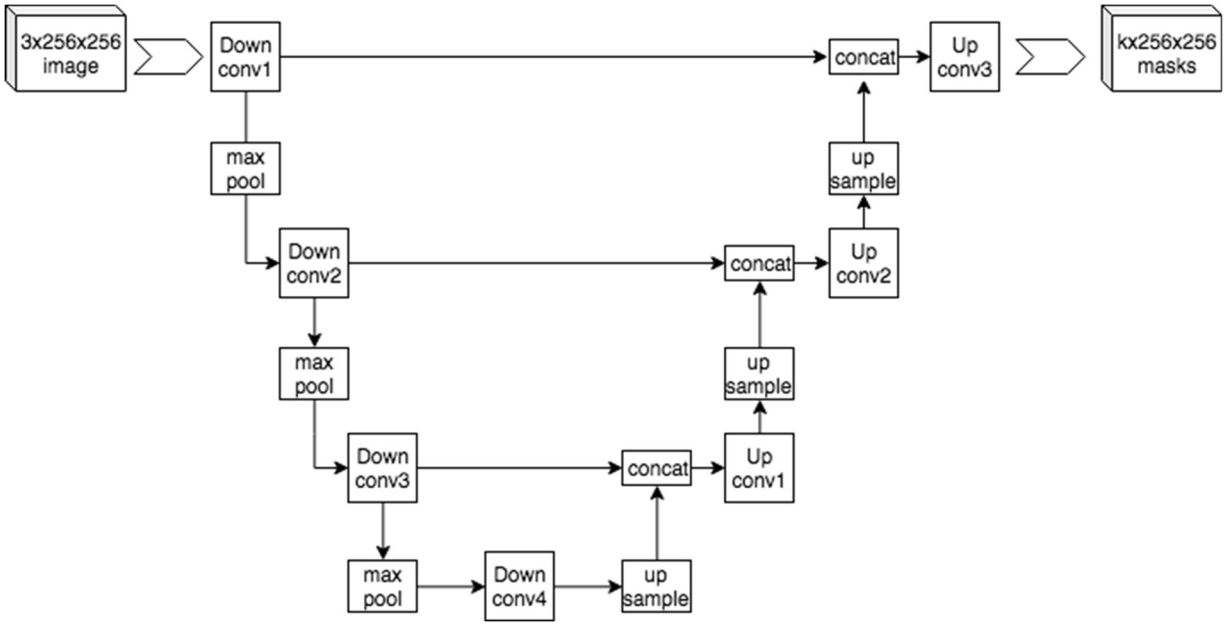


Figure 18 The U-Net architecture (Yazdani, 2019).

2.6.1.2 Linknet

LinkNet is an architecture proposed by Abhishek Chaurasia and Eugenio Culurciello in 2017 for image segmentation. This net allows it to learn without an excessive increase in the number of parameters (Lukas Getautis, 2021). In their paper "*LinkNet: Exploiting Encoder Representations for Efficient Semantic Segmentation*" (Chaurasia & Culurciello, 2017) they affirm that this architecture only needs 11.5 million parameters for processing an image resolution of 3x640x360".

LinkNet potentially works better in segmenting live stream videos with almost real-time applications. Their goal is to get an accurate instance level prediction not delaying too much the time for processing. Usually, a lot of spatial information is lost in the encoder path due to the use of stride convolutions or pooling operators. To fix this, LinkNet proposes to transmit directly the spatial information from encoder to decoder. By doing this, there are no extra parameters that would result from the pooling or convolution operators and no information is lost. Figures 19 and 20 shows the LinkNet architecture with both blocks (19), and the encoder and decoder specifically (20).

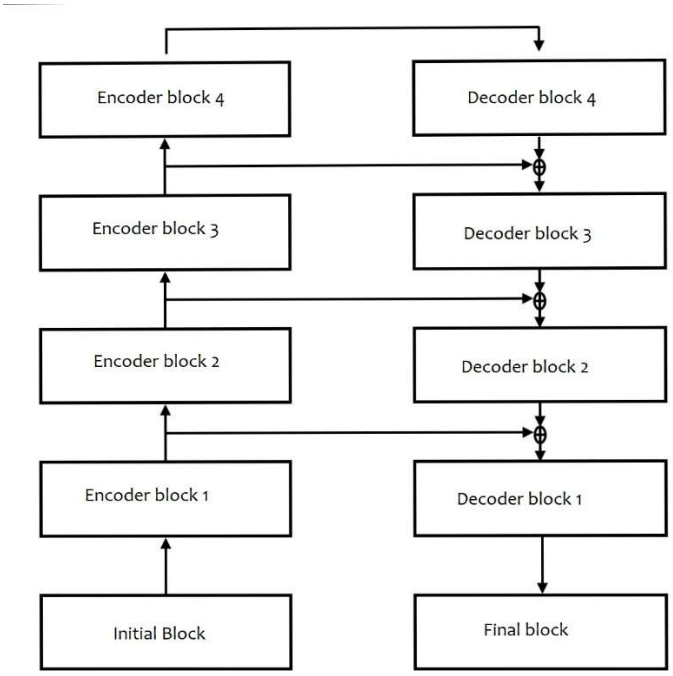


Figure 19 LinkNet architecture with the encoder transmiting information to the decoder path.

Source: Own Elaboration

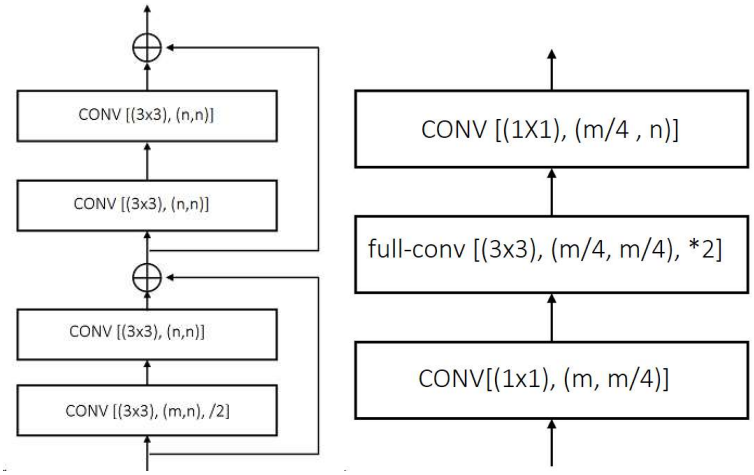


Figure 20 (a) Encoder and (b) Decoder Blocks present in the Linknet architecture.

Source: Own Elaboration.

2.6.2 Model Evaluation: IoU score

IoU (Intersection over Union) is a metric used in image segmentation which calculates how much part of the mask targeted overlaps with the mask predicted. It basically counts the number of pixels that both masks have in common (intersection) and divides into the sum (union) of both (Rosebrock, 2016). Figure 21 pretends to clarify how IoU works.

$$IoU\ score = \frac{targetMask \cap predictedMask}{targetMask \cup predictedMask}$$

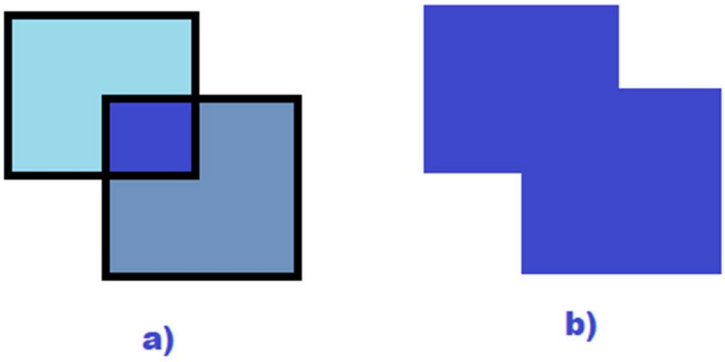


Figure 21. In example (a) we have 2 squares representing both masks and both share a small square which is the mask that they have in common. In the example (b), there is the union from both masks (target and predicted). Source: Own Elaboration.

2.6.3 Callbacks

A callback is a set of functions to be applied at given stages of the training procedure. Callbacks can be used to get a view on internal states and statistics of the model during training, to periodically save a model to disk, or to do early stopping in case the model has reached certain levels of accuracy to avoid overfitting (Team, n.d.-a). As it will be shown in chapter **4.4.2**, this functionality has proven to be quite useful during the realization of my project, as there were models who accidentally got interrupted and thanks to the model checkpoint function it was saved. Later in **4.4.2** I will talk as well about Reduce on Plateau and Early stopping, which focuses on reducing learning rate or stopping the training at an earlier stage.

3. MATERIAL

3.1 DATA

The data which is going to be used in this project can be divided in two groups considering two different approaches: segmentation and recognition. These images belong to the MTN50. This Mapa Topográfico Nacional can be downloaded from the IGN web. There are two scales (1:25,000 and 1:50,000), but I am focusing on the tiles from the 50,000 scale. Among the 1073 set of tiles that form the MTN50, it has been selected 16 to be the focus of the study. The following figure pretends to give the reader an idea of the location of each set of tiles. It was selected like this to get a wide range of areas in terms of vegetation, types of roads and situations (like occlusions, or poor layouts) that could make the study difficult.

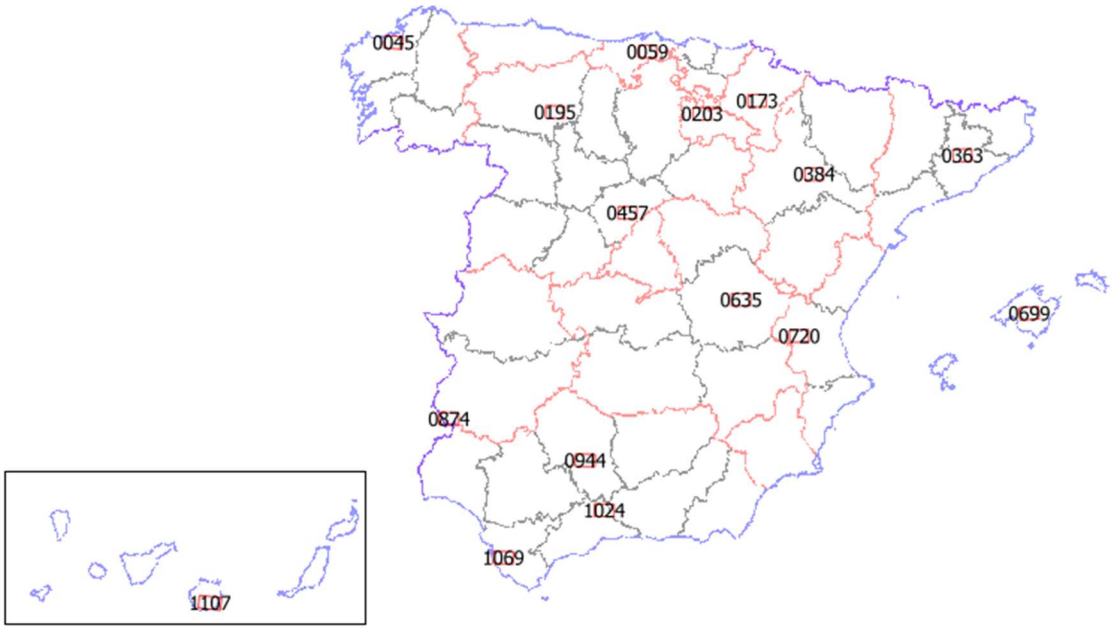


Figure 22: This figure shows the localization from each set of tiles in the Spain territory.

3.1.1 Image recognition

In this project, image recognition can be thought as a binary classification problem in which it has to be differentiated between images containing roads or paths, and images without roads. In order to perform this task correctly, it needs to be labelled before training, as it is a supervised learning technique.

The labelling of these pictures has been done thanks to a Tile Labeler called Cartobot (http://teselasmp.gisai.geoide.upm.es/home), which originally was thought to label solar panels and images containing wind turbines.

3.1.1.1 Images containing roads

The first figure (23) shows the six tiles chosen to carry out the study of semantic segmentation and image recognition in terms of radiometry as well as the quantitative analysis. These images of 256x256 were my first choice because the differences in the areas surrounding the roads or paths could affect the results. For example, in the coastal image, the layout can barely be seen, whereas the urban roads seem to be quite clear. At first sight, I would say that the Mediterranean and rural tiles are going to throw good results as well, but dry areas and green spaces could be difficult to the models as the color of the background and path might not differ enough.

Figure 23: Example of tiles of 256x256 pixels tagged with the "Road" label. Source: Own Elaboration.

The next figure, which is the figure 24, illustrates the 512x512 tiles selected. As it can be seen, the areas are the same as the ones in the previous tile size, so that it could be a more accurate analysis. The difference is that the 512x512 tiles contain 4 images of 256x256. For example, it is easy to see that the previous rural area is placed in the top right of the new rural area.

Figure 24: Example of tiles of 512x512 pixels tagged with the "Road" label. Source: Own Elaboration.

Finally, figure 25 shows the biggest tile size I am studying, which is 1024x1024. In this case, at least at first sight, one could think that some of these tiles are not going to be correctly classified by the networks as, for example in the case of Mediterranean or Green Spaces, the roads only cover a small portion of the image. On the contrary, urban and dry areas seem to be easier to identify because of the amount of roads present in the tile.

Figure 25: Example of tiles of 1024x1024 pixels tagged with the "Road" label. Source: Own Elaboration.

1.1.1.2 Images not containing roads

The criterial followed to choose the tiles not containing roads was to select the closest images to the selected before. Following this procedure, I make sure the areas are as similar as possible. Figure 26 shows the first set of tiles, the 256x256. It is interesting to see if the models are able to classify as “No-Roads” the urban ones or the rural picture, as they might not differ enough from the previous images. By contrast, the rest of examples seem to be quite clear in terms of not roads featuring on them.

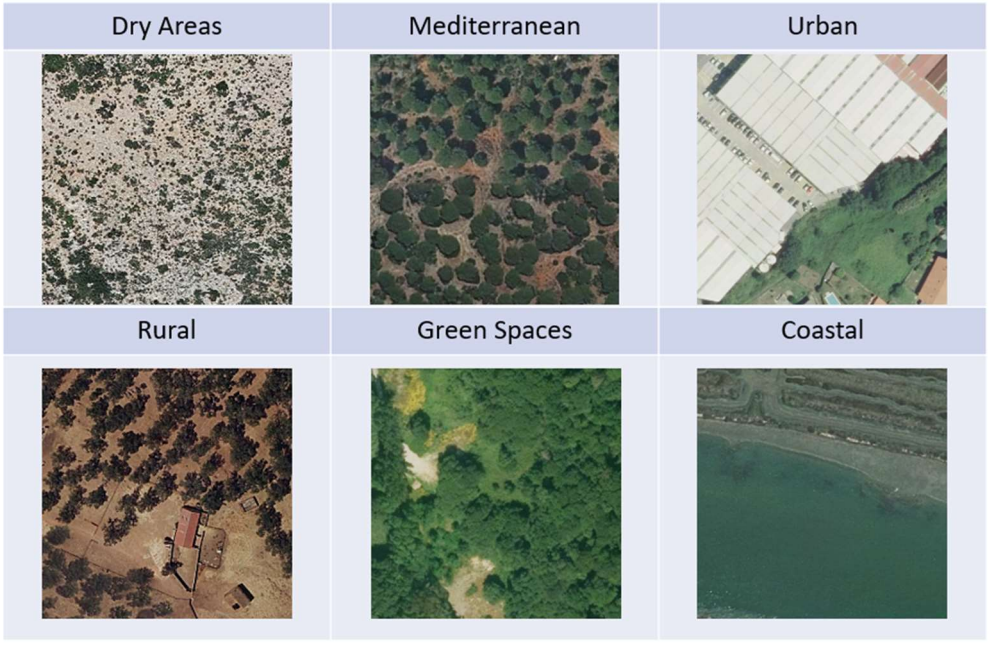


Figure 26: Example of tiles of 256x256 pixels tagged with the “No-Road” label. Source: Own Elaboration.

Figure 27 shows the following tile size (512x512). In this project, as the tile size grows, there are less “no-roads”. This is due to the image considered as the label “road” despite having a little presence of roads. An optimal way to do this, would have been to classify as roads only when the paths reached a certain percentage of the image.

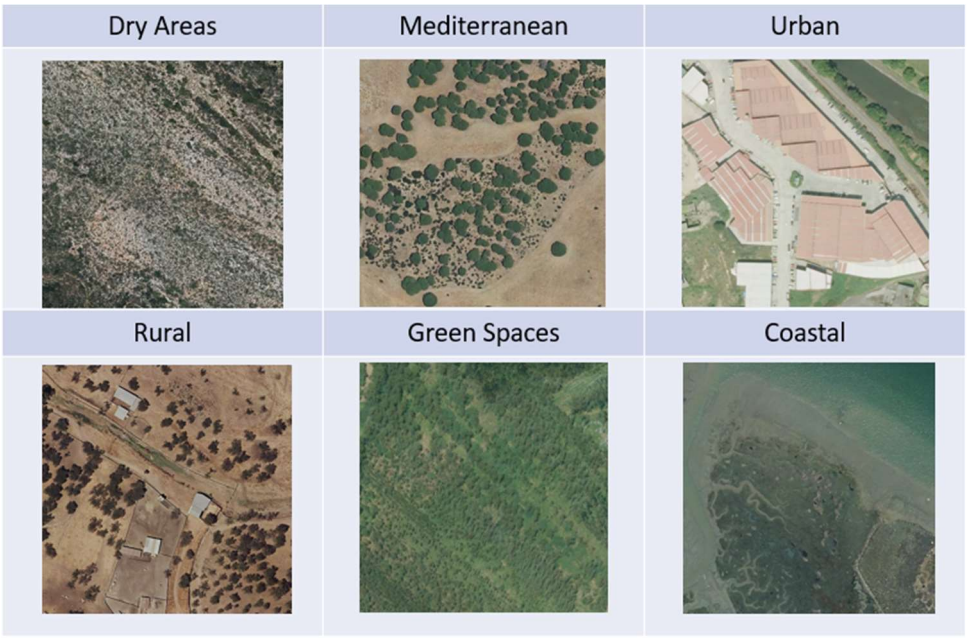


Figure 27: Example of tiles of 512x512 pixels tagged with the "No-Road" label. Source: Own Elaboration.

Figure 28 illustrates the last tile size of study. As mentioned before, the problematic of the no-roads images being that low in these bigger sizes lead to difficult choices in the Urban and Rural areas, as it is rare to have these places with no roads surrounding them.

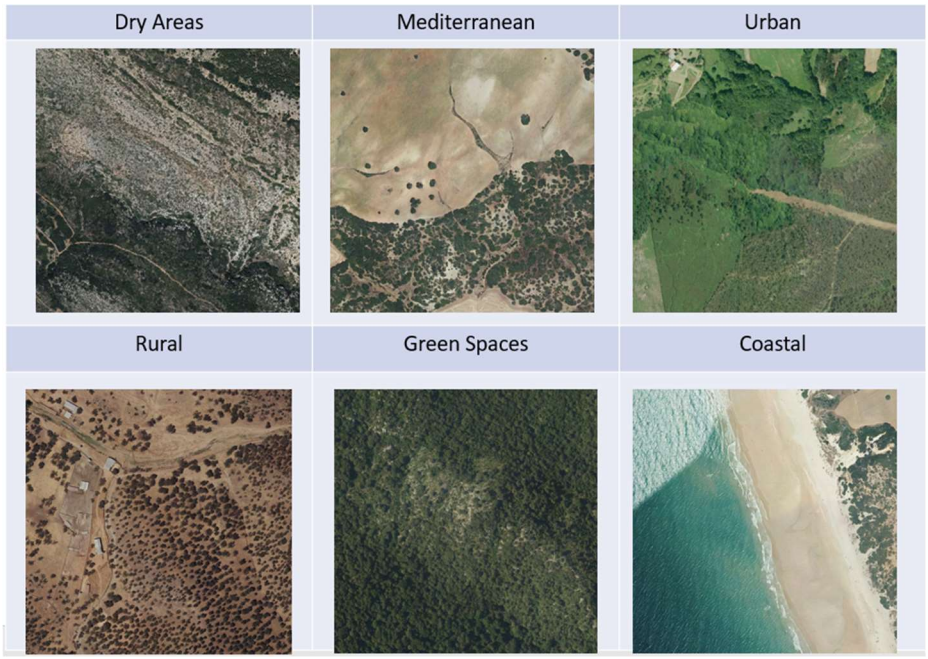


Figure 28: Example of tiles of 1024x1024 pixels tagged with the "No-Road" label.

Source: Own Elaboration.

3.1.2 Image Segmentation

For image segmentation, I have 3 groups of images depending on the tile size.

Each group has 2 sets of images, the ortho-images and its masks. These masks, like the one illustrated in figure 29, have been digitalized before training. After the semantic segmentation it is going to be obtained a mask predicted, which will be compared with the one which has been used for training. The result of this comparison will tell us how good the results have been.

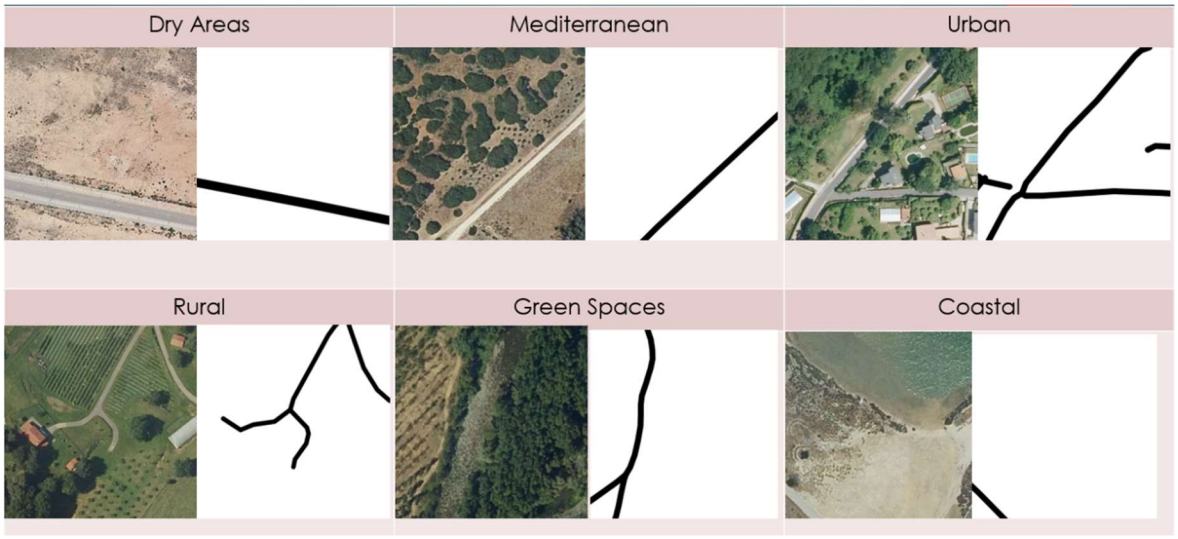


Figure 29: Example of tiles of 256x256 pixels tagged with the "Road" label and its masks.

Source: Own Elaboration.

In figure 30, it can be seen some interesting obstacles that the model can face during the process of predicting the mask's geometry, as it could be the occlusions from the Green Spaces areas. On the contrary, in urban, dry areas and Mediterranean, the model should not have much problems. In the case of the coastal image, the network should be able to differentiate between the real road and the sand which seems to be a path.

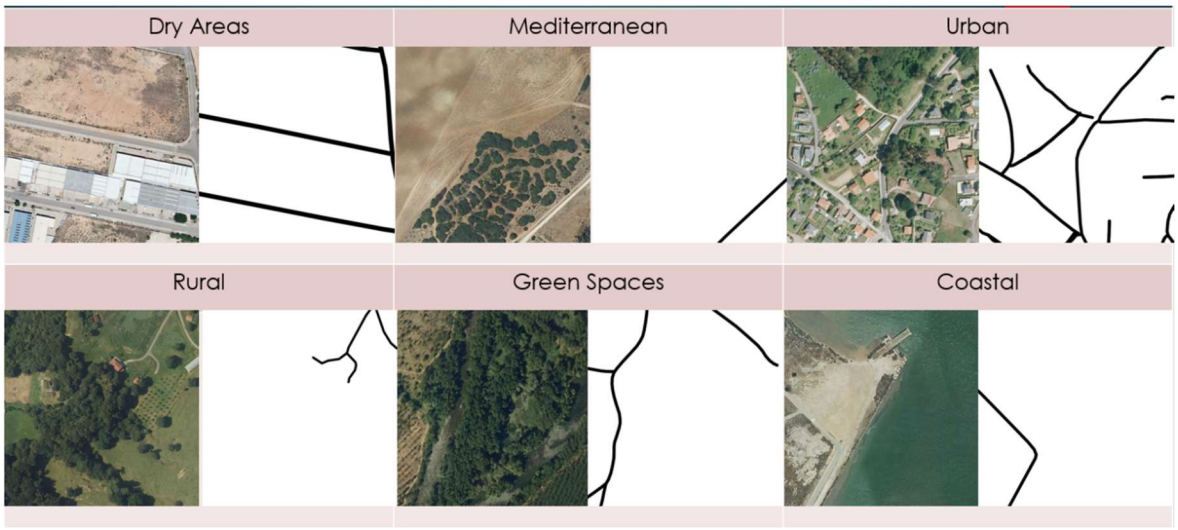


Figure 30: Example of tiles of 512x512 pixels tagged with the "Road" label and its masks.

Source: Own Elaboration.

Lastly, in this 1024x1024 tile size (Figure 31), the model will have to face complex road networks like in the Urban area or even the rural or dry areas. It will be intersting to see as well how semantic segmentation will work in the green spaces which such a quantity of hidden layouts.

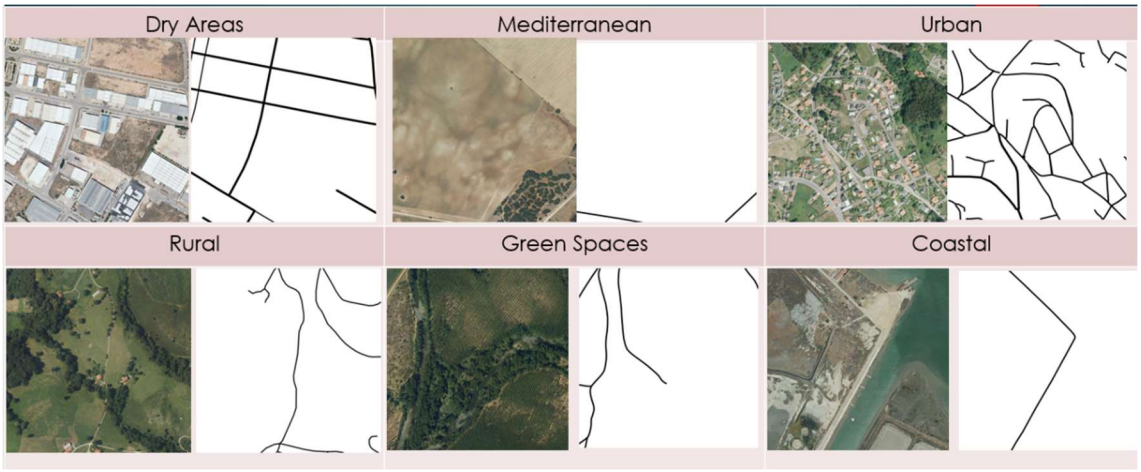


Figure 31: Example of tiles of 1024x1024 pixels tagged with the "Road" label and its masks.

Source: Own Elaboration.

3.2 HARDWARE

I am using three devices to complete this project: my current computer, two servers belonging to the Instituto Geográfico Nacional and a Solid-State Drive.

- My computer: I have a HP Pavillion Laptop 15-cs2xxx, with an Intel® Core™ Processor and a 12GB RAM card. The OS is 64 bits.
- Server (Cartobot1): this server has 62GB RAM, an Intel® Processor as well, NVIDIA Corporation graphic card and a Disk Capacity of 480GB. The OS is 64bits again.
- SSD2: This solid-state-drive has 983 GB of storage.
- Server (IGN1): This server has 125.5 GB RAM, an Intel® Xeon (R) Gold 6148 CPU @ 2.40GHz x80. The OS is 64bits. It has llvmpipe Graphic Card. IGN1, in addition, has 4 GPU, which enables me to work in 4 experiments at the same time.

3.3. SOFTWARE

3.3.1 Python

I am using the 3.8.5 version of Python in IGN1 and the 3.8.3 version in Cartobot1. Python is characterized for being an open-source high level interpreted programming language supported by a large community of users. It is a multi-paradigm programming language

and it is object-oriented. It can be used in a wide range of areas such as web applications, artificial intelligence or data science. Python is currently one of the most popular programming languages (Visus, 2020). Python's philosophy emphasizes code readability, making the language accessible to as many users as possible.

3.3.2 Anaconda

It is a distribution of the Python and R programming language which is used for data science, machine learning tasks, or predictive analysis. As well as python, Anaconda's distribution is open-source code with well-detailed documentation ('Anaconda (Python Distribution)', 2021). One of the reasons for its popularity is the large number of data science and machine learning packages that can be installed from its repository. This distribution is suitable to Windows, Linux and macOS. It is being used the 2020.07 version.

3.3.3 Jupyter-notebook

Jupyter-notebook is a server-client based application which enables the user to run or edit notebook documents via a web browser. These notebooks codes are executed thanks to the kernel (which is a "computation engine"). As soon as a notebook is opened, the kernel is launched (Ingargiola, 2015). In my case, I am running the scripts corresponding to the Image Segmentation or Image Recognition experiments. Jupyter-notebook allows the user to have the code split into cells so that it can be easier interpreted for the programmer or any other user. Another advantage of this cell-division is that it can be noticed where a part of the code is generating an error as it would be the only one stopped or the one that throws the alert. The version used for Jupyter-notebook is the 6.3.0 version in IGN1.

3.3.4 NVIDIA CUDA's libraries

CUDA is a parallel computing platform and programming model developed by NVIDIA. It is suitable for popular programming languages such as C, C++, Fortran, Python and MATLAB (*CUDA Zone*, 2017). For the realization of my project, I have used the CUDA libraries so that I could use 4-GPUs at the same time. Each GPU has been able to train a neural network. The version for both servers (IGN1 and Cartobot1) is the 11.0.

3.3.5 TensorFlow

TensorFlow is an open source developed by Google and designed for deep learning tasks such as training neural network models. *"TensorFlow accepts data in the form of multi-dimensional arrays of higher dimensions called tensors. Multi-dimensional arrays are very handy in handling large amounts of data"* (Simplilearn, 2021)*, "It has a comprehensive, flexible ecosystem of tools, libraries and community resources that lets*

researchers push the state-of-the-art in ML and developers easily build and deploy ML powered applications" (*TensorFlow*, n.d.). For both Cartobot1 and IGN1 I am using the 1.14 version.

3.3.6 Keras

Keras is an open-source Python library which recently has included TensorFlow and Theano as well. Its philosophy believes in a design prepared for human beings and not machines, so it is easy to understand for the users. Keras ranked as #1 for deep learning both among primary frameworks and among all frameworks used (Team, n.d.-b). It is used to develop and evaluate deep learning models. A model in Keras is defined as a sequence of layers. The version of Keras used in this task is the 2.2.4 for both servers.

4. METHODOLOGY

Figure 32 illustrates the methodology used in this project. I have taken into consideration that this TFG consists of 12 ECTS. Each ECTS is equal to 30h approximately. So, in total, I have used 360h in order to finish this project.

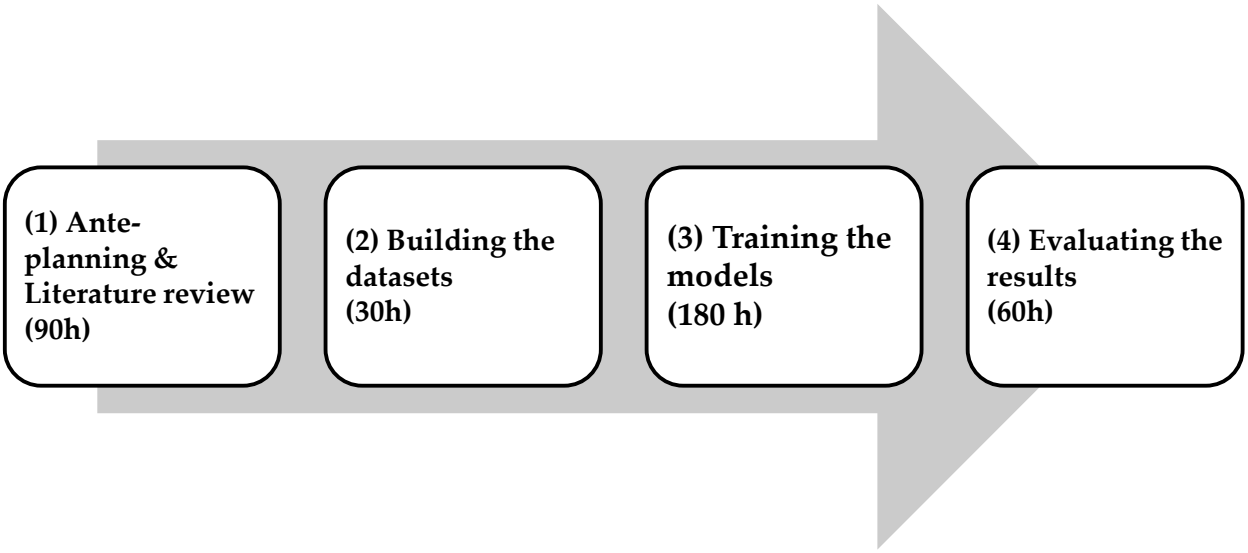


Figure 32: Summary of the methodology applied in this project.

4.1 Building the datasets

The first step to be made is to select the tiles which are going to form my dataset. This means choosing the tile size depending on the task I am performing.

Once the images are selected, I execute a script called “Create_Dataset”, which basically shuffles the dataset and divides the set of images into three groups: train, validation, and test, in a 90-5-5% proportion respectively. These proportions have been chosen because the high number of labelled images does not require the ratio of validation images to be higher, as it will contain enough representations. The shuffling is done to ensure randomness, as if I omitted this, maybe the training results would be weaker due to the model not recognizing roads in certain situations. For example, there could be a lot of images in the training from “Dry Areas” and if I tested the model with a picture from an urban area containing roads, it could not recognize the path as good as in the training set. It is important to mention that this script is not the same for image segmentation and recognition. In image segmentation, the script creates inside the train-test-validation files an input and an output file, which correspond to the image and its mask. On the other

hand, the image recognition approach of this script creates inside the training-test-validation files a couple of directories to differentiate between images containing roads and the ones who do not. Tables 1, 2 and 3 shows the tiles distribution used in the image recognition task. As mentioned in **3.1.1**, it is important to notice the decrease in the number of road images, as in the results section, it will be commentated as one of the problems.

Table 1 Tiles distribution in Recognition task images with size 256x256.

	Image Recognition with tile size: 256x256		
	Training (90%)	Test (5%)	Validation (5%)
Roads	232,822	12,935	12,935
No roads	251,226	13,957	13,958

Table 2: Tiles distribution in Recognition task images with size 512x512.

	Image Recognition with tile size: 512x512		
	Training (90%)	Test (5%)	Validation (5%)
Roads	85,713	4,762	4,762
No roads	32,291	1,794	1,794

Table 3: Tiles distribution in Recognition task images with size 1024x1024.

	Image Recognition with tile size: 1024x1024		
	Training (90%)	Test (5%)	Validation (5%)
Roads	26,245	1,458	1,459
No roads	2,969	165	165

On the other hand, tables 4,5 and 6 correspond to the tile's distribution employed in the segmentation task. It has only been taken into consideration the images containing roads, as they are the ones generating mask's geometries.

Table 4: Tiles distribution in Segmentation task images with size 256x256

	Image Segmentation with tile size: 256x256		
	Training (90%)	Test (5%)	Validation (5%)
Pairs (Tile and mask)	225,397	12,522	12,523

Table 5: Tiles distribution in Segmentation task images with size 512x512.

	Image Segmentation with tile size: 512x512		
	Training (90%)	Test (5%)	Validation (5%)
Pairs (Tile and mask)	85,713	4,762	4,762

Table 6: Tiles distribution in Segmentation task images with size 1024x1024.

	Image Segmentation with tile size: 512x512		
	Training (90%)	Test (5%)	Validation (5%)
Pairs (Tile and mask)	26,245	1,458	1,459

4.2 Image preprocessing

It is the process of applying different algorithms to a dataset, including resizing, shifting an image, modifying the radiometry and contrast of an image, etc. In my case, each tile-size set of images is quite homogeneous in terms of scale and brightness, but there are other datasets with a large variety of tile's sizes and this can potentially affect the training if it is not processed properly.

4.2.1 Image Recognition

The image preprocessing orientated to Image Recognition can easily be handled with Keras' class "Image Data Generator" (Keras Team, n.d.).

The parameters I have chosen from this class are in first place **rescaling**, which multiplies the data by the given value. In my case, I chose this value to be 1/255. The explanation for this is that it will normalize the brightness values for every pixel in the image (being since then from 0 to 1) and, due to this, it will be easier to handle this new numbers through the upcoming operations. Secondly, **rotation range**, which defines a range for the maximum random value of degree which can be applied for image rotation. In my script, I set this value to 25, so that it is not a huge rotation but it is enough to augment the set of images. In third place, **width or height shift range**, which stablishes the maximum random value that the image can be shifted vertically or horizontally. I used the float value 0.1, which means that, the biggest shift the image could experiment would be a 10% of the image size. The following parameter has been **zoom range**, which will determinate the range for a random zoom to set, which in my case is again 0.1 or 10% of the image.

Then, I decided to apply **horizontal** and **vertical flips**. This transformation allows reverting the rows (horizontal) or columns (vertical) from the image. This argument is a Boolean type, and it is set as true. In the case of recognition roads, it makes sense to apply these flips because it would mean that the image is seen from another point of view, but there are some images such as pictures containing dogs, cats or even people, in which applying a vertical flip would not make sense. Lastly, I have used **fill mode**. It defines the method used for filling the values from pixels which are not in the boundaries of an input image. In this case, the method used is Nearest Neighbors.

The class Image Data Generator also has an operation called "flow from directory" which gives the path for the validation and train set, in order to generate the images for these datasets. In order to illustrates examples of data augmentation, I have prepared a script which uses an example image to apply randomly the different transformations before. In figure 33, we can find some examples of data augmentation applied to a random tile.

Figure 33: The original image has been modified with ImageDataGenerator creating 5 examples. This image belongs to the 512-tile size dataset. Source: Own Elaboration.

4.2.2 Image Segmentation

When the dataset is not large enough, what should be done is to augment the size of it by obtaining different versions of existing images as a result of different operations (such as rotation, blur, scaling…).

In order to do these preprocessing tasks, I am using Keras' libraries: Albumentations. Figure 34 shows an example of a 256x256 tile from the segmentation task modified by image segmentation.

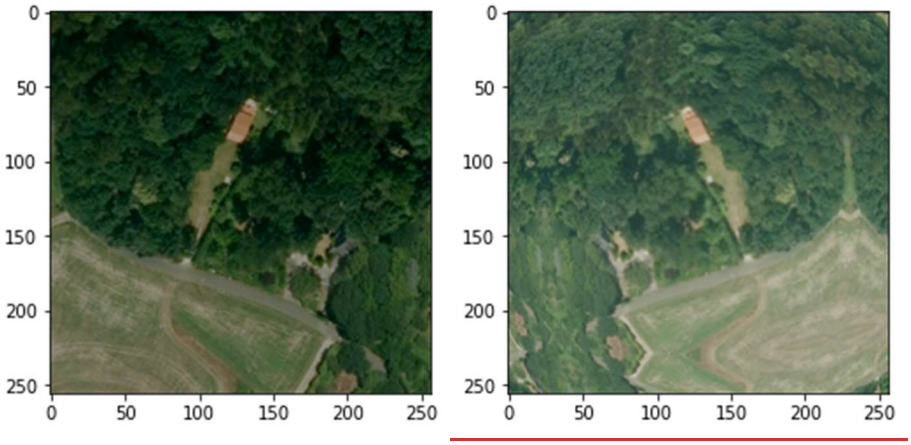


Figure 34: Examples of images generated with data-augmentation. These images belong to the 256x256 pixels size dataset. Source: Own Elaboration.

To obtain these images, it has been applied different operations. First of all, I have used **horizontal and vertical flip**: these transformations move the pixels in a certain direction by specifying a float value from 0 to 1 which will determinate how much it will be transformed. These movement of the pixels result in some pixels disappearing from the image and some new ones having to be generated. I have used 0.5 value to both horizontal and vertical flip. In second place, I have applied **random rotate**, which applies a clockwise rotation to the image from 0 to 360. The rotation applied in my images have been 90 degrees.

Thirdly, I have used a **transposition**. This transformation swaps rows and columns. The following parameter I have selected is **shift scale rotate**. It enables a rotation and scaling operation depending on a certain angle defined previously.

Below, I have used **random brightness contrast**, which, as the name suggests, it applies random changes in the brightness, contrast, and saturation of an image. (*Albumentations Documentation - Transforms*, n.d.). Following with the next parameter, I used **Random Gamma** in order to modify the intensity and brightness of an image. I set 0.25, being the possible values from 0 to 1.

Finally, I made use of the **blur** parameter, which is an image augmentation technique particularly important because a tile from the dataset do not necessarily have to be perfect taken, it might be blurred and it could make the learning task more difficult, so it is interesting to introduce blurred images in the training dataset so that the machine covers a wide range of possibilities.

4.3 Neural Network Architecture

4.3.1 Image Recognition

The VGG convolutional base is one of the most popular choices for classification task. While AlexNet preferred to use tinier window sizes and strides in the first convolutional layers, VGG focuses on deeper networks (Simonyan & Zisserman, 2015b). The receptive fields used in VGG are usually 3x3 with 1 stride. The smaller-size filters are the reason for VGG being able to have that many layers (Wei, 2019a). As a drawback, it can be said that the more layers in an architecture, the longest it will take to train due to the increase in the number of parameters, but as I could train multiple models simultaneously, it was not a problem at all. Having more layers helps the model to perform better at accuracy, and this is one of the reasons why VGG16 stands out at classification tasks (Furaha & Ujjwal, n.d.).

4.3.1.1 VGG-v1

This model's architecture is based on the first of VGG used in the paper "*First dataset of wind turbine data created at national level with deep learning techniques from aerial orthophotographs with a spatial resolution of 0.5 m/pixel*" (Manso-Callejo et al., 2021), which, for image classification, it uses the first two layers of the VGG16 convolutional base and it features 15,240,513 parameters. Figure 35 illustrates the architecture of VGG-v1. The picture has been obtained by using the plot model function.

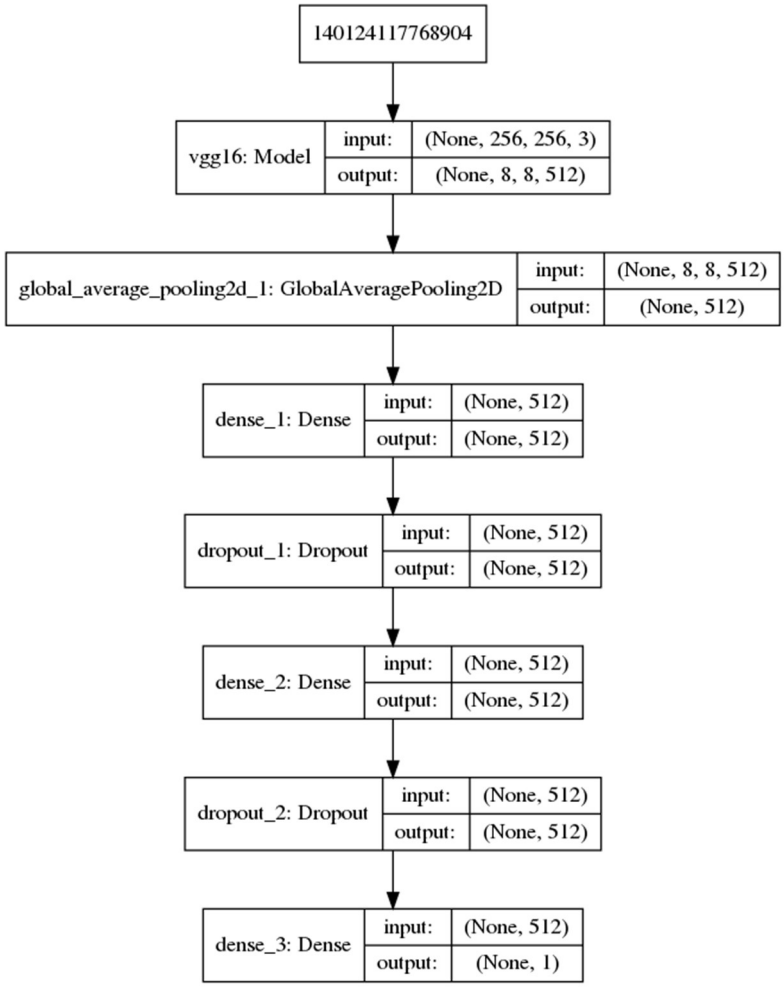


Figure 35: VGG-v1 architecture. Source: Own Elaboration.

The use of two ReLU activation functions, and another Sigmoid one, makes the decision function to be more discriminative, as there are other important architectures that only have one activation function in its structure (Wei, 2019b). Finally, it uses transfer learning, MSRA initialization (for the weights).

4.3.1.2 VGG-v2

This model's architecture uses the same architecture as in VGG-v1 (Manso-Callejo et al., 2021) with an increase in the number of units, which now ascends to 3072. It features 25,733,953 parameters. In this case, there are less modifications than in the VGG from scratch. It uses same initialization as VGG-v1. Figure 36 shows the architecture of this recognition neural network. It can be easily seen the increase in the number of units from the Dense and Dropout layers, which ascends to 3072.

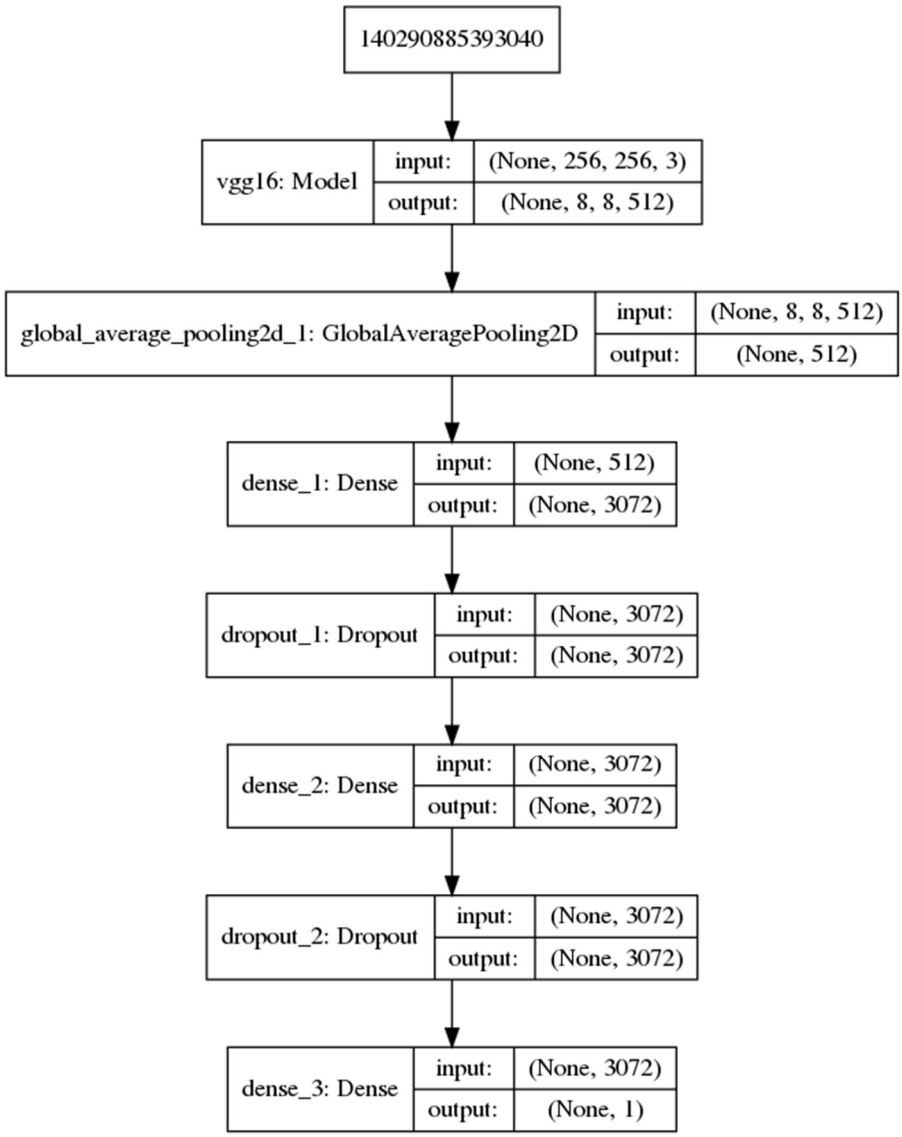


Figure 36: VGG-v2 architecture. Note that it has the same architecture than VGG-v1 with more neurons in each layer. Source: Own Elaboration.

4.3.1.3 VGG from scratch

This model's architecture uses the VGG16 convolutional base and it uses 65,058,625 parameters. For these networks, there are no weights initialized, and the batch size and number of neurons per layer have been adapted in accordance with the tile size. The models corresponding to the 256-tile size used a bigger batch size (50), while the 512-tile size networks reduced it down to 20. Finally, the 1024 architecture used 5 as the batch size.

In regards to the number of neurons used in the dense layers, the 256 and 512-tile size were able to train with 4096 units per dense layer, whereas the 1024 models had to decrease this number to 2048 because otherwise it ran out of memory. Figure 37 shows the last Image Recognition architecture used in this project. Here, there is no dropout layer and instead it features a flatten layer.

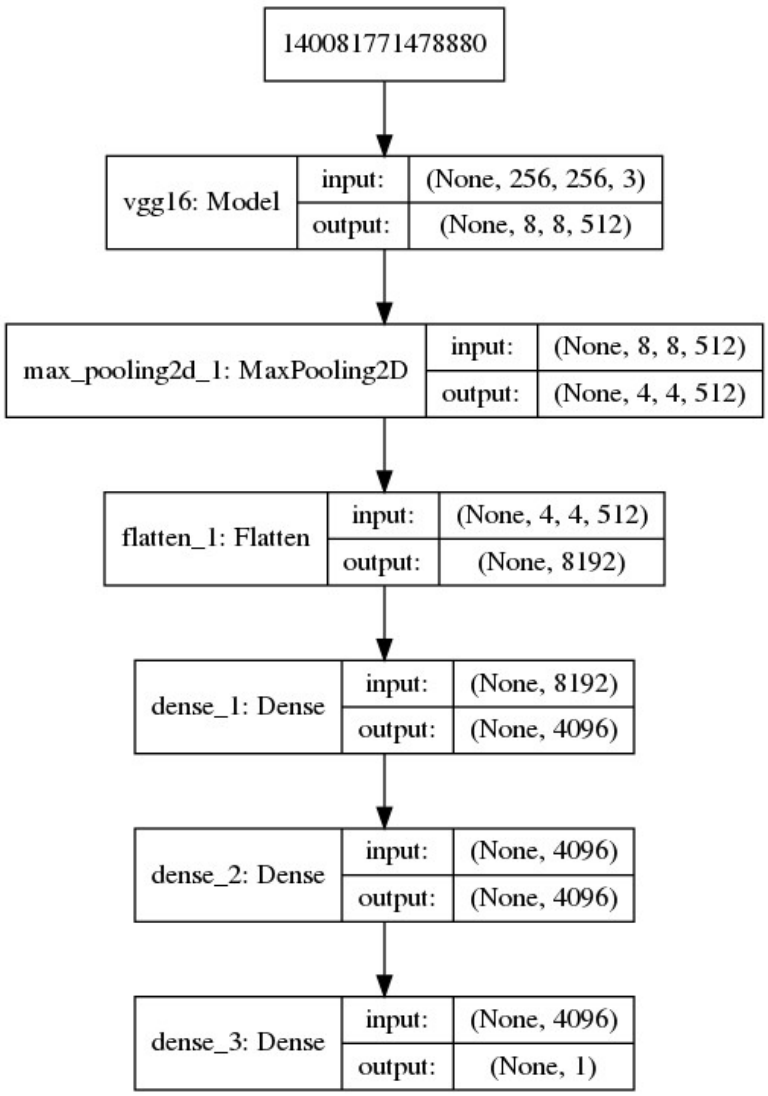


Figure 37 VGG from scratch architecture. Note that this architecture features a flatten layer. On the other hand, there is no dropout layer. Source: Own Elaboration.

Table 7 shows the comparison between the number of parameters used in each image recognition architecture. Note that it increases almost exponentially in the VGG-from-scratch model.

Table 7: Comparison between the number of parameters of the neural networks trained for image recognition.

Model	Number of parameters
VGG-v1	15,240,513
VGG-v2	25,733,953
VGG-from scratch or VGG-no-weights	65,058,625

4.3.2 Image Segmentation

4.3.2.1 U-Net-SEResNeXt50

This model uses as a backbone SE-ResNeXt50, which arises from ResNet (Residual networks) and *"employs squeeze-and-excitation blocks to enable the network to perform dynamic channel-wise feature recalibration."* (*Papers with Code - SE ResNet*, n.d.). A segmentation network uses an encoder and a decoder, and the encoder usually is a classification network, in this case, as the name of the model specifies, U-Net is being used. This whole architecture employs 34,594,177 parameters.

4.3.2.2 LinkNet-EfficientNet-b5

LinkNet-Efficientnetb5 makes use of the LinkNet encoder as the classification network, and this encoder is accompanied by the EfficientNetb5 (Tan & Le, 2020) as the decoder path. *"EfficientNets are a family of image classification models, which achieve state-of-the-art accuracy, yet being an order-of-magnitude smaller and faster than previous models."* (*Tensorflow/Tpu*, 2017/2021). In this case, the number of parameters decreases to 33,700,449.

4.3.2.3 U-Net-InceptionResNetV2

The third of the networks used for my Image Segmentation task has been U-Net-InceptionResNet, which, as SE-ResNeXt, uses the U-Net classifier as the encoder path, and the backbone is in this case InceptionResNet. InceptionResNet is born from the Inception networks but introduces the novelty of using residual connections (Szegedy et al., 2016). For this network, the total of parameters grows to 57,868,721.

Table 8 compares the number of parameters used in each semantic segmentation architecture.

Table 8: Comparison between the number of parameters of the neural networks trained for semantic segmentation.

Model	Number of parameters
U-Net-SE-ResNeXt50	34,594,177
LinkNet-EfficientNetb5	33,700,449
U-Net-InceptionResNet	57,868,721

4.4 Network Training

Once the training process is reached, it is recommended to set the best hyperparameters to reduce the time of processing and loss. One tool which helps to solve both of these problems is the optimizer. An optimizer is a set of algorithms and methods which are going to modify aspects from the learning rate or weights in order to reduce the loss during training.

Regarding to learning rate, it is a tuning mechanism which specifies how big the step to reach the minimum in a loss function should be. If it is big, it can end up making an unstable training, while if it is too low, it can result in a poor training ('Learning Rate', 2021) (Brownlee, 2019). In my case, I have chosen Adam optimizer as it is one of the fastest converging method and it is able to rectify learning rates that are vanishing (Doshi, 2019). This optimizer has been the same for both tasks: image recognition and segmentation.

For both tasks I have used the binary cross-entropy loss function. This function measures how distant from the target value (images containing roads in my case, which will be equal to 1, the opposite is 0) the prediction is for each of the classes. This is followed by the mean of each class to define the loss (*Binary Crossentropy Loss Function | Peltarion Platform*, n.d.) (Godoy, 2018).

4.4.1 Image Recognition

For this task, it has been used a learning rate = 1e-5. In addition, to avoid problems during training arising from the unbalance of the classes (as the number of images containing roads has proven to be bigger as the tile size increases) I have incorporated a class from "sklearn" called class_weights which adjust the weights for both classes to not let the difference in number affect the results at all.

Table 9: The Image recognition networks are preset with the same number of epochs.

Network	Number of epochs
VGG-v1	40
VGG-v2	40
VGG from scratch	40

4.4.2 Image Segmentation

For image segmentation, in addition to the optimizer and loss function specified before, I have used the callbacks, which are different ways to stop a training. These have been imported from Keras.

In my case study, the callbacks I have used are the following ones. In first place, **model checkpoint**, which is used together with model.fit (from Keras) to save a model at a certain point, or its weights, so that it can be loaded after the training. The following one has been **reduce on plateau**. Its principal use is to decrease the learning rate once the model has stopped improving. The arguments I have used for this callback are the factor (value by which the learning rate will be multiplied once it is needed), patience (it specifies the number of epochs with no improvement) and minimum learning rate (the lowest value of learning rate that it can be reached).

Lastly, I have used **early stopping** to stop the process of training once the model has stopped improving. The arguments used here are pretty similar to “reduce on plateau” callback.

Figure 38 shows two graphs corresponding to the loss during training and validation (A), and IoU graphic obtained in the train and validation as well (B).

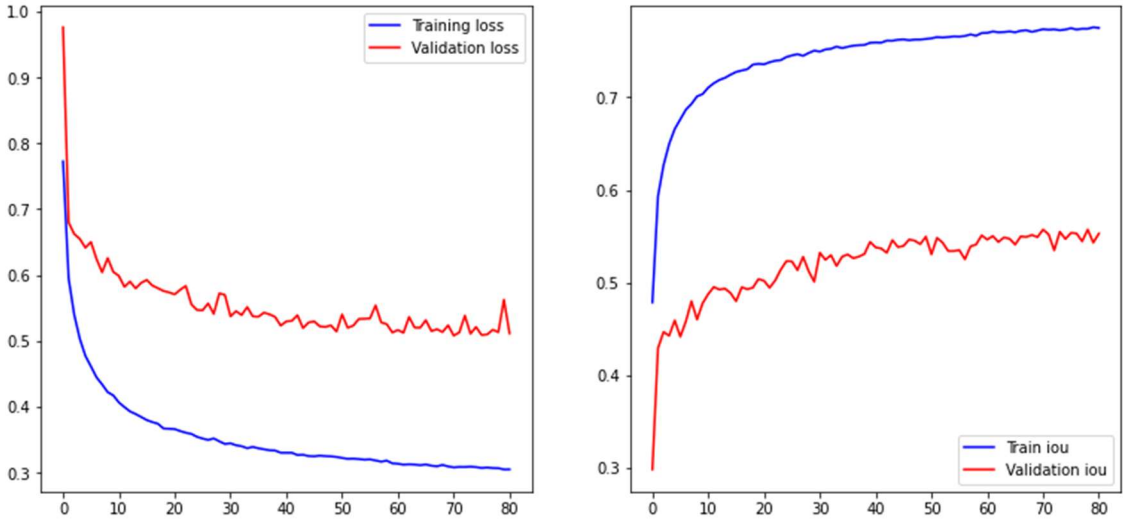


Figure 38 These two graphics have been generated during the traininig of U-Net-InceptionResNet-1 A) Represents the training and validation loss meanwhile B) draws the training and validation increase in the IoU score. Source: Own Elaboration.

Proceeding with table 10, it illustrates the epochs that each model has required to be successfully trained. The high number of epochs in U-Net-InceptionResNet-1 is due to an omission of early stopping in its script. It was solved for the following InceptionResNet models.

Table 10: This table contains the number of epochs per network that have been required for every image segmentation model in images with 256 pixels of tile size.

Network	Epochs
U-Net-SE-ResNeXt50-1	32
U-Net-SE-ResNeXt50-2	27
U-Net-SE-ResNeXt50-3	39
Linknet-Efficientnetb5-1	36
Linknet-Efficientnetb5-2	28
Linknet-Efficientnetb5-3	29
U-Net-InceptionResNet-1	81
U-Net-InceptionResNet-2	49
U-Net-InceptionResNet-3	47

5. RESULTS AND DISCUSSION

5.1 Image recognition results

As happened with other areas of this project, I have divided this one depending on the tile size of the image, so first I am analyzing the results obtained for the 256 images, then I will proceed with the ones from 512 tile size, and eventually, I am going to study the ones belonging to the 1024 dataset.

5.1.1 256x256 tile size

Once the experiments finished, I wrote down the results from every model so that I could compare each one's metrics and decide which would be the most representative architecture. Table 11 contains the results from this tile size.

Table 11: Results of the recognition models which trained the 256x256 tile size images.

Configuration	Loss	Accuracy	Precision	Recall	F1-score	AUC-ROC score
VGG-v1	0.3306	0.8379	0.8463	0.8379	0.8375	0.9386
	0.3590	0.8349	0.8382	0.8349	0.8349	0.9331
	0.3266	0.8345	0.8431	0.8345	0.8341	0.9361
mean	*0.3387*	*0.8358*	*0.8425*	*0.8358*	*0.8355*	*0.9360*
std	*0.0177*	*0.0018*	*0.0041*	*0.0019*	*0.0018*	*0.0028*
VGG-v2	0.3229	0.8377	0.8397	0.8377	0.8377	0.9383
	0.3375	0.8381	0.8399	0.8381	0.8381	0.9362
	0.3186	**0.8395**	**0.8433**	**0.8395**	**0.8395**	**0.9396**
mean	*0.3264*	*0.8385*	*0.8410*	*0.8384*	*0.8384*	*0.9380*
std	*0.0099*	*0.0010*	*0.0020*	*0.0009*	*0.0009*	*0.0017*
VGG from Scratch	0.3392	0.8374	0.8374	0.8374	0.8374	0.9324
	0.3430	0.8371	0.8373	0.8371	0.8371	0.9300
	0.3732	0.8328	0.8328	0.8328	0.8327	0.9213
mean	*0.3518*	*0.8358*	*0.8358*	*0.8358*	*0.8357*	*0.9279*
std	*0.0186*	*0.0026*	*0.0026*	*0.0026*	*0.0026*	*0.0058*

As it can be seen in bold, the best model has been VGG-v2 in its third repetition. I have chosen this one mainly due to its low loss and because of it having the highest AUC-ROC score. in this case, there has not been doubts as the rest of metrics are the highest ones as well (accuracy, precision, recall or F1-score). Despite being a clear choice, it is true that the rest of the models have performed pretty similar to the final selected. In the following tile-sizes there are some models who differ more in terms of accuracy or loss score. Figure 39 illustrates the confusion matrix obtained from the model with best results.

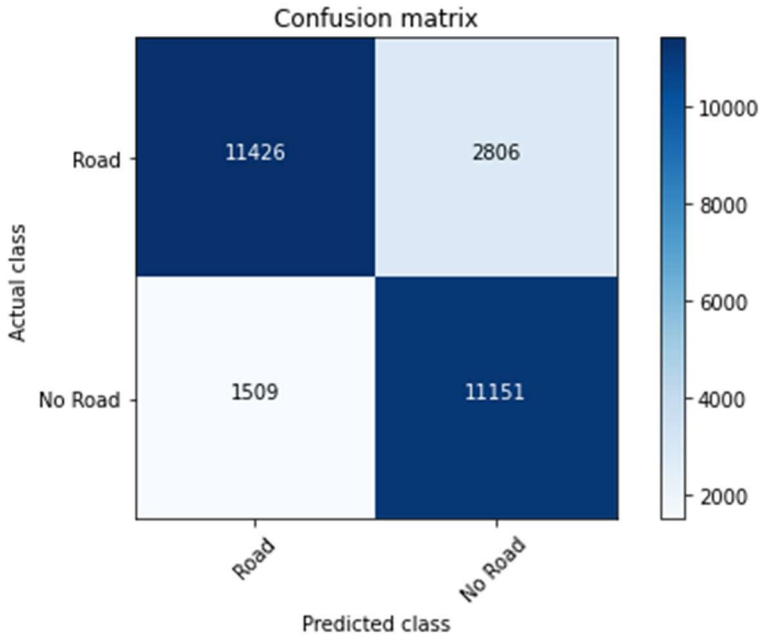


Figure 39: This image represents the confusion matrix obtained from the VGG-v2-3. It can be seen how superior the magnitud of the hits is compared to the number of misses

In this particular case, the network reaches levels of accuracy of 84% approximately, which, considering that the number of roads and no-roads is fairly uniform, it is a good result. It is true that in this model, it can be seen that the False Positives are almost twice the number of False Positives. The reason of this difference can be that in this tile size, the size of the roads is bigger than in the following tile sizes. This bigger size in the image means that the deep learning models could mistake the path thinking it is just a part of the field or a dry area with no roads inside. Anyway, the rest of confusion matrixes have different results in the False positives and False negatives, alternating which of both is bigger depending on the model. As an example, the VGG-from-scratch-3 confusion matrix does have 2110 False Positives and 2387 False Negatives, with a slightly worse True Positives score. I have noticed that if all the confusion matrixes from this tile size are compared, the number of False Positives or Negatives will vary and the affected class from this variation is in almost every case the True positives, with a quite constant True Negative score. For the models, it is more easy-recognizable to identify no-roads that classifying those images containing paths.

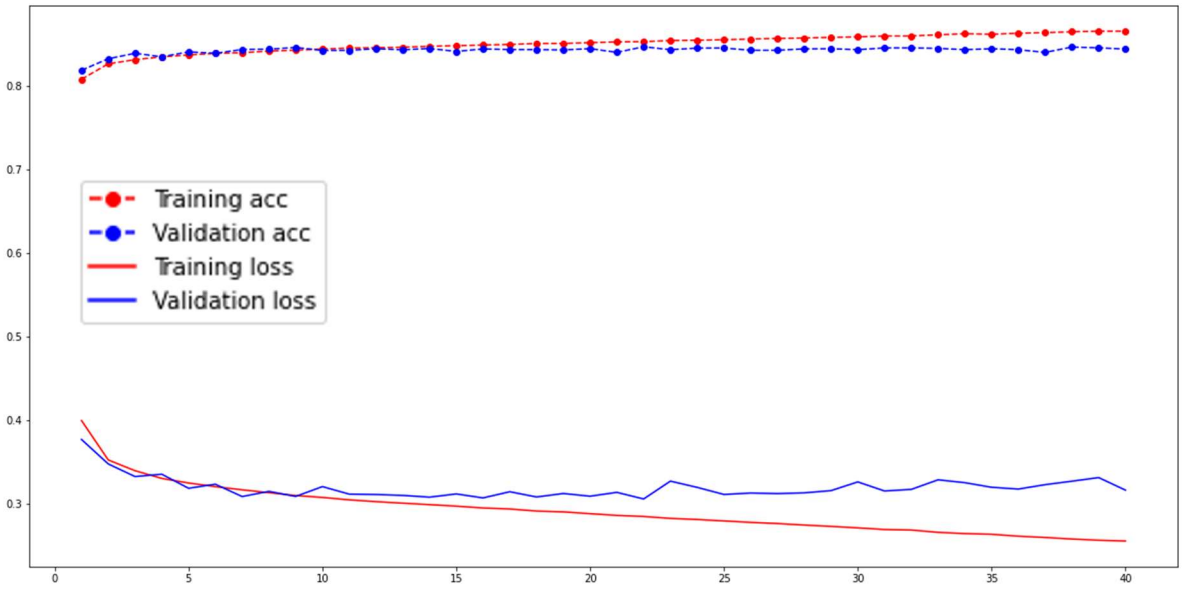


Figure 40: In this graphic, it can be seen the slow increase in the training and validation accuracy, while both training and validation decrease as well. The training loss decreases much faster than the validation one, which turns to be more or less constant. The graphic belongs to the third VGG-v2 model.

Source: Own Elaboration.

5.1.2 512x512 tile size

Moving forward, the next tile size to evaluate has been the 512x512, which has given high quality results.

Table 12: Results of the recognition models which trained the 512x512 tile size images.

Configuration	Loss	Accuracy	Precision	Recall	F1-score	AUC-ROC score
VGG-v1	0.2255	0.9071	0.9071	0.9071	0.9071	0.9638
	0.2323	0.9065	0.9050	0.9065	0.9047	0.9651
	0.2182	0.9092	0.8884	0.8814	0.8848	0.9668
	0.2253	*0.9076*	*0.9002*	*0.8983*	*0.8989*	*0.9652*
	0.0071	*0.0014*	*0.0102*	*0.0147*	*0.0122*	*0.0015*
VGG-v2	0.2345	0.9067	0.9056	0.9067	0.9060	0.9639
	0.2360	0.9109	0.9098	0.9109	0.9101	0.9663
	0.2264	**0.9111**	**0.9100**	**0.9111**	**0.9103**	**0.9664**
	0.2323	*0.9095*	*0.9085*	*0.9096*	*0.9088*	*0.9655*
	0.0052	*0.0025*	*0.0025*	*0.0025*	*0.0024*	*0.0014*
VGG from Scratch	0.2272	0.9025	0.9009	0.9025	0.9007	0.9620
	0.2316	0.9034	0.9025	0.9034	0.9006	0.9632
	0.2405	0.8951	0.8940	0.8951	0.8914	0.9623
	0.2331	*0.9003*	*0.8991*	*0.9003*	*0.8976*	*0.9625*
	0.0068	*0.0046*	*0.0045*	*0.0046*	*0.0053*	*0.0006*

For this intermediate tile size, it has been noticed an increase in the accuracy levels in every model. The percentage is superior to 90% in most of the cases. Only in one case

the accuracy is below 0.9, which is the third VGG from scratch model. This one could be considered as the worst performing model, with the highest loss as well. In general, it could be said that for this tile size the VGG from scratch has performed weaker than the other two architectures (making the exception of VGG from scratch-1, which has performed consistently as well).

In regards to the highest quality performance metrics, the results have been more even than the previous one. Despite having chosen the third VGG-v2 model as the best as happened before, it can be seen that the third VGG-v1 has proven to be a consistent competitor as well, so, why I have selected VGG-v2 as the most representative model in this case is because in this particular image size, I have considered that the difference in the loss (~0.08) and AUC-ROC score (~0.0004) is not that relevant as the other metrics are lower for the benefit of VGG-v2. Obviously, the better the results are, the better confusion matrix are, as shown in Figure 41.

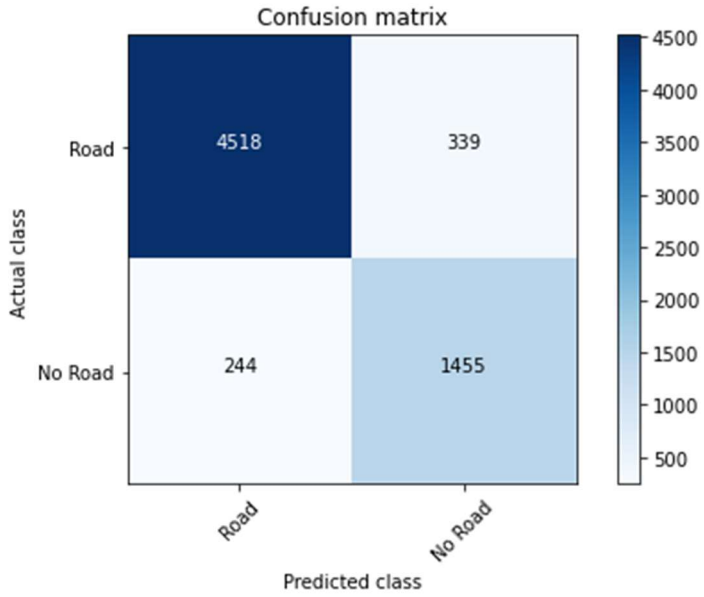


Figure 41: This image represents the confusion matrix obtained from the VGG-v2-3. Again, both classes, "roads" and "no-roads" have been clearly differentiated. Source: Own Elaboration.

This improvement means that all the networks have become better at identifying the images containing roads or not, than the previous case, but what has changed between both tile sizes?

There are two differences with respect to the previous example, the first one is that in the training set, the classes are not equally divided anymore (now I have 72% for roads, 27% for no-roads, while in the 256 tile-size, I had 49 vs 51%). This unbalance does not affect in a negative way as it will be seen in the following case, but rather the network

with this quantity of no-roads is able to tell the difference between the two classes in an easier way. On the other hand, the difference I believe matters the most, is that the bigger size gives the network a better perspective of the image. As an example, this perspective works better against occlusions, while an image with a road that would be miss-classified in the 256 due to a tree occupying part of the road, in 512 tile size the model is able to see a bigger portion of the path making the decision easier. Hence, there are more true positives and less False Negatives in the confusion matrixes.

5.1.3 1024x1024 tile size

The last image size results, 1024x1024, have been more difficult to analyze as there have been some challenges on the road to solve.

I started with a 90-5-5% division in my dataset (90% of images in the training dataset), as it can be seen in **4.1**. I was able to train VGG-v1 and VGG-v2 models correctly, with decent results in the quality metrics and confusion matrixes. Then, the problem came when I finished my VGG from scratch models, as the results were the following ones:

Table 13: First attempt of VGG from scratch obtaining poor results.

Configuration	Loss	Accuracy	Precision	Recall	F1-score	AUC-ROC score
VGG from Scratch	1.6386	0.8983	0.8070	0.8983	0.8502	0.5000
	1.6386	0.8983	0.8070	0.8983	0.8502	0.5000
	1.6386	0.8983	0.8070	0.8983	0.8502	0.5000

In this table, the first thing to notice is the high loss value, which is almost three times higher than the previous loss score from the lower image sizes. Then, the accuracy, precision and other parameters seem to have been quite decent, but AUC-ROC score shows how poorly the models have been trained, as only 1 class seems to have been learnt. This can be demonstrated in the confusion matrixes arising from these models. Here is an example:

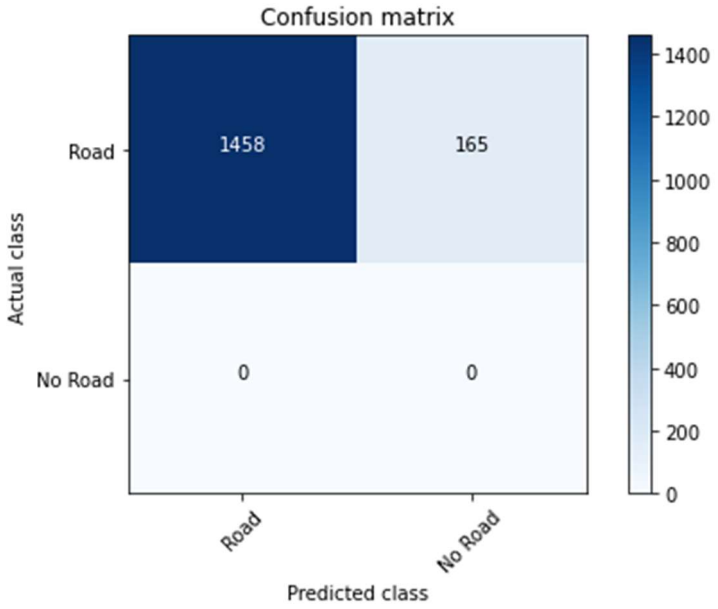


Figure 42: Matrix confusion of the first VGG from scratch that was tried. Only roads have been learnt. Source: Own Elaboration.

At this point, I noticed that the bad learning was being caused by a high unbalance in the classes. Roads had almost 26,000 examples and no-roads had barely 3,000 images. This problem caused the network to perform correctly by just predicting every image as "Road" and still it was able to get good accuracy or precision results.

To fix this, I decided to balance the number of images labelled as roads taking advantage of all the no-roads images that I had, so the new "road vs no-road" distribution in the training and validation dataset was the following one:

- Training set: 3149 images labelled as "road"; 2969 pictures labelled as "no-roads".
- Validation set: 175 images labelled as "road"; 165 pictures labelled as "no-roads".

The test set was not reduced as it does not affect the network during training and, on the other hand, it was better to have as much images as possible for the results. Once the new dataset was prepared, some modifications in the script which would run the models were necessary, as if I keep the number of epochs in 50, the network could overfit. Therefore, I decided to set 10 epochs in the VGG-from-scratch models as the time to learn. In figure 43 it can be seen the improvement in terms of hits in the confusion matrix from the VGG from scratch model.

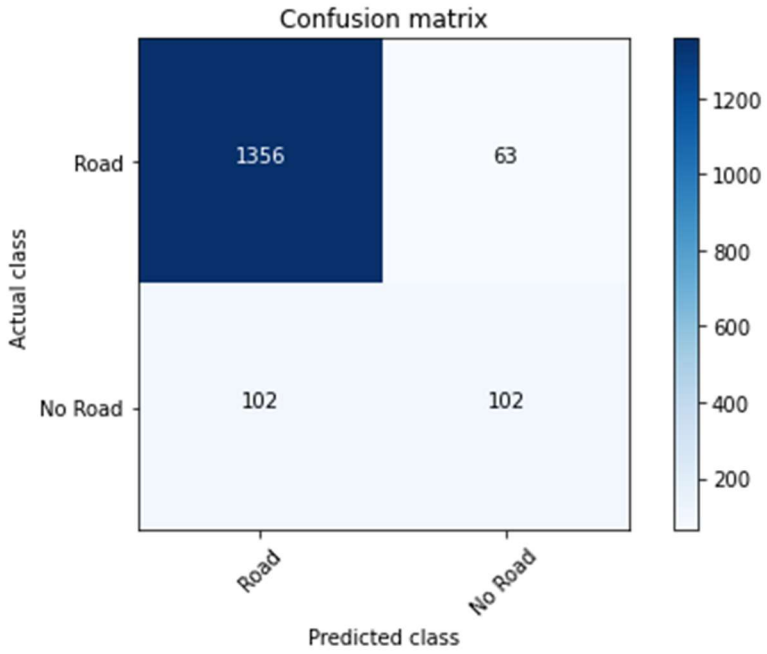


Figure 43: This confusion matrix belongs to the last VGG from scratch. As it can be seen, the results are worse than in other tile sizes but it is able to recognize no-roads. Source: Own Elaboration.

Finally, as the VGG from scratch networks gave good results, I could collect all the quality metrics from the models:

Table 14: Results of the recognition models which trained the 1024x1024 tile size images.

Configuration	Loss	Accuracy	Precision	Recall	F1-score	AUC-ROC score
VGG-v1	0.1208	0.9600	0.9588	0.9600	0.9569	0.9806
	0.1116	0.9618	0.9606	0.9618	0.9593	0.9827
	0.1254	0.9643	0.9649	0.9643	0.9611	0.9822
mean	*0.1193*	*0.9620*	*0.9614*	*0.9620*	*0.9591*	*0.9818*
std	*0.0070*	*0.0022*	*0.0031*	*0.0022*	*0.0021*	*0.0011*
VGG-v2	**0.1033**	**0.9643**	**0.9633**	**0.9643**	**0.9620**	**0.9844**
	0.1157	0.9680	0.9674	0.9680	0.9661	0.9769
	0.1597	0.9544	0.9551	0.9544	0.9490	0.9755
mean	*0.1262*	*0.9622*	*0.9619*	*0.9622*	*0.9590*	*0.9789*
std	*0.0296*	*0.0070*	*0.0063*	*0.0070*	*0.0089*	*0.0048*
VGG from Scratch	0.3371	0.8583	0.9112	0.8583	0.8502	0.9106
	0.5796	0.8897	0.8662	0.8897	0.8743	0.6656
	0.2943	0.8983	0.9093	0.8983	0.9030	0.8782
mean	*0.4037*	*0.8821*	*0.8956*	*0.8821*	*0.8758*	*0.8181*
std	*0.1539*	*0.0211*	*0.0255*	*0.0211*	*0.0264*	*0.1331*

This time, it is clear that the best performing model is the first VGG-v2, with the lowest loss score and the highest AUC-ROC, precision, etc. VGG-v1-2 and VGG-v2-2 have obtained really consistent results in their models, and the AUC-ROC score is higher than in previous tile sizes. On the contrary, VGG from scratch has obtained worse metrics due to its difficulty in tuning the best hyperparameters to get an optimal training. In fact,

the best VGG-from scratch has a loss score twice worse than the worst VGG-v2. These models from scratch definitely have not adapted to the highest tile size as good as the other architectures.

Despite the better results, the true negatives are only twice the number of false positives, as illustrated in figure 44, which means the model struggles identifying those images lacking paths. This time, the reason for this happening has nothing to do with the balance of the number of images but with the noise in the image. This noise is caused by most part of the image not being roads, and just a tiny portion representing paths. In addition to this, the model is being told that most of the images contain roads, so this could confuse the training, leading to incorrect layer's activation. This is proven in the confusion matrixes, as the number of images incorrectly classified as "No-road" is much higher than the False Positives, showing how challenging is for the model to differentiate the ways.

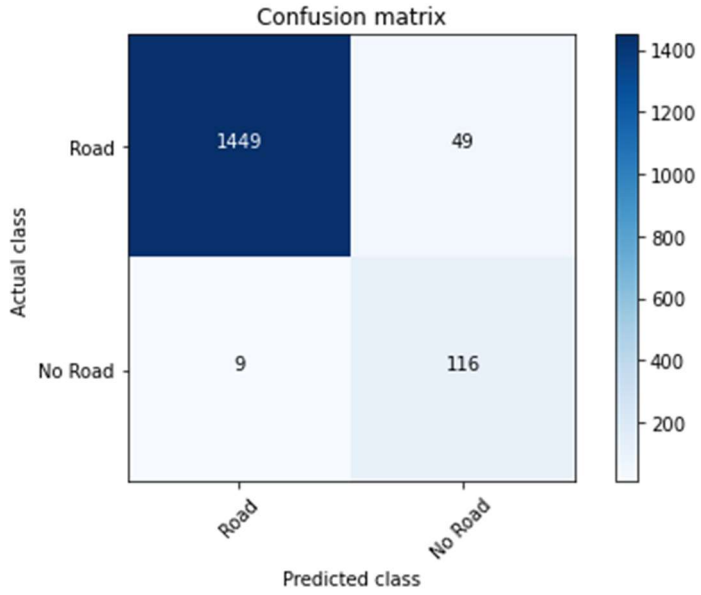


Figure 44: Confusion matrix from the first VGG-v2 model. Source: Own Elaboration.

5.2 Image Segmentation results

As far as image segmentation results is concerned, I find it interesting to analyze the results in the validation and test set in order to choose the best model.

Once the model is chosen, for every tile size, I test the models to perform the segmentation task in six different areas: Mediterranean, dry area, urban, rural, green spaces and coastal. It is going to be shown the predicted mask along with the current mask and the orthoimage.

5.2.1 256x256 tile size

For the first tile size, the results have been the following ones:

Table 15: Results of the image segmentation task from 256x256 tile size.

	Validation			Test		
Configuration	**Loss**	**IoU score**	**F1 score**	**Loss**	**IoU score**	**F1 score**
U-Net – SEResNeXt50	0.6302	0.4879	0.5871	0.6260	0.4966	0.5956
	0.5525	0.5179	0.6177	0.5420	0.5280	0.6274
	0.5641	0.5058	0.6087	0.5548	0.5154	0.6180
mean	*0.5823*	*0.5039*	*0.6045*	*0.5743*	*0.5133*	*0.6137*
std	*0.0419*	*0.0151*	*0.0157*	*0.0453*	*0.0158*	*0.0163*
LinkNet EfficientNet-b5	0.5284	0.5432	0.6422	0.5192	0.5521	0.6504
	0.5458	0.5302	0.6299	0.5360	0.5391	0.6380
	0.5339	0.5347	0.6370	0.5260	0.5426	0.6442
mean	*0.5360*	*0.5360*	*0.6364*	*0.5271*	*0.5446*	*0.6442*
std	*0.0089*	*0.0066*	*0.0062*	*0.0084*	*0.0067*	*0.0062*
U-Net – InceptionResnet	**0.5113**	**0.5529**	**0.6500**	**0.5036**	**0.5610**	**0.6574**
	0.5391	0.5292	0.6286	0.5310	0.5373	0.6364
	0.5417	0.5234	0.6249	0.5343	0.5315	0.6325
mean	*0.5307*	*0.5352*	*0.6345*	*0.5230*	*0.5433*	*0.6421*
std	*0.0169*	*0.0156*	*0.0136*	*0.0169*	*0.0156*	*0.0134*

In this case, the first U-Net-InceptionResNet results stands out above the rest, by having the lowest loss score in both validation and test, and because of having the highest IoU score, which was the specific quality metric to evaluate segmentation models. In this task, the results obtained are far more dispersed than in image recognition. A possible runner up would be the first LinkNet model, which has obtained a decent loss score in both validation and test sets. On the other hand, the Unet-SEResNeXt50 models have proven to give worse results in this particular tile size.

Figure 45 illustrates the training graphs obtained from the U-Net-InceptionResNet-1, which I consider to be the best in 256x256 size.

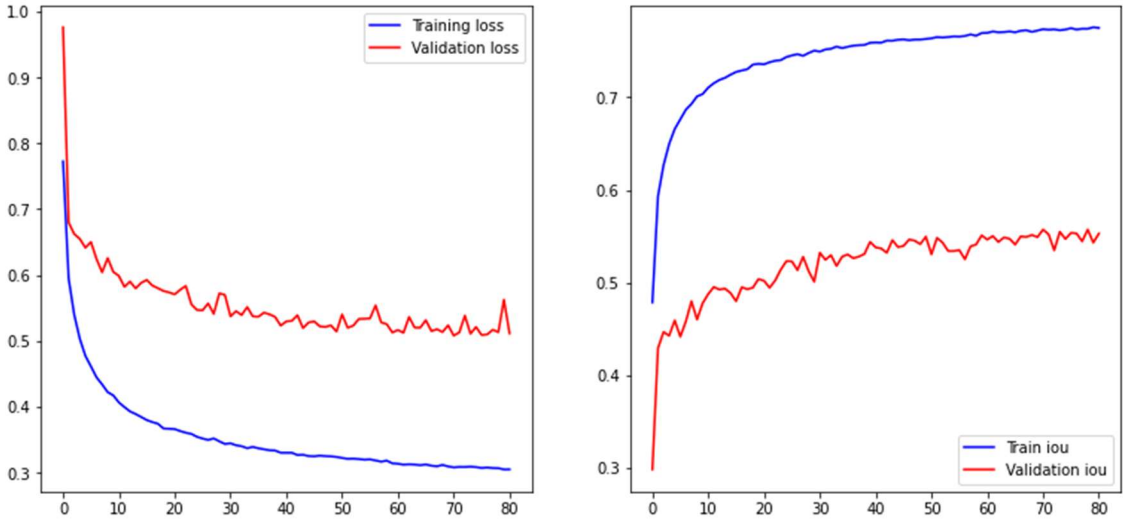


Figure 45: This graphics shows the training progress in terms of loss and IoU score from U-Net-InceptionResNet-1. Source: Own Elaboration.

Figure 46 and 47 illustrates the results from this task in the selected areas chosen before (presented in **3.1.2**). This study is done to support the results from semantic segmentation adding a qualitative analysis. In this tile size, there have been some interesting findings which I have noticed. Firstly, when it comes to images from agricultural areas, where some paths might not be clearly defined, the models tend to struggle, or they doubt in the magnitude of the road (there are cases in which a network has thought the path was bigger than it really is, or they have omitted part of the layout).

Secondly, these 256-tile size, as mentioned before, work worse when occlusions come up covering the whole road. On the other hand, if there are just some little occlusions featuring in the image, the path is usually well predicted. In third place, those images with green spaces and roads seems to be easier to handle for the models than other areas. It can be due to the difference in the brightness of trees and the path or the type of vegetation. The case of dry area seems to have been misinterpreted, as the model has apparently considered the whole image as a road. However, in bigger sizes, this part of road is correctly defined. The results are given in terms of IoU, which defines a percentage obtained from the formula shown at **2.6.2** (it uses the mask from the training dataset, and compares to the predicted one).

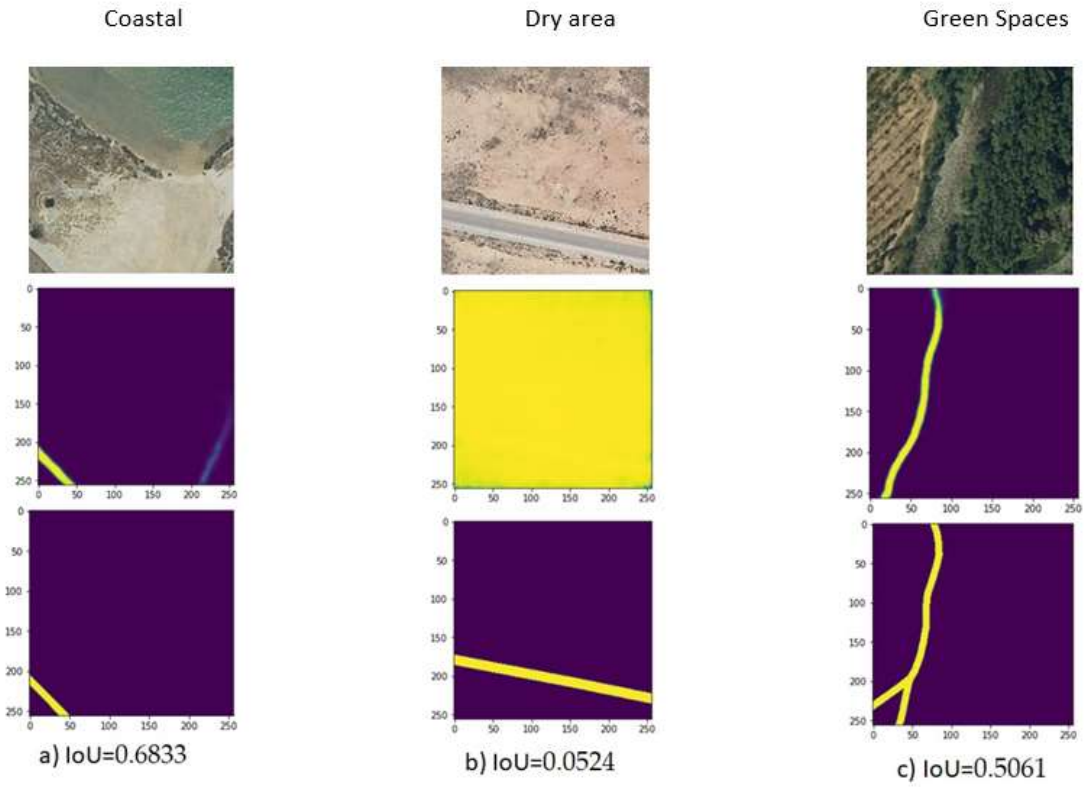


Figure 46: Comparison of the mask predicted (second row) and real mask (third row) depending on the zone, which is the first row. These are the three first areas of study. Source: Own Elaboration.

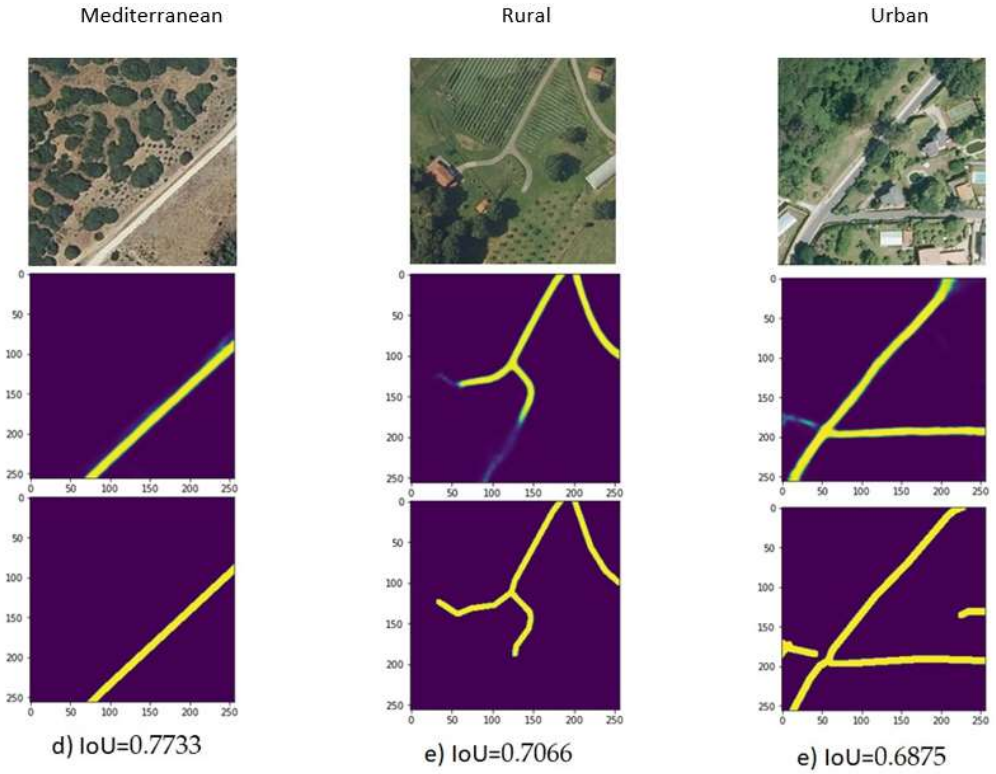


Figure 47: Comparison of the mask predicted (second row) and real mask (third row) depending on the zone, which is the first row. These are the three last areas of study.

5.2.2 512x512 tile size

For the second tile size, these have been the results:

Table 16: Results of the image segmentation task from 512x512 tile size.

	Validation			Testing		
Configuration	**Loss**	**IOU score**	**F1 score**	**Loss**	**IOU score**	**F1 score**
U-Net – SEResNeXt50	0.7363	0.4154	0.5441	0.7522	0.4069	0.5347
	0.7515	0.3425	0.4696	0.7594	0.3363	0.4618
	0.6960	0.4231	0.5503	0.7081	0.4174	0.5448
Mean	*0.7279*	*0.3937*	*0.5213*	*0.7399*	*0.3869*	*0.5137*
Std	*0.0287*	*0.0444*	*0.0449*	*0.0278*	*0.0441*	*0.0453*
LinkNet EfficientNet-b5	0.8860	0.4480	0.5659	0.9116	0.4441	0.5610
	0.7398	0.4636	0.5834	0.7503	0.4582	0.5774
	0.6255	**0.4924**	**0.6154**	**0.6301**	**0.4895**	**0.6127**
Mean	*0.7504*	*0.4680*	*0.5883*	*0.7640*	*0.4639*	*0.5837*
Std	*0.1306*	*0.0225*	*0.0251*	*0.1412*	*0.0233*	*0.0264*
U-Net – InceptionResNetv2	0.8900	0.4313	0.5472	0.9115	0.4236	0.5380
	0.9310	0.4369	0.5488	0.9584	0.4298	0.5395
	0.8975	0.3715	0.4923	0.9186	0.3645	0.4840
Mean	*0.9062*	*0.4132*	*0.5294*	*0.9295*	*0.4060*	*0.5205*
Std	*0.0218*	*0.0362*	*0.0322*	*0.0253*	*0.0361*	*0.0316*

In this case, the third LinkNet-EfficientNet-b5 has proven to give better results than any of the other ones, and despite the metrics being worse than in the 256x256 tile size, what I have noticed is that this model gives better visual results, at least in the specific zones I chose before testing the model.

In regards to visual aspects, as it is shown in figure 48 and 49, 512-tile size usually overcome efficiently the obstacle of occlusion. Here, another obstacle has appeared, and it is the difficulty for the models when deciding where a road ends its layout (or where does it start) if it happens in the middle of the image. The networks work better if a path traverses the whole picture horizontally, vertically or diagonally.

In urban areas, it is confusing to learn a path with differences in color in the asphalt due to this management of roads not being done simultaneously. Lastly, another aspect to bear in mind is the label noise (Song et al., 2021), as there have been some images which no roads defined by the person in charge of the digitalization, and it results in a slight confusion in the training process.

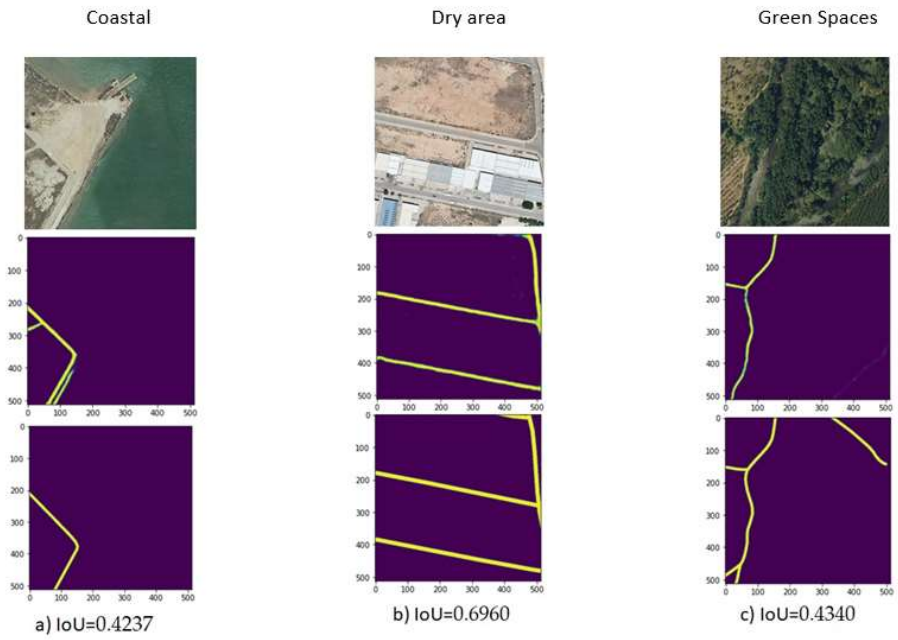


Figure 48: The structure of rows is the same as in the previous tile size: real image, mask predicted and real mask. These are the first three areas of study, and, as it shows, now it does not mistake the dry area, as it happened before.

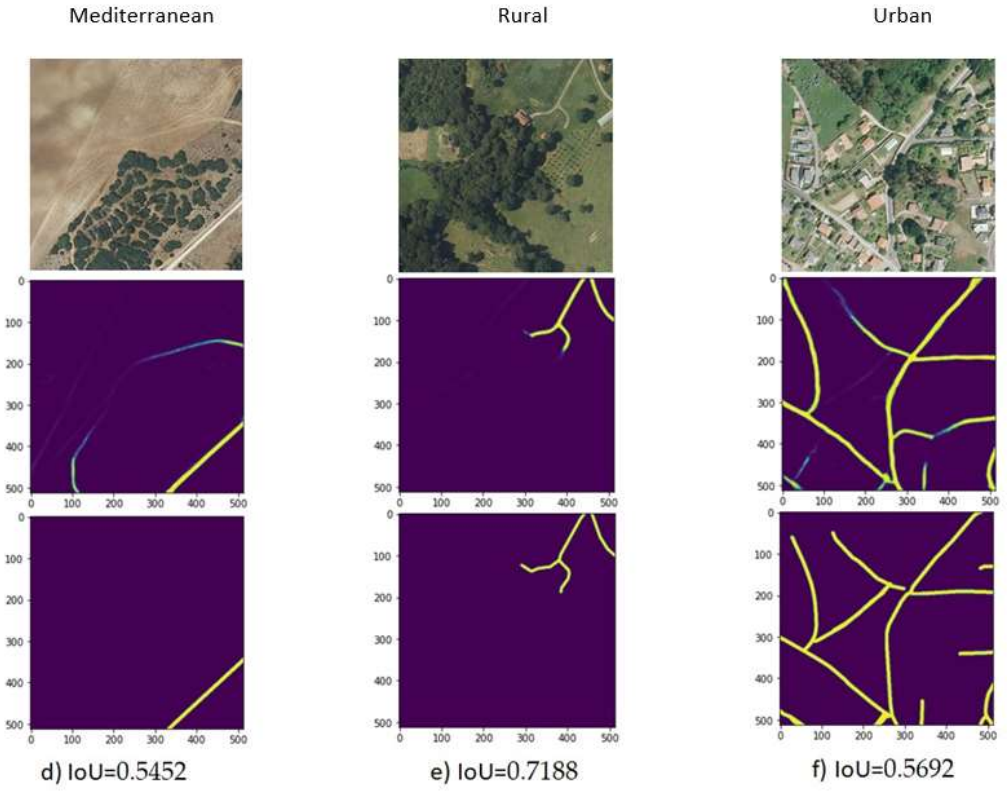


Figure 49: These are the three last areas of study in this tile size images. The model seems to recognize more paths than it should, as in the Mediterranean example.

5.2.3 1024x1024 tile size

Eventually, it is time to see how the segmentation models have learnt from the biggest tile size of all. It is an interesting situation as they need to identify roads, which, in most of the cases are just a small line which represents a tiny proportion of the whole image, leading to possible problems or missing some secondary roads or paths.

Table 17: Table of results corresponding to the 1024x1024 tile size images.

	Validation			Testing		
Configuration	Loss	IOU score	F1 score	Loss	IOU score	F1 score
U-Net – SEResNeXt50	1.1761	0.3642	0.4846	1.2164	0.3675	0.4871
	1.3310	0.3717	0.4896	1.2834	0.3661	0.4847
	1.2712	0.3458	0.4615	1.3157	0.3492	0.4642
mean	1.2594	0.3606	0.4786	1.2718	0.3609	0.4787
std	0.0781	0.0133	0.0150	0.0507	0.0102	0.0126
LinkNet EfficientNet-b5	1.2663	0.2714	0.3781	1.3047	0.2741	0.3826
	1.1728	0.2973	0.4109	1.2064	0.3005	0.4149
	1.2995	0.2758	0.3804	1.3483	0.2748	0.3795
mean	1.2462	0.2815	0.3898	1.2865	0.2831	0.3923
std	0.0657	0.0139	0.0183	0.0727	0.0150	0.0196
U-Net – InceptionResNetv2	**1.1554**	**0.3445**	**0.4646**	**1.1895**	**0.3505**	**0.4715**
	1.2650	0.3111	0.4213	1.3068	0.3149	0.4255
	1.2999	0.3153	0.4322	1.3416	0.3212	0.4396
mean	1.2401	0.3236	0.4394	1.2793	0.3288	0.4455
std	0.0754	0.0182	0.0225	0.0797	0.0190	0.0236

As I could figurate out, these models have been the worst to perform due to the issue I mentioned before. Logically, there is a high loss in these images as it is harder for the model to distinguish small ways in an image with such a big quantity of pixels, when it can be mistaken with any other type of path, such as plowed roads.

For this tile size images, I have been doubting between two models in order to choose the most precise one: U-Net-SEResNeXt50-1 and U-Net-InceptionResNetv2-1. In spite of the first one having better results at IoU and F1 score, I decided to select U-Net-InceptionResNetv2 as it has a lower loss score in both validation and test set, which in other tile sizes might have not been the case, but as in this case there is this problematic of the models struggling to identify the roads properly, I reckoned it should be more deserved to go for the Inception one. On the other hand, almost every model except for the two mentioned before and LinkNet EfficientNet-b5-2, have given worse results in terms of a loss score in the test above 1.30. It can be seen as well that the LinkNet models are the ones with the lowest IoU score, barely reaching 0.30.

As far as the specific zones is concerned, the model has performed like Figure 50 shows.

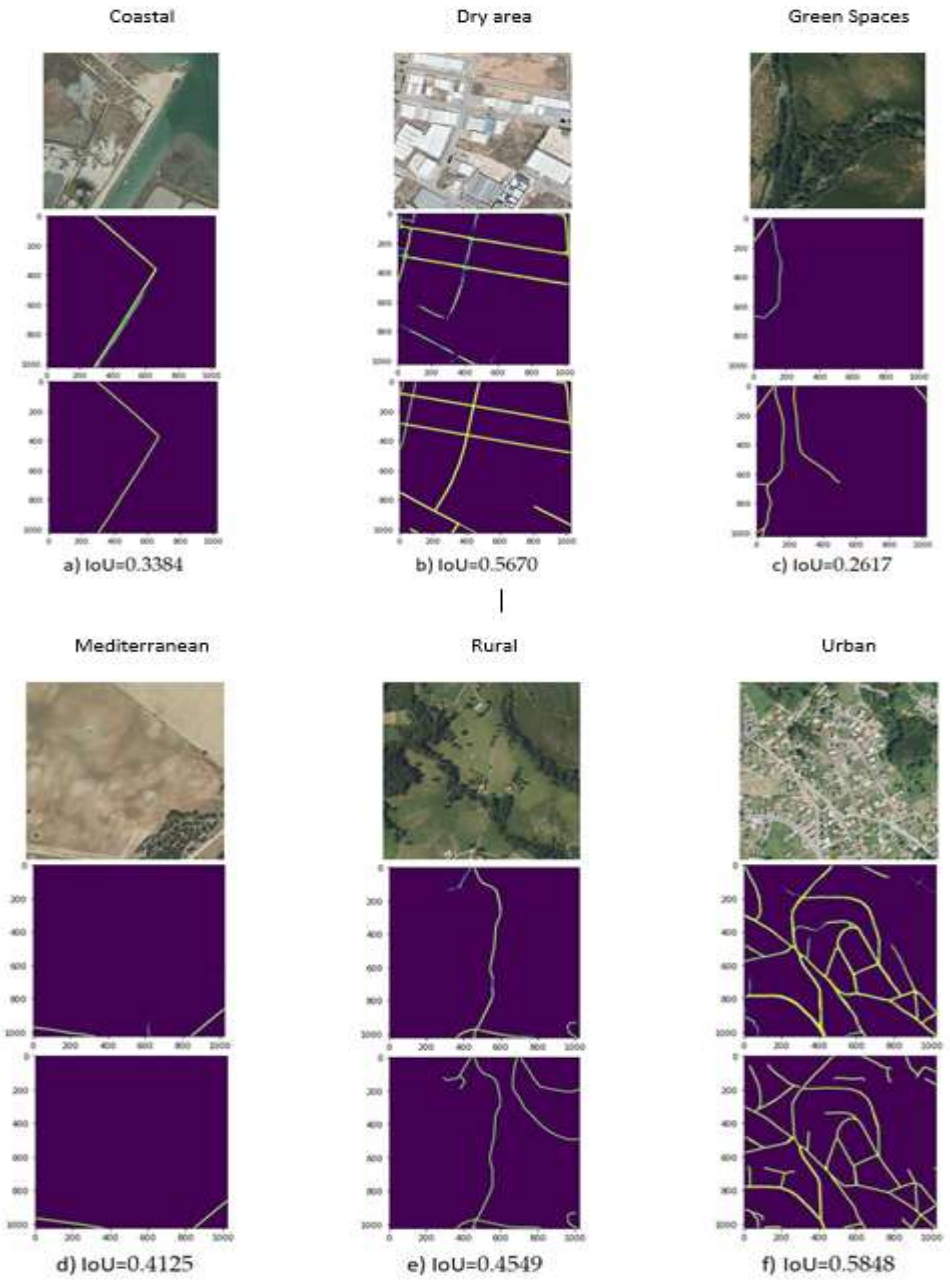


Figure 50: Testing the Inception model with specific zones used in other tile size images.

To conclude with the visual aspect in this tile size, 1024 works better with occlusions in certain parts of the road, but if the whole layout is constantly affected by this, it still struggles. The problem mentioned in the 512-tile size related to difficulties of roads ending in the middle of the image persists here as well. Finally, it is easy to notice that the model misses some small paths in the dry area or green spaces, but it is positive to see that in the urban picture, the model has been able to recognize almost perfectly such

a complex network of roads. In most of the cases, the InceptionResNet model has reached levels of IoU really close to 0.5.

Table 18: Results for each specific zone and tile size

Tile size	Configuration	Area	IOU score zone	Loss	IOU score	F1 score
256	U-Net – InceptionResNet-1	Coastal	0.6833	1.6941	0.50697	0.63716
		Dry Area	0.0524			
		Green Spaces	0.5061			
		Mediterranean	0.7733			
		Rural	0.7066			
		Urban	0.6875			
	mean		0.4139			
	std		0.3254			
512	LinkNet – EfficientNetb5-3	Coastal	0.4237	0.5409	0.52715	0.68314
		Dry Area	0.6960			
		Green Spaces	0.4340			
		Mediterranean	0.5452			
		Rural	0.7188			
		Urban	0.5692			
	mean		0.5179			
	std		0.1543			
1024	U-Net – InceptionResNet-1	Coastal	0.3384	0.67971	0.39808	0.56005
		Dry Area	0.5670			
		Green Spaces	0.2617			
		Mediterranean	0.4125			
		Rural	0.4549			
		Urban	0.5848			
	mean		0.3890			
	std		0.1588			

This table shows how the model has given better results in the 512 tile size images examples I chose. Obviously, 6 images are not representative as a whole dataset is, but this decision is supported with the results mentioned in **5.2.2**.

6. BUDGET

This section pretends to estimate the costs required to complete this project. They have been decomposed into three groups. In first place, the software cost, in which I have considered the software featuring in **3.3** and the office license. Secondly, it has been taken into consideration the hardware costs. Then, I have calculated the Labor cost, considering myself a Geoinformatics engineer.

6.1 Software

The software used in this budget's calculation has been done considering the **3.3** section and checking the price of Windows 10 Enterprise and Microsoft Office yearly license, as Table 19 illustrates. It has been considered an useful life of 4 years and a year for Windows 10 Enterprise and Microsoft Office respectively. Each cost is the result of multiplying the gross cost by the amortization factor (which, at the same time is calculated by dividing the usage time by the useful life). The software resulted in a final cost of **78.02 €**.

Table 19: Budget calculated considering the software used in this project.

Material	Gross Cost	Unit	Usage Time (Months)	Useful Life (Months)	Amortization Factor	Total Cost (€)
Ubuntu 20.04.3 LTS	Free	1	4	-	-	0,00 €
Windows 10 Enterprise	259,00 €	1	7	48	7/48	37,77 €
Microsoft Office	69,00 €	1	7	12	7/12	40,25 €
Anaconda	Free	1	4	-	-	0,00 €
Python	Free	1	4	-	-	0,00 €
TensorFlow	Free	1	4	-	-	0,00 €
Keras	Free	1	4	-	-	0,00 €
NVIDIA CUDA's libraries	Free	1	4	-	-	0,00 €
Jupyter-notebook	Free	1	5	-	-	0,00 €
Total						**78,02 €**

6.2 Hardware

As far as hardware is concerned, I have considered my own personal computer, the two servers belonging to IGN and the solid-state disk. As in **6.1**, the total cost has been

calculated by multiplying the gross cost by the amortization factor. This is illustrated in table 20.

Table 20: Total cost of the hardware used in this project.

Material	Gross Cost (€)	Unit	Usage Time (Months)	Useful Life (Months)	Amortization Factor	Total Cost (€)
Own PC (HP Pavillion Laptop 15-cs2xxx)	1.300,00 €	1	7	120	1/17	75,83 €
Cartobot1 + NVIDIA GeForce RTX 2080 Ti	1.900,00 €	1	3	36	1/12	158,33 €
IGN1 + LLVM 10.0.0	26.500,00 €	1	3	36	1/12	2.208,33 €
SSD2	60,00 €	1	3	120	1/40	1,50 €
Total						**2.444,00 €**

The final hardware cost has resulted in **2,444** €.

6.3 Labor cost

The labor cost has been calculated based on the salary of a Geoinformatics engineer, which, at the moment is 26,084.88€. Then, I have considered 360 hours of work, from section **4.** This number of hours is the result of multiplying 12 ECTs by 30h per ECTs. Then, I have conversed the annual salary into hourly wage, so that I could calculate the total cost by multiplying the salary per hour by the 360h dedicated to this project. This can be seen in table 21.

Table 21: Total labor cost

Position	TFG (ECTs)	ECTs (h)	Annual Salary (€/year)	Hourly Wage (€/h)	Total Cost (€)
Geoinformatic Engineer	12	360	26.084,88 €	13,82 €	4.973,81 €
Total					**4.973,81 €**

Finally, we get a total labor cost of **4,973.81 €**.

6.4 Total Budget

This total budget has been divided into material execution budget and the execution budget by contract

6.4.1 Material Execution Budget

This Material Execution Budget has been calculated by adding the above costs. Table 22 illustrates the result.

Table 22: Total material execution budget.

Section	Total Cost (€)
Labor cost	4.973,81 €
Hardware cost	2.444,00 €
Software cost	78,02 €
Total	**7.495,83 €**

The material execution budget has result in **7,495.83 €**.

6.4.2 Execution budget by contract

This budget is the result of adding the overhead costs (13% of material execution budget) to the industrial benefit (which is usually equal to 6% in this projects) to the previously calculated material execution budget. Then, the VAT (Value-added-Tax) is applied (equal to 21% of the previous sum) (Vilarroig, 2019). This calculation is shown in figure 23.

Table 23: Total execution budget by contract

Components	Total Cost (€)
Material Execution Budget	7.495,83 €
Overhead costs	974,46 €
Benefits (15%)	1.124,37 €
Total Before Taxes	**9.594,67 €**
VAT Tax (21%)	2.014,88 €
Execution Budget by Contract	**11.609,55 €**

The execution budget by contract is equal to **11,609.55 €.**

7. CONCLUSIONS AND FUTURE LINES OF RESEARCH

The purpose of this project was to investigate the tile size images which enabled the models proposed to reach the highest levels of performance in both recognition and segmentation tasks, so that it can be used for future lines of research.

Starting with image recognition, 256x256 tiles gave the best results with the third model from VGG-v2. The model that performed with worse results in this size has been the third VGG-from-scratch, scoring the lowest results in every metrics (and reaching the highest loss score). Following with the 512x512 tile size, these images have turned to be easier to interpretate for the models, in terms of results at least. Its best model has been the third VGG-v2 despite VGG-v1 being a good alterative as well. On the contrary, the third VGG-from-scratch has thrown the lowest results at accuracy, precision, recall and F1-Score, not reaching 0.9 in any of them. Finally, in the 1024 it has been seen low losses and apparently high accuracies (especially in VGG-v2-1, which has been the best model in terms of results), but the results have been influenced by the unbalance of the classes. The second VGG-from scratch has given the lowest results in accuracy, despite none of the VGG-from-scratch being consistent.

The analysis of the confusion matrixes has been important to understand how each tile size's models were performing, as for the case of 1024, it has been clear that the models have struggled identifying tiles with the label "no-roads". In the case of the smallest tile size, the models have a trend of predicting more images as "no-roads" than they should. This has been deducted because of the False Negatives in the confusion matrix being higher in almost every case than the false positive. If there were no trend, it would confuse more or less the same number of tiles. Finally, the 512x512 confusion matrixes gave balanced results with the desired efficiency when detecting roads and images that did not contain roads and a similar number in the False positives and False negatives.

As far as semantic segmentation is concerned, the smallest tile size of images apparently worked better at first. The 256x256 best model has been Unet-InceptionResNetv2 reaching 0.56 in IoU in unseen data. On the contrary, Unet-SEResNeXt50-3 have barely reached 0.50 in IoU in both test and validation sets. Following with the next tile size (512x512), the third LinkNetEfficientNet-b5 has given the best results at IoU (reaching 0.4895 in IoU in the test set) and the lowest loss score in both testing and validation (0.62 and 0.63 respectively). When it comes to the worst results in this intermediate tile size, the model has been the second U-Net-InceptionResNetv2, with a high increase in the loss score (reaching 0.95 in test set). Finally, the model with the highest results has been the first U-Net-InceptionResNetv2 (0.35 in IoU and the lowest loss score with 1.1895).

Here, the lowest results have been obtained by the third LinkNet EfficientNet-b5 model, with just 0.275 in IoU in the test dataset, and 1.348 loss score.

Going further in segmentation, the final statement needed to be complemented by a visual point of view. In this visual approach, it can be said that the 256-tile size could lead the model into some mistakes as the ones seen in the dry areas or just the coastal picture, where the network wanted to identify more roads than it should. This could be due to the paths being a higher part of the image in terms of proportion of pixels. On the other hand, 1024 seems to be bigger than it should, as now the roads are too tiny to be differentiated in many cases, causing the network to omit these little branches. In addition, 256 seemed to be more vulnerable to occlusions or drastic change of color brightness in the layouts.

With that said, the 512-tile size images have demonstrated to fit the network training process, as it has adapted to most situations identifying from simple roads to complex networks of roads (like the urban example shown before).

As a conclusion, I could say with no doubts that the 512-tile size images are the ones I would use if I wanted to perform image recognition the best as possible, and, for image segmentation, I would give a higher importance to the visual statement (considering that the difference in results is not big in the 256 vs 512 decision) and hence, I reckon that 512 tile size is the best one to perform this task as well.

As far as the challenges solved is concerned, the most difficult one has been understanding why the 1024 VGG-from-scratch networks were giving that results (a much higher loss than the other architectures). Once it was shown that big part of the problem was due to an unbalance in the classes, it comes the moment of adjusting the hyperparameters of batch size and learning rate, because the 1024 tile size networks were not able to run with the previous' tile size batch as it exceeded the RAM memory from the GPUs. The batch size needed to be proportionally decreased as the tile size grew.

Another challenge which took me time to notice was understanding which parts of the roads or paths were more difficult to predict in semantic segmentation models and trying to predict myself in which situations a bad mask prediction could happen.

A future line of research would be trying to assemble the results of different tile size segmentations in an efficiently computational way, as it was proven during my studying that the three models obtained complement each other, but obviously it is not really effective to prepare a large-scale segmentation separated and then joining it.

REFERENCES

Albumentations Documentation—Transforms. (n.d.). Retrieved 23 June 2021, from https://albumentations.ai/docs/api_reference/augmentations/transforms/

Alzoubi36. (2021). *English: A convolution between a 5x5 matrix and a 2x2 kernel*. Own work. https://commons.wikimedia.org/wiki/File:Convolution.PNG

Anaconda (Python distribution). (2021). In *Wikipedia*. https://en.wikipedia.org/w/index.php?title=Anaconda_(Python_distribution)&oldid=1025136754

Binary crossentropy loss function | Peltarion Platform. (n.d.). Peltarion. Retrieved 1 July 2021, from https://peltarion.com/knowledge-center/documentation/modeling-view/build-an-ai-model/loss-functions/binary-crossentropy

Brownlee, J. (2017). A Gentle Introduction to Transfer Learning for Deep Learning. *Machine Learning Mastery*. https://machinelearningmastery.com/transfer-learning-for-deep-learning/

Brownlee, J. (2019). Understand the Impact of Learning Rate on Neural Network Performance. *Machine Learning Mastery*. https://machinelearningmastery.com/understand-the-dynamics-of-learning-rate-on-deep-learning-neural-networks/

Brownlee, J. (2021). Difference Between Backpropagation and Stochastic Gradient Descent. *Machine Learning Mastery*. https://machinelearningmastery.com/difference-between-backpropagation-and-stochastic-gradient-descent/

Chaurasia, A., & Culurciello, E. (2017). LinkNet: Exploiting Encoder Representations for Efficient Semantic Segmentation. *2017 IEEE Visual Communications and Image Processing (VCIP)*, 1–4. https://doi.org/10.1109/VCIP.2017.8305148

Chollet, F. (2018). *Deep learning with Python*. Manning Publications Co.

Cira, C.-I., Alcarria, R., Manso-Callejo, M.-Á., & Serradilla, F. (2020). A Framework Based on Nesting of Convolutional Neural Networks to Classify Secondary Roads in High Resolution Aerial Orthoimages. *Remote Sensing*, *12*(5), 765. https://doi.org/10.3390/rs12050765

CS231n Convolutional Neural Networks for Visual Recognition. (n.d.). Retrieved 14 April 2021, from https://cs231n.github.io/

CUDA Zone. (2017). NVIDIA Developer. https://developer.nvidia.com/cuda-zone

Donges, N. (2021). *Gradient Descent: An Introduction to 1 of Machine Learning's Most Popular Algorithms*. Built In. https://builtin.com/data-science/gradient-descent

Doshi, S. (2019). *Various Optimization Algorithms For Training Neural Network*. Medium. https://towardsdatascience.com/optimizers-for-training-neural-network-59450d71caf6

Furaha, D., & Ujjwal, K. (n.d.). *Convulational Neural Networks and simple Multi-Layer Percepton for Image classification*. Retrieved 26 August 2021, from https://furahadamien.com/papers/artificial_neural_nets.pdf

Gandhi, R. (2018). *Build Your Own Convolution Neural Network in 5 mins*. Medium. https://towardsdatascience.com/build-your-own-convolution-neural-network-in-5-mins-4217c2cf964f

Gao, H. (2017). *A Walk-through of AlexNet*. Medium. https://medium.com/@smallfishbigsea/a-walk-through-of-alexnet-6cbd137a5637

Godoy, D. (2018). *Understanding binary cross-entropy / log loss: A visual explanation*. Medium. https://towardsdatascience.com/understanding-binary-cross-entropy-log-loss-a-visual-explanation-a3ac6025181a

Gupta, D. (2019). *A Beginner's guide to Deep Learning based Semantic Segmentation using Keras*. How to Do Semantic Segmentation Using Deep Learning. https://divamgupta.com/image-segmentation/2019/06/06/deep-learning-semantic-segmentation-keras.html

He, K., Zhang, X., Ren, S., & Sun, J. (2015). Deep Residual Learning for Image Recognition. *ArXiv:1512.03385 [Cs]*. http://arxiv.org/abs/1512.03385

How U-net works? | *ArcGIS for Developers*. (n.d.). Retrieved 24 May 2021, from https://developers.arcgis.com/python/guide/how-unet-works/

Ibrahim, A. K. (2020). *English: Illustration of Concept of Transfer Learning*. https://asa.scitation.org/doi/full/10.1121/10.0001943. https://commons.wikimedia.org/wiki/File:Transfer_Learning.png

Ingargiola, A. (2015). *1. What is the Jupyter Notebook? —Jupyter/IPython Notebook Quick Start Guide 0.1 documentation*. https://jupyter-notebook-beginner-guide.readthedocs.io/en/latest/what_is_jupyter.html

Introduction to Loss Functions. (2018). *Algorithmia Blog*. https://algorithmia.com/blog/introduction-to-loss-functions

Jeong, J. (2019). *The Most Intuitive and Easiest Guide for CNN*. Medium. https://towardsdatascience.com/the-most-intuitive-and-easiest-guide-for-convolutional-neural-network-3607be47480

Jinapattanah. (2012). *File:Machine Learning Technique..JPG - Wikimedia Commons*. https://commons.wikimedia.org/wiki/File:Machine_Learning_Technique..JPG

Keras Team, K. (n.d.). *Keras documentation: Image data preprocessing*. Retrieved 24 June 2021, from https://keras.io/api/preprocessing/image/

Kongsilp, P. (2019). *CNN: Step 2 — Flattening*. Medium. https://medium.com/@PK_KwanG/cnn-step-2-flattening-50ee0af42e3e

Krizhevsky, A., Sutskever, I., & Hinton, G. E. (2017). ImageNet classification with deep convolutional neural networks. *Communications of the ACM*, *60*(6), 84–90. https://doi.org/10.1145/3065386

Le, J. (2021). *How to do Semantic Segmentation using Deep learning*. AI & Machine Learning Blog. https://nanonets.com/blog/how-to-do-semantic-segmentation-using-deep-learning/

Learning rate. (2021). In *Wikipedia*. https://en.wikipedia.org/w/index.php?title=Learning_rate&oldid=1029664939

Li, Z., Kamnitsas, K., & Glocker, B. (2021). Analyzing Overfitting under Class Imbalance in Neural Networks for Image Segmentation. *IEEE Transactions on Medical Imaging*, *40*(3), 1065–1077. https://doi.org/10.1109/TMI.2020.3046692

Long, J., Shelhamer, E., & Darrell, T. (2014). *Fully Convolutional Networks for Semantic Segmentation*. 10.

Ltd, C. (2009). *English: This picture shows the internal structure of an Artificial Neural Network (ANN) used to recognize landmarks in a camera image.* http://www.cyberbotics.com. https://commons.wikimedia.org/wiki/File:Artificial_neural_network_image_recognition.png

Lukas Getautis. (2021). *211—U-Net vs LinkNet for multiclass semantic segmentation*. https://www.youtube.com/watch?v=NUvmHYTQxrs

MacGregor, A. (2017). *Neural Networks Without a PhD: Components of a Neural Network | Hacker Noon*. https://hackernoon.com/neural-networks-without-a-phd-components-of-a-neural-network-9d98b056995b

Maier, A. (2019). *English: Figure 6 from Andreas Maier, Christopher Syben. Tobias Lasser. Christian Riess. 'A gentle introduction to deep learning in medical image processing'. Zeitschrift für Medizinische Physik*. https://www.sciencedirect.com/science/article/pii/S093938891830120X#fig0010. https://commons.wikimedia.org/wiki/File:ConvolutionAndPooling.svg

Manso-Callejo, M. A., Cira, C.-I., Alcarria, R., & Gonzalez Matesanz, F. J. (2021). First dataset of wind turbine data created at national level with deep learning techniques from aerial orthophotographs with a spatial resolution of 0.5 m/pixel. *IEEE Journal of Selected Topics in Applied Earth Observations and Remote Sensing*, 1–1. https://doi.org/10.1109/JSTARS.2021.3101934

Papers with Code—SE ResNet. (n.d.). Retrieved 30 June 2021, from https://paperswithcode.com/model/se-resnet?variant=seresnet50

Ph.D Kızrak, A. K. (2019). *Comparison of Activation Functions for Deep Neural Networks*. Medium. https://towardsdatascience.com/comparison-of-activation-functions-for-deep-neural-networks-706ac4284c8a

Ronneberger, O., Fischer, P., & Brox, T. (2015). U-Net: Convolutional Networks for Biomedical Image Segmentation. *ArXiv:1505.04597 [Cs]*. http://arxiv.org/abs/1505.04597

Rosebrock, A. (2016). Intersection over Union (IoU) for object detection. *PyImageSearch*. https://www.pyimagesearch.com/2016/11/07/intersection-over-union-iou-for-object-detection/

Simonyan, K., & Zisserman, A. (2015a). Very Deep Convolutional Networks for Large-Scale Image Recognition. *ArXiv:1409.1556 [Cs]*. http://arxiv.org/abs/1409.1556

Simonyan, K., & Zisserman, A. (2015b). Very Deep Convolutional Networks for Large-Scale Image Recognition. *ArXiv:1409.1556 [Cs]*. http://arxiv.org/abs/1409.1556

Simplilearn. (2021). *What is Tensorflow: Deep Learning Libraries and Program Elements Explained*. Simplilearn.Com. https://www.simplilearn.com/tutorials/deep-learning-tutorial/what-is-tensorflow

Song, H., Kim, M., Park, D., Shin, Y., & Lee, J.-G. (2021). Learning from Noisy Labels with Deep Neural Networks: A Survey. *ArXiv:2007.08199 [Cs, Stat]*. http://arxiv.org/abs/2007.08199

Stanford University School of Engineering. (2017). *Lecture 11 | Detection and Segmentation*.

https://www.youtube.com/watch?v=nDPWywWRIRo&list=PL3FW7Lu3i5JvHM8ljYj-zLfQRF3EO8sYv&index=12&t=775s&ab_channel=StanfordUniversitySchoolofEngineering

Syed, A., & Morris, B. T. (2019). SSeg-LSTM: Semantic Scene Segmentation for Trajectory Prediction. *2019 IEEE Intelligent Vehicles Symposium (IV)*, 2504–2509. https://doi.org/10.1109/IVS.2019.8813801

Szegedy, C., Ioffe, S., Vanhoucke, V., & Alemi, A. (2016). Inception-v4, Inception-ResNet and the Impact of Residual Connections on Learning. *ArXiv:1602.07261 [Cs]*. http://arxiv.org/abs/1602.07261

Szegedy, C., Wei Liu, Yangqing Jia, Sermanet, P., Reed, S., Anguelov, D., Erhan, D., Vanhoucke, V., & Rabinovich, A. (2015). Going deeper with convolutions. *2015 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 1–9. https://doi.org/10.1109/CVPR.2015.7298594

Tan, M., & Le, Q. V. (2020). EfficientNet: Rethinking Model Scaling for Convolutional Neural Networks. *ArXiv:1905.11946 [Cs, Stat]*. http://arxiv.org/abs/1905.11946

Team, K. (n.d.-a). *Keras documentation: Callbacks API*. Retrieved 26 May 2021, from https://keras.io/api/callbacks/

Team, K. (n.d.-b). *Keras documentation: Why choose Keras?* Retrieved 22 August 2021, from https://keras.io/why_keras/

TensorFlow. (n.d.). TensorFlow. Retrieved 26 July 2021, from https://www.tensorflow.org/

Tensorflow/tpu. (2021). [Jupyter Notebook]. tensorflow. https://github.com/tensorflow/tpu (Original work published 2017)

Tsang, S.-H. (2018a, August 22). *Review: VGGNet — 1st Runner-Up (Image Classification), Winner (Localization) in ILSVRC 2014*. Medium. https://medium.com/coinmonks/paper-review-of-vggnet-1st-runner-up-of-ilsvlc-2014-image-classification-d02355543a11

Tsang, S.-H. (2018b, August 24). *Review: GoogLeNet (Inception v1)— Winner of ILSVRC 2014 (Image Classification)*. Medium. https://medium.com/coinmonks/paper-review-of-googlenet-inception-v1-winner-of-ilsvlc-2014-image-classification-c2b3565a64e7

Tsang, S.-H. (2018c, October 5). *Review: FCN — Fully Convolutional Network (Semantic Segmentation)*. Medium. https://towardsdatascience.com/review-fcn-semantic-segmentation-eb8c9b50d2d1

Vilarroig, C. A. (2019). *Diseño y desarrollo de un método de superresolución en imágenes de RM usando Deep Learning*. 106.

Visus, A. (2020). *¿Para qué sirve Python? Razones para utilizarlo | ESIC*. https://www.esic.edu/rethink/tecnologia/para-que-sirve-python

Wei, J. (2019a). *VGG Neural Networks: The Next Step After AlexNet*. Medium. https://towardsdatascience.com/vgg-neural-networks-the-next-step-after-alexnet-3f91fa9ffe2c

Wei, J. (2019b, July 4). *VGG Neural Networks: The Next Step After AlexNet*. Medium. https://towardsdatascience.com/vgg-neural-networks-the-next-step-after-alexnet-3f91fa9ffe2c

What is the difference between Deep Learning and Machine Learning? ⋆ Quantdare. (2019, August 1). *Quantdare*. https://quantdare.com/what-is-the-difference-between-deep-learning-and-machine-learning/

World Circuit Records. (2017, November 3). *¿Qué es la retropropagación y qué hace en realidad? | Aprendizaje profundo, Capítulo 3.* https://www.youtube.com/watch?v=Ilg3gGewQ5U&ab_channel=3Blue1Brown

Yazdani, M. (2019). *English: This is an example architecture of U-Net for producing k 256-by-256 image masks for a 256-by-256 RGB image.* Own work. https://commons.wikimedia.org/wiki/File:Example_architecture_of_U-Net_for_producing_k_256-by-256_image_masks_for_a_256-by-256_RGB_image.png