Xavier Pell Vidal

Image geopositional location in mobile phones using neural networks

Final Master's Project
directed by Dr. Jordi Duch

MESIIA

Tarragona

2018

Abstract

Deep learning in image processing and classification tasks has evolved exponentially in the last few years thanks to new advances and outperforms in existing neural networks. These networks can be trained with labelled images in order to be subsequently used to identify and classify sets of objects in it.
Moreover, some libraries have been recently developed to be executed in mobile phones, so it is possible to use the phone camera to capture an image in real time, classify it with a trained neural network in order to obtain specific information on what you are seeing through the camera.

In this thesis, we take a quick view to some convolutional neural networks that in conjunction with a deep-learning mobile framework let us to develop a prototype that would be able to geolocate users in a space using phone camera. This process consists in applying transfer learning to the pre-trained model MobileNet, to modify the output of the network in order to classify five objects landmarks that are in a geolocation area. This re-trained network is converted to a more light-weight version to allow its execution in a developed mobile application written in Android.

Finally, we evaluate the final data to assure the proper function of the system, the accuracy of the classification process and the geolocation information that is extracted of the final project prototype.

Acknowledgements

First, I would like to thank my advisor Dr. Jordi Duch for their guidance, support and help over the large course of this Master thesis. It has been a long way that would have not be possible without him. I would also like to thank all the professors involved in the MEISSIA master that have make this thesis possible.

Finally, I would like to express my deep gratitude to my fiancée who has help me in the data collection and some evaluation parts of the project when I was physically uncapable to do it.

Contents

List of Figures

1. Introduction

Image-based Geo-localization has been a growing issue that challenged computering vision and deep learning. It is defined as the capability of knowing where a photo was space-taken just having the photo itself. Although it is largely know that Image-based geo-localization is close related to Computer vision, the challenge is to merge it with deep-learning in order to increase and facilitate classification tasks to eventually achieve a better performance of some applications.

Otherwise, the field of neural networks for image processing and classification has grown exponentially in the last few years mainly as a result of the discovery of deep neural networks that are capable of classifying images with very high accuracy[1, 2].
The neural networks consist in training the networks with labelled original images which afterwards will be used to identify the likelihood that another image is similar to the original one. In this regard, it has been recently developed the capability of execute such neural networks in mobile phones using new light-weight libraries developed for mobile systems [3]. This advance offers the advantage of obtaining real time image information of a captured image through the camera using the described trained neural networks.

Many research groups have focused on solving the Image-based Geo-localization challenge using static images [4]. In this master thesis, we intend to go one-step forward using the potential of the phone camera to obtain geopositional information of the phone using the video feed.

1.1 Aim

The aim of this project is to evaluate and use the potential of deep learning methods, specifically convolutional neural networks, to identify and use objects to obtain an approximate location of the users. We will have to focus on different parts of the process, starting from the data collection going through the creation and training of the network and finally creating a small Android app that will execute such trained network. Efficiency will also be evaluated throughout the process.

In order to achieve our final goal, we have divided this work in the following objectives:

- Study and explore the different types of pre-trained networks available in TensorFlow and select the most appropriate one for the study. Take in account TensorFlow Lite and TOCO limitations.

- Parametrize and test the network in a mobile phone application to be able to classify some set of custom objects.

- Capture input images of object landmarks of an extensive space and train the network to be able to classify these landmarks.

- Evaluate the prototype, the accuracy and the efficiency of the mobile app developed with the trained network in order to provide both an estimated location and a classification of the landmark objects.

1.2 Thesis structure

Section 2 describes the background necessary to understand the core of this thesis. We will review existing neural networks used nowadays as well as a brief explanation of TensorFlow and libraries needed to apply these neural networks in a mobile environment.

In Section 3 we review the currently state-of-the-art related to object location using convolutional neural networks applied to embedded environments, specially mobile phones.

Section 4 is related to the prototype development of this project. This part covers the pre-research work, the scheme of the prototype, the neural network used to classify the set objects in space and finally the android app developed for this purpose.

In Section 5 we evaluate the entire project, a study of the performance and the different object classifications types in the geolocalitzation area where project is developed

In Section 6 we report on the conclusions; not only the benefits from this thesis but also its limitations. We suggest different possibilities or ideas to apply to the existing prototype in order to outperform and improve its results.

2. Background

Background information described here is fundamental to understand the start point and further development of this project. We go through some of the most important convolutional neural networks that exists nowadays which could be potentially used in the TensorFlow.

We see a brief introduction and definition of TensorFlow, TensorFlow for mobile and embedded devices and Machine learning in android, especially NNAPI (Neural Networks API). They are the framework that let us integrate a neural network into an Android application to be able to classify and implement our prototype.

2.1 Convolutional Neural networks

Convolutional networks are networks with millions of parameters and lots of hidden layers. Some of the most important and most used networks are AlexNet, MobileNet, VGG, Inception, and ResNet. We will introduce some of these networks during this section.

In order to understand the design of Convolutional neural networks, these two parameters are very important:

- Accuracy**:**

  In every system, it is very important that it must be as accurate as possible. It's said that "accuracy not only depends on the network but also on the amount of data available for training". So for this reason and as we mention before, these networks are always trained with ImageNet dataset that's has more than a million of training images and it's one of the most extensive object image datasets.

- Computation:

  Convolutional networks have huge memory capacity and computation requirements, in special in the training process of the network. Moreover, the size of the final trained model becomes important if we pretend to use these networks in a mobile environment like a smartphone. In order to achieve more accuracy, more computation power is needed. For this reason, there is always a trade-off between accuracy and computation.

Image classification is the process of classify a given image into one of the predefined classes that the network is trained initially. The most common architecture for image classification is divided in 2 processes: feature extraction and classification.

The Hidden layers/Feature extraction is where the network will do some convolutions and pooling operations to see which features are extracted.

The Classification part is related to de fully connected layers that carry out the organizations and decides the probability of the objects to be one of the trained ones.

The main layers of a CNN and its more important parts are described below:

- Convolution layer

Convolutions are the main blocks of a Convolutional networks. This term refers to a combination of two functions to create, as a result a third function. In these networks, convolution operations are applied to the input data and a filter or kernel. Then, the result is a feature map of the image.

These operations (figure 1) are applied in a three dimensional matrix, with, height and depth.

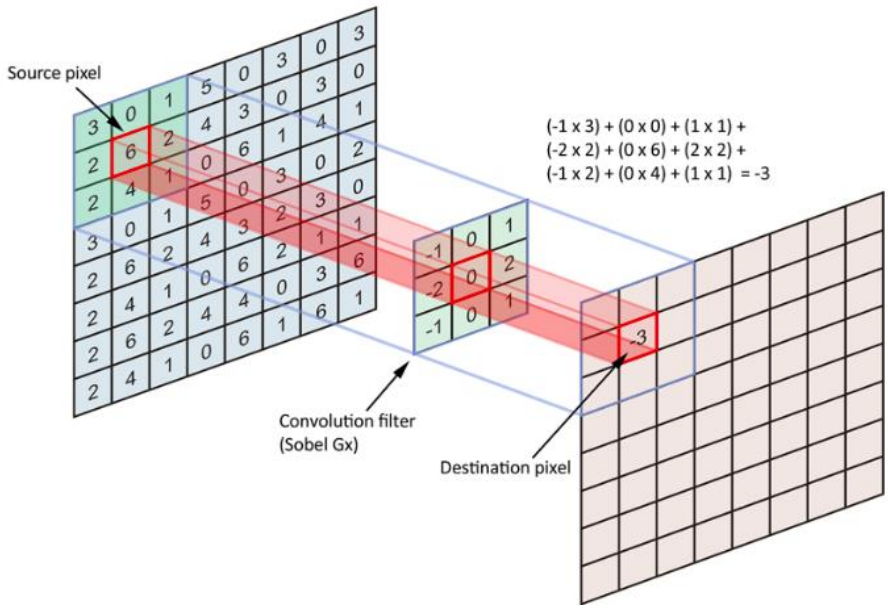


Figure 1: Application of the convolution operation in an image

These convolution operations are applied in the input, each operation uses a different filter. These different operations gives different feature maps that at the end, all these maps were put together to give the final output of the convolution layer of the network.

- Pooling layer

This is a layer that is commonly added between convolutional neural network layers. Its function is to reduce de size of the layers to be able to reduce de number of parameters to be computed. This shortens the training time and controls the overfitting.

The most common polling layer applied to ConvNets is max pooling (figure 2). It consists in take the highest value in each window, window size needs to be specified. This operation reduces feature map size and try, at the same time, to keep the significant data of the feature map

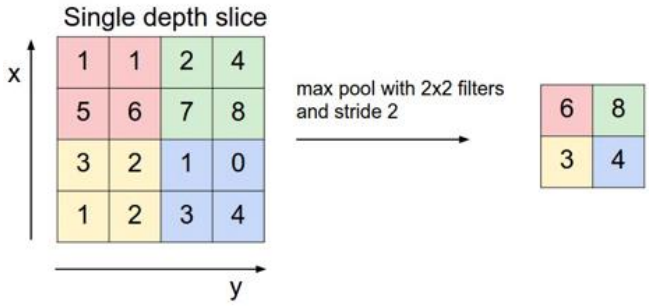


Figure 2: Max pooling gets the highest value of the feature map

- Activation function

As we know also for any other neural networks, an activate function is required to convert the output to a one non-linear. In these type of networks, the output of the convolutional layer is passed to the activation function as the input. There are lots of activation functions like sigmoid, ReLU, etc.

Other important parameters to be set in a ConvNet are those called hyperparameters:

1. Kernel size applied
2. Filter count, that corresponds to the number of filters to use
3. Stride, that are how big are the steps of the filters applied
4. Padding

- Classification

After all convolution and pooling layers we can find few fully connected layers. These layers only accepts 1 dimension data so data need to be converted from 3 dimension to 1 dimension. These fully connected layers have full connections to all the activations of the previous layer.

- Training

The training part is also the same that in a regular neural network. It is achieved normally with backpropagation or gradient descent, but the fact that this ConvNet uses convolutional operations, the training processes are much more complex and need much more computational power.

2.1.1 Transfer Learning

Transfer learning, as described by the authors in [12] is, “Transfer learning is the improvement of learning in a new task through the transfer of knowledge from a related task that has already been learned”. The development of algorithms that facilitate transfer learning is increasing in the machine-learning scenario.

Focused in deep neural networks, they are described as the reutilization of a pre-trained model into a new problem that needs other classification data. For example, if we have a pre-trained model that classifies types of sports balls, we could use the knowledge that this classifier gained during the training process, to change it and be able to recognize other type of objects, like types of candies or other types of objects. The general idea is that transfer learning is used when we don’t have lots of training data labelled to be able to train our model, and the pre-trained model learned knowledge from a data set that has lots of data labelled, as we exposed before, ImageNet is a good example of a dataset to train this neural network models.

The main benefits of transfer learning are:

- Less time to train the new neural network. Can take days or weeks to train a neural network from scratch on a complex classifying problem.

- They don’t need too much data to re-train because pre-trained model is already trained with a big dataset.

2.1.2 AlexNet

This network was one of the first to increment considerably the classification accuracy on the ImageNet dataset in comparison to the previous traditional methodologies and networks. In Figure 3 we can see the layer architecture, 5 convolutional layers followed by a max pooling layer and 3 fully connected layers.

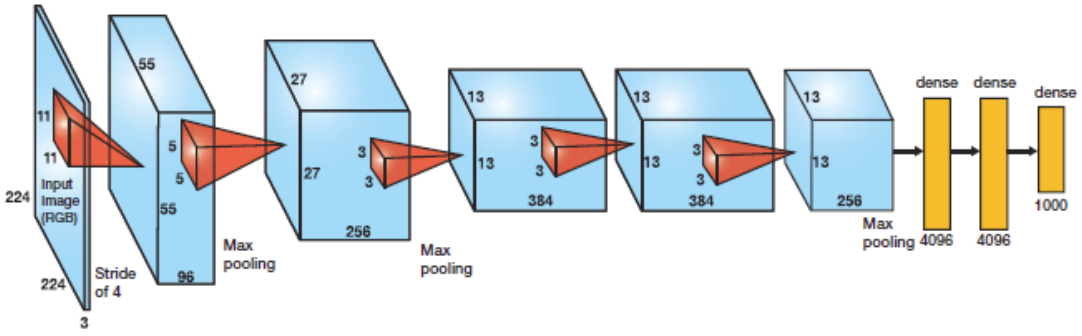


Figure 3: AlexNet layer architecture

AlexNet, proposed by Alex Krizhevsky [1], uses ReLu (Rectified Linear Unit) for the non-linear part, instead of a Tanh or Sigmoid function which was the earlier standard for traditional neural networks.

Another problem that this architecture solved was reducing the over-fitting by using a dropout layer after every fully connected layer. Dropout layer has a probability associated with it and is applied to every neuron of the response map in a separate way. Figure 4 shows that dropout randomly switches off the activation neurons with a certain probability determined.

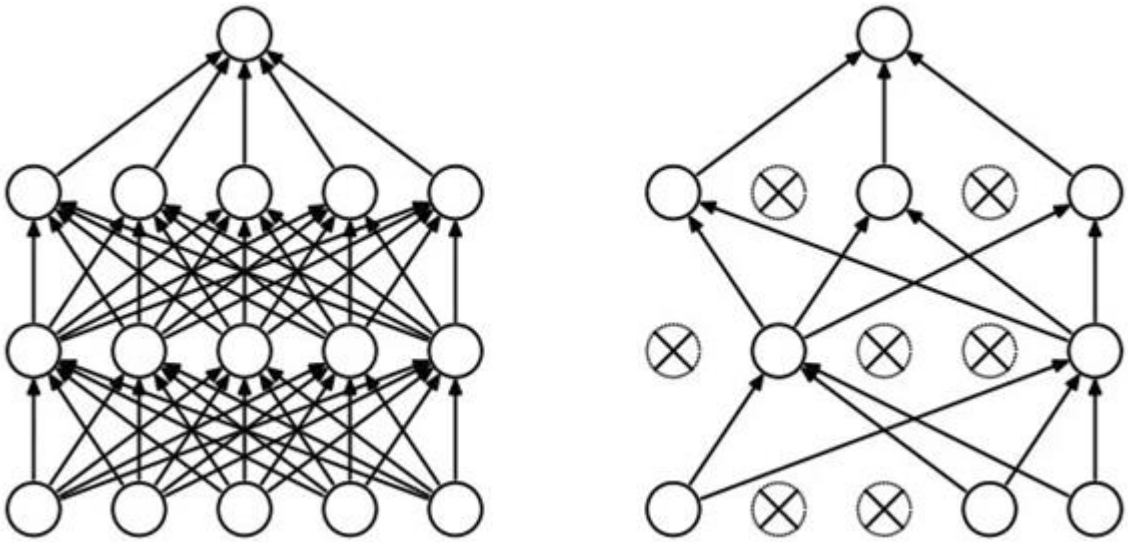


Figure 4: Applying dropout to a standard neural network

- Dropout

The idea behind the dropout [13] is similar to the model. Different combinations of switched off neurons (representing different architectures) generates new network architectures. All these different architectures are trained in parallel mode with a weight applied to every subset.

For n neurons attached to dropout, the number of architectures formed is 2^n. This gives a new model that let to avoid over-fitting problems.

2.1.3 MobileNet

MobileNet [14] is a convolutional neural network that is based on depth-wise separable convolutions. This convolutions make the network more light weight. The

authors introduces two global hyper-parameters that trade-off between latency and accuracy.

2.1.3.1 Depthwise separable convolutions vs normal convolutions

A regular convolutional layer applies a convolution kernel (or "filter") to all channels of the input image. This kernel or filter is slide across the image and at each step performs a weighted sum of the input pixels.

The important thing is that the "normal" convolution operation combines the values of all the input channels (we have 3 channels in a RGB). If the image has 3 input channels, if we run a simple convolution kernel across this image, the result is an output image with only 1 channel per pixel (see figure 5).

So for each input pixel, from 1 to n channels, the convolution results in a new output pixel with only a single channel.

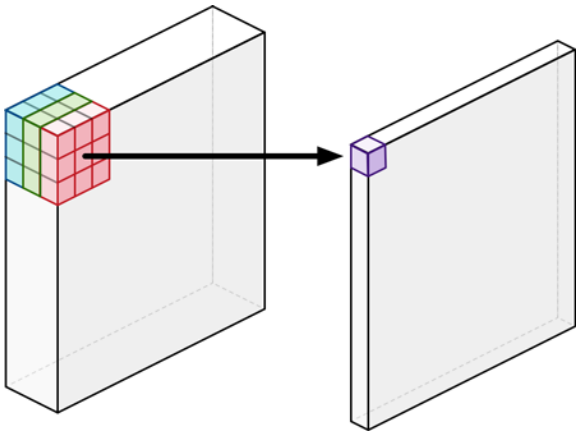


Figure 5: Normal convolution

The MobileNets architecture also uses this "normal" convolution, but only one time in the first layer of the network. In all other layers a depthwise separable convolution is applied.

These depthwise separable convolutions is actually a combination of two different convolution operations: a depthwise convolution and a pointwise convolution. Figure 6 shows a depthwise convolution and figure 7 shows a pointwise convolution.

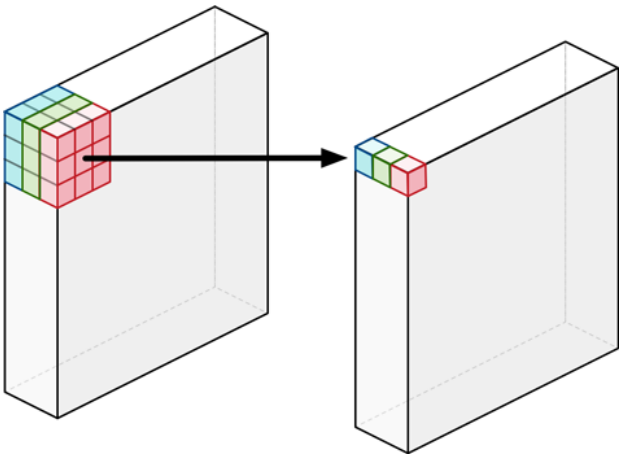


Figure 6: Depthwise convolution

Unlike a regular convolution, it does not combine all the input channels, a convolution is applied one for each channel. For an image with 3 RGB channels per pixel, a depthwise convolution generates an output image that has 3 channels (figure 6). Each channel have its own weights.

The main purpose of this type of convolutions is to filter input channels to try to extract image features like colour filtering, edge detection, etc.

In this type of convolution, a depthwise convolution is followed by another convolution called pointwise convolution (figure 7).

Depthwise convolution is followed by a pointwise convolution. In other words, this is always the same as a regular convolution with a kernel (or filter) size of 1x1 (also represented in figure 7).

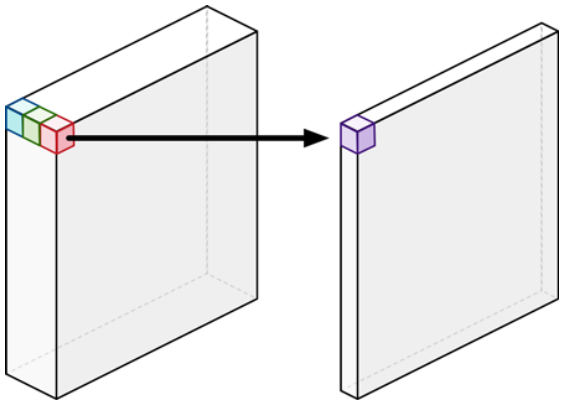


Figure 7: Pointwise convolution

In other words, this is simply a summation of the channels, more or less, as a weighted sum. As with a regular convolution, we usually stack together many of these pointwise kernels to create an output image with many channels. The purpose of this pointwise convolution is to combine the output channels of the depthwise convolution to create new features.

Putting these two type of convolutions together, the resultant convolution is called depthwise separable convolution. Regular convolutions does these 2 steps in the same operation, but this depthwise separable convolution do these two steps in a separated mode.

The results obtained by these 2 kind of convolution are pretty similar, they both filter data and generates image features but the "normal" one has significantly more computational cost and it need to learn more weights than in the depthwise separable convolution.

So even though these 2 convolutions do the same thing, the depthwise separable convolution will need less computational costs than the regular convolution and as a result, it will be much faster than the regular convolution.

Authors in the MobileNet article shows the formula to calculate the speed of these convolutions and as a result of this formula, this depthwise convolution are about to 9 times faster than the regular convolutions. And, what is more important, is also as effective as the normal ones. So it's normal that MobileNets uses 13 of these new convolutions only in a row as we will see when the full architecture was presented.

2.1.3.2 MobileNet V1

The main idea in MobileNet V1 networks is that convolutional layers, that have a very high computational cost, can be replaced with these depthwise separable convolutions explained before.

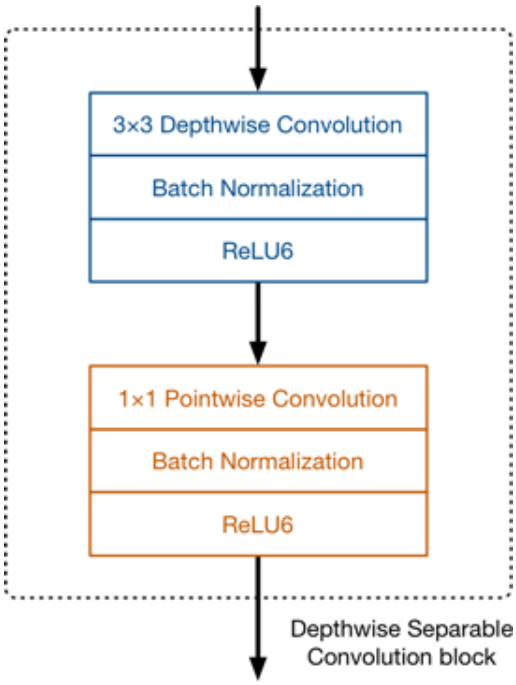


Figure 8: Depthwise separable convolution block

The full architecture of MobileNet V1 consists, at the initial layer, of a normal 3×3 convolution, followed by 13 depthwise separable convolution blocks (figure 8).

We don't find pooling layers between these depthwise separable blocks. Instead, some depthwise layers have a stride of 2 to decrement the data spatial dimensions. When that happens, next pointwise layer doubles the number of output channels. If the input image is 224×224×3 (like our prototype input images) the resulting feature map is of size 7×7×1024.

Often, convolution layers are followed by a batch normalization, after this normalization, a ReLU6 activation function is applied. It's like the "classical" ReLU6 but this previous batch normalization regulates activations size to not being too big.

The authors of the MobileNet found that ReLU6 is better than a classical ReLU when performing low-precision computations so this was the activation function that they used.

In every classifier based on MobileNet has global average pooling layer at the end, followed by a fully-connected classification layer and a softmax.

Actually, there are lots of MobileNet versions that derivate from the original. It MobileNet was designed to be a family of neural network architectures. There are some hyper parameters that let play with different architecture configurations.

The most important of these hyper parameters is the depth multiplier also named "width multiplier". This parameter determines and regulates the number of channels that will have every layer in the network. Using a depth multiplier of 0.5 will divide the number of channels that are used in each layer, and this makes to reduce the number of operations by a factor of 4 and the number of parameters by a factor of. As a result, this will give a

less accurate model but also the computational cost of the network will be drastically improved. It will be much faster.

Thanks to these depthwise separable convolutions introduced by authors, MobileNet are up to 9 times more faster than other neural networks that has the same accuracy as MobileNet. This speed improvement makes the model more suitable to be executed in environments that have less computational power, and obviously, in a real-time mode with good results.

2.1.3.3 MobileNet V2

MobileNet V2 [15] also uses depthwise separable convolutions, but the main building block has some changes. Figure 9 shows the new improved building block:

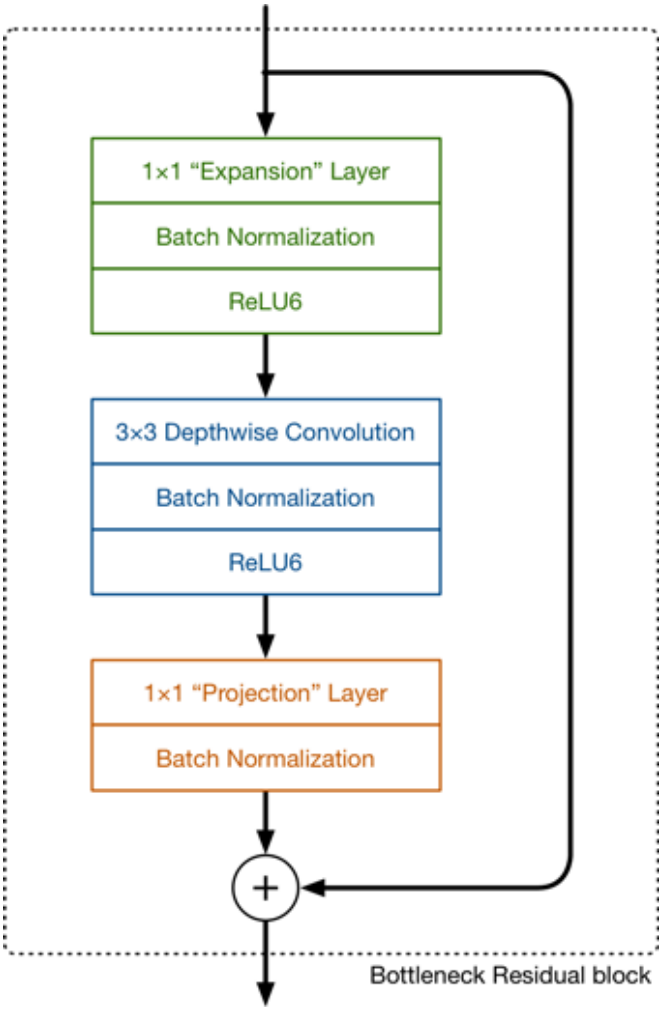


Figure 9: MobileNet building block

This time there are three convolutional layers in the block. The last two are the ones we already know: a depthwise convolution that filters the inputs, followed by a 1×1 pointwise convolution layer. However, this 1×1 layer now has a different job.

This improved block, now has 3 convolutional layers rather than the 2 of the previous MobileNet. The last 2 layers are the same as the MobileNet V1, a depthwise convolution followed by 1x1 pointwise convolution, but now, this last layer does not do the same job as before.

In V1 the pointwise convolution doubled the number of channels or maintain them. In V2 the function is different: it reduces the number of channels. This reduction give the name to the layer, "Projection layer" (it projects input data that have lots of channels to output data that have less channels). This project layer is also called a bottleneck layer because it reduces the amount of data that goes through the network. Also this bottleneck give the name to the entire block because the output of the bloc is a bottleneck.

The new layer introduced in the bloc is the "Expansion" layer, a 1x1 convolution. Its function is to expand data channels before going to the depthwise convolutional layer for this reason this block always have more output channels than input ones. This expansion of the channels data is given by the expansion factor that is one of the hyperparameters that can be set in this network.

As in the V1 version, every layer has a batch normalization followed by a ReLU6 activation function. The main difference is found in the last "projection" layer where we don't have an activation function. This is because this last layer, as commented above, reduces the number of data channels, so the output data is a low-dimensional data. Authors see that using a non-linearity activation function after the batch normalization will destroy useful data across blocks.

The full MobileNet V2 architecture, is formed by 17 bottleneck residual blocks in each row. This is followed by a regular 1×1 convolution, a global average pooling layer, and a final classification layer. But the first block is a bit different, it doesn't have an "expansion layer", it has a classical convolutional bloc of 3x3 with 32 channels.

2.1.4 GoogleNet / Inception

We found that some types of convolutional neural networks achieves high accuracy on ImageNet dataset classification. But its deployments is very difficult to be done in most devices due to its high computational requirements, high memory usage and time costs. These costs are related to the width of the convolutional layers and the compute of the convolution operations.

So GoogLeNet [2] authors created a new module called inception (shown in the figure 10). Only a small number of neurons are effective and the number of the convolutional filters of a particular kernel size is small. Another special thing is that it uses convolution filters of different sizes to be able to capture features in different scales.

Another new feature in this module is the bottleneck layer (in figure 10 represents the 1x1 convolutions) that reduces significantly the computational cost of the entire module as explained above. This layer reduces the number of features and, as a result, the number of operations. The data passed to the convolutional modules can be reduced by up to 4 times, and this makes a big success in computational costs

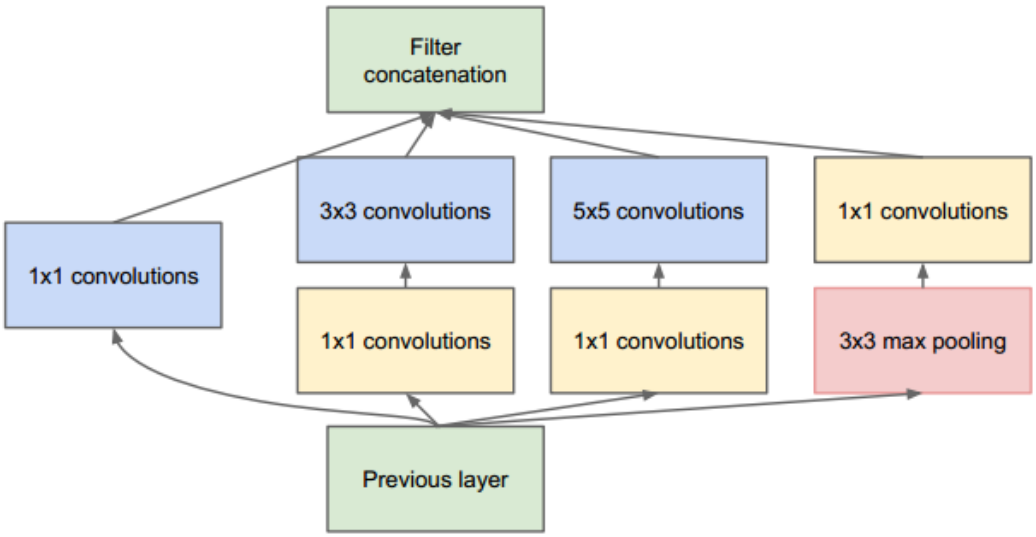


Figure 10: Inception module

Another change in these network is the replacement of the fully-connected layers at the end of the modules with a max pooling average layer at the end of the last convolutional layer. This averages the channel outputs to reduce the total number of parameters. As an example, take in count that AlexNet parameters are mostly in these fully connected layers (90% approx.), so the depth and the width of the GoogleNet network allow authors to remove these layers without affecting the high accuracy if these networks.

2.1 TensorFlow

TensorFlow [16] is an open source library developed by Google. This library is for numerical computations that requires high computational power. Its architecture let user to deploy these numerical computations in an easy way over different architectures and platforms. These architectures can go from CPU's to GPU's or TPU's. Also can be deployed in desktops, in the cloud, on mobile devices and in other embedded systems. This open source library have a strong support for machine learning purposes, but is also

used for other type of scientific disciplines like data scientists, predictive modellers and so on.

Its name come from multidimensional arrays used in neural network layers. These arrays are called tensors.

Before TensorFlow, Google developed DistBelief, a closed framework that the company uses to internal deep learning processes based on very large neural networks. After that, TensorFlow was released in 2017 (v1.0.0) as an open source framework that differs from DistBelief in many ways. TensorFlow is designed to be used by all user, not only Google Company as DistBelief. Also, it is more portable and is not only focused in neural networks like DistBelief.

2.1.1 TensorFlow Mobile

TensorFlow Mobile is a library designed to be used in mobile platforms like Android or iOS. Its purpose is to convert a TensorFlow normal model into one that fits better into a mobile platform. This library is also for those models that can't be converted to TensorFlow Lite due to its lack of supported operations.

Deep learning is often associated with high powered machines and big clusters of data. Sending this data to mobile devices through the network can be so expensive in time and network costs. For this reason TensorFlow Mobile give the opportunity to run these neural networks locally in mobile devices making this more fast and interactive.

2.1.2 TensorFlow Lite

TensorFlow Lite is the lightweight version of TensorFlow Mobile which is specifically designed for the mobile platform and embedded devices. It provides machine learning solution to mobile with low latency and small binary size.

TensorFlow supports set of core operators which have been adapted and optimized for mobile platforms. It also supports to develop custom operations in models.

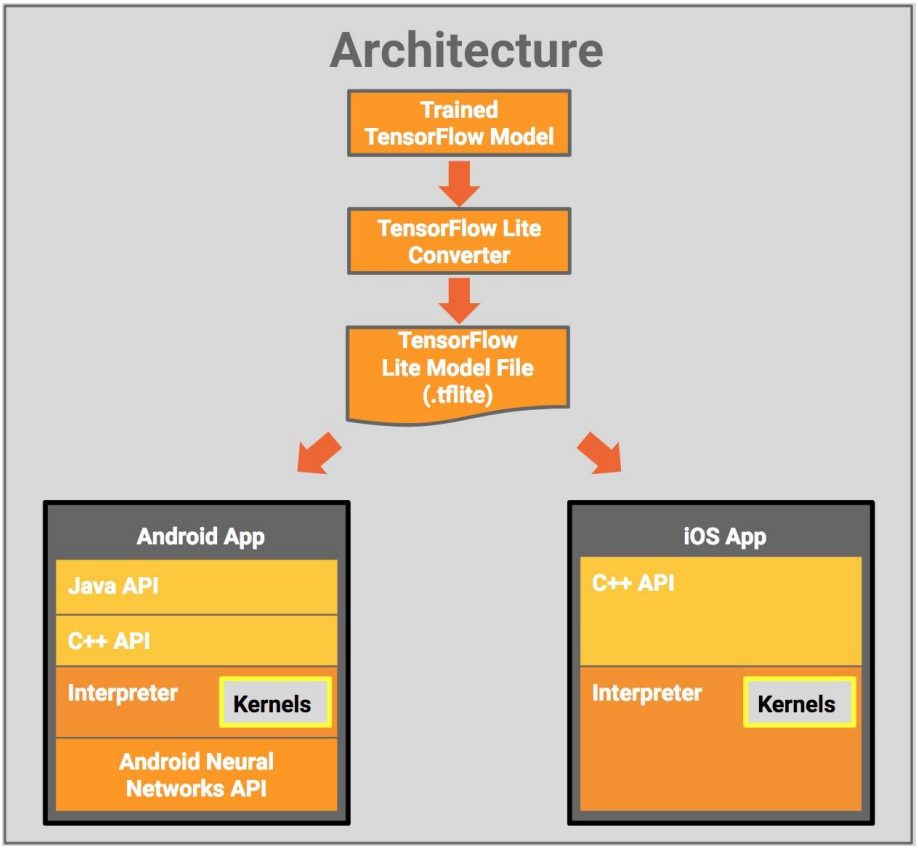


Figure 11: TensorFlow Lite architecture

The diagram in Figure 11 shows TensorFlow Lite architecture. The trained TensorFlow model on the disk are converted into TensorFlow Lite file format (.tflite) using TOCO (TensorFlow Lite converter). This new file can be integrated in mobile platforms in an easy way. On some new devices in Android, the interpreter uses the new Neural Networks API to be able to be executed with hardware acceleration.

Interpreter in TensorFlow Lite is significantly smaller than TensorFlow Mobile, because of the kernel can be loaded in a selective way.

2.1.3 TensorFlow Mobile vs Lite

We could find some differences between these two libraries:

- TensorFlow Lite is an evolution of TensorFlow Mobile. Apps developed in the evolved version of TensorFlow Mobile have a smaller binary size and the performance is better in most cases.

- TensorFlow Mobile supports more operators than TensorFlow Lite by default but also doesn't support all network models available. This lack of operators in

TensorFlow Mobile can be solved creating a new kernel that cover the needed function. But there are use cases that TensorFlow Lite can't support in actual version, in these cases it's only be possible to use TensorFlow Mobile until new versions of TensorFlow Lite will be released.

- TensorFlow Lite can take advantage of hardware acceleration if it exists in the deployed platform. Also of any machine learning library (like NNAPI in Android). In addition, it has fewer dependences than TensorFlow Mobile so is more suitable to be integrated in much more types of embedded systems and scenarios.

2.2 Machine learning on Android

Machine learning works in how the data is provided to the user, it is becoming an essential tool in App development nowadays.

Google presents the ML Kit (Machine Learning Kit) at its I/O developer conference. This Kit is a software development kit (SDK) for mobile platform apps like IOS and Android.

This kit let users to integrate some models pre-developed by Google in an easy way to their apps. These models are pre-defined and all users can use it without being expertise in machine learning and neural networks.

But referring to a more low-level environment, and more in touch with our project, exists an API for only neural networks on Android. The Android Neural Networks API (NNAPI) [17] is an Android C API designed to execute high computational cost operations for machine learning on mobile devices.

This API provides a base to be used by other higher-level frameworks, such as TensorFlow Lite, Cafe2, etc. and it's only available in Android 8.1 or higher.

NNAPI supports inferencing by applying data from Android devices to models that are pre-trained. These types of inferencing could be image classification, predictions in user behaviour, serving responses to a query, etc.

This inferencing has some benefits:

- Latency: It is not needed send requests and receive response to use a neural network. Latency is very small. This can be very useful when classifying objects in real time, such as classifying images of the video feed like our prototype.

- Availability: We don't need to have network connection to be able to use the application. The application can be always running.

- Speed: New hardware that is specific designed to neural networks make the computation much faster that only using CPU.

- Privacy: All data used in the app don’t need to be sent through the network so I remains in the device.

- Cost: No cloud server is needed to perform the operations or classifications, so all is computed locally.

TensorFlow Lite takes advantage of this neural network API provided by Google. If the Android version of the smartphone is high enough to have this API (8.1), classification time costs will be improved in a significant way.

3. State of the art

As mentioned before, object detection in a computer vision and deep learning area are constantly growing nowadays. This could be attributed to the advantages of deep learning and the increasing computation power and storage of embedded devices, especially mobile phones that let these networks to be deployed in these environments.

Every year new algorithms and improved versions of existing ones are outperforming previous models and older architectures. Networks follow the same dynamics improving and developing new versions every year which does not seem to change in the short term.

As an example, Facebook IA team has developed a software called Detection, powered by Caffe2 and written in Python, that incorporates lots of object detection state-of-the-art projects related also with deep learning and more specifically in neural networks.

Caffe2 is like TensorFlow, frameworks that let us to develop and implement deep learning projects and systems in a more easy way. TensorFlow is a more low-level framework than Caffe2 which it is characterized to be more high-level. These frameworks let projects to be deployed in other light-weight environments that has less computational power, such as on the cloud or on a smartphone. Deploying and using these networks in these described environments is a very important advance nowadays, due to the increasing cloud infrastructures, and mobile computation in our smartphones.

There are lots of frameworks for deep-learning purposes that makes both easy to work with and to develop projects based on complex neural networks, like The Microsoft Cognitive Toolkit/CNTK, Pytorch, Mxnet and Chainer among others.

Going through some projects in deep-learning, and more concretely, deep-learning related to object classification, we could talk about [8] that focus on the optimization of object classification using neural networks, improving the most computationally high cost task that are done in convolutional neural networks, convolutions. Convolutions requires not only lots of storage but also computational power to be successfully performed. Authors presented an approach to parallelise a matrix convolution based on multiplication (ConvMM) using graphics processing unit (GPU) in order to applying it in an embedded and mobile context, where a GPU is commonly present.

Many other projects related to object recognition using convolutional networks have emerged due to the demand of systems to classify a vast range of object types (natural and artificial objects, company products, etc.). These type of neural networks show good performance in these computer vision tasks but, as we mention in section 2, its execution needs huge computer power and storage space. Authors in [9] presented a convolutional neural network trained to detect mountain skylines in images. Their networks and training processes present good memory, powerful computer results as well as good runtime execution, which allow fitting and executing it in low to mid-end smartphones.

Going deeper in object recognition using convolutional neural networks in Mobile devices, which is focused in our developed prototype, a case study was realized [10] by its authors. This study investigates the implementation of light-weight convolutional

neural networks on mobile devices to resolve object recognition tasks that can be presented in many classification projects. Concretely, it focus on developing interactive interfaces on mobile platforms, like smartphones or tablets, to suggest interactive contents to users of the “Musée National de la Marine de Brest”. Objects are detected through the devices camera, localizing and identifying these objects in the image, and then tried to be classified to a certain class of object. To achieve this, users investigate fully convolutional neural networks (FCNN) and its deployment to mobile devices.

As we have seen, ImageNet was one of the first largest database image that is used to pre-train lots of neural networks. This database contains over 20 thousand of object categories with more than a million of images. Since 2010 this project celebrates an annual software challenge, ImageNet Large Scale Visual Recognition Challenge (ILSVRC). A contest where participants compete to classify object and scenes that are contained in this dataset.

As an example, authors here [1] used this dataset to train a deep convolutional network to be able to classify all 1.2 million of high-resolution images contained in ImageNet. They introduced the use of GPU’s to make the training process faster.

In this project we will use a pre-trained network that is also pre-trained with ImageNet, then, using transfer learning, we will change the final layer of the network to be able to classify our specified landmark objects and extract some geolocation information between the user and the objects.

Some groups are researching in Image-based Geo-localization. Exists a research group that is much related to this Image-based Geo-localization, a group at Cornell [4] that has presented some related articles like [5], where authors presented an approach to learn to detect and describe key points using deep architectures. A large-scale dataset of patches with multiscale key points is collected to feed the learning based part.

Other work researched by this group, [6], introduced a cross-view feature translation to extend image geo-localization methods existing. They can localize a query even if it has no corresponding ground-level images in database.

They also introduced a deep-learning approach [7] that evade local correspondence framing problem as a classification task. These models used are trained with urban aerial images.

Definitely, state-of-the-art in neural networks and object classification will change fast due to the big improvements in machine learning and artificial intelligence. Mobile devices are also improving very fast and give a new view to be able to integrate new deep-learning processes that has more computational costs.

4. Prototype development

4.1. Pre-development

In the first approach, after starting with the final prototype, we developed a small basic app in order to train a network with some set of objects located randomly in a small space like a terrace.

First, we tested a network pre-trained with the ImageNet dataset to see that it could be integrated in an Android app. Once we have a model working on a test app and also classifying input images of the camera, we start to take an existing model like MobileNet v1 and test if it was able to classify custom objects.

To train an existing network with our custom objects, we investigated what is transfer learning and how it could be achieved using TensorFlow. The purpose of this transfer learning is to get a pre-trained model and re-train it in a small amount of time. Finally we saw that network can be changed to classify our own objects. Once the model was re-trained, we used the same test app to evaluate it the retrain was successful.

We see that the training images to train the system are very important to have good results in classification tasks. Our initial system was trained with 8 different objects like chairs, bicycle, and some plants that can be moved and placed easily. If the training set is small, the accuracy of the classification network will not be good enough and can be easily overfitted.

This initial test application was able to classify our 8 objects mentioned before. One thing that we realized is that the training pictures need to be focused only on the object in order to be well classified. We realize that most of the objects that are located in my terrace have the same background, a grey floor. If this background appears in all type of objects that we are training, the classification process will have more problems to do it correctly. Then, when we try to take object pictures in different backgrounds and situations, the network seems to have better results in the training and classification processes.

After the first test, we realized that the small area we tested was a limitation to integrate the geolocalitzation part of our prototype so we decided to move to a more extensive space like it is “Parc de la ciutat”. There we could be able to carefully develop, test and evaluate this prototype.

In a more extensive place, geolocate user through object classification let us doing more test paths and walkthroughs. These paths allowed us to extract some conclusions to evaluate the entire classification prototype.

Once established the space and tested the first network (MobileNet v1) we moved on evaluating two other networks, Inception v3 and ResNet to compare the results and accuracy with the first one. Unfortunately, we realize that TensorFlow Lite and TOCO (TensorFlow Lite Optimizing Converter) have a lack of training operations due to its recent release. So we decide to only use MobileNet v1 that it’s tested and permits to do the retrain and the conversion to TensorFlow Lite without any problem. These newer and

more efficient networks should be tested and reviewed in a future extensions of this project.

Altogether with the chosen network properly working, and the space determined, we have all system parts needed to start developing and training the prototype project presented in this document.

4.2. Scheme

The prototype scheme is represented in figure 12.

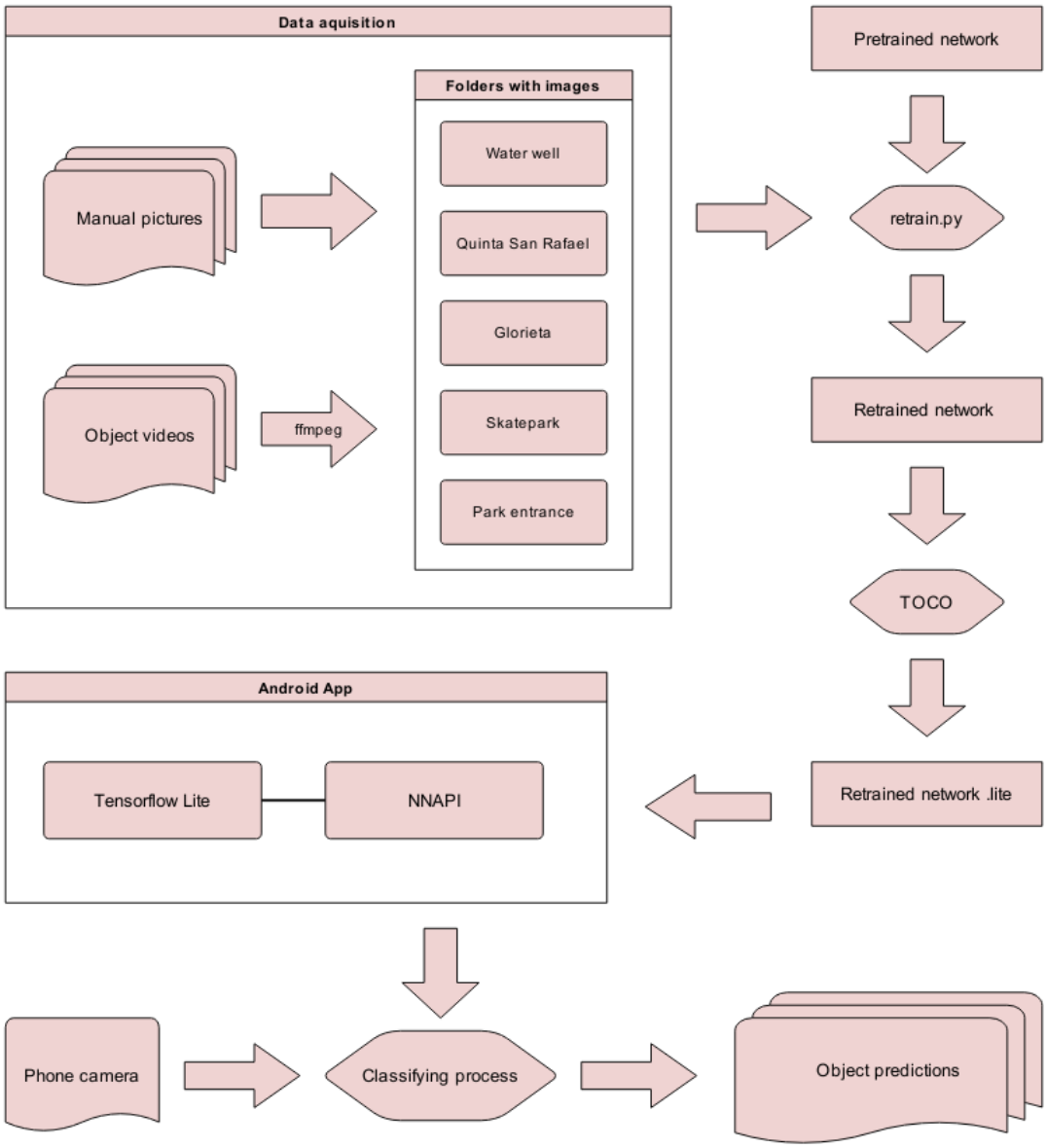


Figure 122: Prototype scheme

- Data acquisition: Is the part related to obtain input data needed to be able to retrain our network. This data is achieved taking manual pictures of the classification objects and also recording one video per object. These videos are used to extract frames that will be used as images for our training process. We will see more in detail in section 4.3 how it can be achieved and what kind of object we are going to classify.

- Training process: Uses de input data of the data acquisition part to retrain an existing trained network to be able to classify our custom objects. These objects have a geolocation where our prototype will be developed.

- Android app: Android application witted in Java that integrate the retrained network, TensorFlow Lite and NNAPI to be able to classify our custom objects and be able to generate evaluation data of the executions.

- Classifying process and object predictions: Is the process where the app classifies input real-time data obtained from the smartphone camera and show object predictions represented as a probability that matches trained objects.

4.3. Training network

In order to achieve our proposal, a MobileNet v1 network is used to classify our objects and buildings in a geolocation space.

These MobileNets are small, require low power and have low latency to fit most of the existing use cases that a network can be integrated. Also can be used for a variety of objectives such as object classification and object detection in a similar way than other large network models like Inception or ResNet do. MobileNets can be run efficiently on mobile devices with TensorFlow Mobile and TensorFlow Lite.

The MobileNet is configurable in two ways:

- Resolution of input images: 128,160,192, or 224px. Classification of input images in a higher resolution takes more time to process the results but the accuracy increases.
- Width multiplier that was introduced above in section 2.1.3, that is the relative size of the model as a fraction of the largest MobileNet: 1.0, 0.75, 0.50, or 0.25.

There are lots of models pre-trained to be used in TensorFlow, but as we mention above, lots of models presents problems when converting they to a .lite version (TensorFlow Lite) in order to be used in a smartphone environment. So I choose to use a Mobile net with 224px, that is shown in the table, https://github.com/tensorflow/tensorflow/blob/master/tensorflow/contrib/lite/g3doc/models.md. Greater input images involves more accuracy in the network. And 1.0 as the width

multiplier because our prototype application is lightweight and can hold the entire mobile net. This final size of the model will be smaller when we have to convert this network to a .lite version.

Our training process is done using transfer learning, which means that we are starting with a model that has been already trained on another problem (ImageNet). We will then retrain it on a similar problem that will be 5 custom objects. Deep learning from scratch can take days to train the entire network, but transfer learning can be done in a less period of time.

We are going to use a MobileNet v1 trained on the ImageNet Large Visual Recognition Challenge dataset. These models trained with this image set can differentiate between more than 1,000 different classes.

To achieve this re-train with our categories, we have used Python and TensorFlow. These categories are 5 objects and buildings placed in “Parc de la ciutat”, shown in figure 13.

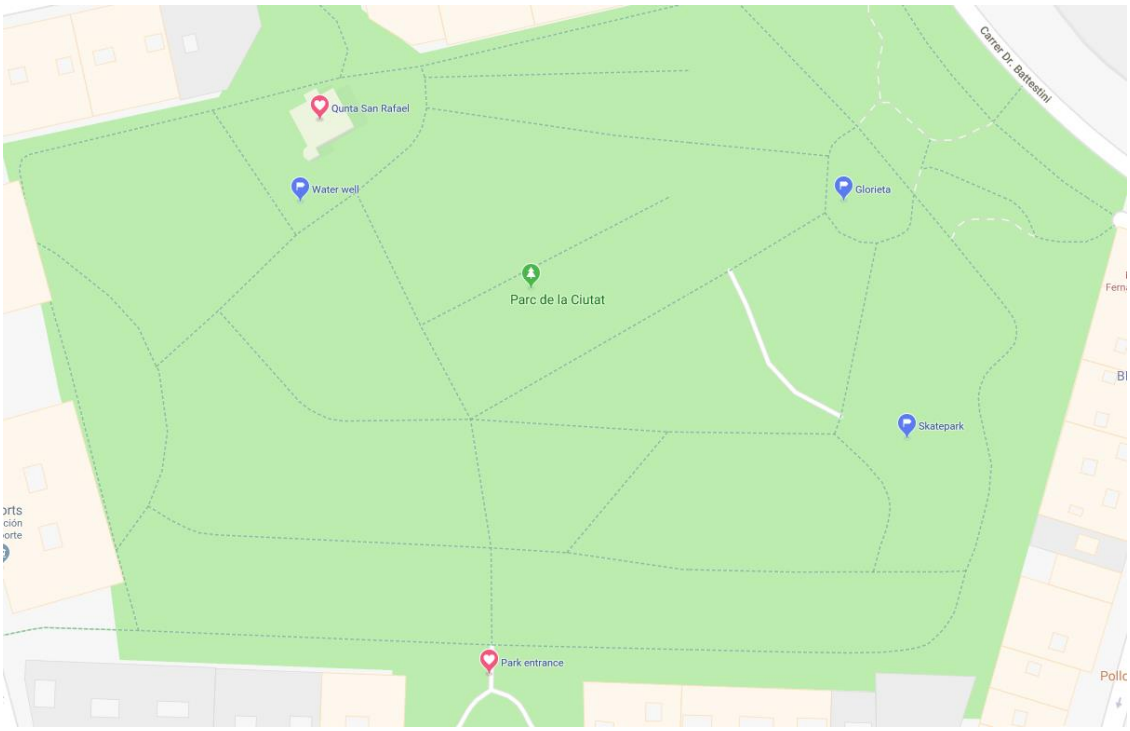


Figure 13: Custom objects geolocations

Figure 14, 15, 16, 17 and 18 shows the images of the five objects to be classified in our prototype.

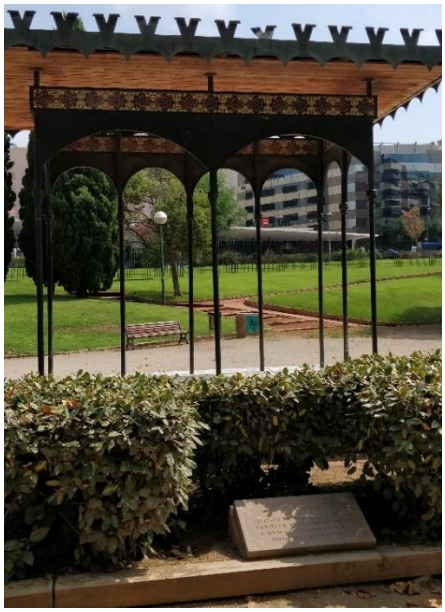


Figure 14: Glorieta

Figure 15: Park entrance

Figure 16: Quinta San Rafael

Figure 17: Skate park

Figure 18: Water well

To retrain the network, we need to have one separated folder for every class we want to classify. Folder's name will be the label of the class. These folders have to have lots of images related to the object we want to classify in our prototype.

This part is the data acquisition part of the prototype. Images are obtained in a manual mode combined with a video mode. In a manual mode, it's easier to take better pictures of the objects. In the video mode we have more noise due to the hand movement of the user. One we have the videos, we use ffmpeg to extract frame images of the video in order to have more training images.

Ffmpeg is an open source multimedia framework to record, convert and stream audio and video and let us to extract image frames of the training videos. Then, these frame images are used to train our network.

To achieve this retraining, we have used a python script developed by google that let us to change the final layer of our network:

https://github.com/tensorflow/hub/blob/master/examples/image_retraining/retrain.py

ImageNet does not include any of the objects we're training. However, the kind of information that make it possible for ImageNet to differentiate among 1,000 classes are also useful to be applied to classify other objects. Using this pre-trained network, we are using that information as input to the final classification layer of the network that distinguishes our prototype custom classes.

This is the execution command for the retrain.py script:

$ python3 -m scripts.retrain --bottleneck_dir=tf_files/bottlenecks --how_many_training_steps=400 --model_dir=tf_files/models/ --summaries_dir=tf_files/training_summaries/"mobilenet_1.0_224_final" --output_graph=tf_files/retrained_graph.pb --output_labels=tf_files/retrained_labels.txt --architecture="mobilenet_1.0_224" --image_dir=tf_files/test_photos

- Training steps: We set this parameter to 400, due to the type of images and the number of images we have. The output training graphics (figure 19 and 20) shows that the script doesn't need more than this number of steps.
- Architecture: As we have said, a MobileNet v1 224-1.0 network pre-trained model is used.
- Image directory: Contains all the 5 labelled folders with the training images in it. Out system will classify these 5 types of objects.

First, bottlenecks are calculated. Bottlenecks are values for each image in the training dataset that are saved and later fed to the classification layer. The bottlenecks are designed to be useful pieces of information that the classification layer can use to distinguish the input images more efficiently. Without these bottlenecks the retraining process would be much slower.

Retraining process is divided in steps. In each step 10 images are randomly chosen from the training set. Their bottlenecks are then fed to the final layer of the model and used to predict the classification. Rather than the classifier is right or wrong, a recalculating of weights is passed to the nodes in the network model. In each step, 3 values are calculated:

- Training accuracy: the training accuracy shows the percentage of the images used in the current training step that is correctly labelled with its corresponding class.
- Validation accuracy: The validation accuracy is the percentage of correctly-labelled images of a group of images from a different set. These images are selected in a random way.
- Cross entropy: is a loss function that let us to know how is going the learning process. Lower cross entropy is better.

Accuracy:

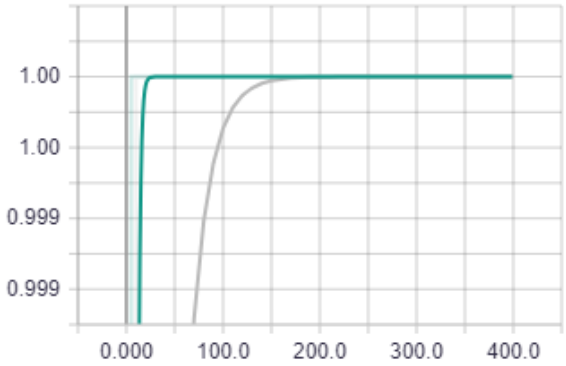


Figure 19: Training vs validation accuracy

There are two lines in figure 19. The green line shows the accuracy of our model related to the training data. The grey line shows the accuracy related to the validation data set (this set is not used for training). This parameter is a good measure to know the performance of the neural network.

After all the training steps are completed, the script runs a final test accuracy evaluation on a set of images that are kept separate from the training and validation pictures. This test evaluation gives an estimation of how the trained model will perform in the final classification task when it will be in a production mode.

$ INFO:tensorflow:Final test accuracy = 100.0% (N=41)

We should see an accuracy value of between 85% and 100%. This exact value can variate in every training execution due to the randomness of the training algorithm. We only train 5 classes so the accuracy is increased drastically. This number value indicates the percentage of the images in the test set that are classified to the correct object after the model is fully trained. I have a 100% of accuracy due to the type of images, and the number of images. All images are similar, very similar, due to be able to well classify it. I took pictures in different angles but all these pictures need to only show object that I need to classify. If I take pictures with more distance between user and object, and these pictures show more perspective that only the desired object, my network doesn't classify pretty well objects due to de environment of the park and the background of the objects.

The park has lots of vegetation and sand ground, so all pictures have a % of the image very similar with other object class pictures.

We will see this behaviour, in section 5, when we evaluate the final prototype. We will see that the park entrance, is the object that is worst classified due to this explained problem.

Cross entropy:

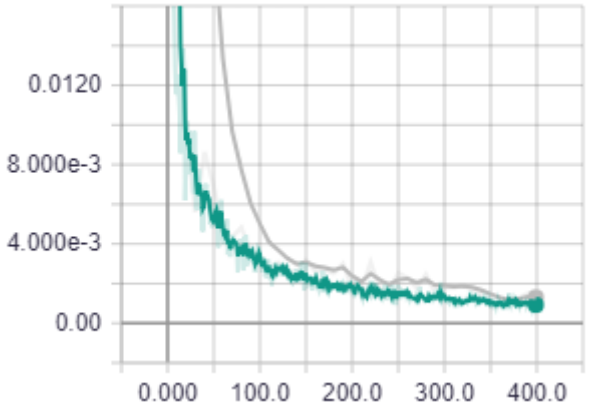


Figure 20: Training vs validation entropy

As we can see in figure 20, cross entropy decreases drastically and very fast, so the network seems to be accurate and it progresses well during the training process.

The retrain script finally outputs 2 files:

- .pb with the network model.
- .txt with the trained object labels that our system can classify.

Then we have to convert this .pb network model file into a .lite file in order to integrate it in our android application. This can be achieved using "TensorFlow Lite Optimizing Converter" or TOCO.

TOCO applies several optimizations that are useful for our graph and convert it in a light-weight version of the original .pb file. These include deleting unused graph-nodes, and joining operations into more efficient composed operations.

The pruning is especially helpful given that TFLite does not support all training operations yet, so these operation should not be included in the graph. This lack of training operations, as we comment before, is the main reason to use MobileNet v1 and no other more accurate networks as GoogleNet.

TFLite uses a different serialization format from regular TensorFlow. TensorFlow uses Protocol Buffers, while TFLite uses FlatBuffers.

The main advantage of FlatBuffers is the fact that this kind of buffers can be saved in memory directly, without being needed to load and parse it after a read.

This advantage gives faster start-up times, and gives the possibility to load and unload parts of the model from de memory, instead of removing entire app from the memory (killing it).

Once we have executed TOCO with the retrained network input “.pb” file generated by the retrain script, a new .lite file is generated. This new generated file is the one that will be imported to android and let our prototype app to classify real-time camera images.

4.4. Android App

The application created in this prototype is based on classification real-time input image of smartphone camera to be able to geolocate user in real-time inside some determined location. The App integrates the TensorFlow Lite network model and classifies Bitmaps extracted of the camera.

App is developed under Android Studio SDK that is a powerful framework powered by Google. This framework is used to create Android apps in Java and Kotlin.

To achieve project purposes, we will need to integrate TensorFlow Lite, Google Maps and Google Location libraries. These Google libraries are only used to evaluate the final result of the project to see where the user locates during a walkthrough in the tested area.

Therefore, the real objective of the project is to extract user geolocation information only filming trough this app and his smartphone.

4.1.1 Dependences and permissions

As stated before, some dependences are needed to execute TensorFlow and Google geolocation services. Moreover permissions are also required by Android systems to access to the camera, external storage and gps of the user's smartphone.

To integrate TensorFlow Lite, we need to add these lines into Gradle. Gradle controls all the libraries imported to android in a real-time and intelligent way:

```
allprojects {
    repositories {
        jcenter()
    }
}

dependencies {
    compile 'org.tensorflow:tensorflow-lite:+@aar'
}
```

Also, we need google play services maps and location to be able to show locations on a map and access user gps location at every moment:

```
dependencies {
   implementation 'com.google.android.gms:play-services-
maps:15.0.1'

implementation 'com.google.android.gms:play-services-
location:15.0.1'

}
```

Android permissions needed to achieve project objectives are:

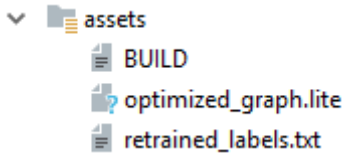
```
<uses-permission
android:name="android.permission.ACCESS_FINE_LOCATION" />

<uses-permission
android:name="android.permission.ACCESS_COARSE_LOCATION" />

<uses-permission
android:name="android.permission.WRITE_EXTERNAL_STORAGE" />

<uses-permission android:name="android.permission.CAMERA" />
```

Location permissions are required to geolocate user and its smartphone in every moment. This is used when clicking the record button in the main classifier activity and also when accessing object location activity (section 4.1.2).

External storage is needed to be able to save result files into internal storage of the smartphone, to be able to use these files later in the evaluation part.

And, obviously, camera permission is required to access camera and show its images into our prototype app.

In assets folder of the project (folder to place external files of an application), we need to locate our network TensorFlow Lite file (.lite) and the label .txt file generated by retrain.py script:



4.1.2 App activities

First Screen is the App welcome activity (showed in figure 21). This activity presents the project app and only have one button to let us to start the classification process.

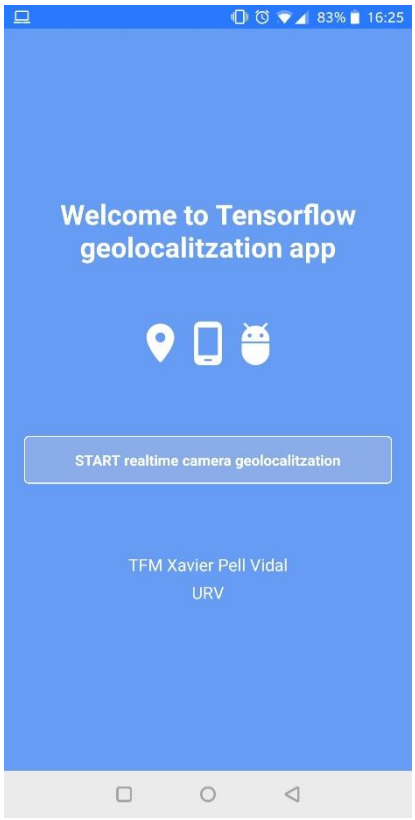


Figure 19: Welcome activity

Once we click in the START button, a new activity is started to show live camera images obtained from the smartphone. Also the classification process starts (Figure 22).

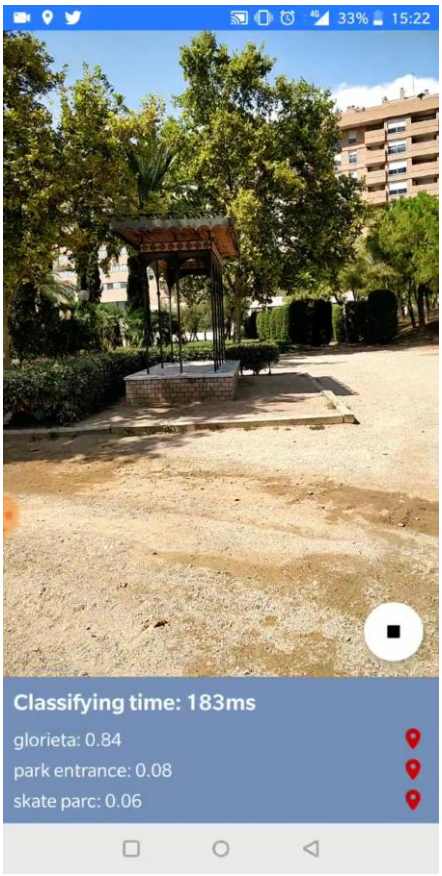


Figure 20: Main classifier activity

At the bottom of the screen, the top 3 object predictions are shown ordered by probability. Also, the classification time used to classify a single bitmap. The upper part is what smartphone camera is capturing in real-time.

App extracts a bitmap from the camera and classifies it. First of all we need to resize image to 224px and then pass this new bitmap to the TensorFlow model.

TensorFlow classifies it and returns one probability for every object trained before.

This screen also let us to recollect and save some statistical files during a period of time. When stop button is clicked, app generates two folders with information about the execution. Concretely, a .csv file with information at every image classification and a screenshot of what it's seen through the camera.

This screen also starts a location tracker that let us to know user geolocation at every moment. This data is used and stored in evaluation files mentioned above. We will see in more detail at section 5 what is saved and how.

App also let user to click location icons that are at right of object prediction. Clicking in it, a new activity is opened to give object location information in a map. This object location activity also shows current location of the user to be compared with the object location (figure 23).

Figure 21: Object location activity

In blue, current location of the user, in red, object location information about the object clicked in the previous classification probability list.

This Activity is used to compare system prediction results and know if the system is working well. In Evaluation section, we will see how and when we could use this activity to check if the system is working well.

5. Evaluation

This part we will try to see how the system works. As I mention before, app generates some files to be able to evaluate result information about geolocations and classification. A brief scheme in figure 24 show the evaluating process of the prototype.

When user click on the “disk” floating button in the main classification activity, it starts to track some relevant information about the user and the trained objects.

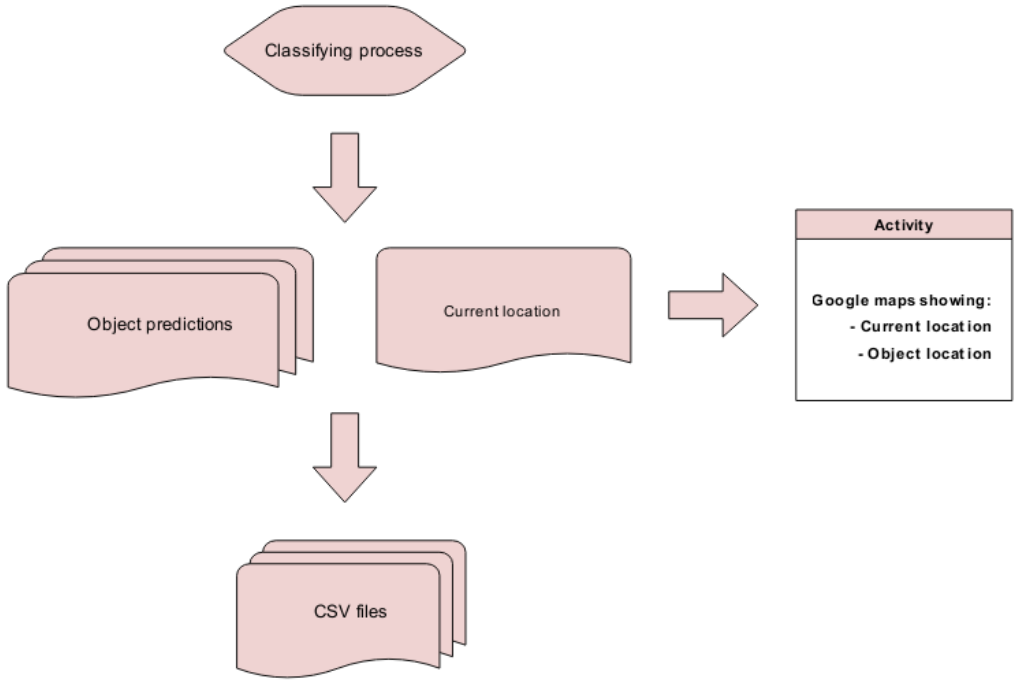


Figure 22: Evaluation scheme

First of all, we need to know, at every moment, the geolocation of the user that is executing the app and using his smartphone. This geolocation is the same as the geolocation of the smartphone camera. This location needs to be represented in a map, and be compared with the geolocation information extracted by our prototype.

As we say before, this is achieved with a location tracker that uses phone GPS to obtain the user coordinates at every moment.

Evaluation process is divided in 2 parts:

- Activity that shows object location: this activity can show us if we are near one of the objects that we need to recognize. Also we can use it to know where we are in the map. So, assuming a threshold to know if the prediction is reliable or not, we

can check if the system geolocation prediction is right and check if we are near the classified object.

- A .csv file and screenshots are generated for every execution: this file contains relevant information about all objects and distances between user current location and objects location. Screenshots are used to know what is showing camera at every moment in order to link it with file information.

The CSV file is populated with these columns:

- Time of the row insertion.
- Time of the row insertion in milliseconds.
- Current location of the phone.
- Probabilities of every object in the system.
- Distance from user geolocation to every object trained in the system (in the park), in meters.

Each row was inserted every 5 frame classifications. For probabilities and distances columns, we do the average of the 5 execution values in order to reduce noise predictions and preventing the file to be too large. This average is what we will record in the evaluation file.

When we have the .csv file and the screenshots, we could extract this information and see how system works. To achieve this evaluation, we made 3 tests:

1. Approximations to each object of the system. One recording path that is composed by 2 approximations per object.

2. Approximations to 2 objects that are near between them and can be captured together in a same camera picture.

3. A walkthrough for the entire park to classify all the objects and see how the system is working.

5.1 Simple approximations

As we said above, this test is done by walking direct to the object to classify, then returned until the system don't recognize the object. This process is repeated 2 times in different directions. Always directing camera to the object that is being analysed.

Now we will evaluate data and check resultant graphics for every object in the park:

Water well

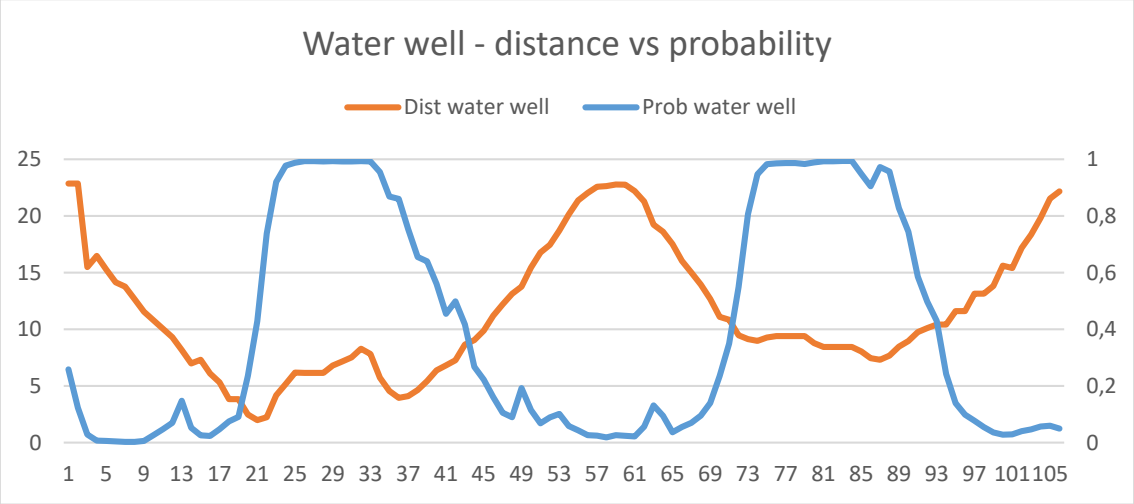


Figure 23: Water well distance vs classification probability graphic

As we see in figure 25, we have done 2 approximations to the water well from 25 meters. In the first approximation, when distance decreases to 4-5 meters, probability of the object in the camera, water well, starts to increase. The same occurs when we walk away from the water well, probability starts to decrease.

More in detail, in this space of time, we see in the distance graphic that the user moves to 2-3 meters, then walk away to 7-8 meters and again, moves to 4-5 meters (points 18-40). This walk away to 7-8 meters doesn't exists in the real path of the evaluation path so it is related to GPS noise. Screenshots (of points 18-37) of figure 26 shows no walk away when we are near water well filming it in the firs approximation.

Figure 24: Screenshots near water well

Similar thing occurs in the second approximation to the object. Figure 25 says that we only went to 7-8 meters near water well, but the approximation process and distance achieved was the same as the first approximation. This approximation, in both times, is at 2-3 meters of water well. This is also that real-time GPS is not as accurate as we expected. Figure 27 shows screenshots of the second approximation. We could see that the distance is more or less the same that the first approximation (points 79-83 in figure 25):

Figure 27: Second approximation to water well

As we mention above, GPS says that we are at two different distances from water well in the two approximations. It's is not as precise as we expect to be but it give more geolocation information than our prototype. Our classification system, says that we are near water well when we are close to water well in a distance less than 25 meters. When the probability is 0.6 or high, and we are focusing our object, our system knows geolocation of user and can say that we are very close to water well.

Figure 28 shows the path followed in this evaluation of water well.

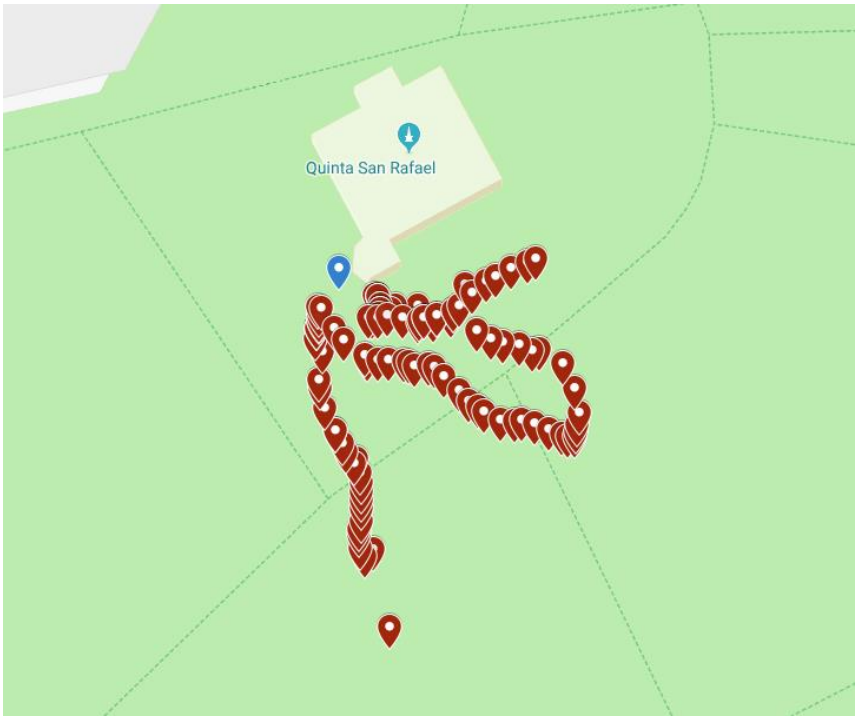


Figure 28: Water well evaluation path

Blue point in figure 28 is the object (water well), and the other red points are the locations saved in the .csv data file during the approximation.

Now, focusing in the object evaluating screen, when system is giving a probability from 0.6 to 1.0, we could say that we are close to water well (figure 29).

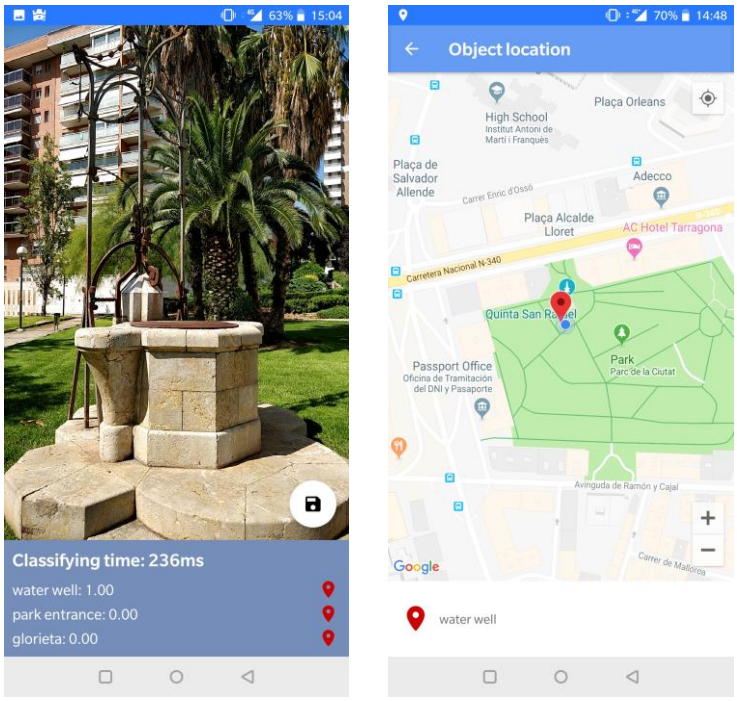


Figure 2925: Water well object location activity

Quinta San Rafael

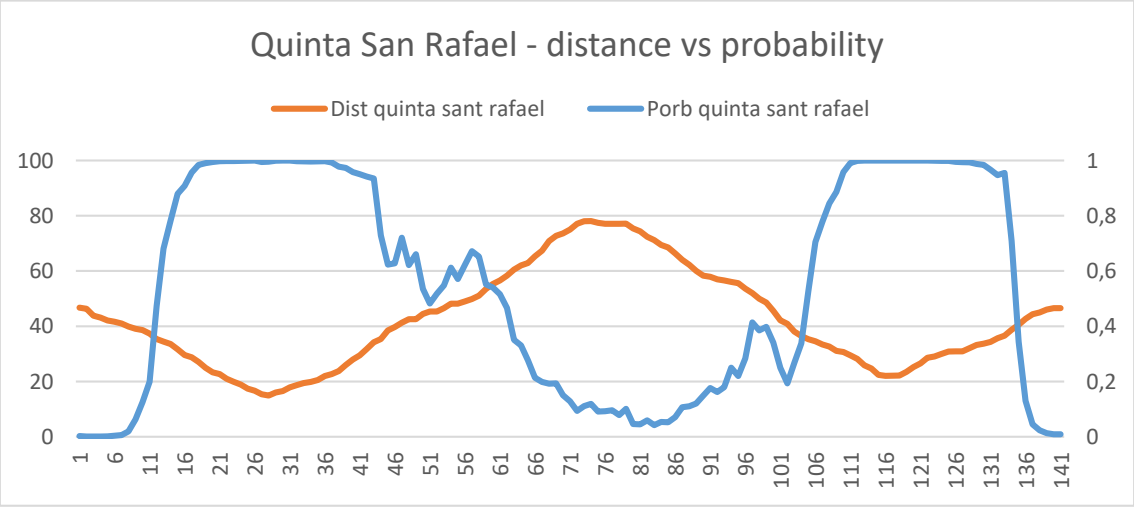


Figure 30: Quinta San Rafael distance vs classification probability graphic

Graph from Quinta San Rafael is showed in figure 30. In this case, we only can approximate to the building until 20 meters due to be a big building where the geolocation point extracted is at the centre of it.

We can see that this approximation is better than the previous one. This building is bigger than the water well, and doesn't have trees and other objects behind it, appearing in the background. So classification process is more accurate and system better recognize it rather than the water well.

Probability increases drastically when we are 40 to 20 meters from Quinta San Rafael. At that point we could say that we are geolocated near the evaluated object.

Some noise in classification process is showed when we walk away the first time and when we approximate to the object at the second approximation. This is due to some big trees that are between user and object. In figure 31 we could see some screenshots from points 100 to 104 (figure 30).

Figure 31: Noise due to trees between user and object (1)

Images in figure 31 show that Quinta is small in comparison to trees, ground and sky, this fact makes probability to decrease at these points.

The same occurs when we walk away from object after the second approximation (see figure 32), trees appears in the camera and probability decreases drastically (137-141 points from figure 30).

Figure 32: Noise due to trees between user and object (2)

Complete evaluating path for this approximation is showed in figure 33.

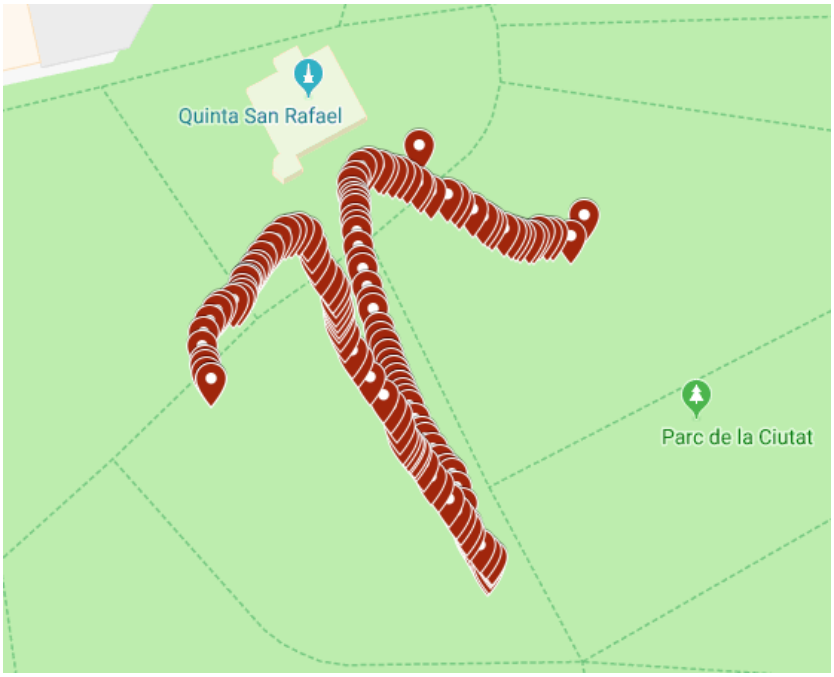


Figure 33: Water well evaluation path

Focusing in the object activity that shows where Quinta San Rafael is and where is user, we can see that if the probability is 0.7 or higher, we are very close the evaluated object (figure 34).

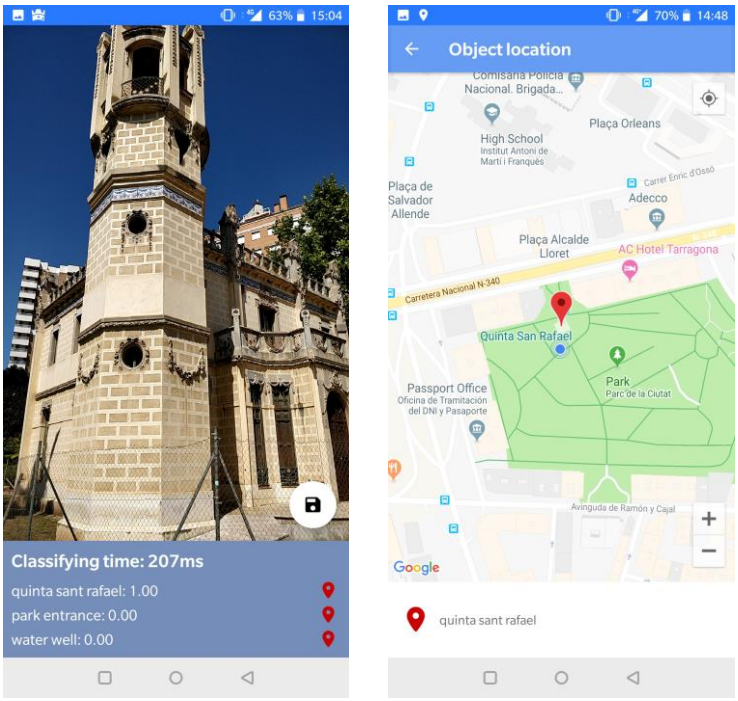


Figure 34: Quinta San Rafael object location activity

Glorieta

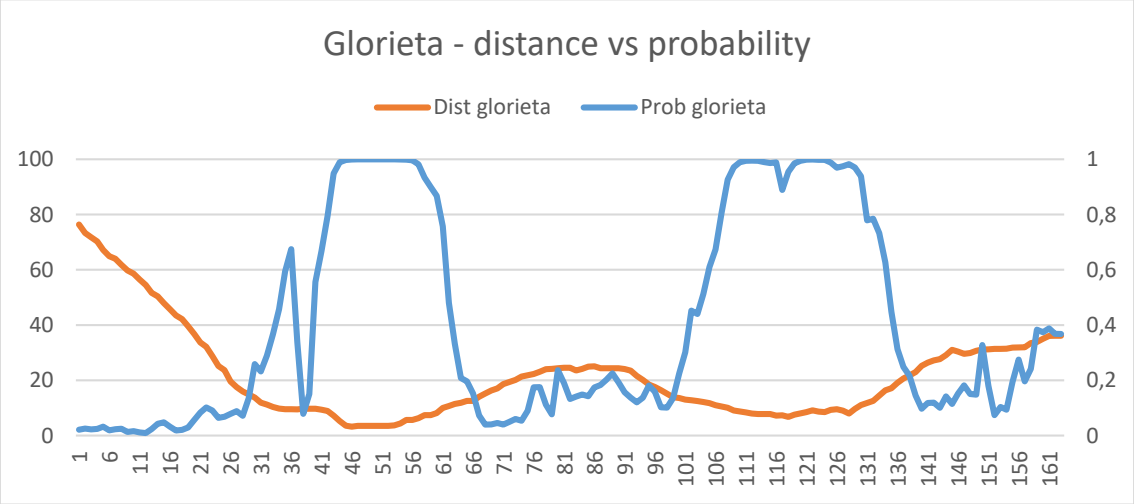


Figure 35: Glorieta distance vs classification probability graphic

In this object classification (figure 35), we could see that we need to go closer to start obtaining confident probabilities. We could obtain probabilities 0.6 to 1.0 when we are 0 to 12-13 meters away from Glorieta.

This object is like water well, it has vegetation at the bottom and it's not solid, we could see what there are behind it. This thing makes the classification more difficult, also the environment near Glorieta is full of trees and ground, that makes probability to decrease very fast when we walk away from the object location.

In the first approximation (figure 36) we also could see a big fall of the probability, this is due to not focusing Glorieta when we was walking to it.

Figure 36: Noise due to camera not focusing to Glorieta

Figure 37 shows the final path of this evaluating execution.

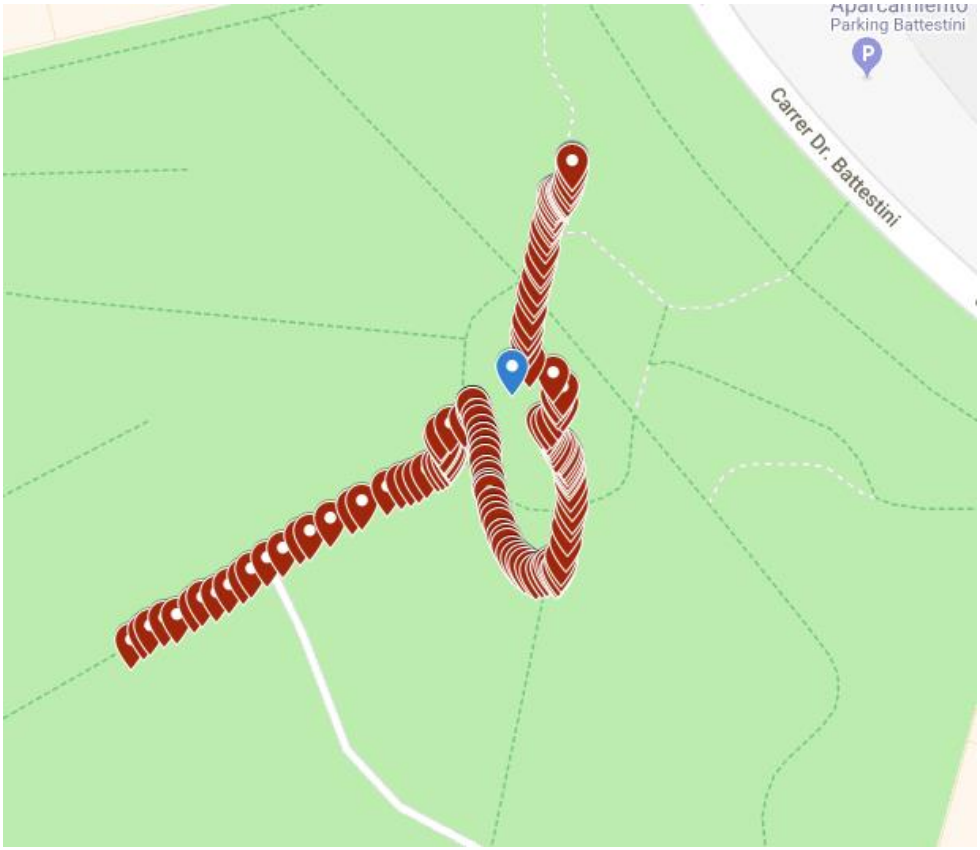


Figure 37: Glorieta evaluation path

Red points in figure 37 are geolocations of user when approximating to the object. Blue point is Glorieta, object being classified.

If we have a probability of 0.7 or more, and we are filming this object without trees, in a clear way, we could say that we are near Glorieta (figure 38).

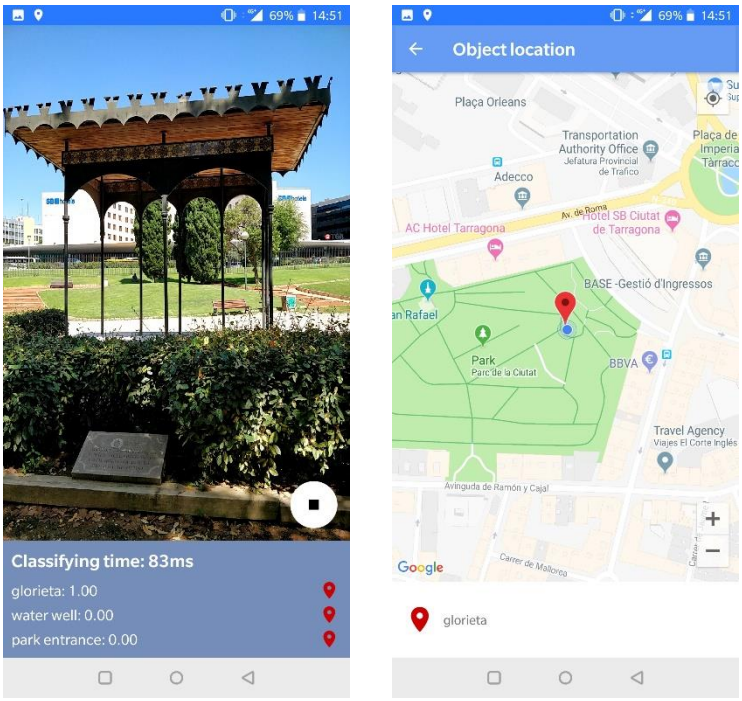


Figure 38: Glorieta object location activity

Skate park

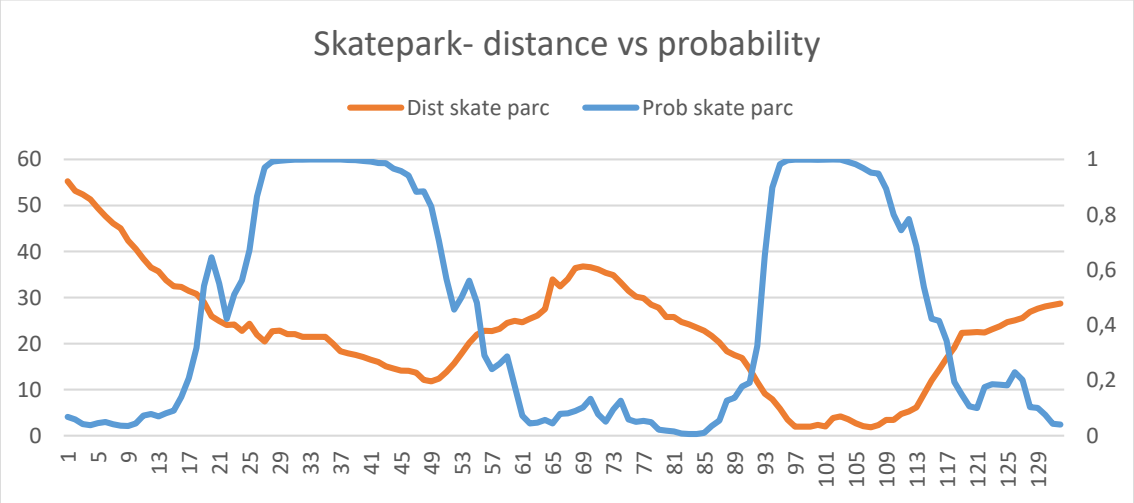


Figure 39: Skate park distance vs classification probability graphic

Figure 39 shows that this object is similar to Quinta San Rafael. It's more solid and doesn't have holes that let other objects to appear in training images.

But probability also depends on the training images. As a result, we can see in the second approximation that it is done in a different angle, and this makes user to be closer to the object to have better classification probability (first approximation probabilities increases faster than the second approximation). Furthermore, like in the other object evaluations, if we are far away more than 20 meters from Skate Park, camera records outer background objects, trees and ground that make difficult to be able to classify Skate Park in a good accuracy. This occurs when object doesn't occupy the major part of the picture area.

Figure 40 shows the final path path followed to record this approximation.

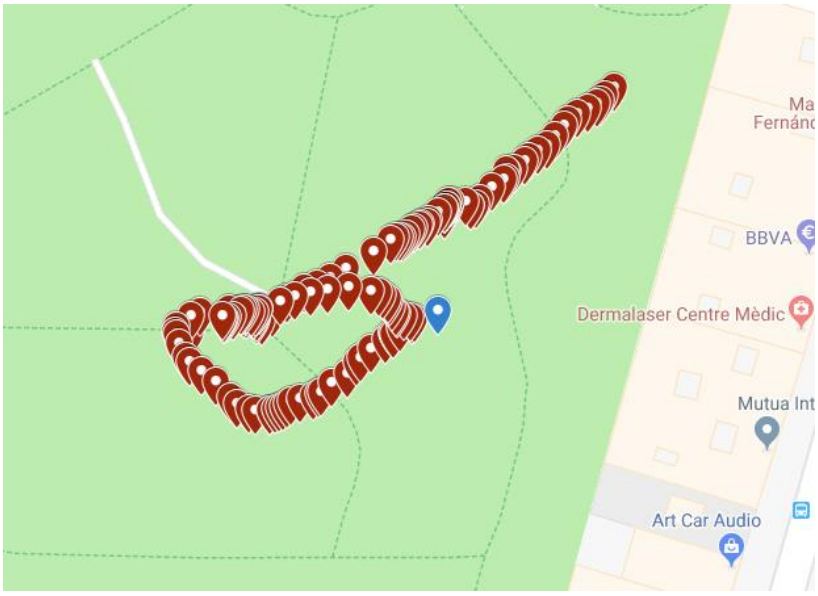


Figure 40: Skate park evaluation path

Red points are locations of user when doing the evaluation path, blue one is the skate park.

Same happens as in previous objects, if we are obtaining a probability between 0.7 and 1 it means that we are very close to the skate park (figure 41).

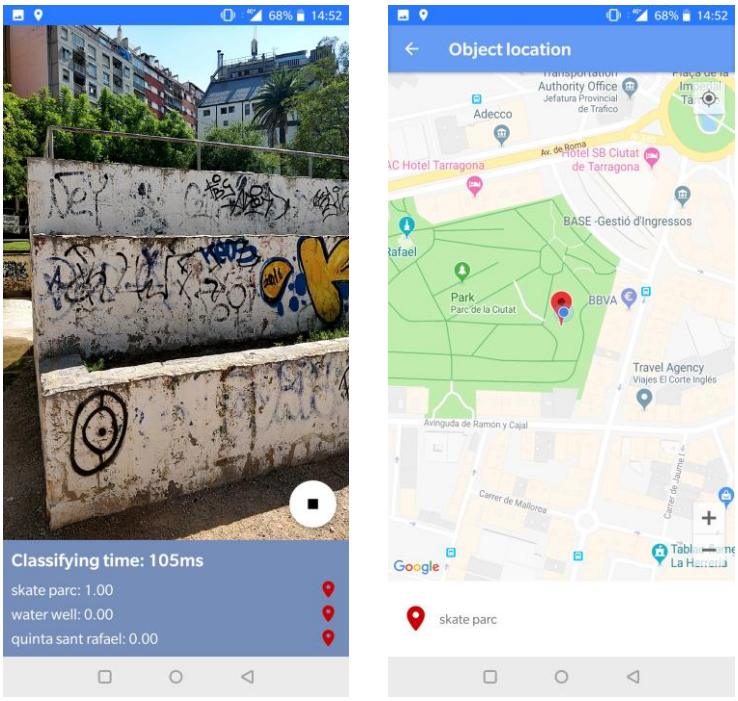


Figure 41: Skate park object location activity

Park entrance

Final object to evaluate is the park main entrance. This object presents more problems than the others. It is bigger and wider, and also have holes in it, so it's not completely solid. Also it has trees in front of it.

Let's see the graphic in figure 42.

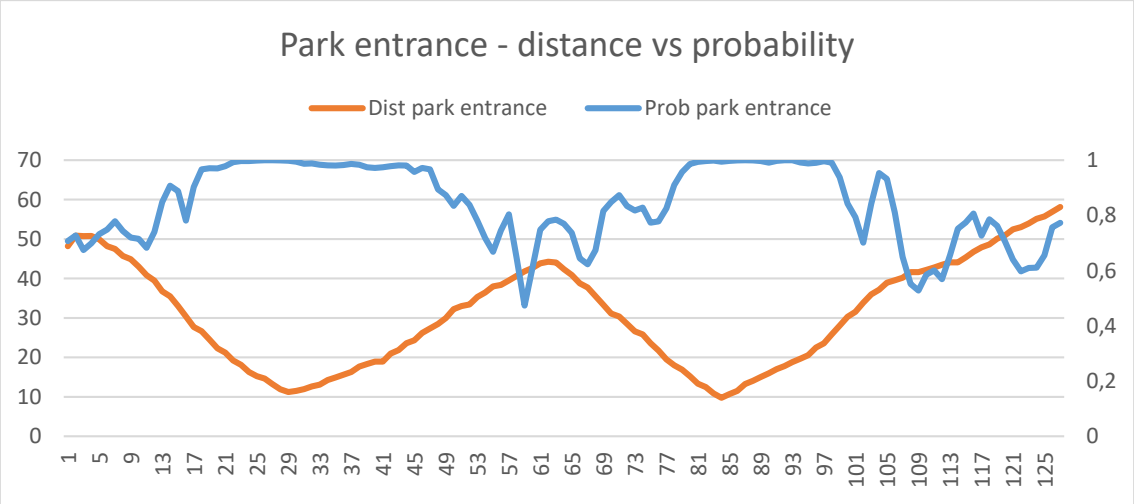


Figure 42: Park entrance distance vs classification probability graphic

Probability doesn't go down under 0.5. This seems to be better than other objects because we are focusing object at every moment and system says that the park entrance is recognized at every location point of the path. But as we mention before, this is a problem that we will see in detail in the final walkthrough in section 5.3. System classifies park entrance with too much probability and not only when we are focusing it.

Figure 43 shows the evaluation path followed to extract evaluation data.

Figure 43: Park entrance evaluation path

Blue point in figure 43 is the park entrance, red ones are location of the user at every data save moment.

In this case, to evaluate it with the object activity we need to have a higher probability due to the results showed above in figure 43. This object is not as good classified as others because we could get a probability of 0.85 when we are far away of the park entrance. So in this case, we need to get a probability of 0.95 or more to be sure that we are close to the classified object (figure 44).

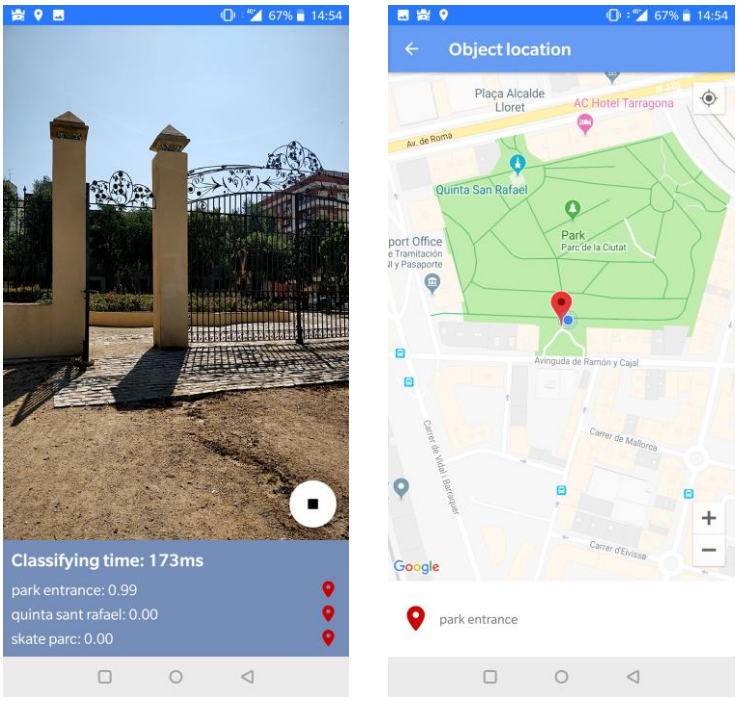


Figure 44: Park entrance object location activity

5.2 Composed approximation

In this part of the evaluation we will see what happens if we record two objects at the same time. This can only be achieved with water well and Quinta San Rafael because of they are very close to.

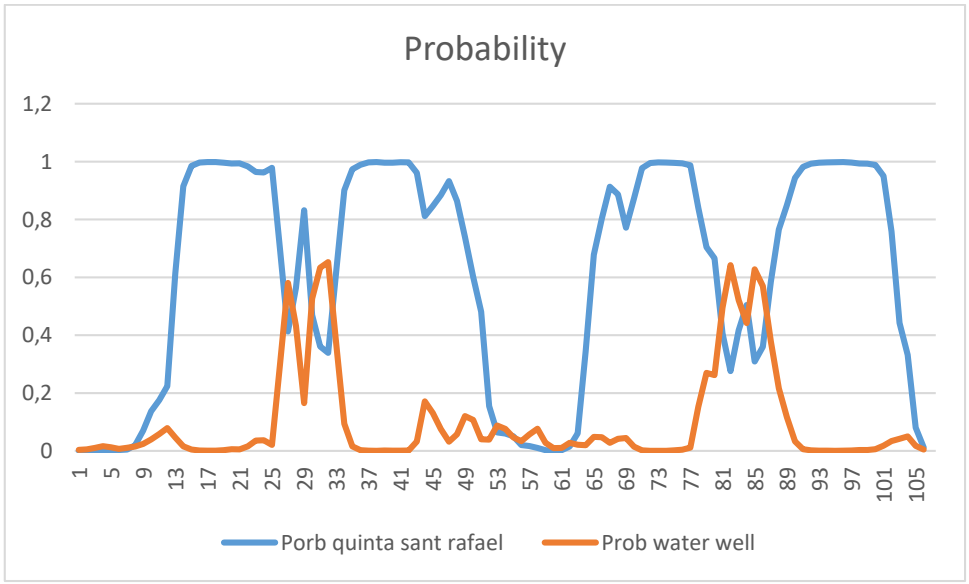


Figure 45: Quinta San Rafael and Water well probability graphic

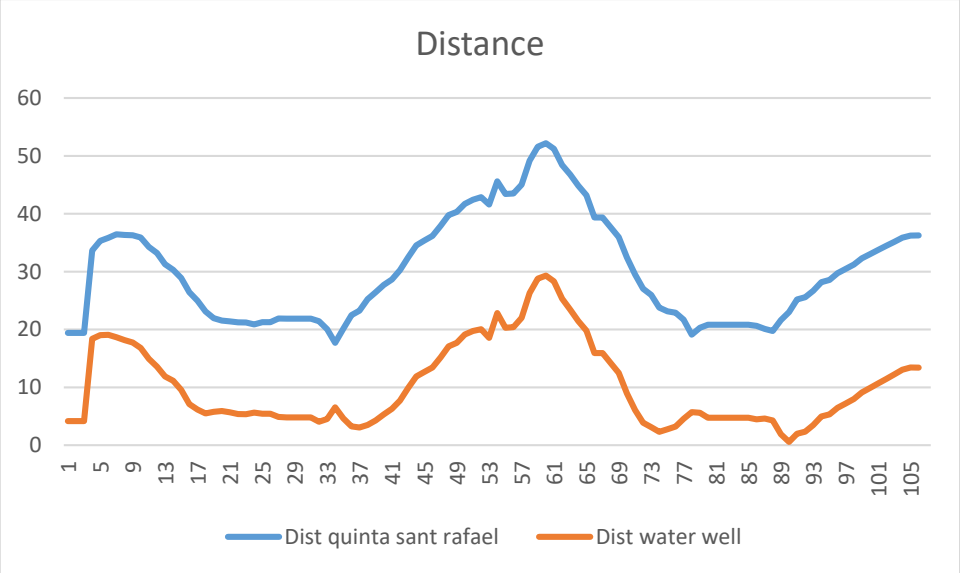


Figure 46: Quinta San Rafael and Water well distance graphic

Distance graphic (figure 46) shows that we have always been near water well during the path. This is because location of Quinta is at the centre of the building and is always more far than the water well. At points 52-63 in this 2 previous figures, any of these 2 objects was detected by the classifier because of the distance and the noise between user and objects.

Probability graphic (figure 45) says that Quinta Sant Rafael always has a bigger probability than Water well. I happens due to its size and characteristics. It is solid, as we saw before. Water well it's always hard to be well classified if we don't be very close to it.

There are 2 parts of the graphic (figure 45), when we are approximating to the 2 objects, and when they are appearing both in the camera, we could see that probability of the two objects is balanced, and classification process says that we have these 2 objects together.

Figure 47: Quinta San Rafael and Water well in the same photo

Figure 47 shows these two objects together. They need to be well focused, if not, Quinta San Rafael takes more protagonism (for its size) and classification process don't recognize water well.

Figure 48 shows the path of this composed approximation evaluation.

Figure 48: Composed approximation path

5.3 Final walkthrough

A final walkthrough is done in the park to record all object together trained in the prototype. The next figures shows information for every object.

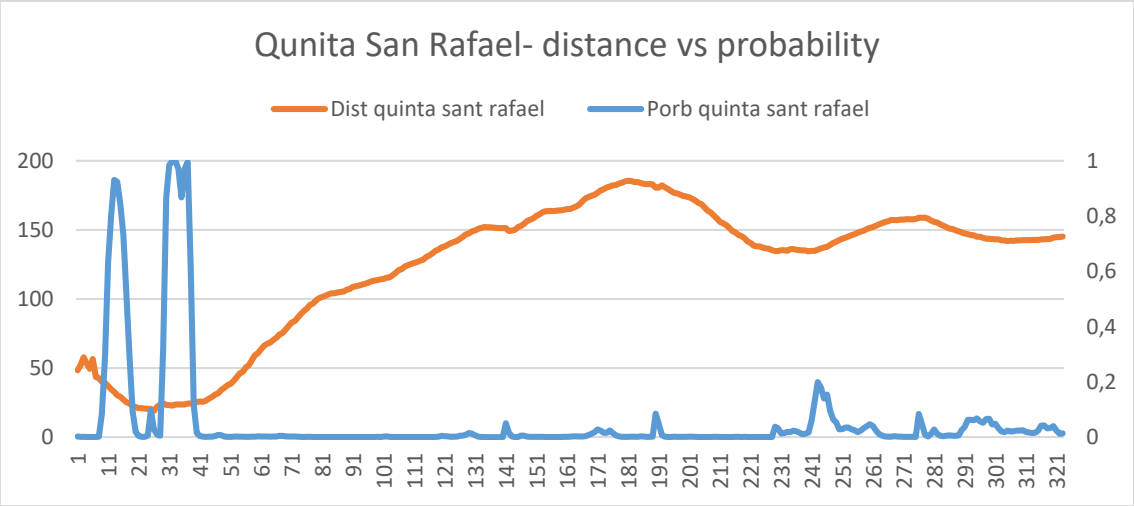


Figure 49: Quinta San Rafael distance vs probability graphic (final evaluation)

As mention before in the single approximation evaluations, when we are going to see water well and Quinta San Rafael, Quinta takes more space in pictures and classification process says that we are very close to it (figure 49). This information is true if we check user location in the object location activity. But in an ideal classifier, water well and Quinta should have probability shared as it happened in the composed approximation.

In figure 50 we can see that at the same point, we focused on water well (Without Quinta appearing in the pictures) and the probability increases to 1 very fast, due to be at the same location (we don't need to walk).

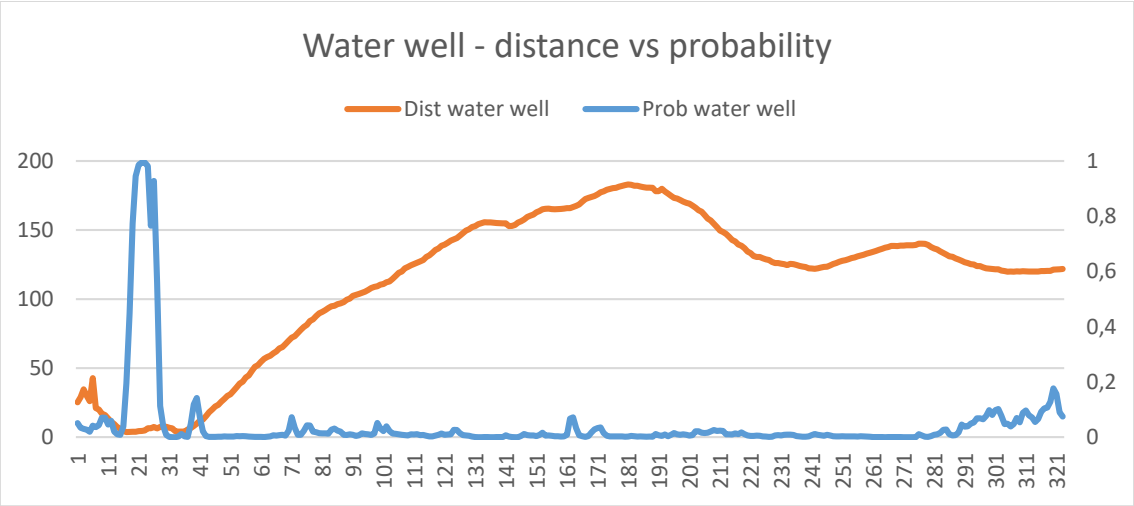


Figure 50: Water well distance vs probability graphic (final evaluation)

After being near water well and Quinta Sant Rafael, we moved through the park to go to see Glorieta. Once we were approximating to it, probability starts increasing fast and telling us that we are in the other part of the park (figure 51), very close to the object. When we leave this location, we can see some noise in classification process that makes it to increase to 0.4-0.5 but at this moment we were far from it and we are going to see Skate Park.

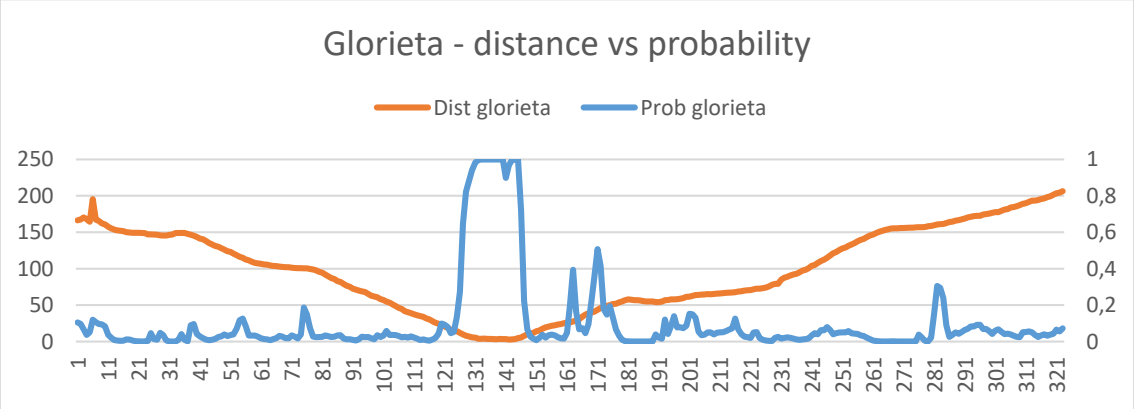


Figure 51: Glorieta distance vs probability graphic (final evaluation)

After that, we walk to skate park. This landmark object is made of cement and has sand ground around it. These two characteristics makes the system to give a high probability for the skate park in other locations, not only in the skate park itself.

When we are seeing a probability bigger than 0.8 (figure 52) we could say that we are near skate park. Other values need to be discarded due to the noise that we record during the path.

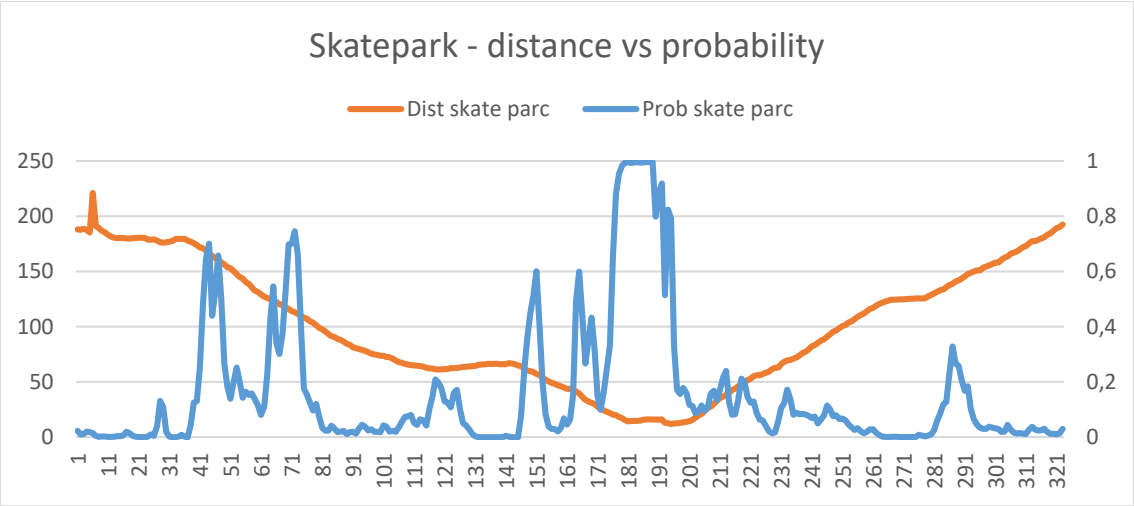


Figure 52: Skate park distance vs probability graphic (final evaluation)

Arriving at the park entrance, we could see that the probability rises to 1.0 for this object. But figure 53 shows that this object presents poor classification results due to its composition and size.

During the walkthrough, the system classifies and says us that it is present in most of locations in the park. We only take this as true value to geolocate user when it's 0.98 or higher because of all the noise probabilities that goes from 0.3 to 0.98.

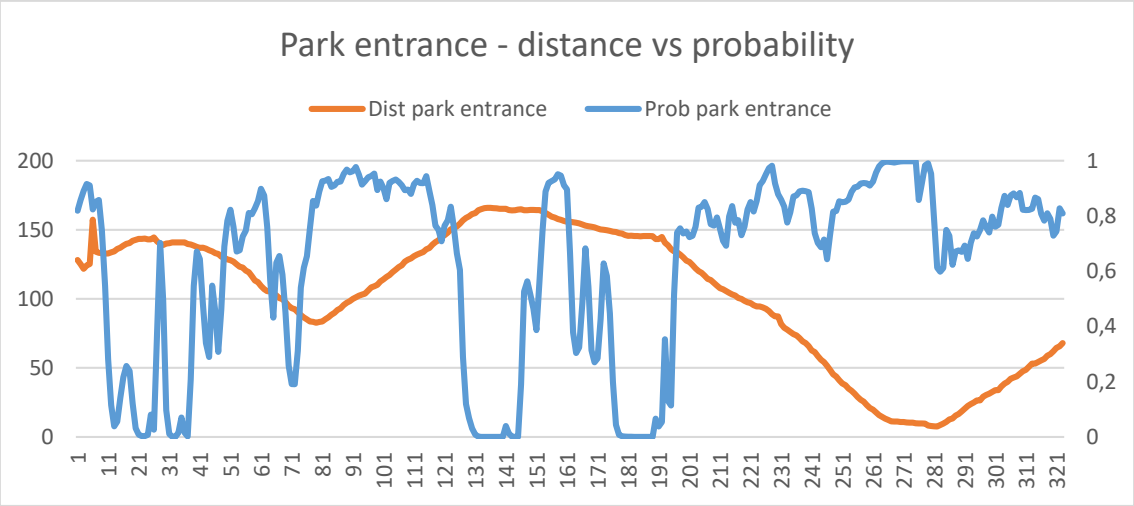


Figure 53: Park entrance distance vs probability graphic (final evaluation)

Figure 54 shows the entire path followed to extract this evaluation.

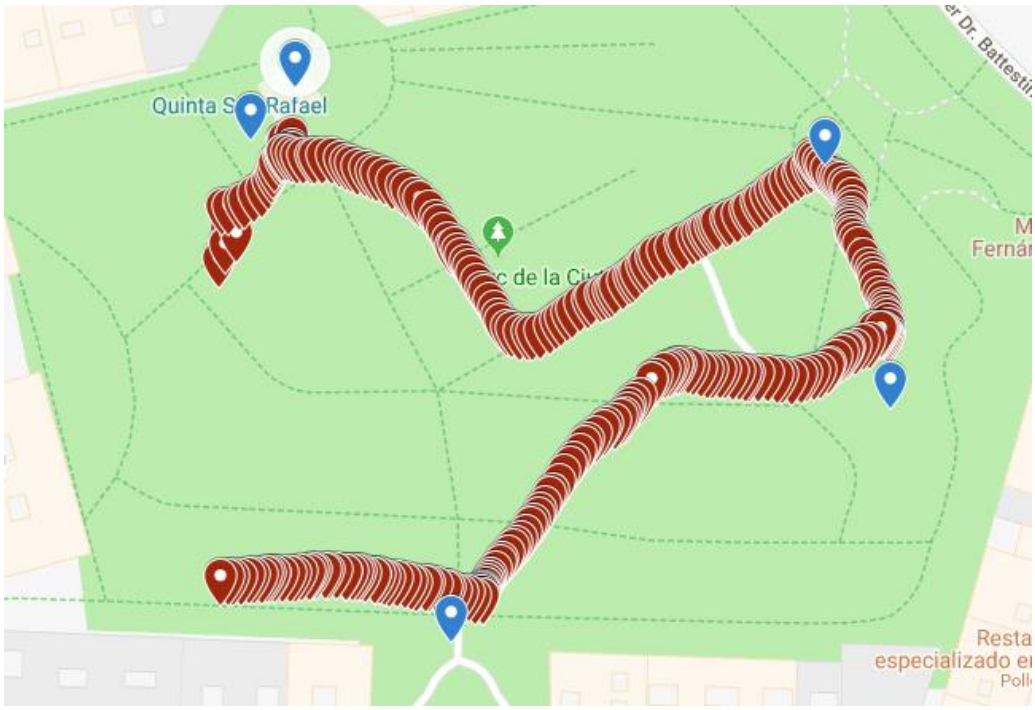


Figure 54: Evaluation path of final walkthrough

6. Conclusions

Deep learning in object classification systems is evolving exponentially on one hand because of the development of new neural networks and on the other hand because of the existing ones are outperformed with new advances. We have seen that there are lots of convolutional networks that are developed and further improved to make this classification processes faster and more accurate.

However, this networks have a high computational cost that make them difficult to deploy and integrate in mobile environments and embedded systems. As we have already seen, TensorFlow is one of the few frameworks that allow us to create a lightweight version (TensorFlow Lite) of these networks. Thus, enabling their application in our smartphones.

In section 4, we have shown our system composition consisting in a re-trained network called MobileNet version 1, and an Android application. Both elements allow us to classify custom objects using transfer learning.

In section 5 we have seen the ability of the prototype system to detect and classify this 5 objects in the evaluated space giving us some geolocation information about where the user locates.

Although the success of the project we believe that the network and the system will need to be future reviewed in order to apply changes to improve the whole system.

In this regard, and going deeper with the selected network used in this prototype, we propose some features that could be reviewed and/or changed to improve the final result:

- We could see that the network is a bit overfitted mainly due to the number of images and the type of images that we take to train the network. Ideally, we would have taken more pictures for every object, in more angles and with more types of backgrounds. That was not possible in our scenario since our training objects were buildings so we couldn't move the set objects to another location.

- MobileNet type could be modified. Referring to the parameter that sets the relative size of the model as a fraction of the entire MobileNet, it could be set as lower than 1.0. The size of the network is related to the number of inputs that it has; if we have less inputs, the network becomes more robust to prevent overfitting.

In the testing and evaluating part of this project, the selected "Parc de la ciutat" space has a very homogeneous views and textures in all its extension. There are many types of trees, vegetation and sand ground all over the park that easily appears in the images as background. This fact added non-expected difficulties to our classification system.

The problem exposed above is independent to the type of network selected to do the classification process. Therefore, as we have previously seen in section 2, there are many new networks that would perform better than MobileNet. Future work should be address in this regard in order to be able to integrate a faster and more accurate network. This only could be done improving TensorFlow and TOCO with newer versions that integrate

new operations required by this new and better networks. We expect these improvements to result in better outcomes than the ones presented in this project.

Focusing in the user geolocalitzation in the map, and extracting geolocation information of the system, the information obtained and evaluated have good and bad things that need to be commented. System allow us to know if we are near the trained objects, in a distance of 0 to 20 meters depending in the object analysed. When the classify probability increases more than 0.6/0.7 it's a sign that we are focusing an object of our system, which we know the geolocation coordinates. At that point, we can geolocate the user in the space although it is still difficult to exactly determine a trusted value for the distance between the user and the object.

Taking into account all the information exposed above there is evidence of the limitation in geolocate a user during a walkthrough. The system only knows, at a certain fractions of time, where the user locates. In this regard two situation can be limiting for the system to locate the user; if the user does not properly focus a trained object or if the classification algorithm can't distinguish a training object (because of the distance, for example).

The specific point where we are not close enough to the landmark object could be improved with a better training of the network or, as we mention above, with a more efficient type of convolutional network. Thus, the system would classify objects located farther away than in this present prototype and we will be able to do some computation to predict the distance and the direction of the user.

To conclude, further work in the evaluation of the composed approximation (section 5.2), when the systems view two objects in the same frame, and classifies them, the resultant geolocation could be improved calculating a triangulation between the two classified objects geolocations in order to give a better approximation of the user's position.

7. References

[1] A. Krizhevsky, I. Sutskever, and G. E. Hinton. Imagenet classification with deep convolutional neural networks. In Advances in neural information processing systems, pages 1097–1105, 2012

[2] C. Szegedy, W. Liu, Y. Jia, P. Sermanet, S. Reed, D. Anguelov, D. Erhan, V. Vanhoucke, and A. Rabinovich. Going deeper with convolutions. In Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, pages 1–9, 2015

[3] https://www.tensorflow.org/mobile/tflite/

[4] https://vision.cornell.edu/se3/geolocalization/

[5] Altwaijry, Hani; Veit, Andreas; Belongie, Serge. Learning to Detect and Match Keypoints with Deep Architectures. British Machine Vision Conference (BMVC), York, UK, 2016.

[6] Lin, Tsung-Yi; Belongie, Serge; Hays, James. Cross-View Image Geo-localization. Zamir, Amir; Hakeem, Asaad; Gool, Luc Van; Shah, Mubarak; Szeliski, Richard (Ed.): Large-Scale Visual Geo-Localization, pp. 59-76, Springer, 2016.

[7] Altwaijry, Hani; Trulls, Eduard; Hays, James; Fua, Pascal; Belongie, Serge. Learning to Match Aerial Images with Deep Attentive Architectures. Computer Vision and Pattern Recognition (CVPR), Las Vegas, NV, 2016.

[8] Rizvi, S. T. H., Cabodi, G., & Francini, G. (2017). Optimized Deep Neural Networks for Real-Time Object Classification on Embedded GPUs. Applied Sciences, 7(8), 826.

[9] Frajberg, D., Fraternali, P., & Torres, R. N. (2017, September). Convolutional neural network for pixel-wise skyline detection. In International Conference on Artificial Neural Networks (pp. 12-20). Springer, Cham.

[10] Tobías, L., Ducournau, A., Rousseau, F., Mercier, G., & Fablet, R. (2016, December). Convolutional Neural Networks for object recognition on mobile devices: A case study. In Pattern Recognition (ICPR), 2016 23rd International Conference on (pp. 3530-3535). IEEE.

[12] Torrey, L., & Shavlik, J. (2010). Transfer learning. In Handbook of Research on Machine Learning Applications and Trends: Algorithms, Methods, and Techniques (pp. 242-264). IGI Global.

[13] Srivastava, N., Hinton, G., Krizhevsky, A., Sutskever, I., & Salakhutdinov, R. (2014). Dropout: a simple way to prevent neural networks from overfitting. The Journal of Machine Learning Research, 15(1), 1929-1958.

[14] Howard, A. G., Zhu, M., Chen, B., Kalenichenko, D., Wang, W., Weyand, T., ... & Adam, H. (2017). Mobilenets: Efficient convolutional neural networks for mobile vision applications. arXiv preprint arXiv:1704.04861.

[15] Sandler, M., Howard, A., Zhu, M., Zhmoginov, A., & Chen, L. C. (2018). MobileNetV2: Inverted Residuals and Linear Bottlenecks. In Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (pp. 4510-4520).

[16] https://www.tensorflow.org/

[17] https://developer.android.com/ndk/guides/neuralnetworks/

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